МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРНІГІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

ТЕЛЕКОМУНІКАЦІЙНІ СИСТЕМИ І ТЕХНОЛОГІЇ

Методичні вказівки

до виконання самостійних робіт
з дисципліни «Телекомунікаційні системи і технології»
для студентів напряму підготовки
6.050102 «Комп'ютерна інженерія»

ЗАТВЕРДЖЕНО

на засіданні кафедри інформаційних
та комп'ютерних систем
протокол № 6 від 29.01.2014

Чернігів ЧНТУ 2014

Телекомунікаційні системи і технології. Методичні вказівки до виконання самостійних робіт з дисципліни "Телекомунікаційні системи і технології" для студентів напряму підготовки 6.050102 "Комп'ютерна інженерія" / Укл.: Зайцев С.В., Риндич Є.В., Нікітенко Є.В. – Чернігів: ЧНТУ, 2014. – 51 с.

Укладачі: ЗАЙЦЕВ СЕРГІЙ ВАСИЛЬОВИЧ, кандидат технічних наук, доцент кафедри інформаційних та комп'ютерних систем

РИНДИЧ ЄВГЕН ВОЛОДИМИРОВИЧ, кандидат технічних наук, доцент кафедри інформаційних та комп'ютерних систем

НІКІТЕНКО ЄВГЕНІЙ ВАСИЛЬОВИЧ, кандидат фізико-математичних наук, доцент кафедри інформаційних та комп'ютерних систем

Відповідальний за випуск: КАЗИМИР ВОЛОДИМИР ВІКТОРОВИЧ, завідувач кафедри інформаційних та комп'ютерних систем, доктор технічних наук, професор

Рецензент: НЕСТЕРЕНКО СЕРГІЙ ОЛЕКСАНДРОВИЧ, кандидат технічних наук, доцент кафедри інформаційних та комп'ютерних систем Чернігівського державного технологічного університету

ЗМІСТ

1 САМОСТІЙНА РОБОТА № 1. МОДЕЛЮВАННЯ ЛОКАЛЬНОЇ МЕРЕЖІ 5
1.1 Теоретичні відомості 5
1.1.1 Packet Tracer 5
1.1.2 Основні сервери Packet Tracer 5
1.2 Матеріали для виконання роботи 6
1.2.1 Робота з Packet Tracer 6
1.2.2 Створення моделі мережі 7
1.2.3 Перевірка працездатності мережі 9
1.3 Завдання 10
1.4 Контрольні питання 10
2 САМОСТІЙНА РОБОТА № 2. ЛОКАЛЬНІ МЕРЕЖІ. РОБОТА З VLAN 11
2.1 Теоретичні відомості 11
2.1.1 VLAN 11
2.1.2 Структура IP-адреси 11
2.1.3 Десятковий запис IP-адреси 12
2.1.4 Мережі та підмережі. Маски 13
2.1.5 Організація підмереж 14
2.2 Матеріали для виконанння роботи 16
2.2.1 Проведення розрахунків для визначення адрес та маски мережі 16
2.2.2 Модель мережі і налагодження свіча 17
2.2.3 Аналіз і тестування мережі 18
2.3 Завдання 19
2.4 Контрольні питання 20
3 САМОСТІЙНА РОБОТА № 3. ЛОКАЛЬНІ МЕРЕЖІ. РОБОТА З VLAN І МАРШРУТИЗАТОРОМ 21
3.1 Теоретичні відомості 21
3.2 Матеріали для виконання роботи 22
3.2.1 Розділення мережі на підмережі 22
3.2.2 Модель мережі і налагодження свіча та маршрутизатора 22
3.2.3 Аналіз і тестування мережі 26
3.3 Завдання 27
3.3.1 Хід роботи 27
3.4 Контрольні питання 29
4 САМОСТІЙНА РОБОТА № 4. ДОСЛІДЖЕННЯ РОБОТИ З ПОТОКОВИМИ СОКЕТАМИ В РЕЖИМІ ОПИТУ 30
4.1 Теоретичні відомості 30
4.1.1 Робота з сокетами 30
4.1.2 Реалізація сервера 31
4.1.3 Реалізація клієнта 34
4.1.4 Приклад роботи з WinSock 36
4.1.5 Приклад роботи для Unix подібних систем 39
4.2 Завдання 40
4.3 Контрольні питання 40
5 САМОСТІЙНА РОБОТА № 5. ДОСЛІДЖЕННЯ РОБОТИ З ДАТАГРАМНИМИ СОКЕТАМИ В РЕЖИМІ ОПИТУ 42
5.1 Теоретичні відомості 42
5.1.1 Приклад роботи WinSock 43
5.1.2 Приклад роботи (для UNIX подібних систем) 45
5.2 Завдання 46
5.3 Контрольні питання 46

6 САМОСТІЙНА РОБОТА № 6. ДОСЛІДЖЕННЯ РОБОТИ АСИНХРОННИХ СОКЕТІВ .. 47
6.1 Теоретичні відомості 47
6.1.1 Текст програми 47
6.2 Завдання 50
6.3 Контрольні питання 50
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 51

1 САМОСТІЙНА РОБОТА № 1
МОДЕЛЮВАННЯ ЛОКАЛЬНОЇ МЕРЕЖІ

Мета роботи: вивчити можливості системи моделювання локальної мережі Packet Tracer, створити модель локальної мережі та перевірити її працездатність.

1.1 Теоретичні відомості

1.1.1 Packet Tracer

Packet Tracer – безкоштовний емулятор мережевого середовища, що випускається фірмою Cisco. Дозволяє робити працездатні моделі мережі, налагоджувати (командами CISCO IOS) маршрутизатори і комутатори, взаємодіяти між декількома користувачами.

Включає серії маршрутизаторів Cisco 1800, 2600, 2800 і комутаторів 2950, 2960, 3650. Крім того є сервери DHCP, HTTP, TFTP, FTP, TIME, робочі станції, різні модулі до комп'ютерів і маршрутизаторів, пристрої Wifi, різні кабелі. Успішно дозволяє створювати навіть складні макети мереж, перевіряти на працездатність топології.

1.1.2 Основні сервери Packet Tracer

DHCP (англ. Dynamic Host Configuration Protocol — протокол динамічної конфігурації вузла) — це мережевий протокол, що дозволяє комп'ютерам автоматично отримувати ip-адресу і інші параметри, необхідні для роботи в мережі Tcp\ip. Даний протокол працює по моделі «клієнт-сервер». Для автоматичної конфігурації комп'ютер-клієнт на етапі конфігурації мережевого пристрою звертається до так званого сервера DHCP, і отримує від нього потрібні параметри. Мережевий адміністратор може задати діапазон адрес, що розподіляються сервером серед комп'ютерів. Це дозволяє уникнути ручного налаштування комп'ютерів мережі і зменшує кількість помилок. Протокол DHCP використовується в більшості великих (і не дуже) мереж Tcp/ip.

HTTP (від англ. Hypertext Transfer Protocol — «протокол передачі гіпертексту») — протокол прикладного рівня передачі даних (спочатку — у вигляді гіпертекстових документів). Основою HTTP є технологія «клієнт-сервер», тобто передбачається існування споживачів (клієнтів), які ініціюють з'єднання і посилають запит, і постачальників (серверів), які чекають з'єднання для здобуття запиту, виробляють необхідні дії і повертають назад повідомлення з результатом. HTTP на даний час скрізь використовується в Всесвітній павутині для здобуття інформації з веб-сайтів. У 2006 році в Північній Америці доля http-трафіку перевищила долю P2p-мереж і склала 46%, з яких майже половина — це передача потокового відео і звуку.

Основним об'єктом маніпуляції в HTTP є ресурс, на який вказує URI (англ. Uniform Resource Identifier) в запиті клієнта. Зазвичай такими ресурсами є

файли, що зберігаються на сервері, але ними можуть бути логічні об'єкти або щось абстрактне. Особливістю протоколу HTTP є можливість вказати в запиті і відповіді спосіб представлення одного і того ж ресурсу по різних параметрах: формату, кодуванню, мові. Саме завдяки можливості задання способу кодування повідомлення клієнт і сервер можуть обмінюватися двійковими даними, хоча даний протокол є текстовим.

DNS (англ. Domain Name System — система доменних імен) — комп'ютерна розподілена система для здобуття інформації про домени. Найчастіше використовується для здобуття ip-адреси по імені хоста (комп'ютера або пристрою), здобуття інформації про маршрутизацію пошти, обслуговуючих вузлах для протоколів в домені (Srv-запис). Розподілена база даних DNS підтримується за допомогою ієрархії dns-серверів, що взаємодіють по певному протоколу. Основою DNS є уявлення про ієрархічну структуру доменного ім’я і зонах. Кожен сервер, що відповідає за ім'я, може делегувати відповідальність за подальшу частину домена іншому серверу (з адміністративної точки зору — іншій організації або людині), що дозволяє покласти відповідальність за актуальність інформації на сервери різних організацій (людей), що відповідають лише за «свою» частину доменного імені.

1.2 Матеріали для виконання роботи

1.2.1 Робота з Packet Tracer

Вибір необхідного елементу мережі необхідно робити в нижній панелі робочого вікна (рисунок 1.1). На панелі є групи приладів і їхні модифікації.

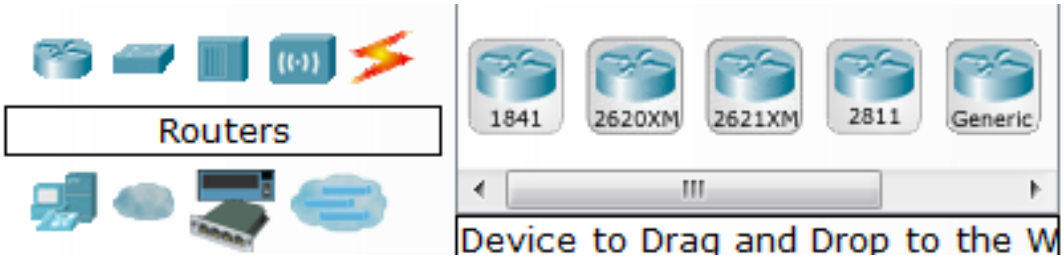


Рисунок 1.1 – Вибір необхідного елементу мережі

Типи кабелів для з’єднання знаходяться на цій же панелі в закладці «Connections». Тут можна вибрати для з’єднання кабель, який необхідний (рисунок 1.2).

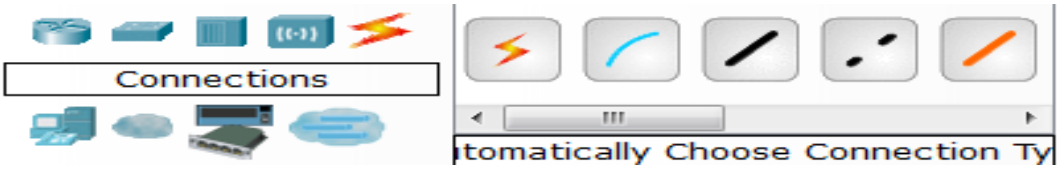


Рисунок 1.2 – Вибір необхідного типу кабеля

1.2.2 Створення моделі мережі

Для створення мережі використано 2 сервери: DNS-DHSP і mail, комутатор Cisco Catalyst 2960, 10 робочих станцій.

Елементи потрібно з’єднати, як показано на рисунку 1.3.

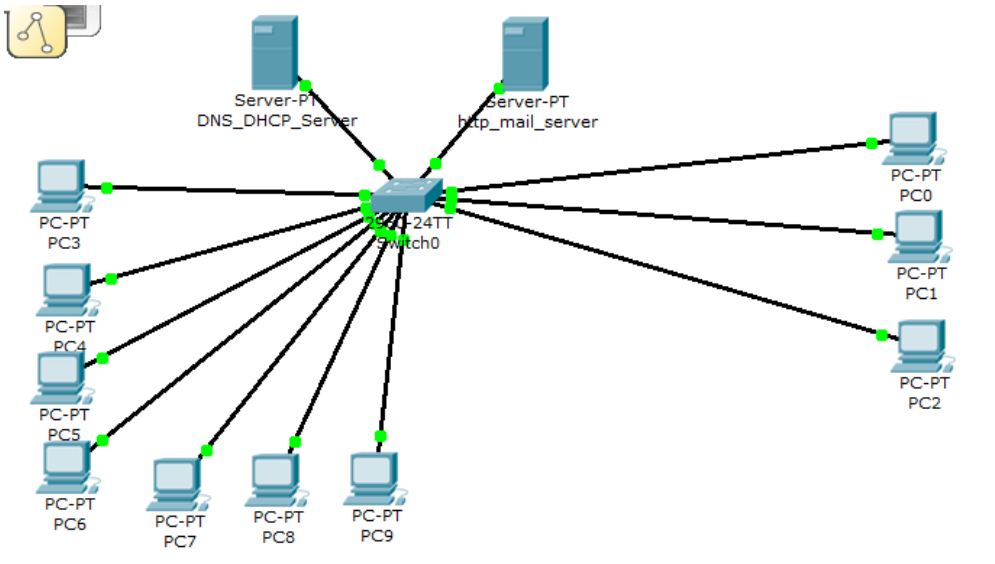


Рисунок 1.3 – Топологія створеної мережі

Під’єднані до мережі хости, сервери або інші елементи потрібно налаштувати. IP-адреси для компонентів можна роздавати автоматично або присвоювати кожному елементу окремо.

Для налагодження хоста необхідно клацнути два рази на ньому лівою клавішею мишки. В відкритому вікні перейти на закладку «Config» і вибрати тип отримання ip-адреси (рисунок 1.4).

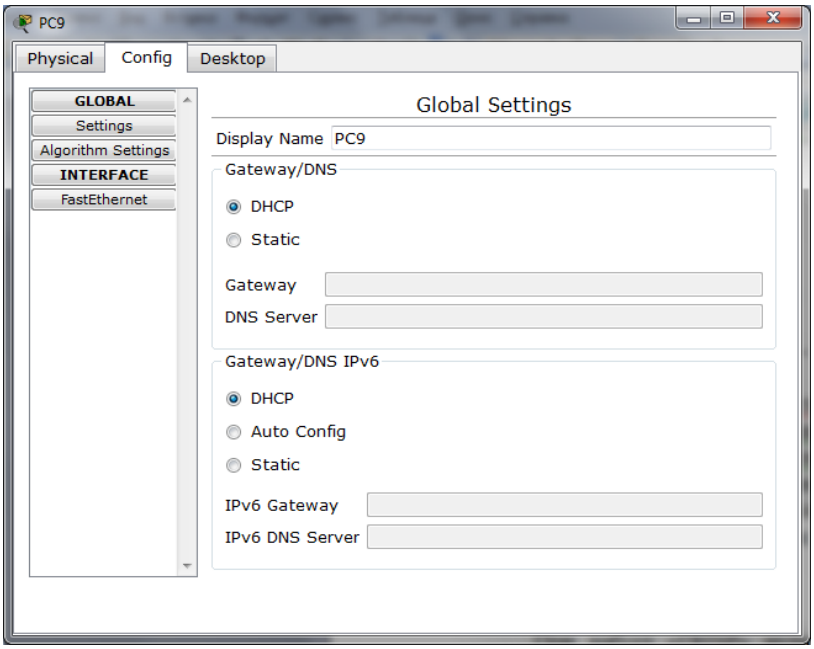


Рисунок 1.4 – Динамічне або статичне отримання адреси для хостів

При виборі «DHSP» ардесу хосту буде присвоєно автоматично в залежності від налаштування DHSP-серверу.

Для статичного отримання адреси необхідно відмітити поле «Static», написати gateway-адресу (необхідна для зв'язування мереж, зазвичай одна з вільних адрес даної підмережі) і DNS-сервер. Після цього перейти на закладку «Desktop», вибрати «IP Cofiguration», заповнити всі поля в відкритому вікні (рисунок 1.5).

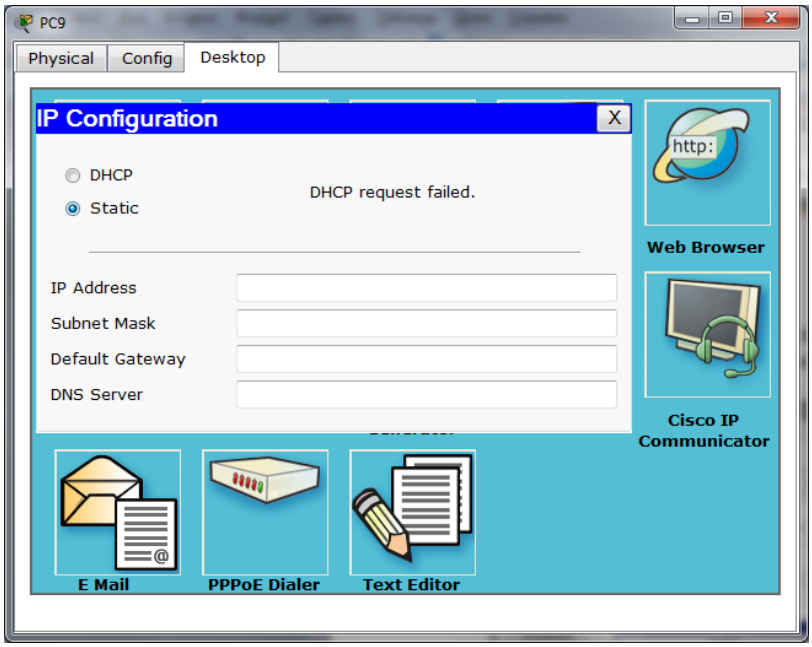


Рисунок 1.5 – Вікно для задання IP-адреси, маски, шлюзу, DNS-серверу для хоста

Також потрібно налаштувати DNS і DHCP сервери, дати їм назви і IP-адреси. Налаштування показане на рисунках 1.6 та 1.7.

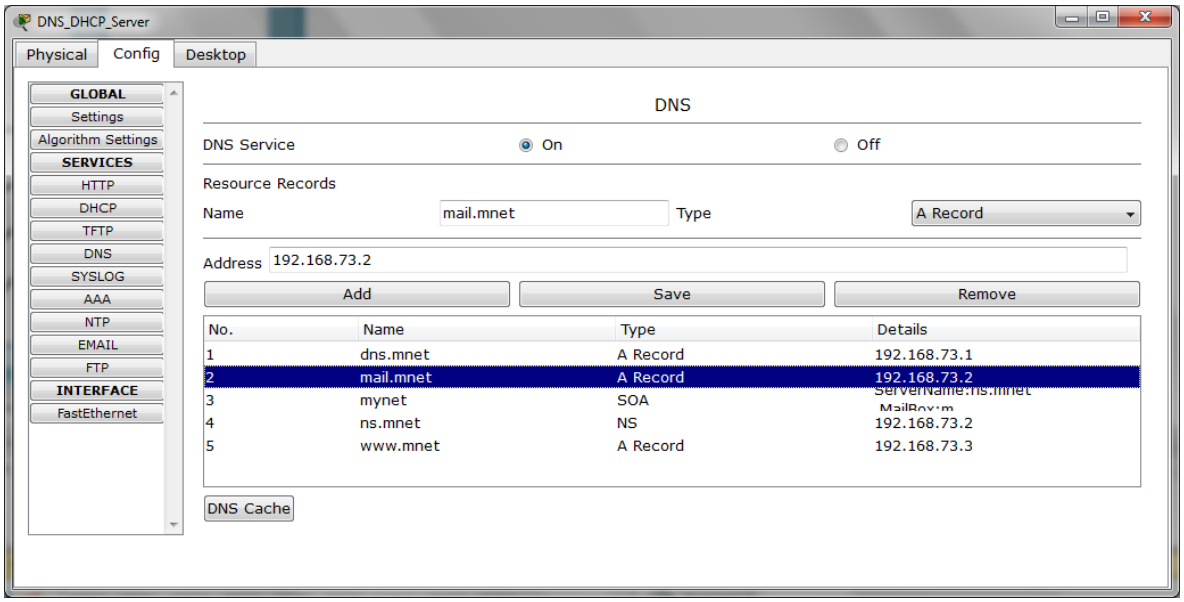


Рисунок 1.6 – Налаштування DNS-сервера мережі

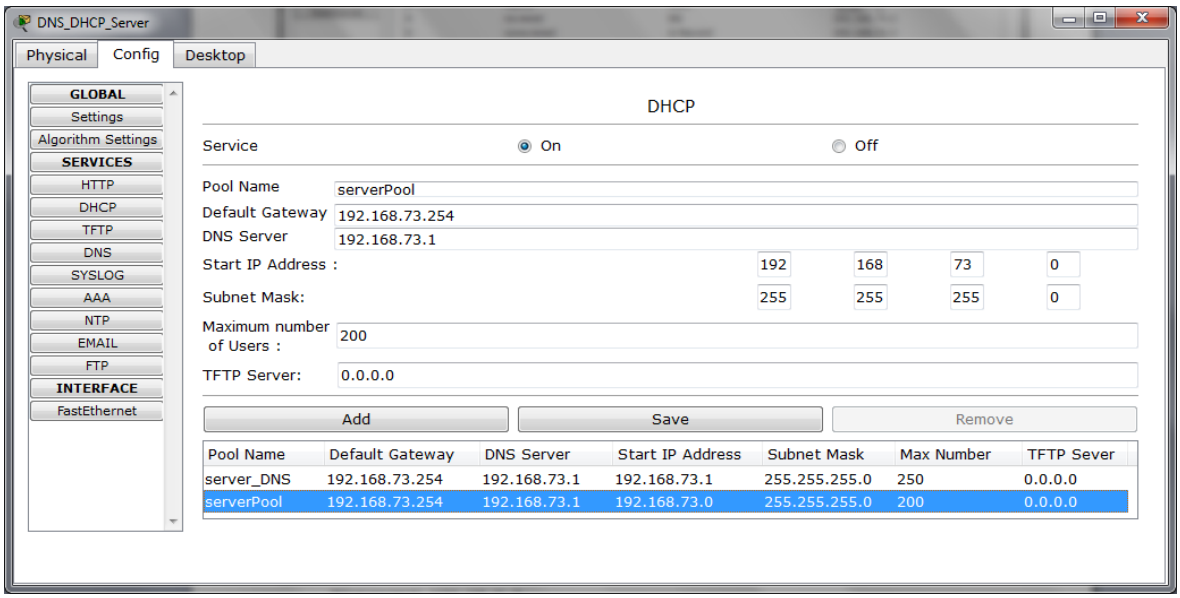


Рисунок 1.7 – Налаштування DHCP-сервера мережі

1.2.3 Перевірка працездатності мережі

В результаті виконання команди ping на один із хостів, отриманий вивід:

```
Packet Tracer SERVER Command Line 1.0
SERVER>ping 192.168.73.7
Pinging 192.168.73.7 with 32 bytes of data:
Reply from 192.168.73.7: bytes=32 time=10ms TTL=128
Reply from 192.168.73.7: bytes=32 time=10ms TTL=128
Reply from 192.168.73.7: bytes=32 time=12ms TTL=128
Reply from 192.168.73.7: bytes=32 time=12ms TTL=128
Ping statistics for 192.168.73.7:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 10ms, Maximum = 12ms, Average = 11ms
```

Результати виконання команди tracert для обох серверів показані нижче:

```
PC>traert 192.168.73.1
Invalid Command.
PC>tracert 192.168.73.1
Tracing route to 192.168.73.1 over a maximum of 30 hops:
  1   8 ms      8 ms      6 ms      192.168.73.1
Trace complete.
PC>tracert 192.168.73.2
Tracing route to 192.168.73.2 over a maximum of 30 hops:

  1   7 ms      7 ms      6 ms      192.168.73.2
Trace complete.
```

Превірку роботи мережі можна також за допомогою елементів (рисунок 1.8) на боковій панелі.

Рисунок 1.8 – Елементи для перевірки передачі пакетів

У панелі на рисунку 1.9 буде зображено виконання або невиконання передачі пакетів.

Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit	Delete
	Successful	PC1	192.168.73.3	ICMP		100.000	N	0	(edit)	(delete)
	Successful	PC0	DNS_DHCP_Server	ICMP		0.000	N	1	(edit)	(delete)
	Successful	DNS_DHCP_Server	http_mail_server	ICMP		0.000	N	2	(edit)	(delete)
	Successful	PC5	DNS_DHCP_Server	ICMP		0.000	N	3	(edit)	(delete)

Рисунок 1.9– Моделювання передачі пакетів серверів та хостів

1.3 Завдання

1. Скласти модель локальної мережі, яка складається з 10 робочих станцій, 2 серверів, 1 коммутатора (Cisco Catalyst 2960).
2. З'єднати між собою вузли мережі кабелями витої пари.
3. Налагодити мережеві інтерфейси робочих станцій і серверів. Для налаштування використовувати мережу, яка має ip-адресу 192.168.73.0 і маску підмережі 255.255.255.0.
4. Перевірити працездатність локальної мережі:
5. В командному рядку робочих станцій виконати команду ping для всіх робочих станцій і серверів;
6. В командному рядку робочих станцій виконати команду tracert до всіх серверів;
7. Виконати команду ping між двома хостами.

1.4 Контрольні питання

1. Як додати елемент в модель мережі?
2. Які групи мережевих елементів існують в Ptracer?
3. Які типи сполучних кабелів існують в Ptracer?
4. Як з'єднати елементи в моделі мережі?
5. Як виконати команду на будь-якому хості?
6. Як створити сценарій?
7. Для чого служить команда ping?
8. Для чого служить команда tracert?

2 САМОСТІЙНА РОБОТА № 2
ЛОКАЛЬНІ МЕРЕЖІ. РОБОТА З VLAN

Мета роботи: вивчити на практиці основи побудови віртуальних мереж (VLAN), використовуючи навики мережевої арифметики, навчитися розбивати мережу на підмережі довільних масок.

2.1 Теоретичні відомості

2.1.1 VLAN

VLAN (аббр. від англ. Virtual Local Area Network) — віртуальна локальна комп'ютерна мережа, є групою хостів із загальним набором вимог, які взаємодіють так, як якби вони були підключені до широкомовного домена, незалежно від їх фізичного місцезнаходження. VLAN має ті ж властивості, що і фізична локальна мережа, але дозволяє кінцевим станціям групуватися разом, навіть якщо вони не знаходяться в одній фізичній мережі. Така реорганізація може бути зроблена на основі програмного забезпечення замість фізичного переміщення пристроїв.

Переваги:

1. Полегшується переміщення, додавання пристроїв і зміна їх з'єднань один з одним.
2. Досягається велика міра адміністративного контролю унаслідок наявності пристрою, що здійснює між мережами VLAN маршрутизацію на 3-м-коді рівні.
3. Зменшується вжиток смуги пропускання в порівнянні з ситуацією одного широкомовного домена.
4. Скорочується невиробниче використання CPU за рахунок скорочення пересилки широкомовних повідомлень.
5. Запобігання широкомовним штормам і запобігання петлям.

2.1.2 Структура IP-адреси

Схема маршрутизації повідомлень в TCP/IP базується на унікальних адресах, названих *адресами Internet* або *IP-адресами*, які утворюють пару:

<адреса локальної мережі, адреса вузла в локальній мережі> або *(<NetID, HostID>)*

IP-адреси представлені 32-бітовим кодом і діляться на класи: A, B, C, D, E (табл. 2.1). Найбільше використання на даний час мають перші 3 класи.

Таблиця 2.1 – IP-адреси

<table>
<tr><td></td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td></td><td></td><td></td><td>…</td><td>16</td><td></td><td></td><td></td><td></td><td></td><td>…
.</td><td>24</td><td></td><td></td><td></td><td></td><td></td><td>…</td><td>32</td></tr>
<tr><td>Клас A</td><td>0
0</td><td colspan="8">Адреса мережі
NetID (7 біт)</td><td colspan="23">Адреса вузла
HostID (24 біти)</td></tr>
<tr><td>Клас B</td><td>0
1</td><td>0
0</td><td colspan="14">Адреса мережі
NetID (14 біт)</td><td colspan="16">Адреса вузла
HostID (16 біт)</td></tr>
<tr><td>Клас C</td><td>1
1</td><td>1
1</td><td>0
0</td><td colspan="21">Адреса мережі
NetID (21 біт)</td><td colspan="8">Адреса вузла
HostID(8 біт)</td></tr>
<tr><td>Клас D</td><td>1
1</td><td>1
1</td><td>1
1</td><td>0
0</td><td colspan="28">Багатоадресна MulticastGroupID (28 біт)</td></tr>
<tr><td>Клас E</td><td>1
1</td><td>1
1</td><td>1
1</td><td>1
1</td><td>1
0</td><td colspan="27">Зарезервовано для майбутніх застосувань (27 біт)</td></tr>
</table>

- **Клас A** - адреса починається з **0**
- **Клас B** - адреса починається з **10**
- **Клас C** - адреса починається з **110**
- **Клас D** - адреса починається з **1110**
- **Клас E** - адреса починається з **1111**

Розподіл кількості адрес мереж та вузлів у різних класах показано в таблиці 2.2.

Таблиця 2.2 – Розподіл кількості адрес у класах

	Число мереж N_d (domains)	Число вузлів N_n (hosts)
Клас **A**	2^8 -2	2^{24}-2
Клас **B**	2^{14} -2	2^{16}-2
Клас **C**	2^{21}-2	2^8 -2

Коли комп’ютер має два або більше фізичних під’єднань він називається multi-home host. Такий комп’ютер або маршрутизатор потребує множину IP-адрес, при цьому кожна адреса відповідає одному з під’єднань машини до мережі.

Оскільки IP-адреси кодують як мережу, так і вузол в мережі, то ці адреси не визначають конкретний комп’ютер, а визначають під’єднання до мережі.

Тому роутер, який з’єднує n мереж, має n різних IP-адрес, по одній для кожного мережевого під’єднання.

2.1.3 Десятковий запис IP-адреси

IP-адреса має чотири поля (байти) у формі aaa.bbb.ccc.ddd, розділених крапками (таблиця 2.3). Кожне поле звичайно подається у формі десяткового

числа. IP-адреси можна розрізняти за класами, використовуючи десяткове значення ааа першого байта:

Таблиця 2.3. – Розподіл IP адрес за класами

Клас	Найменша адреса	Найбільша адреса
A	0.1.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.1.0	223.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

2.1.4 Мережі та підмережі. Маски

В оригінальній схемі IP-адресації будь-якій фізичній мережі призначена унікальна мережева адреса; будь-який вузол в мережі використовує мережеву адресу як префікс до індивідуальної адреси вузла. Такий поділ обумовлений потребами процесу маршрутизації пакетів. Окремі територіальні мережі мають певну свободу в модифікації адрес і маршрутів, що дозволяє розширити кількість адрес.

В багатьох випадках, наприклад з метою зниження трафіка, чи для організації робочих груп, проявляється необхідність розбиття на підмережі або сегменти. Здійснюється таке розбиття за допомогою масок підмереж. Це призводить до зниження кількості вузлів в мережі, а також спрощує адресацію між ними за рахунок скорочення кількості біт, що залишаються для визначення адреси хоста.

Додатково комп’ютер має знати, скільки біт відведено для SubNetID та HostID. Саме за допомогою маски є можливість вказати розмір цих полів. Маска - 32-розрядне число, що має біти, які відповідають полям NetID та SubNetID, рівні 1, а біти для HostID рівні 0. Адреса підмережі визначається шляхом логічного множення:

<Адреса підмережі> = <IP-адреса & маска>

Розглянемо сегментацію мереж IP на прикладі мережі класу C. Організація пімереж в цьому випадку виконується за допомогою “позичення” для адресації мережі декількох біт з останнього октета. Кількість позичених біт залежить від потрібної кількості підмереж або від обмеження щодо кількості вузлів в підмережі.

У випадку розбиття на дві підмережі відповідно відповідно позичаються 2 біти, залишаючи 6 біт для адресації хостів. Ці два біти з останнього октета будуть додані до бітів, що використовуються для адресації мережі. Шість біт, що залишились, дозволяють кожній з двох підмереж підтримувати 62 унікальних адреси (64 адреси в піжмережі, але перша і остання виділені на адресу мережі і адресу бродкасту відповідно), а маска підмережі, що

використовується, буде виглядати як 255. 255. 255.192(11111111.11111111.11111111.11000000).

Для організації шести підмереж потрібно використовувати три біти, що обмежує кількість вузлів в кожній мережі до тридцяти і з маскою 255.255.255.224 (11111111.11111111.11111111.11100000*)*

Легко зауважити, що розбиття на більшу кількість підмереж призведе до різкого зниження кількості доступних адрес в підмережі.

Досить часто використовують запис адресації виду XXXX.XXXX.XXXX.0/**Y**

Число, що стоїть за дробом, визначає кількість біт, позичених для визначення адреси мережі, що дозволяє відмовитись від стандартної форми запису. Таким чином, адреса класу С 204. 251. 122.0 з маскою 255.255.255.224 може бути записана як 204.251.122.0/27, що означає використання 27 з 32 адресних біт для визначення адреси мережі, залишаючи решту адресного простору для призначення адрес хостам.

Це число називають номером CIDR (Classless InterDomain Routing)

Маска також часто записується у шістнадцятковій формі, особливо тоді, коли приходиться маніпулювати SubNetID з розміром, не кратним 8 бітам. Вищеприведена маска у шістнадцятковій формі записується так: 0xFFFFFF00.

Маючи IP адресу і маску, комп'ютер може визначити чи IP адреса вказує:

- на комп'ютер який знаходиться на його ж підмережі;
- на комп'ютер який знаходиться на іншій підмережі;
- на комп'ютер який знаходиться на іншій мережі.

Наприклад адреса комп'ютера 184.12.44.45 (клас В), а його маска рівна 255.255.255.0 (тобто для SubNetID виділено 8 біт). Якщо комп'ютеру необхідно передати інформацію, призначену для іншого комп'ютера, IP-адреса якого рівна: 184.12.80.2, то комп'ютер може визначити що їх NetID однакові, а от SubNetID різні (44 ≠ 80).

- адреса 184.12.44.50, то комп'ютер може визачити, що вони належать як до одної мережі, так і до одної підмережі, оскільки їх NetID та SubNetID однакові;

- адреса 192.168.0.3 (адреса класу С) - оскільки NetID різні, то подальші уточнення не проводяться.

Ці порівняння необхідні наприклад для того щоб можна було визначитися, чи комп'ютер повинен посилати пакети до призначення прямо на мережу (підмережу), до якої він безпосередної під'єднаний, чи до маршрутизатора, який асоціюється з необхідним напрямком.

2.1.5 Організація підмереж

Сам Internet не бачить організації підмереж, так що організація підмереж відома і розпізнається тільки локально всередині загальної мережі. Однак будучи один раз утворена, кожна підмережа локально діє як окрема мережа, і комунікація між підмережами вимагає того ж, що й комунікація між мережами. Комп'ютери в різних підмережах не можуть бачити один одного, доки не пе-

редбачено спеціального способу для цього. Очевидно, що 16777214 адрес станцій мережі класу А незручні для використання, як і 65534 адрес класу В. Звичайно неможливо використати такий розмір мережі, однак існує простий спосіб організації підмереж в мережах класів А і В: підмереж класу А у вигляді еквівалентних мереж класу В і підмереж класу В у еквівалентні мережі класу С. Зауважимо ще раз, що організація підмереж є тільки внутрішньою, а зовні мережа класу А завжди залишається такою.

Використаємо для прикладу мережу класу С і розглянемо, як можна утворити підмережу. Нехай мережева адреса є 192.168.255.0 і мережева маска для неї є 255.255.255.0. Маємо 8 бітів для адрес станцій, що дає можливих 254 адреси. Нагадаємо, що адреси станцій з усіма двійковими одиницями (тобто 255) або з усімома нулями (тобто 0) не можна застосовувати. Для маски можна призначити будь-які з бітів, зарезервованих в мережевій адресі класу С для станції, тобто від 1 до 6 бітів, однак не можна вживати 7 бітів, бо це означатиме 0 станцій. Зауважимо, що перші три 255.255.255 не змінюються. Доброю практикою при організації підмереж є виділення бітів у мережевій масці неперервно зліва направо. Це не вимога, однак впровадження інших варіантів не дає добрих результатів. Використання 2 та 3 адресних біт для утворення підмереж наведені в таблицях 2.4-2.5.

Таблиця 2.4 –Використання 2 адресних біт для утворення підмереж

2 біти Мережева маска 255.255.255.192	Адреси станцій
підмережа 0: $(00)_2$	від 192.168.255.1 до 192.168.255.62
підмережа 1: $(01)_2$	від 192.168.255.65 до 192.168.255.126
підмережа 2: $(10)_2$	від 192.168.255.129 до 192.168.255.190
підмережа 3: $(11)_2$	від 192.168.255.193 до 192.168.255.254

Оскільки у двійковій формі $192_{10}=(1100\ 0000)_2$, то це забезпечує 4 підмережі, кожна з 62 станціями.

Таблиця 2.5 – Використання 3 адресних біт для утворення підмереж

3 біти Мережева маска 255.255.255.224	Адреси станцій
підмережа 0: $(000)_2$	від 192.168.255.1 до 192.168.255.30
підмережа 1: $(001)_2$	від 192.168.255.33 до 192.168.255.62
підмережа 2: $(010)_2$	від 192.168.255.65 до 192.168.255.94
підмережа 3: $(011)_2$	від 192.168.255.97 до 192.168.255.126
підмережа 4: $(100)_2$	від 192.168.255.129 до 192.168.255.158
підмережа 5: $(101)_2$	від 192.168.255.161 до 192.168.255.190
підмережа 6: $(110)_2$	від 192.168.255.193 до 192.168.255.222
підмережа 7: $(111)_2$	від 192.168.255.225 до 192.168.255.254

Оскільки у двійковій формі $224_{10}=(1110\ 0000)_2$, то це забезпечує 8 підмереж, кожна з 30 станціями.

Подібним чином можна отримати діапазони адрес для мережевих масок:

- 4 біти Мережева маска 255.255.255.240 з 16 підмережами по 14 станцій у кожній;
- 5 бітів Мережева маска 255.255.255.248 з 32 підмережами по 6 станцій у кожній;
- 6 бітів Мережева маска 255.255.255.252 з 64 підмережами по 2 станції у кожній.

Перші 24 біти в IP-адресі можна ігнорувати (у нашому прикладі це десяткове 192.168.255), оскільки вони відносяться до базової мережі, в якій організуються підмережі. При використанні двобітової мережевої маски для підмережі 0 бачимо, що останні 8 бітів завжди починаються від 00_8, що означає їх десятковий еквівалент в інтервалі від 0 до 63. Оскільки всі нулі та всі одиниці в адресі станції не можна використовувати, то це виключає із вжитку 0 і 63, так що залишаються числа від 1 до 62. Для підмережі 2 перші два біти останнього октету адреси для всіх станцій підмережі завжди рівні 10_8, так що наявні комбінації решти шести бітів дають десяткові числа від 128 до 191. Виключаючи вживання всіх нулів та всіх одиниць в адресах станцій, отримуємо прийнятні числа від 129 до 190, що знову забезпечує 62 станції. Аналогічно можна пояснити отримані діапазони адрес для інших мережевих масок.

2.2 Матеріали для виконанння роботи

2.2.1 Проведення розрахунків для визначення адрес та маски мережі

Візьмемо базову адресу мережі 10.1.0.0, кількість під мереж: 7, кількість робочих станцій: 55.

Таблиця 2.6 – Розрахунок ip-адрес

Номер підмережі	Адреса мережі	Широковісна адреса	Початкова адреса хостів	Кінцева адреса хостів
0	10.1.0.0	10.1.0.63	10.1.0.2	10.1.0.57
1	10.1.0.64	10.1.0.127	10.1.0.66	10.1.0.121
2	10.1.0.128	10.1.0.191	10.1.0.130	10.1.0.185
3	10.1.0.192	10.1.0.255	10.1.0.194	10.1.0.249
4	10.1.1.0	10.1.1.63	10.1.1.2	10.1.1.57
5	10.1.1.64	10.1.1.127	10.1.1.66	10.1.1.121
6	10.1.1.128	10.1.1.191	10.1.1.130	10.1.1.185
Маска підмережі				
255.255.255.192				

Адреса 10.1.0.1 – адреса gateway в першій підмережі (10.1.0.0 - адреса підмережі). Аналогічно в кожній підмережі значення адреси gateway буде

наступним після адресу підмережі. Створена модель мережі зображена на рисунку 2.1.

2.2.2 Модель мережі і налагодження свіча

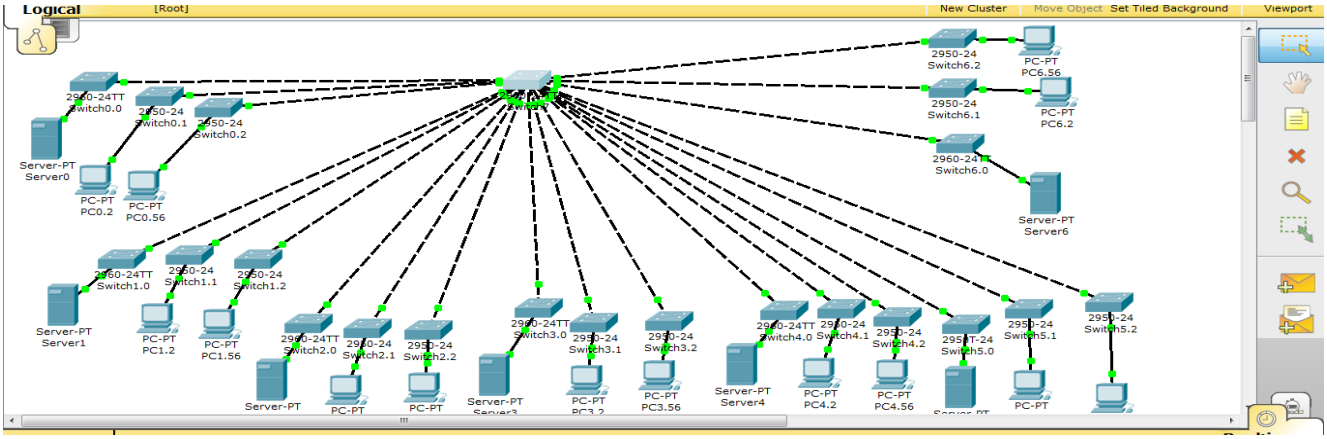


Рисунок 2.1 –Модель мережі

Для створення підмереж треба зайти у закладку «Config» свіча та створити додати необхідну кількість підмереж («VLAN Database»), вказавши в поле VLAN Number номер мережі, починаючи з 2 (1й – зарезервовано), VLAN Name – ім'я мережі (рисунок 2.2).

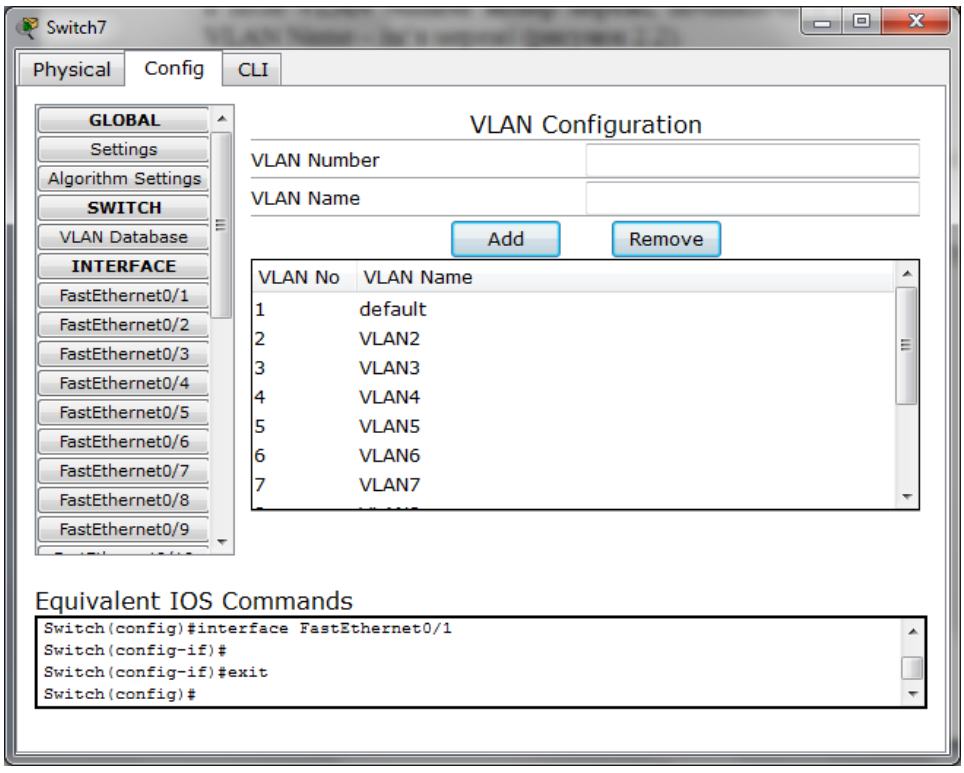


Рисунок 2.2 – Скриншот створення підмереж в базі

На кожний з портів центального свіча встановити необхідну VLAN (рисунок 2.3).

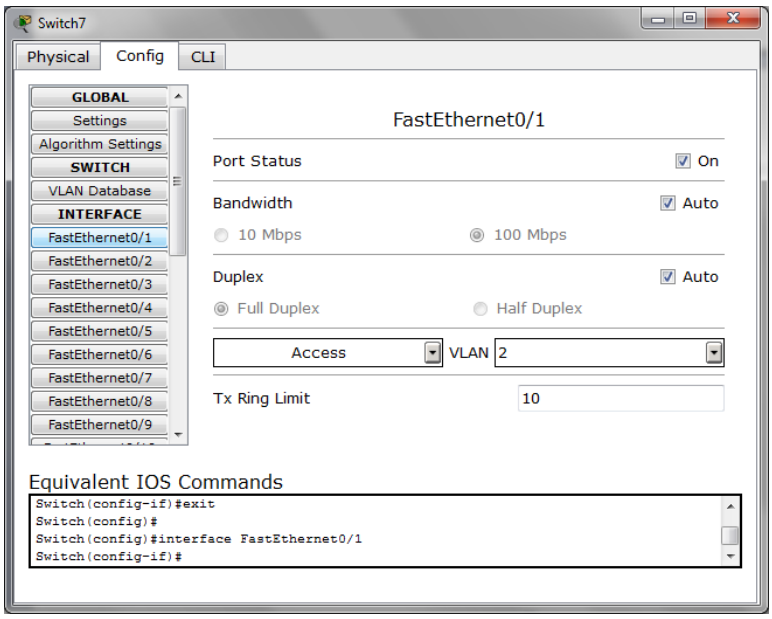


Рисунок 2.3 – Скриншот налаштування портів для підмереж

Свіч складається з 24 портів. Треба під'єднати 55 хостів однієї віртуальної мережі до нього. Отже, використовується 3 додаткових свіча, які під'єднані до центрального свіча, а на портах центрального свіча вказується відповідний однаковий номер підмережі.

Всі підключені порти свіча будуть мати значення «UP» при наведенні на свіч курсором (рисунок 2.4).

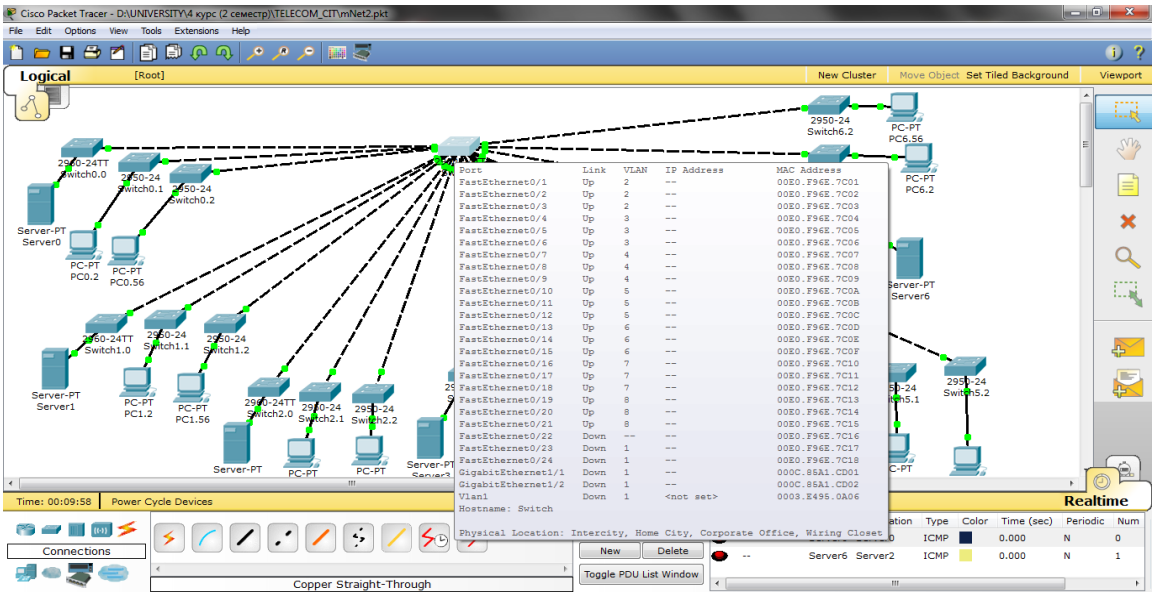
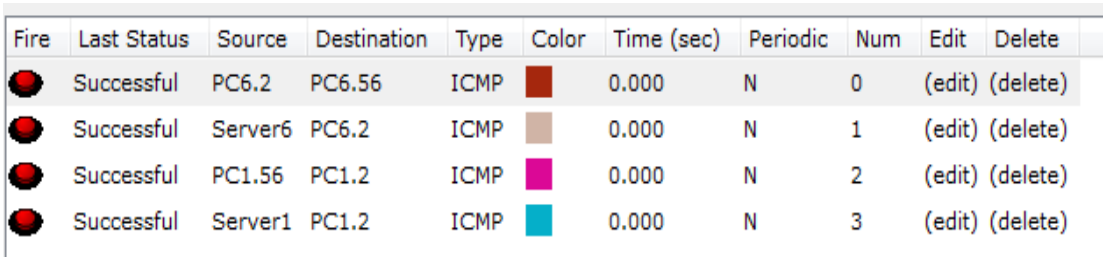


Рисунок 2.4 – Скриншот налагодженого свіча

2.2.3 Аналіз і тестування мережі

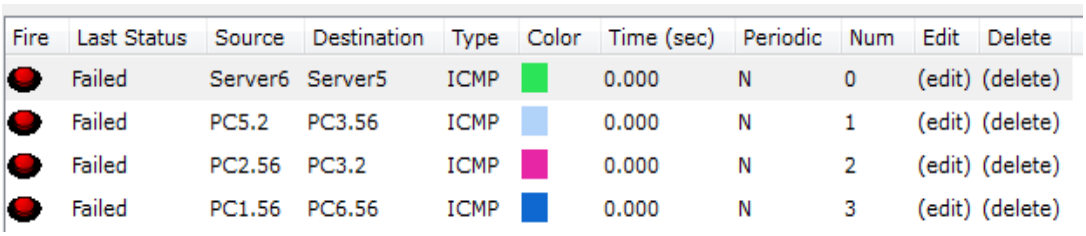
- Відправлення пакетів на робочі станції, які знаходяться в одній віртуальній мережі (рисунок 2.5).



Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit	Delete
	Successful	PC6.2	PC6.56	ICMP		0.000	N	0	(edit)	(delete)
	Successful	Server6	PC6.2	ICMP		0.000	N	1	(edit)	(delete)
	Successful	PC1.56	PC1.2	ICMP		0.000	N	2	(edit)	(delete)
	Successful	Server1	PC1.2	ICMP		0.000	N	3	(edit)	(delete)

Рисунок 2.5 – Скриншот перевірки передачі пакетів в одній віртуальній мережі

- Відправлення пакетів на робочі станції, які знаходяться в різних віртуальних мережах (рисунок 2.6).



Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit	Delete
	Failed	Server6	Server5	ICMP		0.000	N	0	(edit)	(delete)
	Failed	PC5.2	PC3.56	ICMP		0.000	N	1	(edit)	(delete)
	Failed	PC2.56	PC3.2	ICMP		0.000	N	2	(edit)	(delete)
	Failed	PC1.56	PC6.56	ICMP		0.000	N	3	(edit)	(delete)

Рисунок 2.6 – Скриншот перевірки передачі пакетів в різні віртуальні мережі

2.3 Завдання

1. Створити модель мережі що складається з X підмереж і Y робочих станцій. Використовувана адреса мережі, X і Y згідно з варіантом завдання (таблиця 2.7).

Таблиця 2.7 –Варіанти завдань

Номер варіанта	Базова адреса мережі	X кількість підмереж	Y кількість робочих станцій
1	192.168.0.0	7	60
2	172.16.0.0	9	40
3	172.17.0.0	8	35
4	172.18.0.0	11	25
5	172.19.0.0	10	30
6	10.1.0.0	7	55
7	10.2.0.0	8	12
8	10.3.0.0	9	20
9	10.4.0.0	10	15
10	10.5.0.0	11	85
11	10.6.0.0	7	67
12	10.7.0.0	8	16

2. Для кожної підмережі створити свій VLAN.

3. Розрахувати маску підмережі, для реалізації потрібної кількості підмреж.

4. Розрахувати адреси мережі, широкомовну адресу і адреси хостів згідно з варіантом завдання. Надати результати розрахунку у вигляді таблиці 2.8.

Таблиця 2.8 — Результати розрахунку IP-адрес.

Номер підмережі	Адрес мережі	Широковісна адреса	Початкова адреса хостів	Кінцева адреса хостів

5. У Packet Tracer 5.0 створити модель LAN, яка складається з потрібної кількості під мереж. Кількість робочих станцій в кожній підмережі моделі визначається початковою і кінцевою адресою. Таким чином в кожній підмережі буде 2 робочі станції, IP-адреси яких будуть початковий і кінцеві адреси хостів для цієї підмережі.

6. З'єднати необхідне число свічів Cisco Catalyst 2960 між собою і підключити до них робочі станції.

7. Об'єднати робочі станції і сервера в різні віртуальні мережі, номер віртуальної мережі встановити по порядку.

8. Скріншот моделі мережі і налаштувань свіча надати в звіті.

9. Аналіз і тестування мережі :

- відправити пакети на робочі станції, що знаходяться в одній віртуальній мережі. Відобразити результати в звіті;

- відправити пакети на робочі станції, що знаходяться в різних віртуальних мережах. Відобразити результати в звіті.

2.4 Контрольні питання

1. Що таке VLAN? Призначення VLAN?
2. Як реалізований VLAN на рівні протоколу Ethernet?
3. Що таке маска підмережі, на що вона впливає?
4. Що таке адреса мережі? Як виглядає адреса мережі на бітовому рівні?
5. Що таке широкомовна адреса? Як вона виглядає на бітовому рівні? Для чого використовується?
6. Як залежить кількість робочих станцій в мережі від маски?
7. Якою командою призначається VLAN на порт комутатора?
8. Як проглянути список VLAN?
9. Як проглянути налаштування порту?
10. Як поглянути всі налаштування комутатора?

3 САМОСТІЙНА РОБОТА № 3
ЛОКАЛЬНІ МЕРЕЖІ. РОБОТА З VLAN І МАРШРУТИЗАТОРОМ

Мета роботи: вивчити на практиці основи маршрутизації на прикладі маршрутизації віртуальних мереж (VLAN). Використовуючи навички, отримані в попередніх лабораторних роботах, створити мережу, до складу якої входитимуть декілька віртуальних мереж і маршрутизатор, що забезпечує зв'язок між віртуальними мережами і, що надають вихід з віртуальних підмереж в зовнішню мережу.

3.1 Теоретичні відомості

Маршрутизатор – мережевий пристрій, що пересилає пакети даних між різними сегментами мережі і приймає рішення на підставі інформації про топологію мережі і певних правил, заданих адміністратором.

Зазвичай маршрутизатор використовує адресу одержувача, вказану в пакетах даних, і визначає по таблиці маршрутизації шлях, по якому слід передати дані. Якщо в таблиці маршрутизації для адреси немає описаного маршруту, пакет відкидається.

Існують і інші способи визначення маршруту пересилки пакетів, коли, наприклад, використовується адреса відправника, використовувані протоколи верхніх рівнів і інша інформація, що міститься в заголовках пакетів мережевого рівня. Часто маршрутизатори можуть здійснювати трансляцію адрес відправника і одержувача, фільтрацію транзитного потоку даних на основі певних правил з метою обмеження доступу, шифрування/дешифровка даних, що передаються.

Таблиця маршрутизації містить інформацію, на основі якої маршрутизатор приймає вирішення про подальшу пересилку пакетів. Таблиця складається з деякого числа записів — маршрутів, в кожній з яких міститься адреса мережі одержувача, адреса наступного вузла, якому слід передавати пакети і деяку вагу запису, — метрика.

Метрики записів в таблиці відіграють роль в обчисленні найкоротших маршрутів до різних одержувачів. Залежно від моделі маршрутизатора і використовуваних протоколів маршрутизації, в таблиці може міститися деяка додаткова службова інформація.

Таблиця маршрутизації може складатися двома способами:

- статична маршрутизація - коли записи в таблиці вводяться і змінюються вручну. Такий спосіб вимагає втручання адміністратора кожного разу, коли відбуваються зміни в топології мережі. З іншого боку, він є найбільш стабільним і вимагаючим мінімуму апаратних ресурсів маршрутизатора для обслуговування таблиці.
- динамічна маршрутизація - коли записи в таблиці оновлюються автоматично за допомогою одного або декількох протоколів маршрутизації - RIP, OSPF, IGRP, EIGRP, IS-IS, BGP, і ін. Крім того, маршрутизатор будує таблицю оптимальних шляхів до мереж призначення на основі різних критеріїв:

кількості проміжних вузлів, пропускної спроможності каналів, затримки передачі даних.

Критерії обчислення оптимальних маршрутів найчастіше залежать від протоколу маршрутизації, а також задаються конфігурацією маршрутизатора. Такий спосіб побудови таблиці дозволяє автоматично тримати таблицю маршрутизації в актуальному стані і обчислювати оптимальні маршрути на основі поточної топології мережі. Проте динамічна маршрутизація надає додаткове навантаження на пристрої, а висока нестабільність мережі може приводити до ситуацій, коли маршрутизатори не встигають синхронізувати свої таблиці, що приводить до суперечливих відомостей про топологію мережі в різних її частинах і втраті даних, що передаються.

Найчастіше для побудови таблиць маршрутизації використовують теорію графів.

3.2 Матеріали для виконання роботи

3.2.1 Розділення мережі на підмережі

Візьмемо базову адресу першої мережі 10.1.0.0, кількість робочих станцій: 55, базову адресу другої мережі: 10.1.0.64 (таблиця 3.1).

Таблиця 3.1 – Розрахунок ір-адрес для мереж

Номер підмережі	Адреса мережі	Широковісна адреса	Початкова адреса хостів	Кінцева адреса хостів
1	10.1.0.0	10.1.0.63	10.1.0.2	10.1.0.57
2	10.1.0.64	10.1.0.127	10.1.0.66	10.1.0.121
Маска підмережі				
255.255.255.192				

3.2.2 Модель мережі і налагодження свіча та маршрутизатора

Створена модель мережі зображена на рисунку 3.1.

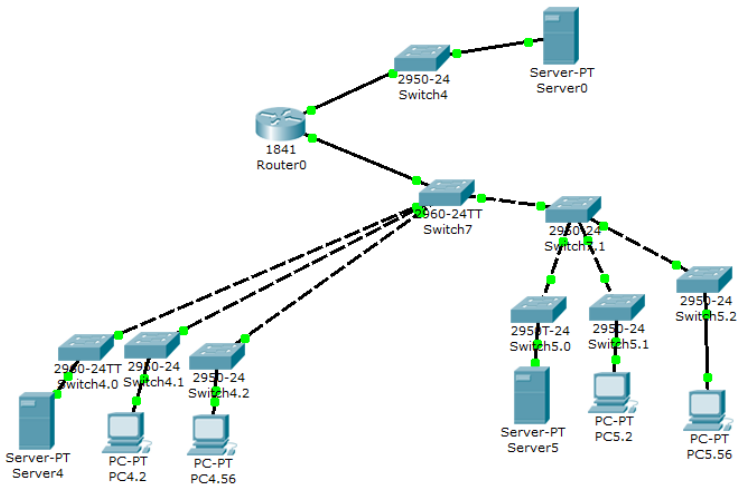


Рисунок 3.1 –Модель мережі

Для створення підмереж треба зайти у закладку «Config» свіча та створити додати необхідну кількість підмереж («VLAN Database»), вказавши в поле VLAN Number номер мережі, починаючи з 2 (1й – зарезервовано), VLAN Name – ім'я мережі (рисунок 3.2).

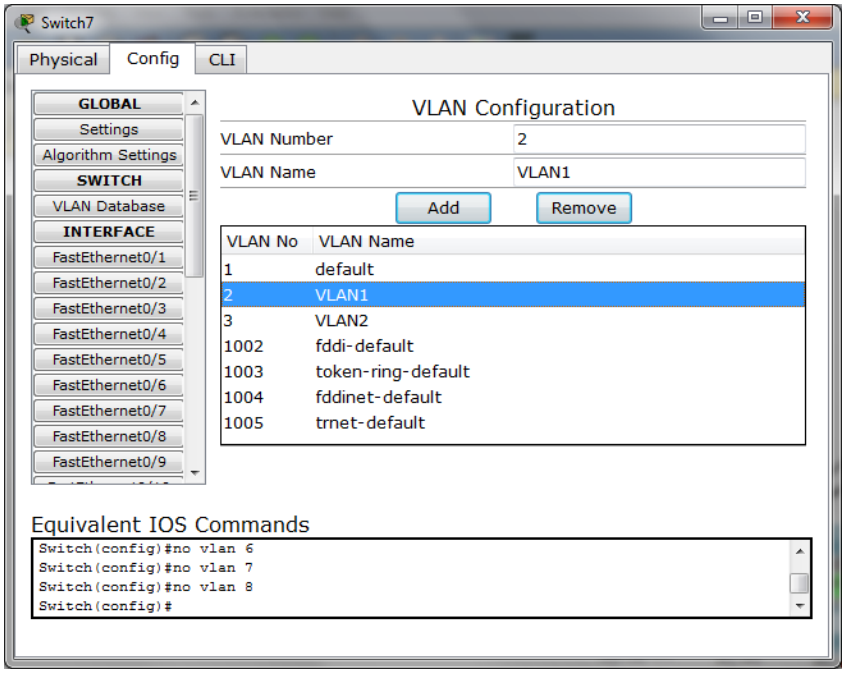


Рисунок 3.2 – Скриншот створення підмереж в базі

На кожний з портів центрального свіча встановити необхідну VLAN (рисунок 3.3).

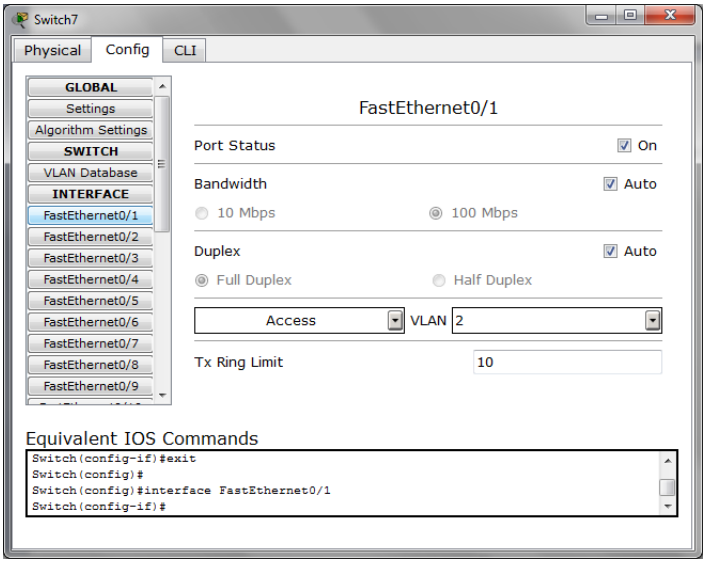


Рисунок 3.3 – Скриншот налаштування портів для підмереж

Підключення VLAN1 було здійснено до портів FastEthernet0/1, FastEthernet0/2, FastEthernet0/3; VLAN2 до порту FastEthernet0/4.

Всі підключені порти свіча будуть мати значення «UP» при наведенні на свіч курсором (рисунок 3.4).

Port	Link	VLAN	IP Address	MAC Address
FastEthernet0/1	Up	2	--	000D.BD2D.8401
FastEthernet0/2	Up	2	--	000D.BD2D.8402
FastEthernet0/3	Up	2	--	000D.BD2D.8403
FastEthernet0/4	Up	3	--	000D.BD2D.8404
FastEthernet0/5	Up	--	--	000D.BD2D.8405
FastEthernet0/6	Down	1	--	000D.BD2D.8406
FastEthernet0/7	Down	1	--	000D.BD2D.8407
FastEthernet0/8	Down	1	--	000D.BD2D.8408
FastEthernet0/9	Down	1	--	000D.BD2D.8409
FastEthernet0/10	Down	1	--	000D.BD2D.840A
FastEthernet0/11	Down	1	--	000D.BD2D.840B
FastEthernet0/12	Down	1	--	000D.BD2D.840C
FastEthernet0/13	Down	1	--	000D.BD2D.840D
FastEthernet0/14	Down	1	--	000D.BD2D.840E
FastEthernet0/15	Down	1	--	000D.BD2D.840F
FastEthernet0/16	Down	1	--	000D.BD2D.8410
FastEthernet0/17	Down	1	--	000D.BD2D.8411
FastEthernet0/18	Down	1	--	000D.BD2D.8412
FastEthernet0/19	Down	1	--	000D.BD2D.8413
FastEthernet0/20	Down	1	--	000D.BD2D.8414
FastEthernet0/21	Down	1	--	000D.BD2D.8415
FastEthernet0/22	Down	1	--	000D.BD2D.8416
FastEthernet0/23	Down	1	--	000D.BD2D.8417
FastEthernet0/24	Down	1	--	000D.BD2D.8418
Vlan1	Down	1	<not set>	00E0.B042.E726

Hostname: Switch

Рисунок 3.4 – Скриншот налагодженого свіча

До порту FastEthernet0/5 свіча необхідно підключити маршрутизатор до порту FastEthernet0/1. Для цього необхідно вибрати тип підключення «Trunk» на свічі (рисунок 3.5).

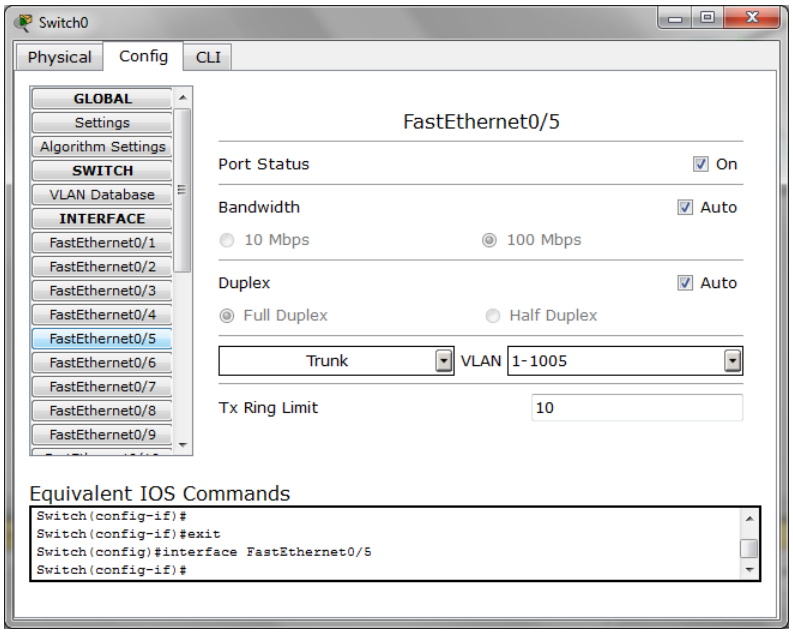


Рисунок 3.5 – Скриншот налаштування порту для маршрутизатора

Після цього слід налаштувати маршрутизатор. Для цього треба зайти у закладку «CLI» маршрутизатора і виконати такі команди :

- Включення роутера:

Router>enable

- Команда для налаштування:

Router#configure terminal

- Вибір порту, який буде налаштовуватись (0/1 – порт маршрутизатора; цифра після крапки – це номер порту свіча):

Router(config-subif)#interface fastEthernet 0/1.1

- Встановлення номеру VLAN (після Dot1Q вказується номер VLAN):

Router(config-subif)#encapsulation Dot1Q 2

- Встановлення ip адреси (ip адреса гейтвею і маска підмережі):

Router(config-subif)#ip address 10.1.0.62 255.255.255.192

Таким чином слід налаштувати маршрутизатор для всіх VLAN, в залежності від того до якого з портів свіча вони під'єднані.

Налаштунки роутера зображено на рисунку 3.6.

Port	Link	VLAN	IP Address	IPv6 Address	MAC Address
FastEthernet0/0	Down	--	<not set>	<not set>	0000.0CB1.D501
FastEthernet0/1	Up	--	<not set>	<not set>	0000.0CB1.D502
FastEthernet0/1.1	Up	--	10.1.0.62/26	<not set>	0000.0CB1.D502
FastEthernet0/1.2	Up	--	<not set>	<not set>	0000.0CB1.D502
FastEthernet0/1.3	Up	--	<not set>	<not set>	0000.0CB1.D502
FastEthernet0/1.4	Up	--	10.1.0.126/26	<not set>	0000.0CB1.D502
Vlan1	Down	1	<not set>	<not set>	0040.0B12.3CDE

Hostname: Router

Physical Location: Intercity, Home City, Corporate Office, Wiring Closet

Рисунок 3.6 – Скриншот налагодженого роутера

Для створення зовнішньої мережі необхідно до іншого порту маршрутизатора (FastEthernet0/0) підключити інший комутатор до порту FastEthernet0/1. Також до цього свіча під'єднати сервер з ip адресою гейтвею і маскою (наприклад, 192.168.10.1 255.255.255.0).

Налаштовувати роутер для зовнішньої мережі слід такими командами:

Router>enable

Router#configure terminal

Router(config-subif)#interface fastEthernet 0/0

Router(config-subif)#ip address 192.168.10.1 255.255.255.0

Налаштунки роутера з двома VlAN і зовнішньою мережею зображено на рисунку 3.6.

Port	Link	VLAN	IP Address	IPv6 Address	MAC Address
FastEthernet0/0	Up	--	192.168.10.1/24	<not set>	0000.0CB1.D501
FastEthernet0/1	Up	--	<not set>	<not set>	0000.0CB1.D502
FastEthernet0/1.1	Up	--	10.1.0.62/26	<not set>	0000.0CB1.D502
FastEthernet0/1.2	Up	--	<not set>	<not set>	0000.0CB1.D502
FastEthernet0/1.3	Up	--	<not set>	<not set>	0000.0CB1.D502
FastEthernet0/1.4	Up	--	10.1.0.126/26	<not set>	0000.0CB1.D502
Vlan1	Down	1	<not set>	<not set>	0040.0B12.3CDE

Hostname: Router

Physical Location: Intercity, Home City, Corporate Office, Wiring Closet

Рисунок 3.7 – Скриншот налагодженого роутера з двома VlAN і зовнішньою мережею

3.2.3 Аналіз і тестування мережі

Перевірка мережі до підключення маршрутизатора:

Відправлення пакетів на робочі станції, які знаходяться в одній віртуальній мережі (рисунок 3.8).

Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit	Delete
	Successful	Server5	PC5.2	ICMP		0.000	N	0	(edit)	(delete)
	Successful	PC5.56	Server5	ICMP		0.000	N	1	(edit)	(delete)
	Successful	PC4.56	PC4.2	ICMP		0.000	N	2	(edit)	(delete)
	Successful	Server4	PC4.56	ICMP		0.000	N	3	(edit)	(delete)

Рисунок 3.8– Скриншот перевірки передачі пакетів в одній віртуальній мережі

Відправлення пакетів на робочі станції, які знаходяться в різних віртуальних мережах (рисунок 3.9).

Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit	Delete
	Failed	Server5	Server4	ICMP		0.000	N	0	(edit)	(delete)
	Failed	PC5.56	PC4.2	ICMP		0.000	N	1	(edit)	(delete)
	Failed	PC4.56	Server5	ICMP		0.000	N	2	(edit)	(delete)
	Failed	PC5.2	Server4	ICMP		0.000	N	3	(edit)	(delete)

Рисунок 3.9 – Скриншот перевірки передачі пакетів в різні віртуальні мережі

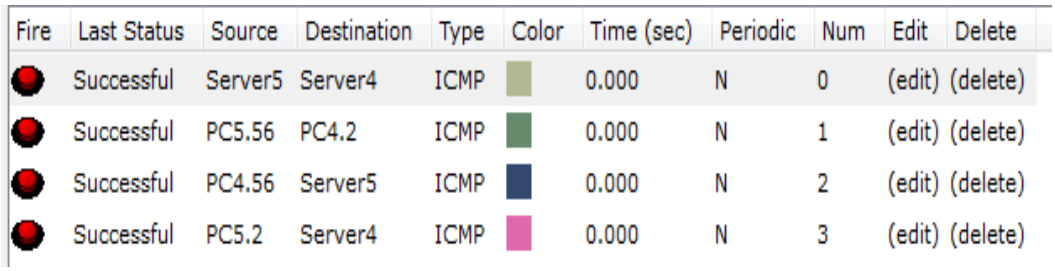
Перевірка мережі після підключення маршрутизатора:

- Відправлення пакетів на робочі станції, які знаходяться в одній віртуальній мережі (рисунок 3.10).

Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit	Delete
	Successful	Server5	PC5.2	ICMP		0.000	N	0	(edit)	(delete)
	Successful	PC5.56	Server5	ICMP		0.000	N	1	(edit)	(delete)
	Successful	PC4.56	PC4.2	ICMP		0.000	N	2	(edit)	(delete)
	Successful	Server4	PC4.56	ICMP		0.000	N	3	(edit)	(delete)

Рисунок 3.10– Скриншот перевірки передачі пакетів в одній віртуальній мережі

- Відправлення пакетів на робочі станції, які знаходяться в різних віртуальних мережах (рисунок 3.11).



Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit	Delete
	Successful	Server5	Server4	ICMP		0.000	N	0	(edit)	(delete)
	Successful	PC5.56	PC4.2	ICMP		0.000	N	1	(edit)	(delete)
	Successful	PC4.56	Server5	ICMP		0.000	N	2	(edit)	(delete)
	Successful	PC5.2	Server4	ICMP		0.000	N	3	(edit)	(delete)

Рисунок 3.11 – Скриншот перевірки передачі пакетів в різні віртуальні мережі

- Виконання трасування пакетів до робочих станцій, які знаходяться в різних віртуальних мережах.

```
SERVER>tracert 10.1.0.121
Tracing route to 10.1.0.121 over a maximum of 30 hops:
  1    15 ms      9 ms      14 ms     10.1.0.62
  2    30 ms     30 ms      30 ms     10.1.0.121

Trace complete.

SERVER>tracert 10.1.0.66
Tracing route to 10.1.0.66 over a maximum of 30 hops:
  1    12 ms     11 ms      14 ms     10.1.0.62
  2    27 ms     13 ms      28 ms     10.1.0.66
Trace complete.
```

- Виконання трасування пакетів до зовнішньої мережі, використовувати IP адресу сервера.

```
SERVER>tracert 192.168.10.2
Tracing route to 192.168.10.2 over a maximum of 30 hops:
  1    11 ms     12 ms      13 ms     10.1.0.62
  2    20 ms     22 ms      22 ms     192.168.10.2
Trace complete.
```

3.3 Завдання

1 Створити модель мережі, що складається з 2 підмереж, що входять до складу двох VLAN з номерами Номер Варіанта+1 і X. Розбити базову підмережу по масці Y. Використовувати адресу мережі, X і Y згідно з варіантом завдання.

2 З'єднати мережі 2-ма комутаторами, налаштувати порти комутатора для роботи з відповідними VLAN і з'єднати комутатори між собою.

3 До одного з комутаторів підключити маршрутизатор з 2 мережевими інтерфейсами.

4 До одного з комутаторів підключити маршрутизатор з 2 мережевими інтерфейсами. Зовнішня мережа емулюється комутатором і сервером.

3.3.1 Хід роботи

1. Визначити адресу мережі, X і Y для свого варіанту (таблиця 3.2), варіант згідно з номером за списком.

Таблиця 3.2 – Варіанти завдань

Номер варіанту	Базова адреса мережі	X номер 2-го VLAN	Y маска підмережі
1	192.168.0.0	7	/ 24
2	172.16.0.0	9	/ 23
3	172.17.0.0	8	/ 25
4	172.18.0.0	11	/ 28
5	172.19.0.0	10	/ 27
6	10.1.0.0	6	/ 26
7	10.2.0.0	7	/ 23
8	10.3.0.0	8	/ 28
9	10.4.0.0	6	/ 24
10	10.5.0.0	10	/ 25
11	10.6.0.0	7	/ 26
12	10.7.0.0	8	/ 27

2. Розрахувати які IP адреси потрапляють в задану маску підмережі, для реалізації потрібної кількості підмереж.

3. У Packet Tracer 5.0 створити модель LAN такою, що складається з двох підмереж, кількість робочих станцій в кожній підмережі моделі визначається початковою і кінцевою адресою. Таким чином в кожній підмережі буде 2 робочі станції, IP адреси яких будуть початковими і кінцевми адресами хостів для цієї підмережі.

4. З'єднати комутатори Cisco Catalyst 2960 між собою і підключити до них робочі станції.

5. Об'єднати робочі станції в різні віртуальні мережі, згідно з варіантом завдання.

6. Провести аналіз і тестування мережі, згідно з алгоритмом наведеному нижче:

- до одного з комутаторів підключити один з інтерфейсів маршрутизатора;
- налаштувати порт комутатора для пропускання всіх Vlan-ів на маршрутизатор;
- на маршрутизаторі створити 2 підінтерфейси, з IP адресою, маскою і номером відповідного Vlan-а;
- на другому інтерфейсі маршрутизатора налаштувати IP адреси з зовнішньої мережі, для зовнішньої підмережі використовувати довільні IP адреси, що не потрапляють в діапазон локальних адрес;

- з'єднати другий інтерфейс маршрутизатора з комутатором зовнішньої мережі, до якого підключений сервер;
- на робочих станціях прописати основний шлюз, відповідний IP адресі підінтерфейсу маршрутизатора для даного VLAN.

7. Знімок екрану моделі мережі, налаштування комутатора і маршрутизатора надати в звіті.

8. Аналіз і тестування мережі:

1) До підключення маршрутизатора:

- відправити пакети на робочі станції, що знаходяться в одній віртуальній мережі. Відобразити результати в звіті;
- відправити пакети на робочі станції, що знаходяться в різних віртуальних мережах. Відобразити результати в звіті.

2) Після підключення маршрутизатора:

- відправити пакети на робочі станції, що знаходяться в одній віртуальній мережі. Відобразити результати в звіті;
- відправити пакети на робочі станції, що знаходяться в різних віртуальних мережах. Відобразити результати в звіті;
- виконати трасування пакетів до робочих станцій, що знаходяться в різних віртуальних мережах. Відобразити результати в звіті;
- виконати трасування пакетів до зовнішньої мережі, використовувати IP адреса сервера. Відобразити результати в звіті.

3.4 Контрольні питання

1. Якою командою призначається VLAN на порт комутатора?
2. Як проглянути список VLAN?
3. Як проглянути налаштування порту?
4. Як поглянути всі налаштування комутатора?
5. Як створити підінтерфейс на маршрутизаторі?
6. Як додати підінтерфейс маршрутизатора в потрібний VLAN?
7. Як перевірити трасу по якій йдуть пакети до кінцевої крапки?
8. Як проглянути всі налаштування маршрутизатора?
9. Як включити маршрутизацію?
10. Якими властивостями повинен володіти порт комутатора, до якого підключений маршрутизатор?

4 САМОСТІЙНА РОБОТА № 4
ДОСЛІДЖЕННЯ РОБОТИ З ПОТОКОВИМИ СОКЕТАМИ В РЕЖИМІ ОПИТУ

Мета роботи: вивчити роботу з потоковими сокетами в режимі опиту. Створити сервер котрий буде відповідати на запити клієнтів.

4.1 Теоретичні відомості

Socket API був вперше реалізований в операційній системі UNIX. Зараз цей програмний інтерфейс доступний практично в будь-якій операційній системі. Хоча всі реалізації чимось відрізняються один від одного, основний набір функцій у них збігається. Спочатку сокети використовувалися в програмах на C/C++, але в даний час вони є майже в всіх нових мовах програмування (Perl, C#, Java та ін.).

Сокети надають дуже потужний і гнучкий механізм взаємодії між процесами (IPC). Вони можуть використовуватися для організації взаємодії програм на одному комп'ютері, по локальній мережі або через Інтернет, що дозволяє вам створювати розподілені додатки різної складності. Крім того, з їх допомогою можна організувати взаємодію з програмами, що працюють під управлінням інших операційних систем.

Сокети підтримують багато стандартних мережевих протоколів (конкретний їх список залежить від реалізації) і надають уніфікований інтерфейс для роботи з ними. Найбільш часто сокети використовуються для роботи в IP-мережах.

Сокети, незалежно від виду, поділяються на три типи: потокові, сирі і дейтаграмні. Потокові сокети працюють з установкою з'єднання, забезпечуючи надійну ідентифікацію обох сторін і гарантують цілісність і успішність доставки даних, спираючись на протокол TCP. Дейтаграмні сокети працюють без встановлення з'єднання і не забезпечують ні ідентифікації відправника, ні контролю успішності доставки даних, зате вони швидше потокових, спираючись на протокол UDP. Сирі сокети, вони надають можливість ручного формування TCP \ IP-пакетів.

Також існує 2 види сокетів:

- синхронні – затримують управління на час виконання операції;
- асинхронні – повертають управління, але продовжують виконувати роботу в фоні та після закінчення повідомляють про це.

4.1.1 Робота з сокетами

В даному випадку розглянемо роботу з синхронними потоковими сокетами (TCP).

Реалізація сервера:

1) підготувати бібліотеку до використання;
2) створити об’єкт типу Socket;

3) зв’язати цей об’єкт з локальною адресою/портом;
4) перейти в режим очікування;
5) витягнути із черги запит на з’єднання;
6) прийняти/передати дані;
7) закрити сокети, звільнити ресурси.

Реалізація клієнта:

1) підготувати бібліотеку до використання;
2) створити об’єкт типу Socket;
3) з’єднатися з сервером;
4) прийняти/передати дані;
5) закрити сокети, звільнити ресурси.

4.1.2 Реалізація сервера

Крок перший. «Підготовка бібліотеки»

Для роботи з бібліотекою Winsock 2.x потрібно підключити директиву `#include <winsock2.h>` та підготувати саму бібліотеку. Для цього використаємо функцію WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData).

- wVersionRequested – параметр типу WORD, приймає значення версії сокетів (старший байт слова – номер версії, молодший – номер під версії);
- lpWSAData – структура типу WSADATA, в котру при успішній ініціалізації буде занесена інформація про виробника бібліотеки.

Приклад:

```
    WORD sockVersion;
    WSADATA wsaData;
    sockVersion = MAKEWORD(2,2);
WSAStartup(sockVersion, &wsaData);
```

В UNIX подібних системах потрібно підключити такі директиви:

```
#include <sys/types.h>
#include <sys/socket.h>
```

Крок другий. «Створення об’єкту типу Socket»

Створимо об’єкт типу socket. Використовуємо функцію SOCKET s (int af, int type, int protocol).

- af - сімейство протоколів (Зазвичай використовують AF_INET, тобто Internet протоколи.);
- type – тип сокету – спосіб передачі даних через мережу. В даному курсі ми будемо використовувати: SOCK_STREAM з встановленням з’єднання, використовується в основному з TCP та SOCK_DFRAM без встановлення з’єднання, використовуються з UDP;
- protocol – протокол для передачі даних.

Функція повертає дескриптор сокету, інакше INVALID_SOCKET.

Приклад:

```
SOCKET s = socket (AF_INET, SOCK_STREAM, 0);
```

```
// 0 – параметр за про мовчанням, означає TCP.
```

Крок третій «З'єднання сокета з локальною адресою»

Клієнт при підключені повинен вказати адресу/порт сервера. Отже на стороні сервера ми повинні вказати адресу/порт, а потім поєднати їх з сокетом. Для цього потрібно використати функцію *int **bind** (SOCKET s, const struct sock-addr FAR* name, int namelen)*.

s – дескриптор сокету з котрим ми поєднаємо адресу/порт;

name - структура sockaddr, котра містить адресу/порт;

namelen – довжина структури sockaddr.

Якщо все добре, то функція повертає «0».

Структура *sockaddr* має вигляд:

```
struct sockaddr {
    unsigned short    sa_family;  // сімейство адрес, AF_xxx
    char              sa_data[14];// 14 байтів для
                                  // зберігання адреси
};
```

Так як працювати з *sockaddr* не зручно, то можна використовувати альтернативну структуру *sockaddr_in:*

```
struct sockaddr_in {
    short int          sin_family;  // сімейство адресів
    unsigned short int sin_port;    // номер порта
    struct in_addr     sin_addr;    // IP-адреса
    unsigned char      sin_zero[8]; // "доповнення" до
                                    //  розміру структури
                                    //  sockaddr
};
```

Приклад:

```
sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(8888);
sin.sin_addr.s_addr = INADDR_ANY;
result = bind(s, (LPSOCKADDR)&sin, sizeof(sin));
```

Крок четвертий «Режим очікування»

Коли ми створили сокет, пов'язали його з певною адресою/портом, далі ми повинні перейти в режим очікування, тобто очікування клієнтів на підключення. Для цього використовуємо функцію *int **listen** (SOCKET s, int bac_klog)*.

s – дескриптор сокету;

bac_klog – максимальний розмір черги на підключення (кількість з'єднань, котрі сервер може одночасно оброблювати).

Приклад:

```
result = listen(s, 5);
```

Крок п’ятий «Витягти клієнта з черги»

В режимі очікування ми чекаємо поки клієнти не з’явиться в черзі на підключення (запит на з’єднання). Як тільки в черзі з’являється запит на підключення, витягнути звідти його можна за допомогою функції *SOCKET* ***accept*** *(SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen)..*

s- дескриптор сокету;

addr- структура sockaddr. При підключені клієнта в неї записується його адреса/порт. Якщо нас не цікавить адреса клієнта, можна передати NULL;

addrlen – довжина структури sockaddr;

Функція створює новий об’єкт типу *Socket*, з котрим ми потім будемо працювати.

Приклад:

```
SOCKET client;
result = accept(s, NULL, NULL);
```

Крок шостий «Прийом/передача даних»

Нам потрібно реалізувати обмін інформацією між сервером та клієнтом.

Для прийому повідомлень використовують функцію *int* ***send*** *(SOCKET s, const char FAR * buf, int len,int flags).*

s - дескриптор сокету клієнта;

buf – буфер в котрий ми отримаємо повідомлення;

len – розмір буфера;

flags – прапорці, (Якщо не використовуємо то «0»).

Повертає кількість прийнятих байт.

Функція *send* повертає управління відразу після виконання, незалежно від того чи отримала інша сторона інформацію.

Для відправлення повідомлень використовують функцію *int* ***recv*** *(SOCKET s, char FAR* buf, int len, int flags).*

s - дескриптор сокету клієнта;

buf – буфер з повідомленням;

len – розмір буфера;

flags – прапорці, (Якщо не використовуємо то «0»).

Функція recv повертає управління тільки після того як отримала інформацію.

Приклад:

```
SOCKET client;
result = accept(s, NULL, NULL);

     char recv_buf[40];
     char send_buf[40] = "ANSWER";

result = recv(client, recv_buf, 40, 0);
     send(client, send_buf, 40, 0);
```

Крок сьомий «Закрити сокет, звільнити ресурси»

Після роботи с клієнтом потрібно закрити з'єднання. Також не забуваємо після закінчення роботи сервера закривати основний сокет, робити де ініціалізацію бібліотеки, звільнення ресурсів.

Приклад

```
closesocket(client);
…
closesocket(s);
WSACleanup();
```

В UNIX подібних системах використовують:

```
#include <unistd.h>
int close(int fd);
```

4.1.3 Реалізація клієнта

Крок перший. «Підготовка бібліотеки»

Для роботи з бібліотекою Winsock 2.x потрібно підключити директиву #include <winsock2.h> та підготувати саму бібліотеку. Для цього використаємо функцію *WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData).*

- *wVersionRequested* – параметр типу WORD, приймає значення версії сокетів (старший байт слова – номер версії, молодший – номер під версії);
- *lpWSAData* – структура типу WSADATA, в котру при успішній ініціалізації буде занесена інформація про виробника бібліотеки.

Приклад:

```
    WORD sockVersion;
    WSADATA wsaData;
    sockVersion = MAKEWORD(2,2);
WSAStartup(sockVersion, &wsaData);
```

В UNIX подібних системах потрібно підключити такі директиви:

```
#include <sys/types.h>
#include <sys/socket.h>
```

Крок другий. «Створення об'єкту типу Socket»

Створимо об'єкт типу *socket*. Використовуємо функцію *SOCKET s (int af, int type, int protocol).*

- *af* - сімейство протоколів (Зазвичай використовують AF_INET, тобто Internet протоколи.);
- *type* – тип сокету – спосіб передачі даних через мережу. В даному курсі ми будемо використовувати: *SOCK_STREAM* з встановленням з'єднання, використовується в основному з *TCP* та *SOCK_DFRAM* без встановлення з'єднання, використовуються з *UDP*;
- *protocol* – протокол для передачі даних;

Функція повертає дескриптор сокету, інакше INVALID_SOCKET.

Приклад:

```
SOCKET s = socket (AF_INET, SOCK_STREAM, 0);
// 0 – параметр за про мовчанням, означає TCP.
```

Крок третій «З'єднання з сервером»

На стороні клієнта нам потрібно з'єднатися з сервером вказавши адресу/порт сервера, для цього використовуємо функцію *int **connect** (SOCKET s, const struct sockaddr FAR* name, int namelen)*.

s – дескриптор сокету;

name – структура *sockaddr*, котра містить в собі адресу і порт віддаленого вузла, з котрим ми будемо з'єднуватися;

namelen – розмір структури *sockaddr*.

Приклад:

```
hostEntry = gethostbyname("127.0.0.1");
SOCKADDR_IN serverInfo;
serverInfo.sin_family = AF_INET;
serverInfo.sin_addr      =      *((LPIN_ADDR)*hostEntry-
>h_addr_list);
serverInfo.sin_port = htons(8888);

result=connect(s,(LPSOCKADDR)&serverInfo,
sizeof(serverInfo));
```

Крок четвертий «Прийом/передача даних»

Нам потрібно реалізувати обмін інформацією між сервером та клієнтом.

Для прийому повідомлень використовують функцію *int **send** (SOCKET s, const char FAR * buf, int len,int flags)*.

s - дескриптор сокету клієнта;

buf – буфер в котрий ми отримаємо повідомлення;

len – розмір буфера;

flags – прапорці.

Вона повертає кількість прийнятих байт.

Функція *send* повертає управління відразу після виконання, незалежно від того чи отримала інша сторона інформацію.

Для відправлення повідомлень використовують функцію *int **recv** (SOCKET s, char FAR* buf, int len, int flags)*.

s - дескриптор сокету клієнта;

buf – буфер з повідомленням;

len – розмір буфера;

flags – прапорці.

Функція recv повертає управління тільки після того як отримала інформацію.

Приклад:

```
SOCKET client;
result = accept(s, NULL, NULL);
```

```
      char recv_buf[40];
      char send_buf[40] = "ANSWER";

result = recv(client, recv_buf, 40, 0);
      send(client, send_buf, 40, 0);
```

Крок п’ятий «Закрити сокет, звільнити ресурси»

Після роботи с клієнтом потрібно закрити з’єднання. Також не забуваємо після закінчення роботи сервера закривати основний сокет, робити де ініціалізацію бібліотеки, звільнення ресурсів.

Приклад

```
closesocket(client);
…
closesocket(s);
WSACleanup();
```

В UNIX подібних системах використовують:

```
#include <unistd.h>
int close(int fd);
```

4.1.4 Приклад роботи з WinSock

Реалізація сервера:

```
#include "stdafx.h"
#include <winsock2.h>

#pragma comment(lib, "wsock32.lib")

#define SERVER_SOCKET_ERROR 1
#define SOCKET_OK 0

int _tmain(int argc, _TCHAR* argv[])
{
      int result;
      WORD sockVersion;
      WSADATA wsaData;
      sockVersion = MAKEWORD(2,2);
      WSAStartup(sockVersion, &wsaData);
sockaddr_in sin;
      sin.sin_family = AF_INET;
      sin.sin_port = htons(8888);
      sin.sin_addr.s_addr = INADDR_ANY;
      SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
      if(s == INVALID_SOCKET)
      {
            printf ("%s", "ERROR  (don't create serrver)\n");
            WSACleanup();
            return SERVER_SOCKET_ERROR;
      }
      else
      {
                  printf ("%s", "  >>> Create socket \n");
      }

result = bind(s, (LPSOCKADDR)&sin, sizeof(sin));
      if(result == SOCKET_ERROR)
      {
            printf ("%s", "ERROR  (don't associates a
```

```
                        local address with a socket)");
      WSACleanup();
      return SERVER_SOCKET_ERROR;
}
else
{
      printf ("%s", "  >>> Associates a local
                                    addres with a socket\n");
}

result = listen(s, 5);
if(result == SOCKET_ERROR)
{
      printf ("%s", "  ERROR (don't listen a socket )\n");
      WSACleanup();
      return SERVER_SOCKET_ERROR;
}
else
{
      printf ("%s", "  >>> get socket to listen \n");
}

while (1)
{
      SOCKET client;
      client = accept(s, NULL, NULL);
      if(client == INVALID_SOCKET)
      {
            printf ("%s", "  >>> ERROR  (don't
                                    accept a new socket)\n");
            WSACleanup();
            return SERVER_SOCKET_ERROR;
      }
      else
      {
            printf ("%s", "  >>> new socket
                        client ACCEPT (client found)\n");
      }

      char recv_buf[40];
      char send_buf[40] = "ANSWERRRRRRR";

      result = recv(client, recv_buf, 40, 0);
      recv_buf[result]=0;
      send(client, send_buf, 40, 0);

      printf ("%s", recv_buf );
      printf ("%s", "\n" );

      if(result == SOCKET_ERROR)
      {
            int val = WSAGetLastError();
            if(val == WSAENOTCONN)
            {
                  printf ("%s", " ERROR  (socket not
                                          connected) ?? \n");
            }
            else if(val == WSAESHUTDOWN )
            {
                  printf ("%s", " ERROR (socket has
                                    been shut down!)  ?? \n");
            }
            printf ("%s", " ERROR (FAILD RECV)  \n");
            return SERVER_SOCKET_ERROR;
```

```
            }
            closesocket(client);
            printf ("%s", "  >>> Close client socket \n");

      } // END WHILE

      closesocket(s);
      printf ("%s", "  >>> Close main socket \n");
      WSACleanup();
      return SOCKET_OK;
}
```

Реалізація клієнта:

```
      #include "stdafx.h"
      #include <winsock.h>
      #pragma comment(lib, "wsock32.lib")

      #define CS_ERROR 1
      #define CS_OK 0

      int _tmain(int argc, _TCHAR* argv[])
      {
            WORD version;
            WSADATA wsaData;
            int result;
            version = MAKEWORD(2,2);
            WSAStartup(version,(LPWSADATA)&wsaData);
            LPHOSTENT hostEntry;
      hostEntry = gethostbyname("127.0.0.1");

            if(!hostEntry)
            {
            printf ("%s", "  >>> ERROR  (hostEntry NULL)\n");
                  WSACleanup();
                  return CS_ERROR;
            }

            SOCKET theSocket = socket(AF_INET, SOCK_STREAM, 0);
            if(theSocket == SOCKET_ERROR)
            {
            printf ("%s", "  ERROR  (don't create socket)\n");
                  return CS_ERROR;
            }
      else
      {
            printf ("%s", "  >>> Create socket \n");
      }

      sockaddr_in serverInfo;
      serverInfo.sin_family = AF_INET;
            serverInfo.sin_addr = *((LPIN_ADDR)*hostEntry->h_addr_list);
            serverInfo.sin_port = htons(8888);

            result=connect(theSocket,(LPSOCKADDR)&serverInfo,
                                                sizeof(serverInfo));
      if(result==SOCKET_ERROR)
            {
            printf ("%s", "  ERROR (don't connect to Server)\n");
                  return CS_ERROR;
            }
      else
      {
            printf ("%s", "  >>> Connect to Server\n");
      }
```

```
            char send_buf[40] = "";
        char recv_buf[40] = "";
        scanf ("%s", send_buf);

            result = send(theSocket, send_buf, strlen(send_buf), 0);
            if(result == SOCKET_ERROR)
            {
            printf ("%s", "  ERROR (Failed send () )\n");
                  return CS_ERROR;
            }

        recv(theSocket, recv_buf, 40, 0);
        printf ("%s", recv_buf);
        printf ("%s", "\n");

            closesocket(theSocket);
        printf ("%s", "  >>> Close socket\n");
        WSACleanup();
            return CS_OK;
}
```

4.1.5 Приклад роботи для Unix подібних систем

Ехо-Клієнт:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

char message[] = "Hello there!\n";
char buf[sizeof(message)];

int main()
{
    int sock;
    struct sockaddr_in addr;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if(sock < 0)
    {
        perror("socket");
        exit(1);
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(3425);
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
    if(connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("connect");
        exit(2);
    }
    send(sock, message, sizeof(message), 0);
    recv(sock, buf, sizeof(message), 0);
    printf(buf);
    close(sock);
    return 0;
}
```

Ехо-Сервер:

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>

int main()
{
    int sock, listener;
    struct sockaddr_in addr;
    char buf[1024];
    int bytes_read;

    listener = socket(AF_INET, SOCK_STREAM, 0);
    if(listener < 0)
    {
        perror("socket");
        exit(1);
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(3425);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    if(bind(listener, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("bind");
        exit(2);
    }
   listen(listener, 1);
    while(1)
    {
        sock = accept(listener, NULL, NULL);
        if(sock < 0)
        {
            perror("accept");
            exit(3);
        }
        while(1)
        {
            bytes_read = recv(sock, buf, 1024, 0);
            if(bytes_read <= 0) break;
            send(sock, buf, bytes_read, 0);
        }
        close(sock);
    }

    return 0;
}
```

4.2 Завдання

Написати сервер котрий одночасно буде відповідати на запити від клієнтів.

Для студентів котрі хочуть отримати відмінну оцінку: клієнт повинен надсилати щосекунди повідомлення с поточною датою. На стороні сервера реалізувати перевірку, якщо прийнята дата парна (секунди), у відповідь відсилати дату що прислав клієнт та повідомлення «Число парне», інакше «Число не парне»;

4.3 Контрольні питання

1. Що таке сокет? Які існують типи сокетів?
2. Яка функція використовується при створені сокету? Охарактеризуйте параметри котрі передаються у функцію?

3. Навіщо вказувати на стороні сервера адресу/порт?

4. Поясніть, чому при адресації сокетів використовуються різні структури перетворення типів?

5. Яка функція використовується для читання даних із сокету? Охарактеризуйте параметри котрі передаються у функцію?

6. Які функції обов’язково повинні бути присутніми при створення сервера/клієнта? Охарактеризуйте відповідь?

5 САМОСТІЙНА РОБОТА № 5
ДОСЛІДЖЕННЯ РОБОТИ З ДАТАГРАМНИМИ СОКЕТАМИ В РЕЖИМІ ОПИТУ

Мета роботи: вивчити роботу з датаграмними сокетами в режимі опиту. Створити сервер котрий буде відповідати на запити клієнтів.

5.1 Теоретичні відомості

Датаграмні сокети використовуються в програмах не так часто, оскільки надійність дуже низька. Корисні вони тоді, коли потрібно постійно передавати звук і відео по мережі. Оскільки для датаграмних сокетів не потрібно встановлювати з'єднання, використовувати їх значно простіше.

Реалізація сервера:

1) підготувати бібліотеку до використання;
2) створити об'єкт типу Socket;
3) зв'язати цей об'єкт з локальною адресою/портом;
4) прийняти/передати дані;
5) закрити сокети, звільнити ресурси.

Реалізація клієнта:

1) підготувати бібліотеку до використання;
2) створити об'єкт типу Socket;
3) прийняти/передати дані;
4) закрити сокети, звільнити ресурси.

Створивши сокет за допомогою *socket* та *bind*, ви вже можете використовувати його для відправки і прийому повідомлень за допомогою функцій *sendto* та *recvfrom*.

Для відправлення повідомлення використовують функцію *int* ***sendto*** *(SOCKET s, const char FAR * buf, int len,int flags, const struct sockaddr FAR * to, int tolen)*.

Вона майже така як і *send*, але має 2 додаткових параметри:

- *to* – вказуємо структуру *sockaddr*, в котрій є адреса призначення;
- *tolen* – розмір структури.

Для прийому даних використовують функцію *int* ***recvfrom*** *(SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen).*

Відповідно і тут є 2 додаткових параметри, один із них структура *sockaddr*, в котру ми запишемо адресу/порт відправника (на цю ж адресу і будемо відповідати в зворотному направці) і саме розмір цієї структури.

Доречі, можна використовувати і функції *send* та *recv*, зробивши *connect*, але майте на увазі, що ніякого з'єднання не відбувається, ми просто запам'ятовуємо адресу.

5.1.1 Приклад роботи WinSock

Реалізація сервера:

```
#include "stdafx.h"
#include <winsock2.h>
#pragma comment(lib, "wsock32.lib")

#define SERVER_SOCKET_ERROR 1
#define SOCKET_OK 0

int _tmain(int argc, _TCHAR* argv[])
{
	int result;
	WORD sockVersion;
	WSADATA wsaData;
	sockVersion = MAKEWORD(2,2);
	WSAStartup(sockVersion, &wsaData);
	sockaddr_in sin;
	sin.sin_family = AF_INET;
	sin.sin_port = htons(8888);
	sin.sin_addr.s_addr = INADDR_ANY;

	SOCKET s = socket(AF_INET, SOCK_DGRAM, 0);
	if(s == INVALID_SOCKET)
	{
		printf ("%s", "ERROR (don't create server)\n");
		WSACleanup();
		return SERVER_SOCKET_ERROR;
	}
	else
	{
		printf ("%s", " >>> Create socket \n");
	}

	result = bind(s, (LPSOCKADDR)&sin, sizeof(sin));
	if(result == SOCKET_ERROR)
	{
		printf ("%s", "ERROR (don't associates a local address with a
socket)");
		WSACleanup();
		return SERVER_SOCKET_ERROR;
	}
	else
	{
		printf ("%s", " >>> Associates a local addres with a sock-
et\n");
	}

	while (1)
	{
		char recv_buf[1024];
		char send_buf[1024] = "ANSWER\n";

		sockaddr_in client_addr;
		int client_addr_size = sizeof(client_addr);

		int result = recvfrom ( s, recv_buf, 1024, 0, (LPSOCK-
ADDR)&client_addr, &client_addr_size);
		if (result == SOCKET_ERROR)
			printf("	ERROR	recvfrom()	error:
%d\n",WSAGetLastError());

		HOSTENT *hst;
```

```
                hst=gethostbyaddr((char *) &client_addr.sin_addr,4,AF_INET);
                printf(                        inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));
                recv_buf[result]=0;
                printf(" = Message from client: %s\n",&recv_buf[0]);
                sendto    (s,&recv_buf[0],40,0,    (sockaddr    *)&client_addr,
sizeof(client_addr));
          }
          closesocket(s);
          printf ("%s", " >>> Close main socket \n");
          WSACleanup();
          return SOCKET_OK;
    }
```

Реалізація клієнта:

```
#include "stdafx.h"
#include <winsock.h>
#pragma comment(lib, "wsock32.lib")

#define CS_ERROR 1
#define CS_OK 0

int _tmain(int argc, _TCHAR* argv[])
{
      WORD version;
      WSADATA wsaData;
      int result;
      version = MAKEWORD(2,2);
      WSAStartup(version,(LPWSADATA)&wsaData);

LPHOSTENT hostEntry;
hostEntry = gethostbyname("127.0.0.1");
      if(!hostEntry)
      {
            printf ("%s", "  >>> ERROR  (hostEntry NULL)\n");
            WSACleanup();
            return CS_ERROR;
      }

      SOCKET theSocket = socket(AF_INET, SOCK_DGRAM, 0);
      if(theSocket == SOCKET_ERROR)
      {
            printf ("%s", "  ERROR  (don't create socket)\n");
            return CS_ERROR;
      }
      else
      {
            printf ("%s", "  >>> Create socket \n");
      }

      SOCKADDR_IN serverInfo;
      serverInfo.sin_family = AF_INET;
      serverInfo.sin_port = htons(8888);
      serverInfo.sin_addr.s_addr=inet_addr("127.0.0.1");

      while(1)
      {
            char send_buf[1000] = "";
            char recv_buf[400] = "";
            printf("\nWrite message: ");
            scanf ("%s", send_buf);
            if (!strcmp(&send_buf[0],"quit")) break;

            sendto(theSocket,&send_buf[0], strlen(&send_buf[0]),0,
```

```
                                    (sockaddr                                   *)
&serverInfo,sizeof(serverInfo));

                  sockaddr_in server_addr;
                  int server_addr_size=sizeof(server_addr);
                  int n=recvfrom (theSocket,&recv_buf[0], 999, 0 ,
                              (sockaddr *) &server_addr, &server_addr_size);
                  if (n==SOCKET_ERROR)
                  {
                        printf("recvfrom() error: %d\n",WSAGetLastError());
                        closesocket(theSocket);
                        WSACleanup();
                  }
                  recv_buf[n]=0;
                  printf(" Answer from Server: %s",&recv_buf[0]);
            }

            closesocket(theSocket);
            printf ("%s", "  >>> Close socket\n");
            WSACleanup();
            char a[100];
            scanf ("%s", a);
            return CS_OK;
      }
```

5.1.2 Приклад роботи (для UNIX подібних систем)

Программа sender:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

char msg1[] = "Hello there!\n";
char msg2[] = "Bye bye!\n";

int main()
{
    int sock;
    struct sockaddr_in addr;

    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if(sock < 0)
    {
        perror("socket");
        exit(1);
    }
    addr.sin_family = AF_INET;
    addr.sin_port = htons(3425);
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
    sendto(sock, msg1, sizeof(msg1), 0,
           (struct sockaddr *)&addr, sizeof(addr));

    connect(sock, (struct sockaddr *)&addr, sizeof(addr));
    send(sock, msg2, sizeof(msg2), 0);

    close(sock);

    return 0;
}
```

Программа receiver:

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <stdio.h>

int main()
{
    int sock;
    struct sockaddr_in addr;
    char buf[1024];
    int bytes_read;

    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if(sock < 0)
    {
        perror("socket");
        exit(1);
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(3425);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    if(bind(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("bind");
        exit(2);
    }

    while(1)
    {
        bytes_read = recvfrom(sock, buf, 1024, 0, NULL, NULL);
        buf[bytes_read] = '\0';
        printf(buf);
    }

    return 0;
}
```

5.2 Завдання

Написати клієнт-серверний застосунок, котрий буде працювати з датаграмними сокетами.

Для студентів котрі хочуть отримати відмінну оцінку: клієнт повинен надіслати число від 1 до 10, а у відповідь від сервера отримати стільки повідомлень, скільки було вказано клієнтом;

5.3 Контрольні питання

1. Опишіть принцип роботи датаграмних сокетів?
2. В чому полягає різниця між датаграмними сокетами і потоковими?
3. Назвіть функції для отримання і відправлення повідомлень?
4. Які поля містить структура *sockaddr*?

6 САМОСТІЙНА РОБОТА № 6
ДОСЛІДЖЕННЯ РОБОТИ АСИНХРОННИХ СОКЕТІВ

Мета роботи: вивчення роботи асинхронних сокетами в режимі потоків виконання, тобто в режимі, коли робота з кожним клієнтом проводиться в окремому серверному потоці.

6.1 Теоретичні відомості

Як ви пам'ятаєте, синхронні сокети затримують управління на час виконання операцію, в свою чергу асинхронні повертають управління, але продовжують роботу в фоні та після закінчення повідомляють про це.

У випадку з синхронними сокетами сервер прийнявши нового клієнта працює з ним (обмінюється інформацією), але інші клієнти чекають в черзі, поки сервер не завершить роботу з цим. Асинхронні сокети працюють паралельно - витягує клієнта з черги, породжує потік/процес, передає йому дескриптор клієнта (котрий повернула функція accept), цей потік/процес починає працювати в фоні, в свою чергу сервер знову витягує нового клієнта з черги і так далі.

Асинхронні сокети слід використовувати там, де є велике навантаження при передачі даних.

Реалізація програмної частини залишається такою ж як і при синхронних сонетах. Винятком є те, що ми оброблюємо інформацію в потоці, а саме, витягнувши запит від клієнта з черги, ми передаємо його дескриптор в функцію, котра виконується в потоці.

6.1.1 Текст програми

Реалізація сервера:

```
#include "stdafx.h"
#include <winsock2.h>
#include <conio.h>
#include <process.h>
#pragma comment(lib, "wsock32.lib")

#define SERVER_SOCKET_ERROR 1
#define SOCKET_OK 0

char buffer[]="ANY DATA";
char recv_buf[1000];

void MyFunction( void* Arg )
{
      SOCKET Client=(*(int *)Arg);
      recv(Client,recv_buf,1000,0);
      printf(" = Message from client: %s\n",&recv_buf[0]);
      send(Client,recv_buf,1000,0);
      _endthreadex(0);
}

int _tmain(int argc, _TCHAR* argv[])
{
      int result;
      WORD sockVersion;
```

```
    WSADATA wsaData;
    sockVersion = MAKEWORD(2,2);
    WSAStartup(sockVersion, &wsaData);

    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(8888);
    sin.sin_addr.s_addr = INADDR_ANY;
    int size_sin=sizeof(sin);

    SOCKET Client;
    SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
    if(s == INVALID_SOCKET)
    {
          printf ("%s", "ERROR  (don't create serrver)\n");
          WSACleanup();
          return SERVER_SOCKET_ERROR;
    }
    else
    {
                printf ("%s", "  >>> Create socket \n");
    }

    result = bind(s, (LPSOCKADDR)&sin, sizeof(sin));
    if(result == SOCKET_ERROR)
    {
          printf ("%s", "ERROR  (don't associates a local address
                                                  with a socket)");
      WSACleanup();
      return SERVER_SOCKET_ERROR;
  }
    else
    {
          printf ("%s", "  >>> Associates a local addres with
                                                      a socket\n");
    }

    result = listen(s, 5);
    if(result == SOCKET_ERROR)
    {
          printf ("%s", "  ERROR (don't listen a socket )\n");
          WSACleanup();
          return SERVER_SOCKET_ERROR;
    }
    else
    {
          printf ("%s", "  >>> get socket to listen \n");
    }

    while (1)
    {
          Client=accept(s,(struct sockaddr*)&sin,&size_sin);
          _beginthread(MyFunction,0,(void *)&Client);

    }
    closesocket(s);
    printf ("%s", "  >>> Close main socket \n");
    WSACleanup();
    getch();

  return SOCKET_OK;
}
```

Реалізація клієнта:

```
#include "stdafx.h"
#include<conio.h>
#include <winsock.h>
#include<process.h>
#pragma comment(lib, "wsock32.lib")

#define CS_ERROR 1
#define CS_OK 0

char send_buf[1000] = "";
char recv_buf[1000] = "";

void MyFunction(void * Arg)
{
      while(1)
      {
            int Socket=(*(int *)Arg);
            send(Socket,send_buf,1000,0);
            int n = recv(Socket,recv_buf,1000,0);
            recv_buf[n]=0;
            printf(" Answer from Server: %s",&recv_buf[0]);
            printf("\n");
      }
      _endthread();
}

int _tmain(int argc, _TCHAR* argv[])
{
      WORD version;
      WSADATA wsaData;
      int result;
      version = MAKEWORD(2,2);
      WSAStartup(version,(LPWSADATA)&wsaData);

      LPHOSTENT hostEntry;
      hostEntry = gethostbyname("127.0.0.1");
      if(!hostEntry)
      {
            printf ("%s", "  >>> ERROR  (hostEntry NULL)\n");
            WSACleanup();
            return CS_ERROR;
      }

      SOCKET theSocket = socket(AF_INET, SOCK_STREAM, 0);
      if(theSocket == SOCKET_ERROR)
      {
            printf ("%s", "  ERROR  (don't create socket)\n");
            return CS_ERROR;
      }
      else
      {
            printf ("%s", "  >>> Create socket \n");
      }

      sockaddr_in serverInfo;
      serverInfo.sin_family = AF_INET;
      serverInfo.sin_addr = *((LPIN_ADDR)*hostEntry->h_addr_list);
      serverInfo.sin_port = htons(8888);

      result=connect(theSocket,(LPSOCKADDR)&serverInfo,
                                          sizeof(serverInfo));
      if(result==SOCKET_ERROR)
      {
```

```
            printf ("%s", "  ERROR (don't connect to Server)\n");
            return CS_ERROR;
      }
      else
      {
            printf ("%s", "  >>> Connect to Server\n");
      }

      printf("\nWrite message: ");
      scanf ("%s", send_buf);
      _beginthread(MyFunction,0,(void *)&theSocket);
      char a[100];
      scanf ("%s", a);
      return CS_OK;
}
```

6.2 Завдання

Написати клієнт-серверний застосунок, котрий буде працювати з потоковими або датаграмними сокетами в асинхронному режимі.

Для студентів котрі хочуть отримати відмінну оцінку: клієнт повинен надіслати якийсь файл, наприклад рисунок. Задача сервера, зберегти його в папці, з назвою яка відповідає даті прийняття файлу;

6.3 Контрольні питання

1. Що таке сокет, які є типи та види сокетів?
2. В чому полягає різниця між Winsock та сокетами, котрі використовуються в UNIX подібних системах?
3. Які функції потрібно використовувати при роботі з потоками?
4. Коли бажано використовувати асинхронні сокети? Чому ви так вважаєте?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Бесекерский В.А. Теория систем автоматического управления / В. Бесекерский, Е. Попов. – СПб.: «Профессия», 2003. – 752 с.

2. Волков Л. Н. Системы цифровой радиосвязи: базовые методы и характеристики: Учебное пособие / Волков Л. Н., Немировский М. С., Шинаков Ю. С. – М.: Эко-Трендз, 2005. – 392 с.

3. Григорьев В. А. Сети и системы радиодоступа / Григорьев В. А., Лагутенко О. И., Распаев Ю. А. – М.: Эко-Трендз, 2005. – 384 с.

4. Скляр Б. Цифровая связь. Теоретические основы и практическое применение / Скляр Б. – [2-е изд]. – М.: Вильямс, 2003. – 1104 с.

5. Советов Б.Я. Построение адаптивных систем передачи информации для автоматизированного управления / Б. Советов, В. Стах. – Л.: Энергоиздат, 1982. – 120с.

6. Cho Y. MIMO-OFDM Wireless Communications with Mathlab / Cho Y., Kim J., Yang W. [et al.]. – Singapore: John Wiley & Sons, 2010. – 457 p.

7. Holma H. HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications / H. Holma, A. Toskala. – John Wiley & Sons. – 2006. 268 p.

8. IEEE 802.16. Broadband Wireless Metropolitan Area Network (Wireless-MAN) [Electronic resource] // Mode of access: http://standards.ieee.org/getieee802/802.16.html. – Title from the screen.

9. Khan F. LTE for 4G Mobile Broadband. Air Interface Technologies and Performance / Khan F. – Cambridge: Cambridge University Press, 2009. – 509 p.