НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2022 р.

Дипломний проект

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Система багатоканального транспорту для віртуальних приватних мереж

Виконав: студент 4 курсу, групи ІО-81
(шифр групи)

Смірнов Назар Олександрович
(прізвище, ім’я, по батькові) (підпис)

Керівник Роковий Олександр Петрович
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) проф., д. т. н. Сімоненко В. П.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали)(підпис)

Рецензент Коган Алла Вікторівна ктн. Доцент каф. ІСТ
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент
(підпис)

Київ – 2022 р.

АННОТАЦІЯ

У даній дипломній роботі була розроблена програма для автоматичного налаштування маршрутів, які будуть в подальшому використовуватись для створення тунелю віртуальної приватної мережі. Дана програма дозволяє користувачу налаштувати маршрути для мережевих інтерфейсів.

Програма розроблена за допомогою спеціальних інструментів, що спеціалізуються на розробці такого виду, а саме мови програмування Dart, її фреймворку Flutter у середовищі розробки VsCode.

ANNOTATION

In this thesis, a program was developed for automatic configuration of routes, which will be used in the future to create a virtual private network tunnel. This program allows the user to configure routes for network interfaces.

The program is developed using special tools that specialize in the development of this type, namely the Dart programming language, its Flutter framework in the VsCode development environment.

Національний технічний університет України
«Київський політехнічний інститут»
ім. Ігоря Сікорського
Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ С.Г. Стіренко
(підпис) (ініціали, прізвище)
“___”_____2022 р.

ЗАВДАННЯ

на бакалаврський дипломний проект студента

Смірнов Назар Олександрович
(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) *Система багатоканального транспорту для віртуальних приватних мереж*,
керівник проекту (роботи) Роковий Олександр Петрович, к.т.н., доцент_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від “___”_____20__ р. №____

2. Термін здачі студентом закінченого проекту (роботи)
_____

3. Вихідні дані до роботи *наукова та технічна документація про реалізацію окремих модулів системи для вибору найоптимальнішого рішення щодо реалізації поставленої задачі з розробки програмного забезпечення.*_____

4. Зміст пояснювальної записки: *опис предметної області і напрямків дослідження, аналіз та характеристика об’єкту досліджень, аналіз існуючих рішень, опис архітектури системи.*_____

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо):

Принципова схема системи, структурна схема системи, діаграма класів.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	д.т.н., проф. Сімоненко В. П.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Затвердження теми роботи	06.04.2022-07.04.2022	
2.	Вивчення та аналіз завдання	07.04.2022-13.05.2022	
3.	Написання вступної частини та огляду предметної області	13.05.2022-23.05.2022	
4.	Розробка архітектури системи	23.05.2022-01.06.2022	
5.	Програмна реалізація системи	01.06.2022-06.06.2022	
6.	Виправлення помилок	06.06.2022-08.06.2022	
7.	Оформлення документації дипломної роботи	08.06.2022-10.06.2022	
8.	Передзахист	11.06.2022	
9.	Захист	20.06.2022	

Студент _____ Смірнов Н. О.
(підпис) (ініціали, прізвище)

Керівник проекту _____ Роковий О.П.
(підпис) (ініціали, прізвище)

№ рядка	Формат	Позначення	Найменування	Кільк.	Примітка
			Документація загальна		
	А4	ІАЛЦ.467100.001 ОА	Система багатоканального транспорту для віртуальних приватних мереж Опис альбому	2	
	А4	ІАЛЦ.467100.002 ТЗ	Система багатоканального транспорту для віртуальних приватних мереж Технічне завдання	3	
	А4	ІАЛЦ.467100.003 ПЗ	Система багатоканального транспорту для віртуальних приватних мереж Пояснювальна записка	59	
	А4	ІАЛЦ.467100.004 Д1	Система багатоканального транспорту для віртуальних приватних мереж Принципова схема	1	
	А4	ІАЛЦ.467100.005 Д2	Система багатоканального транспорту для віртуальних приватних мереж Структурна схема	1	
	А4	ІАЛЦ.467100.006 Д3	Система багатоканального транспорту для віртуальних приватних мереж Діаграма класів	1	

					ІАЛЦ.467100.001 ОА		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Смірнов Н. О.			Система багатоканального транспорту для віртуальних приватних мереж Опис альбому	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Роковий О. П.					
Реценз.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81	
Н. контр.		Сімоненко В. П.					
Затвердив							

№ рядка	Формат	Позначення	Найменування	Кільк.	Примітка
	А4	ІАЛЦ.467100.007 Д4	Система багатоканального транспорту для віртуальних приватних мереж Програмний код	23	

					ІАЛЦ.467100.001 ОА	Арк
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

ТЕХНІЧНЕ ЗАВДАННЯ
До дипломного проекту
на тему: «Система багатоканального транспорту для віртуальних приватних мереж»

Київ – 2022

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ..... 2
2. ПІДСТАВИ ДЛЯ РОЗРОБКИ..... 2
3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ..... 2
4. ДЖЕРЕЛА РОЗРОБКИ..... 2
5. ТЕХНІЧНІ ВИМОГИ..... 3
   5.1. Вимоги до програмного продукту, що розробляється.....3
   5.2. Вимоги до інструментального програмного забезпечення 3
6. ЕТАПИ РОЗРОБКИ..... 3

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи багатоканального транспорту для віртуальних приватних мереж.

Метою розробки є покращення та спрощення роботи з існуючими системами.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки постає створення програми для покращення та спрощення роботи і налаштування віртуальних мереж з багатоканальним транспортом, яка буде доступна і зрозуміла для будь-яких користувачів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література, технічна документація технологій, що використовувались для розробки системи, публікації в мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

Система, що розроблюється, повинна відповідати поставленим вимогам, що наведені нижче:

- Простий і зрозумілий інтерфейс системи;
- Технічна коректність;
- Можливість створювати, редагувати, та видаляти запит з потрібною інформацією;
- Швидкий доступ до системи;

5.2. Вимоги до інструментального програмного забезпечення

Для розробки проекту на локальному ПК необхідне наступне програмне забезпечення:

- Операційна система Linux;
- Мова програмування Dart версії 2.17.1;
- Фреймворк Flutter версії 3.0.1 чи вище;
- Середовище розробки VsCode;

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	06.04.2022-07.04.2022
Вивчення та аналіз завдання	07.04.2022-13.05.2022
Написання вступної частини та огляду предметної області	13.05.2022-23.05.2022
Розробка архітектури системи	23.05.2022-01.06.2022
Програмна реалізація системи	01.06.2022-06.06.2022
Виправлення помилок	06.06.2022-08.06.2022
Оформлення документації дипломної роботи	08.06.2022-10.06.2022
Передзахист	11.06.2022
Захист	20.06.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проекту
на тему: «Система багатоканального транспорту для віртуальних приватних мереж»

Київ – 2022

ЗМІСТ

Список скорочень та умовних позначень.....3

ВСТУП 4

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ СИСТЕМ БАГАТОКАНАЛЬНОГО ТРАНСПОРТУ ДЛЯ ВІРТУАЛЬНИХ ПРИВАТНИХ МЕРЕЖ.....5

1.1.Віртуальна приватна мережа 5

1.1.1. Опис VPN.....5

1.1.2. Конфігурації VPN 5

1.2.Огляд специфікацій MPTCP 6

1.2.1. Використання TCP поверх TCP.....6

1.2.2. Опис MPTCP 7

1.2.3. Порівняння MPTCPv0 та MPTCPv1.....10

1.3.Огляд існуючих VPN 11

1.3.1. OpenVpn 11

1.3.2. Wireguard 12

1.3.3. Vtrunkd 13

Висновок до розділу 1 15

РОЗДІЛ 2. ОГЛЯД ТА АНАЛІЗ РОБОТИ ІСНУЮЧИХ СИСТЕМ 16

2.1.Опис тестового стенду.....16

2.2.Тестові операції.....22

2.3.Результати тестування 25

2.4.Тестування відмовостійкості 37

Висновок до розділу 2 39

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ ДЛЯ АВТОМАТИЧНОГО НАЛАШТУВАННЯ ТРАНСПОРТНОГО ТА ПРИКЛАДНОГО РІВНЯ ВІРТУАЛЬНОЇ ПРИВАТНОЇ МЕРЕЖІ НА БАЗІ MPTCP ТА VTRUNKD 40

3.1.Причини розробки. 40

3.2.Планування структури проекту та реалізації основного функціоналу 40

3.3.Результати розробки 49

Висновок до розділу 3 51

ВИСНОВКИ 52

Використані джерела: 53

Список скорочень та умовних позначень

MPTCP – протокол багатоканального транспортного з’єднання

VPN – віртуальна приватна мережа.

Хост (host) – будь-який комп'ютерний пристрій, що має доступ до IP мережі тобто синонім терміна вузол мережі.

Підпотік (subflow) – термін, який позначає певний канал транспортування.

ПЗ – програмне забезпечення.

RTT (Round Trip Time) – час, потрібний для пересилання сигналу від передавача до отримувача, а потім у зворотньому напрямку.

TCP – протокол транспортного рівня, який забезпечує передачу даних між процесами на хостах.

ВСТУП

Використання віртуальних приватних мереж (VPN) стало поширеною практикою в багатьох компаніях. Ця потреба в VPN зумовлена не тільки в віддаленому доступі до офісної мережі та інфраструктури без використання фізичних з'єднань, але й також в підвищенні безпеки та стабільності доступу до приватної та комерційної інформації. VPN досить легко встановити поверх існуючої фізичної мережі, а також її можна гнучко налаштувати для забезпечення конфіденційності.

Підвищення стабільності та бажання обійти фізичне з'єднання призводить до додавання додаткової логіки на транспортному рівні. В VPN даний рівень представлений тунелем через який передається інформація, частіше за все поверх UDP або TCP протоколів.

На сьогоднішній день існує багато рішень для підвищення стабільності з'єднання. Одним з таких рішень є "Multihoming". Використання декількох каналів в один і той самий час та балансування трафіку між ним використовуючи конфігурації маршрутизації. Нажаль даний підхід не завжди покращує швидкість з'єднання але дуже добре справляється з покращення відмовостійкості.

Multipath TCP (MPTCP) це розширення протоколу TCP, ціллю якого є підтримка використання декількох транспортних каналів між парою хостів. Переваги MPTCP включають підвищення продуктивності щодо пропускної здатності та здатність підтримки декількох з'єднань.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ СИСТЕМ БАГАТОКАНАЛЬНОГО ТРАНСПОРТУ ДЛЯ ВІРТУАЛЬНИХ ПРИВАТНИХ МЕРЕЖ

1.1. Віртуальна приватна мережа

Далі буде детально описано роботу віртуальної приватної мережі.

1.1.1. Опис Vpn

VPN — це загальний термін, який охоплює використання загальнодоступних або приватних мережі для створення груп користувачів, відокремлених від інших користувачів мережі, які можуть спілкуватися між собою так, ніби вони знаходяться в приватній мережі. Це визначення VPN з [1].

1.1.2. Конфігурації VPN

VPN створюються з різних причин. Це може знадобитися, щоб клієнт міг підключитися його корпоративні офіси, які використовують готельний Wi-Fi під час подорожі. Можливо, а малий бізнес розширюється в нове географічне розташування та потребує підтримки безпеки комунікації між двома локаціями. Це лише два з багатьох сценаріїв може вимагати налаштування VPN. Існує три поширені конфігурації VPN:

- шлюз до шлюзу
- хост до шлюзу
- хост до хоста

Модель шлюз-шлюз з'єднує один мережевий шлюз з іншим мережевим шлюзом без необхідності фізичного підключення. Цю конфігурацію можна використовувати для підключення віддалені офіси корпорації до місцевих офісів і підтримувати безпечний зв'язок ніби віддалені офіси були частиною локальної мережі.

Модель хост-шлюз дуже схожий. Замість використання маршрутизатора мережевого шлюзу хост-машина стає шлюзом, щоб встановити з’єднання зі шлюзом потрібного VPN-сервера.

Остання і найменше використовувана модель використовує кожну хост-машину як мережевий шлюз і встановлює безпечне з'єднання між хостами. Цей спосіб не дуже поширений через простіші методи, такі як SSH, для досягнення тієї ж мети.

1.2. Огляд специфікації MPTCP

1.2.1. Використання TCP поверх TCP

Олаф Тітц [2] описує сценарій, коли зовнішній рівень TCP має коротший RTO, ніж внутрішній рівень, що призводить до непотрібних тайм-аутів і ретрансляції. Щоб протидіяти передбачуваним перевантаженням, викликаним цими тайм-аутами і повторних передач, будуть передані менші пакети, що, у свою чергу, призводить до падіння продуктивності. У "Understanding TCP over TCP: Effects of TCP Tunneling on End-to-End Throughput and Latency", автори обговорюють параметри TCP, які можуть бути вказані для усунення проблем TCP поверх TCP [3]. Одним з параметрів TCP, який став стандартним у Linux, є опція вибіркового підтвердження (SACK). Ця опція розроблена, щоб дозволити клієнту надсилати повідомлення ACK для певних пакетів, що дозволяє повторно передавати лише ті пакети які були втрачені. Було показано, що завдяки використанню параметра TCP SACK в тунельному потоці TCP, випробувана продуктивність зросла [3]. Ще один налаштований параметр, який може вплинути на TCP підключення – це розмір буфера використовуваного сокета. Встановивши невеликий розмір буфера, вхідні пакети TCP потрібно буде швидко обробляти, інакше буфер буде переповнений і пакети будуть скинуті. Надмірно великий буфер може призвести до затримки пакетів в буфері довше, ніж значення RTO,

установлене на посиланні. Це призведе до непотрібності повторні передачі через уявну втрату, якої не було. Також було визначено що налаштувати розміри буфера сокета як для наскрізних тунелів, так і тунелю TCP для збільшення пропускної здатності лінії зв'язку також дав сприятливі результати [3]. Незважаючи на ці удосконалення, ефект TCP над TCP все ще присутній і є перешкодою для використання TCP з VPN.

1.2.2. Опис MPTCP

MPTCP працює на транспортному рівні і прагне бути прозорим як для вищих, так і для нижчих шарів. Це набір додаткових функцій поверх стандартного TCP; Рисунок 1.1 ілюструє це розшарування. MPTCP розроблено так, що може використовуватись застарілими програмами без змін.

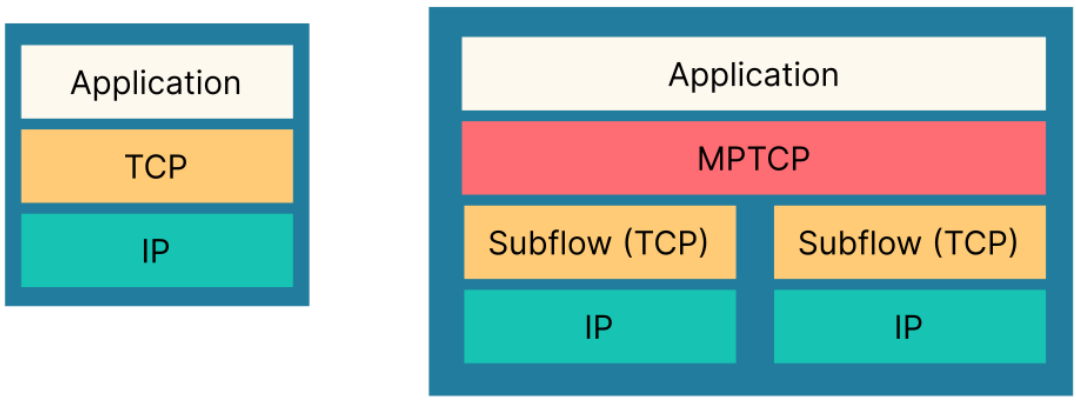


Рис 1.1 – MPTCP в стеку протоколів TCP/IP

Завдяки цьому не змінюється інтерфейс API сокету для програм, який дозволяє всім програмам користуватися ним без використання необхідність модифікації. Це усуває жорсткий зв’язок між IP і TCP, щоб пристрої могли використовувати кілька IP-адрес для одного MPTCP-з'єднання.

Налаштування з’єднання MPTCP виглядає наступним чином:

1. Спочатку ініціатор з'єднання оголошує, що він сумісний з MPTCP (MP_CAPABLE).
2. Потім, якщо приймач сумісний з MPTCP, він встановлює нове з’єднання MPTCP.
3. Після цього він добавляє новий підпотік для кожного відомого шляху до існуючого з’єднання MPTCP (MP_JOIN)
4. Після цього можна здійснювати передачу інформації через MPTCP з’єднання

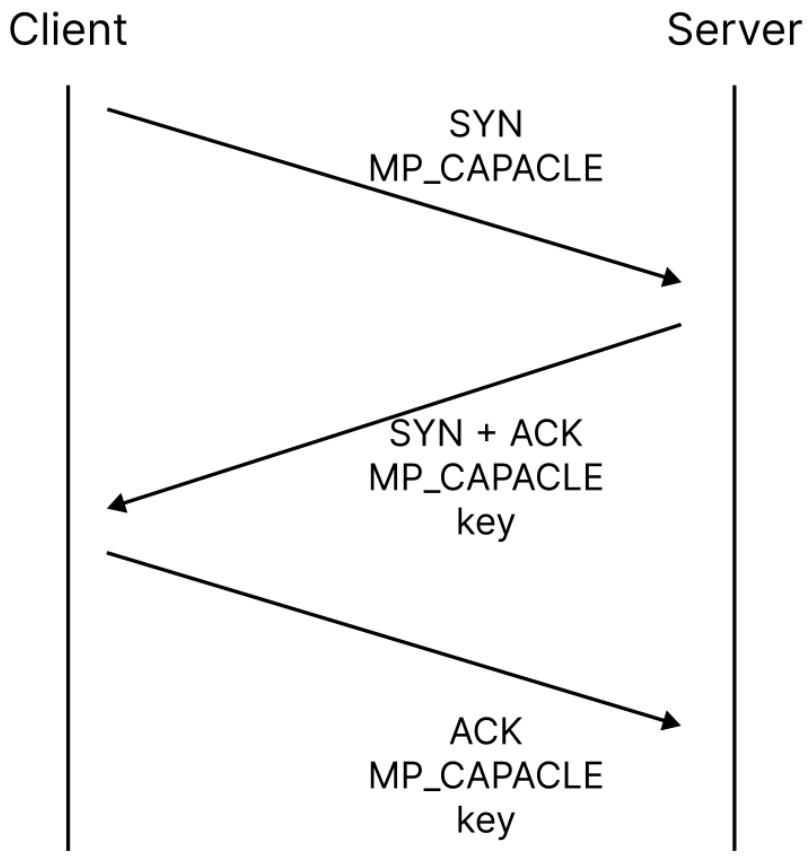


Рис 1.2 –

Для встановлення з’єднання MPTCP він використовує тристороннє рукостискання, як показано на рисунку 1.2. Клієнт надсилає пакет з параметром MP_CAPABLE, щоб повідомити серверу, що він є сумісні. Сервер надсилає назад ACK, що містить параметр MP_CAPABLE і

випадковий ключ, якщо він сумісний. Після цього клієнт надсилає ACK на сервер, що містить ключ, це налаштує перший підпотік.

Якщо клієнт має більше одного фізичного посилання, підключеного до сервера, MPTCP буде спробуйте додати інший підпотік до існуючого з'єднання. Для цього буде використовуватися тристоронній рукостискання та параметр MP_JOIN з унікальним маркером token (рис. 1.3)

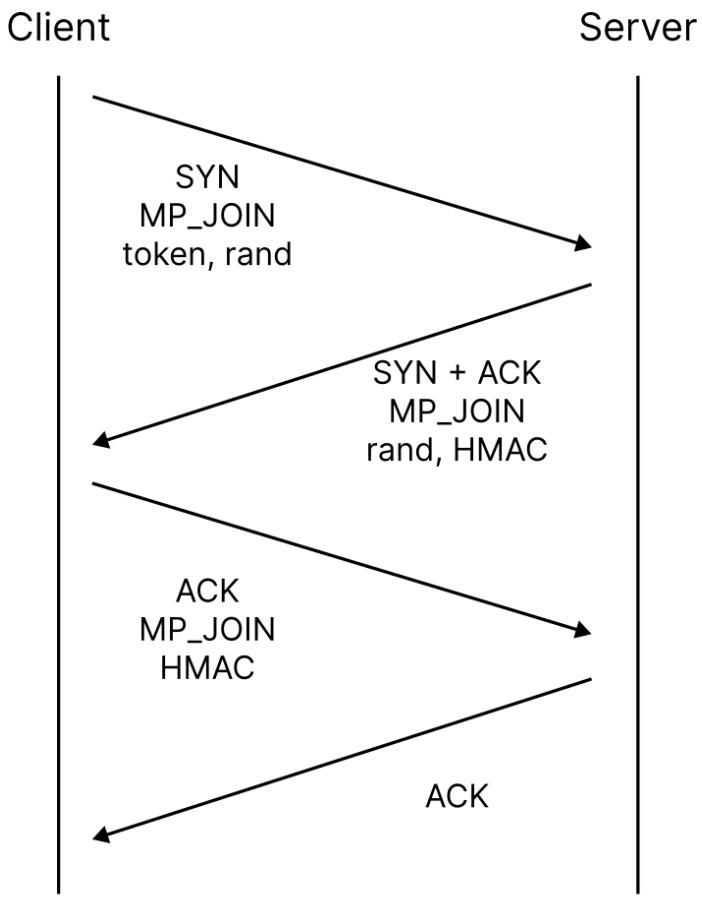


Рис 1.3 –

Спочатку SYN з опцією MP_JOIN, також буде надіслано унікальний маркер (token) і випадковий ключ (rand).

Потім клієнт і сервер обчислять аутентифікаційний хеш код повідомлення (HMAC), обчислений за випадковим ключем, яким обмінялися.

Маркер призначений для того, щоб мати можливість однозначно ідентифікувати підпотік, оскільки в звичайному TCP з'єднання ідентифікується за допомогою ip адрес та портів відправника та

отримувача, що в даному випадку недостатньо для однозначної ідентифікації під потоку, якщо хост стоїть за NAT.

Коли створено декілька під потоків, дані можуть передаватися по будь-якому з них. Це можна використовувати для збільшення пропускної здатності з’єднання або для повторної передачі даних, якщо один підпотік не працює. Кожен підпотік еквівалентний одному TCP-з'єднанню.

Використання кількох під потоків також вимагає впорядкування даних у кожному під потоку, а потім впорядкування даних між цими під потоками, оскільки деякі можуть бути швидшими за інші. При отриманні даних спочатку змінюється їх порядок на рівні під потоку, використовуючи число послідовності під потоку. Потім пере упорядковуються дані між різними під потоками на рівень підключення за допомогою порядкового номера даних.

Д. Каспар наводить у своїй дисертації вираз що перевага агрегації багатоканального протоколу в пропускній здатності окремих шляхів. Цей вираз показує переваги агрегації, дозволяючи порівнювати різні середовища та враховує продуктивність багатошляхового TCP порівняно зі звичайним TCP через найкращий шлях [7].

1.2.3. Порівняння MPTCPv0 та MPTCPv1

На даний момент також існують різні версії протоколу:

- RFC 6824 (MPTCPv0) [9]
- RFC 8684 (MPTCPv1) [10]

MPTCPv1 має значні зміни, які роблять його несумісним з v0. Дані версії відрізняються структурно. Ось чому curl http://multipath-tcp.org/ завжди буде говорити, що ви не підтримуєте MPTCP(v0) [8].

1.3. Огляд існуючих VPN

1.3.1. OpenVpn

OpenVPN - це програмна система з відкритим кодом яка реалізує VPN для створення захищених каналів. Розроблена James Yonan.

В openVPN використовується SSL/TLS та власний протокол безпеки та використовується OpenSSL бібліотека для шифрування інформації та аутентифікації. Є декілька методів аутентифікації таких як: статичний ключ, сертифікат або пара логін та пароль. Використання шифрування підвищує рівень безпеки, але потребує додаткових ресурсів, бо пакети повинні бути дешифровані.

OpenVPN використовує два канали, інформаційний канал, який обробляє IP дейтаграми користувача, і канал контролю, який обробляє ключі та конфігурації.

OpenVPN може використовувати UDP або TCP щоб транспортувати пакети. Зазвичай віддають перевагу UDP так як він в загальному швидший.

OpenVPN доступний на багатьох платформах: Linux, Windows, Mac OS X а також на різних спеціалізованих операційних системах для роутерів таких як DD-WRT, OpenWRT. Це дає можливість налаштувати роутер як OpenVPN клієнта і користувачі підключені до даного роутера будуть мати доступ до VPN без потреби налаштовувати його для кожного хоста окремо.

OpenVPN виконується в userspace, що забезпечує хороший рівень захищеності і портативності, проте пакети потрібно переміщати між kernel space та userspace, що потребує додаткових циклів виконання на процесорі і може мати вплив на продуктивність системи.

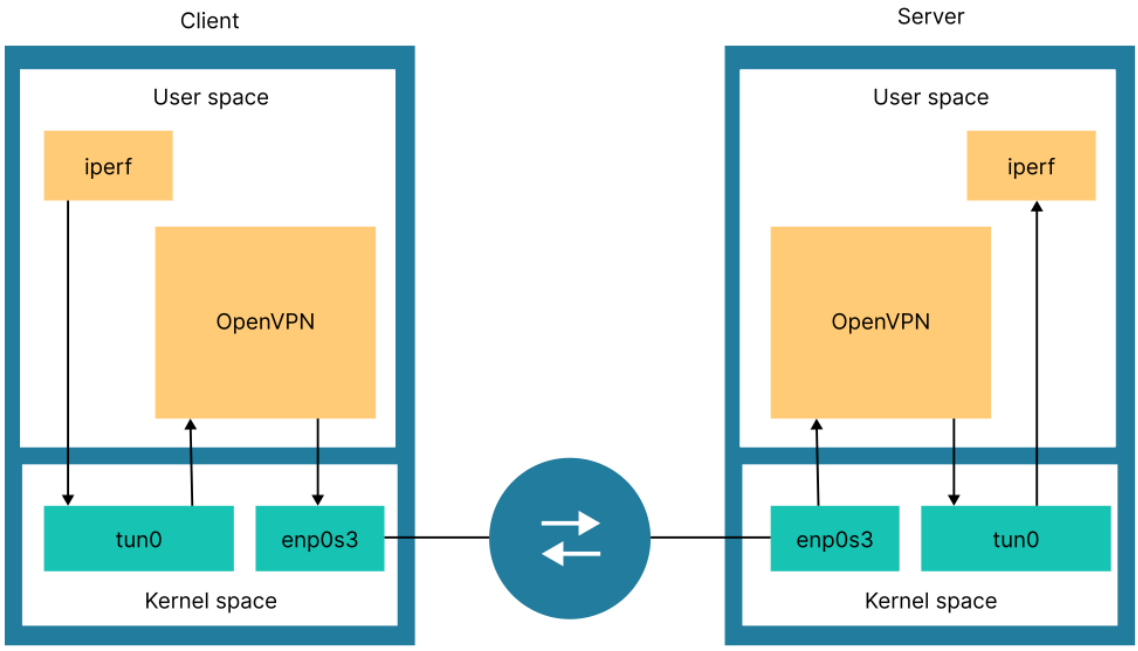


Рис 1.4 –

На рисунку 1.4 зображено шлях по якому пакети передаються між програмами на різних хостах.

Для створення тунелю OpenVPN використовує драйвер тунелю, який надається як модуль для linux. Цей драйвер дозволяє створювати віртуальні інтерфейси, які називаються tun або tap пристроями. Різниця між tun і tap полягає в рівні, на якому вони знаходяться, tap - це канальний рівень, а tun — це мережевий рівень.

На практиці це означає, що tap може надсилати кадри Ethernet і додає більше накладних витрат а tun може надсилати лише пакети, але додає менше накладних витрат [5].

1.3.2. Wireguard

WireGuard — це надзвичайно проста, але швидка й сучасна VPN. WireGuard має на меті бути швидшим, простішим, компактним та кориснішим, ніж IPsec, уникаючи при цьому величезного головного болю. WireGuard має намір бути значно продуктивнішим, ніж OpenVPN.

WireGuard розроблено як VPN загального призначення для роботи як на вбудованих пристроях, так і на суперкомп’ютерах, що підходить для

багатьох різних обставин. Спочатку випущений для ядра Linux, тепер існують реалізації для Windows, macOS, BSD, iOS, Android.

WireGuard використовує найсучаснішу криптографію, як-от протоколи Noise, Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF [4].

WireGuard надійно інкапсулює IP-пакети через UDP. Ви додаєте інтерфейс WireGuard, налаштовуєте його за допомогою свого приватного ключа та відкритих ключів інших користувачів, а потім надсилаєте через нього пакети. WireGuard імітує модель SSH і Mosh; обидві сторони мають відкриті ключі один одного, а потім вони можуть просто розпочати обмін пакетами через інтерфейс.

В основі WireGuard лежить концепція під назвою Cryptokey Routing, яка працює шляхом пов’язування відкритих ключів зі списком тунельних IP-адрес, які дозволені всередині тунелю. Кожен мережевий інтерфейс має приватний ключ і список клієнтів. Кожен клієнтів має відкритий ключ. Відкриті ключі короткі та прості, і використовуються клієнтами для аутентифікації один одного [4].

1.3.3. Vtrunkd

Vtrunkd — це Linux VPN, який використовується для об’єднання кількох шляхів підключення в один агрегований канал. Реалізує затримки, зміни порядку, оптимізації аналізу поведінки для інкапсульованих протоколів, керування буфером, резервування пакетів та використання кількох ядер процесора. Підтримується до 30 гетерогенних з'єднань. Використовується для прямої трансляції, підключення LTE/3G/Wi-Fi. Підтримується 32/64-розрядна версія, x86, MIPS і ARM. Підтримує плагіни Python для впровадження нових алгоритмів [6].

Висновок до розділу 1

У першому розділі були розглянуті існуючі реалізації VPN та основні підстави для функціонування MPVPN. MPTCP – це транспортний протокол, що дозволяє об’єднувати пропускну здатність за допомогою кількох TCP-з'єднання. Відомо, що VPN зазнають втрати продуктивності при використанні тунелів TCP і передача даних також за допомогою TCP. MPTCP здатний подолати частину втрат продуктивності, які спостерігаються при використанні TCP над TCP. MPTCP також надає перевагу мобільності VPN, які можуть бути важливими в моделях VPN "хост-хост" або "хост-шлюз".

РОЗДІЛ 2. ОГЛЯД ТА АНАЛІЗ РОБОТИ ІСНУЮЧИХ СИСТЕМ

2.1. Опис тестового стенду

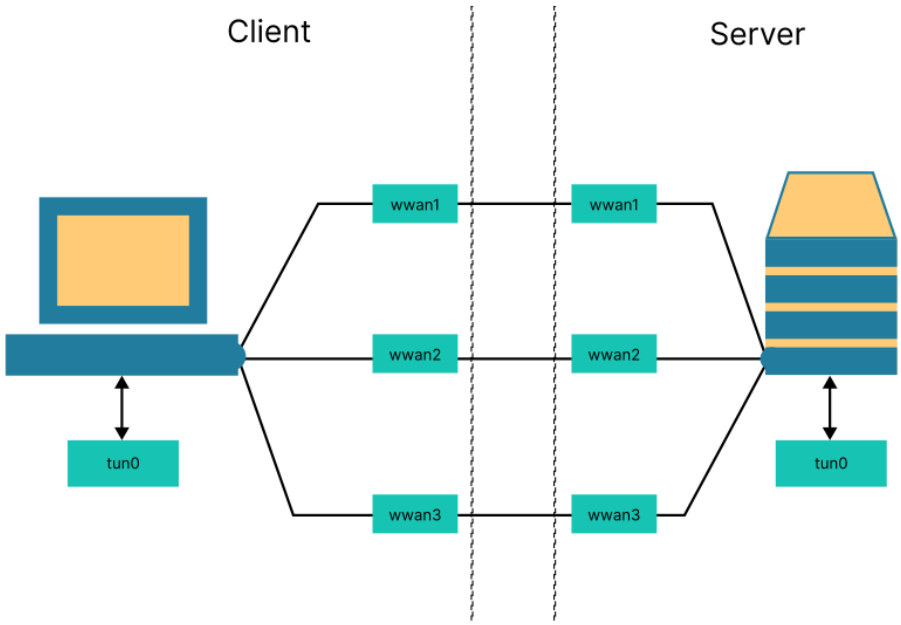


Рис 2.1 –

Для тестування роботи існуючих систем потрібно визначитись з компонентами системи та функціоналом за який вони відповідають. На щастя для створення мінімального середовища для тестування не потрібно багато хостів, а обов'язкових є лише два.

Першим є VPN сервер, який відповідає за доступ до приватної мережі. Наданому хості має бути запущена одна з вищеперелічених реалізацій VPN сервера, а також має бути підтримка протоколу MPTCP.

Другим є VPN клієнт який і буде підключатися до приватної мережі та передавати дані через декілька каналів зв'язку. На даному хості має також бути підтримка MPTCP та встановлений VPN клієнт для підключення до сервера.

Також між даними хостами має існувати декілька каналів зв'язку як показано на рисунку 2.2, через які буде передаватись інформація, що і несе найбільшу цікавість та інформативність в даній роботі.

Основними характеристиками для аналізу на даному стенді будуть: час затримок, втрата пакетів, середнє квадратичне відхилення та полоса.

Також для кращого аналізу потрібно розглянути зміну даних характеристик при різній кількості мережевих каналів і базовим значенням, яким є значення характеристик при одному каналі зв'язку.

Тестовий стенд був розроблений з використанням віртуальних машин у середовищі Virtualbox. Параметри каналів були встановлені за допомогою Virtualbox, контролера трафіку Linux (tc) і мережевий емулятор Linux (netem).

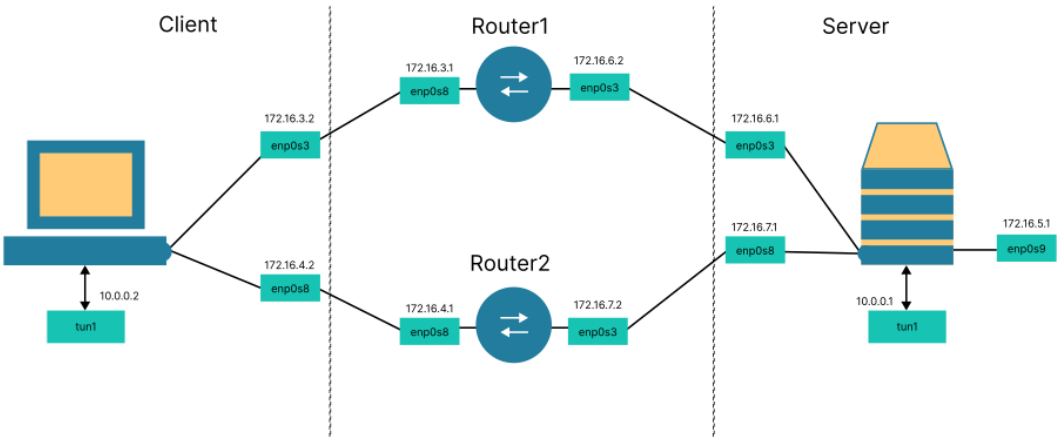


Рис 2.2 –

Перед початком роботи на хостах потрібно підготувати та встановити програми які будуть використовуватись в подальшому, для налаштування, тестування та програми для створення приватної мережі.

Список програм для клієнта:

- mptcp (https://github.com/multipath-tcp/mptcp/releases)
- net-tools
- vtrunkd (https://github.com/VrayoSystems/vtrunkd)
- openvpn
- iperf3
- netperf

Список програм для серверу:

- mptcp
- net-tools
- vtrunkd
- openvpn
- iperf3
- netperf
- ssh

Список програм для роутерів:

- openssh-server
- iptables

Схема мережевих інтерфейсів з встановленими ip адресами зображена на рисунку 2.2. Для налаштування потрібно відредагувати файли “/etc/network/interfaces” для встановлення статичних адрес.

Налаштування клієнта:

```
auto lo
iface lo inet loopback
auto enp0s3
iface enp0s3 inet static
        address 172.16.3.2
        netmask 255.255.255.0
auto enp0s8
iface enp0s8 inet static
        address 172.16.4.2
        netmask 255.255.255.0
```

Налаштування серверу:

```
auto lo
iface lo inet loopback
auto enp0s3
iface enp0s3 inet static
        address 172.16.6.1
        netmask 255.255.255.0
auto enp0s8
iface enp0s8 inet static
        address 172.16.7.1
        netmask 255.255.255.0
```

Налаштування першого роутера:

```
auto lo
iface lo inet loopback
auto enp0s3
iface enp0s3 inet static
```

```
        address 172.16.6.2
        netmask 255.255.255.0
    auto enp0s8
    iface enp0s8 inet static
        address 172.16.3.1
        netmask 255.255.255.0
```

Налаштування другого роутера:

```
    auto lo
    iface lo inet loopback
    auto enp0s3
    iface enp0s3 inet static
        address 172.16.7.2
        netmask 255.255.255.0
    auto enp0s8
    iface enp0s8 inet static
        address 172.16.4.1
        netmask 255.255.255.0
```

Після налаштування адрес на мережевих інтерфейсах потрібно налаштувати маршрутизацію та vtrunkd на обох хостах то nat на роутерах. Для налаштування роутерів використаємо утиліту iptables а для клієнта та сервера нам знадобиться route для встановлення gateway для кожного маршруту.

Конфігурація vtrunkd для серверу:

```
options {
  port 5000;          # Listen on this port.
  timeout 5;

  max_tunnels_num 10;
  # Syslog facility
  syslog    daemon;

  # Path to various programs
  ppp             /usr/sbin/pppd;
  ifconfig  /sbin/ifconfig;
  route     /sbin/route;
  firewall  /sbin/iptables;
```

```
  ip          /sbin/ip;
}

# Default session options
default {
  compress no;        # Compression is off by default
  speed 0;        # By default maximum speed, NO shaping
  proto tcp;              # UDP|TCP protocol
  encrypt  no;          # Encryption
  keepalive yes;       # Keep connection alive
  multi killold;  # has no effect now
  stat yes;
       tick_secs 3;
       max_latency 2000;
       max_latency_drop 50;
       max_idle_timeout 20;
       ping_interval 2;
       tun_txqueue_len 4000;
       tcp_conn_amount 1;
  type tun;
}

000000_1 {
  passwd  testpasswd;
  device tun10;
  up {
    ifconfig "%% 10.0.0.1 pointopoint 10.0.0.2  mtu 1350 up";
  };
  down {
       ifconfig "%% down";
  };
}
000000_2 {
  passwd  testpasswd;
  device tun10;
  up {
    ifconfig "%% 10.0.0.1 pointopoint 10.0.0.2  mtu 1350 up";
  };
  down {
       ifconfig "%% down";
  };
}
```

[6]

Конфігурація openvpn для серверу:

```
local 172.16.5.1
server 10.0.0.0 255.255.255.0
port 1194
proto tcp
verb 3
dev tun

topology subnet

ca ca.crt
cert server.crt
key server.key
crl-verify crl.pem
dh dh.pem
tls-crypt tc.key
ifconfig-pool-persist ipp.txt


push "redirect-gateway def1 bypass-dhcp"
keepalive 10 120
cipher none

user nobody
group nogroup

persist-key yes
persist-tun yes
```

Налаштування першого роутера:

```
iptables -t nat -A POSTROUTING -j MASQUERADE
route add default gw 172.16.6.1
```

Налаштування другого роутера:

```
iptables -t nat -A POSTROUTING -j MASQUERADE
route add default gw 172.16.7.1
```

Налаштування клієнта:

```
route add default gw 172.16.3.1
route add default gw 172.16.4.1
```

Налаштування маршрутизацій для клієнта:

```
ip rule add fwmark 0x1 lookup 101
ip rule add fwmark 0x2 lookup 102
```

```
ip route add default via 172.16.3.1 dev enp0s3 table 101 metric 200
ip route add default via 172.16.4.1 dev enp0s3 table 102 metric 200
```

Налаштування (запуск) vtrunkd клієнта:

```
vtrunkd -P 6000 -f /etc/vtrunkd.conf 000000_1 172.16.5.1
vtrunkd -P 6000 -f /etc/vtrunkd.conf 000000_2 172.16.5.1
```

Спочатку за допомогою перших двух команд створюються нові таблиці маршрутизації які позначаються 0x1 та 0x2. Далі налаштовуються відповідні мережеві інтерфейси кожен на свою таблицю. І в кінці створюються з'єднання з vtrunkd сервером, що і створює тунель tun1.

Спочатку за допомогою перших двух команд створюються нові таблиці маршрутизації які позначаються 0x1 та 0x2. Далі налаштовуються відповідні мережеві інтерфейси кожен на свою таблицю. І в кінці створюються з'єднання з vtrunkd сервером, що і створює тунель tun1.

2.2. Тестові операцій

Щоб виміряти відмінності в продуктивності під час використання певного VPN, ми виміряємо пропускну здатність та затримки, з використанням різних конфігурацій, таких як наведені нижче.

Тести проводитимуться з використанням різної швидкості з'єднання, щоб імітувати реальну швидкість інтернет-з'єднання. Для кращої наглядності зміни характеристик було вирішено розглянути діапазон швидкостей від 10 Мбіт\сек до 100 Мбіт\сек з шагом 10 Мбіт\сек.

Були вибрані також затримки, для то що показати як буде поводити себе певна реалізація на при різних робочих ситуаціях. А саме були вибрані слідуючі затримки в кожну сторону:

- 1ms
- 25ms
- 50ms
- 75ms

- 100ms
- 125ms
- 150ms
- 175ms
- 200ms

Для кращого розуміння, як буде працювати vpn в реальних умовах, не можна обійтися без тестування з втратою пакетів. Це покаже як буде змінювати пропускна здатність від втрати пакетів. Тому також до тестових операцій додається тестування при втраті пакетів від 1% до 10% з кроком 1%.

Для налаштування даних параметрів була використана утиліта tc, яка дозволяє гнучко налаштовувати трафік. Для швидкого налаштування був написаний невеликий скрипт, за допомогою якого можна простіше встановити налаштування вказавши лише мережевий інтерфейс пропускну здатність та час затримки для даного інтерфейсу.

Для автоматичного збору статистики було вирішено розробити скрипти для автоматичного тестування та розмістити їх за шляхом “/scripts/”.

Auto_test.sh

```
#!/bin/bash

# Run this script on client host

declare -a delays = (1, 25, 50, 75, 100, 125, 150, 175, 200)
declare -a bandwidths = (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)

# set your interfaces
declare -a client_interfaces = ("enp0s3", "enp0s8")
declare -a server_interfaces = ("enp0s3", "enp0s8")

ssh root@172.16.4.1 iperf3 -s &
ssh root@172.16.4.1 netserver &
sleep 5 # wait for the start iperf server and netserver


for delay in ${delays[@]}; do
   for bandwidth in ${bandwidths[@]}; do
      echo "delay: $delay, bandwidth: $bandwidth"
```

```
        #clear qdics
        for interface in ${client_interfaces[@]}; do
            tc qdisc del dev $interface root
        done
        for interface in ${server_interfaces[@]}; do
            ssh root@172.16.4.1 tc qdisc del dev $interface root
        done

        # setup for all interfaces on client
        for interface in ${client_interfaces[@]}; do
            exec "./client_qdisc_setup.sh $interface $bandwidth $delay"
        done
        # setup for all interfaces on server
        for interface in ${server_interfaces[@]}; do
            exec "./server_qdisc_setup.sh $interface $bandwidth $delay"
        done

        sleep 2

        iperf3 -c 10.0.0.1 -t 60 > "./data/$bandwidth:$delay_tcp.log"
        netperf -t tcp_rr -H 10.0.0.1 -l 40 -- -r 1,1 -o stddev_letency | sed -
n 3p > "./data/$bandwidth:$delay_latency.log"
    done
done
```

Client_qdisc_setup.sh

```
#!/bin/bash

# arguments description
# $1: network interface name (look ifconfig)
# $2: bandwidth
# $3: delay


tc qdisc add dev $1 root handle 1:0 tbf rate $2mbit burst 16000 latency
5ms
tc qdisc add dev $1 parent 1:1 handle 10: netem delay $3ms
```

Server_qdisc_setup.sh

```
#!/bin/bash

# arguments description
# $1: network interface name (look ifconfig)
# $2: bandwidth
# $3: delay
```

```
ssh root@172.16.4.1 tc qdisc add dev $1 root handle 1:0 tbf rate $2mbit
burst 16000 latency 5ms
ssh root@172.16.4.1 tc qdisc add dev $1 parent 1:1 handle 10: netem
delay $3ms
```

2.3. Результати тестування

У цьому розділі представлені результати тестування.

Спочатку я представляю загальну таблицю з результатами, які були отриманні при тестуванні Vtrunkd та OpenVpn, тестуючи пропускну здатність.

Тестування поведінки відбувається з використанням каналів з різними RTT і пропускною здатністю, дані представленні в таблиці 2.1. Також були перевірені поведінки при різному відсотку втрати пакетів, та занесені в таблицю 2.2.

Таблиця 2.1

Кількість каналів	**Час затримки (ms)**	**Пропускна здатність каналу (Mbits/sec)**	**Vtrunkd Пропускна здатність (Mbits/sec)**	**Vtrunkd Середнє квадратичне відхилення (ms)**	**OpenVpn Пропускна здатність (Mbits/sec)**	**OpenVpn Середнє квадратичне відхилення (ms)**
2	1	10	17.6	2.03	12.8	2.81
2	25	10	17.8	2.55	16.5	9.81
2	50	10	13.7	6.69	17.5	21.45
2	75	10	17.3	9.65	15.5	38.89
2	100	10	15.1	14.87	15.8	74.49
2	125	10	15.4	20.25	14.4	103.57
2	150	10	16.3	30.05	16.7	137.11
2	175	10	14.8	33.62	11.0	172.37
2	200	10	14.9	40.66	12.6	211.46

Продовження таблиці 2.1

Кількість каналів	**Час затримки (ms)**	**Пропускна здатність каналу (Mbits/sec)**	**Vtrunkd Пропускна здатність (Mbits/sec)**	**Vtrunkd Середнє квадратичне відхилення (ms)**	**OpenVpn Пропускна здатність (Mbits/sec)**	**OpenVpn Середнє квадратичне відхилення (ms)**
2	1	20	35.6	2.01	35.2	1.86
2	25	20	35.4	3.64	29.5	9.71
2	50	20	32.2	5.33	32.5	26.64
2	75	20	31.3	9.34	31.2	48.40
2	100	20	30.6	14.32	22.1	81.78
2	125	20	29.0	20.32	28.4	83.57
2	150	20	28.1	26.63	21.9	142.66
2	175	20	26.3	33.39	25.9	172.94
2	200	20	23.4	40.89	23.4	211.50
2	1	30	55.3	1.38	54.5	3.08
2	25	30	53.5	2.25	51.6	10.57
2	50	30	50.5	5.39	48.4	26.54
2	75	30	46.7	9.68	45.5	56.97
2	100	30	41.9	14.73	42.8	77.72
2	125	30	38.8	21.00	33.7	84.06
2	150	30	32.7	26.83	28.2	137.03
2	175	30	30.8	33.39	30.9	138.88
2	200	30	25.9	40.62	27.8	176.64
2	1	40	72.8	2.32	73.0	0.89
2	25	40	68.6	2.57	69.1	9.65
2	50	40	62.4	5.42	62.4	33.43

Продовження таблиці 2.1

Кількість каналів	**Час затримки (ms)**	**Пропускна здатність каналу (Mbits/sec)**	**Vtrunkd Пропускна здатність (Mbits/sec)**	**Vtrunkd Середнє квадратичне відхилення (ms)**	**OpenVpn Пропускна здатність (Mbits/sec)**	**OpenVpn Середнє квадратичне відхилення (ms)**
2	75	40	55.5	9.83	56.9	48.02
2	100	40	50.8	14.97	45.8	59.96
2	125	40	42.8	20.23	42.2	87.09
2	150	40	34.4	26.63	35.2	136.45
2	175	40	31.1	33.47	31.6	172.23
2	200	40	29.0	40.67	28.7	210.76
2	1	50	92.3	1.79	92.3	0.69
2	25	50	87.8	2.04	85.0	9.67
2	50	50	75.2	5.45	63.6	31.71
2	75	50	64.1	9.68	65.4	42.14
2	100	50	53.2	14.77	54.2	74.13
2	125	50	46.5	20.46	43.6	104.53
2	150	50	35.2	26.62	38.4	109.97
2	175	50	32.7	34.40	32.6	172.39
2	200	50	28.3	40.77	28.8	210.87
2	1	60	110	1.13	109	0.85
2	25	60	101	2.45	99.3	9.59
2	50	60	85.2	5.42	87.8	31.81
2	75	60	70.7	9.60	70.1	59.20
2	100	60	57.2	14.86	53.3	74.37
2	125	60	45.0	20.30	44.6	83.58

Продовження таблиці 2.1

Кількість каналів	**Час затримки (ms)**	**Пропускна здатність каналу (Mbits/sec)**	**Vtrunkd Пропускна здатність (Mbits/sec)**	**Vtrunkd Середнє квадратичне відхилення (ms)**	**OpenVpn Пропускна здатність (Mbits/sec)**	**OpenVpn Середнє квадратичне відхилення (ms)**
2	150	60	37.5	26.66	38.2	136.51
2	175	60	32.4	33.02	32.5	172.42
2	200	60	30.8	40.70	29.1	211.46
2	1	70	127	1.93	130	1.93
2	25	70	118	2.57	111	10.08
2	50	70	95.2	5.90	95.0	31.65
2	75	70	71.6	9.64	74.9	48.24
2	100	70	57.7	15.06	56.5	74.34
2	125	70	38.7	20.36	45.2	103.98
2	150	70	39.0	26.82	38.4	110.01
2	175	70	31.1	33.61	35.1	138.73
2	200	70	29.1	40.92	29.5	176.55
2	1	80	147	1.29	146	1.46
2	25	80	131	3.13	121	13.79
2	50	80	95.2	5.28	101	21.11
2	75	80	75.6	9.74	76.6	48.37
2	100	80	56.2	14.48	57.0	77.15
2	125	80	47.4	20.75	49.2	85.05
2	150	80	38.0	26.57	51.6	136.35
2	175	80	32.3	33.40	34.2	138.56
2	200	80	29.7	40.80	26.5	210.94

Кінець таблиці 2.1

Кількість каналів	**Час затримки (ms)**	**Пропускна здатність каналу (Mbits/sec)**	**Vtrunkd Пропускна здатність (Mbits/sec)**	**Vtrunkd Середнє квадратичне відхилення (ms)**	**OpenVpn Пропускна здатність (Mbits/sec)**	**OpenVpn Середнє квадратичне відхилення (ms)**
2	1	90	166	2.88	161	0.63
2	25	90	130	2.74	135.1	10.44
2	50	90	102	5.29	106	26.59
2	75	90	76.3	9.68	77.7	45.73
2	100	90	60.2	14.63	53.8	74.18
2	125	90	46.3	20.35	49.2	83.83
2	150	90	39.9	26.35	41.0	136.43
2	175	90	33.6	33.43	36.5	172.42
2	200	90	32.2	40.70	30.1	169.25
2	1	100	173	1.03	166	1.70
2	25	100	152	2.97	145	10.51
2	50	100	110	5.60	110	26.98
2	75	100	74.6	9.66	77.7	48.51
2	100	100	54.4	14.71	60.3	75.03
2	125	100	46.9	20.42	50.1	84.05
2	150	100	38.9	26.77	42.1	110.36
2	175	100	37.8	33.49	35.7	172.26
2	200	100	31.0	40.57	33.4	176.66

Таблиця 2.2

Відсоток втрати пакетів	Пропускна здатність каналу (Mbits/sec)	Vtrunkd Пропускна здатність (Mbits/sec)	OpenVpn Пропускна здатність (Mbits/sec)
1	10	11.6	19.2
2	10	12.0	18.8
3	10	9.76	17.6
4	10	7.50	18.5
5	10	8.33	18.5
6	10	6.28	6.35
7	10	4.81	14.4
8	10	5.11	11.0
9	10	5.08	8.99
10	10	3.99	7.81
1	20	22.4	22.5
2	20	26.6	24.6
3	20	16.9	15.4
4	20	12.1	15.4
5	20	10.4	7.79
6	20	10.6	9.26
7	20	8.30	7.04
8	20	8.64	7.77
9	20	7.79	4.96
10	20	6.42	3.44
1	30	34.5	32.8
2	30	34.1	23.5
3	30	23.2	18.8
4	30	12.7	13.2
5	30	13.1	18.3

Продовження таблиці 2.2

Відсоток втрати пакетів	Пропускна здатність каналу (Mbits/sec)	Vtrunkd Пропускна здатність (Mbits/sec)	OpenVpn Пропускна здатність (Mbits/sec)
6	30	10.1	15.2
7	30	7.25	7.60
8	30	6.81	8.27
9	30	6.10	5.31
10	30	6.13	4.31
1	40	31.4	33.0
2	40	24.7	31.0
3	40	19.0	29.1
4	40	17.1	21.5
5	40	16.8	14.1
6	40	11.0	9.14
7	40	16.3	7.65
8	40	9.25	7.07
9	40	8.80	4.74
10	40	5.68	3.75
1	50	51.3	85.3
2	50	38.2	67.5
3	50	29.6	63.9
4	50	22.4	51.7
5	50	15.3	54.1
6	50	10.5	24.8
7	50	9.17	13.3
8	50	7.27	10.5
9	50	5.31	8.85
10	50	6.56	7.41

Продовження таблиці 2.2

Відсоток втрати пакетів	Пропускна здатність каналу (Mbits/sec)	Vtrunkd Пропускна здатність (Mbits/sec)	OpenVpn Пропускна здатність (Mbits/sec)
1	60	50.8	112
2	60	34.5	99.0
3	60	29.9	81.4
4	60	34.3	50.8
5	60	12.8	31.2
6	60	11.3	21.8
7	60	10.7	16.7
8	60	5.41	12.0
9	60	5.15	11.2
10	60	5.17	6.31
1	70	51.3	118
2	70	44.5	82.9
3	70	24.4	72.3
4	70	11.8	45.1
5	70	13.8	23.4
6	70	12.0	22.5
7	70	9.32	17.1
8	70	8.58	11.3
9	70	5.98	8.05
10	70	5.15	4.75
1	80	61.8	126.3
2	80	34.8	106
3	80	29.0	52.4
4	80	24.6	36.3
5	80	11.2	20.7

Кінець таблиці 2.2

Відсоток втрати пакетів	Пропускна здатність каналу (Mbits/sec)	Vtrunkd Пропускна здатність (Mbits/sec)	OpenVpn Пропускна здатність (Mbits/sec)
6	80	11.6	18.8
7	80	12.8	14.5
8	80	5.61	13.7
9	80	4.97	8.85
10	80	5.15	7.24
1	90	40.7	147
2	90	29.5	105
3	90	16.5	55.6
4	90	14.2	31.8
5	90	16.4	34.1
6	90	11.1	21.7
7	90	7.61	16.6
8	90	4.78	11.4
9	90	5.55	8.34
10	90	4.37	6.14
1	100	50.4	150
2	100	40.1	91.5
3	100	24.4	69.0
4	100	27.7	41.0
5	100	15.1	28.5
6	100	7.49	19.4
7	100	6.76	20.8
8	100	8.48	13.9
9	100	7.90	7.13
10	100	4.65	4.89

Було почергово встановлено швидкості починаючи з 10 Мбіт/с зі збільшенням до 100 Мбіт/с з шагом 10 Мбіт/с для всіх каналів між машинами VPN клієнта і сервера. Щоб встановити затримок поширення між клієнтом і сервером, контролер трафіку використовувався на кожному інтерфейсі хостів. Наприклад, якщо була встановлена затримка поширення 50 мс, тоді кожен інтерфейс матиме затримку поширення 25 мс. Таким чином затримка вважалася коригуванням RTT, а не односторонньою затримкою.

Зібрані результити представлені в вигляді графіків залежності середньої квадратичного відхилення від затримок та залежності пропускної здатності від затримок для одноканального та двоканального режимів роботи на рисунках рис 2.3, рис 2.4, рис 2.5, рис 2.6.

Аналізуючи дані графіки можна побачити що пропускна здатність особливо не відрізняється при використанні vtrunkd чи openvpn, вона залишається приблизно на одному і тому самому рівні. Також можна побачити значне зменшення продуктивності при збільшенні затримок, що є очевидним.

Результати середнього квадратичного відхилення виявилися більш різноманітними. Vtrunkd показав стабільні значення, яке мало залежать від пропускної здатності каналів. На відміну від нього, openVpn показав досить велике відхилення значень. При порівнянні також видно що vtrunkd паказав значення менший разброс в порівнянні з openVpn, в 5 разів.

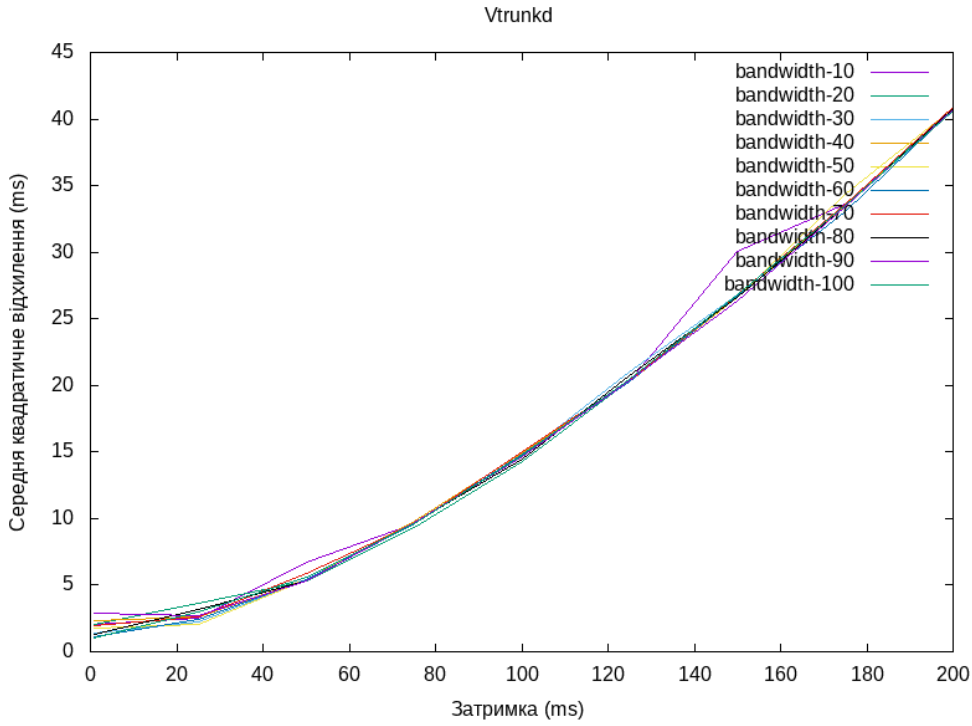


Рисунок 2.3

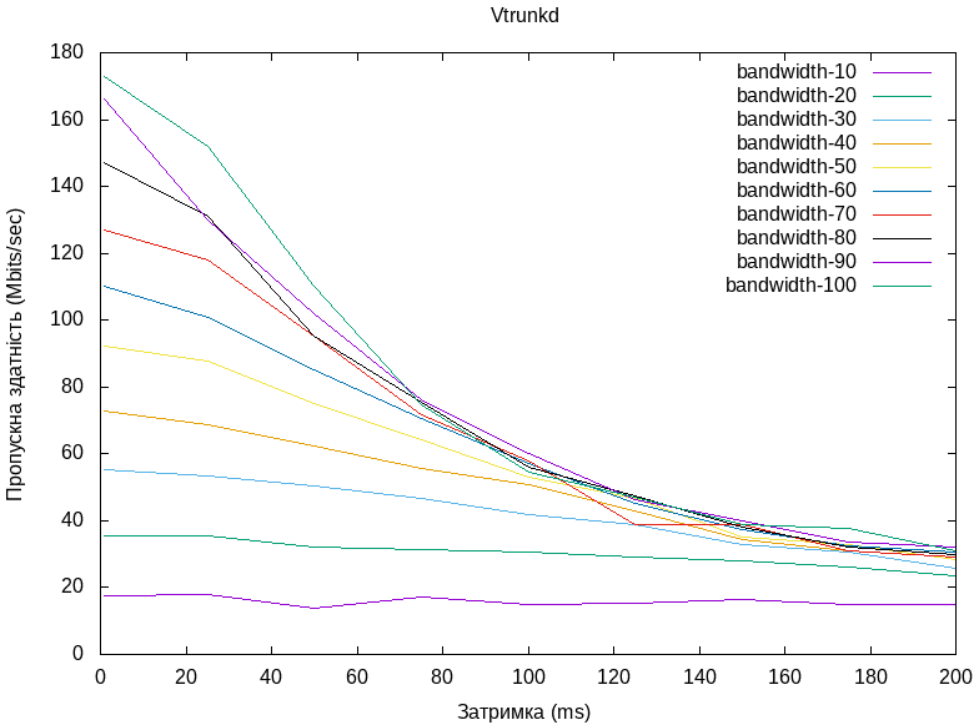


Рисунок 2.4

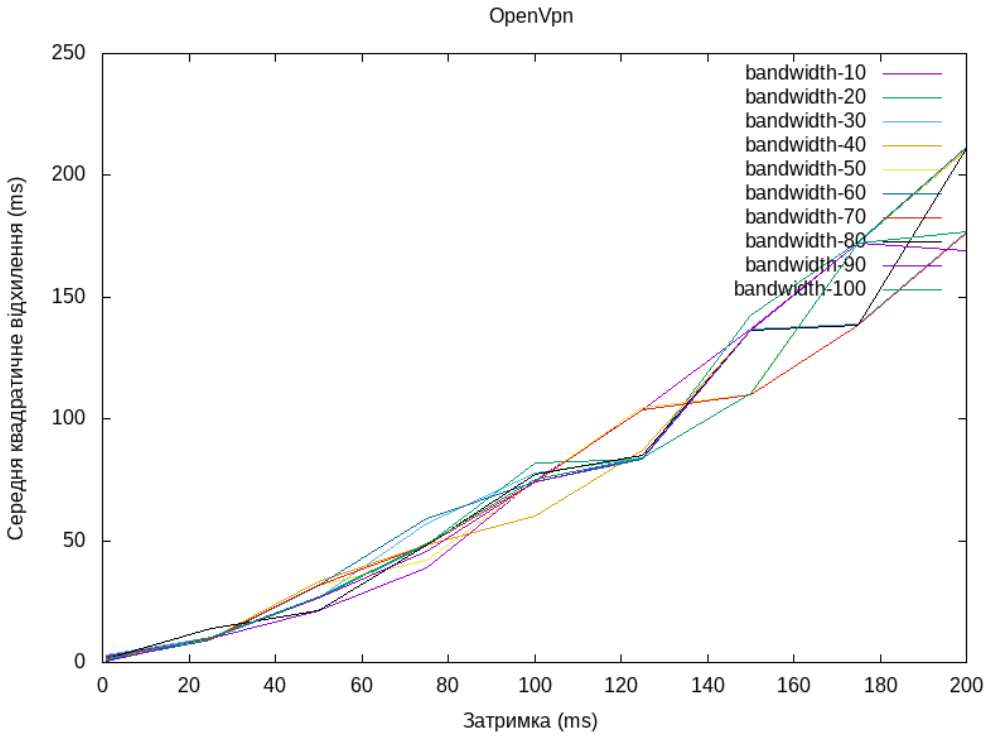


Рисунок 2.5

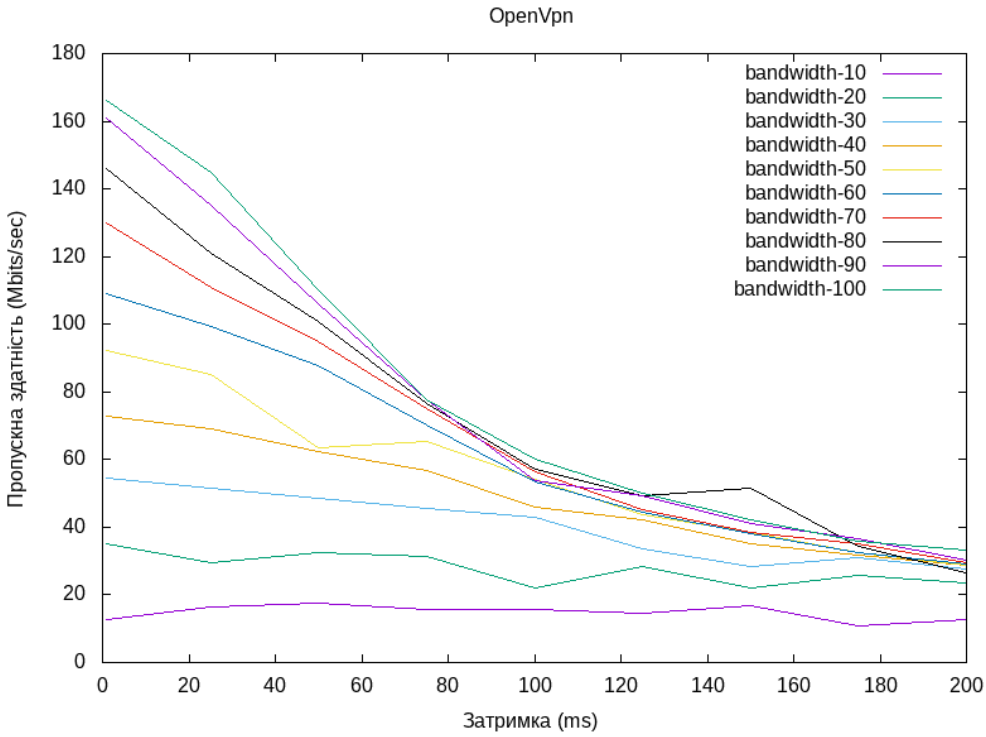


Рисунок 2.6

В графіках які зображені на рисунках 2.7 та 2.8 наводиться ,для порівнянні, зміна пропускної здатності каналу при різних відсотках втрати пакетів для vtrunkd та openvpn відповідно. З даних графіків видно, що openvpn справляється з цією задачею краще, хоча дана різниця не так замітна при низькій ширині каналів.

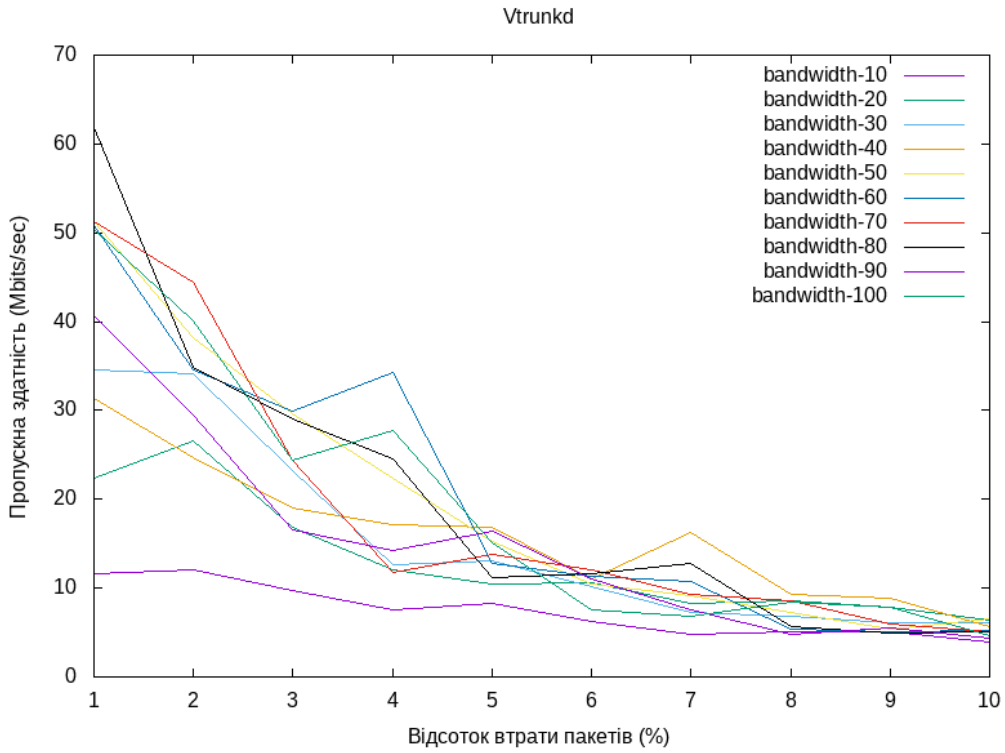


Рисунок 2.7

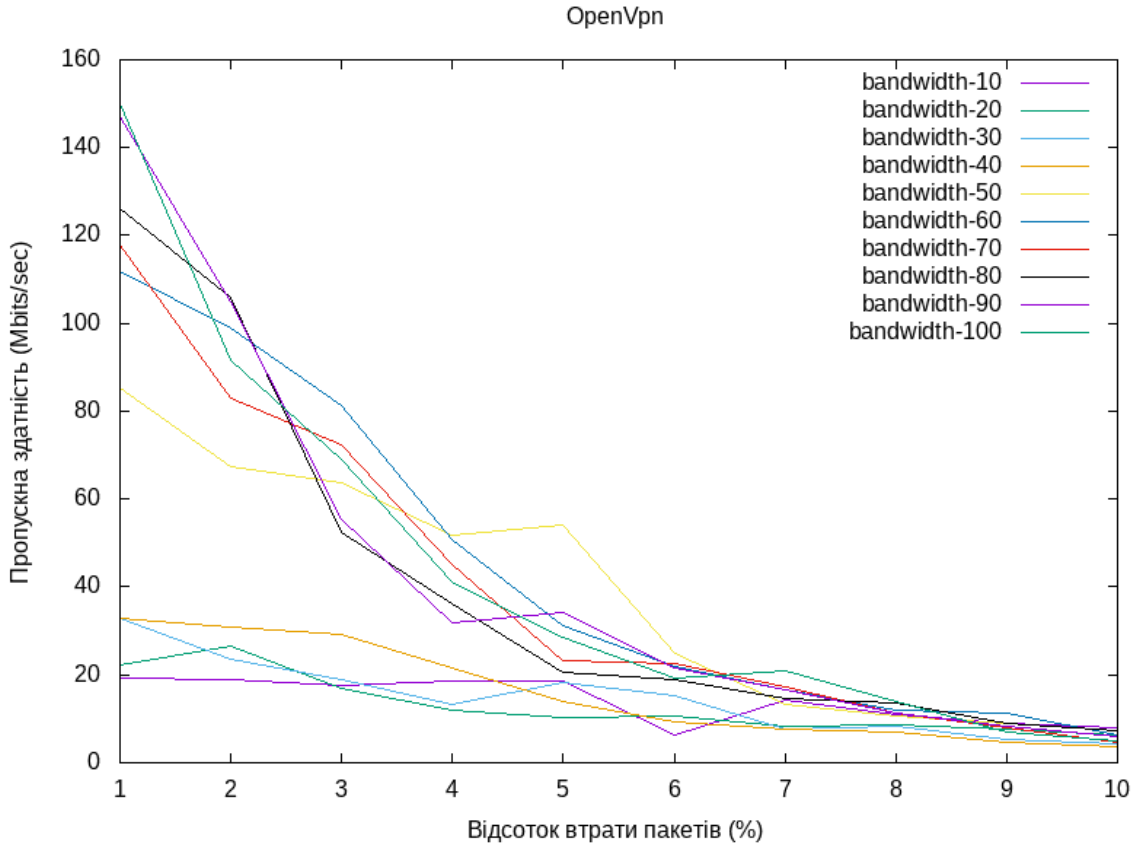


Рисунок 2.8

2.4. Тестування відмовостійкості

Для тестування відмовостійкості був використаний той самий стенд який описувався вище. Основною ціллю було розглянути як буде

поводити себе MPTCP при відключенні мережевих інтерфейсів, та при їх відновленні.

З рисунку 2.7 видно що між 20 та 40 секундами пропало одне з з'єднання між хостами, але під час цього MPTCP з'єднання не пропало. Після відновлення інтерфейсів відновилося і з’єднання з VPN сервером через новий канал. Тобто клієнт зміг не втратити з’єднання.

Таким чином була перевірена вудмовостійкість з’єднання. Це показує що використання багатоканального транспорту є ефективним способом підвищення відмовостійкості, так як при втраті одного з каналів інші залишаються в працездатному стані і можуть виконувати свої функції, що забезпечує підтримання з’єднання з vpn сервером.

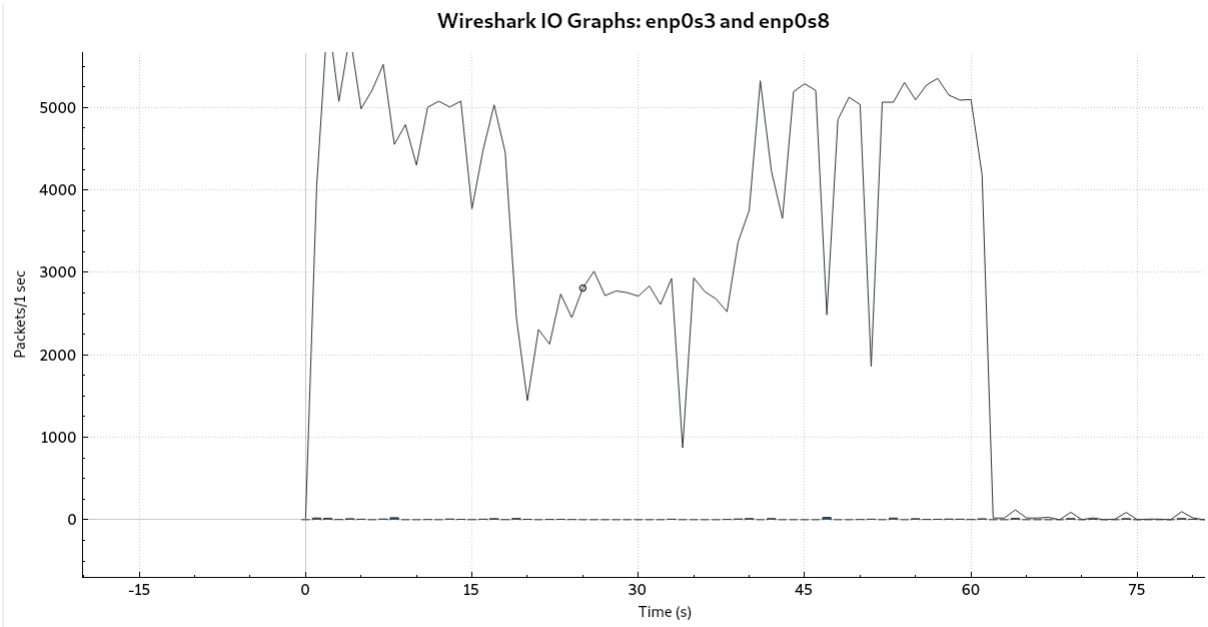


Рисунок 2.7

Висновок до розділу 2

У цьому розділі було порівняно два vpn при використанні MPTCP для багатоканального транспорту. Тести де був низький рівень втрат і RTT, показали кращу продуктивність. Vtrunkd, в порівнянні з OpenVpn, показав кращу стабільність в плані затримок, це видно з кращих показників середньоквадратичного відхилення.

Також було розглянуто поведінку з'єднання при різному відсотку втрат пакетів, що показало що vtrunkd не так добре справляється з даною задачею, а на противагу йому openvpn показує більш стабільні результати, проте ця різниця видна тільки при великих пропускних здатностях самих каналів та низькому відсотку втрат. При збільшенні відсотку втрат пакетів ця різниця стає не такою значною.

Незалежно від продуктивності, користувачі можуть приділяти більше уваги відмово стійкості. З точки зору відмово стійкості, ми змогли реалізувати переваги використання з'єднання MPTCP, зосереджуючись на мобільних хостах, які підключаються до шлюзу VPN. Дуже помітно була можливість відкидати один або кілька інтерфейсів і підтримувати завантаження або швидко відновити завантаження, щойно буде встановлено нову IP-адресу. Показано використання VPN тунелю поверх протоколу MPTCP забезпечує переваги продуктивності з точки зору досяжної пропускної здатності.

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ ДЛЯ АВТОМАТИЧНОГО НАЛАШТУВАННЯ ТРАНСПОРТНОГО ТА ПРИКЛАДНОГО РІВНЯ ВІРТУАЛЬНОЇ ПРИВАТНОЇ МЕРЕЖІ НА БАЗІ MPTCP ТА VTRUNKD

3.1. Причини розробки

Під час довгого процесу тестування та збору тестових результатів були виявленні деякі незручності. Зокрема налаштування таблиць маршрутизації та запуск vtrunkd процесів для кожного каналу окремо виявилися досить часозатратними операціями, особливо в ті моменти коли виявляється після тестування що один з процесів був не запущеним або маршрутизація неправильно налаштована. В такі моменти прийшлося заново пере налаштовувати тестовий стенд та запускати тестування по новій.

Тому було вирішено автоматизувати даний процес, щоб спростити налаштування даної системи. Також теоретичним плюсом такої системи буде підвищення відмовостійкості за рахунок автоматичної перевірки наявних мережевих інтерфейсів, що дозволяє динамічно змінювати налаштування шляхів.

3.2. Планування структури проекту та реалізації основного функціоналу

Дана система повинна мати зручний та зрозумілий інтерфейс для внесення налаштувань mptcp та vtrunkd. Налаштування потрібно зберігати в локальному сховищі щоб користувачу не довелось постійно пере заповнювати конфігурації. І самою основною вимогою є постійний моніторинг наявних мережевих інтерфейсів та певна реакція на появу або зникнення їх, що буде проявлятись в налаштуваннях маршрутизації.

Для реалізації був вибраний фреймворк flutter так як досить добре підходить для розробки прототипів невеликих проектів, та легкий в освоєнні.

Flutter — це фреймворк з відкритим вихідним кодом від Google для створення красивих, скомпільованих, багатоплатформних програм із єдиної кодової бази.[11]

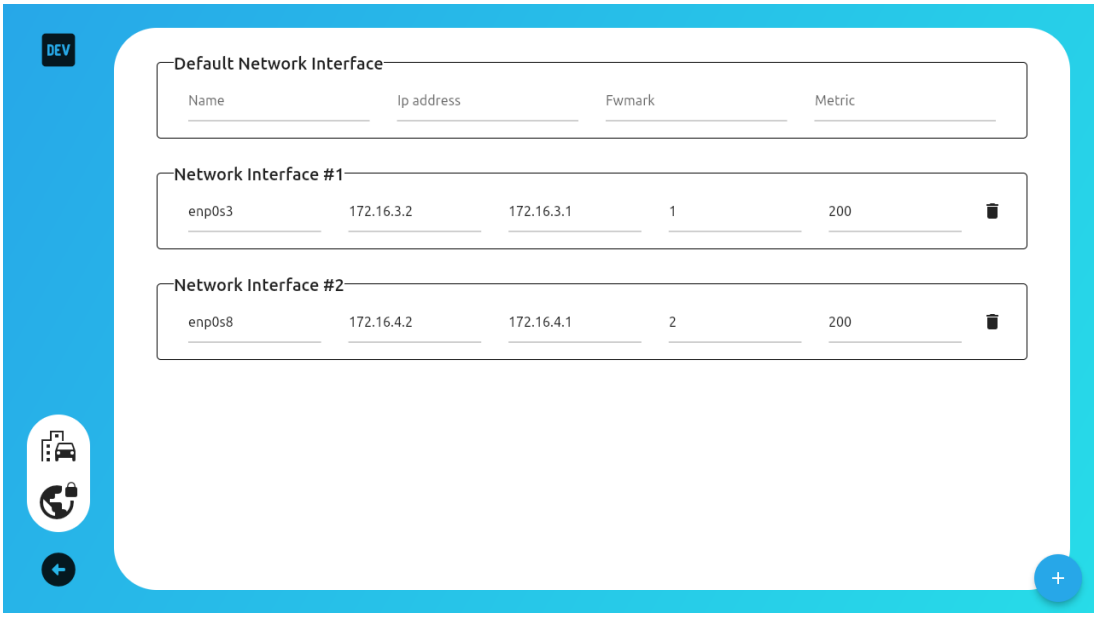


Рисунок 3.1

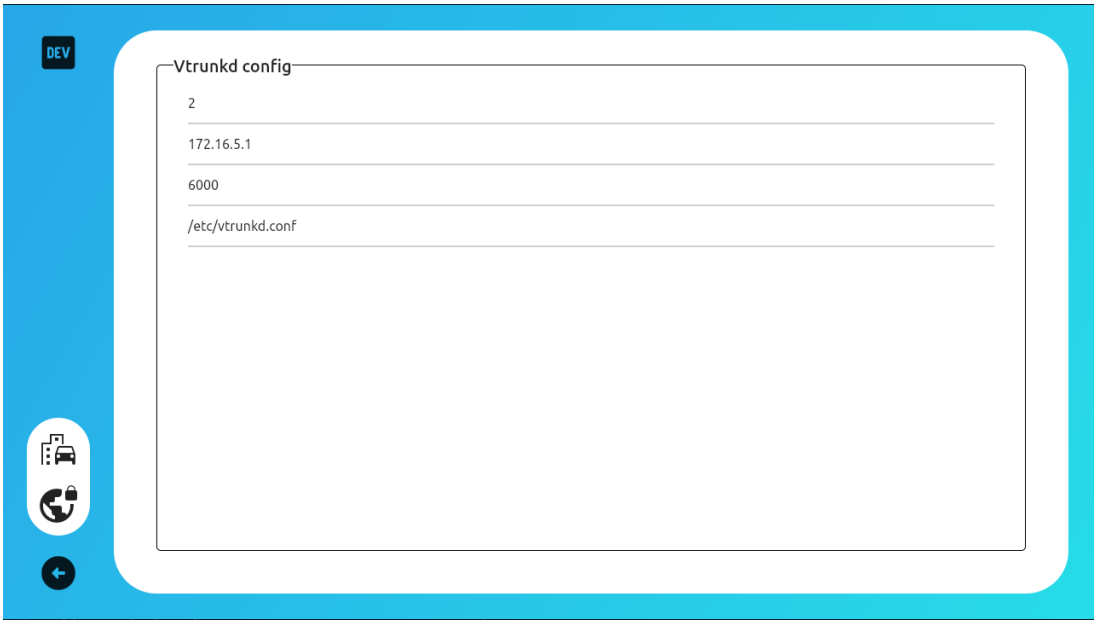


Рисунок 3.2

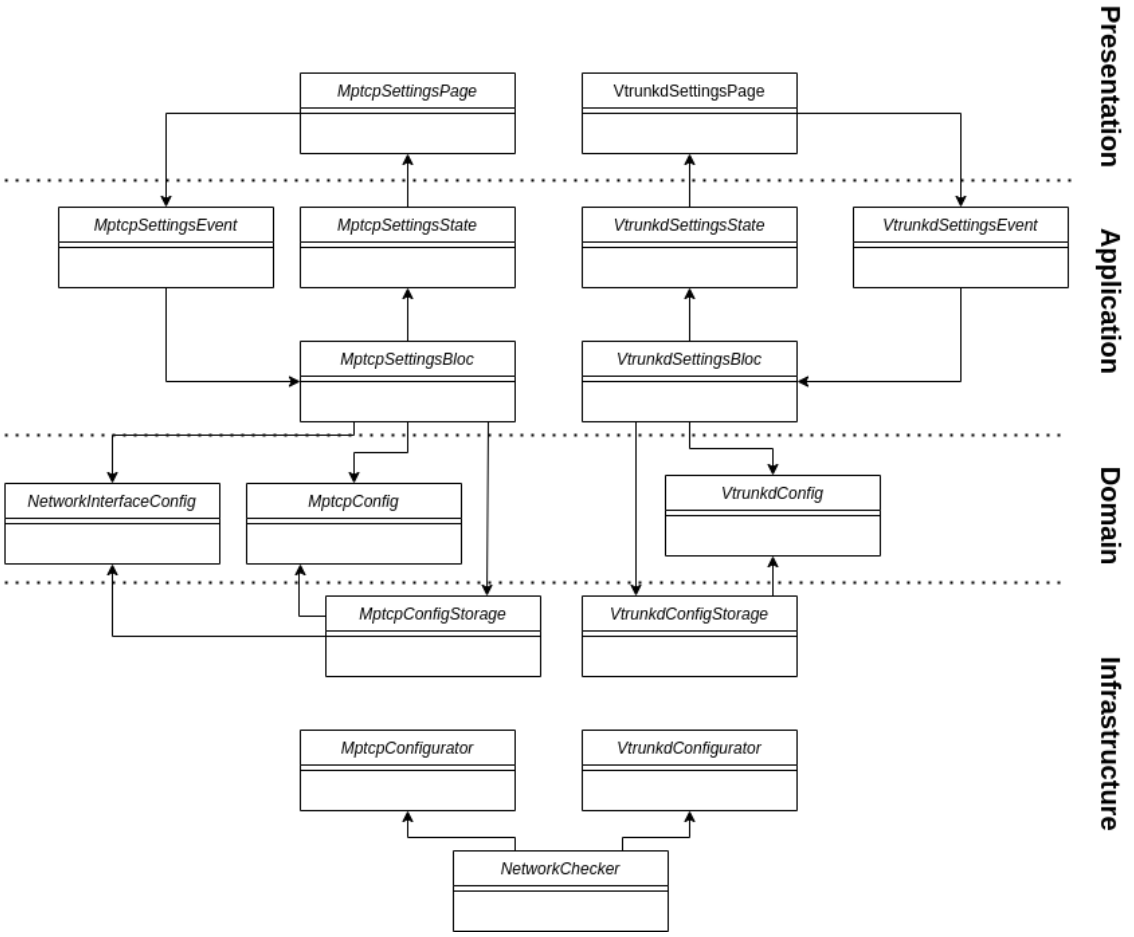


Рисунок 3.3

На рисунку 3.3 зображена архітектура проекту, що розділена на 4 основні області.

Першою є область під назвою “Presentation”, яка відповідає за класи які відповідають за зовнішній вигляд програми (інтерфейс) з яким буде працювати користувач. Ця область представлення двома класами MptcpSettingsPage та VtrunkdSettingsPage, які відповідають за налаштування відповідних конфігурацій. Інтерфейс користувача зображений на рисунках 3.1 та 3.2 для кращого розуміння.

Другою є область яка відповідає за стан відображення та його зміну, та назва цього розділу “Application”. Для кожної сторінки були описані свої сигнали (event) та можливі стани (state), для цього був використаний популярний шаюлон під назвою BLoC. За допомогою даного способу можна гручко добавляти нові сигнали та стани в майбутньому.

Третя область відповідає за прикладну область “Domain”. Там описані основні об'єкти з якими працює програма. Дана область відповідає за опис

базових об'єктів які можуть використовуватись усіма іншими частинами програми.

І заключною четвертою область є “Infrastructure”. Всі програмні елементи які знаходяться в даній області використовуються як інфраструктура, тобто зовнішні класи які забезпечують виконання певної функціональності, за яку не повинна відповідати жодна інша область. В нашому випадку там знаходяться класи для конфігурації mptcp та vtrunkd на машині клієнта та клас для перевірки мережевих інтерфейсів.

Найбільш цікавою для розгляду є реалізація саме автоматичного налаштування та логіка яка описує дану реалізацію.

Перед операціями налаштування наперед створюються таблиці маршрутизації та кожна позначається відповідними мітками. Налаштування починається з визначення активних мережевих інтерфейсів. Після чого потрібно відібрати ті інтерфейси які були відключені в порівнянні з попередньою перевіркою, та відібрати нові мережеві інтерфейси. Далі можна переходити до самих налаштувань на інтерфейсах.

Для тих інтерфейсів які були відключені потрібно звільнити відповідну таблицю маршрутизації, проте це відбувається автоматично при відключенні інтерфейсу тому додаткової реалізації не потребує. Для нових інтерфейсів потрібно добавити правило маршрутизації скориставшись утилітою ip route. Після того як для всіх нових мережевих інтерфейсів буде налаштовано маршрутизацію можна приступити до налаштування vtrunkd. Якщо змінились якісь параметри vtrunkd потрібно зупинити всі попередні процеси та запустити нові з зміненими параметрами.

Для того щоб перевірка працювала постійно було вирішено використовувати пакет cron для перевірки кожні 10 секунд нових з'єднань.

Класс MptcpConfigurator реалізовував логіку для конфігурації маршрутизації.

- prepare(tableCount) - метод для підготовки усіх позначок для кожної таблиці маршрутизації
- configure(config, networkInterfaces) - метод для конфігурації нових мережевих інтерфейсів

```
class MptcpConfigurator {
  _getGatewayAddress(String address) {
    final addressData = address.split(".");
    addressData[3] = "1";
    final routerAddress = addressData.join(".");
    return routerAddress;
  }

  NetworkInterfaceConfig _getInterfaceConfig(
    MptcpConfig config,
    NetworkInterface networkInterface,
  ) {
    for (final networkInterfaceConfig
        in config.networkInterfaceConfigs.values) {
      if (networkInterfaceConfig.ipAddress != null &&
          networkInterfaceConfig.ipAddress != networkInterface.addresses[0]) {
        continue;
      }
      if (networkInterfaceConfig.dev != networkInterface.name) {
        continue;
      }
      if (networkInterfaceConfig.index != null &&
          networkInterfaceConfig.index != networkInterface.index) {
        continue;
      }
      return networkInterfaceConfig;
    }
    return config.defaultConfig;
  }

  prepare(int tableCount) async {
    for (int i = 101; i <= 100 + tableCount; i++) {
      await Process.run("ip", ["rule", "add", "fwmark", "0x$i", "lookup", "$i"])
          .then((ProcessResult res) {
        if (kDebugMode) {
          print(res.exitCode);
          print(res.stdout);
          print(res.stderr);
        }
      });
```

```
    }
  }

  configure(
    MptcpConfig config,
    List<NetworkInterface> networkInterfaces,
  ) async {
    for (final networkInterface in networkInterfaces) {
      // Вибірка конфігурації для даного інтерфейсу
      final networkInterfaceConfig =
          _getInterfaceConfig(config, networkInterface);
      if (networkInterfaceConfig.dev == "") continue;
      if (networkInterfaceConfig.fwmark == null) continue;

      final gateway = networkInterfaceConfig.via ??
          _getGatewayAddress(
            networkInterface.addresses[0].address,
          );

      final tableNumber = networkInterfaceConfig.fwmark;

      await Process.run("ip", [
        "route", "add", "default", "via", gateway, "dev",
networkInterfaceConfig.dev,
        "table", "$tableNumber", "metric", "${networkInterfaceConfig.metric}"
      ]).then((ProcessResult res) {
        if (kDebugMode) {
          print(res.exitCode);
          print(res.stdout);
          print(res.stderr);
        }
      });
    }
  }
}
```

Класс VtrunkdConfigurator реалізовував логіку для конфігурації vtrunkd.

- configure(config, networkInterfaces) - метод для конфігурації vtrunkd сесій.

```
class VtrunkdConfigurator {
  configure(
      VtrunkdConfig config, List<NetworkInterface> networkInterfaces) async {
```

```
    for (final sessionNumber in config.sessions) {
      await Process.run("vtrunkd", [
        "-P", "${config.port}",
        "-f",              config.configPath,              "000000_$sessionNumber",
"${config.serverIpAddress}",
      ]).then((ProcessResult res) {
        if (kDebugMode) {
          print(res.exitCode);
          print(res.stdout);
          print(res.stderr);
        }
      });
    }
  }
}
```

Класс NetworkChecker реалізовує визначення нових інтерфейсів та передає їх в MptcpConfigurator та VtrunkdConfigurator.

- checkInterfaces - метод для визначення нових мережевих інтерфейсів та передачі їх для подальшої конфігурації mptcp та vtrunkd.

```
class NetworkChecker {
  final MptcpConfigurator mptcpConfigurator;
  final MptcpConfig mptcpConfig;
  final VtrunkdConfigurator vtrunkdConfigurator;
  final VtrunkdConfig vtrunkdConfig;

  final Map<String, int> _configuredDevices = {};
  static const tablesCount = 10;
  final List<int> tableUnused = List.generate(tablesCount, (i) => i + 101);

  NetworkChecker({
    required this.mptcpConfigurator,
    required this.mptcpConfig,
    required this.vtrunkdConfigurator,
    required this.vtrunkdConfig,
  });

  Future<List<NetworkInterface>> getInrefaces() async {
    final interfaces = await NetworkInterface.list();
    return interfaces;
  }
```

```
  Future checkInterfaces() async {
    await mptcpConfigurator.prepare(tablesCount);
    // take current interfaces
    var newInterfaces = await getInrefaces();

    // Remove existed devices
    final removedDevices = Map<String, int>.from(_configuredDevices);
    removedDevices.removeWhere(
      (dev, _) => newInterfaces.any((interface) => interface.name == dev),
    );

    newInterfaces = newInterfaces
        .where(
          (interface) => !_configuredDevices.containsKey(interface.name),
        )
        .toList();

    for (final dev in removedDevices.keys) {
      tableUnused.add(removedDevices[dev]!);
      _configuredDevices.remove(dev);
    }

    await mptcpConfigurator.configure(
      mptcpConfig,
      newInterfaces,
    );
    await vtrunkdConfigurator.configure(
      vtrunkdConfig,
      newInterfaces,
    );
  }
}
```

Для збереження даних використовувались класи MptcpConfigStorage та VtrunkdConfigStorage. Дані класи реалізовували збереження та завантаженна конфігурацій методами saveConfig та loadOrDefaultConfig відповідно. Дані класи реалізовані на основі сховища GetStorage з однойменного модуля get_storage. Код даних класів наведений нище.

```
class MptcpConfigStorage implements IMptcpConfigStorage {
  static final defaultConfig = MptcpConfig();

  @override
```

```
  saveConfig(MptcpConfig config) {
    final box = GetStorage();
    box.write("mptcp_config", config);
    return config;
  }

  @override
  MptcpConfig loadOrDefaultConfig() {
    final box = GetStorage();
    final config =
        box.hasData("mptcp_config") ? box.read("mptcp_config") : defaultConfig;
    return config;
  }
}

class VtrunkdConfigStorage implements IVtrunkdConfigStorage {
  static final defaultConfig = VtrunkdConfig(
    configPath: "/etc/vtunkd.conf",
    port: 6000,
    serverIpAddress: InternetAddress("172.16.5.1"),
    sessions: [1, 2],
  );

  @override
  saveConfig(VtrunkdConfig config) {
    final box = GetStorage();
    box.write("vtrunkd_config", config);
    return config;
  }

  @override
  VtrunkdConfig loadOrDefaultConfig() {
    final box = GetStorage();
    final config = box.hasData("vtrunkd_config")
        ? box.read("vtrunkd_config")
        : defaultConfig;
    return config;
  }
```

}

3.3. Результати розробки

Встановивши дану програму на клієнтський хост та налаштувавши як зображено на рисунках 3.1 та 3.2 автоматично протягом 10 секунд були налаштовані маршрутизація та створений тунель vtrunkd.

Також для перевірки динамічного налаштування. Був добавлений новий роутер з ip адресами 172.16.8.1 та 172.16.9.2 відповідно на enp0s8 та enp0s3 відповідно, аналогічно попереднім роутерам, від мав з'єднання з сервером через enp0s3. Добавлено в конфігураційний файл vtrunkd нову сесію 000000_3 аналогічну сесіям 000000_1 та 000000_2 .У розробленій програмі були добавленні нові налаштування які виглядають наступним чином:

- Name - enp0s9
- Ip address - 172.16.8.2
- Via - 172.16.8.1
- Fwmark - 3
- 200

Через незначний час після підняття даного каналу було помічено створення нової таблиці маршрутизації та нового з'єднання vtrunkd, що і очікувалось.

Запустивши автоматичне тестування як при тестах в другому розділі результати були ідентичними, що ще раз доводить працездатність даного конфігуратора.

Висновок до розділу 3

У даному розділі розроблено програму для автоматичного налаштування мережевих транспортних каналів та віртуальної приватної мережів vtrunkd. Було описано принцип роботи даного конфігуратора та наведені частини реалізації. Також даний конфігуратор був перевірений на тестовому стенді який описаний в попередньому розділі. Аналізуючи результати тестування можна сказати що з поставленою задачею конфігуратор справляється ефективно.

ВИСНОВКИ

Метою даної дипломної роботи було розглянути підтримку існуючих систем багатоканального транспорту для віртуальних приватних мереж та покращити їх.

Робота над дипломним проектом була спрямована на налаштування, тестування та покращення багатоканального транспорту, в результаті чого була розроблена програма автоматичного налаштування транспортних каналів та vtrunkd для клієнтського хоста.

У першому розділі було проаналізовано існуючі vpn та протокол mptcp. Було розглянуто принципи роботи протоколу багатоканального транспорту, його плюси та недоліки.

У другому розділі було проведене порівняльне тестування для визначення проблем які присутні в таких відкритих реалізаціях vpn як openvpn та vtrunkd при використанні mptcp на транспортному рівні. Результати здебільшого виявилися досить схожими, хоча при тестуванні з різним рівнем втрат пакетів openvpn показав себе більш стабільно.

У третьому розділі було описано систему автоматичного налаштування. Було описано декілька класів, які відіграють основну роль, їх методи та їх взаємодію одне між одним. Також були наведені скріншоти розробленої програми. Було наведено детальна архітектура проекту в додатку 1 для кращої наглядності.

В результаті розробки була отримана програма яка полегшує взаємодію з vtrunkd та покращує відмово стійкість за рахунок постійного оновлення активних мережевих інтерфейсів. Протестувавши отриману програму, можна сказати, що цілі які ставились при розробці виконані та програма відпрацьовує коректно.

Використані джерела:

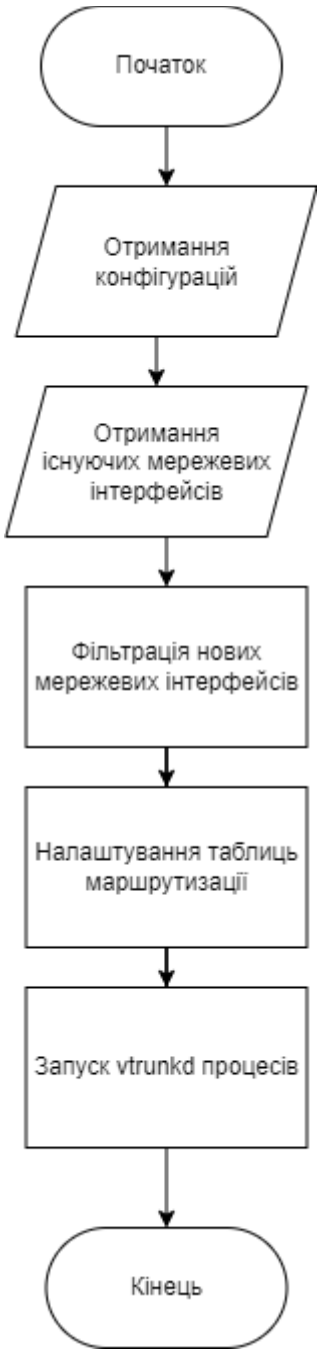
1. B. Gleeson, A. Lin, J. Heinanen, Telia Finland, G. Armitage, and A. Malis, “A Framework for IP Based Virtual Private Networks”, RFC 2764, February 2000. [Електронний ресурс] - http://www.rfc-editor.org/rfc/rfc6824.txt
2. O. Titz. “Why TCP over TCP is a bad idea.” [Електронний ресурс] - http://sites.inka.de/bigred/devel/tcp-tcp.html. Accessed January 15, 2017.
3. O. Honda, H. Ohsaki, M. Imase, M. Ishizuka, and J. Murayama, “Understanding TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency,” in Proc. SPIE, vol. 6011, 2005, pp. 60 110H–60 110H. [Електронний ресурс] - http://dx.doi.org/10.1117/12.630496
4. Wireguard [Електронний ресурс] - https://www.wireguard.com/
5. Alois Paulus “OpenVPN and Multipath TCP” [Електронний ресурс] - https://dial.uclouvain.be/memoire/ucl/en/object/thesis:2667/datastream/PDF_01/view
6. “Vtrunkd” [Електронний ресурс] - https://github.com/VrayoSystems/vtrunkd
7. Christoph Paasch “Improving Multipath TCP” [Електронний ресурс] - https://inl.info.ucl.ac.be/system/files/phd-thesis_1.pdf
8. “Mptcp_net-next Wiki” [Електронний ресурс] - https://github.com/multipath-tcp/mptcp_net-next/wiki/
9. A. Ford, Cisco, C. Raiciu, U. Politechnica of Bucharest, M. Handley, U. College London, O. Bonaventure, catholique de Louvain “TCP Extensions for Multipath Operation with Multiple Address” January 2013 [Електронний ресурс] - https://www.rfc-editor.org/rfc/rfc6824.html
10. A. Ford, C. Raiciu, M. Handley, Pexip, U. Politechnica of Bucharest, U. College London, O. Bonaventure, C. Paasch ,U. catholique de Louvain, Apple, Inc “TCP Extensions for Multipath Operation with Multiple Address” March 2020 [Електронний ресурс] - https://www.rfc-editor.org/rfc/rfc6884.html
11. ”Flutter – Build apps for any screen” [Електронний ресурс] - https://flutter.dev/

Додаток 1

ПРИНЦИПОВА СХЕМА РОБОТИ

до дипломного проєкту на тему: «Система багатоканального транспорту для віртуальних приватних мереж»

Київ – 2022



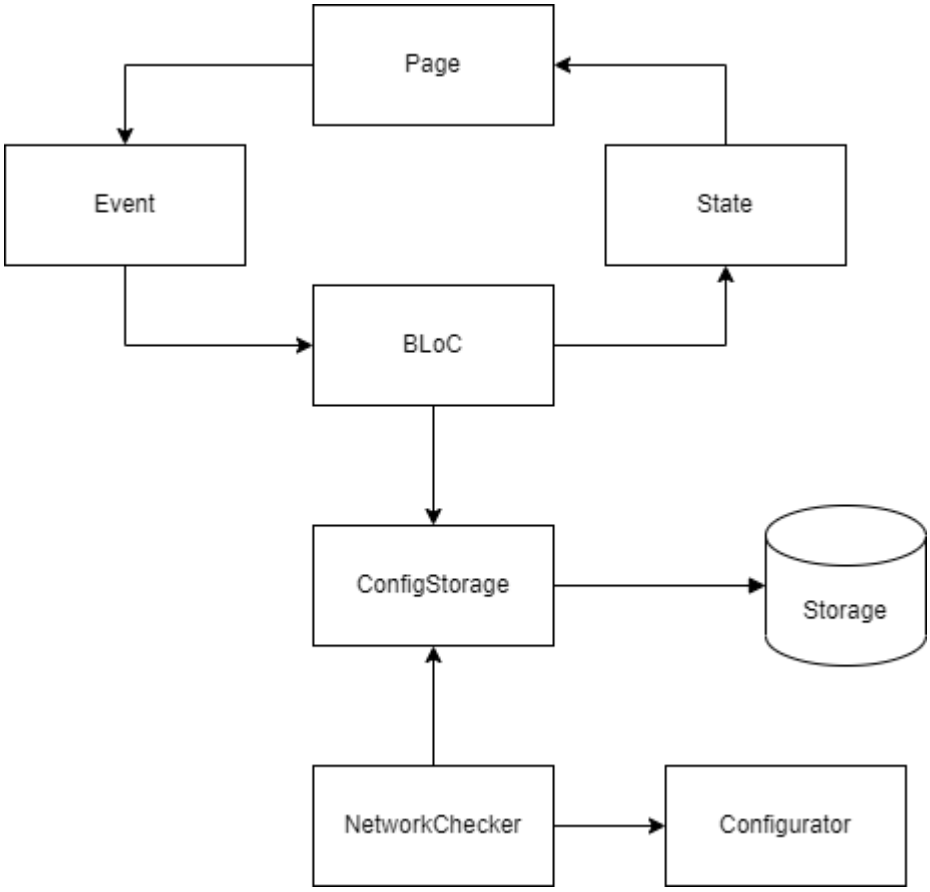
					ІАЛЦ 467100.004 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Смірнов Н. О.			Система багатоканального транспорту для віртуальних приватних мереж. Принципова схема ситеми	Лит.	Аркуш 1 / Аркушів 1
Перевірив		Роковий О. П.					
Т. Контр.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81	
Н. Контр.		Сімоненко В. П					
Затвердив							

Додаток 2

СТРУКТУРНА СХЕМА РОБОТИ

до дипломного проєкту на тему: «Система багатоканального транспорту для віртуальних приватних мереж»

Київ – 2022



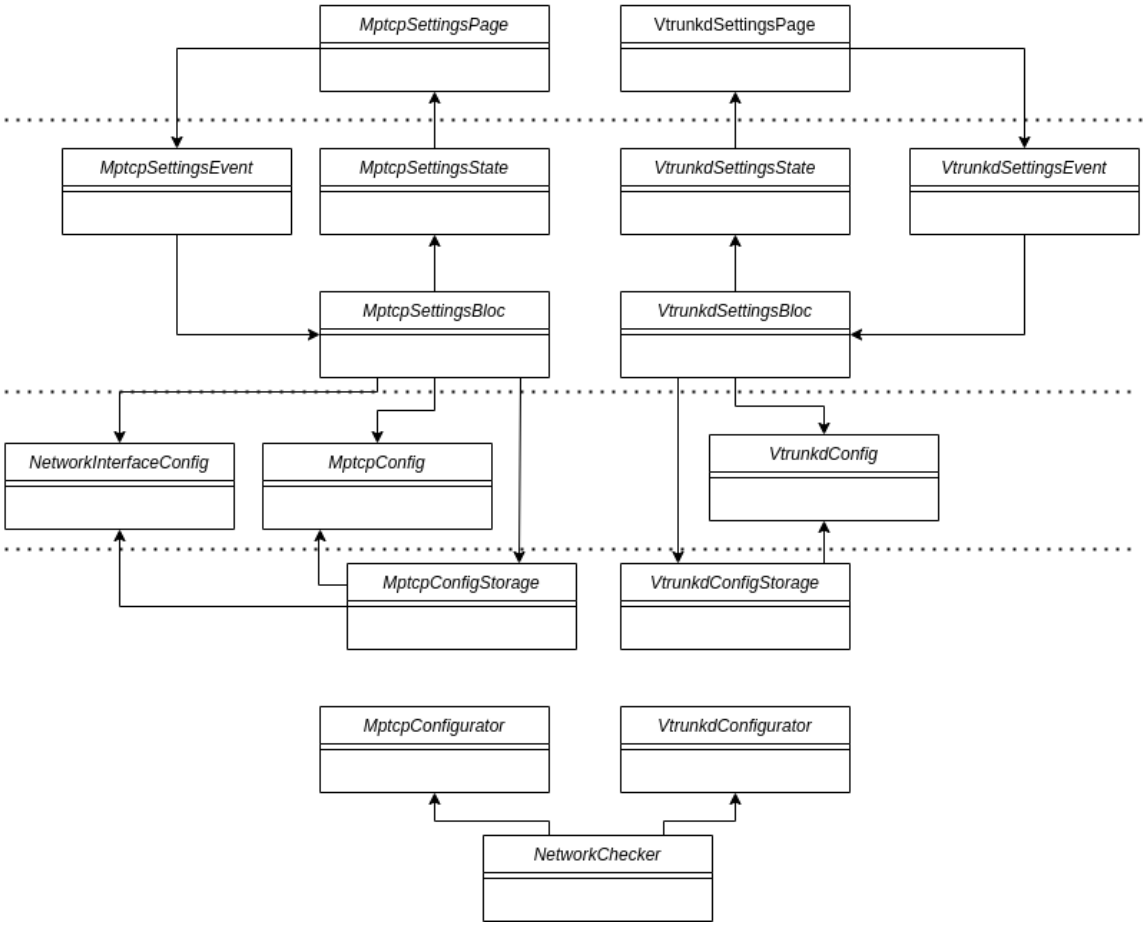
					ІАЛЦ 467100.005 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Смірнов Н. О.			Система багатоканального транспорту для віртуальних приватних мереж Структурна схема системи	Лит.	Аркуш 1 / Аркушів 1
Перевірив		Роковий О. П.					
Т. Контр.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81	
Н. Контр.		Сімоненко В. П					
Затвердив							

Додаток З

ФУНКЦІОНАЛЬНА СХЕМА КЛАСІВ

до дипломного проєкту на тему: «Система багатоканального транспорту для віртуальних приватних мереж»

Київ – 2022



					ІАЛЦ 467100.006 Д3	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Смірнов Н. О.			Система багатоканального транспорту для віртуальних приватних мереж Функціональна схема (Діаграма класів)	Лит. / Аркуш 1 / Аркушів 1
Перевірив		Роковий О. П.				
Т. Контр.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81
Н. Контр.		Сімоненко В. П				
Затвердив						

Додаток 4

ТЕКСТ ПРОГРАМИ
до дипломного проєкту на тему: «Система багатоканального транспорту для віртуальних приватних мереж»

Київ – 2022

mptcp_settings_page.dart

```
import 'dart:io';

import
'package:auto_conf/application/mptcp/mptcp_settings/mptcp_settings_bloc.
dart';
import 'package:auto_conf/domain/mptcp/network_interface_config.dart';
import
'package:auto_conf/infrastructure/mptcp/config_storage/mptcp_config_sto
rage.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';

import 'package:flutter_bloc/flutter_bloc.dart';

import '../common/component/sidebar.dart';
import '../theme/default.dart';
import '../vtrunkd/vtrunkd_settings_page.dart';

class MptcpSettingsPage extends StatelessWidget {
  const MptcpSettingsPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: BlocProvider(
        create: (_) => MptcpSettingsBloc(MptcpConfigStorage()),
        child: const MptcpSettingsView(),
      ),
    );
  }
}
```

					ІАЛЦ 467100.007 Д4			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Смірнов Н. О.			Система багатоканального транспорту для віртуальних приватних мереж Діаграма класів	Лит.	Аркуш	Аркушів
Перевірив		Роковий О. П.					1	23
Т. Контр.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81		
Н. Контр.		Сімоненко В. П						
Затвердив								

```
class MptcpSettingsView extends StatelessWidget {
  const MptcpSettingsView({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: const BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topLeft,
            end: Alignment.bottomRight,
            colors: [
              mainDarkColor,
              mainColor,
            ],
          ),
        ),
        child: Row(
          children: [
            prepareSidebar(context),
            prepareContent(context),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          if (kDebugMode) {
            print("add new network interface");
          }
          context.read<MptcpSettingsBloc>().add(
                MptcpSettingsAddNetworkInterfaceEvent(),
              );
        },
        backgroundColor: mainDarkColor,
        child: const Icon(Icons.add),
      ),
    );
  }

  Widget prepareSidebar(BuildContext context) {
```

```
    return Sidebar(
      logo: const Icon(
        Icons.logo_dev,
        size: 52,
      ),
      groups: [
        [
          IconButton(
            icon: const Icon(Icons.emoji_transportation),
            iconSize: 48,
            onPressed: () {},
          ),
          IconButton(
            icon: const Icon(Icons.vpn_lock),
            iconSize: 48,
            onPressed: () {
              Navigator.push(
                context,
                PageRouteBuilder(
                  pageBuilder: (context, TextFormFieldanimation1, animation2)
=>
                      const VtrunkdSettingsPage(),
                  transitionDuration: Duration.zero,
                  reverseTransitionDuration: Duration.zero,
                ),
              );
            },
          ),
        ]
      ],
      actionButton: IconButton(
        icon: const Icon(Icons.arrow_circle_left_rounded),
        iconSize: 48,
        onPressed: () {},
        padding: EdgeInsets.zero,
      ),
    );
  }

  Color getColor(Set<MaterialState> states) {
```

```
  const Set<MaterialState> interactiveStates = <MaterialState>{
    MaterialState.pressed,
    MaterialState.hovered,
    MaterialState.focused,
  };
  if (states.any(interactiveStates.contains)) {
    return mainColor;
  }
  return Colors.black;
}

Widget prepareContent(BuildContext context) {
  return Expanded(
    child: Padding(
      padding: const EdgeInsets.only(
        top: 30,
        right: 30,
        bottom: 30,
      ),
      child: FractionallySizedBox(
        heightFactor: 1.0,
        child: Container(
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.circular(50),
          ),
          child: Padding(
            padding: const EdgeInsets.only(
              top: 30,
              left: 50.0,
              bottom: 50.0,
              right: 50.0,
            ),
            child: BlocBuilder<MptcpSettingsBloc, MptcpSettingsState>(
              builder: (context, state) {
                if (state is MptcpSettingsInitial) {
                  context
                      .read<MptcpSettingsBloc>()
                      .add(MptcpSettingsLoadingEvent());
                  return Container();
```

```
            }
            return ListView.builder(
              itemCount: 1 + state.config.networkInterfaceConfigs.length,
              itemBuilder: (BuildContext context, int index) {
                if (index == 0) return defaultNetworkInterface(context);
                return networkInterface(
                  context,
                  state.config.networkInterfaceConfigs.values
                      .elementAt(index - 1),
                );
              },
            );
          },
        ),
      ),
    ),
  ),
 );
}

Widget onlyDefaultButton(BuildContext context) {
 return Row(
  mainAxisAlignment: MainAxisAlignment.end,
  // Only default
  children: [
   Checkbox(
    checkColor: Colors.white,
    fillColor: MaterialStateProperty.resolveWith(getColor),
    value: false,
    onChanged: (bool? value) {
     if (kDebugMode) {
      print("click");
     }
    },
   ),
   const Text('Only default'),
  ],
 );
}
```

```
Widget defaultNetworkInterface(BuildContext context) {
  return Stack(
    children: [
      Container(
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(5),
          border: Border.all(color: Colors.black),
        ),
        margin: const EdgeInsets.only(bottom: 30, top: 10),
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Row(
            children: const [
              Flexible(
                child: Padding(
                  padding: EdgeInsets.symmetric(horizontal: 16),
                  child: TextField(
                    decoration: InputDecoration(
                      hintText: 'Name',
                    ),
                  ),
                ),
              ),
              Flexible(
                child: Padding(
                  padding: EdgeInsets.symmetric(horizontal: 16),
                  child: TextField(
                    decoration: InputDecoration(
                      hintText: 'Ip address',
                    ),
                  ),
                ),
              ),
              Flexible(
                child: Padding(
                  padding: EdgeInsets.symmetric(horizontal: 16),
                  child: TextField(
                    decoration: InputDecoration(
```

```
                hintText: 'Fwmark',
              ),
            ),
          ),
        ),
        Flexible(
          child: Padding(
            padding: EdgeInsets.symmetric(horizontal: 16),
            child: TextField(
              decoration: InputDecoration(
                hintText: 'Metric',
              ),
            ),
          ),
        ),
      ],
    ),
  ),
  Positioned(
    top: 0,
    left: 20,
    child: Container(
      color: Colors.white,
      child: const Text(
        "Default Network Interface",
        style: TextStyle(
          fontWeight: FontWeight.w500,
          fontSize: 20,
        ),
      ),
    ),
  ),
 ],
 );
}

Widget networkInterface(
   BuildContext context, NetworkInterfaceConfig interfaceConfig) {
 return Stack(
```

```
children: [
  Container(
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(5),
      border: Border.all(color: Colors.black),
    ),
    margin: const EdgeInsets.only(bottom: 30, top: 10),
    child: Padding(
      padding: const EdgeInsets.all(20.0),
      child: Row(
        children: [
          Flexible(
            child: Padding(
              padding: const EdgeInsets.symmetric(horizontal: 16),
              child: TextFormField(
                initialValue: interfaceConfig.dev,
                decoration: const InputDecoration(
                  hintText: 'Device',
                ),
                onChanged: (val) {
                  print("update dev");
                  interfaceConfig.dev = val;
                  context.read<MptcpSettingsBloc>().add(
                        MptcpSettingsUpdateNetworkInterfaceEvent(
                          networkInterfaceConfig: interfaceConfig,
                        ),
                      );
                },
              ),
            ),
          ),
          Flexible(
            child: Padding(
              padding: const EdgeInsets.symmetric(horizontal: 16),
              child: TextFormField(
                initialValue: interfaceConfig.ipAddress?.address ?? "",
                decoration: const InputDecoration(
                  hintText: 'Ip address',
                ),
```

```
        onChanged: (val) {
          try {
            print("update ip address");
            interfaceConfig.ipAddress = InternetAddress(val);
            context.read<MptcpSettingsBloc>().add(
                  MptcpSettingsUpdateNetworkInterfaceEvent(
                    networkInterfaceConfig: interfaceConfig,
                  ),
                );
          } catch (e) {
            print("error");
          }
        },
      ),
    ),
  ),
  Flexible(
    child: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 16),
      child: TextFormField(
        initialValue: interfaceConfig.via?.address ?? "",
        decoration: const InputDecoration(
          hintText: 'Via',
        ),
        onChanged: (val) {
          try {
            print("update via");
            interfaceConfig.via = InternetAddress(val);
            context.read<MptcpSettingsBloc>().add(
                  MptcpSettingsUpdateNetworkInterfaceEvent(
                    networkInterfaceConfig: interfaceConfig,
                  ),
                );
          } catch (e) {
            print("error");
          }
        },
      ),
    ),
  ),
```

```
Flexible(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 16),
    child: TextFormField(
      initialValue: interfaceConfig.fwmark?.toString() ?? "",
      decoration: const InputDecoration(
        hintText: 'Fwmark',
      ),
      onChanged: (val) {
        try {
          print("update fwmark");
          interfaceConfig.fwmark = int.parse(val);
          context.read<MptcpSettingsBloc>().add(
                MptcpSettingsUpdateNetworkInterfaceEvent(
                  networkInterfaceConfig: interfaceConfig,
                ),
              );
        } catch (e) {
          print("error");
        }
      },
    ),
  ),
),
Flexible(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 16),
    child: TextFormField(
      initialValue: interfaceConfig.metric.toString(),
      decoration: const InputDecoration(
        hintText: 'Metric',
      ),
      onChanged: (val) {
        try {
          print("update metric $val");
          interfaceConfig.metric = int.parse(val);
          context.read<MptcpSettingsBloc>().add(
                MptcpSettingsUpdateNetworkInterfaceEvent(
                  networkInterfaceConfig: interfaceConfig,
                ),
```

```
                    );
                } catch (e) {
                  print("error");
                  print(e);
                }
              },
            ),
          ),
        ),
        IconButton(
          icon: const Icon(Icons.delete),
          onPressed: () {
            if (kDebugMode) {
              print("delete configuration");
            }
            context.read<MptcpSettingsBloc>().add(
                MptcpSettingsDeleteNetworkInterfaceEvent(
                  networkInterfaceConfig: interfaceConfig,
                ),
              );
          },
        ),
      ],
    ),
  ),
),
Positioned(
  top: 0,
  left: 20,
  child: Container(
    color:  Colors.white,
    child: Text(
      "Network Interface #${interfaceConfig.id}",
      style: const TextStyle(
        fontWeight: FontWeight.w500,
        fontSize: 20,
      ),
    ),
  ),
),
```

```
      ],
    );
  }
}
```

vtrunkd_settings_page.dart

```
import 'dart:io';

import
'package:auto_conf/application/vtrunkd/vtrunkd_settings/vtrunkd_settings_
bloc.dart';
import
'package:auto_conf/infrastructure/vtrunkd/config_storage/vtrunkd_config_
storage.dart';
import 'package:auto_conf/presentation/mptcp/mptcp_settings_page.dart';
import 'package:flutter/material.dart';

import 'package:flutter_bloc/flutter_bloc.dart';

import '../common/component/sidebar.dart';
import '../theme/default.dart';

class VtrunkdSettingsPage extends StatelessWidget {
  const VtrunkdSettingsPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: BlocProvider(
        create: (_) => VtrunkdSettingsBloc(VtrunkdConfigStorage()),
        child: const VtrunkdSettingsView(),
      ),
    );
  }
}
```

```
class VtrunkdSettingsView extends StatelessWidget {
 const VtrunkdSettingsView({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: Container(
    decoration: const BoxDecoration(
     gradient: LinearGradient(
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
      colors: [
       mainDarkColor,
       mainColor,
      ],
     ),
    ),
    child: Row(
     children: [
      prepareSidebar(context),
      prepareContent(context),
     ],
    ),
   ),
  );
 }

 Widget prepareSidebar(BuildContext context) {
  return Sidebar(
   logo: const Icon(
    Icons.logo_dev,
    size: 52,
   ),
   groups: [
    [
     IconButton(
      icon: const Icon(Icons.emoji_transportation),
      iconSize: 48,
      onPressed: () {
       Navigator.push(
```

```
            context,
            PageRouteBuilder(
              pageBuilder: (context, animation1, animation2) =>
                  const MptcpSettingsPage(),
              transitionDuration: Duration.zero,
              reverseTransitionDuration: Duration.zero,
            ),
          );
        },
      ),
      IconButton(
        icon: const Icon(Icons.vpn_lock),
        iconSize: 48,
        onPressed: () {},
      ),
    ]
  ],
  actionButton: IconButton(
    icon: const Icon(Icons.arrow_circle_left_rounded),
    iconSize: 48,
    onPressed: () {},
    padding: EdgeInsets.zero,
  ),
 );
}

Widget prepareContent(BuildContext context) {
 return Expanded(
  child: Padding(
    padding: const EdgeInsets.only(
      top: 30,
      right: 30,
      bottom: 30,
    ),
    child: FractionallySizedBox(
      heightFactor: 1.0,
      child: Container(
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(50),
```

```
          ),
          child: Padding(
            padding: const EdgeInsets.only(
              top: 30,
              left: 50.0,
              bottom: 50.0,
              right: 50.0,
            ),
            child: prepareVtrunkdConfigView(context)),
        ),
      ),
    ),
  );
}

Widget prepareVtrunkdConfigView(BuildContext context) {
  return BlocBuilder<VtrunkdSettingsBloc, VtrunkdSettingsState>(
    builder: (context, state) {
      return Stack(
        children: [
          Container(
            decoration: BoxDecoration(
              color: Colors.white,
              borderRadius: BorderRadius.circular(5),
              border: Border.all(color: Colors.black),
            ),
            margin: const EdgeInsets.only(top: 10),
            child: Padding(
              padding: const EdgeInsets.all(20.0),
              child: Column(
                children: [
                  Flexible(
                    child: Padding(
                      padding: const EdgeInsets.symmetric(horizontal: 16),
                      child: TextFormField(
                        initialValue: state.config.sessions.length.toString(),
                        decoration: const InputDecoration(
                          hintText: 'Sessions count',
                        ),
                        onChanged: (val) {
```

```
          try {
            print("update session count");
            state.config.sessions = List.generate(
                int.parse(val), (index) => index + 1);
            context.read<VtrunkdSettingsBloc>().add(
                  VtrunkdSettingsUpdateEvent(state.config),
                );
          } catch (e) {
            print("error");
            print(e);
          }
        },
      ),
    ),
  ),
),
Flexible(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 16),
    child: TextFormField(
      initialValue: state.config.serverIpAddress.address,
      decoration: const InputDecoration(
        hintText: 'Server ip address',
      ),
      onChanged: (val) {
        try {
          print("update server ip address");
          state.config.serverIpAddress =
              InternetAddress(val);
          context.read<VtrunkdSettingsBloc>().add(
                VtrunkdSettingsUpdateEvent(state.config),
              );
        } catch (e) {
          print("error");
          print(e);
        }
      },
    ),
  ),
),
Flexible(
```

```
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 16),
    child: TextFormField(
      initialValue: state.config.port.toString(),
      decoration: const InputDecoration(
        hintText: 'port',
      ),
      onChanged: (val) {
        try {
          print("update port");
          state.config.port = int.parse(val);
          context.read<VtrunkdSettingsBloc>().add(
              VtrunkdSettingsUpdateEvent(state.config),
            );
        } catch (e) {
          print("error");
          print(e);
        }
      },
    ),
  ),
),
Flexible(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 16),
    child: TextFormField(
      initialValue: state.config.configPath,
      decoration: const InputDecoration(
        hintText: 'configPath',
      ),
      onChanged: (val) {
        try {
          print("update config path");
          state.config.configPath = val;
          context.read<VtrunkdSettingsBloc>().add(
              VtrunkdSettingsUpdateEvent(state.config),
            );
        } catch (e) {
          print("error");
          print(e);
```

```
                    }
                  },
                ),
              ),
            ),
          ],
        ),
      ),
    ),
    Positioned(
      top: 0,
      left: 20,
      child: Container(
        color: Colors.white,
        child: const Text(
          "Vtrunkd config",
          style: TextStyle(
            fontWeight: FontWeight.w500,
            fontSize: 20,
          ),
        ),
      ),
    ),
  ],
 );
},
);
}
}
```

main.dart

```
void main() async {
  final cron = Cron();
  cron.schedule(Schedule.parse('*/10 * * * * *'), () async {
    final mptcpConfigurator = MptcpConfigurator();
    final mptcpConfig = MptcpConfigStorage().loadOrDefaultConfig();
    final vtrunkdConfigurator = VtrunkdConfigurator();
    final vtrunkdConfig = VtrunkdConfigStorage().loadOrDefaultConfig();
    final NetworkChecker networkChecker = NetworkChecker(
```

```
      mptcpConfigurator: mptcpConfigurator,
      mptcpConfig: mptcpConfig,
      vtrunkdConfigurator: vtrunkdConfigurator,
      vtrunkdConfig: vtrunkdConfig,
    );
    await networkChecker.checkInterfaces().catchError(
          (e) => print(e),
        );
  });

  await GetStorage.init();
  await GetStorage().erase();
  runApp(const MptcpSettingsPage());
}
```

mptcp_configurator.dart

```
import 'dart:io';

import 'package:auto_conf/domain/mptcp/network_interface_config.dart';
import 'package:flutter/foundation.dart';

import '../../domain/mptcp/mptcp_config.dart';

class MptcpConfigurator {
  _getGatewayAddress(String address) {
    final addressData = address.split(".");
    addressData[3] = "1";
    final routerAddress = addressData.join(".");
    return routerAddress;
  }

  NetworkInterfaceConfig _getInterfaceConfig(
    MptcpConfig config,
    NetworkInterface networkInterface,
  ) {
    for (final networkInterfaceConfig
        in config.networkInterfaceConfigs.values) {
      if (networkInterfaceConfig.ipAddress != null &&
          networkInterfaceConfig.ipAddress != networkInterface.addresses[0])
{
        continue;
```

```
      }
      if (networkInterfaceConfig.dev != networkInterface.name) {
        continue;
      }
      if (networkInterfaceConfig.index != null &&
          networkInterfaceConfig.index != networkInterface.index) {
        continue;
      }
      return networkInterfaceConfig;
    }
    return config.defaultConfig;
  }

  prepare(int tableCount) async {
    for (int i = 101; i <= 100 + tableCount; i++) {
      await Process.run("ip", ["rule", "add", "fwmark", "0x$i", "lookup",
"$i"])
          .then((ProcessResult res) {
        if (kDebugMode) {
          print(res.exitCode);
          print(res.stdout);
          print(res.stderr);
        }
      });
    }
  }

  configurate(
    MptcpConfig config,
    List<NetworkInterface> networkInterfaces,
  ) async {
    for (final networkInterface in networkInterfaces) {
      // Вибірка конфігурації для даного інтерфейсу
      final networkInterfaceConfig =
          _getInterfaceConfig(config, networkInterface);
      if (networkInterfaceConfig.dev == "") continue;
      if (networkInterfaceConfig.fwmark == null) continue;

      final gateway = networkInterfaceConfig.via ??
          _getGatewayAddress(
            networkInterface.addresses[0].address,
```

```
        );

      final tableNumber = networkInterfaceConfig.fwmark;

      await Process.run("ip", [
        "route",
        "add",
        "default",
        "via",
        gateway,
        "dev",
        networkInterfaceConfig.dev,
        "table",
        "$tableNumber",
        "metric",
        "${networkInterfaceConfig.metric}"
      ]).then((ProcessResult res) {
        if (kDebugMode) {
          print(res.exitCode);
          print(res.stdout);
          print(res.stderr);
        }
      });
    }
  }
}
```

vtrunkd_configurator.dart

```
import 'dart:io';

import 'package:auto_conf/domain/vtrunkd/vtrunkd_config.dart';
import 'package:flutter/foundation.dart';

class VtrunkdConfigurator {
  configurate(
      VtrunkdConfig config, List<NetworkInterface> networkInterfaces)
async {
    for (final sessionNumber in config.sessions) {
      await Process.run("vtrunkd", [
        "-P",
```

```
        "${config.port}",
        "-f",
        config.configPath,
        "000000_$sessionNumber",
        "${config.serverIpAddress}",
      ]).then((ProcessResult res) {
        if (kDebugMode) {
          print(res.exitCode);
          print(res.stdout);
          print(res.stderr);
        }
      });
    }
  }
}
```

network_checker.dart

```
import 'dart:io';

import 'package:auto_conf/domain/mptcp/mptcp_config.dart';
import 'package:auto_conf/domain/vtrunkd/vtrunkd_config.dart';
import 'package:auto_conf/infrastructure/mptcp/mptcp_configurator.dart';
import
'package:auto_conf/infrastructure/vtrunkd/vtrunkd_configurator.dart';

class NetworkChecker {
  final MptcpConfigurator mptcpConfigurator;
  final MptcpConfig mptcpConfig;
  final VtrunkdConfigurator vtrunkdConfigurator;
  final VtrunkdConfig vtrunkdConfig;

  final Map<String, int> _configuredDevices = {};
  static const tablesCount = 10;
  final List<int> tableUnused = List.generate(tablesCount, (i) => i + 101);

  NetworkChecker({
    required this.mptcpConfigurator,
    required this.mptcpConfig,
    required this.vtrunkdConfigurator,
    required this.vtrunkdConfig,
```

```
  });

  Future<List<NetworkInterface>> getInrefaces() async {
    final interfaces = await NetworkInterface.list();
    return interfaces;
1 }

  Future checkInterfaces() async {
    await mptcpConfigurator.prepare(tablesCount);
    // take current interfaces
    var newInterfaces = await getInrefaces();

    // Remove existed devices
    final removedDevices = Map<String, int>.from(_configuredDevices);
    removedDevices.removeWhere(
      (dev, _) => newInterfaces.any((interface) => interface.name == dev),
    );

    newInterfaces = newInterfaces
        .where(
          (interface) => !_configuredDevices.containsKey(interface.name),
        )
        .toList();

    for (final dev in removedDevices.keys) {
      tableUnused.add(removedDevices[dev]!);
      _configuredDevices.remove(dev);
    }

    await mptcpConfigurator.configurate(
      mptcpConfig,
      newInterfaces,
    );
    await vtrunkdConfigurator.configurate(
      vtrunkdConfig,
      newInterfaces,
    );
  }
}
```