НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Програмно-апаратне забезпечення для Edge серверів для систем IoT»

Виконав:
студент ІІ курсу, групи ІО-21мп
Сірий Іван Миколайович _____

Керівник:
Професор кафедри ОТ, д.т.н.,
доцент Клименко Ірина Анатоліївна _____

Консультант з нормоконтролю:
проф., д.т.н., проф.,
Кулаков Юрій Олексійович _____

Рецензент:
Професор кафедри ІСТ, д.т.н., проф.
Корнієнко Богдан Ярославович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Освітньо-професійна програма *Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
«_____» _____ 2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Сірому Івану Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Програмно-апаратне забезпечення для Edge серверів для систем IoT.

Науковий керівник дисертації Клименко Ірина Анатоліївна, доцент, доктор технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «7» листопада 2023 р. № 5168-с

2. Строк подання студентом дисертації 25.12.2023

3. Об'єкт дослідження серверна складова Edge для систем IoT.

4. Предмет дослідження процес розробки програмно-апаратного забезпечення для Edge сервера

5. Перелік завдань, які потрібно розробити: узагальнення та аналіз систем IoT з використанням Edge серверів, підготовка до сокетного програмування для віддалених Edge серверів відмінної архітектури, розробка клієнт-серверних застосунків для Edge серверів для моніторингу доступних WiFi мереж.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Кулаков Ю.О		

7. Дата видачі завдання 1.09.2023

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	Затвердження теми дослідження. Визначення предмету дослідження	1.09.23-7.09.23	
2.	Дослідження систем IoT з використанням Edge серверів	8.09.23-15.10.23	
3.	Дослідження необхідних типів сокетів	16.10.23-28.10.23	
4.	Розробка базового та розширеного TCP клієнт-серверного застосунку	29.10.23-05.11.23	
5.	Розробка HTTP сервера	06.11.23-19.11.23	
5.	Розробка WiFi сканера мереж	20.11.23-09.12.23	
6.	Тестування та аналіз ефективності.	10.12.23-13.12.23	
7.	Розробка стартап-проекту	14.12.23-18.12.2	
8.	Оформлення магістерської дисертації.	19.12.23-25.12.23	

Студент Іван СІРИЙ _____
(підпис)

Науковий керівник дисертації Ірина КЛИМЕНКО _____
(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Програмно-апаратне забезпечення для Edge серверів для систем IoT.

студентом: Сірим Іваном Миколайовичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 98 аркушів основного тексту, 42 ілюстрації, 23 таблиці. При підготовці використовувалася література з 24 різних джерел.

Мета дослідження. Мета дослідження полягає у розробці та оптимізації IoT систем з використанням Edge серверів для підвищення ефективності обробки даних та забезпечення безпеки в сучасних IoT мережах. Ця мета є відповіддю на проблематику забезпечення високої швидкості обробки великих обсягів даних та вирішення безпекових викликів в IoT мережах.

Предметом дослідження є процеси реалізації та застосування вбудованих систем на рівні Edge обчислень для систем IoT, які породжують проблеми пов'язані з обробкою великих обсягів даних, забезпеченням низької затримки та підвищенням рівня безпеки.

Об’єктом дослідження є методи та технології реалізації вбудованих систем на рівні Edge обчислень для систем IoT, які спрямовані на вирішення проблем обробки даних, забезпечення низької затримки та підвищення рівня безпеки.

Для досягнення поставленої мети магістерської дисертації були сформульовані наступні завдання дослідження:

1. Провести аналіз типів сокетів, які слугують для передачі цілісних даних у вигляді файлів, а також тих, які дозволяють комунікувати програмам середовища користувача із середовищем ядра операційної системи з метою визначення проблематики і постановки завдань дослідження.
2. Розробити методику сокетного програмування для Edge серверів для систем IoT, яка дозволить поглибити знання предметної області та підвищить ефективність розроблення Embedded систем.

3. Розробити клієнт-серверний застосунок для обміну буферизованими текстовими даними та файлами.
4. Розробити сервер для відправки даних у локальній мережі з доступом за допомогою браузера.
5. Розробити WiFi сканер мереж, який дозволяє отримувати дані про стан мереж їх безпекові та фізичні характеристики та представити надані ним результати пристроям в мережі за допомогою сервера.
6. Перевірити ефективність та відповідність системи поставленим вимогам.

Наукова новизна

1. Розроблена методика сокетного програмування для Edge серверів для розширення знань предметної області та підвищення ефективності розроблення Embedded систем для систем IoT.

Практична цінність

1. Розроблені методи та технології можуть бути використані в різних сферах, де застосовуються IoT системи, зокрема у смарт-містах, промисловості, охороні здоров'я та інших галузях для підвищення ефективності обробки даних та забезпечення безпеки. Застосування цих технологій сприятиме розвитку сучасних IoT систем, підвищуючи їх продуктивність та надійність.
2. Запропонована методика сокетного програмування для Edge серверів може бути використана в навчальному процесі для розширення знань та надбання практичних навичок в області розробки вбудованих систем.

Ключові слова

Edge Computing, IoT, WiFi сканування, TCP клієнт-сервер, HTTP сервер, віддалене завантаження, NFS, TFTP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9
РОЗДІЛ 1. СУЧАСНІ ІОТ СИСТЕМИ З ВИКОРИСТАННЯМ EDGE СЕРВЕРІВ 10
1.1 Інтернет речей як складова нової промислової революції 10
1.2 IoT та Edge для смарт міст 13
1.3 IoT та Edge в автомобільній промисловості 14
1.4 Безпекові виклики IoT та Edge 17
Висновки до розділу 1 19
РОЗДІЛ 2. ОГЛЯД МАТЕРІАЛІВ ТА МЕТОДІВ 20
2.1 Клієнт-серверна архітектура. Програмування сокетів 21
2.1.1 Клієнт-серверна архітектура 21
2.1.2 TCP сокети 22
2.1.3 Netlink сокети 23
2.2 Розробка програмного забезпечення для плати BeagleBone Black 24
2.2.1 Крос-компіляція 24
2.2.2 Завантаження TFTP 25
2.2.2.1 Конфігурація хоста 28
2.2.2.2 Запуск завантаження TFTP на пристрої 31
2.2.2.3 Завантаження NFS 32
2.2.2.4 Запуск завантаження TFTP із rootfs NFS 33
Висновки до розділу 2 35
РОЗДІЛ 3. ОПИС ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ EDGE СЕРВЕРА 36
3.1 Знайомство з сокетами 36
3.1.1 TCP з’єднання 36
3.1.2 TCP сервер 39
3.1.3 TCP клієнт 49
3.1.4 Підключення клієнта до сервера 51
3.1.5 Передача файлів 53

3.1.6 HTTP сервер 59
3.1.7 Сканер WiFi 63
3.1.8 Відображення отриманих даних через HTTP сервер 68
Висновки до розділу 3 70
РОЗДІЛ 4. РОЗРОБКА СТАРТАП-ПРОЕКТУ 71
4.1 Маркетинговий аналіз стартап-проекту 71
4.2 Технологічний аудит ідеї проекту 74
4.3 Аналіз ринкових можливостей запуску стартап-проекту 74
4.4 Розробка ринкової стратегії стартап-проекту 84
4.5 Розробка маркетингової програми стартап-проекту 87
Висновки до розділу 4 92
ВИСНОВКИ 93
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 95

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API	(Address Resolution Protocol) Прикладний програмний інтерфейс
BSD	(Berkeley Software Distribution) Дистрибутив програм Берклі
DP	(Differential Privacy) розподілена приватність
HTTP	(Hyper-text Transfer Protocol)
ICNMEC	(Information-centric Networking System using Multiaccess Edge Computing) Інформаційно-орієнтована мережева система з використанням граничних обчислень
IIoT	(Industrial Internet of Things) Промисловий Інтернет речей
IoT	(Internet of Things) Інтернет речей
IoV	(Internet of Vehicles) Інтернет транспортних засобів
IP	(Internet Protocol) Інтернет протокол
NFS	(Network File System) Мережева файлова система
OMNM	(Optimized Memory Network Management) Оптимізоване керування мережею пам’яті
TCP	(Transmission Control Protocol) Протокол контролю передачі
TFTP	(Trivial File Transfer Protocol) Тривіальний протокол передачі файлів
UDP	(User Datagram Protocol) Протокол датаграм користувача

ВСТУП

У сучасному світі швидко зростає потреба в ефективних та інтелектуальних системах, які можуть швидко обробляти великі обсяги даних, забезпечуючи надійність та швидкодію. Це особливо актуально у сфері Інтернету речей (IoT), де мільярди пристроїв постійно збирають та передають дані. Впровадження Edge серверів в IoT системи стає ключовим елементом для підвищення ефективності та зниження затримок, що особливо важливо в умовах розвитку малого та великого бізнесу, промисловості, держави загалом.

Передача даних від множини пристроїв IoT до централізованих хмарних систем може призвести до значного збільшення трафіку та затримок, що негативно впливає на продуктивність системи. Edge сервери, розміщені ближче до джерела даних, дозволяють обробляти інформацію на місці, тим самим знижуючи затримки та забезпечуючи більш швидку реакцію системи.

У даній роботі ми розглядаємо різні аспекти інтеграції Edge серверів в IoT системи, зосереджуючись на таких ключових темах, як вплив IoT на промисловість та смарт-міста, безпекові виклики в IoT та Edge, а також на розробці програмного забезпечення для Edge серверів. Особлива увага приділяється програмуванню сокетів, віддаленому доступу, налаштуванню Edge сервера, яким виступає плата BeagleBoneBlack.

Робота покликана надати глибоке розуміння поточного стану справ в області IoT та Edge обчислень, а також висвітлити перспективи та виклики, які постають перед дослідниками та розробниками в галузі, що швидко розвивається.

РОЗДІЛ 1
СУЧАСНІ ІОТ СИСТЕМИ З ВИКОРИСТАННЯМ EDGE СЕРВЕРІВ

1.1 Інтернет речей як складова нової промислової революції

В 2011 році взяла свій початок четверта промислова революція. Докорінні зміни нашого теперішнього та майбутнього визначатимуться впровадженням різноманітних «розумних» кіберфізичних систем у промислове виробництво, у сферу обслуговування потреб людини, її побуту, праці і дозвілля.

Такі системи працюватимуть спільно, складатимуть одну мережу, підлаштовуватимуться під особливості їхнього співіснування з навколишнім середовищем та людиною. Обмін даними в реальному часі між компонентами системи чи між підсистемами якнайрізноманітніших масштабів дозволять і прагнутимуть до підвищення ефективності людського побуту, виробництва, державного і бізнес управління, сфери охорони здоров'я. Такі зміни слугуватимуть пришвидшенню розвитку, сприятимуть допуску меншої кількості помилок, більш глибокому аналізу через кількість та різноманіття даних, швидкій адаптації до змін чи нових умов.

Складовою четвертої промислової революції є поняття Інтернету речей (Internet of Things, IoT). Інтернет речей означає мережу фізичних пристроїв, у які вбудовано датчики, які мають програмне забезпечення та підключення до мережі, що дозволяє їм збирати та обмінюватися даними [1].

Також, виділяється поняття про промисловий Інтернет речей (Industrial Internet of Things, IIoT). Промисловий Інтернет речей — це система об'єднаних комп'ютерних мереж і підключених до них промислових (виробничих) об'єктів з вбудованими датчиками і програмним забезпеченням для збору та обміну даними, з можливістю віддаленого контролю і управління в автоматизованому режимі, без участі людини [2].

Ключові зміни неминуче торкнуться різноманітних сфер життя, політичних систем країн світу, ринку праці, вплинуть на життєве середовище та людську ідентичність. На основі попереднього аналізу можливого

прогнозованого впливу цього етапу розвитку людства, задля надання певних маяків для різних ланок суспільства, вченими, політиками, бізнесменами в тісній співпраці було сформульовано основні принципи етапу, названого поняттям «Індустрія 4.0», впровадженого на Hannover Messe в 2011 році, та вперше відкрито озвученого професором Вольфгангом Вальстером, генеральним директором Німецького дослідницького центру штучного інтелекту, під час промови на відкритті заходу [3].

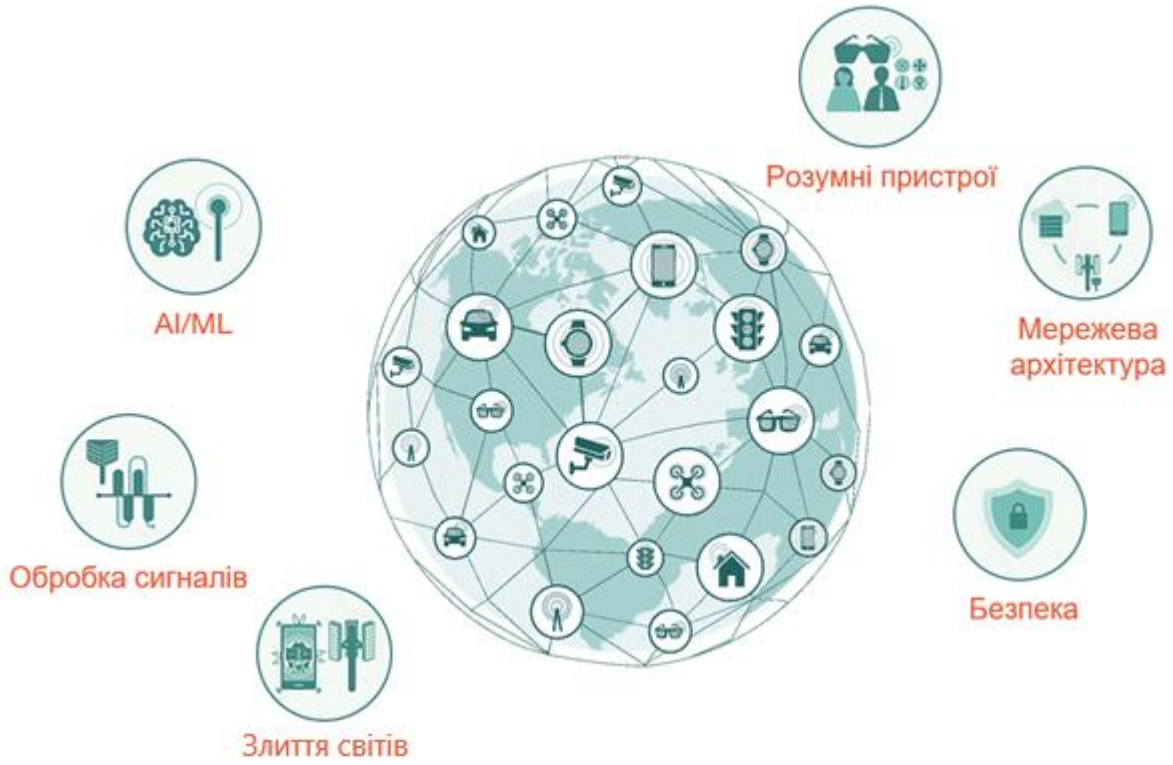

Рисунок 1.1 – Світ з IoT

Сумісність, прозорість, технічна підтримка та деталізація управлінських рішень стали чотирма стовпами, що були представлені робочою групою під час заходу в Гановері:

- сенсори, пристрої, машини в більш широкому розумінні, мають взаємодіяти один з одним та людиною, будуючи спільне середовище, цілий світ, як показано на рисунку 1.1;
- в результаті взаємодії генеруватимуться, зберігатимуться та накопичуватимуться дані, які формують вичерпну інформацію, яку,

використовуючи збережений контекст подій, варто та необхідно досліджувати для вдосконалення та виправлення неточностей;

- побудовані системи допомагатимуть людям в прийнятті рішень, правильність яких базуватиметься на зібраних та проаналізованих даних;
- частково чи повністю, рутинні та небезпечні заняття виконуватимуться машинами, що в принципі є базовим призначенням промислових революцій, та якщо глибше підходити до їх суті, й винаходів загалом;
- людина має виконувати контрольну роль, а не виробничу всюди, де автоматизація призведе до покращення ефективності, швидкості та якості виконання роботи. Лише екстрені випадки, які неможливо покрити автоматизацією виробничих операцій потребуватимуть людського втручання в процес.

Доктор Клаус Шваб, — відомий швейцарський економіст, засновник Всесвітнього економічного форуму, заявив: «Четверта революція йде на нас, як цунамі! Швидкість цієї революції така висока, що політичній спільноті важко або навіть неможливо встигати з необхідними нормативними та законодавчими рамками. ... Подібного масштабу і складності змін людству ще ніколи не доводилося відчувати. Вже зараз очевидно, що вона торкнеться всіх груп, шарів і прошарків людства і всіх професій» [4].

Для досягнення та реалізації ефективності та цінностей Інтернету речей, необхідно перемістити обробку для даних програм із ядра хмари (cloud core) на пристрої, що знаходяться на місцях (Edge). Ця радикальна трансформація сприятиме підтримці трильйонів датчиків і мільярдів систем.

Сучасні тенденції в сферах хмарних обчислень, побудови мереж, Інтернету речей та інших областях вимагають якнайменшої затримки в спілкуванні між складовими таких систем а також прийнятті рішень, які базуються на отриманих при спілкуванні даних. До таких сфер належать

інтелектуальне виробництво, безпека, автомобільна промисловість, інтелектуальні міські системи (Смарт-сіті), тощо.

1.2 IoT та Edge для смарт міст

За поняттям розумного міста (Smart City) одразу очевидно, — системі доведеться мати справу в величезними об’ємами даних, згенерованими різноманітними датчиками та пристроями, які в теорії мали би бути однорідними. Тоді як на практиці, коли використовуються найдоступніші методи та всі можливі (наявні) датчики, пристрої, така інформація окрім збору і обробки, ще й, скоріше за все, потребуватиме попередньої стандартизації перед її обробкою та подальшими операціями.

Обчислювальні ресурси та дата-центри мають бути гнучкими та ефективними аби впоратись із поставленим завданням. В цей же час сама монополія цих, хоча й розгалужених, ресурсів створює високі затримки в обчисленнях. Подолання цих перешкод можливе завдяки веденню обчислень, які необхідні «тут і зараз», на Edge периферійних пристроях.

Система ICNMEC (інформаційно-орієнтована мережева система з використанням граничних обчислень, Information-centric Networking System using Multiaccess Edge Computing), при використанні Edge computing з підтримкою 5G та використанням алгоритмів OMNM (оптимізоване керування мережею пам’яті, Optimized Memory Network Management) дозволяє покращити коефіцієнт ефективності на 95,141%, коефіцієнт використання сховища на 60,1% і швидкість доступу на 0,9, зменшивши мережевий трафік і затримку на 0,6 [5].

У середовищі розумного міста з підтримкою Інтернету речей однією з головних проблем є споживання найменшої кількості енергії під час виконання завдань, які задовольняють обмеження затримки [6].

Розумне місто — це система, яка використовує технологічні засоби для створення, впровадження та просування екологічно чистих рішень проблем, спричинених зростаючою урбанізацією [7].

Прийнято вважати, що впровадження рішень розумного міста на відповідальних та обґрунтованих засадах сприяє досягненню цілей сталого розвитку (ЦСР). До яких належать економічне зростання, інноваційний бум, підвищення ефективності праці та виробництв, зменшення викидів, зменшення впливу на кліматичні зрушення, підвищення обізнаності громадян. Але таке твердження є певним упередженням, бо для цього потрібно знайти компроміси, пов’язані з такими проблемами, як конфіденційність і кібербезпека, витрати на модернізацію інфраструктури, негативний соціальний вплив, цифровий розрив і відсутність навичок, неправильне використання штучного інтелекту [8].

1.3 IoT та Edge в автомобільній промисловості

Як конкретний приклад можна представити побудову системи як локального — місцевого, так і національного, або ж навіть транс-національного дорожнього руху. Всі учасники цього руху, і автомобілі, і пішоходи, і користувачі персонального транспорту матимуть доступ до актуальних даних, що стосуватимуться їх, та їхнього найближчого майбутнього на їхньому маршруті. Система, як на рисунку 1.2 аналізуватиме дані всіх учасників, займатиметься комунікацією між ними та впливатиме на них та дорожню інфраструктуру, задля того, щоб надати учасникам руху можливості для максимально ефективного використання як часових, так і енергетичних ресурсів, які витрачаються під час руху.

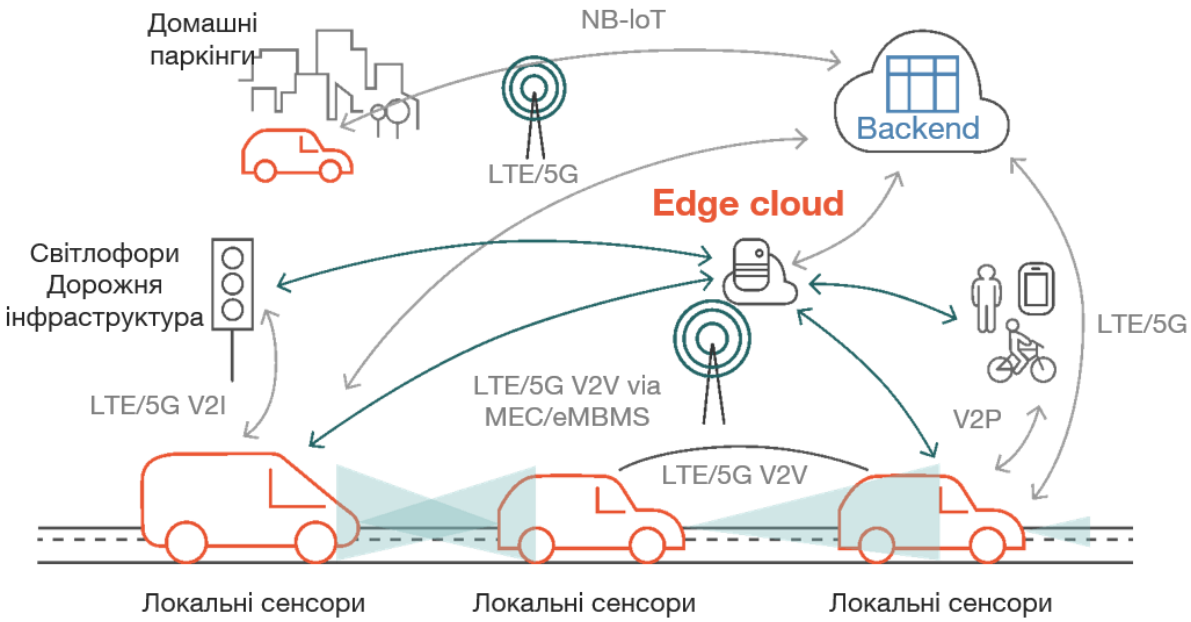

Рисунок 1.2 – Схематичне зображення системи обміну інформацією між учасниками дорожнього руху [9].

За допомогою таких систем вирішуються різноманітні задачі, одна з яких — надання пріоритету проїзду перехресть транспортним засобам екстреної допомоги. Існуюча система, якщо і де вона застосовується, викликає плутанину серед учасників руху, через негайну подачу зеленого сигналу. В експерименті [10] запропонований оптимальний алгоритм з використанням датчиків GPS та Edge сервера, який приймає дані та обчислює на їх основі оптимальний момент для ефективного регулювання дорожнього руху. В результаті чого встановлено, що середній час очікування транспортних засобів, що не мають пріоритету на перехресті зменшено на 73,23%, а з використанням обчислень на Edge сервері, затримка зв'язку в побудованій системі (датчики-сервер-світлофор) було зменшено до менше ніж 100 мс.

Концепція IoV (Інтернету транспортних засобів, Internet of Vehicles) при використанні лише хмарних обчислень втрачає свою мобільність, отримує затримки в часі, які в такій сфері несуть проблеми безпеки життя та здоров'я. Також, ситуація на дорозі змінюється динамічно — різноманітні небезпеки пов'язані із пішоходами, велосипедистами, водіями інших транспортних

засобів трапляються доволі часто і не можуть бути наперед повні прорахованими. Автомобілі можуть об’єднуватись в групи, виконуючи локальні завдання, для зменшення навантаження на мережу завдяки multicast відправкам пакетів.

Для ефективного використання ресурсів в такій динамічній системі, необхідно використовувати стратегію розумного, динамічного розподілу задач між терміналами, їхньої співпраці один із одним, стикаючись з різними викликами, як пропускною здатністю терміналів, їхньою безпекою, пріоритетом вирішення задач. Також, через динамічну зміну ситуації, мережева топологія має бути гнучкою, аби уникнути проблем із невчасним вирішенням важливих задач, через проблеми із маршрутизацією чи затримками передачі даних.

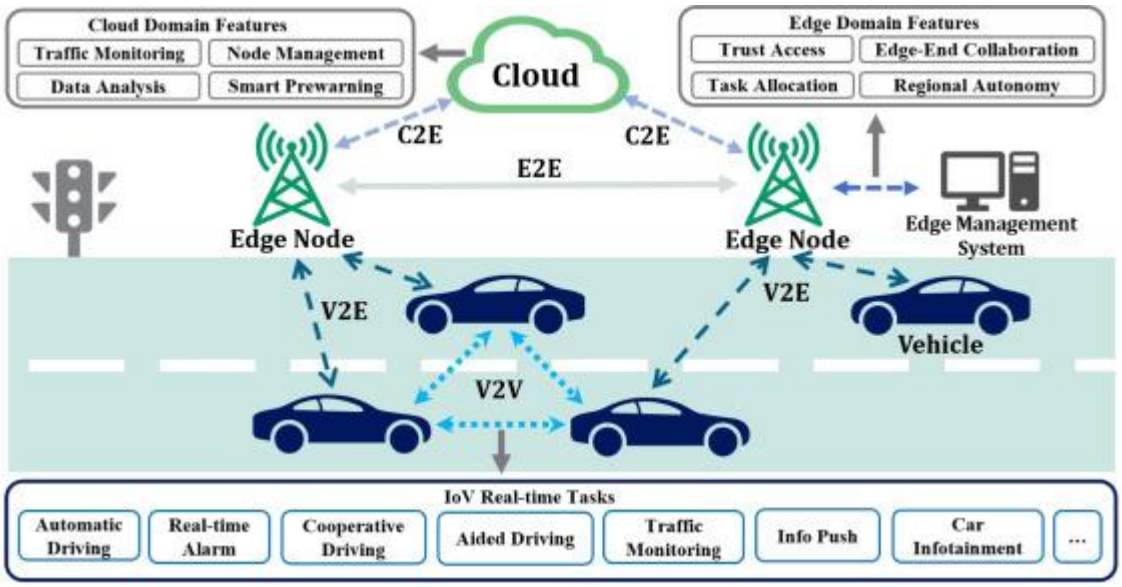

Рисунок 1.3 – Трирівнева взаємодія в IoV [11].

Зображена на рисунку 1.3 трирівнева архітектура взаємодії між терміналами, Edge та хмарою, порівняно з чистою архітектурою периферійних обчислень, може ефективніше обробляти різні завдання, а розподіл завдань може бути більш цілеспрямованим. Результати моделювання показують, що запропонована стратегія має кращу корисність для розподілу завдань, ніж інші стратегії, які можуть ефективно вирішити проблему розподілу кількох завдань [11].

1.4 Безпекові виклики IoT та Edge

Розглянуті в попередніх розділах системи породжують багато безпекових викликів. Через обмеження ресурсів, в IoT системах, а також і через погану безпеку системи вцілому, її дизайну виникає загроза для даних та самих фізичних. Застосування проміжної ланки у периферійних обчислень можуть запропонувати нові можливості для розробки та розгортання нових рішень безпеки для IoT.

В статті [12] представлено схему агрегації даних зі збереженням конфіденційності (Lightweight Privacy-preserving Data Aggregation, LPDA). У цій схемі пристрої IoT, спілкуючись із Edge серверами, повідомляють про локально оброблені ними дані датчиків разом із кодом автентифікації повідомлення (Message Authentication Code, MAC). В свою чергу периферійні сервери проводять аутентифікацію пристроїв IoT за наданими значеннями. Використовуючи гомоморфне шифрування Пайє [13], китайську теорему про залишки [14] і методи одностороннього хешування, така схема може стати вирішенням проблем із кількістю зв'язків, захистом від помилкових данихнадати додаткові можливості агрегування гібридних даних IoT [12].

Програми розумного міста, які набувають стрімкого поширення та всебічного розвитку в області обслуговування, логістики, охорони здоров'я генерують великі об'єми даних, які широко поширюються в Інтернеті. Характер інформації, яку зберігають такі системи є конфіденційним а, отже, потребує збереження цілісності та недоторканості.

Цього можна досягти, наприклад, за допомогою використання алгоритмів, що базуються на засадах DP (розподіленої приватності, Differential Privacy), що дозволяє отримувати точні запити до бази даних, мінімізуючи ризик ідентифікації окремих записів [15]. Перевагою використання такого підходу зпоміж інших є захист кожного окремого запису в базу, а не всього набору [16]. Проведені досліди [17] демонструють, що використання периферійних обчислень може надати і інші переваги, окрім

захисту конфіденційності даних (використовуючи наприклад Blockchain технології), включаючи підвищення енергоефективності і покращену обчислювальну потужність.

Висновки до розділу 1

У першому розділі було розглянуто поняття четвертої промислової революції, однією з основ якої є розвиток та поширення IoT систем, принципи, на яких вона базується, засади, яких варто дотримуватись державі, бізнесу та індивідам аби бути успішними учасниками процесу розбудови індустрії в цілому.

Зокрема, було розглянуто декілька сфер людського життя, а саме смарт міста, автомобільна промисловість та безпекові виклики, як складові, які вже на сьогоднішній день піддаються кардинальним змінам, покращують свою ефективність, прагнуть до вирішення раніше неможливих задач.

Проаналізовано досліди проведені провідними науковцями світу, щодо аспектів реалізації таких систем, покращення ефективності вирішення поставлених завдань. Приведені конкретні результати, що свідчать про ефективність використання гнучкого підходу до конструювання подібних систем, а саме використання проміжного шару обчислювальних потужностей у вигляді периферійних Edge серверів, які наближені до різноманітних датчиків IoT систем.

РОЗДІЛ 2

ОГЛЯД МАТЕРІАЛІВ ТА МЕТОДІВ ДОСЛІДЖЕННЯ

У цьому розділі ми зосереджуємо увагу на об'єкті дослідження — програмно-апаратному забезпеченні Edge серверів для систем IoT. Сучасний світ технологій постійно розвивається, і в цьому контексті, Edge обчислення відіграють ключову роль у забезпеченні ефективності IoT систем. Наше дослідження базується на гіпотезі, що інтеграція і оптимізація програмно-апаратних компонентів Edge серверів може значно підвищити їх продуктивність та надійність.

Важливим елементом дослідження є розробка базового сервера та клієнта, що дозволяє встановлювати з'єднання та передавати вітальні повідомлення. Цей етап закладає фундамент для подальшого розширення функціональності системи. Одним із ключових напрямків розвитку є імплементація можливості передачі файлів, що розширює можливості сервера з простого обміну повідомленнями до обробки більш складних даних.

На наступному етапі, серверна частина програмного забезпечення трансформується у HTTP сервер, що відкриває нові горизонти для взаємодії з клієнтськими програмами, включаючи веб-браузери. Це розширення має на увазі створення більш гнучкої та доступної платформи для кінцевих користувачів.

Специфічним завданням, у рамках дослідження, є розробка програмного забезпечення для сканування доступних WiFi мереж. В загальному розумінні, сокети є способом комунікації в мережі, «спілкування» процесів в межах середовища користувача (user space) однієї машини, але також вони дозволяють пов’язати середовище ядра (kernel space) із користувацькими програмами. В даному випадку використання сокетів для налагодження спілкування між ядром операційної системи Linux та програмами користувача дозволяє збирати дані про характеристики оточуючих девайс мереж.

Останній етап включає інтеграцію розробленого HTTP сервера та програмного забезпечення для сканування, що дає змогу користувачам переглядати результати сканування WiFi мереж через веб-браузер. Це забезпечує високий рівень доступності та зручності для кінцевого користувача.

Важливо відзначити, що у дослідженні прийнято ряд спрощень для зосередження на ключових аспектах кожного компоненту. Це дозволяє уникнути надмірної складності для спрощення розуміння процесів, розширення різноманітності поставлених задач.

2.1 Клієнт-серверна архітектура. Програмування сокетів

2.1.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура є основоположною моделлю в розподілених системах і мережевих застосунках. Саме завдяки такому підходу відбувається комунікація між пристроями в мережі, обмін даними. В цьому контексті, сервер — це програма, яка запущена на пристрої, або, якщо абстрагуватись, сам пристрій, який надає ресурси, дані, послуги або програми клієнтам, які їх запитують. При чому сервер може не володіти ресурсами на своїй машині, а бути лише їхнім постачальником. Клієнти зазвичай підключаються до сервера для отримання або використання цих ресурсів. Ця архітектура дозволяє централізувати функції та послуги, що спрощує управління і масштабування систем. Пристрій може виступати як клієнтом так і сервером, в залежності від обставин та задач, які перед ним стоять.

Якщо ми детально розглянемо модель клієнт-сервер, то побачимо, що задіяно два процеси (тобто запущені програми): один на клієнтській машині, а інший на серверній. Комунікація приймає форму клієнтського процесу, який надсилає повідомлення через мережу серверному процесу. Потім процес клієнта чекає на повідомлення відповіді. Коли серверний процес отримує запит, він виконує потрібну роботу або шукає запитані дані та надсилає відповідь [18].

Аби клієнт мав змогу отримати доступ до сервера, а сервер — відправити відповідні дані клієнту, вони мають бути приєднані один до одного на момент обміну інформацією. При побудові такої архітектури в мешах домашньої мережі, а при використанні домашнього роутера вона вже застосовується (пристрої в домашній мережі спілкуються з роутером як із сервером для надання їм необхідної інформації, наприклад надання конфігурації, роблячи запит до DHCP сервера, що запущений на роутері) ми маємо змогу фізично бачити це з'єднання. При віддаленому доступі використовується та сама модель, де віддалений веб-сервер є сервером, а персональний комп'ютер користувача – клієнтом. При загальному підході, один сервер має витримувати навантаження з обслуговування сотен, тисяч, або й більше клієнтів одночасно.

2.1.2 TCP сокети

TCP (Transmission Control Protocol) сокети – це кінцеві точки для відправки та отримання пакетів, які гарантують доставку і цілісність даних у мережах TCP/IP. Вони грають ключову роль у забезпеченні надійності та послідовності в передачі даних у мережі. Ця надійність особливо важлива, коли мова йде про передачу файлів або даних, які не можуть бути втрачені або пошкоджені без вагомих наслідків.

UDP (User Datagram Protocol), на відміну від TCP, не гарантує доставку пакетів, порядок їх надходження або навіть цілісність даних. Це робить UDP швидшим і ефективнішим за TCP у випадках, де висока швидкість передачі даних є більш важливою, ніж їх надійність. Наприклад, UDP та протоколи, які базуються на ньому часто використовується для потокового відео, аудіо, онлайн-ігор, де затримки в передачі даних більш мають більше вагу, аніж ризик втрати деяких даних.

Однак, коли мова йде про передачу файлів або необхідність забезпечення точної та надійної доставки веб-запитів, TCP є безсумнівним вибором. Наприклад, веб-сервери, які використовують HTTP або HTTPS

протоколи, зазвичай покладаються на TCP через його здатність гарантувати доставку даних без помилок і в правильному порядку. Це особливо важливо для таких операцій, як наприклад моніторинг стану систем, чи їх віддалене управління, де надійність та точність передачі даних є критичною.

2.1.3 Netlink сокети

Netlink сокети є спеціалізованим механізмом зв'язку, що використовується для передачі інформації між ядром і простором користувача у ОС Linux. Вони дозволяють реалізувати різні мережеві задачі, такі як управління маршрутизацією, моніторинг трафіку та інші функції системного адміністрування. Netlink сокети підтримують повідомлення з обмеженою структурою, що робить їх гнучкими для реалізації різних протоколів на рівні ядра.

Netlink був доданий Аланом Коксом під час розробки ядра Linux версії 1.3 як інтерфейс драйвера символьного пристрою, щоб забезпечити кілька двосторонніх комунікаційних зв'язків між ядром і середовищем користувача. Пізніше, Олексій Кузнєцов розширив його можливості під час розробки ядра Linux 2.1, надавши гнучкий та розширюваний інтерфейс обміну повідомленнями для нової передової інфраструктури маршрутизації. З тих пір сокети Netlink стали одним з основних інтерфейсів, які підсистеми ядра надають застосункам середовища користувача [19].

Netlink слідує тому ж принципу розвитку, що й Linux (цитуючи Лінуса Торвальдса: "Linux розвивається еволюційно, а не завдяки інтелектуальному проектуванню"). Це означає, що не існує повної специфікації або проектних документів Netlink, крім вихідного коду.

Netlink - це система обміну повідомленнями, орієнтована на датаграми, яка дозволяє передавати повідомлення від ядра до користувацького простору і навпаки. Вона також може використовуватися як система міжпроцесного спілкування (IPC). У цій роботі ми зосереджуємося на комунікації між ядром і користувацьким простором, який дозволяє здійснювати спілкування між

двома процесами користувацького простору або навіть між двома різними підсистемами ядра.

Netlink реалізований на базі загальної інфраструктури BSD сокетів, тому підтримує звичайні примітиви, такі як socket(), bind(), sendmsg() та recvmsg(), а також загальні механізми роботи із сокетами.

2.2 Розробка програмного забезпечення для плати BeagleBone Black

2.2.1 Крос-компіляція

Розглянемо інструментальні засоби, які є фундаментальною частиною операційної системи Linux для вбудованих систем, які в свою чергу майже завжди слугують компонентою для систем IoT. Тулчейн (набір інструментів, дослівно — ланцюжок інструментів) повинен ефективно використовувати ваше обладнання, оптимально застосовувати набір інструкцій процесора. Також, він має підтримувати мови програмування, які використовуватимуться для розробки програмного забезпечення, мати реалізацію POSIX (англ. Portable Operating System Interface) та інших системних інтерфейсів. Тулчейн, для вашого проєкту, має бути гнучким, мати підтримку зараз та в майбутньому, оновлюватись, у випадках виявлення критичних помилок, проблем із безпековою складовою.

У випадку використання стандартного тулчейну, його складових, отримання його — це простий процес, що полягає в завантаженні та встановленні пакету інтструментів.

Тулчейн — це комплект програмних інструментів, призначених для крос-компіляції програмного забезпечення, яке використовує архітектуру, відмінну від архітектури хост-системи. В нашому випадку, плата BeagleBone Black має архітектуру ARM, а для розробки програмного забезпечення, використовуватимемо персональний комп’ютер на архітектурі x86_64 (можна дізнатись в налаштуваннях системи, для ОС Linux можна, наприклад, виконати команду *lscpu*) [20].

Linux тулчейн використовується для створення програм середовища користувача, що захоплюють середовище Linux. Він містить повну версію GNU libc (glibc), яка спирається на інтерфейс ядра системного виклику (syscall kernel interface). Він використовується для створення rootfs та user-space інструментів.

Також важливим елементом є файли-заголовки ядра — двійковий програмний інтерфейс (ABI, Application Binary Interface), необхідні для взаємодії з ядром. Їх роль — надавати доступ до системних викликів, структур даних та визначень констант, що дозволяє виконати скомпільований код на машині, яка має такі ж параметри, задані заголовками ядра. Звісно, кожна C-бібліотека визначає набори специфікацій, які є специфічними для кожної апаратної архітектури [21]

Для крос-компіляції програмного забезпечення необхідно попередньо виконати наступні кроки:

- Мати rootfs цільової системи на хост-машині;
- Завантажити архів тулчейна, наприклад з сайту Linaro, для потрібної нам архітектури, розпакувати його;
- Вміти користуватись make, cmake;

Для використання тулчейну необхідно додати нову змінну середовища — в середовищі оболонки (shell) нам потрібно додати шлях до каталога bin/ завантаженого тулчейна до змінної середовища $PATH.

Крім того, зазвичай потрібно надавати змінну $CROSS_COMPILE з рядком префікса вашого тулчейна (це буде далі використовуватися в Makefile скомпільованого проєкту для запуску правильного інструменту, наприклад, ${CROSS_COMPILE}gcc). [20]

2.2.2 Завантаження TFTP

Працюючи із BeagleBone Black можливо не використовувати локальне сховище пристрою. Найкращий спосіб завантажувати ядро — через мережу,

особливо якщо rootfs монтується через мережеву файлову систему (Network File System, NFS). Завантажувач U-Boot дозволяє використовувати даний спосіб, бо підтримує тривіальний протокол передачі файлів (Trivial File Transfer Protocol, TFTP).

Необхідно під’єднати плату до хост-комп’ютера за допомогою Ethernet. Користувач налаштованої системи робить запит до сервера про надання йому файлів та директорій скомпільованого ядра системи, дерева пристроїв (device tree blob, dtb), кореневої файлової системи. Отримане програмне забезпечення буде розміщене в оперативній пам’яті пристрою, монтуючи rootfs як RAM-диск, звідки в подальшому буде проведено завантаження плати [22].

На рисунку 2.1 можна побачити, як працює завантаження TFTP:

1. плата робить запит необхідного програмного забезпечення;
2. сервер надає відповідь, яка містить запитувані ядро, файлову систему, дерево пристроїв.

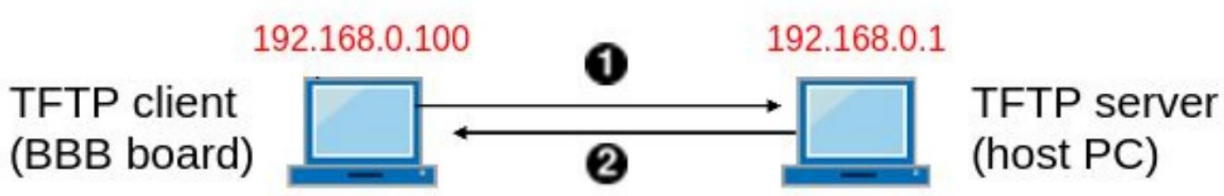

Рисунок 2.1 – Завантаження TFTP зі статичним IP [23]

Аналізуючи описану процедуру, ми можемо побачити плюси та мінуси такого методу завантаження та можливі випадки використання, при яких це може бути зручно.

Серед переваг такого методу варто зазначити наступні:

- це альтернативний спосіб завантаження системи, а значить він не задіює пам’ять eMMC чи картки пам’яті, які в цей час можуть мати інші збірки системи, збережені файли;
- доволі висока швидкість роботи із даними (обмежена максимальною швидкістю інтерфейсу підключення)

Недоліки цього підходу є такими:

- необхідно провести попередню конфігурацію, що буде розглянуто протягом цього розділу
- кожне нове завантаження системи плати потребує надання потрібного програмного забезпечення через TFTP

Аби не використовувати DHCP налаштування (не в останню чергу, для спрощення розробки програмного забезпечення в наступному розділі) використовуватимемо статичну конфігурацію Ethernet на платі.

Щоб запустити Linux за допомогою мережевого завантаження TFTP, нам знадобиться ramdisk у так званому форматі “застарілого образу U-Boot”. У цьому випадку ви навіть можете видалити каталог /boot у вашій rootfs, оскільки файли zImage і dtb будуть отримані через Ethernet.

Відома методика конфігурації TFTP описана в роботі [22], послідовність основних етапів наведена далі:

1. Встановіть пакет з утилітою mkimage: $ sudo apt install u-boot-tools
2. Створіть CPIO архів ваших rootfs і стисніть (помістіть в архів) його за допомогою GZip:

```
$ cd ~/repos/busybox
$ cd _install
$ find . | cpio -o -H newc | gzip > ../rootfs.cpio.gz
$ cd ..
```

3. Тепер створіть застарілий образ U-Boot вашого RAM-диска:

```
$ mkimage                \
        -A arm           \
        -T ramdisk       \
        -C gzip          \
        -O linux         \
        -n "Ramdisk Image"  \
```

```
-d rootfs.cpio.gz          \
uRamdisk
```

Отриманий файл uRamdisk. Його буде передано через мережу дуже швидко, оскільки це в основному заархівована (стиснута) rootfs, і тому його розмір досить малий.

2.2.2.1 Конфігурація хоста

2.2.2.1.1 Конфігурація демона TFTP

Завдання демона tftpd запускати TFTP сервер. Надсилати файли можна вказуючи повний шлях в командному рядку демона. Існує декілька способів його запуску:

- зробити демон бекграунд процесом, вказавши це у конфігураційному файлі tftpd-hpa (конфігурація TFTP-сервера);
- надати керування демоном суперсерверу.

Xinetd — це демон суперсервера. Його завдання — слухати мережу та запускати служби, відповідні відловленим запитам. В нашому випадку, при перехопленні TFTP запиту, він запустить демон tftpd. Для зручності та впевненості у свідомому використанні tftpd оберемо варіант надання керування демоном утиліті Xinetd.

Далі в магістерській роботі запропонована методика налаштування TFTP, яка базується на описаному раніше підході.

Встановимо TFTP-сервер і Xinetd на хост-комп’ютер:

$ sudo apt install xinetd tftpd-hpa

Перевіримо налаштування сервера у файлі /etc/default/tftpd-hpa, переконаємося у відсутності непотрібних нам встановлених параметрів. Відредагуємо файл /etc/xinetd.d/tftp або створимо його в разі відсутності, наповнивши наступними параметрами конфігурації (з повним списком

доступних параметрів можна ознайомитись за допомогою «man tftpd»), які зазначені в таблиці 2.1, помістивши їх у блок:

```
service tftp
{
        … параметри
}
```

Таблиця 2.1

Налаштування tftp конфігурації

protocol = udp	Протокол
port = 69	Порт для сервера
socket_type = dgram	Тип сокета для протоколу UDP
wait = yes	Очікування на запуск
server = /usr/sbin/in.tftpd	Шлях до демона сервера
disable = no	Увімкнути створення процесу TFTPD
user = root	Запустити демон tftp від імені користувача root. Це потрібно для прослуховування привілейованого порту 69 (оскільки він нижче 1024) і щоб зробити chroot (операція зміни кореневого каталогу) для каталогу /srv/tftp
server_args = --secure	Використовувати відносні (а не абсолютні) шляхи
server_args = --user tftp	Читати файли з дозволами цього користувача
server_args = /srv/tftp	Шлях до файлів TFTP

Аби зміни застосувались, необхідно зупинити сервер TFTP і перезапустити суперсервер, перенісши попередньо необхідне програмне забезпечення за шляхом, вказаним в налаштуваннях, змінивши власника цих файлів за допомогою $sudo chown -R tftp:tftp /srv/tftp :

$ sudo service tftpd-hpa stop

$ sudo service xinetd restart

2.2.2.1.2 Налаштування інтерфейсу Ethernet

Як було згадано раніше, ми не використовуватимемо налаштування сервера DHCP, натомість використовуватимемо статичні IP-адреси для спрощення. Впевнимось, що хост-комп'ютер має вільний порт для підключення плати, при цьому маючи доступ до мережі через інший фізичний інтерфейс.

Визначимо назву інтерфейсу Ethernet:

$ ifconfig

Отримаємо щось на зразок enp0s3, eth0. Використовуючи дану інструкцію, не забувайте про ваш власний інтерфейс, бо далі використовується саме eth0 для позначення інтерфейсу.

Тепер давайте встановимо статичну IP-адресу для інтерфейсу Ethernet. Відредагуйте файл /etc/network/interfaces наступним чином:

auto eth0

iface eth0 inet static

address 192.168.0.1

netmask 255.255.255.0

Для застосування наданої конфігурації запустіть наступні команди, зл перезавантажать мережевий сервіс на платі:

$ sudo ip addr flush eth0

$ sudo service networking stop

$ sudo service networking start

Переконавшись в наданні статичної адреси інтерфейсу за допомогою ifconfig, можемо спробувати відправити пакети в мережі, наприклад ping <адреса-хост-комп’ютера> з плати чи навпаки. Наші зміни залишаться при наступних завантаженнях.

2.2.2.2 Запуск завантаження TFTP на пристрої

Підготуйте середовище U-Boot для завантаження TFTP з оперативної пам’яті [22]:

=> setenv ipaddr 192.168.0.100

=> setenv serverip 192.168.0.1

=> setenv rdfile uRamdisk

=> setenv netloadrd 'tftp ${rdaddr} ${rdfile}'

=> setenv tftpramboot 'run findfdt; setenv autoload no; run netloadfdt; run netloadimage; run netloadrd; run ramboot'

=> env save

Перевірте, чи працює сервер TFTP:

=> ping <адреса плати>

Тепер ви можете використовувати команду “tftpramboot” (вона також буде доступна під час наступного завантаження, оскільки її було збережено в середовищі). Запустіть її, щоб виконати завантаження TFTP[23]:

=> run tftpramboot

2.2.2.3 Завантаження NFS

Запропонована в даній магістерській роботі методологія спонукає використовувати завантаження NFS, яке дозволить отримувати із сервера кореневу файлову систему, в той час як ядро та дерево пристроїв передаватимуться на плату за раніше представленою процедурою за допомогою TFPT, що схематично зображено на рисунку 2.2 та відбувається наступним чином:

1. Отримаємо файли ядра та дерева пристроїв через TFTP
2. Запустіть ядро, вказавши джерело rootfs — сервер NFS
3. RootFS буде змонтовано, використовуючи мережу

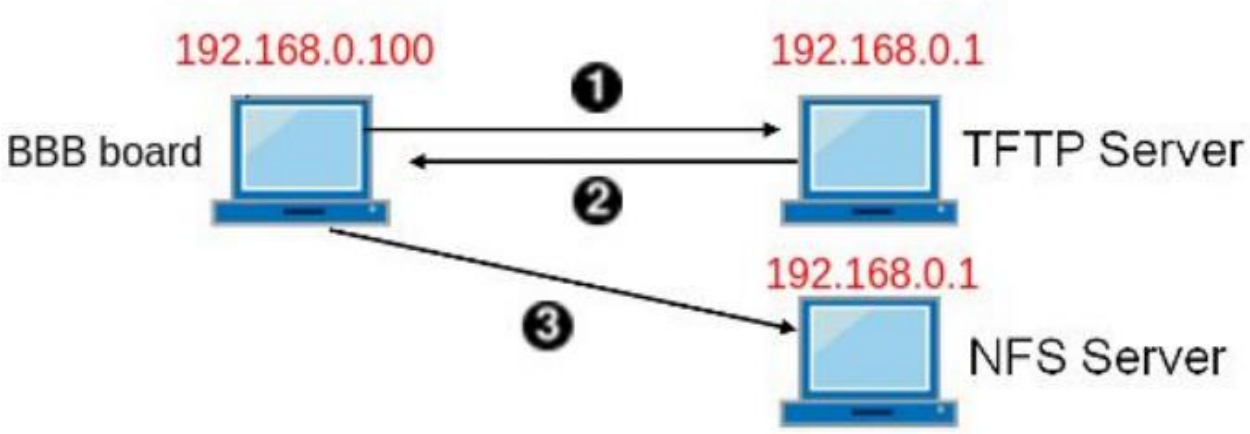

Рисунок 2.2 - Завантаження NFS зі статичним IP [23]

Щоб переконатися, що ваше ядро може монтувати RootFS через NFS, слід увімкнути наступні параметри за допомогою defconfig при збірці ядра:

- CONFIG_ROOT_NFS=y: підтримка монтування rootfs через NFS
- CONFIG_TI_CPSW=y: драйвер для контролера Ethernet на BeagleBone Black

2.2.2.3.1 Конфігурація хоста

Наступним кроком є встановлення та налаштувати сервера NFS на вашому хості. На ОС Ubuntu ви можете зробити це за допомогою наступної команди:

$ sudo apt install nfs-kernel-server

Підготуйте RootFS для доступу NFS:

$ sudo mkdir -p /srv/nfs/busybox

$ sudo cp -r ~/repos/busybox/_install/. /srv/nfs/busybox

При запуску на платі власник директорії, який є в мета даних (inode) для цієї директорії збережеться з хост-комп’ютера, а тому необхідно його змінити — ми хочемо, щоб усі файли RootFS належали користувачеві root:

$ sudo chown -R root:root /srv/nfs

Оновимо файл експорту NFS /etc/exports і додамо туди інформацію про NFS клієнта та зазначимо права доступу до файлів для потрібного нам користувача:

/srv/nfs/busybox \

192.168.0.100/24(rw,sync,no_subtree_check,no_root_squash)

Запустимо сервер NFS, переконавшись що RootFS експортована:

$ sudo /etc/init.d/nfs-kernel-server start

$ sudo exportfs -a

2.2.2.4 Запуск завантаження TFTP із rootfs NFS

Після успішного налаштування для завантаження TFTP в попередньому розділі, необхідно надати йому інструкції для використання rootfs із сервера NFS. Налаштуємо завантаження TFTP з RootFS, змонтованою з NFS в U-Boot [23]:

=> setenv ipaddr 192.168.0.100

=> setenv serverip 192.168.0.1

=> setenv rootpath /srv/nfs/busybox

=> setenv nfsopts nolock,nfsvers=3

```
=> setenv netboot 'run findfdt; setenv autoload no; run netloadimage; run
netloadfdt; run netargs; bootz ${loadaddr} - ${fdtaddr}'
=> setenv netargs 'setenv bootargs console=${console} ${optargs}
root=/dev/nfsnfsroot=${serverip}:${rootpath},${nfsopts} rw ip=${ipaddr}'
=> env save
=> run netboot
```

Серед параметрів, на які слід звернути увагу, є параметр “nfsvers=3”, без якого ядро не зможе підключити RootFS через NFS, причини та наслідки, а також. Інші параметри та їх значення можна переглянути за допомогою «man nfs».

Запустимо команду “mount”, щоб перевірити, чи було змонтовано RootFS з сервера NFS, що також можна зробити за допомогою перегляду журналу ядра.

Висновки до розділу 2

У другому розділі даної магістерської роботи було розглянуто особливості використання клієнт-серверної архітектури та її складових, а саме TCP та Netlink сокетів, на яких побудовано програмне забезпечення в рамках даної роботи.

Запропоновано спосіб завантаження кореневої файлової системи за допомогою механізму завантаження через підняття та налаштування серверу TFTP, використовуючи мережеву файлову систему, яка використовується Edge сервером.

Приділено окрему увагу кожному із аспектів успішного завантаження файлової системи на віддалений сервер. Запропоновано набір потрібних для завантаження налаштувань мережевого інтерфейсу, демона TFTP.

Детально описано послідовність дій для успішного завантаження необхідних утиліт, та їхнього ефективного функціонування запланованим чином.

РОЗДІЛ 3

ОПИС ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ EDGE СЕРВЕРА

3.1 Знайомство з сокетами

3.1.1 TCP з’єднання

Почнемо знайомство з сокетами із розгляду класичного прикладного програмного інтерфейсу – сокетів Берклі та реалізованим TCP сервером на їх основі. Розроблений нами TCP сервер стане основою для подальшого поглиблення отриманих знань в області сокетного програмування.

Аби пояснити доцільність розробки нашої першої програми, використовуючи саме протокол транспортного рівня TCP, представимо подальший план дій:

- розробка базового сервера та клієнта для нього (встановлення з’єднання, передача вітального повідомлення);
- розширення функціональності програмного забезпечення за допомогою імплементації можливості передачі вже цілих файлів, замість hard-code рядків;
- перетворення імплементованої серверної частини програмного забезпечення на HTTP сервер, клієнтською частиною виступатиме браузер;
- розробка програмного забезпечення для сканування доступних WiFi мереж та їхніх характеристик, використовуючи спілкування ядра операційної системи Linux із програмою середовища користувача за допомогою сокетів;
- додавання можливості переглядати результати роботи програмного забезпечення для сканування WiFi мереж за допомогою браузера, використовуючи розроблений попередньо HTTP сервер.

Із зазначеного вище плану випливає необхідність встановлення з'єднання між клієнтом і сервером, також ми потребуватимемо цілісності переданої інформації, а тому використовуватимемо саме TCP протокол як базу для нашої системи.

Настав час зробити перші кроки.

Бібліотека <sys>, що входить до базового набору бібліотек мови C, містить необхідні нам компоненти для роботи із сокетами <sys/socket.h>. Цей файл надасть нам можливість користуватись інтерфейсом створення сокетів, взаємодії із ними, завершення роботи у вигляді їхнього закриття.

Скористаємося практичними настановами до використання компонентів даної бібліотеки. Почнемо ознайомлення зі створення сокета за допомогою виклику

$man socket

результат якого можемо побачити на рисунку 3.1.

```
SOCKET(2)                  Linux Programmer's Manual                 SOCKET(2)

NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);

DESCRIPTION
       socket()  creates  an endpoint for communication and returns a file de-
       scriptor that refers to that endpoint.  The file descriptor returned by
       a  successful call will be the lowest-numbered file descriptor not cur-
       rently open for the process.

       The domain argument specifies a communication domain; this selects  the
       protocol  family  which will be used for communication.  These families
       are defined in <sys/socket.h>.  The formats currently understood by the
       Linux kernel include:
```

Рисунок 3.1 – Документація socket.

Ознайомимося із описом та скористаємося пошуком по файлу документації за допомогою вводу символа "\" за яким слідує наш пошуковий запит. Натиснемо клавішу вводу ("Enter") на клавіатурі та виконуватимемо

перехід між знайденими результатами за допомогою клавіш “n” та “shift+n” для прямого та зворотнього переходу між результатами пошуку відповідно.

Ми маємо попередню мету - створити TCP сокет, а тому підійдемо до вибору аргументів, що передаються під час створення сокета враховуючи нашу ціль.

Почнемо із комунікаційного домена, який іноді називають простором імен або адрес. Він включає в себе наступне:

- правила використання та тлумачення імен;
- набір пов’язаних форматів адрес, які формують сімейство адрес;
- набір протоколів, який називається сімейством протоколів.

TCP протокол входить до сімейства інтернет протоколів, а тому нашим вибором буде AF_INET домен.

Тип сокета також залежить від обраного протоколу, визначити який ми можемо навіть за описом типів в документації, або ж, для певності, звернемось до документації TCP протокола, в якій буде явно зазначено тип, пов’язаний із необхідним нам протоколом

$man tcp

результат якого можемо побачити на рисунку 3.2.

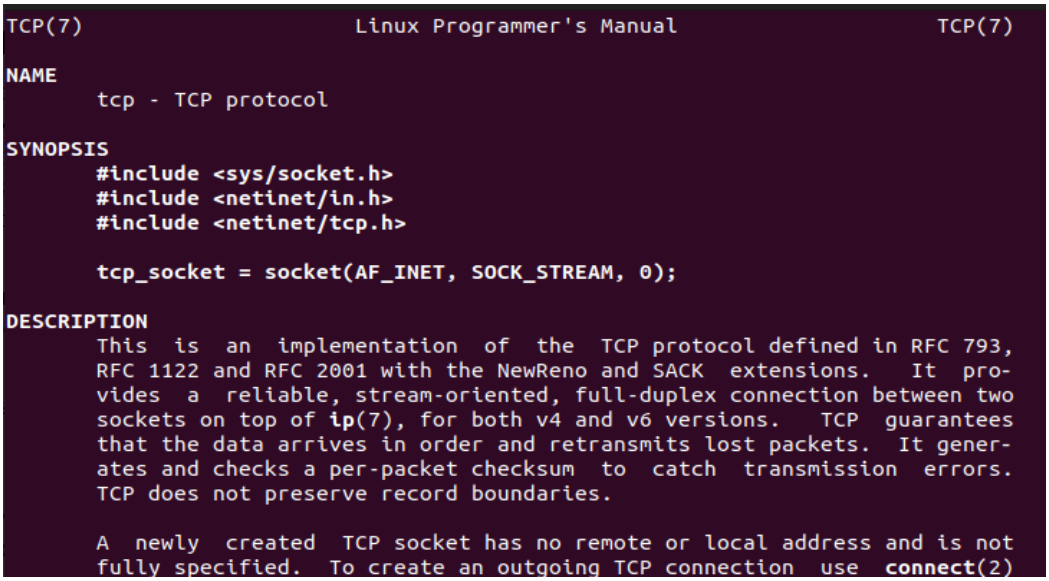

Рисунок 3.2 – Документація tcp.

В описі ми одразу можемо побачити, що для створення необхідного нам сокета потрібно зробити іще декілька кроків.

Створений TCP-сокет не має віддаленої чи локальної адреси та не є повністю визначеним. Щоб створити вихідне TCP-з'єднання, необхідно скористатись функцією connect(), щоб встановити з'єднання з іншим TCP-сокетом.

Щоб мати можливість отримувати нові вхідні з'єднання, необхідно спочатку прив'язати сокет до локальної адреси та порту за допомогою функції bind(), та перевести сокет у стан очікування на під'єднання, або ж прослуховування за допомогою функції listen(). Після цього новий сокет зможе приймати вхідні з'єднання за допомогою функції accept().

Сокет, який успішно викликав accept() для прийняття вхідних з'єднань або connect() - для створення вихідного з'єднання є повністю визначеним і може передавати дані. Дані не можуть бути передані на прослуховуваних або ще не підключених сокетах.

Кожна зі згаданих вище функцій містить опис, викладений в Linux Programmer's Manual (man). Для ознайомлення із ними варто викликати команду $man, за якою слідуватиме бажане ім'я функції, в терміналі.

3.1.2 TCP сервер

Продовжимо знайомство з інтерфейсом створення та роботи із сокетами під час написання коду.

Як ми вже знаємо для початку нам необхідно створити сокет, та як результат, отримати файловий дескриптор, який більш притаманно називати дескриптором сокета, для даного випадку.

Дескриптор сокета (або ж іноді номер сокета) — це ціле число в двійковій системі числення, яке слугує індексом до таблиці сокетів серед сокетів, які на даний момент призначені для даного процесу. Дескриптор сокета є унікальним в межах процесу представленням сокета, але не є самим сокетом.

```
// Create a socket
int SocketFD = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (SocketFD == -1)
{
    perror("cannot create socket");
    exit(EXIT_FAILURE);
}
```

Рисунок 3.3 – Створення сокета.

При створенні сокета, як на рисунку 3.3, судячи із представленої вище документації, щось може піти не так, і чим більше місць у коді, де наші очікування можуть не співпасти з реальністю, тим складніше і довше знаходити це місце. Тут і в подальшому, для відловлення неприємних ситуацій ми використовуватимемо error handling. У випадку нестворення сокета, замість файлового дескриптора ми отримаємо “-1”, як код, який повідомляє про помилку.

В такому випадку розробнику, або ж користувачу варто знати, що відбулось, а тому ми повідомимо про помилку, вивівши її суть у стандартний потік помилок та отримаємо код глобальної змінної помилок за допомогою стандартної функції perror().

Наша програма має точку входу - функцію main(), а також і точку виходу із неї - неявну -, в кінці успішно виконаного коду, або ж явну - задану розробником. У випадку розробки програмного забезпечення для подальшого виходу на ринок, ПЗ має вміти повертати систему до стабільного стану, повторювати намагання виконати певний код, чи мати іншу логіку, яка б задовольняла поняття користувача про безвідмовність роботи ПЗ.

В нашому випадку, для простоти розробки ПЗ та його тестування, у випадку настання критичної, для роботи нашого ПЗ, ситуації робота програми завершуватиметься, завдяки заданій точці виходу за допомогою функції exit(), яка приймає код помилки, з якою ми би хотіли завершити нашу програму. Погодьтесь, відсутність створеного сокета для сервера є достатньо критичною причиною завершити виконання його коду в цей самий момент.

Створений сокет можна конфігурувати. Завдяки використанню функцій setsocketopt()/getsocketopt() можна задавати та отримувати значення певних налаштувань. Зі списком доступних опцій можна ознайомитись в документації до socket в розділі “Socket options”.

Варто звернути увагу на деякі із них. При частому перезапуску програмного коду, виникатиме ситуація, некоректного закриття сокета, який ще певний час буде сприйматись системою як незакритий та все ще володітиме виділенними ресурсами.

Для уникнення даної ситуації можна дозволити сокету при створенні перевикористовувати адресу або порт із локального пулу, як показано на рисунку 3.4.

```
    int optval = 1;
#ifdef SO_REUSEPORT
    if (setsockopt(SocketFD, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &optval, sizeof(optv
#else
    if (setsockopt(SocketFD, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)))
#endif
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
```

Рисунок 3.4 – Опції сокета.

SO_REUSEPORT опція сокета дозволяє кільком сокетам на одному хості прив’язуватися до одного порту та призначена для покращення продуктивності багатопоточних мережевих серверних програм, що працюють на основі багатоядерних систем.

SO_REUSEADDR вказує, що серед правил перевірки адрес при виклику bind(), повинно бути дозволено повторне використання локальних адрес. Для сокетів сімейства AF_INET це означатиме, що сокет може виконати прив’язку до адреси, за винятком випадків, коли існує активний відкритий прослуховуючий сокет, прив’язаний до цієї адреси.

Аргументом виклику функції setsocketopt() є значення *const void *optval*, яке в даному випадку є цілочисельним прапорцем, який відповідальний за увімкнення даної опції в налаштування створеного сокета.

Наступним кроком є прив’язка створеного сокета до порта та адреси.

Більшість функцій сокетів вимагають вказівник на структуру адреси сокета як аргумент. Кожен підтримуваний набір протоколів визначає власну структуру адреси сокета. Назви цих структур починаються з sockaddr_ і закінчуються унікальним суфіксом для кожного набору протоколів.

Структура адреси сокету IPv4, яку зазвичай називають «структурою адреси сокета Інтернету», називається sockaddr_in і визначається за допомогою заголовка <netinet/in.h>. На рисунку 3.5 показано визначення даної структури

```
/* Structure describing an Internet socket address.  */
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;     /* Port number.  */
    struct in_addr sin_addr;    /* Internet address.  */

    /* Pad to size of `struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr)
                           - __SOCKADDR_COMMON_SIZE
                           - sizeof (in_port_t)
                           - sizeof (struct in_addr)];
  };
```

Рисунок 3.5 – Структура адреси сокета Інтернету.

що також містить і сімейство даної структури, яке визначено за допомогою макроса на рисунку 3.6.

```
#define __SOCKADDR_COMMON(sa_prefix) sa_family_t sa_prefix ## family
Expands to:
sa_family_t sin_family
```

Рисунок 3.6 – Визначення сімейства.

Надана адреса виступатиме адресою хоста, за якою він буде доступний в мережі. Цей хост на визначеному порті матиме наш розвернутий сервер. Порт оберемо будь-який із незайнятих портів в нашій системі.

Можемо переглянути порти, які вже використовуються нашою системою за допомогою виклику

$sudo netstat -tulpn

та отримаємо результат, подібний рисунку 3.7

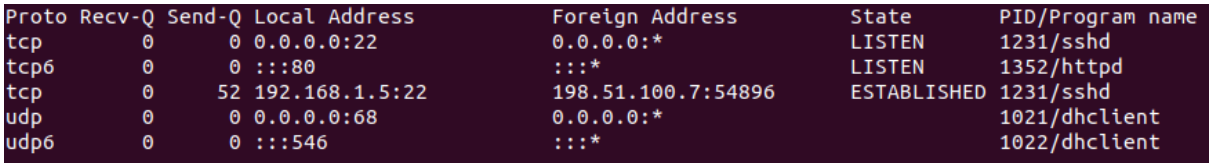

Рисунок 3.7 – Результат виконання netstat.

або ж подивимось лише на підключення в режимі прослуховування за допомогою виклику

$sudo netstat -tulpn | grep LISTEN

та отримаємо результат, подібний рисунку 3.8

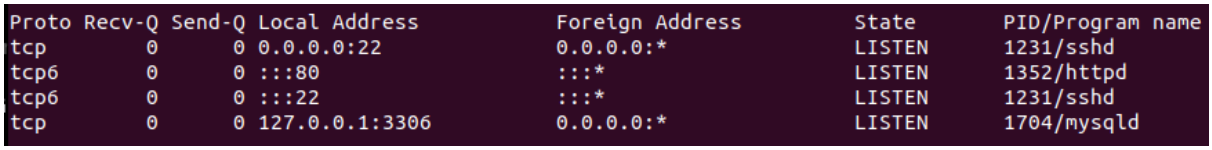

Рисунок 3.8 – Список сокетів в прослуховуючому стані.

Часто, вибір припадає на порти 8080, 1100, 1010, або ж будь-які інші не зайняті за бажанням.

Для визначеної мережі, в якій буде розгорнуто наш сервер, нам необхідно покладатись на DHCP сервер, який видаватиме нам адресу на свій смак, і тоді в коді програми необхідно буде робити виклик утиліти ifconfig для заданого мережевого інтерфейса, парсити вивід утиліти, та отримувати адресу. Також можливо скористатися налаштуваннями dhclient на хості та просити видавати лише певну адресу, яку ми визначимо в коді програми, або ж надати статичні налаштування певному інтерфейсу, який не отримуватиме налаштування від dhcp сервера.

Набір утиліт, доступний на платі BeagleBone Black може не містити утіліт для динамічної конфігурації хоста в мережі, а тому в нашому випадку ми скористаємося статичними налаштуваннями і видамо сталу адресу.

На іншому хості дізнаємося адресу потрібної нам мережі за допомогою виклику утиліти ifconfig та виділимо нашій платі адресу, наприклад:

$ip addr add 192.168.10.100/24 dev eth0

та маршрут за замовчуванням

$ip route add default via 192.168.10.100 dev eth0

Визначимо порт та маршрут в нашій програмі використовуючи обрані раніше адресу та порт як показано на рисунку 3.9.

```
#define PORT 8080
#define RESERVED_ADDR "192.168.10.100"
```

Рисунок 3.9 – Порт та адреса сервера.

Створимо структуру адреси сокета для сімейства інтернет протоколів із заданими значеннями як на рисунку 3.10.

```
// Create an internet socket address structure
struct sockaddr_in sa;
memset(&sa, 0, sizeof sa);

// Fill the internet socket address structure
sa.sin_family = AF_INET;
sa.sin_port = htons(PORT);
sa.sin_addr.s_addr = inet_addr(RESERVED_ADDR);

// Bind the socket to the port
if (bind(SocketFD, (struct sockaddr *)&sa, sizeof sa) == -1)
{
    perror("bind failed");
    close(SocketFD);
    exit(EXIT_FAILURE);
}
```

Рисунок 3.10 – Заповнення структури адреси сокета.

Виклик memset() з нулем на місці другого аргумента заповнить нашу структуру нульовими значеннями. Функції hton() та inet_addr() сконвертують

надані нами значення в формат змінної того типу даних, якого потребують поля заповнюваної структури.

Структури адреси сокета завжди передаються за посиланням, коли передаються як аргумент до будь-якої функції сокета. Але будь-яка функція сокета, яка приймає один із цих посилань як аргумент, повинна мати справу зі структурами адрес сокетів із будь-якого сімейства підтримуваних протоколів.

Виникає проблема, як оголосити тип покажчика, який передається. В стандарті мови C - ANSI C, рішення просте: void * є загальним типом покажчика. Але функції сокетів передували ANSI C, і рішенням, вибраним у 1982 році, було визначення загальної структури адреси сокета в заголовку <sys/socket.h>, тому передаючи посилання на структуру адреси необхідно провести приведення до цієї загальної структури за допомогою (struct sockaddr *), що передує посиланню.

Якщо ми опускаємо приведення "(struct sockaddr *)," компілятор C генерує попередження у формі "warning: passing argument 2 of ‘bind’ from incompatible pointer type", припускаючи, що заголовки системи мають прототип ANSI C.

```
// Start listening on the socket
if (listen(SocketFD, LISTENING_QUEUE_SIZE) == -1)
{
    perror("listen failed");
    close(SocketFD);
    exit(EXIT_FAILURE);
}
```

Рисунок 3.11 – Виклик функції listen.

На рисунку 3.11 зображено виклик функції listen(), яка перетворює непідключений сокет на пасивний, вказуючи, що ядро операційної системи має приймати вхідні запити на підключення, спрямовані до цього сокета. З точки зору стану TCP, виклик listen() переміщує сокет із стану CLOSED у стан LISTEN.

Другим аргументом передається ціле число, природу якого необхідно розуміти наступним чином. Ядро операційної системи має дві черги:

- черга неповних підключень, яка містить запис для кожного SYN (TCP пакета), який надійшов від клієнта, для якого сервер очікує завершення тристороннього рукостискання TCP. Ці сокети знаходяться в стані SYN_RCVD;
- черга повних підключень, яка містить записи для кожного клієнта, з яким завершено тристороннє рукостискання TCP. Ці сокети знаходяться в стані ESTABLISHED.

Легкої відповіді, яким це значення має бути не існує. Найкращий вибір, динамічно виділяти розмір цієї черги, а значить збільшувати за потреби. В нашому випадку, для місцевого неглобального користування нам вистачить і розміру в десять місць у черзі.

Для прийняття вхідних запитів нам лишилось лише прийняти підключення від клієнта, тобто встановити TCP підключення із клієнтом. Ця операція проводиться завдяки функції accept() - наступний клієнт в черзі повних підключень зможе скористатись серверним функціоналом.

Ми би хотіли аби наш сервер приймав підключення допоки він знаходиться в робочому стані, а тому необхідно виконання функції accept в цикл, як на рисунку 3.12.

```
struct sockaddr_in client_sa;
socklen_t client_addr_len;
char buff[BUFFER_SIZE];
memset(&client_sa, 0, sizeof client_sa);
for (;;)
{
    printf("Waiting for a connection...\n");

    // Accept a connection
    int ConnectFD = accept(SocketFD, (struct sockaddr *)&client_sa, &client_addr_len);
    // Check if the connection was accepted
    if (ConnectFD == -1)
    {
        perror("accept failed");
        close(SocketFD);
    }

    printf("Connection from %s client on port %d is accepted!\n",
           inet_ntop(AF_INET, &client_sa.sin_addr, buff, sizeof(buff)), ntohs(client_sa.sin_port));
```

Рисунок 3.12 – Цикл роботи сервера.

Додамо індикацію очікування підключення та успішного підключення клієнта конкретного клієнта, як зображено на малюнку, - функція accept() запише в надану йому структуру дані структури підключеного сокета клієнта. При підключенні отримаємо вивід, подібний наступному:

Waiting for a connection...

Connection from 192.168.10.101 client on port 43470 is accepted!

Тепер настав час обміну даними в мережі з уже встановленим підключенням. Тривіальною задачею є обмін вітальними повідомленнями з обидвох сторін. Для кращого масштабування нашої програми в подальшому, помістимо логіку зчитування повідомлення від клієнта та відправку йому відповіді, як показано на рисунку 3.13.

```
// We need to define a function that will handle the connection
void handle_connection(int client_socket_FD)
{
    printf("Connection accepted!\n");
    // Create a buffer for a received message
    char buffer[BUFFER_SIZE];
    // Read the data from the client into the buffer
    read(client_socket_FD, buffer, BUFFER_SIZE);
    // Print the received message
    printf("Received message from client: %s\n", buffer);
    // Send a message to the client
    char *message = "Hello from the server!";
    write(client_socket_FD, message, strlen(message));
}
```

Рисунок 3.13 – Функція обробки з’єднання.

Зчитування даних отриманих від клієнтського сокета в буфер здійснюється завдяки функції read(), відправка назад - завдяки функції write().

Після чого нам необхідно закрити створений при під’єднанні сокет комунікації з конкретним клієнтом та чекати на наступні підключення до завершення роботи нашого сервера (що зображено на рисунку), яке поки

відбувається лише під час надсилання сигналу ядра операційної системи SIGINT за допомогою комбінації клавіш “Ctrl+C” в терміналі.

```
        handle_connection(ConnectFD);

        printf("Closing connection!\n\n");
        // Close the connection
        if (shutdown(ConnectFD, SHUT_RDWR) == -1)
        {
            perror("shutdown failed");
        }
        close(ConnectFD);
    }
    // Close the server socket
    close(SocketFD);
    return EXIT_SUCCESS;
}
```

Рисунок 3.14 – Функція обробки з’єднання.

Закриття з’єднання повністю або його частини (на рисунку 3.14), відкритого на дескрипторі сокета відбувається завдяки виклику функції shutdown(), яка приймає повернутий функцією accept() дескриптор сокета підключення та спосіб закриття з’єднання другим аргументом. Як його закрити визначається за допомогою прапорців:

- SHUT_RD = закриття прийому даних;
- SHUT_WR = закриття передачі;
- SHUT_RDWR = закриття прийому та передачі разом.

Таким чином ми завершили розгляд серверної частини TCP сервера, який виконує тривіальну задачу з прийому та передачі вітального повідомлення.

3.1.3 TCP клієнт

Після докладного розбору серверної частини, для написання клієнтського коду для передачі та прийому повідомлення від сервера при підключенні та завершення його роботи, ми вже готові майже на сто відсотків. Для клієнтської частини нам необхідно лише створити сокет функцією socket(), під'єднатись до працюючого сервера за допомогою connect(), виконати основну задачу, закрити клієнтський сокет та завершити виконання клієнтської частини ПЗ.

```
// Create a socket
int SocketFD = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (SocketFD == -1)
{
    perror("cannot create socket");
    exit(EXIT_FAILURE);
}


struct sockaddr_in sa;
memset(&sa, 0, sizeof sa);

// Fill the internet socket address structure
sa.sin_family = AF_INET;
sa.sin_port = htons(PORT);
sa.sin_addr.s_addr = inet_addr(RESERVED_ADDR);

printf("Connecting to the server...\n");

// Connect to the server
if (connect(SocketFD, (struct sockaddr *)&sa, sizeof sa) == -1)
{
    perror("connect failed");
    close(SocketFD);
    exit(EXIT_FAILURE);
}

printf("Connected!\n");
```

Рисунок 3.15 – Ініціалізація сокета, заповнення структури та під'єднання.

На рисунку 3.15 бачимо, що для нас вже немає нічого невідомого, лише маємо функцію connect(), яка ініціює потрійне рукостискання TCP, замість

функції accept(), яку мав сервер та поведінку якої описано вище. Клієнтський сокет не потребує прив’язки функцією bind(), це завдання ляже на плечі ядра операційної системи неявно. В свою чергу, функція listen() нам не знадобиться, бо ми знаходимося в активному пошуку слухаючого сокета, listen() змінить поведінку нашого сокета на очікування SYN пакета, коли без її виклику - переводу сокета в пасивний стан, наш сокет самостійно стане ініціатором надсилання SYN пакета, тобто клієнтом в потрійному рукостисканні TCP.

Тепер напишемо повідомлення серверу, та отримаємо від нього відповідь, таким самим чином, як це відбувалось на серверній частині, як зображено на рисунку 3.16.

```
void handle_connection(int SocketFD)
{
    // Send a message to the server
    char *message = "Hello from the client!";
    write(SocketFD, message, strlen(message));

    // Create a buffer for a received message
    char buffer[BUFFER_SIZE];
    // Read the data from the server into the buffer
    read(SocketFD, buffer, BUFFER_SIZE);
    // Print the received message
    printf("Received message from server: %s\n", buffer);
}
```

Рисунок 3.16 – Обробка з’єднання із сервером.

На рисунку 3.17 зображено виклик функції, яка виконає нашу основну задачу підключення та закриття сокета.

```
    // Handle the connection
    handle_connection(SocketFD);

    // Close the socket
    close(SocketFD);
    return EXIT_SUCCESS;
}
```

Рисунок 3.17 – Завершення виконання клієнтського коду.

Таким чином завершиться успішне виконання коду клієнтської частини.

3.1.4 Підключення клієнта до сервера

Запустимо серверну частину нашої програми на платі та клієнтську – на власному комп’ютері.

Для цього спершу необхідно скомпілювати наш код для його виконання платою. Цього можна досягнути за допомогою крос-компіляції.

У випадку включеного secure shell (ssh) сервера в збірку прошивки плати скористаємося наступним варіантом перенесення виконуваного файлу на плату.

Маючи доступ до створеного користувача виконаємо наступну дію на хості:

$scp <source_file_path> <user>@<IP_address>:/<BeagleBone_filepath>

Наприклад:

$scp ~/development/server bbb@192.168.10.100:/tmp/

У випадку відсутності такої можливості, нам необхідно мати фізичний доступ до плати, та завантажити виконуваний файл через sd-карту.

При приєднанні карти за допомогою кард-рідера до комп’ютера, файлова система плати маунтиться на /media/<user_name>/<rootfs_partition_name>, куди ми можемо скопіювати наш

виконуваний файл скориставшись правами суперкористувача наступним чином:

$sudo scp ~/development/server /media/ivan/rootfs

Попередньо варто створити окрему директорію на платі для кожного компонента нашого великого проєкта tcp, http серверів та файлів, які вони надаватимуть, сканера вайфай мережі.

Переходимо на плату та запускаємо сервер та бачимо від нього повідомлення, як на рисунку 3.18.

```
/ # ./server
Waiting for a connection...
Connection from 192.168.0.109 client on port 53766 is accepted!
Message transfer request
Received message from client: Hello from the client!
Closing connection!

Waiting for a connection...
```

Рисунок 3.18 – Запуск сервера.

На клієнті запускаємо клієнтську частину, запуск якої зображено на рисунку 3.19

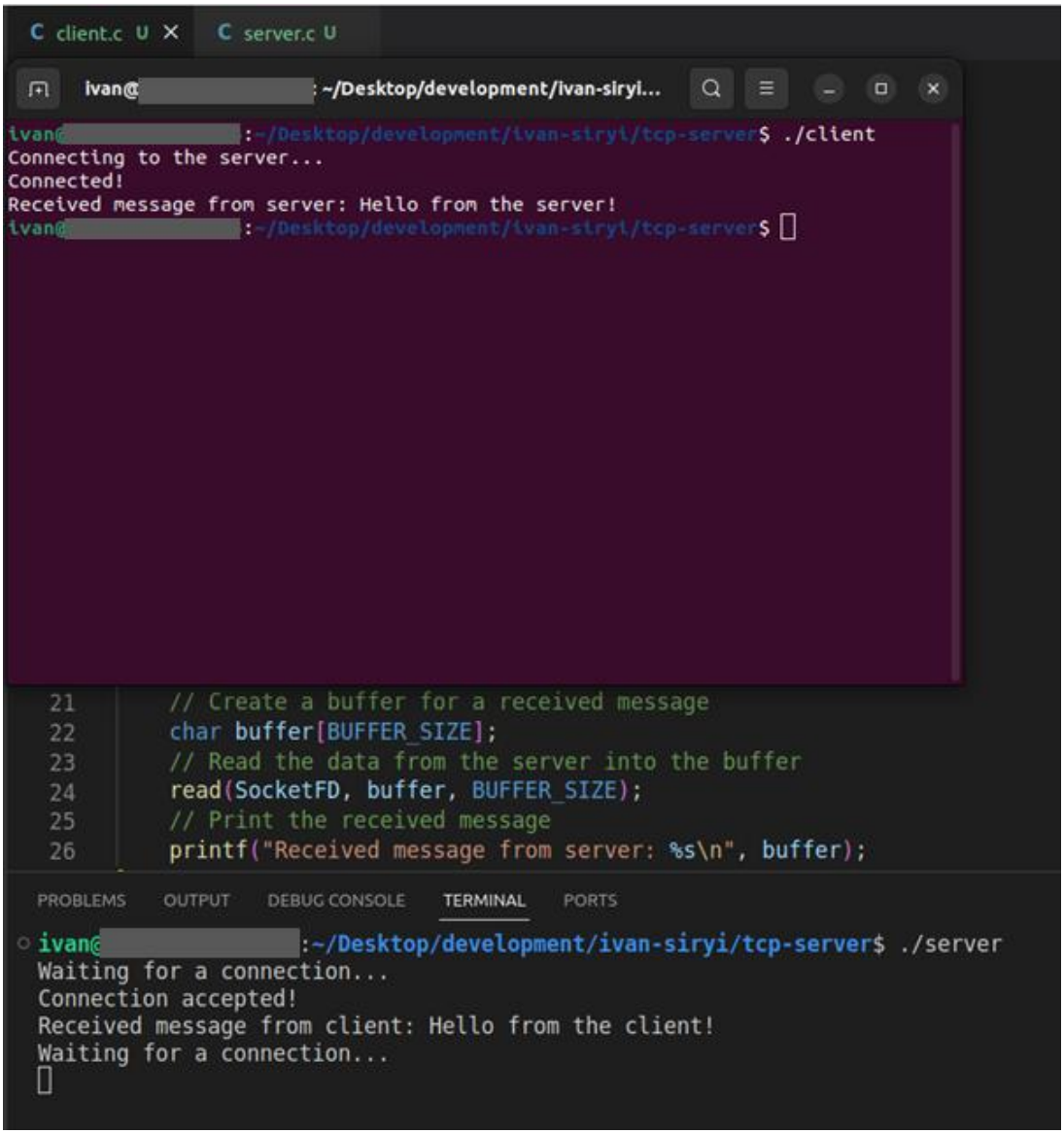

Рисунок 3.19 – Запуск клієнтського коду.

та бачимо повідомлення про успішне завершення з’єдання.

3.1.5 Передача файлів

Відійдемо від тривіальної задачі з передачі одиночних повідомлень між клієнтом та сервером та спробуємо передавати файли. Ця задача не є нашою кінцевою метою, а тому виставимо певні обмеження. Клієнт запитуватиме файл за його назвою у сервера, який зберігатиме цей файл. Запитуваний файл передаватиметься в директорію, звідки відбувався запуск клієнтського коду. Клієнт не посилатиме файли серверу, також залишимо можливість обміну повідомленнями, розроблену раніше.

Для цього наприклад, навантажимо TCP пакет, який відправлятимемо з клієнта наступним чином, як показано на рисунку 3.20 .

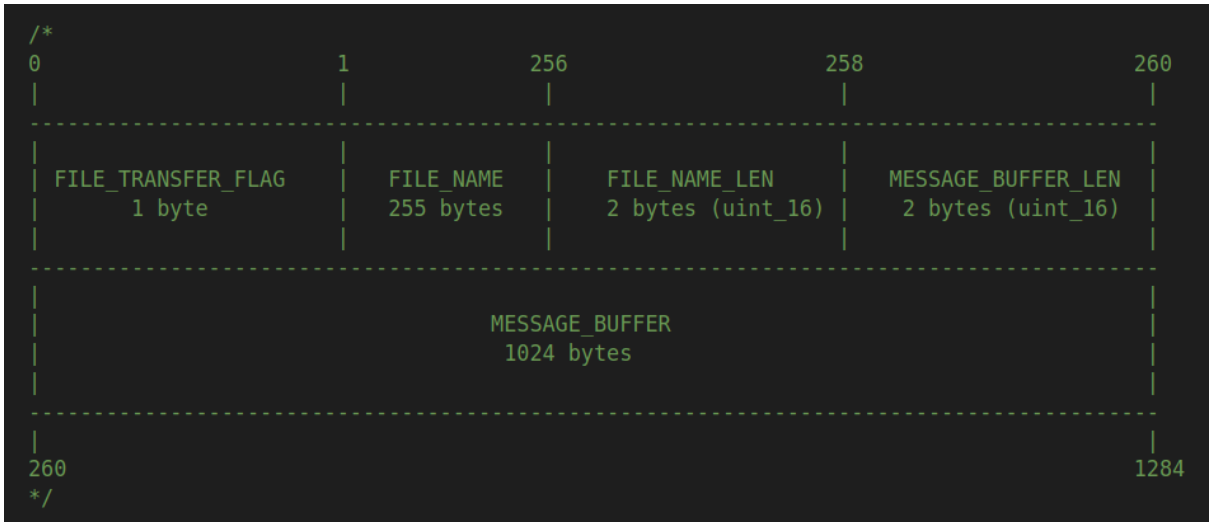

Рисунок 3.20 – Формат повідомлення.

В одному буфері ми передаватимемо різнотипну інформацію, а тому нам треба явно зазначити межі та тип цієї інформації, аби сервер зміг однозначно інтерпретувати наш запит. Так як ми лишаємо можливість обміну повідомленнями, введемо прапорець FILE_TRANSFER_FLAG, який відповідатиме за те, що саме повинен зробити сервер із нашим запитом: зчитати назву запитуваного файла та відправити його нам, або ж прочитати повідомлення клієнта, та відповісти на нього своїм. Максимальна довжина імені файла в операційній системі Linux 255 байтів, тому виділяємо саме такий розмір для розмішення імені запитуваного файла.

Обмежимо зчитування розміру імені файла сервером, аби сервер намагався знайти саме потрібний нам файл, якщо буфер міститиме “сміття” після імені файла. Хоча 255 байтів можна описати за допомогою цілого числа розміро 1 байт, вирівняємо наш буфер додавши один лишній байт для FILE_NAME_LEN області буфера. Так само обмежимо зчитування області, що містить повідомлення.

Перейдемо до реалізації запланованого навантаження. Для цього нам знадобиться сформувати запит на клієнтській частині, розпарсити його на серверній, додати логіку запису в файл і читання з нього. Режим з’єднання -

обмін повідомленнями чи передача файла, реалізуємо за допомогою параметрів запуску клієнтського коду.

```
char *query_file;

void form_request(char *buffer, bool file_transfer)
{
    if (file_transfer)
    {
        buffer[FILE_TRANSFER_BYTE] = (uint8_t)1;
        strncpy(&buffer[FILE_NAME_BYTE], query_file, FILE_NAME_SIZE - 1); // Copy FILE_NAME_SIZE - 1 characters
        buffer[FILE_NAME_BYTE + FILE_NAME_SIZE - 1] = '\0'; // Null-terminate the string
        buffer[FILE_NAME_LEN_BYTE] = (uint16_t)strlen(query_file);
    }
    else
    {
        buffer[FILE_TRANSFER_BYTE] = (uint8_t)0;
        strncpy(&buffer[MESSAGE_BUFFER_BYTE], MESSAGE, MESSAGE_BUFFER_SIZE - 1);
        buffer[MESSAGE_BUFFER_BYTE + MESSAGE_BUFFER_SIZE - 1] = '\0'; // Null-terminate the string
        buffer[MESSAGE_BUFFER_LEN_BYTE] = (uint16_t)strlen(MESSAGE);
    }
}
```

Рисунок 3.21 – Формування повідомлення.

Сформуємо запит наступним чином, як зображено на рисунку 3.21. Прапорець file_transfer відповідатиме за режим з’єднання, тобто визначатиме які байти нашого буфера необхідно заповнити наданими даними. Його значення визначатиметься на початку виконання коду як показано на рисунку 3.22.

```
int main(int argc, char *argv[])
{
    bool file_transfer = false;
    if (argc > 3)
    {
        printf("Usage:\n\t%s\n\t%s -f <query_file>\n", argv[0], argv[0]);
        exit(EXIT_FAILURE);
    }
    else if (argc == 3 && strcmp(argv[1], "-f") == 0)
    {
        file_transfer = true;
        query_file = argv[2];
    }
```

Рисунок 3.22 – Обробка аргументів запуску.

Для запиту файла, нам необхідно надати прапорець “-f” та назву файла при запуску клієнтського коду. Обмін повідомленнями відбувається при виклику виконуваного файла без аргументів. Використання клієнтської частини має наступний вигляд:

Usage:

./client

./client -f <query_file>

Логіка з’єднання виглядатиме як показано на рисунку 3.23. Формуємо наш запит та відправляємо його серверу. Натомість очікуємо на дві різні поведінки в залежності від сформованого запиту: відкриємо файл та запишемо в нього надісланий сервером вміст або ж сприймемо дані надіслані сервером як просте повідомлення та виведемо його в консоль.

```
void handle_connection(int SocketFD, char *request_buffer, bool file_transfer)
{
    // Create a buffer for a received message
    char buffer[BUFFER_SIZE];
    form_request(request_buffer, file_transfer);
    write(SocketFD, request_buffer, REQUEST_BUFFER_SIZE);

    if (file_transfer)
    {
        // Open the file
        FILE *file = fopen(query_file, "wb");
        if (file == NULL)
        {
            perror("Cannot open file");
            fprintf(stderr, "File: %s\n", query_file);
        }
        // read the buffer sent from server in cycle and write it to file
        int n;
        while ((n = read(SocketFD, buffer, BUFFER_SIZE)) > 0)
        {
            fwrite(buffer, sizeof(char), n, file);
        }
        // Close the file
        fclose(file);
    }
    else
    {
        // Read the data from the server into the buffer
        read(SocketFD, buffer, BUFFER_SIZE);
        // Print the received message
        printf("Received message from server: %s\n", buffer);
    }
}
```

Рисунок 3.23 – Логіка з’єднання.

На сервері розпарсимо запит в створену структуру запиту як зазначено на рисунку 3.24. В залежності від встановелного клієнтом прапорця,

заповнимо потрібні для успішного виконання клієнтського запиту поля структури.

```
// Create a buffer for a received message
char request_buffer[REQUEST_BUFFER_SIZE];

typedef struct Request
{
    uint8_t file_transfer_flag;
    char file_name[FILE_NAME_SIZE];
    uint16_t file_name_len;
    uint16_t message_buffer_len;
    char message_buffer[MESSAGE_BUFFER_SIZE];
} Request;

void parse_request(Request* request)
{
    // parse request_buffer
    // get file transfer flag
    request->file_transfer_flag = request_buffer[FILE_TRANSFER_BYTE];
    if (request->file_transfer_flag == 1)
    {
        // get file name len
        request->file_name_len = request_buffer[FILE_NAME_LEN_BYTE];
        // get file name
        strncpy(request->file_name, &request_buffer[FILE_NAME_BYTE], request->file_name_len);
    }
    else
    {
        // get message buffer len
        request->message_buffer_len = request_buffer[MESSAGE_BUFFER_LEN_BYTE];
        // get message buffer
        strncpy(request->message_buffer, &request_buffer[MESSAGE_BUFFER_BYTE], request->message_buffer_len);
    }
}
```

Рисунок 3.24 – Парсинг запиту.

Щоб надіслати файл відкриємо його на читання та відправлятимемо його у циклі частинами, які рівні або менше створеного для цього буфера, допоки матимемо що відправляти. Це можна зробити за допомогою створення умови для виконання циклу while() у вигляді функції fread().

За допомогою все того ж прапорця file_transfer_flag визначимо поведінку сервера: або відправимо файл, який в нас запитує клієнт, або обмежимось повідомленням. Не забувайте звільняти виділену динамічно пам’ять та закривати відкриті файли.

Ми готові до тестування. Скомпілюємо вихідний код та перенесемо серверну частину на плату. На платі створимо директорію для зберігання файлів, які ми передаватимемо клієнту. Клієнт нам надсилає запит на відправку файла, та зазначає відносний шлях. На сервері цей шлях перетворюється в абсолютний шлях до директорії, де зберігатимуться файли (який визначається нами), до якого додаємо відносний шлях наданий

клієнтом. При такому підході постає проблема доступу до файлів на сервері не обмежуючись даною директорією, бо операційна система Linux дозволяє нам формувати абсолютний шлях разом з додаванням символів крапка, які у вигляді “../” дозволяють отримати доступ до директорій вище на будь-яку кількість рівнів до кореневого каталогу шляхом об’єднання поданої комбінації символів у послідовність. Наразі ми лишимо цю проблему невирішеною. Заборону проходу по файловій системі додамо в наступному підрозділі, коли реалізовуватимемо HTTP сервер.

Наповнимо файл, який передаватимемо, наступним вмістом:
“This text is written to be transferred through a TCP connection in a file transfer operation and displayed using Wireshark.”

Рисунок 3.25 – Клієнт, сервер та пакети, якими вони обмінялись.

Запустимо сервер, під’єднаємо клієнта, а також поглянемо на пакети (рисунок 3.25), що містять наш сформований запит від клієнта та відповідь у вигляді вмісту файла від сервера за допомогою програми Wireshark.

3.1.6 HTTP сервер

Наступне завдання, яке поставлене перед нами, - розробка HTTP сервера. HTTP є протоколом прикладного рівня, який побудований на основі, або ж інкапсулюється в TCP - протокол транспортного рівня. Аби перетворити серверну частину програмного коду на таку, яка б вміла обмінюватись за протоколом HTTP нам треба, приймати надіслані клієнтом пакети, за формальними ознаками визначати, що надіслані запити є саме HTTP запитами, розпарсити запит, та надати на нього відповідь. Виходить, що у попередньому розділі, виконуючи завдання розширення функціональних можливостей класичного TCP сервера, ми виконували дуже схожі кроки. Лише, в тому випадку, наш функціонал не можна було означити “стандартним”, а тому не дивно, що для спілкування вигаданою “мовою”, або ж за вигаданим “протоколом”, нам необхідно було мати власні специфічні як клієнт, так і сервер. Навряд, хтось зміг би стати нашим клієнтом, окрім нас самих.

Тепер же, нашим завданням буде слугувати “загальному” клієнтові. Окрім того, нам варто звернути увагу на вирішення наступних задач: сервер має обслуговувати якусь осяжну кількість клієнтів одночасно, тобто бути багатопоточним, зберігати і оперувати контекстом кожного зі з’єднань окремо, також, варто вирішити обмежити доступ до файлової системи, окрім наданої директорії.

За формування запитів відповідатимуть браузер, утиліти, такі як wget, curl і подібні. Наше завдання навчитись на них відповідати. Обмежимось наданням відповідей на “GET” запити. Такі запити направлені на отримання вмісту ресурсу, який вказано в URI такого запиту. Для формування повного стартового рядка пакету HTTP не вистачає лише вказання версії протоколу. Він матиме наступний вигляд:

GET resourse_URI HTTP/version

Версія протоколу відповідає за те, яким стандартам має відповідати відповідь на наш запит, а також про можливість підтримки заголовків. Заголовки йдуть після стартового рядка і містять додаткову інформацію для сервера. Вони є опціональними, тому для нашого випадку ми можемо вільно вирішувати про імплементацію функціональних особливостей для обробки тих чи інших типів заголовків.

Маємо наступну послідовність дій:

- ініціалізація сервера (створення сокета, усіх необхідних ресурсів, наприклад, пулу потоків для одночасного підключення клієнтів);
- виділення певного потоку для конкретного з’єднання, створення контексту цього з’єднання;
- прийняття запиту;
- парсинг запиту;
- обробка запиту;
- формування і надсилання відповіді (стартового рядка і заголовків) і надання ресурсу у випадку здатності успішно обробити запит;
- закриття даного з’єднання.

Із додаткових задач варто реалізувати логування роботи сервера, для можливості аналізу коректності його роботи, статус-коди відповідей, обробку заголовків, можливість змінювати конфігурацію для ініціалізації сервера (порт, документ за замовчуванням, директорія, що містить ресурси, логування та його рівні, і таке інше), шляхом застосування заданих параметрів у вигляді файлу.

```c
int handleRequest(ConnectionArgs* arguments, char* clientIP)
{
    Request request;
    Response response;
    initResponse(&response);
    int status_code = 0;

    // receive client request
    LOG_DEBUG("Receiving request from client %s...", clientIP);
    char request_buffer[BUFFER_SIZE];
    status_code = receiveRequest(arguments->context, request_buffer, BUFFER_SIZE);
    if (status_code == INTERNAL_ERROR)
    {
        LOG_ERROR("Internal error block");
        response.statusCode = InternalServerError;
        response.contentLength = 0;
        char* msg = getStatusCodeMsg(InternalServerError);
        strncpy(response.statusText, msg, MAX_STATUS_LENGTH);
        strncpy(request.path, INTERNAL_ERROR_HTML, PATH_LENGTH);
        strncpy(request.type, "GET", TYPE_LENGTH);
        free(msg);
    }
    else if (status_code == CLIENT_CLOSED_CONNECTION)
    {
        return CLIENT_CLOSED_CONNECTION;
    }
    else
    {
        LOG_INFO("Request received: \n%s", request_buffer);
        LOG_DEBUG("Parsing request line...");
        status_code = parseRequestLine(arguments->server, &request, request_buffer, BUFFER_SIZE);
        if (status_code == BAD_REQUEST_ERROR)
        {
            LOG_ERROR("Bad request block");
            response.statusCode = BadRequest;
            char* msg = getStatusCodeMsg(BadRequest);
            strncpy(response.statusText, msg, MAX_STATUS_LENGTH);
            strncpy(request.path, BAD_REQUEST_HTML, PATH_LENGTH);
            strncpy(request.type, "GET", TYPE_LENGTH);
            free(msg);
        }
        else
        {
            char* msg = getStatusCodeMsg(OK);
            strncpy(response.statusText, msg, MAX_STATUS_LENGTH);
            free(msg);
            LOG_DEBUG("Parsing request fields...");
            parseRequestFields(&request, request_buffer);
        }
        LOG_INFO("Request: type=%s, uri=%s", request.type, request.path);
        if (request.keepAlive)
        {
            arguments->context->timeout = KEEP_ALIVE_TIMEOUT;
            arguments->context->max_requests = KEEP_ALIVE_MAX;
        }
    }

    LOG_DEBUG("Sending response...");
    sendResponse(arguments->server, arguments->context, &response, &request);
    LOG_DEBUG("Response sent...");
    return EXIT_SUCCESS;
}
```

Рисунок 3.26 – Обробка з’єднання.

Рисунок 3.26 зображує функцію, відповідальну за з’єднання. Вона займається отриманням запиту, його обробкою при його некоректності,

парсингом першого рядка та заголовків, відправкою відповіді сервера, в залежності від запиту та його коректності.

Вирішення задачі заборони проходження файловою системою може виглядати, як показано на рисунку 3.27. Таким чином ми не станемо відповідати на запити, що містять послідовність “../” символів для переходу файловою системою на рівень вище.

```
bool checkDirectoryTraversal(char* path)
{
    int i = 0;
    do
    {
        if (path[i] == '.' && path[i + 1] == '.' && path[i + 2] == '/')
        {
            LOG_INFO("Directory traversal detected");
            return true;
        }
    } while (path[i++] != '\0');

    return false;
}
```

Рисунок 3.27 – Заборона проходження файловою системою.

Файл нашим сервером відправлятиметься так само, як і у попередньому розділі, лише перед його відправкою, ми маємо сформувати пакет, який містить перший рядок (HTTP/version Code Status) та заголовки, такі як “Content-Type” та “Content-Length”, в яких ми повідомлятимемо кліжнта про тип та довжину ресурса, який надаємо. Наприклад:

HTTP/1.1 200 OK

Content-Length: 123

Content-Type: text/html

Створимо HTML сторінку та переконаємося в спроможності нашого сервера віддавати html сторінку при підключенні.

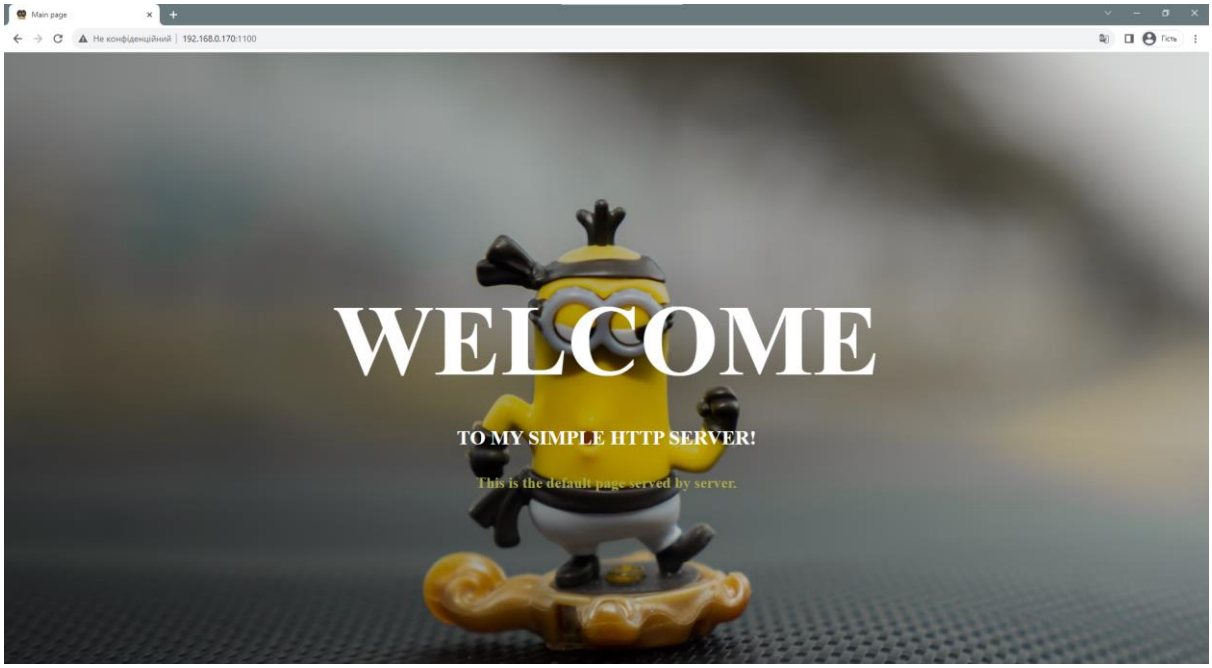

Рисунок 3.28 – Основна сторінка сервера.

Після запуску, підключимося до сервера через інтернет браузер та отримаємо результат як показано на рисунку 3.28.

3.1.7 Сканер WiFi

За допомогою вже закріпленої концепції сокетів створимо програму, яка скануватиме всі доступні WiFi мережі.

Таку програму реалізуємо за допомогою бібліотеки мови C Netlink, яка використовується для передачі інформації між ядром та процесами середовища користувача. Бібліотека складається зі стандартного інтерфейсу на основі сокетів для процесів простору користувача та внутрішнього API ядра для модулів ядра.

Створення і розгортання сокета є схожим до сокетів бібліотеки <socket.h>, яку ми використовували для вирішення попередніх задач. Матимемо наступну послідовність дій:

- створимо (виділимо) новий Netlink сокет;
- приєднаємось до сокета на стороні ядра;
- надамо сокету ядра інформацію про сімейство нашого сокета;

- згенеруємо наше повідомлення-запит до ядра та відправимо його;
- імплементуємо кол-бек функцію для прийняття, парсингу та обробки відповіді ядра;
- зареєструємо кол-бек функцію для відкритого нами сокета;
- очікуватимемо на відповідь.

Ми маємо запитувати у ядра саме ту інформацію, яка нам потрібна. Для цього необхідно правильно формувати заголовок запиту як показано на рисунку 3.29. Нам достатньо вказати отриманий раніше і записаний у змінну ідентифікатор сімейства сокетів “nl80211”, ідентифікатор команди для сканування вайфай мереж, опціонально, вкажемо прапорець, який забов’яже видавати нам повну таблицю з усіма підходящими нашому запиту записами. Додаткову інформацію можна отримати за допомогою виклику $man netlink.

```
// Add Generic Netlink headers to Netlink message

// Parameters:
// msg      | Netlink message object
// port     | Netlink port or NL_AUTO_PORT
// seq      | Sequence number of message or NL_AUTO_SEQ
// family   | Numeric family identifier
// hdrlen   | Length of user header
// flags    | Additional Netlink message flags (optional)
// cmd      | Numeric command identifier
// version  | Interface version
genlmsg_put(msg, 0, 0, family_id, 0, NLM_F_DUMP, NL80211_CMD_GET_SCAN, 0);
```

Рисунок 3.29 – Заголовок запиту.

Щоб отримати результати сканування, нам попередньо необхідно ініціювати або ж стригерити сканування WiFi мереж. Тому перед відправкою згенерованого вище повідомлення нам треба сформувати та відправити наступне, як показано на рисунку 3.30.

```c
int buildTriggerScanMessage(struct nl_sock *nl_socket, struct nl_msg *msg, int if_index, int family_id)
{
    // Setup the message
    genlmsg_put(msg, 0, 0, family_id, 0, 0, NL80211_CMD_TRIGGER_SCAN, 0);
    nla_put_u32(msg, NL80211_ATTR_IFINDEX, if_index);

    // Add SSIDs to scan
    struct nl_msg *ssid_nested = nlmsg_alloc();
    if (!ssid_nested)
    {
        printf("ERROR: Failed to allocate netlink message for ssid_nested.\n");
        return -ENOMEM;
    }
    nla_put(ssid_nested, 1, 0, "");
    nla_put_nested(msg, NL80211_ATTR_SCAN_SSIDS, ssid_nested);
    nlmsg_free(ssid_nested);

    return EXIT_SUCCESS;
}
```

Рисунок 3.30 – Створення тригер-повідомлення.

У бездротових мережах підсистема базових станцій (Basic Service Set, BSS) представляє групу станцій (пристроїв, таких як ноутбуки, смартфони тощо), які спільно використовують точку доступу. На рисунку 3.31 ми парсимо відповідь отриману від ядра. За допомогою наданих API функцій ми запишемо всі атребути в таблицю атримбутів, після чого з цих атрибутів виділимо BSS атрибути, які в подальшому будемо самостійно парсити в потрібні нам за функціоналом характеристики мереж.

```c
int parseMessage(struct nl_msg *msg, void *arg)
{
    struct genlmsghdr *genlmsg_header = nlmsg_data(nlmsg_hdr(msg));
    Network *network = (Network *)arg;

    if (nla_parse(attr_table, NL80211_ATTR_MAX, genlmsg_attrdata(genlmsg_header, 0),
                  genlmsg_attrlen(genlmsg_header, 0), NULL) < 0)
    {
        perror("Cannot parse kernel message");
        return NL_SKIP;
    }

    if (!attr_table[NL80211_ATTR_BSS])
    {
        perror("Did not find NL80211_ATTR_BSS");
        return NL_SKIP;
    }

    nla_parse_nested(bss_table, NL80211_BSS_MAX, attr_table[NL80211_ATTR_BSS], NULL);

    parseIE(network);
    printNetworkInfo(network);

    return NL_SKIP;
}
```

Рисунок 3.31 – Парсинг повідомлення.

Коли ви скануєте мережі Wi-Fi, ваша мережева карта шукає маяки (beacon повідомлення), надіслані найближчими точками доступу. Кожен маяк представляє BSS. Результати сканування включають різноманітну інформацію BSS, як-от назва мережі (Service Set ID, SSID), MAC-адреса (Basic Service Set Identifier, BSSID), рівень сигналу та протоколи безпеки.

Ці дані називаються інформаційними елементами — це блоки даних різного розміру. Кожен блок даних позначається номером типу та розміром, і визначено, що інформаційний елемент певного типу має своє поле даних, яке інтерпретується певним чином. Вичерпну інформацію надає Метью Гаст в своїй книзі про мережі 802.11 [24].

Як приклад, зображений на рисунку 3.32, приведемо функцію парсингу SSID. Пройдемося за допомогою арифметики вказівників по інформаційним елементам, знаючи індекс SSID, зупинимось на ньому, отримаємо його довжину, та запишемо отрману назву мережі в структуру, яка описує знайдену мережу.

```
static int parseSSID(Network *network)
{
    const uint8_t *data = nla_data(bss_table[NL80211_BSS_INFORMATION_ELEMENTS]);
    const size_t data_len = nla_len(bss_table[NL80211_BSS_INFORMATION_ELEMENTS]);
    printf("data_len: %d\n", (int)data_len);

    const uint8_t *ssid = NULL;

    for (const uint8_t *ptr = data; ptr < data + data_len; ptr += IE_HEADER_SIZE + ptr[LENGTH_INDEX])
    {
        if (ptr[ID_INDEX] == IE_SSID_IE)
        {
            ssid = ptr;
            break;
        }
    }

    if (ssid == NULL)
        return EINVAL;

    const size_t ssid_len = ssid[LENGTH_INDEX]; // length of SSID from header
    if (ssid_len >= sizeof(network->ssid))
        return EINVAL;

    memcpy(network->ssid, ssid + IE_HEADER_SIZE, ssid_len);
    network->ssid[ssid_len] = '\0';

    return EXIT_SUCCESS;
}
```

Рисунок 3.32 – Парсинг назви мережі.

Подібним чином розпарсимо й інші характеристики мережі, які нас цікавлять, виведемо результати в консоль та продовжимо сканування в циклі із заданим інтервалом.

Рисунок 3.33 – WiFi модуль.

На рисунку 3.33 зображено WiFi модуль. Такий чи подібний необхідно приєднати до плати для забезпечення сканування доступних мереж.

```
NL80211_CMD_TRIGGER_SCAN sent 36 bytes to the kernel.
Waiting for scan to complete...
Got NL80211_CMD_NEW_SCAN_RESULTS.
Scan is done.

IY Network                      50:64:2b:4c:9a:95       2437            -65     48              WPA1, WPA2      6
WLAN-23D5E5                     44:d4:53:b0:f3:34       2462            -54     54              WPA2            11
TP-LINK_E91C_5G                 50:c7:bf:23:e9:1a       5180            -65     54              WPA2            36
WLAN-23D5E5                     44:d4:53:b0:f3:33       5220            -43     54              WPA2            44
Gaming-23D5E5                   4e:d4:53:b0:f3:33       5220            -43     54              WPA2            44
TP-LINK_E91C                    50:c7:bf:23:e9:1b       2412            -66     48              WPA2            1
FERRARY                         34:60:f9:89:1f:f8       2422            -67     48              WPA2            3

NL80211_CMD_TRIGGER_SCAN sent 36 bytes to the kernel.
Waiting for scan to complete...
Got NL80211_CMD_NEW_SCAN_RESULTS.
Scan is done.

IY Network                      50:64:2b:4c:9a:95       2437            -72     48              WPA1, WPA2      6
WLAN-23D5E5                     44:d4:53:b0:f3:34       2462            -49     54              WPA2            11
TP-LINK_E91C_5G                 50:c7:bf:23:e9:1a       5180            -56     54              WPA2            36
WLAN-23D5E5                     44:d4:53:b0:f3:33       5220            -40     54              WPA2            44
Gaming-23D5E5                   4e:d4:53:b0:f3:33       5220            -39     54              WPA2            44
TP-LINK_E91C                    50:c7:bf:23:e9:1b       2412            -56     48              WPA2            1
FERRARY                         34:60:f9:89:1f:f8       2422            -66     48              WPA2            3

NL80211_CMD_TRIGGER_SCAN sent 36 bytes to the kernel.
Waiting for scan to complete...
Got NL80211_CMD_NEW_SCAN_RESULTS.
Scan is done.

IY Network                      50:64:2b:4c:9a:95       2437            -65     48              WPA1, WPA2      6
WLAN-23D5E5                     44:d4:53:b0:f3:34       2462            -50     54              WPA2            11
TP-LINK_E91C_5G                 50:c7:bf:23:e9:1a       5180            -64     54              WPA2            36
WLAN-23D5E5                     44:d4:53:b0:f3:33       5220            -46     54              WPA2            44
Gaming-23D5E5                   4e:d4:53:b0:f3:33       5220            -46     54              WPA2            44
TP-LINK_E91C                    50:c7:bf:23:e9:1b       2412            -67     48              WPA2            1
FERRARY                         34:60:f9:89:1f:f8       2422            -78     48              WPA2            3

NL80211_CMD_TRIGGER_SCAN sent 36 bytes to the kernel.
Waiting for scan to complete...
Got NL80211_CMD_NEW_SCAN_RESULTS.
Scan is done.
```

Рисунок 3.34 – Створення тригер-повідомлення.

Рисунок 3.34 зображає вивід розробленого сканера мереж, а саме: SSID, BSSID, частоту, RSSI, максимальну швидкість, протокол захисту та канал роботи.

3.1.8 Відображення отриманих даних через HTTP сервер

Приведемо консольний вигляд результату сканування доступних мереж до вигляду інтернет сторінки та транслюватимемо його через розроблений нами попередньо HTTP сервер.

Для цього перетворимо вивід сканера на дані у вигляді файлу формату JSON, де кожна мережа матиме наступний вигляд, як показано на рисунку 3.35.

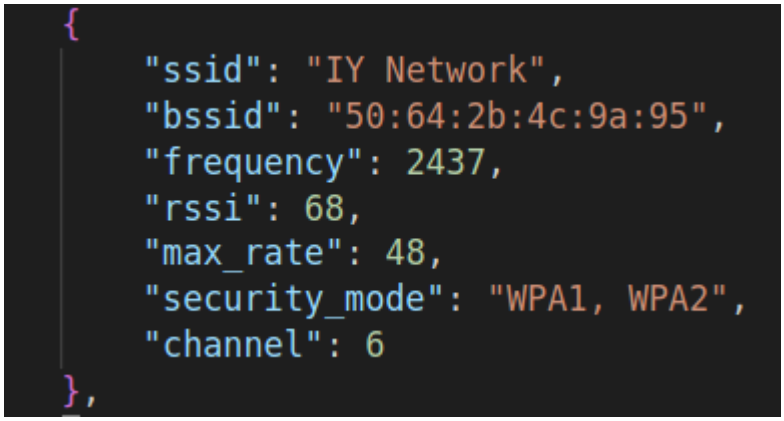

Рисунок 3.35 – Отримані дані в JSON форматі.

HTTP сервер видаватиме нам запитуваний ресурс лише після запиту, але в нашому випадку ресурс перестав бути статичним. Тому, за допомогою простого скрипта, зображеного на рисунку 3.36, написаного мовою JavaScript та прикріпленого до HTML сторінки, що відображає інформацію про знайдені мережі, оновлюватимемо за інтервалом наші дані.

```
let refresh = setInterval(function () {
fetch("networks.json")
    .then(res => res.json())
    .then(function (datas) {
        let placeholder = document.querySelector("#table-networks")
        let output = ""
        for (let data of datas) {
            output += `
        <tr>
            <td>${data.ssid}</td>
            <td>${data.bssid}</td>
            <td>${data.frequency}</td>
            <td>${data.rssi}</td>
            <td>${data.max_rate}</td>
            <td>${data.security_mode}</td>
            <td>${data.channel}</td>
        </tr>
        `
        }
        placeholder.innerHTML = output;
    }

    )
}, 1000)
```

Рисунок 3.36 – Скрипт для отримання актуальних даних сканування.

Запустимо сервер, запитаємо через браузер HTML сторінку, яка надасть нам інформацію про мережі.

SSID	BSSID	FREQUENCY	RSSI	MAX RATE	SECURITY	CHANNEL
TP-LINK_E91C	50:c7:bf:23:e9:1b	2412	67	48	WPA2	1
FERRARY	34:60:f9:89:1f:f8	2422	67	48	WPA2	3
IY Network	50:64:2b:4c:9a:95	2437	70	48	WPA1, WPA2	6
WLAN-23D5E5	44:d4:53:b0:f3:34	2462	51	54	WPA2	11
TP-LINK_E91C_5G	50:c7:bf:23:e9:1a	5180	64	54	WPA2	36
WLAN-23D5E5	44:d4:53:b0:f3:33	5220	46	54	WPA2	44
Gaming-23D5E5	4e:d4:53:b0:f3:33	5220	46	54	WPA2	44
IY Nerwork_5G	50:64:2b:4c:9a:96	5220	79	54	WPA1, WPA2	44

Рисунок 3.37 – Список доступних мереж.

Отримаємо актуальний список доступних мереж — результат роботи сканера, як показано на рисунку 3.37.

Висновки до розділу 3

У даному розділі було детально розглянуто процес розробки програмного забезпечення для Edge сервера, акцентуючи увагу на використанні TCP протоколу для забезпечення надійного з'єднання між сервером і клієнтами. Висвітлено ключові аспекти створення базового TCP сервера та клієнта, включаючи налагодження з'єднань та передачу вітальних повідомлень.

Далі, розглянуто еволюцію програмного забезпечення через імплементацію можливості передачі файлів, перетворення серверної частини на HTTP сервер і розробку клієнтської частини, що використовує веб-браузер. Описано розробку програмного забезпечення для сканування мереж WiFi за допомогою Netlink сокетів, та спілкування програм середовища користувача із ядром операційної системи. В подальшому, було описано спосіб представлення динамічно оновлюваних даних, які надає сканер засобами для візуалізації через HTTP сервер.

Визначено ключові етапи розробки та інтеграції різних компонентів програмного забезпечення, що дозволяють ефективно використовувати мережеві ресурси, забезпечувати гнучкість та масштабованість системи. Також зазначено важливість впровадження TCP протоколу для надійності з'єднань між сервером та клієнтами, що є критично важливим для Edge обчислень.

У підсумку, розглянуто переваги інтеграції сканера WiFi та відображення даних через веб-інтерфейс, що значно розширює функціональні можливості системи. Підкреслено, що розроблене програмне забезпечення відіграє ключову роль у сфері Edge обчислень, забезпечуючи високу надійність, масштабованість і гнучкість, необхідні для різноманітних застосувань у сучасному цифровому світі.

РОЗДІЛ 4

РОЗРОБКА СТАРТАП-ПРОЕКТУ

У сучасному світі інформаційних технологій, постійно зростаюча роль IoT систем визначає нові вимоги до ефективності та безпеки мережевих рішень. Наш стартап, орієнтований на програмне забезпечення для Edge серверів для IoT систем, представляє інноваційний комплекс, що складається з низки ключових компонентів, розглянутих у попередньому розділі. В рамках курсу споживачу вдасться пройти від простого до комплексного, зупиняючись на важливих аспектах побудови програмно-апаратного забезпечення для подібних систем. Цей стартап є відповіддю на потреби сучасного ринку в високоефективних та надійних рішеннях для IoT систем, забезпечуючи нові можливості для навчання побудові систем моніторингу у складних мережевих середовищах.

4.1 Маркетинговий аналіз стартап-проекту

Основна ідея проєкту полягає в представленні начального курсу з розробки мережевого програмного забезпечення для Edge систем у вигляді плати BeagleBone Black. Курс має на меті ознайомлення із сокетним програмуванням для операційної системи Linux, а саме TCP клієнта та сервера, односторонньої передачі запитуваних файлів, на базі чого — HTTP сервера, який слугує постачальником моніторингової інформації наданої, в нашому випадку, сканером мереж WiFi, що демонструє характеристики усіх доступних безпровідних мереж.

Таблиця 4.1.

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Начальний курс з розробки мережевого програмного забезпечення для Edge систем	Підготовка слухачів в області Embedded мережевого програмування	Поглиблення та розширення знань та вмінь
	Представлення основи для розробки власного програмного забезпечення чи її вдосконалення	Можливість втілювати власні задумки та масштабувати їх

Проведемо аналіз потенційних техніко-економічних переваг ідеї та визначимо, чим ідея відрізняється від існуючих аналогів та замінників. А саме:

- визначимо перелік техніко-економічних властивостей та характеристик ідеї;
- визначимо попередні кола проектів-конкурентів, що вже існують на ринку та проведемо збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів;
- проведемо порівняльний аналіз показників: для власної ідеї визначимо показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні).

Таблиця 4.2.

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Проекти				W	N	S
		Даний	1	2	3			
1.	Сокетне програмування для Linux	+	+	+	-		+	
2.	Доступна плата BeagleBone Black	+	-	-	-			+
3.	Застосування на різних платформах	+	+	-	+		+	
4.	Масштабованість	+	+	-	+		+	
5.	Гнучкість системи	+	+	-	+		+	
6.	Простота подачі інформації	+	+	-	-			+
7.	Забезпечення надійності системи	+	-	-	+			+
8.	Високий рівень безпеки	-	-	+	+	+		

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності. Згідно з таблицею 4.2, серед основних переваг розробленого продукту є простота подачі інформації слухачам курсу, доступність апаратних можливостей курсу, програмне забезпечення в рамках курсу має не лише зрозумілі функціональні особливості, а й звертається увага на надійність таких систем. Порівняно з існуючими рішеннями, головними недоліками системи є недостатній безпековий рівень системи, що потребує доопрацювання в майбутньому.

4.2 Технологічний аудит ідеї проекту

Таблиця 4.3.

Технологічна здійсненність ідеї проекту

№	Ідея	Технології реалізації	Наявність технологій	Доступність технологій
1	Начальний курс з розробки мережевого ПЗ для Edge систем на BeagleBone Black	Linux	Широко доступні та відомі технології	Висока доступність, використання відкритих джерел та загальнодоступного обладнання
2		TCP/IP		
3		HTTP сервер		
4		WiFi аналізатори		
5		Сокетне програмування		

Згідно з описаними технологіями, необхідними для реалізації проекту, створення і розробка системи конструювання трафіка в SDN за допомого лямбда-архітектури є можливим. Усі необхідні технології є широко доступними та відомими технології, при цьому потребують глибокого розуміння процесів взаємодії в мережі та в межах операційної системи. WiFi аналізатори є доступними у вигляді утиліт для ОС Linux, але самостійна реалізація потрібує кропіткого дослідження та пошуку інформації в мережі інтернет серед документації та публікацій.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Здійснимо визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть

перешкодити реалізації проекту. Цей етап дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. У таблиці 4.4 наведена попередня характеристика потенційного ринку для розроблюваного стартап-проекту.

Таблиця 4.4.

Попередня характеристика потенційного ринку стартап-проекту

№	**Показники стану ринку (найменування)**	**Характеристика**
1	Кількість головних гравців на ринку	Декілька великих компаній, а також численні малі та середні підприємства
2	Загальний обсяг продаж, грн/ум.од.	Високий, з урахуванням зростаючого попиту на Edge технології
3	Динаміка ринку (якісна оцінка)	Зростаючий, очікується збільшення попиту на освітні курси та розробки в цій сфері
4	Наявність обмежень для входу (характер обмежень)	Високі технологічні бар'єри, потреба в спеціалізованих знаннях і навичках
5	Специфічні вимоги до стандартизації та сертифікації	Відповідність міжнародним стандартам, можливі вимоги сертифікації продуктів та послуг
6	Середня норма рентабельності в галузі або по ринку, %	Залежить від специфіки продукту/послуги та ринкової ніші

За результатами аналізу таблиці ринок є привабливим для входження по причинах достатнього рівня рентабельності, має високу конкурентність та зростає.

Подальший аналіз ринкових можливостей потребує характеристики потенційних груп клієнтів. Характеристика та орієнтовний перелік вимог до продукту стартап-проекту представлені в таблиці 4.5.

Таблиця 4.5.

Характеристика потенційних клієнтів стартап-проекту

№	**Потреба, що формує ринок**	**Цільова аудиторія**	**Відмінності у поведінці різних потенційних цільових груп клієнтів**	**Вимоги до споживачів продукту**
1	Потреба в освітніх ресурсах для розробників мережевих рішень	Студенти технічних спеціальностей, молоді фахівці в IT	Студенти можуть шукати базове розуміння технологій, фахівці - глибше та практичне застосування знань	Базові знання в області програмування та мережевих технологій
2	Попит на кваліфікованих фахівців у сфері Edge обчислень	IT компанії, навчальні заклади	IT компанії зацікавлені в підвищенні кваліфікації своїх співробітників, навчальні заклади - в	Готовність інвестувати в освіту та розвиток співробітників

			оновленні навчальних програм	
3	Інтерес до розробки мережевих рішень на базі BeagleBone Black	Хобісти, DIY-ентузіасти	Хобісти шукають доступні і гнучкі рішення для експериментів та навчання	Технічний інтерес та бажання експериментувати з новими технологіями

Проведемо аналіз ринкового середовища, а саме: складемо таблиці факторів, що сприяють ринковому впровадженню проекту (таблиця 4.7.), та факторів, що йому перешкоджають (таблиця 4.6.).

Таблиця 4.6.

Фактори загроз

№	**Фактор**	**Зміст загрози**	**Можлива реакція компанії**
1	Технологічні зміни	Швидкий розвиток технологій може зробити проект застарілим	Адаптація до нових технологій, постійне оновлення курсу
2	Конкуренція	Наявність сильних конкурентів на ринку	Розробка унікальних особливостей та переваг курсу
3	Залучення клієнтів	Труднощі у залученні та утриманні студентів	Ефективний маркетинг та залучення через партнерства

Таблиця 4.7.

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Розвиток ринку IoT	Збільшення попиту на спеціалістів у галузі IoT	Розширення та актуалізація курсу, фокус на IoT
2	Партнерства з навчальними закладами	Співпраця з університетами та коледжами	Проведення семінарів та майстер-класів у навчальних закладах
3	Цифровізація освіти	Перехід до онлайн-форматів навчання	Розвиток онлайн-платформи та курсів для дистанційного навчання

Далі проведемо аналіз пропозиції: визначемо риси конкуренції на ринку (таблиця 4.8.).

Таблиця 4.8.

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
Різноманітність продуктів	Наявність різних рішень від простих до складних	Розробка унікальних та інноваційних рішень для виділення на ринку
Цінова конкуренція	Наявність продуктів з різними ціновими діапазонами	Встановлення конкурентних цін та пропонування додаткової вартості

Технологічні інновації	Швидкий розвиток технологій у галузі	Неперервне оновлення та удосконалення продукту
Репутація бренду	Стійкість та впізнаваність брендів конкурентів	Розробка сильної маркетингової стратегії та бренду

Проведемо більш детальний аналіз умов конкуренції в галузі за моделлю 5 сил М. Портера.

М. Портер вирізняє п’ять основних факторів, що впливають на привабливість вибору ринку з огляду на характер конкуренції. Це:

- конкуренти, що вже є в галузі;
- потенційні конкуренти;
- наявність товарів-замінників;
- постачальники, що конкурують за ринкову владу;
- споживачі (аналогічно).

Сильні позиції продукту за кожним з факторів означають її можливість забезпечити необхідні темпи обороту капіталу та її здатність впливати на інших агентів ринку, диктуючи їм власні умови співпраці. Характеристики факторів моделі відрізняються для різних галузей та змінюються з часом.

Таблиця 4.9.

Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційн конкурент в галузі	Постачаль-ники	Клієнти	Продукти-замінники
Складові аналізу	Існуючі компанії, що пропонують подібні курси та рішення	Нові стартапи та компанії з суміжних галузей, які можуть	Виробники та дистриб'ютори обладнання та програмного забезпечення	Студенти, фахівці в IT, навчальні заклади	Онлайн платформи для самоосвіти, безкоштов ні

		вийти на ринок			навчальні ресурси
Висновки	Присутня конкуренція на міжнародному ринку.	Компанії займаються розробкою та підтримкою рішень і можуть випустити схожий продукт	Постачальниками є міжнародними лідерами, показник залучення є низьким	Більшість клієнтів давно присутні на ринку і мають контакти з постачальниками та можуть бути зацікавлені у продукті	Є декілька подібних рішення для конструювання трафіка, які є платними та покривають частину функціоналу

За результатами таблиці робимо висновок, що робота на ринку з огляду на ситуацію є можливою, конкуренція на даному етапі є, постачальники не зацікавлені у наданні подібних рішень, а існуючі проекти покривають лише частину запропонованого функціоналу.

На основі аналізу конкуренції (таблиця 4.9.) а також із урахуванням характеристик ідеї проекту (таблиця 4.2.), вимог споживачів до товару (таблиця 4.5.) та факторів маркетингового середовища (таблиці 4.6.-4.7.) визначимо та обґрунтуємо перелік факторів конкурентоспроможності (таблиця 4.10.).

Таблиця 4.10.

Обґрунтування факторів конкурентоспроможності

№	**Фактор конкурентоспроможності**	**Обґрунтування**

1	Інноваційність технологій	Використання новітніх технологій забезпечує перевагу в швидкозмінному технічному середовищі, приваблюючи клієнтів бажаючих залишатися на передньому краї інновацій.
2	Якість та доступність навчання	Висока якість курсу, поєднана з доступністю, залучає більш широку аудиторію, включаючи студентів і професіоналів з різним бюджетом.
3	Адаптація до потреб ринку	Гнучкість та здатність швидко адаптуватися до мінливих потреб ринку робить проект більш привабливим для інвесторів та клієнтів.
4	Партнерські відносини	Розвиток міцних відносин з партнерами та клієнтами сприяє довгостроковому успіху та лояльності клієнтів.
5	Маркетингова стратегія	Ефективна маркетингова стратегія та промоція забезпечують високу видимість проекту на ринку, залучаючи нових клієнтів і підтримуючи інтерес до продукту.
6	Репутація та довіра	Розбудова сильної репутації та довіри серед клієнтів та партнерів сприяє залученню нових клієнтів та утриманню існуючих.

За визначеними факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап-проекту.

Таблиця 4.11.

Порівняльний аналіз сильних та слабких сторін стартап-проекту

№	Фактор конкурентоспроможності	Бали (1-20)	Рейтинг товарів-конкурентів у порівнянні з продуктом (-3 до +3)
1	Інноваційність технологій	17	+2
2	Якість навчання	15	+1
3	Цінова конкурентоспроможність	12	0
4	Маркетингова стратегія	14	-1
5	Розвиток та підтримка клієнтів	16	+1
6	Адаптація до ринкових умов	13	+2
7	Технічна підтримка та обслуговування	11	-2

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 4.12.) на основі виділених ринкових загроз та можливостей, а також сильних та слабких сторін.

Таблиця 4.12.

SWOT- аналіз стартап-проекту

Сильні сторони:	**Слабкі сторони:**
1. Інноваційний підхід до навчання та розробки. 2. Використання передових технологій.	1. Обмежений досвід на ринку. 2. Високі технологічні бар'єри для входу. 3. Потенційно високі витрати на

3. Гнучкість та адаптивність до потреб ринку.	розвиток та дослідження.
Можливості: 1. Зростаючий попит на Edge технології. 2. Потреба в кваліфікованих фахівцях в цій галузі. 3. Розширення ринку через технологічні інновації.	**Загрози:** 1. Швидкий розвиток технологій може зробити продукт застарілим. 2. Сильна конкуренція на ринку. 3. Зміни у технологічних стандартах та нормативах.

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок (таблиця 4.13.)

Таблиця 4.13.

Альтернативи ринковоговпровадження стартап-проекту

№	**Альтернатива ринкової поведінки**	**Ймовірність отримання ресурсів**	**Терміни реалізації**
1	Стартап через краудфандинг	Середня до висока	6-12 місяців
2	Співпраця з великими IT-компаніями	Висока (залежить від умов договору)	12-18 місяців
3	Самостійний вихід на ринок з мінімально життєздатним продуктом (MVP)	Низька до середньої	3-6 місяців
4	Розробка та реалізація через акселератори стартапів	Середня	6-12 місяців

4.4 Розробка ринкової стратегії стартап-проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 4.14).

Таблиця 4.14.

Вибір цільових груп потенційних споживачів

№	**Опис профілю цільової групи потенційних клієнтів**	**Готовність споживачі в сприйняти продукт**	**Орієнтовний попит в межах цільової групи**	**Інтенсивність конкуренції в сегменті**	**Простота входу в сегмент**
1	Студенти та молоді фахівці в сфері IT	Висока	Середній	Висока	Середня
2	IT компанії, зацікавлені у підвищенні кваліфікації співробітників	Середня	Високий	Середня	Висока
3	Навчальні заклади, що шукають сучасні навчальні програми	Низька до середньої	Середній	Низька	Низька

4	Хобісти та DIY-ентузіасти	Висока	Низький	Середня	Висока

Згідно з аналізом потенційних комерційних груп споживачів на запропонований продукт, було обрано стратегію диференційованого маркетингу, оскільки компанія працює водночас із кількома сегментами, з розробкою окремих програм для ринкового впливу.

Таблиця 4.15.

Визначення базової стратегії розвитку

№	**Обрана альтернатива розвитку проекту**	**Стратегія охоплення ринку**	**Ключові конкурентоспроможні позиції**	**Базова стратегія розвитку**
1	Розробка та надання навчальних курсів з мережевого програмування для Edge систем	Фокус на нішевих сегментах ринку, таких як освітні заклади та IT компанії	Висока якість та актуальність навчального матеріалу, практичний досвід	Поступове зростання ринкової частки через партнерства та акцент на якості
2	Створення платформи для онлайн-навчання	Онлайн-продаж та маркетинг, партнерства з онлайн-платформами	Інноваційність та доступність, зручність онлайн-формату	Розширення ринкової присутності через онлайн-канали

3	Проведення воркшопів та майстер-класів	Організація мероприятий та співпраця з освітніми закладами	Експертність та практичний досвід, сильна освітня програма	Розвиток бренду через мережу партнерів та організацію заходів

Таблиця 4.16.

Визначення базової стратегії конкурентної поведінки

№	**Чи є проект «першопрохідцем» на ринку?**	**Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?**	**Чи буде компанія копіювати основні характеристики товару конкурента, і які?**	**Стратегія конкурентної поведінки**
1	Ні, вже є конкуренти на ринку	Шукати нових споживачів, пропонуючи унікальні особливості	Ні, пропонувати інноваційні рішення, які виділяються на тлі конкурентів	Диференціація продукту, фокус на унікальних характеристиках та інноваціях

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та вимог до продукту (табл. 4.5.), а також в залежності від обраної базової стратегії розвитку (табл. 4.15) та стратегії конкурентної поведінки (табл. 4.16) розробляється стратегія позиціонування (табл. 4.17). що

полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17.

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції	Вибір асоціацій
1	Актуальність, інноваційність, доступність	Розвиток інноваційних навчальних курсів	Висока якість, передові технології, унікальний досвід	Лідерство у сфері освіти, передові технічні рішення
2	Практична значимість, гнучкість	Адаптація курсу до ринкових потреб	Ефективне навчання, практичне застосування знань	Висока практична цінність, орієнтація на результат

У результаті отримали узгоджену систему рішень щодо ринкової поведінки стартап-компанії, що визначатиме напрями роботи стартап-компанії на ринку.

4.5 Розробка маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції продукту, який отримає споживач (таблиця 4.18.).

Таблиця 4.18.

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Актуальні знання в галузі Edge обчислень	Надання практичних та актуальних знань, які застосовуються в реальних проектах	Інноваційний підхід до навчання, зосередженість на практичному застосуванні знань
2	Потреба у фахівцях з мережевих технологій	Глибоке розуміння мережевих протоколів та програмування	Унікальність курсу, забезпечення глибоких та спеціалізованих знань
3	Необхідність у доступних навчальних матеріалах	Доступність та ефективність навчання	Конкурентоспроможні ціни, висока якість м

Далі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

Таблиця 4.19.

Опис трьох рівнів моделі товару

Рівні	Сутність і складові
Основний товар (Core Product)	Це основна вигода або потреба, яку задовольняє продукт. Для стартап-проекту з розробки мережевого програмного забезпечення для Edge систем, основний товар полягає у наданні знань та навичок в області мережевих технологій.
Фактичний товар (Actual Product)	Включає фізичні або видимі характеристики товару. Сюди

	входять сам курс, його зміст, матеріали, платформа для навчання, методика викладання, викладачі, та інші особливості, які відрізняють курс від конкурентів.
Розширений товар (Augmented Product)	Включає додаткові послуги та вигоди, які перевищують очікування клієнтів. Це може бути післякурсова підтримка, доступ до спільноти випускників, можливості для мережевого зв'язку, кар'єрне консультування тощо.

Таблиця 4.20.

Визначення меж встановлення ціни

№	**Рівень цін на товари-замінники**	**Рівень цін на товари-аналоги**	**Рівень доходів цільової групи споживачів**	**Верхня та нижня межі становлення ціни**
1	Середній	Середній або вищий	Середній до високий	Верхня межа: вища за середній рівень товарів-аналогів з урахуванням доданої вартості; Нижня межа: вище або на рівні товарів-замінників

Таблиця 4.21.

Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Пошук якісних та доступних освітніх ресурсів	Інформування, просування, забезпечення доступності товару	Прямий (онлайн курси, вебінари)	Онлайн-платформи, співпраця з освітніми закладами
2	Інтерес до інноваційних та практичних рішень	Підтримка клієнтів, управління замовленнями	Непрямий (через партнерів, дистриб'юторів)	Комбінація прямих та непрямих каналів збуту
3	Потреба у фахівцях з глибокими технічними знаннями	Післяпродажний сервіс, технічна підтримка	Прямий та непрямий	Персоналізовані навчальні сесії, партнерства з ІТ-компаніями

Таблиця 4.22.

Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій	Ключові позиції для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення

1	Шукають якісне та доступне навчання	Соціальні мережі, освітні платформи, форуми	Інноваційність, практичність, доступність	Інформування про унікальність курсу, його практичне застосування	Створення історій успіху, відео з відгуками студентів
2	Інтерес до нових технологій	Фахові вебсайти, блоги, вебінари	Експертність, глибина знань	Показати авторитетність курсу в галузі	Експертні статті, відео-уроки з презентацією курсу
3	Потреба в підвищенні кваліфікації	Професійні мережі, освітні заходи	Новаторство, гнучкість	Підкреслити можливості для кар'єрного росту	Проведення воркшопів, публікація статей з кейсами

Висновки до розділу 4

Четвертий розділ присвячено розробці стартап-проекту на основі розробленої системи, що реалізує спосіб конструювання трафіка в програмно-конфігурованих мережах на основі лямбда-архітектури.

Для дослідження в рамках розробки стартап-проекту було виконано наступні завдання:

1. Наведено загальний опис ідеї стартап-проекту, описано функціонал, визначено основних конкурентів та порівняно функціонал існуючих продуктів для визначення переваг розроблюваного продукту над конкурентами.

2. Проаналізовано сучасні ринкові можливості для запуску проекту та успішного виходу на ринок. Виявлено можливості та загрози стартап проекту та сплановані дії щодо їх запобігання.

3. Порівняно перспективи запуску в порівнянні з конкурентами, встановлено план та ринкову стратегію розвитку продукту, визначено цільові групи та способи просування проекту.

4. Розроблено маркетингову програму для просування продукту на ринку, описано способи та основні канали збуту, визначно пріорітетні цільові групи та маркетингові повідомлення для розширення клієнтської бази.

В підсумку, отримано стартап-проект для запуску розробленого програмного продукту на базі дипломного проекту для виходу на ринок.

ВИСНОВКИ

Ця магістерська дисертація проводить всебічне дослідження Інтернету речей (IoT) та його застосування у сучасних системах з використанням Edge серверів, а також розробку програмного забезпечення та стартап-проекту, що базується на цих технологіях.

У першому розділі аналізуються сучасні IoT системи, зокрема їх роль у промисловій революції, їх вплив на розвиток смарт міст та автомобільної промисловості. Приділено увагу безпековим викликам, що є актуальними для IoT та Edge технологій.

Другий розділ присвячений детальному огляду матеріалів та методів, що використовуються у розробці IoT систем. Розглядається клієнт-серверна архітектура, програмування сокетів, а також специфіка розробки програмного забезпечення для плати BeagleBone Black, її завантаження, налаштування.

Третій розділ зосереджується на практичній частині дослідження — описі та розробці програмного забезпечення для Edge сервера. Детально розглянуті аспекти створення TCP з'єднань, роботи TCP сервера та клієнта, передачі файлів, реалізації HTTP сервера та інших важливих компонентів.

Четвертий розділ присвячений розробці стартап-проекту, який базується на IoT та Edge технологіях. У цьому розділі здійснено маркетинговий аналіз, технологічний аудит, аналіз ринкових можливостей та розробку маркетингової стратегії проекту.

В даній магістерські роботі було розглянуто розроблена методика сокетного програмування для Edge серверів для розширення знань предметної області та підвищення ефективності розроблення Embedded систем для систем IoT.

Запропонована методика сокетного програмування для Edge серверів впровадженя в навчальному процесі на кафедрі обчислювальної техніки факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» дисципліни.

Узагальнюючи проведене дослідження, можна зробити висновок, що інтеграція IoT та Edge технологій відіграє ключову роль у розвитку сучасних технологічних систем. Подальший розвиток та застосування цих технологій мають великий потенціал у багатьох сферах, включаючи смарт міста, автомобільну промисловість та інші ключові галузі. Розробка програмного забезпечення та стартап-проектів, що базуються на IoT та Edge, відкриває нові можливості для інновацій та розвитку бізнесу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What is the internet of things? | IBM. *IBM in Deutschland, Österreich und der Schweiz | IBM*. URL: https://www.ibm.com/topics/internet-of-things (date of access: 25.12.2023).
2. Технології інтернету речей. Навчальний посібник [Електронний ресурс]: навч. посіб. для студ. спеціальності 126 «Інформаційні системи та технології», спеціалізація «Інформаційне забезпечення робототехнічних систем» / Б. Ю. Жураковський, І.О. Зенів; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 12,5 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2021. – 271 с.
3. The 4th industrial revolution, industry 4.0, unfolding at hannover messe 2014. *automation.com*. URL: https://www.automation.com/en-us/articles/2014-1/the-4th-industrial-revolution-industry-40-unfoldin (date of access: 25.12.2023).
4. Калита П. Україна і четверта промислова революція: загрози та можливості. *Дзеркало тижня*. 2016. 18 листоп. URL: https://zn.ua/ukr/macrolevel/ukrayina-i-chetverta-promislova-revolyuciya-zagrozi-ta-mozhlivosti-_.html.
5. Application of edge computing-based information-centric networking in smart cities / H. s. salih et al. *Computer Communications*. 2023. URL: https://doi.org/10.1016/j.comcom.2023.09.003 (date of access: 25.12.2023).
6. A learning automata based edge resource allocation approach for IoT-enabled smart cities / S. Sahoo et al. *Digital communications and networks*. 2023. URL: https://doi.org/10.1016/j.dcan.2023.11.009 (date of access: 25.12.2023).
7. Smart city implementation based on Internet of Things integrated with optimization technology / R. V. Kolhe et al. *Measurement: sensors*. 2023. P. 100789. URL: https://doi.org/10.1016/j.measen.2023.100789 (date of access: 25.12.2023).

8. Smart cities and sustainable development goals (SDGs): a systematic literature review of co-benefits and trade-offs / A. Sharifi et al. *Cities*. 2024. Vol. 146. P. 104659. URL: https://doi.org/10.1016/j.cities.2023.104659 (date of access: 25.12.2023).
9. Mughal, A.A. (2018). Big Data in Automotive Industry. In: Sakr, S., Zomaya, A. (eds) Encyclopedia of Big Data Technologies. Springer, Cham. URL: https://doi.org/10.1007/978-3-319-63962-8_34-1
10. An optimal control strategy for emergency vehicle priority system in smart cities using edge computing and IOT sensors / P. Rosayyan et al. *Measurement: sensors*. 2023. Vol. 26. P. 100697. URL: https://doi.org/10.1016/j.measen.2023.100697 (date of access: 25.12.2023).
11. Multi task dynamic edge-end computing collaboration for urban Internet of Vehicles / S. Shao et al. *Computer networks*. 2023. P. 109690. URL: https://doi.org/10.1016/j.comnet.2023.109690 (date of access: 25.12.2023).
12. A survey of edge computing-based designs for IoT security / K. Sha et al. *Digital communications and networks*. 2020. Vol. 6, no. 2. P. 195–202. URL: https://doi.org/10.1016/j.dcan.2019.08.006 (date of access: 25.12.2023).
13. Gentry, Craig. *A fully homomorphic encryption scheme*. Stanford university, 2009. URL: https://crypto.stanford.edu/craig/craig-thesis.pdf
14. Pei, Dingyi, Arto Salomaa, and Cunsheng Ding. Chinese remainder theorem: applications in computing, coding, cryptography. World Scientific, 1996. URL: https://opus.govst.edu/cgi/viewcontent.cgi?article=1000&context=capstones_math

15. J. Feng, L. T. Yang, B. Ren, D. Zou, M. Dong and S. Zhang, "Tensor Recurrent Neural Network With Differential Privacy," in IEEE Transactions on Computers, doi: 10.1109/TC.2023.3236868. URL: https://ieeexplore.ieee.org/document/10081283
16. Dwork, C., ”Differential Privacy”. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds) Automata, Languages and Programming. ICALP 2006. Lecture Notes in Computer Science, vol 4052. Springer, Berlin, Heidelberg. URL: https://doi.org/10.1007/11787006_1
17. Differential privacy in edge computing-based smart city Applications:security issues, solutions and future directions / A. Yao et al. *Array*. 2023. P. 100293. URL: https://doi.org/10.1016/j.array.2023.100293 (date of access: 25.12.2023).
18. ANDREW S, TANENBAUM; DAVID J, WETHERALL. COMPUTER NETWORKS FIFTH EDITION. 2011.
19. Neira-Ayuso, Pablo, Rafael M. Gasca, and Laurent Lefevre. "Communicating between the kernel and user-space in Linux using Netlink sockets." *Software: Practice and Experience* 40.9 (2010): 797-810. URL: https://perso.ens-lyon.fr/laurent.lefevre/pdf/JS2010_Neira_Gasca_Lefevre.pdf
20. Сірий І. М. Навчальний програмно-апаратний комплекс для вбудованої системи на базі процесора ARM32 : Дипломний проєкт. Київ, 2022. 79 с. URL: https://ela.kpi.ua/bitstream/123456789/49864/1/Siryi_bakalavr.pdf.
21. Vaduva, Alexandru, Alex Gonzalez, and Chris Simmonds. *Linux: Embedded Development*. Packt Publishing Ltd, 2016 URL: https://learning.oreilly.com/library/view/embedded-linux-for/9781787124202/
22. Гер, В. М. Навчальний комплекс для IoT пристроїв на базі вбудованих систем : дипломний проект ... бакалавра : 123 Комп’ютерна інженерія / Гер Владислав Маратович. - Київ, 2021. - 98 с.

23. Protsenko S. BeagleBone black: platform bring-up with upstream components. Share & Discover Presentations | SlideShare. URL: https://www.slideshare.net/GlobalLogicUkraine/beaglebone-black-platform-bringup-with-upstream-components (date of access: 25.12.2023).
24. Gast, Matthew. *802.11 wireless networks: the definitive guide*. " O'Reilly Media, Inc.", 2005 URL: https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/