ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Автоматизована система конфігурації маршрутизатора на базі OpenWRT з використанням центрального конфігураційного файлу»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ _____
(підпис) *Данііл Лясковський*
Ім'я, ПРІЗВИЩЕ здобувача

Виконав: здобувач вищої освіти гр. КІД-41
Данііл ЛЯСКОВСЬКИЙ
Ім'я, ПРІЗВИЩЕ
Керівник: Ігор БУЧЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних Технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____Наталія ЛАЩЕВСЬКА
_____
«_____»_____2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лясковський Данііл Олександрович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Автоматизована система конфігурації маршрутизатора на базі OpenWRT з використанням центрального конфігураційного файлу
керівник кваліфікаційної роботи Ігор БУЧЕНКО ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026р. № 51
2. Строк подання кваліфікаційної роботи «8» травня 2026р.
3. Вхідні дані до кваліфікаційної роботи: науково-технічна література; операційна система OpenWrt; система конфігурації UCI; вебінтерфейс LuCI; формат обміну даними JSON; мова програмування Python; бібліотеки Python (Paramiko, zoneinfo, ipaddress); протокол SSH; мережеві сервіси та протоколи: DHCP, DNS; міжмережевий екран (firewall); бездротові технології Wi-Fi (IEEE 802.11); команди операційної системи Linux (iw list); методи валідації даних;
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Концепції автоматизації конфігурації маршрутизаторів
2. Аналіз та дослідження процесу конфігурації маршрутизаторів
3. Розробка автоматизованої системи конфігурації маршрутизатора на базі OpenWrt з використанням центрального конфігураційного файлу
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «20» лютого 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	14.04-18.04.2026	Виконано
2	Розробка конструктивної частини	19.04-23.04.2026	Виконано
3	Аналіз та обробка зібраних даних	24.04-27.04.2026	Виконано
4	Розробка теоретичної частини	28.04-01.05.2026	Виконано
5	Апробація методів та засобів	02.05-02.05.2026	Виконано
6	Розробка опису конфігуратора	03.05-04.05.2026	Виконано
7	Висновки за результатами аналізу	04.05-04.05.2026	Виконано
8	Розробка презентації	04.05-04.05.2026	Виконано
9	Оформлення пояснювальної записки	04.05-04.05.2026	Виконано
10	Здача в деканат	05.05-08.05.2026	Виконано
11	Передзахист	11.05-15.05.2026	Виконано

Здобувач вищої освіти _____ Даніїл ЛЯСКОВСЬКИЙ

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ігор БУЧЕНКО

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 58 стор., 50 рис., 37 джерел.

Мета роботи – Розробка автоматизованої системи конфігурації маршрутизатора на базі OpenWrt з використанням центрального конфігураційного файлу для спрощення конфігурації пристрою.

Об'єкт дослідження – Процес конфігурації мережевих маршрутизаторів на базі операційної системи OpenWrt.

Предмет дослідження – Методи та засоби автоматизації конфігурації маршрутизаторів.

Короткий зміст роботи: У кваліфікаційній роботі досліджено особливості налаштування та конфігурації маршрутизаторів під управлінням операційної системи OpenWRT. Виконано аналіз сучасних підходів до автоматизації конфігурації мережевих пристроїв. Розглянуто існуючі інструменти керування налаштуваннями пристрою та їхні обмеження, зокрема складність масштабування та залежність від конкретних платформ.

У роботі запропоновано підхід до автоматизації конфігурації маршрутизаторів із використанням JSON конфігураційного файлу. Розроблено програмний засіб мовою програмування Python, описано його архітектуру та принцип роботи. Проведено тестування розробленої системи на реальному обладнанні, оцінено її ефективність, зручність використання та можливості масштабування.

КЛЮЧОВІ СЛОВА: КОНФІГУРАЦІЯ, АВТОМАТИЗАЦІЯ, МАРШРУТИЗАТОР, OPENWRT, UCI, JSON, SSH, PYTHON, LINUX, СИСТЕМА UCI, ВІДДАЛЕНЕ НАЛАШТУВАННЯ, БЕЗДРОТОВА МЕРЕЖА, КОНФІГУРАЦІЙНИЙ ФАЙЛ, ВАЛІДАЦІЯ ДАНИХ, ІНТЕРПРЕТАЦІЯ КОНФІГУРАЦІЇ

ЗМІСТ

ВСТУП....8
1 КОНЦЕПЦІЇ АВТОМАТИЗАЦІЇ КОНФІГУРАЦІЇ МАРШРУТИЗАТОРІВ....9
1.1 Визначення поняття маршрутизатора, його використання в сучасних комп'ютерних мережах та недоліки вбудованого програмного забезпечення....9
1.2 Операційна система OpenWrt, її переваги, вбудовані методи та засоби конфігурації....12
1.3 Методи та засоби конфігурації різних класів маршрутизаторів, їх переваги та недоліки....20
1.4 Проблематика автоматизації процесу конфігурації роутерів, вибір методів та засобів реалізації системи....25
2 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРОЦЕСУ КОНФІГУРАЦІЇ МАРШРУТИЗАТОРІВ....32
2.1 Аналіз процесу налаштування пристрою OpenWrt за допомогою вбудованого інструменту UCI....32
2.2 Аналіз проблем ручної конфігурації маршрутизаторів та формування вимог до системи конфігурації....39
3 РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ КОНФІГУРАЦІЇ МАРШРУТИЗАТОРА НА БАЗІ OPENWRT....44
3.1 Загальна архітектура системи та структура конфігураційного файлу....44
3.2 Алгоритм функціонування системи та реалізація програмного забезпечення....50
3.3 Тестування розробленої системи конфігурації маршрутизаторів на реальному обладнанні....57
ВИСНОВКИ....65
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ....67
Додаток А. Зразковий конфігураційний файл JSON....70
Додаток Б. Вихідний код модуля wrtcfg.py....72
Додаток В. Вихідний код модуля validator.py....74
Додаток Г. Вихідний код модуля uci_builder.py....80
Додаток Д. Вихідний код модуля ssh_client.py....86
Додаток Е. Вихідний код модуля wifi_features_parser.py....87
Додаток Ж. Вихідний код модуля loader.py....89
Додаток И. Демонстраційні матеріали....90

ВСТУП

Зважаючи на стрімкий розвиток інформаційних технологій та необхідності створення умов для передачі даних між пристроями в мережах, ефективне налаштування мережевої інфраструктури набуває особливого значення. Маршрутизатори, вони ж роутери, є ключовими елементами комп'ютерних мереж різних масштабів. Від їх правильного налаштування залежать продуктивність, безпека та стабільність роботи мережі.

Операційна система OpenWrt є однією з поширених платформ для побудови мережевих рішень. Вона має широкі можливості для налаштування пристроїв під її керуванням, відкритий вихідний код, відкриту архітектуру системи, та складається з незалежних складових. Один з способів налаштування пристроїв під керуванням OpenWrt – використання системи UCI для налаштування через SSH або скрипти. Вона передбачає ручне налаштування через набори команд, або вебінтерфейс, що може бути складним завданням та неефективно при роботі з великою кількістю пристроїв.

Один із сучасних та перспективних підходів для вирішення цієї проблеми – використання централізованого конфігураційного файлу, який описує всі необхідні параметри пристрою у структурованому вигляді. Формат JSON є зручним та загальноприйнятим рішенням через свою простоту, читабельність та широку підтримку в різних мовах програмування. Автоматичне перетворення такого файлу у команди UCI та їх подальше застосування на пристрої через протокол SSH дозволяє значно спростити процес конфігурації.

Актуальність теми зумовлена необхідністю зменшення часу на налаштування пристроїв та зменшення кількості помилок обумовлених людським фактором. Розробка програмних засобів, які забезпечують автоматизоване налаштування та керування маршрутизаторами, є важливим завданням в галузі мережевих технологій.

1 КОНЦЕПЦІЇ АВТОМАТИЗАЦІЇ КОНФІГУРАЦІЇ МАРШРУТИЗАТОРІВ

1.1 Визначення поняття маршрутизатора, його використання в сучасних комп'ютерних мережах та недоліки вбудованого програмного забезпечення.

Роутер – це пристрій, що з'єднує дві або більше мереж чи підмереж з комутацією пакетів. Його основний функціонал: керування трафіком між цими мережами пересиланням пакетів даних за відповідними IP-адресами, та створення умов для використання підключення до Інтернету кільком пристроям [1].

Існують різні типи роутерів, але більшість з них передають пакети даних між локальною мережею (LAN) та глобальною мережею (WAN). Локальна мережа – це обмежена певною географічною територією група з'єднаних пристроїв. Така мережа здебільшого потребує одного роутеру [1].

Глобальна мережа – це велика мережа, яка охоплює значну географічну територію. Великим організаціям та корпораціям, які працюють у різних місцях в межах певної країни, потрібні окремі локальні мережі для кожного з цих місць, які потім з'єднуються між собою і утворюють глобальну мережу. Оскільки глобальна мережа охоплює велику територію, для її функціонування необхідні численні маршрутизатори та комутатори [1].

Виходячи з цього, наявність цих пристроїв в сучасних мережах є мінімальною необхідністю. Вони виконують важливу роль передачі трафіку між різними IP-мережами. В основному домашні мережі використовують приватну IP-мережу 192.168.1.1/24, де і відбувається локальний обмін даними. Для того щоб ці користувачі могли користуватись інтернетом – їм необхідний шлюз, який виконає запит за необхідною IP-адресою, та поверне пристрою що робив цей запит відповідь.

Інтернет-провайдери (ISP) – це компанія, що надає послуги підключення користувача до інтернету, розпізнає доменні імена, виконує маршрутизацію інтернет-трафіку та відповідає за підтримку мережевої інфраструктури, яка забезпечує функціонування інтернету[2].

Маршрутизація трафіку між різними IP-мережами є основним завданням роутерів. В більшості випадків підприємницькі рішення виконують саме цю роль, при чому мають велику кількість портів, можливості розширення функціоналу через додавання модулів, довгу підтримку програмного забезпечення, релевантні програмні та апаратні компоненти, фізичну витривалість та надійність.

Прикладом таких рішень може послугувати маршрутизатор Cisco C1101-4P (рис 1.1). Це одне з бюджетних рішень лінійки Cisco 1000 Series. Він має необхідні для бізнесу програмні та апаратні можливості: підтримка віддаленого менеджменту (Cisco vManage), малий форм-фактор, висока доступність, наявність micro-USB консольного порту для налаштувань пристрою, сумісність з іншими Cisco пристроями та сервісами [3].



Рисунок 1.1 – Маршрутизатор Cisco C1101-4P [3]

Якщо ж оглядати кінцеві, користувацькі маршрутизатори, апаратно вони виконують одночасно роль маршрутизатора, комутатора, та бездротової точки доступу. До програмних функцій більшості таких пристроїв відносяться: інтерфейс для налаштування пристрою, який в більшості представлений вебінтерфейсом, міжмережевий екран для фільтрації трафіку та захисту пристрою, система оновлення програмного забезпечення. Інші додаткові функції довільні та різняться в залежності від виробника.

Прикладом користувацького роутеру може послугувати Cudy WR3000H (рис 1.2). Він задовольняє більшість користувацьких потреб. До його функціоналу належать: підтримка сучасної версії Wi-Fi 6, використання технології MU-MIMO для бездротової мережі, вбудований 2.5 GbE WAN порт, середнє споживання електроенергії 5W, може працювати як VPN-сервер або клієнт, підтримка Cudy Mesh для об’єднання Wi-Fi пристроїв від Cudy в єдину бездротову мережу та віддалене, або локальне налаштування пристрою через Cudy App [4].



Рисунок 1.2 – Wi-Fi маршрутизатор Cudy WR3000H, вид ззаду [4]

Проблеми програмного забезпечення користувацьких маршрутизаторів. Роутери постачаються з вбудованим програмним забезпеченням, яке встановлене на нього з заводу. В більшості таких пристроїв воно є власницьким та має закритий вихідний код. Створення оновлень та виправлення вразливостей програмного забезпечення таких пристроїв виконується на власний розсуд компаній що виробляють роутери. З цього виникає одна з проблем – немає повної гарантії того, що в пристрої, що купує споживач, немає критичних вразливостей, які можуть бути навмисно залишені компанією виробником пристрою.

Також існує проблема того, що користувачі можуть мати обладнання яке втратило підтримку програмного забезпечення. Інша частина пристроїв що й досі підтримуються не будуть обов'язково отримувати виправлення одразу при оприлюдненні вразливостей. Застарілі маршрутизатори не будуть мати виправлень та захисту програмного забезпечення від вразливостей, а отже вважаються небезпечними.

Також, оскільки користувацькі маршрутизатори в більшості мають закрите програмне забезпечення, користувач має обмежений контроль над конфігурацією пристрою. Деякі пристрої можуть мати функції які доступні апаратно та програмно, але заблоковані на розсуд виробника.

Більшість користувацьких роутерів комплектуються з увімкнутими з заводу небезпечними, але простими для споживача налаштуваннями, такими як: стандартні SSID та паролі точок доступу згенеровані виробником, стандартний пароль конфігураційного вебінтерфейсу, увімкнутий WPS для Wi-Fi, UPnP, та інші. Також, роутери можуть збирати телеметрію або інші дані про використання та відправляти їх на сервер виробника пристрою, що зменшує рівень приватності користувача.

1.2 Операційна система OpenWrt, її переваги, вбудовані методи та засоби конфігурації

Для усунення проблем з непостійністю випуску оновлень, програмного обмеження функціоналу пристрою, неоднозначність щодо збору даних про використання існують різноманітні рішення з відкритим вихідним кодом. Прикладом таких є OpenWrt, який і взятий за основну для цієї роботи через свої переваги.

OpenWrt (рис. 1.3) – це дистрибутив GNU/Linux із широкими можливостями розширення, призначений для мережевих пристроїв (зазвичай бездротових маршрутизаторів). На відміну від більшості інших дистрибутивів, OpenWrt з самого початку розроблялась як повнофункціональна, легко модифікована операційна система для маршрутизатора. На практиці це означає, що користувач можете мати підтримку сучасного обладнання та виправлення вразливостей завдяки ядру Linux, яке є новішим, ніж у більшості інших дистрибутивів [5].

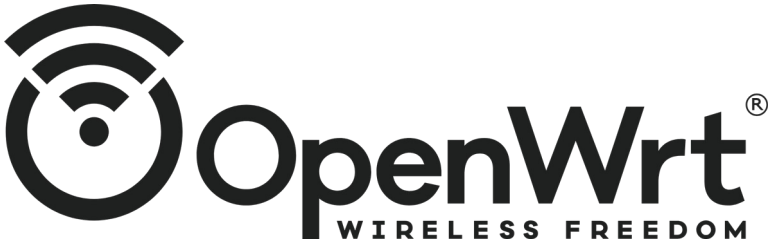


Рисунок 1.3 – Монохромний логотип OpenWRT [6]

Насправді, більшість операційних систем маршрутизаторів використовує модифіковане ядро Linux, до якого додаються всі необхідні на думку постачальника програмні рішення та модулі. Сюди також можуть включатись власницькі драйвери з закритим кодом. OpenWrt на противагу використовує відкритий код для програмного забезпечення, що дозволяє користувачам та розробникам цієї операційної системи виконувати поліпшення та виправлення коду, та пошуки вразливостей.

Одночасно це створює проблему того, що не все обладнання отримає підтримку OpenWrt, оскільки є апаратні платформи що мають закритий вихідний код, через що створення прошивки для них не передбачається можливим. Також є пристрої в яких недостатньо апаратних засобів, таких як оперативна пам’ять або вбудована пам’ять. Через це вони будуть працювати нестабільно через перевищення використання апаратних ресурсів, або не зможуть запустити операційну систему.

OpenWrt розроблено професіоналами в галузі мереж та іншими користувачами, які дбають про продуктивність мережі та прагнуть досягти максимальної пропускної здатності й мінімальної затримки. OpenWrt використовує численні алгоритми, розроблені в рамках останніх наукових досліджень, які демонструють значно кращі результати, ніж в програмному забезпеченні, що постачається виробниками [7].

До основних переваг OpenWrt над іншими вбудованими операційними системами входять:

− продуктивність та стабільність. OpenWrt є стабільною операційною системою і безпомилково працює протягом довгого часу. Оптимальна кодова база операційної системи адаптована для бюджетного обладнання та забезпечує максимум вільного простору вбудованої та оперативної пам’яті. Також в ній відсутні фонові процеси з операційної системи виробника пристрою, та зайві компоненти [7];

− безпека. В операційній системі відсутні приховані вразливості, залишені виробниками обладнання. Міжмережевий екран має точні налаштування для безпеки мережі від різноманітних атак. OpenWrt активно отримує оновлення, тому вразливості швидко виправляються після їх виявлення. Багато застарілих, не підтримуваних виробником пристроїв, досі підтримуються OpenWrt, та отримують сучасні версії програмного забезпечення та виправлення вразливостей. OpenWrt має сучасну кодову базу Linux та стандарти шифрування [7];

− розширюваність. Програмне забезпечення що надається виробником має фіксований, обмежений набір можливостей, в той час як OpenWrt надає приблизно 27,000 різноманітних пакетів та сервісів готових до використання та встановлення на пристрої. Приклади найкорисніших: Блокування реклами, підтримка OpenVPN та WireGuard клієнтів, підтримка гостьової мережі, батьківський контроль, шифрування DNS трафіку, підтримка DDNS, використання USB-порту пристрою для різних задач [7];

− поліпшений процес конфігурації. Однаковий та уніфікований інтерфейс між різними пристроями. Присутнє налаштування пристрою через командний рядок SSH (UCI) та вебінтерфейс (LuCI). Конфігураційна інформація зберігається як текстовий файл для полегшеного редагування та копіювання. OpenWrt надає генератор файлів прошивки, який дозволяє користувачу додавати в неї зміни (запрограмовані паролі, налаштування Wi-Fi, тощо). Присутні налаштування тегування VLAN, поліпшені налаштування Wi-Fi та IP-мереж [7].

Вбудовані методи та засоби конфігурації операційної системи OpenWrt.

Операційна система OpenWrt використовує наступні методи налаштування пристроїв: вебінтерфейс, командний рядок, конфігураційні файли. Пристрої можна налаштовувати виключно через SSH підключення. OpenWrt надає багато опцій щодо налаштування та керування SSH-сервером та вебінтерфейсом.

Єдиний інтерфейс конфігурації в OpenWrt представлений централізованою системою UCI (Unified Configuration Interface). Це основний конфігураційний користувацький інтерфейс для налаштування параметрів мережі. Також більшість пакетів операційної системи OpenWrt працюють на основі його конфігурацій. Всі файли налаштування UCI зберігаються за шляхом «/etc/config/». Файли, що зберігаються за цим шляхом, можна редагувати напряму текстовим редактором, або використовуючи консольну команду «uci» для взаємодії з ними[8].

Синтаксис налаштувань у конфігураційних файлах системи UCI відрізняється від синтаксису, що використовується при виконанні команд. Дані, що знаходяться в конфігураційних файлах, організовані у вигляді секцій, які можуть бути іменними, або безіменними. Кожній секції присвоюється відповідний тип та набір параметрів (рис 1.4). Параметри секцій задаються за допомогою ключового слова «option» та розміщуються після секції, до якої вони належать, із використанням відступів пробілами або табуляцією. Для забезпечення читабельності параметри записуються послідовно з однаковим форматуванням, а окремі секції відділяються одна від одної порожніми рядками [8].

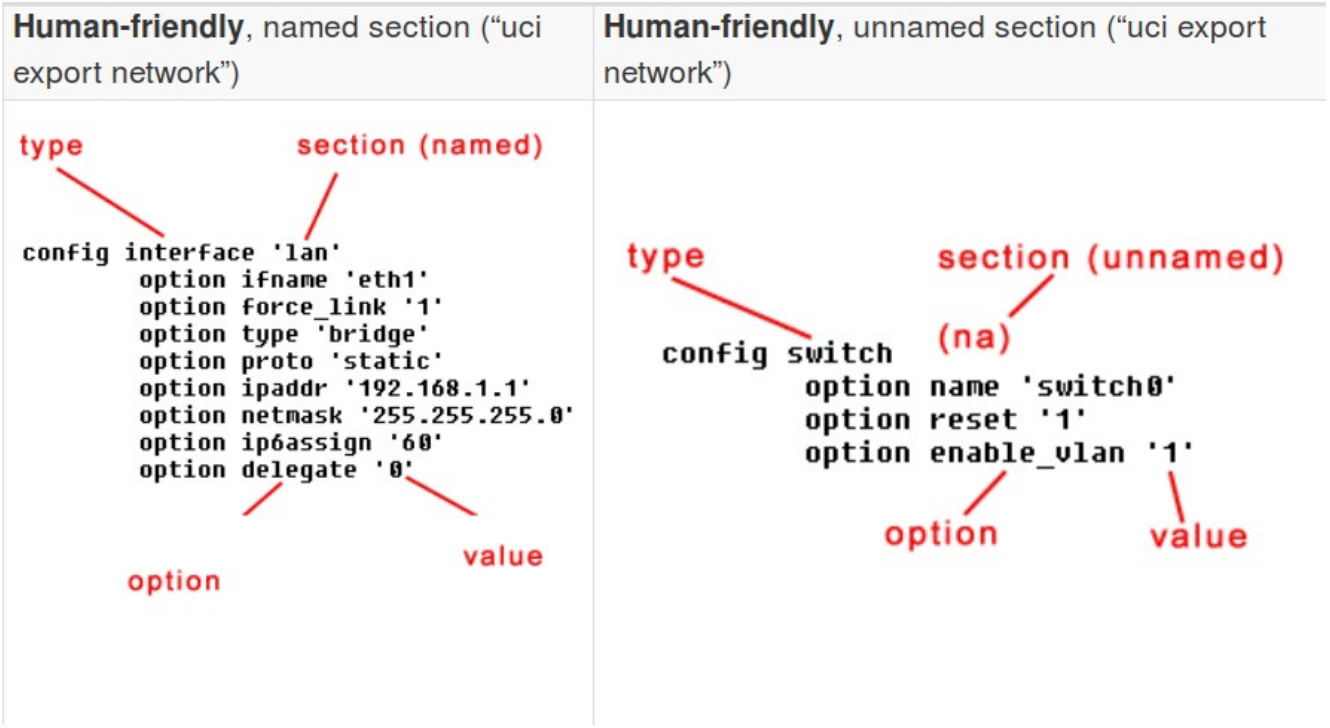


Рисунок 1.4 – Приклади іменних та безіменних секцій з присвоєними опціями в конфігурації «network» [8]

Під час використання команд системи UCI застосовується інший принцип запису параметрів. Команда формується у вигляді: «<назва конфігураційного файлу>.<секція>.[<опція>]=<значення>» (рис. 1.5). Секції можуть бути іменними або анонімними. Для доступу до безіменних секцій використовуються автоматично згенерований системою «CFGID», або запис «@<тип секції>[<номер>]». У такому випадку номер секції може бути як абсолютним, так і відносним. Наприклад, значення «-1» означає звернення до секції, що розташована перед останньою секцією відповідного типу у конфігураційному файлі [8].

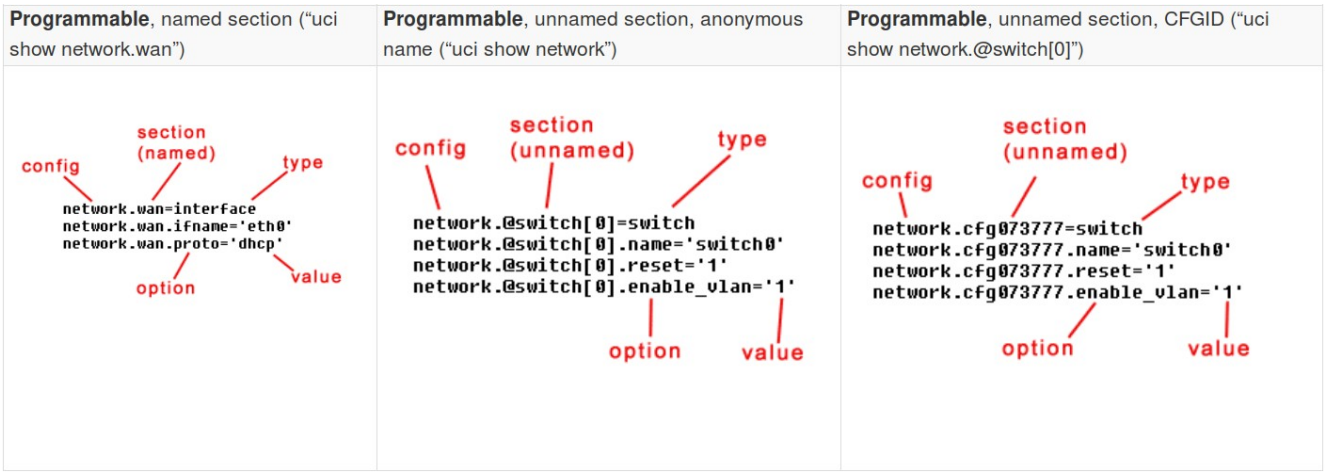


Рисунок 1.5 – Приклади формування аргументів до команд системи UCI для отримання доступу до іменної та безіменної секції конфігурації «network» [8]

Найголовніші конфігураційні файли базової системи UCI OpenWrt пристроїв та їх вміст[8]:

– «/etc/config/network» – інформація про мережеві пристрої, інтерфейси, VLAN, та подібна мережева інформація;

– «/etc/config/wireless» – конфігурації бездротової мережі, доступних пристроїв, налаштування точок доступу та подібне;

– «/etc/config/firewall» – налаштування міжмережевого екрану, його зон, правил передачі трафіку, переадресація портів, інші релевантні налаштування;

– «/etc/config/dhcp» – об'єднані конфігурації DNS та DHCP, що використовують програму Dnsmasq за основу;

– «/etc/config/dropbear» – налаштування програми Dropbear, яка є стандартним SSH-сервером та SSH-клієнтом пристроїв під керуванням OpenWrt;

– «/etc/config/system» – інші системні налаштування пристрою.

Кожен з перелічених конфігураційних файлів містить в собі секції, які мають присвоєний їм тип. В секціях містяться параметри з відповідними значеннями. До прикладу, в команді «uci set wireless.device0.channel='1'» «wireless» - це конфігураційний файл, «device0» - це секція, «channel» - це параметр, а «1» - присвоюване значення параметру [8].

LuCI (рис. 1.6) – це вебінтерфейс для керування OpenWrt, який працює поверх вбудованого вебсерверу та взаємодіє з системою конфігурації UCI [9]. Він однаковий на всіх пристроях OpenWrt в межах однієї версії, оскільки з новими версіями прошивки можуть вноситись косметичні зміни. Інтерфейс інтуїтивно зрозумілий та надає доступ до всіх можливих та необхідних функцій. Більше того, LuCI можна поширити за допомогою розширень, щоб, наприклад, можна було конфігурувати встановлені на пристрій пакети без використання командного рядку.

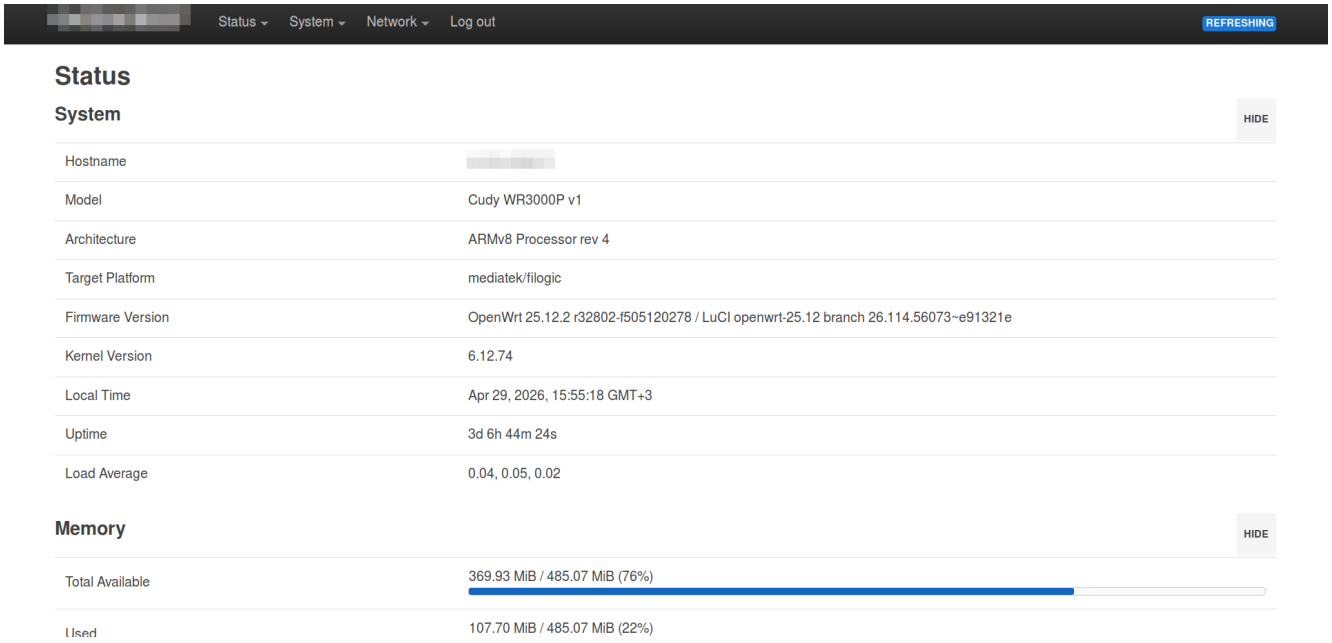


Рисунок 1.6 – Головна сторінка вебінтерфейсу LuCI

Цей інтерфейс дає користувачеві можливість внести налаштування, але не застосовує їх одразу. Таким чином, можна уникати проблем з помилковим застосуванням параметрів. Налаштування зберігаються в кеші, тому після перезавантаження сторінки, або налаштування з різних пристроїв, вони зберігаються. Зміни зроблені користувачем перетворюються в команди системи UCI (рис. 1.7), та застосовуються до пристрою тільки після підтвердження користувача.

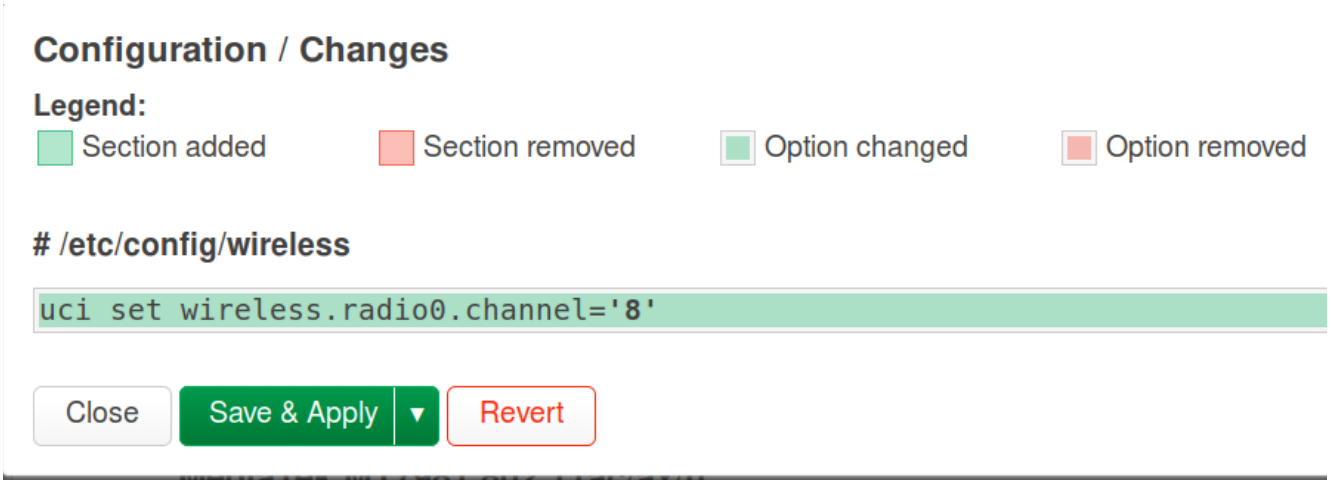


Рисунок 1.7 – Зміна каналу бездротової мережі в LuCI перетворена в команду системи UCI

В OpenWrt містяться засоби для резервного копіювання конфігурацій пристрою та їх відновлення (рис. 1.8), але, на жаль, через специфіку конфігураційних файлів, вони не зможуть працювати на різних платформах роутерів однаково. Невідповідність моделі мережевого обладнання може викликати непередбачувані конфлікти та помилки, оскільки різні пристрої можуть підтримувати різні технології, та мають різні найменування пристроїв в операційній системі. Перенесення з попередніх на новіші версії OpenWrt допускається.

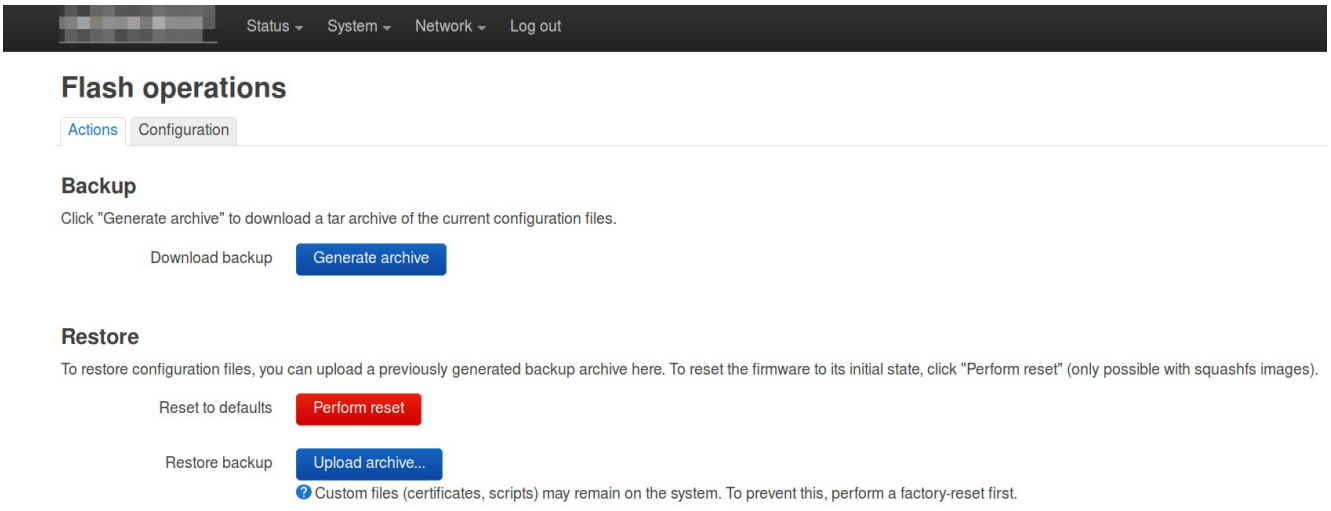


Рисунок 1.8 – Засоби резервного копіювання та відновлення в OpenWrt

1.3 Методи та засоби конфігурації різних класів маршрутизаторів, їх переваги та недоліки

Більшість популярних користувацьких роутерів, які купляють або орендують споживачі, вже готові до роботи та мають стандартні налаштування встановлені виробником, оскільки не всі користувачі цих пристроїв досвідчені в їх налаштуванні. Якщо пристрій орендується в Інтернет-провайдера, то їхні спеціалісти займаються налаштуванням пристрою.

Основний метод конфігурації користувацьких роутерів – вебінтерфейс. Вебсервер знаходиться на пристрої, доступ до інтерфейсу отримується за допомогою браузеру. Достатньо ввести в полі пошуку IP-адресу пристрою щоб потрапити до панелі входу. Для того щоб почати налаштування маршрутизатора в більшості випадків також необхідні дані для входу, такі як ім’я користувача та пароль, або лише пароль. Вебінтерфейс використовується для простоти налаштування пристрою, оскільки він інтуїтивно зрозумілий та всі необхідні функції знаходяться в потрібних вкладках та панелях.

Прикладом стандартного користувацького інтерфейсу можна запропонувати вебінтерфейси, що постачаються в прошивці маршрутизаторів від TP-Link (рис 1.9). Необхідна інформація про роутер, інформація про WAN-інтерфейс та бездротові пристрої знаходяться на головній сторінці пристрою. Всі інші необхідні налаштування знаходяться в додаткових вкладках. Це є прикладом дружнього до користувача інтерфейсу.

Також більшість користувацьких пристроїв мають засоби пришвидшеної конфігурації – Quick Setup (рис. 1.10). Це вікно, яке зустрічає користувача при першому вході на вебсервер маршрутизатора, та має набір важливих для налаштування параметрів. Як приклад, такими налаштуваннями можуть бути: часовий пояс, мова інтерфейсу, тип підключення до Інтернету, MAC-адрес пристрою, назви та паролі точок доступу, та інше. Це одне з покращень

користувацького досвіду, оскільки пристрій сам пропонує змінити більшість необхідних налаштувань.

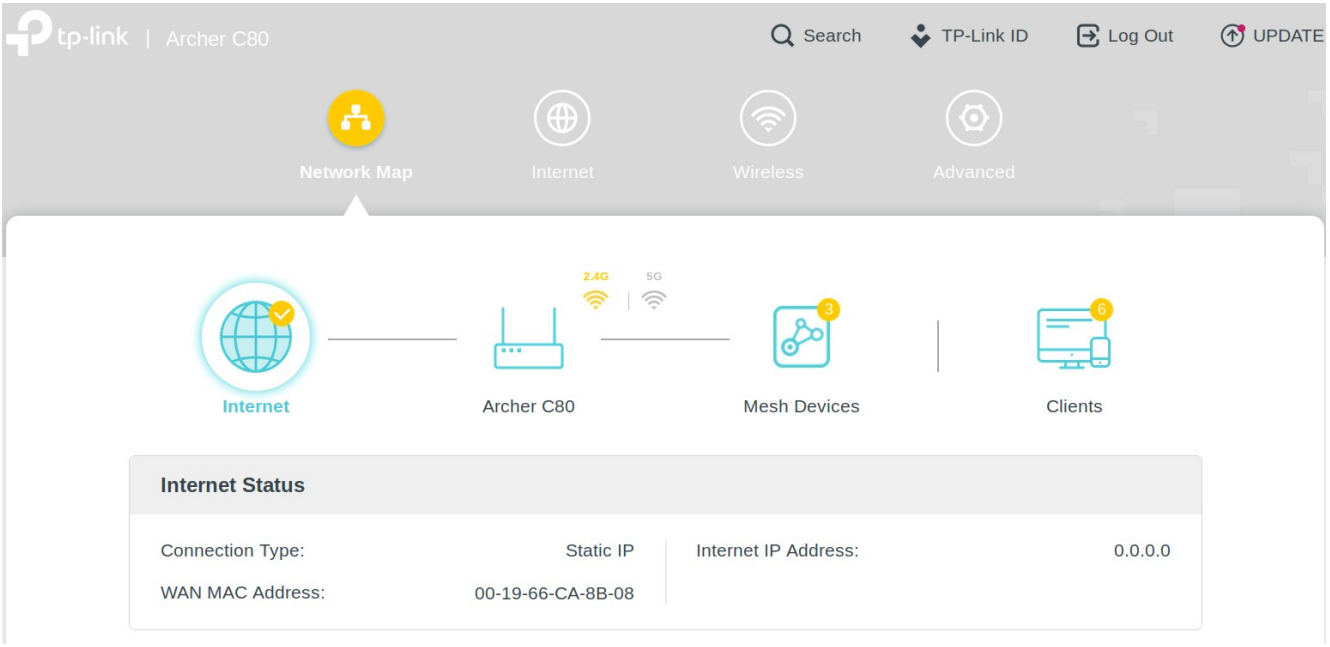


Рисунок 1.9 – Зразок головної сторінки вебінтерфейсу маршрутизатора TP-Link Archer C80 [10]

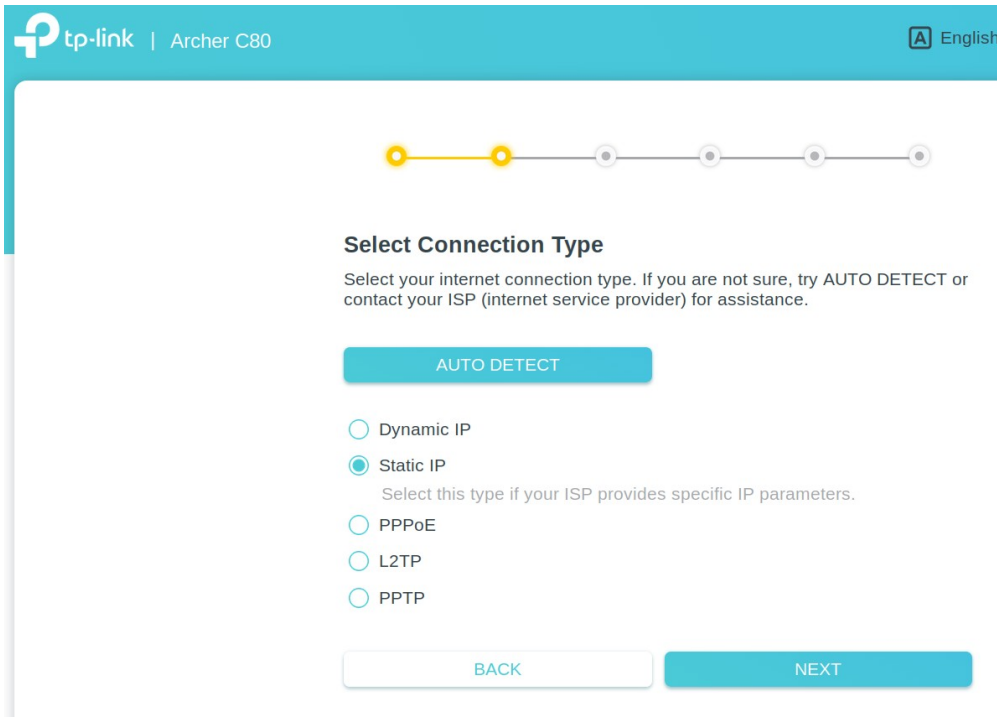


Рисунок 1.10 – Одна з вкладок Quick Setup в вебінтерфейсі маршрутизатора TP-Link Archer C80 [10]

Більшість пристроїв TP-Link мають засоби резервного копіювання конфігурації та її відновлення (рис. 1.11), яке також виконується через вебінтерфейс [11]. На жаль, такий спосіб перенесення конфігурації лише можливий в межах однієї і тієї ж моделі пристрою та ревізії апаратного забезпечення. Допускається перенесення конфігурації в межах кількох оновлень версії прошивки.

Спільнота TP-Link має існуючі від користувачів запитання про несумісність конфігурацій між різними версіями прошивки, що підтверджує існування проблеми[12]. Це робить такий засіб не універсальним та робочим лише для ідентичних пристроїв.

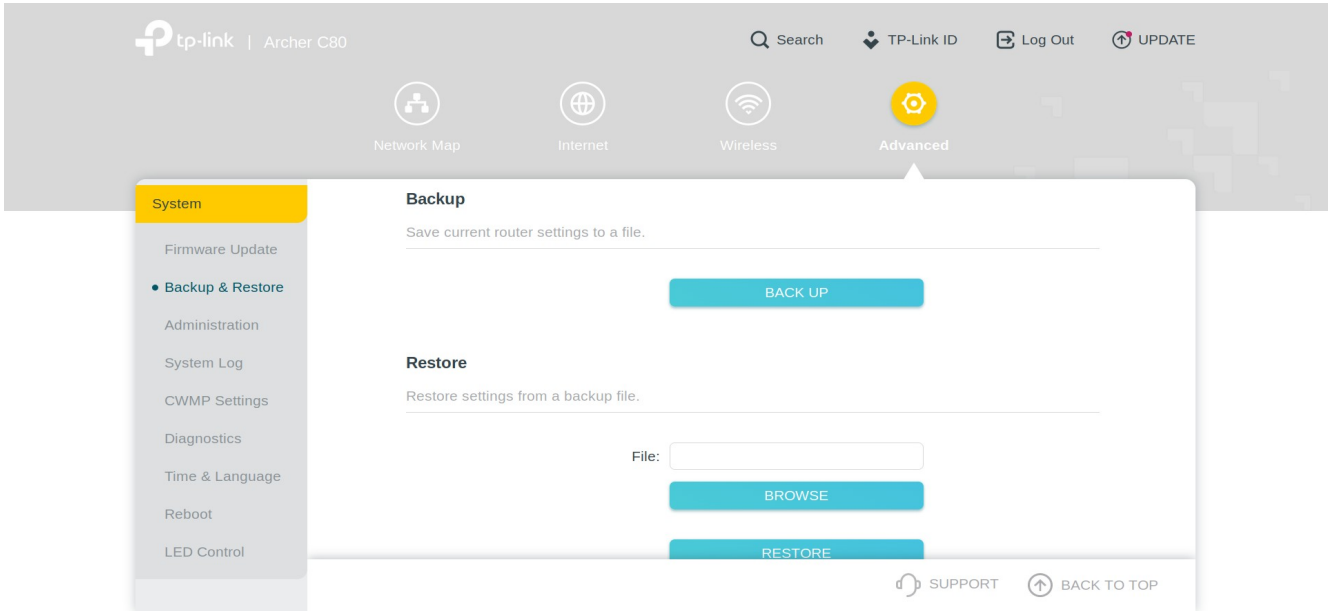


Рисунок 1.11 – Зразок меню резервного копіювання та відновлення в вебінтерфейсі маршрутизатора TP-Link Archer C80 [10]

На противагу користувацьким маршрутизаторам, існують корпоративні. В більшості з них для налаштування та конфігурації використовується командний рядок (CLI). Налаштування пристрою виконується через консольний порт, SSH, або рідше, Telnet. Професіонали з мереж вміють працювати з цими інструментами, тому складності в конфігурації пристроїв таким способом не виникає.

Протокол Secure Shell (SSH) – це метод для захищеної передачі команд на пристрій через незахищену мережу. SSH використовує криптографію для автентифікації та шифрування з'єднань між пристроями. SSH також дозволяє здійснювати тунелювання, або переадресацію портів, коли пакети даних можуть передаватись через мережі, які в іншому випадку були б для них недоступні. SSH часто використовується для віддаленого керування серверами, управління інфраструктурою та передачі файлів [13]. Як приклад, SSH може бути використаний для підключення до пристрою OpenWrt (рис. 1.12) з метою його налаштування, або отримання необхідних даних про стан роботи пристрою, його програмних підсистем.

```
[                    ~]$ ssh root@192.168.1.1
root@192.168.1.1's password:


BusyBox v1.37.0 (2026-03-24 21:24:12 UTC) built-in shell (ash)

 _______                     ________        __
|       |.-----.-----.-----.|  |  |  |.----.|  |_
|   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
|_______||   __|_____|__|__||________||__|  |____|
         |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 OpenWrt 25.12.2, r32802-f505120278 Dave's Guitar
 -----------------------------------------------------

 OpenWrt recently switched to the "apk" package manager!

 OPKG Command           APK Equivalent      Description
 ------------------------------------------------------------------
 opkg install <pkg>     apk add <pkg>       Install a package
 opkg remove <pkg>      apk del <pkg>       Remove a package
 opkg upgrade           apk upgrade         Upgrade all packages
 opkg files <pkg>       apk info -L <pkg>   List package contents
 opkg list-installed    apk info            List installed packages
 opkg update            apk update          Update package lists
 opkg search <pkg>      apk search <pkg>    Search for packages
 ------------------------------------------------------------------

For more information visit:
https://openwrt.org/docs/guide-user/additional-software/opkg-to-apk-cheatsheet

root@          :~# echo $PWD
/root
```

Рисунок 1.12 – Демонстрація використання SSH-клієнта для підключення до OpenWrt роутеру

Використання командного рядку як інтерфейсу для конфігурації пристроїв надає низку переваг:

- менше споживання ресурсів в порівнянні з вебсервером;
- захищений канал зв’язку. Налаштування застосовуються через кабель, або віддалене SSH підключення, в той час як більшість вебінтерфейсів пристроїв використовують незахищений HTTP протокол для передачі даних;
- віддалене керування. SSH надає можливість доступу до пристрою поза мережею цього пристрою, наприклад через VPN. Оскільки SSH безпечний, використовує шифрування трафіку та має захист від підбору логінів та паролів, його можна використовувати для цієї мети;
- простір для автоматизації. Інструменти SSH та використання консольного порту дозволяють створювати скрипти для застосування налаштувань до пристрою, або автоматизації процесів;
- швидкість та ефективність. Всі налаштування можна застосовувати виключно клавіатурою, копіюванням команд, або можна повністю автоматизувати процес;
- стандартизація. Якщо для порівняння брати пристрої Cisco, в операційній системі більшості пристроїв синтаксис команд та структура однакові. Це доцільно при автоматизації, оскільки можна налаштовувати різні пристрої одним набором команд.

Інший спосіб налаштування пристроїв – через використання конфігураційних файлів. В більшості випадків вони представлені звичайним текстовим файлом, який можна редагувати.

В пристроях Cisco конфігураційний файл представлено одним файлом, в якому знаходяться всі застосовані адміністратором команди (рис. 1.13). Його можна копіювати на накопичувачі та віддалені сервери. В внутрішній пам’яті пристрою можна зберігати різні конфігурації для різних сценаріїв. Якщо пристрої сумісні та мають схожі сервіси, то такий конфігураційний файл можна застосовувати на інші пристрої від компанії Cisco [14].

```
Router>en
Router#show running-config
Building configuration...

Current configuration : 653 bytes
!
version 16.6.4
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
!
hostname Router
!
!
!
!
!
!
!
!
no ip cef
no ipv6 cef
```

Рисунок 1.13 – Отримана частина конфігураційного файлу пристрою Cisco

1.4 Проблематика автоматизації процесу конфігурації роутерів, вибір методів та засобів реалізації системи

З зібраної інформації можна припустити, що домашні маршрутизатори під керуванням операційної системи OpenWrt та корпоративні пристрої сумісні з інструментами автоматизації процесів конфігурації. Через те що вони використовують SSH та командний інтерфейс, їх можна налаштовувати напряму через самостійно написані скрипти.

Для налаштування пристрою через командний рядок потрібна документація з командами, параметрами, логікою найменувань, тощо. Операційна система Cisco iOS має інструменти допомоги користувачу при вводі команд, а OpenWrt має експліцитну документацію. З цього випливає одна з проблем - всі потрібні команди не обов’язково зібрані в одному місці, а можуть знаходитись в різних вкладках, документах, на різних сайтах, що приносить незручності в пошуку необхідних.

В новачків, які вперше використовують ці методи та засоби, можуть виникати проблеми та помилки при налаштуванні. Також, більшості початківців потрібні схожі, не занадто специфічні налаштування, такі як: створення точок доступу, налаштування Wi-Fi пристроїв, зміна стандартного DNS-серверу, налаштування системних опцій пристрою, тощо. З цього випливає питання: чому б не об’єднати більшість потрібних команд, їх параметри та принцип дії в одному місці для уникнення помилок, та полегшити їх застосування ?

Для написання скриптів з метою оптимізації процесу конфігурації пристроїв необхідні також навички в роботі з скриптовими мовами. В Linux зазвичай використовуються оболонки sh, ash, dash, bash які приймають команди від користувача та мають функції виклику скриптових файлів. Самі по собі скриптові файли – це послідовність команд до виконання, функцій, змінних які інтерпретуються оболонкою[15].

В відкритому доступі є багато готових скриптів для різних сценаріїв, але немає гарантії що вони спрацюють без перевірки, тому потрібно аналізувати їх вміст перед виконанням. Пристою, що виконує налаштування віддаленого пристрою, обов’язково потрібен SSH-клієнт для відправлення команд. Операційна система Windows без сторонніх інструментів, наприклад, Windows WSL, не може працювати з sh та bash скриптами [16], тому скрипти налаштування мають бути написаними на підтримуваних скриптових мовах. Тут виникає наступне питання: чому б не зробити універсальний інструмент для побудови таких скриптів, щоб їх можна було виконувати на більшості платформ ?

Існують різні інструменти для автоматизації конфігурації, але більшість з них мають свої недоліки та складності. Якщо не брати конкретні приклади конфігураторів, процес налаштування роутерів через вебінтерфейс є довгим, а через командний рядок – заплутаним. Самі по собі ці методи не мають певного макету, шаблону за яким можна застосовувати багато налаштувань одночасно, а потрібно виконувати кожне налаштовання окремо, шукаючи їх у різних вкладках. З цього можна скласти таке питання: чи можна скласти певний набір команд для

виконання сценарії за потребами користувача, та приймати від користувача лише значення параметрів, щоб це було швидко та ефективно ?

Прикладом автоматизації для OpenWrt є використання Ansible. Він спрощує взаємодії людини з скриптами та командами. Ansible використовує мову YAML для написання послідовностей дій, щоб перетворити ці дії, присвоєні їм значення та параметри в код, що можна застосувати до пристрою. Це такий собі інтерпретатор, який перетворює текст з YAML файлів в команди, та має інструменти щоб їх застосувати. Його плюсом також є використання станів, що надає передбачуваності виконуваним командам та є опціональним інструментом [17].

Ansible має модуль для взаємодії з OpenWrt, який може збирати дані про пристрій, встановлювати пакети та застосовувати UCI команди. Всі дії, що знаходяться в YAML файлі (рис. 1.14), перетворюються в скрипти оболонки ash, що використовує OpenWrt. Пристрій, що здійснює конфігурації потребує мови програмування Python для використання Ansible [18]. Це вже ближче до спрощення конфігурації, але це скоріше засіб для застосування ніж набір команд.

```yaml
---
- hosts: routers
  gather_facts: false
  roles:
    - community.openwrt.init
  tasks:
    - name: Gather OpenWrt facts
      community.openwrt.setup:

    - name: Install a package
      community.openwrt.apk:
        name: luci
        state: present

    - name: Configure UCI settings
      community.openwrt.uci:
        command: set
        key: system.@system[0].hostname
        value: myrouter
```

Рисунок 1.14 – Приклад YAML конфігураційного файлу для OpenWrt пристрою [18]

HashiCorp Terraform – це один з інструментів для управління інфраструктурою як кодом, який дозволяє визначати як хмарні, так і локальні ресурси у вигляді зрозумілих для людини конфігураційних файлів, які можна розділяти за версіями, повторно використовувати та ділитися ними. Terraform може керувати як низькорівневими компонентами, такими як обчислювальні пристрої, сховища та мережеві ресурси, так і високорівневими компонентами, такими як записи DNS та функції SaaS [19].

Terraform має модуль для налаштування пристроїв Mikrotik що не використовує команди, а перетворює створені файлові запити користувача (рис. 1.15) в зрозумілі для пристрою команди. Цей засіб використовує специфічний синтаксис та потребує досвіду роботи з програмою для її використання [20]. Команди та засоби Mikrotik не можна застосовувати до пристроїв OpenWrt, оскільки ці дві різні операційні системи використовують

різний синтаксис та підхід, але мета автоматизації процесу конфігурація розкрита краще.

ip-address.tf

```
import {
  to = routeros_ip_address.lan
  id = "*1"
}
resource "routeros_ip_address" "lan" {
  address   = "192.168.88.1/24"
  interface = routeros_interface_bridge.bridge.name
  network   = "192.168.88.0"
}

import {
  to = routeros_ip_dhcp_client.wan
  id = "*1"
}
resource "routeros_ip_dhcp_client" "wan" {
  interface = "ether1"
}
```

Рисунок 1.15 – Задання локальної мережі та інтерфейсу з DHCP клієнтом пристрою MikroTik в файлі конфігурації Terraform [20]

Існують готові інструменти для віддаленої конфігурації OpenWrt, як прилад можна взяти openwrt-configurator, що опублікований на платформі Github від розробника jasrusable, та має відкритий вихідний код. Тут використовується мова програмування TypeScript, та написаний JSON конфігураційний файл перетворюється в UCI команди. В JSON файлі використовується синтаксис UCI, тому користувачу програми необхідно знати як працює ця система для конфігурації пристрою. Також тут додано використання умов, за допомогою яких, як заявляє автор, одним файлом можна налаштовувати пристрої за моделлю та типом пристрою, або його роллю [21].

Програма окрім самого перетворення конфігураційного файлу на команди UCI, також може застосовувати їх через SSH-клієнт. Це надзвичайно близько до мети роботи. Хоча це й теоретично значно полегшує процес налаштування пристроїв, конфігураційний файл все одно прив’язаний до синтаксису системи UCI. Це додає програмі можливості конфігурувати встановлені користувачем пакети, що є унікальною функцією.

Для того щоб вміти налаштовувати пристрій за допомогою програми користувач має використовувати OpenWrt документацію, або знати необхідні команди та конфігураційні файли. З цього можна створити наступні питання – чому б не полегшити задачу користувача, та не зробити синтаксис конфігураційного файлу та перетворення простішим ? Чому б не перетворювати загальновідомі секції та їх значення на зрозумілі пристрою команди ?

Таким чином, існує необхідність розробки підходу, який поєднує простоту використання, гнучкість конфігурації, та можливість автоматизації процесу налаштування маршрутизаторів на базі OpenWrt.

Вибір методів та засобів реалізації системи. Для вирішення задачі автоматизованої конфігурації пристрою запропоновано підхід, який передбачає використання центрального конфігураційного файлу, що буде перетворено у команди системи UCI, та їх виконанням на віддаленому пристрої.

Конфігураційний файл буде представлено файлом формату JSON (рис. 1.16). Цей формат забезпечує зручне представлення даних, є читабельним для людини, легко обробляється програмними засобами та отримав широке поширення серед сучасних мов програмування. JSON побудовано на двох структурах: колекція назв та значень, та впорядкований список значень. В цьому форматі є розділення на типи даних, що додатково корисно [22]. Цього достатньо для перетворення опису налаштувань в зрозумілий для програми формат.

```
{
    "name": "France",
    "capital": "Paris",
    "population": 67364357,
    "area": 551695,
    "currency": "Euro",
    "languages": ["French"],
    "region": "Europe",
    "subregion": "Western Europe",
    "flag": "https://upload.wikimedia.org/wikipedia/commons/c/c3/Flag_of_France.svg"
}
```


Рисунок 1.16 – Зразок JSON об’єкту [23]

Для реалізації програмного засобу, що уможливлює роботу системи конфігурації, обрано мову програмування Python. Вона проста у використанні та має широкий спектр бібліотек для роботи з мережею та обробкою даних. Python також підтримується різними операційними системами. В цій мові програмування є вбудовані інструменти для роботи з JSON файлами та перевірки важливих даних. Реалізація SSH-клієнту потребує зовнішньої бібліотеки.

Для організації віддаленого підключення до пристрою використано бібліотеку Python під назвою Paramiko. Вона реалізує протокол SSH, різні методи підключення до SSH-серверу (з використанням паролю та без), та дозволяє виконувати команди на віддаленому OpenWrt пристрої. Навіть якщо пристрій не має LuCI вебінтерфейсу, до нього завжди можна підключитись через SSH.

Ключовим елементом системи є механізм інтерпретації конфігураційного файлу. Налаштування описуються в JSON файлі зрозумілими користувачу виразами з мінімальним використанням синтаксису системи UCI, та потім перетворюються у відповідні команди. Такий спосіб надає гнучкості, оскільки заміна перетворювача може уможливити розробку такої методики для інших, не OpenWrt пристроїв.

2 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРОЦЕСУ КОНФІГУРАЦІЇ МАРШРУТИЗАТОРІВ

2.1 Аналіз процесу налаштування пристрою OpenWrt за допомогою вбудованого інструменту UCI

Процес налаштування пристрою під керуванням OpenWrt передбачає використання користувачем вебінтерфейсу LuCI, або використання команд UCI для налаштування пристрою. Для перевірки було сформовано два сценарії для налаштування пристрою – створення точок доступу в діапазонах 2.4 та 5 ГГц, та створення гостьового Wi-FI.

Перший сценарій налаштувань передбачає конфігурацію існуючних пристроїв, які представлені підписом «radio» та цифрою, що йде після нього. Після цього з прив’язкою до них створюються точки доступу, які в системі UCI мають тип «wifi-iface», а пристрої – «wifi-device».

Для роботи з інтерфейсами спочатку потрібно переглянути їх налаштування за допомогою команди «uci show wireless» (рис 2.1) [13]. З зібраної інформації визначено, що пристрій «radio0» відповідає за роботу в діапазоні 2.4 ГГц («band='2g'»), а «radio1» – 5 ГГц («band='5g'»). Версія Wi-Fi та ширина каналу («htmode='HT20'», «htmode='VHT80'») відповідають типовим налаштуванням для відповідних діапазонів.

В обох інтерфейсів від початку встановлені значення каналів (channel), які необхідно замінити для оптимізації мережі. Також пристрої перебувають у вимкненому стані (присутній параметр «disabled») та мають стандартні точки доступу з типовими ідентифікаторами («default_radio0», «defaulf_radio1»).

```
root@OpenWrt:~# uci show wireless
wireless.radio0=wifi-device
wireless.radio0.type='mac80211'
wireless.radio0.path='platform/10300000.wmac'
wireless.radio0.band='2g'
wireless.radio0.channel='1'
wireless.radio0.htmode='HT20'
wireless.radio0.disabled='1'
wireless.default_radio0=wifi-iface
wireless.default_radio0.device='radio0'
wireless.default_radio0.network='lan'
wireless.default_radio0.mode='ap'
wireless.default_radio0.ssid='OpenWrt'
wireless.default_radio0.encryption='none'
wireless.default_radio0.disabled='1'
wireless.radio1=wifi-device
wireless.radio1.type='mac80211'
wireless.radio1.path='pci0000:00/0000:00:00.0/0000:01:00.0'
wireless.radio1.band='5g'
wireless.radio1.channel='44'
wireless.radio1.htmode='VHT80'
wireless.radio1.disabled='1'
wireless.default_radio1=wifi-iface
wireless.default_radio1.device='radio1'
wireless.default_radio1.network='lan'
wireless.default_radio1.mode='ap'
wireless.default_radio1.ssid='OpenWrt'
wireless.default_radio1.encryption='none'
wireless.default_radio1.disabled='1'
```

Рисунок 2.1 – Вивід поточних параметрів конфігураційного файлу wireless пристрою OpenWrt

На основі вивчення документації конфігураційного файлу «wireless» та наявної конфігурації, визначено необхідні параметри для налаштування бездротової мережі. Зокрема, змінено канал для діапазону 2.4 ГГц на 8, а для діапазону 5 ГГц обрано автоматичний вибір каналу («auto») [24].

Точки доступу мають тип «wifi-iface», що визначає їх як окремі конфігураційні об’єкти. Для них задано наступні параметри: прив’язаний до них радіоінтерфейс («radio0», «radio1»), використана мережа (існуюча мережа «lan»),

режим роботи (стандартний – «ap»), ідентифікатор мережі (SSID), метод шифрування («sae-mixed» відповідає режиму WPA2/WPA3 Personal), пароль доступу до мережі. Після створення та налаштування точок доступу також необхідно увімкнути пристрої шляхом видалення параметру «disabled» [24].

Далі за допомогою команд «uci commit» та «wifi reload» конфігурація застосовується, та виконується перезавантаження бездротової системи відповідно. Після виконання команд пристрої та точки доступу налаштовані та переведені в робочий режим (рис. 2.2).

```
root@OpenWrt:~# uci set wireless.radio0.channel='8'
root@OpenWrt:~# uci set wireless.radio0_ap1='wifi-iface'
root@OpenWrt:~# uci set wireless.radio0_ap1.device='radio0'
root@OpenWrt:~# uci set wireless.radio0_ap1.network=''
root@OpenWrt:~# uci set wireless.radio0_ap1.network='lan'
root@OpenWrt:~# uci set wireless.radio0_ap1.mode='ap'
root@OpenWrt:~# uci set wireless.radio0_ap1.ssid='HomeWireless2.4'
root@OpenWrt:~# uci set wireless.radio0_ap1.encryption='sae-mixed'
root@OpenWrt:~# uci set wireless.radio0_ap1.key='sPECialKEEEY_@8765'
root@OpenWrt:~# uci set wireless.radio1.channel='auto'
root@OpenWrt:~# uci set wireless.radio1_ap1='wifi-iface'
root@OpenWrt:~# uci set wireless.radio1_ap1.device='radio1'
root@OpenWrt:~# uci set wireless.radio1_ap1.network='lan'
root@OpenWrt:~# uci set wireless.radio1_ap1.mode='ap'
root@OpenWrt:~# uci set wireless.radio1_ap1.ssid='HomeWireless5'
root@OpenWrt:~# uci set wireless.radio1_ap1.encryption='sae-mixed'
root@OpenWrt:~# uci set wireless.radio1_ap1.key='sPECialKEEEY_@8765'
root@OpenWrt:~# uci commit
root@OpenWrt:~# wifi reload
root@OpenWrt:~# uci del wireless.radio0.disabled
root@OpenWrt:~# uci del wireless.radio1.disabled
root@OpenWrt:~# uci commit
root@OpenWrt:~# wifi reload
```

Рисунок 2.2 – Налаштування пристроїв та точок доступу конфігурації wireless пристрою OpenWrt

Для перевірки коректності налаштування точок доступу використовується команда «wifi status». Вивід команди повідомляє про те, що пристрої працюють та використовують налаштовані точки доступу (рис. 2.3).

```
root@OpenWrt:~# wifi status
{
        "radio0": {
                "up": true,
                "pending": false,
                "autostart": true,
                "disabled": false,
                "retry_setup_failed": false,
                "config": {
                        "type": "mac80211",
                        "band": "2g",
                        "channel": "8",
                        "htmode": "HT20",
                        "path": "platform/10300000.wmac"
                },
                "interfaces": [
                        {
                                "section": "radio0_ap1",
                                "config": {
                                        "network": [
                                                "lan"
                                        ],
                                        "device": [
                                                "radio0"
                                        ],
                                        "mode": "ap",
                                        "encryption": "sae-mixed",
                                        "key": "sPECialKEEEY_@8765",
                                        "ssid": "HomeWireless2.4",
                                        "radios": [

                                        ]
                                },
```

Рисунок 2.3 – Статус бездротового пристрою radio0 та відомості про його точку доступу «HomeWireless2.4»

Другий сценарій передбачає покрокове створення ізольованого мережевого сегменту, створення точок доступу, пов’язаних із цим віртуальним інтерфейсом, конфігурація DHCP-серверу, створення окремої зони міжмережевого екрану, налаштування переадресації трафіку цієї зони та додання правил для доступу клієнтів зони до сервісів DHCP та DNS. Цей сценарій потребує більшу кількість налаштувань та більш критичний до допущення помилок. Документація на офіційному сайті OpenWrt містить готовий список команд для реалізації цього сценарію [25].

Першим етапом є створення віртуального мережевого пристрою та інтерфейсу. До пристрою застосовується ідентифікатор «guest_dev», тип – міст («bridge»), системна назва – «br-guest». Віртуальний інтерфейс з ідентифікатором «guest» використовує протокол статичної адресації, підключений до пристрою «br-guest», та використовує IP-мережу 192.168.3.1/24. Після виконання команд достатньо застосувати конфігурації та перезапустити мережеву підсистему (рис. 2.4).

```
root@OpenWrt:~# uci set network.guest_dev="device"
root@OpenWrt:~# uci set network.guest_dev.type="bridge"
root@OpenWrt:~# uci set network.guest_dev.name="br-guest"
root@OpenWrt:~# uci set network.guest="interface"
root@OpenWrt:~# uci set network.guest.proto="static"
root@OpenWrt:~# uci set network.guest.device="br-guest"
root@OpenWrt:~# uci set network.guest.ipaddr="192.168.3.1/24"
root@OpenWrt:~# uci commit network
root@OpenWrt:~# service network restart
```

Рисунок 2.4 – Команди для створення гостьового пристрою та інтерфейсу в операційній системі OpenWrt

Наступним кроком є налаштування бездротових точок доступу. Оскільки точки доступу вже були створені у попередньому сценарії, доцільно змінити прив’язану до них мережу з «lan» на «guest», а також оновити ідентифікатори мереж (SSID) (рис. 2.5).

```
root@OpenWrt:~# uci set wireless.radio0_ap1.network='guest'
root@OpenWrt:~# uci set wireless.radio0_ap1.ssid='HomeGuest2.4'
root@OpenWrt:~# uci set wireless.radio1_ap1.network='guest'
root@OpenWrt:~# uci set wireless.radio1_ap1.ssid='HomeGuest5'
```

Рисунок 2.5 – Команди для зміни мережі та ідентифікатора бездротових точок доступу

Далі необхідно додати DHCP-сервер до пристрою. Для цього використовується конфігураційний файл «dhcp», куди додається новий об’єкт

«guest» типу «dhcp». До нього застосовуються наступні налаштування: прив’язка до інтерфейсу «guest», початкова та кінцева адреси доступного діапазону IP-адрес, обмеження на час використання. Після цих налаштувань виконується команда «uci commit dhcp» для застосування параметрів до теперішньої конфігурації, та перезапуск сервісу DHCP/DNS – Dnsmasq (рис. 2.6).

```
root@OpenWrt:~# uci set dhcp.guest="dhcp"
root@OpenWrt:~# uci set dhcp.guest.interface="guest"
root@OpenWrt:~# uci set dhcp.guest.start="100"
root@OpenWrt:~# uci set dhcp.guest.limit="150"
root@OpenWrt:~# uci set dhcp.guest.leasetime="1h"
root@OpenWrt:~# uci commit dhcp
root@OpenWrt:~# service dnsmasq restart
udhcpc: started, v1.37.0
udhcpc: broadcasting discover
udhcpc: no lease, failing
```

Рисунок 2.6 – Застосовані параметри для створення DHCP-серверу для гостьової зони та результати їх виконання

Наступний крок – налаштування міжмережевого екрану. Для цього необхідно виконати наступну послідовність дій [25]:

1. створити зони міжмережевого екрану «guest»;
2. налаштувати політики зони – створити дозвіл на вихід трафіку з зони, відмовляти в переадресації між іншими зонами, відмовляти при вході трафіку в зону;
3. створити дозвіл на переадресацію з зони «guest» у «wan» для доступу користувачів до Інтернету;
4. створити правило на дозвіл DNS запитів «порт 53, протоколи TCP/UDP» з зони «guest»;
5. створити правило на дозвіл DHCP запитів «порт 67, протокол UDP, сімейство протоколів IPv4» з зони «guest».

Після цього необхідно застосувати правила міжмережевого екрану до теперішньої конфігурації та перезапустити його підсистему (рис. 2.7).

```
root@OpenWrt:~# uci set firewall.guest="zone"
root@OpenWrt:~# uci set firewall.guest.name="guest"
root@OpenWrt:~# uci set firewall.guest.network="guest"
root@OpenWrt:~# uci set firewall.guest.input="REJECT"
root@OpenWrt:~# uci set firewall.guest.output="ACCEPT"
root@OpenWrt:~# uci set firewall.guest.forward="REJECT"
root@OpenWrt:~# uci set firewall.guest_wan="forwarding"
root@OpenWrt:~# uci set firewall.guest_wan.src="guest"
root@OpenWrt:~# uci set firewall.guest_wan.dest="wan"
root@OpenWrt:~# uci set firewall.guest_dns="rule"
root@OpenWrt:~# uci set firewall.guest_dns.name="Allow-DNS-Guest"
root@OpenWrt:~# uci set firewall.guest_dns.src="guest"
root@OpenWrt:~# uci set firewall.guest_dns.dest_port="53"
root@OpenWrt:~# uci set firewall.guest_dns.proto="tcp udp"
root@OpenWrt:~# uci set firewall.guest_dns.target="ACCEPT"
root@OpenWrt:~# uci set firewall.guest_dhcp="rule"
root@OpenWrt:~# uci set firewall.guest_dhcp.name="Allow-DHCP-Guest"
root@OpenWrt:~# uci set firewall.guest_dhcp.src="guest"
root@OpenWrt:~# uci set firewall.guest_dhcp.dest_port="67"
root@OpenWrt:~# uci set firewall.guest_dhcp.proto="udp"
root@OpenWrt:~# uci set firewall.guest_dhcp.family="ipv4"
root@OpenWrt:~# uci set firewall.guest_dhcp.target="ACCEPT"
root@OpenWrt:~# uci commit firewall
root@OpenWrt:~# service firewall restart
```

Рисунок 2.7 – Використані команди для налаштування міжмережевого екрану

Для перевірки коректності налаштування мережі точок доступу використовується команда «wifi status» (рис 2.8). Результат повідомляє про те, що в точок доступу змінена мережа на «guest», змінено SSID та вони знаходяться в працюючому режимі.

Таким чином, виконані налаштування забезпечують ізоляцію гостьової мережі від основної мережі «lan», а також обмеження доступу до внутрішніх ресурсів із збереженням можливості доступу до мережі Інтернет.

Розглянуті сценарії демонструють значну складність ручної конфігурації та необхідність виконання великої кількості взаємопов’язаних налаштувань, що підтверджує доцільність автоматизації даного процесу.

```
root@OpenWrt:~# wifi status
{
        "radio0": {
                "up": true,
                "pending": false,
                "autostart": true,
                "disabled": false,
                "retry_setup_failed": false,
                "config": {
                        "type": "mac80211",
                        "band": "2g",
                        "channel": "8",
                        "htmode": "HT20",
                        "path": "platform/10300000.wmac"
                },
                "interfaces": [
                        {
                                "section": "radio0_ap1",
                                "config": {
                                        "network": [
                                                "guest"
                                        ],
                                        "device": [
                                                "radio0"
                                        ],
                                        "mode": "ap",
                                        "encryption": "sae-mixed",
                                        "key": "sPECialKEEEY_@8765",
                                        "ssid": "HomeGuest2.4",
                                        "radios": [

                                        ]
                                },
```

Рисунок 2.8 – Статус бездротового пристрою radio0 та відомості про його точку доступу «HomeGuest2.4»

2.2 Аналіз проблем ручної конфігурації маршрутизаторів та формування вимог до системи конфігурації

Проведений аналіз процесу налаштування пристрою з операційною системою OpenWrt показав, що для реалізації типових сценаріїв налаштування необхідно виконувати значну кількість послідовних дій. Проаналізовані сценарії налаштування бездротової мережі та гостьового сегменту потребують взаємодії з

декількома підсистемами, такими як мережеві інтерфейси, бездротові параметри, DHCP-сервер та міжмережевий екран.

Однією з основних проблем є складність використання інтерфейсу командного рядку. Адміністратор повинен знати синтаксис команд, структуру конфігураційних файлів та принцип взаємодії між різними компонентами системи. Це підвищує поріг входження та ускладнює процес конфігурації для користувачів без достатнього досвіду.

Процес налаштування характеризується високим рівнем повторюваності. Це проблема що зберігається як для графічного, так і консольного інтерфейсу більшості мережевих пристроїв. Для кожного нового пристрою та подібних сценаріїв використання потрібно виконувати аналогічну послідовність команд, що збільшує витрату часу та знижує ефективність адміністрування мережі.

Однією з важливих проблем є підвищена ймовірність виникання помилок. Неправильне введення команд, некоректні параметри, пропуск важливих команд або порушення логіки виконання можуть призвести до некоректної роботи мережі або зниження рівню її безпеки. Наприклад, якщо при налаштуванні гостьового сегменту не налаштувати DHCP-сервер для інтерфейсу, або неправильно вказати його параметри, користувачі не зможуть підключитись до точки доступу автоматично, бо не зможуть отримати необхідні конфігурації IP-адреси.

Команди для налаштування гостьової мережі можливо виконати через вебінтерфейс LuCI, але це не значно ефективніший спосіб. Необхідні параметри знаходяться в різних вкладках та на різних сторінках. Разом з цим підвищується шанс на допускання посилок через більшу повторюваність дій, тому потрібно виконувати дії за інструкцією з уважністю. Тому, це хоч і простіший, але не набагато швидший та ефективніший процес ніж консольне налаштування [26].

Також відсутність централізованого підходу до зберігання та управління конфігураціями ускладнює масштабування мережевої інфраструктури. У випадку необхідності розгортання однакових налаштувань на декількох різних пристроях процес конфігурації виконується повторно для кожного. Як приклад, вбудований

інструмент резервного копіювання не повноцінний і не гарантує сумісності між різними платформами та версіями OpenWrt.

Отже, проведений аналіз підтверджує та демонструє наявність проблем, пов’язаних із складністю, трудомісткістю, повторюваністю та низьким рівнем автоматизації процесу конфігурації маршрутизаторів. Це обґрунтовує доцільність розробки автоматизованої системи, яка дозволить спростити процес налаштування, зменшити кількість помилок та підвищити ефективність адміністрування мереж.

Розглянуті сценарії налаштування були апробовані на практиці шляхом виконання відповідних команд на пристрої з операційною системою OpenWrt. Отримані результати підтверджують працездатність описаних підходів та дозволяють оцінити складність процесу конфігурації.

Формування вимог до системи конфігурації. Автоматизована система має реалізовувати такий список налаштувань:

- налаштування бездротових мереж: розділення налаштувань за пристроями, налаштування використовуваної версії Wi-Fi, каналу та його ширини;
- створення бездротових точок доступу, вказання їх ідентифікатору мережі (SSID), режиму шифрування, паролю доступу до мережі, стану, режиму ізоляції, та використання випадкової MAC-адреси;
- створення гостьових точок доступу разом зі створенням гостьового мережевого сегменту;
- налаштування міжмережевого екрану: увімкнення захисту від SYN-Flood атак, відкидання невідомих пакетів, відкидання замість відмовлення в доступі, налаштування розвантаження потоку (flow offloading);
- застосування змінного DNS-серверу до WAN інтерфейсу;
- налаштування DNS-кешування;
- системні налаштування: перевірка на оновлення, назва пристрою (hostname), часовий пояс, формат часу, пароль для доступу до SSH, обмеження

доступу до SSH лише через «lan» сегмент, переведення вебінтерфейсу з HTTP на HTTPS, встановлення змінних NTP серверів.

Більшість команд UCI можна представити у вигляді простіших користувацьких потреб. Наприклад, встановлення назви пристрою (hostname), використання зрозумілого переліку параметрів бездротових пристроїв, створення гостьового сегменту з переліком точок доступу, що використовують цю мережу.

Зазначені параметри та описи доцільно зберігати у форматі JSON. Даний формат надає можливість використання об’єктів, зв’язків між назвами та параметрами, масивів значень та різних типів даних. Це є необхідним та достатнім набором для створення структури конфігураційного файлу, який у подальшому може бути перетворений у команди UCI.

Секції конфігураційного файлу, як і параметри в них, мають бути опціональними. Не всі пристрої потребують однакового набору налаштувань, тому система має бути гнучкою. Для окремих пристроїв може не виникати потреби в налаштуванні бездротових мереж або інших параметрів.

Система має перевіряти конфігураційний файл на відповідність формату JSON, а також виконувати перевірку коректності введених даних. Користувач може ввести числове або логічне значення у вигляді рядка, що може призвести до некоректної роботи програми. Також, необхідно враховувати обмеження параметрів, наприклад довжину пароля бездротової мережі та допустимі символи [27].

Оскільки без використання вебінтерфейсу не можна однозначно визначити які версії Wi-Fi підтримує пристрій, та які ширини каналів доступні, необхідно реалізувати функціонал для отримання таких даних. У системах на базі Linux є вбудовані інструменти для отримання цієї інформації, тому необхідно забезпечити коректне перетворення отриманих текстових даних у зрозумілі для користувача вигляд.

Існує проблема того, що не на всіх пристроях OpenWrt бездротовий пристрій radio0 відповідає за діапазон 5 ГГц, а radio1 – 2.4 ГГц. Також існують

маршрутизатори, які мають більше ніж два бездротових інтерфейсів. У зв’язку з цим має бути реалізована прив’язка параметрів та точок доступу до самих пристроїв, а не лише діапазонів частот.

Однією з функціональних вимог є наявність різних способів застосування конфігурації. Не всі користувачі потребують негайного застосування налаштувань на пристрої. Система повинна підтримувати режим перегляду згенерованих команд, створення скриптів для подальшого використання на пристрої, а також основний режим – підключення до пристрою через SSH з подальшим застосуванням налаштувань.

3 РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ КОНФІГУРАЦІЇ МАРШРУТИЗАТОРА НА БАЗІ OPENWRT

3.1 Загальна архітектура системи та структура конфігураційного файлу

Автоматизована система конфігурації складається з декількох основних компонентів. Основою, на базі якої формуються команди для маршрутизатора, є конфігураційний файл у форматі JSON. У ньому здійснюється опис налаштувань компонентів операційної системи, які реалізуються програмними засобами. JSON-файл не використовує синтаксис UCI команд безпосередньо, а використовує загальні поняття, що спрощує взаємодію користувача з програмним забезпеченням.

Наступним важливим елементом є валідатор конфігурації. Програма очікує визначені типи даних та використання коректних значень що застосовуються. Даний компонент виконує перевірку введених у конфігураційному файлі даних на відповідність формату, наявність необхідних параметрів та правильність введених значень параметрів, що забезпечує коректність подальшої роботи системи. Як приклад, режим роботи Wi-Fi та ширина каналу бездротової мережі мають задаватись узгоджено, оскільки в системі UCI ці дані знаходяться в одній змінній.

Основним компонентом системи є інтерпретатор. Саме цей елемент відповідає за перетворення даних із конфігураційного файлу у команди системи UCI. Більшість команд UCI є повторюваними та типовими для схожих сценарії, та відрізняються лише значеннями параметрів. Цей перетворювач вносить перевірені дані з конфігураційного файлу в шаблони команд, та зберігає їх у вигляді послідовності.

У системи передбачені різні режими роботи з створеними командами: застосування команд до пристрою, виведення даних, створення скрипту оболонки

з цими командами. Перший режим використовує протокол SSH для доступу до пристрою та потребує від користувача введення IP-адреси, імені користувача та паролю. Режим виведення дозволяє переглянути згенеровані команди. Режим створення скрипту для пристрою передбачає формування файлу типу .sh, у який записуються створені команди для подальшого виконання їх на пристрої.

Система передбачає також додаткові режими роботи. Для того щоб користувач міг окремо перевірити JSON-файл на помилки та виправити їх, передбачено режим перевірки конфігураційного файлу, який виконує лише крок валідації. Для коректного застосування параметрів бездротової мережі необхідно знати назви бездротових пристроїв, та підтримувані ними режими роботи Wi-Fi, ширини каналів. Режим отримання можливостей бездротової мережі передбачає отримання та виведення цих даних користувачеві, який може використати їх для формування конфігураційного файлу.

Як приклад, згідно з архітектурою програми, при виконанні перетворення конфігураційного файлу в команди системи UCI, виконується наступний алгоритм дій (рис. 3.1): отримання JSON-файлу, валідація даних в ньому, інтерпретація параметрів з файлу в команди системи UCI, перенесення команд в масив, його обробка та виведення масиву як результат. Також, при файлу перевіряється його існування та коректність формату. При валідації даних з файлу помилки накопичуються в масив, та якщо вони присутні, користувач отримує повідомлення про це, та такий конфігураційний файл не може далі використовуватись.

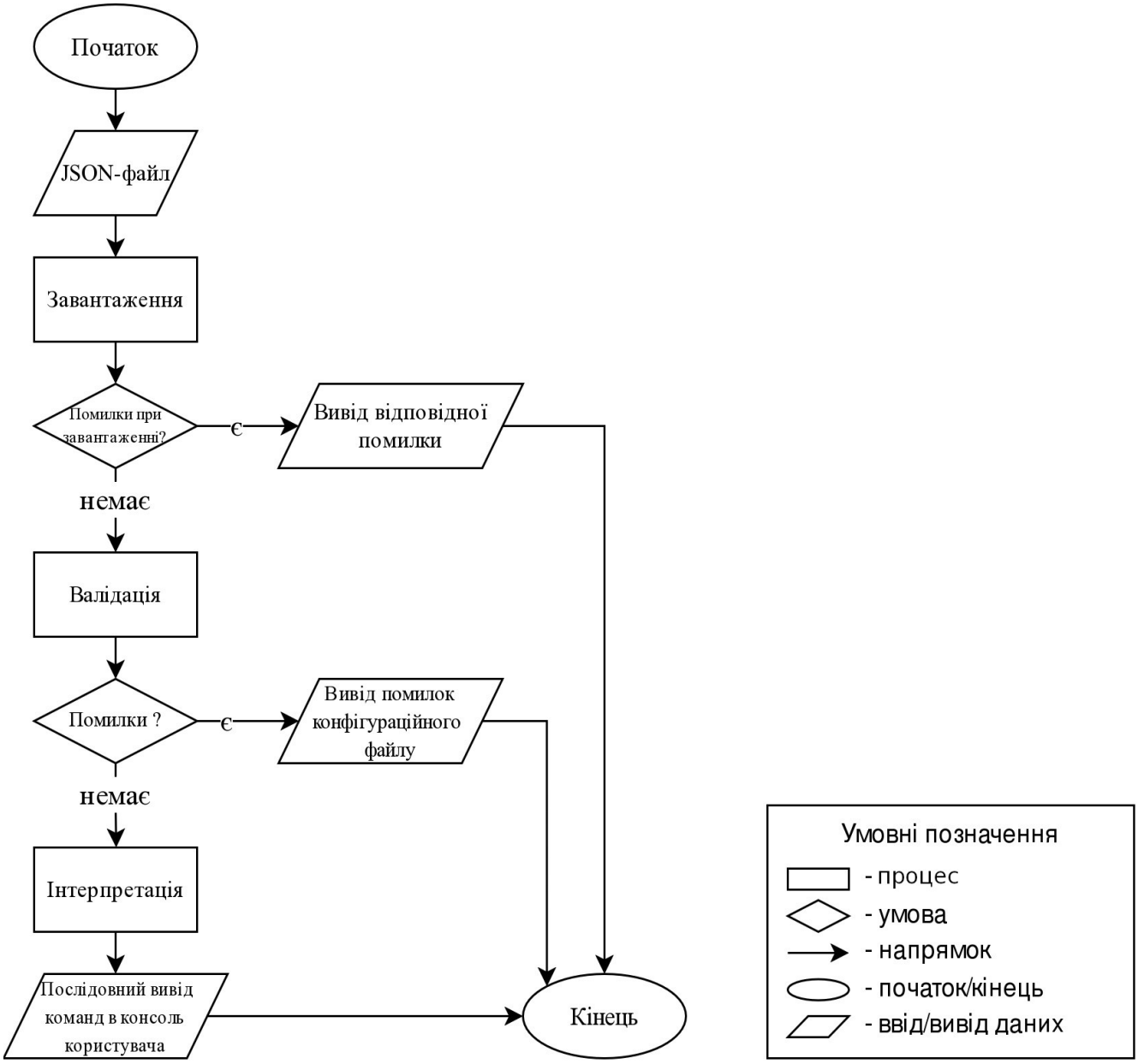


Рисунок 3.1 – Послідовність дій при перетворенні JSON-файлу в команди

Структура конфігураційного файлу. Конфігураційний файл системи складається з визначеного набору об’єктів, що містять необхідні параметри та їх значення. Цей файл розділений на наступні секції:

- «wifi»;
- «guest_wifi»;
- «firewall»;
- «wan»;
- «dns_caching»;

– «system».

Секція «wifi» (рис. 3.2) містить в собі набір бездротових пристроїв до яких потрібно застосувати зміни. Кожен пристрій описується наступними параметри: діапазон роботи пристрою «band», режим роботи Wi-Fi «mode», використовуваний канал «channel», ширина каналу «width».

Для кожного пристрою може бути визначено набір точок доступу. Точка доступу має наступні параметри: ідентифікатор мережі «ssid», режим шифрування «encryption», використовуваний точкою доступу пароль «password», перемикач стану точки доступу «disabled», перемикач ізоляції клієнтів точки доступу «isolate», перемикач вказання випадкової MAC-адреси точки доступу «random_macaddr».

Наступним блоком конфігураційного файлу є «guest_wifi» (рис. 3.3). Його структура ідентична до «wifi», але в ньому не містяться параметри бездротових пристроїв. Блок «guest_wifi» відповідальний лише за гостьові точки доступу, параметри бездротових пристроїв мають вказуватись в об’єкті «wifi».

```
"wifi":{
    "devices":{
        "radio0":{
            "band": "2g",
            "mode": "ax",
            "channel": 8,
            "width": 20,
            "aps":[
                {
                    "ssid": "Home_2.4",
                    "encryption": "wpa2",
                    "password": "12345678",
                    "disabled": true,
                    "isolate": false,
                    "random_macaddr": false
                }
            ]
        },
```

Рисунок 3.2 – Зразок секції «wifi» конфігураційного файлу з заданими параметрами бездротового пристрою «radio0» та зразковою точкою доступу

```
"guest_wifi":{
    "devices":{
        "radio0":{
            "aps":[
                {
                    "ssid": "HomeGuests_2.4",
                    "encryption": "wpa2",
                    "password": "87654321",
                    "disabled": true,
                    "isolate": true,
                    "random_macaddr": false
                }
            ]
        },
```

Рисунок 3.3 – Секція «guest_wifi» із зразковою точкою доступу на пристрої «radio0»

У блоці «firewall» (рис. 3.4) містяться параметри, що застосовуються до міжмережевого екрану. Тут знаходяться наступні параметри: перемикач захисту від SYN-Flood атак «syn-flood_protection», перемикач відкидання невідомих пакетів «drop_invalid_packets», перемикач, що заміняє відхилення пакетів відкиданням «drop_instead_of_reject», параметр розвантаження потоків «flow_offloading».

```
"firewall":{
    "syn-flood_protection": true,
    "drop_invalid_packets": true,
    "drop_instead_of_reject": true,
    "flow_offloading": "software"
},
```

Рисунок 3.4 – Секція «firewall» з зразковими налаштуваннями

Блок «wan» (рис. 3.5) має одне головне налаштування – список використовуваних IP-адрес DNS-серверів у форматі списку, до яких буде перенаправлено DNS-запити мережі. Секція «dns_caching» (рис. 3.5) містить в

собі наступні параметри: список кешованих DNS-записів, розмір кешу, мінімальний та максимальний час життя DNS-запису в кеші.

```
"wan":{
    "dns_servers":[
        "9.9.9.9",
        "149.112.112.112"
    ]
},
"dns_caching":{
    "cache_records":[
        "A",
        "CNAME",
        "PTR",
        "SRV"
    ],
    "cache_size": 10000,
    "min_cache_ttl": 60,
    "max_cache_ttl": 86400
},
```

Рисунок 3.5 – Секції «wan» та «dns_caching» з зразковими налаштуваннями

Останнім блоком налаштувань є «system» (рис 3.6). В ньому знаходяться релевантні до системних налаштувань параметри: перемикач перевірки оновлень операційної системи «attendedsysupgrade», назва пристрою «hostname», часова зона «timezone», формат часу «time_format», пароль для доступу до SSH «ssh_root_password», перемикач увімкнення доступу до SSH лише з локальної мережі «ssh_through_lan_only», перемикач увімкнення переадресації з HTTP вебінтерфейсу пристрою на HTTPS «redir_http_https» та список NTP-серверів використовуваних пристроєм «ntp_servers».

```
"system":{
    "attendedsysupgrade": true,
    "hostname": "ProjectRouter",
    "timezone": "Europe/Kyiv",
    "time_format": "24h",
    "ssh_root_password": "passwordd",
    "ssh_through_lan_only": true,
    "redir_http_https": true,
    "ntp_servers":[
        "0.ua.pool.ntp.org",
        "1.ua.pool.ntp.org",
        "2.ua.pool.ntp.org",
        "3.ua.pool.ntp.org"
    ]
}
```

Рисунок 3.6 – Секція «system» з зразковими налаштуваннями

До параметрів в секціях конфігураційного файлу застосовано відповідні типи даних JSON – рядкові, числові та логічні типи даних, що забезпечує коректність обробки конфігурації та зменшує ймовірність помилок. Повний зразковий конфігураційний файл розміщено в додатку А.

3.2 Алгоритм функціонування системи та реалізація програмного забезпечення

Система конфігурації реалізована у вигляді консольної програми та приймає аргументи від користувача для виконання відповідних дій (рис. 3.7, 3.8). Програма підтримує наступні режими роботи: перевірка конфігураційного файлу «check», перетворення цього файлу у команди UCI «convert», застосування сформованих команд через SSH до пристрою на базі OpenWrt «commit», отримання можливостей бездротової мережі маршрутизатора «get_wifi_features», а також формування скрипту з командами «make_script». У разі відсутності аргументів або

їх недостатній кількості під час виклику програми, відображається довідка з можливими діями та їх параметрами.

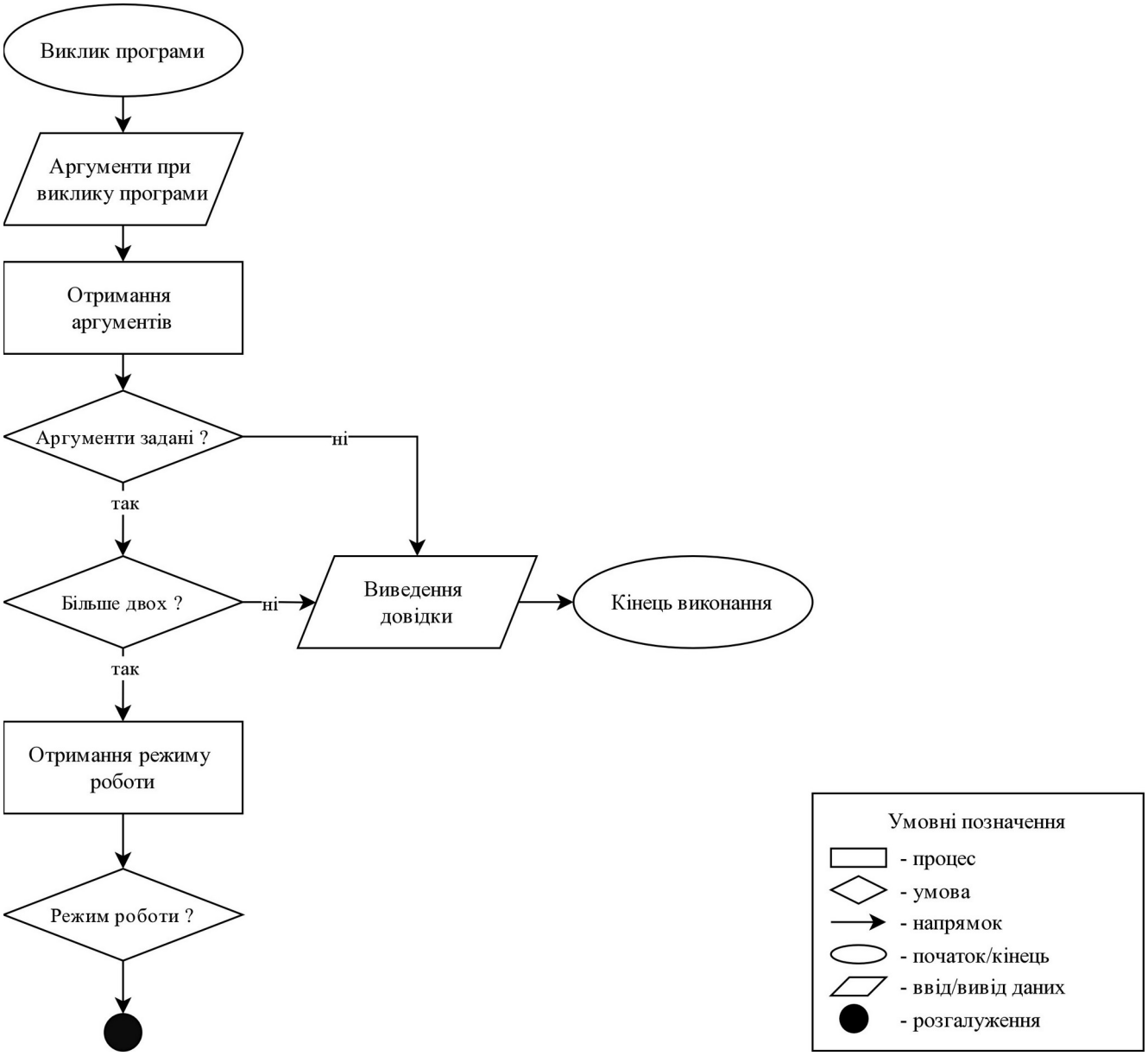


Рисунок 3.7 – Частина спрощеного алгоритму програми до розгалуження

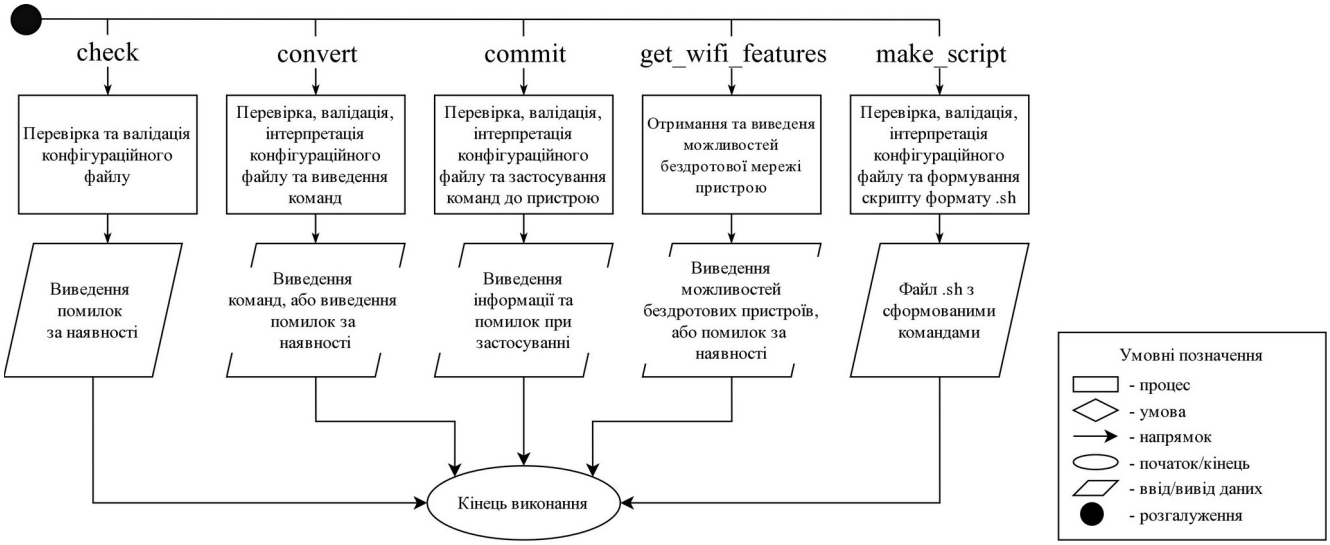


Рисунок 3.8 – Частина спрощеного алгоритму програми після розгалуження

Перевірка конфігураційного файлу може виконуватися як окрема дія. На вхід подається шлях до файлу конфігурації, після чого виконується перевірка його існування та відповідності формату JSON. У разі невиконання цих умов відображається повідомлення про помилку. Далі здійснюється валідація параметрів конфігурації. У випадку виявлення некоректних значень валідатор збирає помилки у вигляді списку та після завершення обробки відображає їх користувачу. За наявності помилок конфігураційний файл не може бути використаний для подальших дій.

Перетворення конфігураційного файлу у команди UCI виконується після успішної перевірки конфігурації. Якщо помилки відсутні, значення параметрів обробляються та розміщуються у відповідні команди UCI. Сформовані команди зберігаються у вигляді списку та після завершення обробки відображаються користувачеві для подальшого використання.

Перед застосуванням конфігурації до пристрою через SSH виконується попередній етап формування команд, однак без їх виведення користувачу. Для підключення користувач в аргументи програми вводить шлях до конфігураційного файлу та IP-адресу пристрою. Якщо IP-адреса не була введена, використовується стандартна для OpenWrt адреса – 192.168.1.1. Ім’я користувача та пароль запитуються інтерактивно після виклику. Якщо їх залишити пустими,

використовуються стандартні ім’я користувача та пароль – користувач root, пароль пустий. Після встановлення з’єднання всі сформовані команди послідовно виконуються на пристрої. Результати виконання команд, а саме інформація та помилки, відображається користувачу.

Формування скрипту також базується на етапі перетворення конфігурації у команди UCI. Додатково користувач задає шлях до файлу, який необхідно створити. Після генерації команд формується файл типу .sh, який містить відповідним чином оформлені команди для подальшого виконання на пристрої.

Отримання можливостей бездротової мережі передбачає підключення до віддаленого пристрою через SSH. Після цього виконується збір інформації з пристрою та її обробка в програмному забезпеченні. На основі отриманих даних здійснюється пошук ключових параметрів, що дозволяє визначити підтримувані можливості пристрою. Результати подаються користувачу у відповідному до конфігураційного файлу вигляді та можуть бути використані при його формуванні.

Реалізація програмного забезпечення. Автоматизована система конфігурації маршрутизаторів реалізована мовою програмування Python у вигляді консольного програмного забезпечення. Розроблена система має наступні функції: обробку конфігураційного файлу у форматі JSON, його перевірку та валідацію, перетворення у команди системи UCI та реалізує різні режими роботи. Логіка роботи програми відповідає розробленому алгоритму функціонування програми (рис. 3.7, 3.8).

Загальна структура програмного забезпечення будується з таких модулів:

- wrtcfg.py – модуль, з яким взаємодіє користувач. Він приймає аргументи та виконує послідовності команд відповідно до дій;
- loader.py – завантажує конфігураційний файл JSON та попередньо перевіряє існування файлу та його відповідний формат;
- validator.py – виконує валідацію даних конфігураційного файлу;
- uci_builder.py – перетворює дані з конфігураційного файлу у команди UCI;

– ssh_client.py – реалізує протокол SSH для підключення до пристрою та виконання на ньому команд;

– wifi_features_parser.py – оброблює вивід команди та повідомляє користувачу про апаратні та програмні можливості бездротових пристроїв.

Реалізація перевірки конфігураційного файлу передбачає кілька етапів: перевірку існування файлу, відповідність формату JSON та валідацію параметрів. Для цього використовується вбудований механізм обробки JSON та розроблені функції перевірки значень. Для обробки JSON (рис. 3.9) та перевірки значень також використовуються вбудовані модулі Python:

– «json», який відповідальний за перевірку формату та перетворення формату JSON в формат, який можна використовувати в системі;

– «zoneinfo» перевіряє існування часової зони;

– «ipaddress» перевіряє правильність введеної IP-адреси.

```
import json
try:
    with open(path, "r") as f:
        return json.load(f)
except FileNotFoundError:
    raise Exception(f"Config file not found: {path}")
except json.JSONDecodeError:
    raise Exception("Invalid JSON format")
```

Рисунок 3.9 – Фрагмент коду, відповідальний за зчитування файлу та обробку помилок

Після успішного зчитування виконується валідація параметрів конфігурації. Кожен з них обробляється відповідним механізмом, щоб підтвердити можливість їх використання в наступних кроках. У разі виявлення помилок вони накопичуються у вигляді списку та відображаються користувачу після завершення перевірки. Наприклад, пароль для підключення до точок доступу не використовується лише якщо вказано відсутність методу шифрування (відкрита мережа). Також цей пароль також має містити від 8 до 63 символів строго [27] (рис. 3.10).

```
password = ap.get("password")
if not password and encryption != encryptions[0]:
    error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
    "no password for specified encryption"))
elif not password and encryption in encryptions[0]:
    return
elif len(password) < 8:
    error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
    "password should be at least 8 symbols long"))
elif len(password) > 63:
    error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
    "password is too long"))
```

Рисунок 3.10 – Частина коду, відповідальна за перевірку паролю точки доступу

Інтерпретацію конфігураційного файлу у команди системи UCI реалізовано шляхом формування наборів команд, та розміщення в командах відповідних значень параметрів. Для кожного елемента конфігурації створюються відповідні команди та додаються до списку команд. Формування команд здійснюється відповідно до документації OpenWrt [24-25; 28-33]. Сформовані команди зберігаються у вигляді масиву та можуть бути використані в різних режимах роботи системи. Наприклад, для формування команд для використання сторонніх DNS-серверів WAN-інтерфейсу, виконується наступна послідовність команд (рис. 3.11) [32-33]:

1. вимкнення отримання DNS-серверів від WAN-інтерфейсу;
2. очищення списку сторонніх DNS-серверів;
3. розміщення DNS-серверів з конфігураційного файлу в команди для їх додання в список сторонніх DNS-серверів WAN-інтерфейсу.

```
wan = config.get("wan")
if not wan:
    return

dns_servers = wan.get("dns_servers")
if dns_servers:
    commands.append("uci set network.wan.peerdns='0'")
    commands.append("uci del network.wan.dns> /dev/null 2&>1")
    for dns_server in dns_servers:
        commands.append(f"uci add_list network.wan.dns='{dns_server}'")
```

Рисунок 3.11– Фрагмент коду, відповідальний за застосування сторонніх DNS-серверів з конфігураційного файлу до пристрою

Для взаємодії з маршрутизатором використовується протокол SSH. Підключення до пристрою та виконання команд реалізовано з використанням зовнішньої бібліотеки Python Paramiko [34]. Оскільки пристрої під керуванням OpenWrt по замовчуванню не мають паролю для входу, такий сценарій також обробляється програмою. Після встановлення з'єднання команди послідовно передаються на виконання, а результати їх виконання відображаються користувачу (рис. 3.12).

```
full_script = "\n".join(commands)

stdin, stdout, stderr = ssh.exec_command("sh -s")
stdin.write(full_script)
stdin.flush()
stdin.channel.shutdown_write()

output = stdout.read().decode()
errors = stderr.read().decode()

if output:
    print(output)

if errors:
    print("Errors:")
    print(errors)
```

Рисунок 3.12 – Фрагмент коду, відповідальний за відправлення команд на віддалений пристрій через SSH-клієнт

Окрім безпосереднього застосування команд, реалізовано можливість формування скриптового файлу. У цьому режимі всі згенеровані команди записуються у файл типу .sh, що дозволяє виконувати їх безпосередньо на пристрої. За допомогою вбудованого модулю Python «os» також застосовуються права на виконання файлу в операційній системі.

Також була реалізована дія отримання можливостей бездротових пристроїв. Отримання даних пристроїв виконується з використанням команди «iw list» операційної системи Linux [35]. Вивід команди передається на вхід модуля обробки можливостей бездротових пристроїв. Надалі модуль обробки збирає

можливості за ключовими словами (рис. 3.13), та виводить ці дані користувачу. Підтримувані можливості бездротових пристроїв формуються з урахуванням синтаксису конфігураційного файлу та можуть використовуватись для його написання.

```
if "HT40" in text:
    ht20(widths, modes)
    widths.append(40)
elif "HT20" in text:
    ht20(widths, modes)
if band == bands[1]:
    if "160 MHz" in text:
        width80(widths, modes)
        widths.append(160)
    elif "80 MHz" in text:
        width80(widths, modes)
if "HE Iftypes" in text:
    modes.append("ax")
```

Рисунок 3.13 – Частина коду обробки можливостей бездротових пристроїв, відповідальна за отримання режимів роботи Wi-Fi та ширин каналів

Вихідний код реалізованих модулів наведено в додатках: модуль wrtcfg.py — у додатку Б, validator.py — у додатку В, uci_builder.py — у додатку Г, ssh_client.py — у додатку Д, wifi_features_parser.py — у додатку Е, loader.py — у додатку Ж. Таким чином, програмне забезпечення реалізує повний цикл обробки конфігураційного файлу — від перевірки до застосування налаштувань на пристрої.

3.3 Тестування розробленої системи конфігурації маршрутизаторів на реальному обладнанні

Для перевірки коректності перетворення конфігураційного файлу в команди системи UCI системою, було використано команду «convert» системи з шляхом до зразкового JSON-файлу «config.json». Програмне забезпечення викликається через Python, тому команда для перетворення виконується в директорії з програмною та

формується наступним чином: «python wrtcfg.py convert <шлях до конфігураційного файлу>». Згідно з результатами інтерпретації (рис 3.14 – 3.19), команди формуються відповідно до документації OpenWrt [24-25; 28-33] та можуть використовуватись на реальних пристроях.

```
$ python wrtcfg.py convert /media/data/university/bach_work/config.json
Config is valid
uci del wireless.default_radio0> /dev/null 2&>1
uci set wireless.radio0.cell_density='0'
uci set wireless.radio0.channel='8'
uci set wireless.radio0.htmode='HE20'
uci del wireless.radio0_main1> /dev/null 2&>1
uci set wireless.radio0_main1='wifi-iface'
uci set wireless.radio0_main1.network='lan'
uci set wireless.radio0_main1.device='radio0'
uci set wireless.radio0_main1.mode='ap'
uci set wireless.radio0_main1.ssid='Home_2.4'
uci set wireless.radio0_main1.encryption='psk2'
uci set wireless.radio0_main1.key='12345678'
uci set wireless.radio0_main1.disabled='1'
uci del wireless.default_radio1> /dev/null 2&>1
uci set wireless.radio1.cell_density='0'
uci set wireless.radio1.channel='44'
uci set wireless.radio1.htmode='HE80'
uci del wireless.radio1_main1> /dev/null 2&>1
uci set wireless.radio1_main1='wifi-iface'
uci set wireless.radio1_main1.network='lan'
uci set wireless.radio1_main1.device='radio1'
uci set wireless.radio1_main1.mode='ap'
uci set wireless.radio1_main1.ssid='Home_5'
uci set wireless.radio1_main1.encryption='psk2'
uci set wireless.radio1_main1.key='12345678'
uci set wireless.radio1_main1.disabled='1'
```

Рисунок 3.14 – Виклик програми, перевірка конфігурації та отримані команди з конфігурацій секції «wifi» для пристроїв «radio0» та «radio1»

```
uci del network.guest_dev> /dev/null 2&>1
uci set network.guest_dev='device'
uci set network.guest_dev.name='br-guest'
uci set network.guest_dev.type='bridge'
uci del network.guest> /dev/null 2&>1
uci set network.guest='interface'
uci set network.guest.proto='static'
uci set network.guest.device='br-guest'
uci set network.guest.ipaddr='192.168.66.1'
uci set network.guest.netmask='255.255.255.0'
uci del dhcp.guest> /dev/null 2&>1
uci set dhcp.guest='dhcp'
uci set dhcp.guest.interface='guest'
uci set dhcp.guest.start='100'
uci set dhcp.guest.limit='200'
uci set dhcp.guest.leasetime='1h'
```

Рисунок 3.15 – Отримані команди для формування ізольованого мережевого пристрою та інтерфейсу, та створення DHCP-серверу для інтерфейсу «guest»

```
uci del firewall.guest_zone> /dev/null 2&>1
uci set firewall.guest_zone='zone'
uci set firewall.guest_zone.name='guest'
uci set firewall.guest_zone.network='guest'
uci set firewall.guest_zone.input='REJECT'
uci set firewall.guest_zone.output='ACCEPT'
uci set firewall.guest_zone.forward='REJECT'
uci del firewall.guest_wan> /dev/null 2&>1
uci set firewall.guest_wan='forwarding'
uci set firewall.guest_wan.src='guest'
uci set firewall.guest_wan.dest='wan'
uci del firewall.guest_dns> /dev/null 2&>1
uci set firewall.guest_dns='rule'
uci set firewall.guest_dns.name='Allow-DNS-Guest'
uci set firewall.guest_dns.src='guest'
uci set firewall.guest_dns.dest_port='53'
uci set firewall.guest_dns.proto='tcp udp'
uci set firewall.guest_dns.target='ACCEPT'
uci del firewall.guest_dhcp> /dev/null 2&>1
uci set firewall.guest_dhcp='rule'
uci set firewall.guest_dhcp.name='Allow-DHCP-Guest'
uci set firewall.guest_dhcp.src='guest'
uci set firewall.guest_dhcp.dest_port='67'
uci set firewall.guest_dhcp.proto='udp'
uci set firewall.guest_dhcp.target='ACCEPT'
```

Рисунок 3.16 – Сформовані команди для створення зони міжмережевого екрану для мережі «guest», налаштування направлення трафіку та формування правил дозволу на DNS та DHCP запити з зони

```
uci del wireless.radio0_guest1> /dev/null 2&>1
uci set wireless.radio0_guest1='wifi-iface'
uci set wireless.radio0_guest1.network='lan'
uci set wireless.radio0_guest1.device='radio0'
uci set wireless.radio0_guest1.mode='ap'
uci set wireless.radio0_guest1.network='guest'
uci set wireless.radio0_guest1.ssid='HomeGuests_2.4'
uci set wireless.radio0_guest1.encryption='psk2'
uci set wireless.radio0_guest1.key='87654321'
uci set wireless.radio0_guest1.disabled='1'
uci set wireless.radio0_guest1.isolate='1'
uci del wireless.radio1_guest1> /dev/null 2&>1
uci set wireless.radio1_guest1='wifi-iface'
uci set wireless.radio1_guest1.network='lan'
uci set wireless.radio1_guest1.device='radio1'
uci set wireless.radio1_guest1.mode='ap'
uci set wireless.radio1_guest1.network='guest'
uci set wireless.radio1_guest1.ssid='HomeGuests_5'
uci set wireless.radio1_guest1.encryption='psk2'
uci set wireless.radio1_guest1.key='87654321'
uci set wireless.radio1_guest1.disabled='1'
uci set wireless.radio1_guest1.isolate='1'
```

Рисунок 3.17 – Сформовані команди створення бездротових точок доступу для мережі «guest» за параметрами секції «guest_wifi»

```
uci del firewall.@defaults[0].syn_flood> /dev/null 2&>1
uci set firewall.@defaults[0].synflood_protect='1'
uci set firewall.@defaults[0].drop_invalid='1'
uci set firewall.@defaults[0].input='DROP'
uci set firewall.@defaults[0].forward='DROP'
uci set firewall.@zone[1].input='DROP'
uci set firewall.@zone[1].forward='DROP'
uci set firewall.guest_zone.input='DROP'> /dev/null 2&>1
uci set firewall.guest_zone.forward='DROP'> /dev/null 2&>1
uci set firewall.@defaults[0].flow_offloading='1'
uci set network.wan.peerdns='0'
uci del network.wan.dns> /dev/null 2&>1
uci add_list network.wan.dns='9.9.9.9'
uci add_list network.wan.dns='149.112.112.112'
```

Рисунок 3.18 – Сформовані команди відповідно до налаштувань секцій конфігураційного файлу «firewall» та «wan»

```
uci del dhcp.@dnsmasq[0].nonwildcard> /dev/null 2&>1
uci del dhcp.@dnsmasq[0].boguspriv> /dev/null 2&>1
uci del dhcp.@dnsmasq[0].filterwin2k> /dev/null 2&>1
uci del dhcp.@dnsmasq[0].filter_aaaa> /dev/null 2&>1
uci del dhcp.@dnsmasq[0].filter_a> /dev/null 2&>1
uci del dhcp.@dnsmasq[0].nonegcache> /dev/null 2&>1
uci del dhcp.@dnsmasq[0].cache_rr> /dev/null 2&>1
uci add_list dhcp.@dnsmasq[0].cache_rr='A'
uci add_list dhcp.@dnsmasq[0].cache_rr='CNAME'
uci add_list dhcp.@dnsmasq[0].cache_rr='PTR'
uci add_list dhcp.@dnsmasq[0].cache_rr='SRV'
uci set dhcp.@dnsmasq[0].cachesize='10000'
uci set dhcp.@dnsmasq[0].min_cache_ttl='60'
uci set dhcp.@dnsmasq[0].max_cache_ttl='86400'
uci set attendedsysupgrade.client.login_check_for_upgrades='1'
uci set system.@system[0].hostname='ProjectRouter'
uci set system.@system[0].zonename='Europe/Kyiv'
uci set system.@system[0].clock_hourcycle='h23'
uci del system.ntp.server
uci add_list system.ntp.server='0.ua.pool.ntp.org'
uci add_list system.ntp.server='1.ua.pool.ntp.org'
uci add_list system.ntp.server='2.ua.pool.ntp.org'
uci add_list system.ntp.server='3.ua.pool.ntp.org'
uci del dropbear.main.enable> /dev/null 2&>1
uci del dropbear.main.RootPasswordAuth> /dev/null 2&>1
uci del dropbear.main.Interface> /dev/null 2&>1
uci set dropbear.main._direct='1'
uci set dropbear.main.DirectInterface='lan'
uci set uhttpd.main.redirect_https='1'
echo -e 'passwordd\npasswordd' | passwd root> /dev/null 2&>1
uci commit
wifi reload
```

Рисунок 3.19 – Сформовані команди відповідно до налаштувань секцій «dns_caching» та «system», застосування виконаних налаштувань та перезавантаження підсистеми бездротової мережі

Для перевірки працездатності розробленого програмного забезпечення на реальному обладнанні було виконано його практичне застосування на двох маршрутизаторах під керуванням OpenWrt: Cudy WR3000P (рис. 1.2) та TP-Link Archer C50 (рис 3.20). Були виконані два сценарії на кожному з обраних пристроях: збір даних про бездротові пристрої та застосування конфігураційного файлу.



Рисунок 3.20 – Wi-Fi маршрутизатор TP-Link Archer C50, вид ззаду [36]

До двох маршрутизаторів виконано команду «get_wifi_features» розробленої системи. Отримані результати про можливості бездротових пристроїв маршрутизаторів (рис. 3.21, 3.22) відповідали їх реальним можливостям, та були використані при формуванні їх конфігураційних файлів.

```
$ /media/data/university/bach_work/venv/bin/python wrtcfg.py get_wifi_features
SSH username [root]:
SSH password [empty]:
radio0: 2.4GHz. Supported modes: n, ax. Supported widths: 20, 40
radio1: 5GHz. Supported modes: n, ac, ax. Supported widths: 20, 40, 80, 160
```

Рисунок 3.21 – Отримані дані про можливості бездротових пристроїв маршрутизатора Cudy WR3000P

```
$ /media/data/university/bach_work/venv/bin/python wrtcfg.py get_wifi_features
SSH username [root]:
SSH password [empty]:
radio0: 2.4GHz. Supported modes: n. Supported widths: 20, 40
radio1: 5GHz. Supported modes: n, ac. Supported widths: 20, 40, 80, 160
```

Рисунок 3.22 – Отримані дані про можливості бездротових пристроїв маршрутизатора TP-Link Archer C50

За допомогою сформованих конфігураційних файлів було виконано налаштування пристроїв з використанням доступних параметрів файлів конфігурації (рис. 3.23, 3.25). Додатково було виміряно час необхідний на виконання програми за допомогою вбудованої в Bash команди «time» [37]. Отримані результати (рис. 3.24, 3.26) підтвердили коректність роботи системи, правильність застосування та можливість її використання на різних платформах у реальних умовах.

```
(venv) [ wrtcfg]$ time python wrtcfg.py commit ../config_priv.json
Config is valid
SSH username [root]:
SSH password [empty]:

real    0m5.376s
user    0m0.285s
sys     0m0.055s
```

Рисунок 3.23 – Команда для застосування конфігураційного файлу «config_priv.json» до маршрутизатора Cudy WR3000P

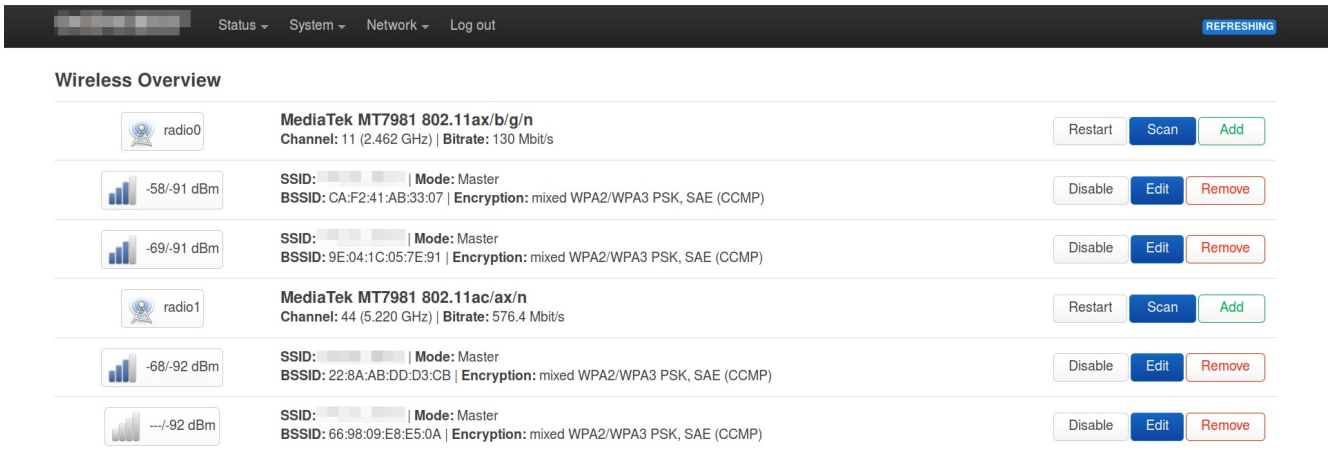


Рисунок 3.24 – Сформовані точки доступу маршрутизатора Cudy WR3000P в панелі Wireless вебінтерфейсу LuCI

```
(venv) [                wrtcfg]$ time python wrtcfg.py commit ../config_archer_c50.json
Config is valid
SSH username [root]:
SSH password [empty]:

real    0m6.846s
user    0m0.276s
sys     0m0.051s
```

Рисунок 3.25 – Команда для застосування конфігураційного файлу «config_archer_c50.json» до маршрутизатора TP-Link Archer C50

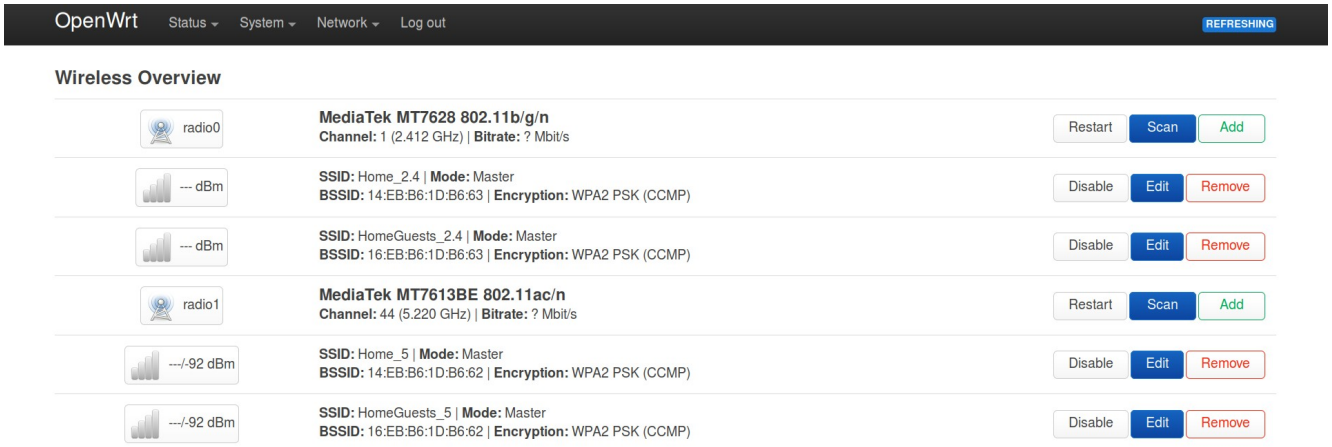


Рисунок 3.26 – Сформовані точки доступу маршрутизатора TP-Link Archer C50 в панелі Wireless вебінтерфейсу LuCI

Порівняно з ручним налаштуванням, розроблена система дозволяє суттєво зменшити кількість необхідних дій користувача, скоротити час конфігурації та потребує меншого досвіду роботи з системою UCI. Крім того, автоматизація процесу зменшує ймовірність виникнення помилок, пов’язаних з ручним введенням команд, а також забезпечує можливість повторного використання конфігураційних файлів.

Теоретично така система може бути використаною одночасно на декількох пристроях, якщо вони мають різні IP-адреси. Розроблена система є зручною у використанні через зменшену взаємодію з пристроєм та гнучкою, оскільки її можна адаптувати під інші, не OpenWrt платформи.

ВИСНОВКИ

У кваліфікаційній роботі було вирішено актуальне завдання автоматизації процесу конфігурації маршрутизаторів на базі операційної системи OpenWrt із використанням центрального конфігураційного файлу. Актуальність теми зумовлена складністю ручного налаштування пристроїв, необхідність використання спеціалізованих знань та високою ймовірністю виникнення помилок при роботі з командним інтерфейсом.

Перший розділ роботи окреслює загальні концепції конфігурації маршрутизаторів, до яких належать теоретичні основи функціонування маршрутизаторів, їх роль у сучасних мережах, та проаналізовано основні підходи до конфігурації. Особливу увагу приділено операційній системі OpenWrt та її засобам налаштування, зокрема системі UCI. Проведений аналіз дозволив виявити недоліки існуючих підходів, серед яких складність використання, відсутність уніфікованих механізмів автоматизації та обмежена придатність наявних рішень для OpenWrt. На основі цього було сформульовано проблемні питання та обґрунтовано доцільність розробки автоматизованої системи конфігурації.

У другому розділі проведено аналіз процесу налаштування маршрутизаторів вбудованими інструментами OpenWrt. Було розглянуто типові сценарії конфігурації, такі як створення бездротових точок доступу та організація ізольованого гостьового сегменту. Отримані результати продемонстрували, що ручне налаштування пристрою потребує виконання значної кількості команд та розуміння їх синтаксису, та є чутливим до помилок. На основі проведеного аналізу сформовано функціональні та технічні вимоги до автоматизованої системи, які враховують необхідність спрощення процесу налаштування, підвищення його надійності та гнучкості.

У третьому розділі розроблено автоматизовану систему конфігурації маршрутизаторів. Було визначено архітектуру програмного забезпечення, структуру конфігураційного файлу та алгоритм роботи системи. Реалізація виконана у вигляді програми командного рядку мовою програмування Python. Для забезпечення необхідного функціоналу програми були використані внутрішні та зовнішні модулі мови Python. Розроблена система забезпечує перевірку та валідацію конфігураційного файлу, перетворення його у команди системи UCI, а також їх застосування до маршрутизатора через протокол SSH. Додатково реалізовано функціонал формування скриптів та отримання інформації про можливості бездротових пристроїв.

Практичне застосування розробленої системи на реальних маршрутизаторах підтвердило її працездатність та коректність роботи. Використання запропонованого підходу зменшує кількість необхідних дій користувача, скорочує час конфігурації та знижує ймовірність виникнення помилок. Крім того, централізований конфігураційний файл забезпечує можливість повторного використання налаштувань та підвищує зручність адміністрування мережевих пристроїв.

Отже, поставлена мета роботи досягнута, а розроблена автоматизована система може бути використана для спрощення та підвищення ефективності процесу конфігурації маршрутизаторів на базі OpenWrt.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a router?. Cloudflare. URL: https://www.cloudflare.com/learning/network-layer/what-is-a-router (дата звернення: 27.04.2026).
2. What is an ISP?. Proton VPN. URL: https://protonvpn.com/blog/isp (дата звернення: 27.04.2026).
3. Cisco 1000 Series Integrated Services Routers Data Sheet. Cisco. URL: https://www.cisco.com/c/en/us/products/collateral/routers/1000-series-integrated-services-routers-isr/datasheet-c78-739512.html (дата звернення: 27.04.2026).
4. AX3000 2.5G Mesh Wi-Fi 6 Router. Cudy. URL: https://www.cudy.com/en-eu/products/wr3000h-1-0#features (дата звернення: 28.04.2026).
5. About the OpenWrt/LEDE project. OpenWrt. URL: https://openwrt.org/about (дата звернення: 28.04.2026).
6. OpenWrt Branding. GitHub. URL: https://github.com/openwrt/branding (дата звернення: 28.04.2026).
7. Reasons to use OpenWrt. OpenWrt. URL: https://openwrt.org/reasons_to_use_openwrt (дата звернення: 28.04.2026).
8. The UCI System. OpenWrt. URL: https://openwrt.org/docs/guide-user/base-system/uci (дата звернення: 29.04.2026).
9. LuCI Wiki. GitHub. URL: https://github.com/openwrt/luci/wiki (дата звернення: 29.04.2026).
10. Archer C80 Router Emulator. TP-Link. URL: https://emulator.tp-link.com/Archer%20C80%202.20-Router-Emulator/index.html (дата звернення: 29.04.2026).
11. How to create and restore from a backup file on your TP-Link Router. TP-Link. URL: https://www.tp-link.com/us/support/faq/2015 (дата звернення: 29.04.2026).
12. Looks like there are issues restoring a backup from a 5.7.4 controller to a 5.13.22. TP-Link Community. URL: https://community.tp-link.com/en/business/forum/topic/646188 (дата звернення: 29.04.2026).
13. What is SSH ? | Secure Shell (SSH) protocol. Cloudflare. URL: https://www.cloudflare.com/learning/access-management/what-is-ssh (дата звернення: 29.04.2026).
14. Managing Configuration Files Configuration Guide, Cisco IOS Release 15SY. Cisco. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/config-mgmt/configuration/15-sy/config-mgmt-15-sy-book/m_cm-config-files.html (дата звернення: 29.04.2026).

15. Difference between sh and bash. GeeksforGeeks. URL: https://www.geeksforgeeks.org/computer-networks/difference-between-sh-and-bash (дата звернення: 30.04.2026).

16. What is the Windows Subsystem for Linux?. Microsoft Learn. URL: https://learn.microsoft.com/en-us/windows/wsl/about (дата звернення: 30.04.2026).

17. Introduction to Ansible. Ansible Community Documentation. URL: https://docs.ansible.com/projects/ansible/latest/getting_started/introduction.html (дата звернення: 30.04.2026).

18. Community OpenWrt User Guide. Community.OpenWrt Collection Docs. URL: https://ansible-collections.github.io/community.openwrt/branch/main/docsite/user_guide.html (дата звернення: 30.04.2026).

19. What is Terraform | Terraform. Hashicorp. URL: https://developer.hashicorp.com/terraform/intro (дата звернення: 30.04.2026).

20. Getting Started With Mikrotik and Terraform. Mircea Anton. URL: https://mirceanton.com/posts/mikrotik-terraform-getting-started/#modifying-the-default-config (дата звернення: 01.05.2026).

21. OpenWrt Configurator. Github. URL: https://github.com/jasrusable/openwrt-configurator (дата звернення: 01.05.2026).

22. Introducing JSON. JSON.org. URL: https://www.json.org/json-en.html (дата звернення: 01.05.2026).

23. JSON files examples. Jsoning. URL: https://jsoning.com/examples (дата звернення: 01.05.2026).

24. Wi-Fi /etc/config/wireless. OpenWrt. URL: https://openwrt.org/docs/guide-user/network/wifi/basic (дата звернення: 02.05.2026).

25. Guest Wi-Fi basics. OpenWrt. URL: https://openwrt.org/docs/guide-user/network/wifi/guestwifi/guest-wlan (дата звернення: 02.05.2026).

26. Guest Wi-Fi using LuCI. OpenWrt. URL: https://openwrt.org/docs/guide-user/network/wifi/guestwifi/configuration_webinterface (дата звернення: 02.05.2026).

27. WiFi Passwords. RouterSecurity. URL: https://routersecurity.org/wifi.passwords.php (дата звернення: 02.05.2026).

28. Firewall configuration /etc/config/firewall. OpenWrt. URL: https://openwrt.org/docs/guide-user/firewall/firewall_configuration (дата звернення: 03.05.2026).

29. Dropbear configuration. OpenWrt. URL: https://openwrt.org/docs/guide-user/base-system/dropbear (дата звернення: 03.05.2026).

30. System configuration /etc/config/system. OpenWrt. URL: https://openwrt.org/docs/guide-user/base-system/system_configuration (дата звернення: 03.05.2026).

31. DHCP and DNS configuration /etc/config/dhcp. OpenWrt. URL: https://openwrt.org/docs/guide-user/base-system/dhcp (дата звернення: 03.05.2026).

32. Network configuration /etc/config/network. OpenWrt. URL: https://openwrt.org/docs/guide-user/network/network_configuration (дата звернення: 03.05.2026).

33. IPv4 configuration. OpenWrt. URL:https://openwrt.org/docs/guide-user/network/ipv4/configuration (дата звернення: 03.05.2026).

34. Welcome to Paramiko’s documentation!. Jeff Forcier. URL: https://docs.paramiko.org/en/stable (дата звернення: 03.05.2026).

35. About iw. Linux Wireless Documentation Project. URL: https://wireless.docs.kernel.org/en/latest/en/users/documentation/iw.html (дата звернення: 03.05.2026).

36. Archer C50 | AC1200 Wireless Dual Band Router. TP-Link. URL: https://www.tp-link.com/en/home-networking/wifi-router/archer-c50 (дата звернення: 04.05.2026).

37. time command in Linux with examples. GeeksforGeeks. URL: https://www.geeksforgeeks.org/linux-unix/time-command-in-linux-with-examples (дата звернення: 04.05.2026).

Зразковий конфігураційний файл JSON


```
{
    "wifi":{
        "devices":{
            "radio0":{
                "band": "2g",
                "mode": "ax",
                "channel": 8,
                "width": 20,
                "aps":[
                    {
                        "ssid": "Home_2.4",
                        "encryption": "wpa2",
                        "password": "12345678",
                        "disabled": true,
                        "isolate": false,
                        "random_macaddr": false
                    }
                ]
            },
            "radio1":{
                "band": "5g",
                "mode": "ax",
                "channel": 44,
                "width": 80,
                "aps":[
                    {
                        "ssid": "Home_5",
                        "encryption": "wpa2",
                        "password": "12345678",
                        "disabled": true,
                        "isolate": false,
                        "random_macaddr": false
                    }
                ]
            }
        }
    },
    "guest_wifi":{
        "devices":{
            "radio0":{
                "aps":[
                    {
                        "ssid": "HomeGuests_2.4",
                        "encryption": "wpa2",
                        "password": "87654321",
                        "disabled": true,
                        "isolate": true,
                        "random_macaddr": false
                    }
                ]
            },
            "radio1":{
                "aps":[
                    {
                        "ssid": "HomeGuests_5",
                        "encryption": "wpa2",
                        "password": "87654321",
```

```
                        "disabled": true,
                        "isolate": true,
                        "random_macaddr": false
                    }
                ]
            }
        }
    },
    "firewall":{
        "syn-flood_protection": true,
        "drop_invalid_packets": true,
        "drop_instead_of_reject": true,
        "flow_offloading": "software"
    },
    "wan":{
        "dns_servers":[
            "9.9.9.9",
            "149.112.112.112"
        ]
    },
    "dns_caching":{
        "cache_records":[
            "A",
            "CNAME",
            "PTR",
            "SRV"
        ],
        "cache_size": 10000,
        "min_cache_ttl": 60,
        "max_cache_ttl": 86400
    },
    "system":{
        "attendedsysupgrade": true,
        "hostname": "ProjectRouter",
        "timezone": "Europe/Kyiv",
        "time_format": "24h",
        "ssh_root_password": "passwordd",
        "ssh_through_lan_only": true,
        "redir_http_https": true,
        "ntp_servers":[
            "0.ua.pool.ntp.org",
            "1.ua.pool.ntp.org",
            "2.ua.pool.ntp.org",
            "3.ua.pool.ntp.org"
        ]
    }
}
```

Додаток Б

Вихідний код модуля wrtcfg.py

```
import sys
import os
import loader
import validator
import uci_builder
import ssh_client
import wifi_features_parser

def get_ssh_credentials():
    import getpass
    username = input("SSH username [root]: ") or "root"
    password = getpass.getpass("SSH password [empty]: ") or None
    return username, password

def write_script(commands, path):
    script = "#!/bin/sh\n"
    script += "\n".join(commands)

    with open(path, "w") as f:
        f.write(script)

    print(f"Script written to {path}")

def main():
    if len(sys.argv) < 2:
        print("Usage:")
        print("  wrtcfg check <config.json>")
        print("  wrtcfg make_script <config.json> [output.sh]")
        print("  wrtcfg commit <config.json> [ip]")
        print("  wrtcfg get_wifi_features [ip]")
        print("  wrtcfg convert <config.json>")
        return

    command = sys.argv[1]

    try:
        if command in ["check", "make_script", "commit", "convert"]:
            if len(sys.argv) < 3:
                print("Config required")
                return

            config_path = sys.argv[2]
            config = loader.load_config(config_path)

            if command == "check":
                validator.validate_config(config)
                return

            elif command == "make_script":
                validator.validate_config(config)
                commands = uci_builder.build_commands(config)
                output = sys.argv[3] if len(sys.argv) > 3 else "routercfg.sh"
                write_script(commands, output)
                os.chmod(output, 0o755)
                return
```

```
            elif command == "commit":
                ip = sys.argv[3] if len(sys.argv) > 3 else "192.168.1.1"
                validator.validate_config(config)
                commands = uci_builder.build_commands(config)
                username, password = get_ssh_credentials()
                ssh = ssh_client.connect(ip, username, password)
                ssh_client.execute_commands(ssh, commands)
                ssh_client.close_ssh(ssh)
                return

            elif command == "convert":
                validator.validate_config(config)
                commands = uci_builder.build_commands(config)
                for command in commands:
                    print(command)
                return

        elif command == "get_wifi_features":
            ip = sys.argv[2] if len(sys.argv) > 2 else "192.168.1.1"
            username, password = get_ssh_credentials()
            ssh = ssh_client.connect(ip, username, password)
            iw_output = ssh_client.execute_command(ssh, "iw list")
            wifi_features_parser.iw_parse(iw_output)
            ssh_client.close_ssh(ssh)
            return

        else:
            print(f"Unknown command: {command}")
            return

    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

Додаток В

Вихідний код модуля validator.py

```
bands = ["2g", "5g", "6g", "60g"]
modes = ["legacy", "n", "ax", "ac"]
widths = [20, 40, 80, 160]
encryptions = ["none", "wpa", "wpa+wpa2", "wpa2", "wpa3+wpa2", "wpa3"]

bool_fields_ap = ["disabled", "isolate", "random_macaddr"]
bool_fields_firewall     =     ["syn-flood_protection",     "drop_invalid_packets",
"drop_instead_of_reject"]
bool_fields_system      =      ["ssh_through_lan_only",      "attendedsysupgrade",
"redir_http_https"]

def section_error(section, msg):
    return f"{section}: {msg}"

def error(errors, msg):
    errors.append(msg)

def wifi_suffix(guest):
    return "guest_wifi" if guest else "wifi"

def validate_aps(radio, aps, guest, errors):
    suffix = wifi_suffix(guest)

    if aps is None:
        error(errors, section_error(f"{suffix}, {radio}",
        "missing aps section, leave it empty if no aps should be added"))
        return

    for i, ap in enumerate(aps):
        ssid = ap.get("ssid")
        if not ssid:
            error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
            "missing ssid"))
        if len(ssid) > 32:
            error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
            "ssid can only contain from 1 to 32 symbols in it"))

        encryption = ap.get("encryption")
        if not encryption:
            error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
            "missing encryption type"))
        elif encryption not in encryptions:
            error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
            f"nonexisting encryption type: {encryption}"))

        password = ap.get("password")
        if not password and encryption != encryptions[0]:
            error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
            "no password for specified encryption"))
        elif not password and encryption in encryptions[0]:
            return
        elif len(password) < 8:
            error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
            "password should be at least 8 symbols long"))
        elif len(password) > 63:
            error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
            "password is too long"))
```

```
        for field in bool_fields_ap:
            value = ap.get(field)
            if value is None:
                error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
                f"missing {field}"))
            elif not isinstance(value, bool):
                error(errors, section_error(f"{suffix}, {radio}, ap[{i}]",
                f"{field} should be boolean value"))

def validate_device_settings(radio, device, errors):
    suffix = wifi_suffix(False)

    band = device.get("band")
    if band is None:
        error(errors, section_error(f"{suffix}, {radio}",
        "missing band"))
    if band:
        if band in bands[2:]:
            error(errors, section_error(f"{suffix}, {radio}",
            f"{band} band handling is not implemented yet"))
        elif band not in bands[:2]:
            error(errors, section_error(f"{suffix}, {radio}",
            f"unknown band: {band}"))

    mode = device.get("mode")
    if mode is None:
        error(errors, section_error(f"{suffix}, {radio}",
        "missing mode"))
    if mode:
        if band == bands[0]:
            if mode not in modes[:3]:
                error(errors, section_error(f"{suffix}, {radio}",
                f"mode {mode} is not supported by 2.4GHz band"))
        elif band == bands[1]:
            if mode not in modes[1:4]:
                error(errors, section_error(f"{suffix}, {radio}",
                f"mode {mode} is not supported by 5GHz band"))

    channel = device.get("channel")
    if channel is None:
        error(errors, section_error(f"{suffix}, {radio}",
         "missing wifi channel"))
    elif channel == "auto":
        pass
    elif isinstance(channel, int):
        if band == bands[0]:
            if channel < 1 or channel > 13:
                error(errors, section_error(f"{suffix}, {radio}",
                f"channel {channel} is not supported by 2.4GHz band"))
        elif band == bands[1]:
            if channel < 36 or channel > 165:
                error(errors, section_error(f"{suffix}, {radio}",
                f"channel {channel} is not supported by 5GHz band"))
    else:
        error(errors, section_error(
            f"{suffix}, {radio}",
            f"invalid channel type: {type(channel)}, should be 'int' "\
            "if not 'auto'"))
```

```
    width = device.get("width")
    if width is None:
        error(errors, section_error(radio,
        "missing width of channel"))
    elif isinstance(width, int):
        if band == bands[0]:
            if mode == modes[0] and width != widths[0]:
                error(errors, section_error(radio,
                "width but 20 is not implementet for legacy mode yet"))
            elif width not in widths[:2]:
                error(errors, section_error(radio,
                f"unsupported width of channel for 2.4GHz: {width}"))
        if band == bands[1]:
            if width not in widths:
                error(errors, section_error(radio,
                f"unsupported width of channel for 5GHz: {width}"))
    else:
        error(errors, section_error(
        f"{suffix}, {radio}",
        "width should be integer value"))

def validate_wifi(config, errors, guest):
    wifi = config.get(wifi_suffix(guest))

    if wifi is None:
        return

    devices = wifi.get("devices")
    if devices is None:
        error(errors, section_error(wifi_suffix(guest),
        "no devices specified"))
        return

    for radio, device in devices.items():
        if not guest:
            validate_device_settings(radio, device, errors)
        aps = device.get("aps")
        validate_aps(radio, aps, guest, errors)

def validate_firewall(config, errors):
    firewall = config.get("firewall")
    if firewall is None:
        return

    for field in bool_fields_firewall:
        value = firewall.get(field)
        if value:
            if not isinstance(value, bool):
                error(errors, section_error("firewall",
                f"{field} should be boolean value"))

    flow = firewall.get("flow_offloading")
    if flow:
        if flow not in ["disabled", "software", "hardware"]:
            error(errors, section_error("firewall",
            f"invalid flow_offloading: {flow}"))

def validate_wan(config,errors):
    wan = config.get("wan")
    if wan is None:
        return
```

```
    dns_servers = wan.get("dns_servers")
    if not dns_servers:
        error(errors, section_error("wan",
        "dns_servers list is empty"))
        return

    import ipaddress

    for i, dns_server in enumerate(dns_servers):
        try:
            ipaddress.ip_address(dns_server)
        except ValueError:
            error(errors, section_error(
            "wan",
            f"invalid dns_server[{i}]: {dns_server}"
        ))

def validate_dns_caching(config, errors):
    dns = config.get("dns_caching")
    if dns is None:
        return

    cache_records = dns.get("cache_records")
    if cache_records:
        for i, cache_record in enumerate(cache_records):
            if not cache_record:
                error(errors, section_error("dns_caching",
                f"cache_record[{i}] can't be empty"))

    cache_size = dns.get("cache_size")
    if cache_size:
        if not isinstance(cache_size, int):
            error(errors, section_error("dns_caching",
            "cache_size should be integer value"))
        elif cache_size < 0:
            error(errors, section_error("dns_caching",
            "cache_size can't be an negative value"))
        elif cache_size > 10000:
            error(errors, section_error("dns_caching",
            "cache_size can't be higher than 10000"))

    min_cache_ttl = dns.get("min_cache_ttl")
    if min_cache_ttl:
        if not isinstance(min_cache_ttl, int):
            error(errors, section_error("dns_caching",
            "min_cache_ttl should be integer value"))
        elif min_cache_ttl < 0:
            error(errors, section_error("cache_size",
            "min_cache_ttl can't be an negative value"))
        elif min_cache_ttl > 3600:
            error(errors, section_error("cache_size",
            "min_cache_ttl can't be higher than 3600"))

    max_cache_ttl = dns.get("max_cache_ttl")
    if max_cache_ttl:
        if not isinstance(max_cache_ttl, int):
            error(errors, section_error("dns_caching",
            "min_cache_ttl should be integer value"))
        elif max_cache_ttl < 0:
            error(errors, section_error("cache_size",
            "min_cache_ttl can't be an negative value"))
```

```
def validate_system(config, errors):
    system = config.get("system")
    if system is None:
        return

    for field in bool_fields_system:
        value = system.get(field)
        if value:
            if not isinstance(value, bool):
                error(errors, section_error("system",
                f"{field} should be boolean value"))

    hostname = system.get("hostname")
    if hostname:
        if len(hostname) > 63:
            error(errors, section_error("system",
            "hostname too long (max 63 chars)"))
        for ch in hostname:
            if not ch.isalnum() or ch == "-":
                error(errors, section_error("system",
                "hostname contains invalid characters"))
        if hostname.startswith("-") or hostname.endswith("-"):
            error(errors, section_error("system",
            "hostname cannot start or end with '-'"))

    timezone = system.get("timezone")
    if timezone:
        from zoneinfo import ZoneInfo, ZoneInfoNotFoundError
        try:
            ZoneInfo(timezone)
        except ZoneInfoNotFoundError:
            error(errors, section_error("system",
            f"unknown timezone: {timezone}"))

    time_format = system.get("time_format")
    if time_format:
        if time_format not in ["12h", "24h"]:
            error(errors, section_error("system",
            f"unknown time format: {time_format}"))

    password = system.get("ssh_root_password")
    if password:
        if len(password) < 8:
            error(errors, section_error("system",
            "ssh password should be at least 8 symbols"))

    ntp_servers = system.get("ntp_servers")
    if ntp_servers:
        for i, ntp_server in enumerate(ntp_servers):
            if "." not in ntp_server:
                error(errors, section_error("system",
                f"invalid ntpserver[{i}]: {ntp_server}"))

def validate_config(config):
    errors = []

    validate_wifi(config, errors, False)
    validate_wifi(config, errors, True)
    validate_firewall(config, errors)
    validate_wan(config, errors)
    validate_dns_caching(config, errors)
```

Продовження додатка В

```
validate_system(config, errors)

if errors:
    for e in errors:
        print("Error:", e)
    raise Exception("Validation failed")
else:
    print("Config is valid")
```

Додаток Г

Вихідний код модуля uci_builder.py

```
mode_map = {
    "n": "HT",
    "ac": "VHT",
    "ax": "HE"
}

encryption_map = {
    "wpa": "psk",
    "wpa+wpa2": "psk-mixed",
    "wpa2": "psk2",
    "wpa3+wpa2": "sae-mixed",
    "wpa3": "sae"
}

dns_legacy   =   ["nonwildcard",   "boguspriv",   "filterwin2k",   "filter_aaaa",
"filter_a", "nonegcache"]

def build_htmode(mode, width):
    if mode == "legacy":
        return "NOHT"

    prefix = mode_map.get(mode)
    return f"{prefix}{width}"

def build_encryption(encryption):
    if encryption == "none":
        return encryption

    uci_enncryption = encryption_map.get(encryption)
    return uci_enncryption

def get_wifi_section(guest):
    return "guest_wifi" if guest else "wifi"

def build_aps(ap, radio, guest, commands, index):
    if not ap:
        return

    suffix = "guest" if guest else "main"
    iface_name = f"{radio}_{suffix}{index + 1}"

    commands.append(f"uci del wireless.{iface_name}> /dev/null 2&>1")
    commands.append(f"uci set wireless.{iface_name}='wifi-iface'")
    commands.append(f"uci set wireless.{iface_name}.network='lan'")
    commands.append(f"uci set wireless.{iface_name}.device='{radio}'")
    commands.append(f"uci set wireless.{iface_name}.mode='ap'")

    if guest:
        commands.append(f"uci set wireless.{iface_name}.network='guest'")

    ssid = ap.get("ssid")
    commands.append(f"uci set wireless.{iface_name}.ssid='{ssid}'")

    encryption = build_encryption(ap.get("encryption"))
    commands.append(f"uci set wireless.{iface_name}.encryption='{encryption}'")
    if encryption in ["sae-mixed", "sae"]:
        commands.append(f"uci set wireless.{iface_name}.ocv='0'")
```

```
    if encryption != "none":
        password = ap.get("password")
        commands.append(f"uci set wireless.{iface_name}.key='{password}'")

    disabled = ap.get("disabled")
    if disabled:
                                        commands.append(f"uci     set     wireless.
{iface_name}.disabled='{int(disabled)}'")

    isolate = ap.get("isolate")
    if isolate:
        commands.append(f"uci set wireless.{iface_name}.isolate='{int(isolate)}'")

    random_macaddr = ap.get("random_macaddr")
    if random_macaddr:
        commands.append(f"uci set wireless.{iface_name}.macaddr='random'")

def build_device_config(radio, device, commands):
    commands.append(f"uci del wireless.default_{radio}> /dev/null 2&>1")

    band = device.get("band")
    commands.append(f"uci set wireless.{radio}.cell_density='0'")

    channel = device.get("channel")
    commands.append(f"uci set wireless.{radio}.channel='{channel}'")

    mode = device.get("mode")
    width = device.get("width")
    htmode = build_htmode(mode, width)
    commands.append(f"uci set wireless.{radio}.htmode='{htmode}'")

def build_guest_config(commands):
    build_guest_device(commands)
    build_guest_interface(commands)
    build_guest_dhcp(commands)
    build_guest_firewall_zone(commands)
    build_guest_firewall_forwarding(commands)
    build_guest_firewall_dns_rule(commands)
    build_guest_firewall_dhcp_rule(commands)

def build_guest_device(commands):
    commands.append("uci del network.guest_dev> /dev/null 2&>1")
    commands.append("uci set network.guest_dev='device'")
    commands.append("uci set network.guest_dev.name='br-guest'")
    commands.append("uci set network.guest_dev.type='bridge'")

def build_guest_interface(commands):
    commands.append("uci del network.guest> /dev/null 2&>1")
    commands.append("uci set network.guest='interface'")
    commands.append("uci set network.guest.proto='static'")
    commands.append("uci set network.guest.device='br-guest'")
    commands.append("uci set network.guest.ipaddr='192.168.66.1'")
    commands.append("uci set network.guest.netmask='255.255.255.0'")

def build_guest_dhcp(commands):
    commands.append("uci del dhcp.guest> /dev/null 2&>1")
    commands.append("uci set dhcp.guest='dhcp'")
    commands.append("uci set dhcp.guest.interface='guest'")
    commands.append("uci set dhcp.guest.start='100'")
    commands.append("uci set dhcp.guest.limit='200'")
```

```
    commands.append("uci set dhcp.guest.leasetime='1h'")

def build_guest_firewall_zone(commands):
    commands.append("uci del firewall.guest_zone> /dev/null 2&>1")
    commands.append("uci set firewall.guest_zone='zone'")
    commands.append("uci set firewall.guest_zone.name='guest'")
    commands.append("uci set firewall.guest_zone.network='guest'")
    commands.append("uci set firewall.guest_zone.input='REJECT'")
    commands.append("uci set firewall.guest_zone.output='ACCEPT'")
    commands.append("uci set firewall.guest_zone.forward='REJECT'")

def build_guest_firewall_forwarding(commands):
    commands.append("uci del firewall.guest_wan> /dev/null 2&>1")
    commands.append("uci set firewall.guest_wan='forwarding'")
    commands.append("uci set firewall.guest_wan.src='guest'")
    commands.append("uci set firewall.guest_wan.dest='wan'")

def build_guest_firewall_dns_rule(commands):
    commands.append("uci del firewall.guest_dns> /dev/null 2&>1")
    commands.append("uci set firewall.guest_dns='rule'")
    commands.append("uci set firewall.guest_dns.name='Allow-DNS-Guest'")
    commands.append("uci set firewall.guest_dns.src='guest'")
    commands.append("uci set firewall.guest_dns.dest_port='53'")
    commands.append("uci set firewall.guest_dns.proto='tcp udp'")
    commands.append("uci set firewall.guest_dns.target='ACCEPT'")

def build_guest_firewall_dhcp_rule(commands):
    commands.append("uci del firewall.guest_dhcp> /dev/null 2&>1")
    commands.append("uci set firewall.guest_dhcp='rule'")
    commands.append("uci set firewall.guest_dhcp.name='Allow-DHCP-Guest'")
    commands.append("uci set firewall.guest_dhcp.src='guest'")
    commands.append("uci set firewall.guest_dhcp.dest_port='67'")
    commands.append("uci set firewall.guest_dhcp.proto='udp'")
    commands.append("uci set firewall.guest_dhcp.target='ACCEPT'")

def build_wifi(config, commands, guest):
    wifi = config.get(get_wifi_section(guest))
    if not wifi:
        return

    devices = wifi.get("devices")
    if not devices:
        return

    if guest:
        build_guest_config(commands)

    for radio, device in devices.items():
        if not guest:
            build_device_config(radio, device, commands)
        aps = device.get("aps", [])
        if aps:
            for i, ap in enumerate(aps):
                build_aps(ap, radio, guest, commands, i)

def build_firewall(config, commands):
    firewall = config.get("firewall")
    if not firewall:
        return
```

```
    synflood = firewall.get("syn-flood_protection")
    if synflood:
        commands.append("uci del firewall.@defaults[0].syn_flood> /dev/null 2&>1")
        commands.append("uci set firewall.@defaults[0].synflood_protect='1'")

    dropinvalid = firewall.get("drop_invalid_packets")
    if dropinvalid:
        commands.append("uci set firewall.@defaults[0].drop_invalid='1'")

    drop = firewall.get("drop_instead_of_reject")
    if drop:
        commands.append("uci set firewall.@defaults[0].input='DROP'")
        commands.append("uci set firewall.@defaults[0].forward='DROP'")
        commands.append("uci set firewall.@zone[1].input='DROP'")
        commands.append("uci set firewall.@zone[1].forward='DROP'")
             commands.append("uci  set  firewall.guest_zone.input='DROP'>  /dev/null
2&>1")
            commands.append("uci  set  firewall.guest_zone.forward='DROP'>  /dev/null
2&>1")

    flow = firewall.get("flow_offloading")
    if flow in ["software", "hardware"]:
        commands.append("uci set firewall.@defaults[0].flow_offloading='1'")
        if flow == "hardware":
                                                    commands.append("uci    set
firewall.@defaults[0].flow_offloading_hw='1'")

def build_wan(config, commands):
    wan = config.get("wan")
    if not wan:
        return

    dns_servers = wan.get("dns_servers")
    if dns_servers:
        commands.append("uci set network.wan.peerdns='0'")
        commands.append("uci del network.wan.dns> /dev/null 2&>1")
        for dns_server in dns_servers:
            commands.append(f"uci add_list network.wan.dns='{dns_server}'")

def build_dns_caching(config, commands):
    dns_caching = config.get("dns_caching")
    if not dns_caching:
        return

    for record in dns_legacy:
        commands.append(f"uci del dhcp.@dnsmasq[0].{record}> /dev/null 2&>1")

    cache_records = dns_caching.get("cache_records")
    if cache_records:
        commands.append("uci del dhcp.@dnsmasq[0].cache_rr> /dev/null 2&>1")
        for cache_record in cache_records:
                                              commands.append(f"uci    add_list
dhcp.@dnsmasq[0].cache_rr='{cache_record}'")

    cache_size = dns_caching.get("cache_size")
    if cache_size:
        commands.append(f"uci set dhcp.@dnsmasq[0].cachesize='{cache_size}'")

    min_cache_ttl = dns_caching.get("min_cache_ttl")
```

```
    if min_cache_ttl:
                                                    commands.append(f"uci      set
dhcp.@dnsmasq[0].min_cache_ttl='{min_cache_ttl}'")

    max_cache_ttl = dns_caching.get("max_cache_ttl")
    if max_cache_ttl:
                                                    commands.append(f"uci      set
dhcp.@dnsmasq[0].max_cache_ttl='{max_cache_ttl}'")

def build_system(config, commands):
    system = config.get("system")
    if not system:
        return

    sysupgrade = system.get("attendedsysupgrade")
    if sysupgrade:
                                                     commands.append("uci      set
attendedsysupgrade.client.login_check_for_upgrades='1'")
    else:
                                                     commands.append("uci      set
attendedsysupgrade.client.login_check_for_upgrades='0'")

    hostname = system.get("hostname")
    if hostname:
        commands.append(f"uci set system.@system[0].hostname='{hostname}'")

    timezone = system.get("timezone")
    if timezone:
        commands.append(f"uci set system.@system[0].zonename='{timezone}'")

    time_format = system.get("time_format")
    if time_format:
        time_format_map = {
            "12h": "h12",
            "24h": "h23",
        }
                                                    commands.append(f"uci      set
system.@system[0].clock_hourcycle='{time_format_map.get(time_format)}'")

    ntp_servers = system.get("ntp_servers")
    if ntp_servers:
        commands.append("uci del system.ntp.server")
        for ntp_server in ntp_servers:
            commands.append(f"uci add_list system.ntp.server='{ntp_server}'")

    sshlan = system.get("ssh_through_lan_only")
    if sshlan:
        commands.append("uci del dropbear.main.enable> /dev/null 2&>1")
        commands.append("uci del dropbear.main.RootPasswordAuth> /dev/null 2&>1")
        commands.append("uci del dropbear.main.Interface> /dev/null 2&>1")
        commands.append("uci set dropbear.main._direct='1'")
        commands.append("uci set dropbear.main.DirectInterface='lan'")

    redir = system.get("redir_http_https")
    if redir:
        commands.append("uci set uhttpd.main.redirect_https='1'")

    password = system.get("ssh_root_password")
    if password:
              commands.append(f"echo  -e  '{password}\\n{password}'  |  passwd  root>
/dev/null 2&>1")
```

```
def build_commands(config):
    commands = []
    build_wifi(config, commands, False)
    build_wifi(config, commands, True)
    build_firewall(config, commands)
    build_wan(config, commands)
    build_dns_caching(config, commands)
    build_system(config, commands)
    commands.append("uci commit")
    commands.append("wifi reload")
    return commands
```

Додаток Д

Вихідний код модуля ssh_client.py

```
def connect(ip, username, password):
    import paramiko
    import socket

    sock = socket.create_connection((ip, 22), timeout=5)

    transport = paramiko.Transport(sock)
    transport.banner_timeout = 5
    transport.auth_timeout = 5

    transport.start_client(timeout=5)

    try:
        if password:
            transport.auth_password(username, password)
        else:
            transport.auth_none(username)
    except paramiko.BadAuthenticationType:
        raise Exception("Authentication failed")

    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh._transport = transport

    return ssh

def close_ssh(ssh):
    if ssh:
        ssh.close()

def execute_commands(ssh, commands):
    full_script = "\n".join(commands)

    stdin, stdout, stderr = ssh.exec_command("sh -s")
    stdin.write(full_script)
    stdin.flush()
    stdin.channel.shutdown_write()

    output = stdout.read().decode()
    errors = stderr.read().decode()

    if output:
        print(output)

    if errors:
        print("Errors:")
        print(errors)

def execute_command(ssh, command):
    stdin, stdout, stderr = ssh.exec_command(command)
    return stdout.read().decode()
```

Вихідний код модуля wifi_features_parser.py

```
bands = ["2.4", "5"]
keywords = [""]

def ht20(widths, modes):
    widths.append(20)
    modes.append("n")

def width80(widths, modes):
    widths.append(80)
    modes.append("ac")

def iw_parse(iw_output):
    result = {}

    blocks = iw_output.split("Wiphy ")

    for block in blocks[1:]:
        lines = block.strip().splitlines()

        phy = lines[0].strip()
        radio = phy.replace("phy", "radio")

        band = None
        widths = []
        modes = []

        text = "\n".join(lines)

        if "Band 1" in text:
            band = bands[0]
        elif "Band 2" in text:
            band = bands[1]

        if "HT40" in text:
            ht20(widths, modes)
            widths.append(40)
        elif "HT20" in text:
            ht20(widths, modes)

        if band == bands[1]:
            if "160 MHz" in text:
                width80(widths, modes)
                widths.append(160)
            elif "80 MHz" in text:
                width80(widths, modes)
        if "HE Iftypes" in text:
            modes.append("ax")

        result[radio] = {
            "phy": phy,
            "band": band,
            "widths": widths,
            "modes": modes
        }

    for radio, data in sorted(result.items()):
        band = f'{data["band"]}GHz'
```

```
        modes = ", ".join(data["modes"])
        widths = ", ".join(map(str, data["widths"]))
            print(f"{radio}: {band}. Supported modes: {modes}. Supported widths:
{widths}")
```

Додаток Ж

Вихідний код модуля loader.py

```
def load_config(path):
    import json
    try:
        with open(path, "r") as f:
            return json.load(f)
    except FileNotFoundError:
        raise Exception(f"Config file not found: {path}")
    except json.JSONDecodeError:
        raise Exception("Invalid JSON format")
```

Додаток И

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій
Кафедра Комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

на тему: «Автоматизована система конфігурації маршрутизатора на базі OpenWRT з використанням центрального конфігураційного файлу**»**

Виконав студент групи КІД-41
Лясковський Даніїл Олександрович

Керівник кваліфікаційної роботи:
Бученко Ігор Анатолійович,
старший викладач кафедри КІ

Київ – 2026

Мета роботи – Розробка автоматизованої системи конфігурації маршрутизатора на базі OpenWRT з використанням центрального конфігураційного файлу для спрощення конфігурації пристрою.

Об`єкт дослідження – Процес конфігурації мережевих маршрутизаторів на базі операційної системи OpenWRT

Предмет дослідження – Методи та засоби автоматизації конфігурації маршрутизаторів

Актуальність обраної теми

– Маршрутизатори є ключовими елементами комп'ютерних мереж, від налаштування яких залежать продуктивність, стабільність та безпека мережі.
– Операційна система OpenWrt надає широкі можливості конфігурації, проте її використання через систему UCI передбачає виконання великої кількості команд вручну, що ускладнює процес адміністрування.
– Ручне налаштування є неефективним при роботі з великою кількістю пристроїв та підвищує ймовірність помилок.
– Використання централізованого конфігураційного файлу у форматі JSON та автоматичне перетворення його у команди UCI дозволяє спростити процес налаштування.
– Розробка автоматизованих засобів конфігурації маршрутизаторів є актуальним завданням для підвищення ефективності та надійності мережевого адміністрування.

Проблематика конфігурації маршрутизаторів

– відсутність єдиного підходу до конфігурації
– відсутність зручних засобів автоматизації
– повторюваність команд у типових сценаріях
– необхідність глибоких знань конфігураційної системи при роботі з командним рядком
– висока ймовірність помилок при виконанні повторюваних сценаріїв
– складність використання як командного рядку, так і вебінтерфейсу

Засоби конфігурації OpenWrt та їх недоліки

```
[                    ~]$ ssh root@192.168.1.1
root@192.168.1.1's password:


BusyBox v1.37.0 (2026-03-24 21:24:12 UTC) built-in shell (ash)

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 OpenWrt 25.12.2, r32802-f505120278 Dave's Guitar
 -----------------------------------------------------

 OpenWrt recently switched to the "apk" package manager!

 OPKG Command           APK Equivalent      Description
 ------------------------------------------------------------------
 opkg install <pkg>     apk add <pkg>       Install a package
 opkg remove <pkg>      apk del <pkg>       Remove a package
 opkg upgrade           apk upgrade         Upgrade all packages
 opkg files <pkg>       apk info -L <pkg>   List package contents
 opkg list-installed    apk info            List installed packages
 opkg update            apk update          Update package lists
 opkg search <pkg>      apk search <pkg>    Search for packages
 ------------------------------------------------------------------

For more information visit:
https://openwrt.org/docs/guide-user/additional-software/opkg-to-apk-cheatsheet

root@                :~# echo $PWD
/root
```

UCI – командний рядок
– складний синтаксис
– повторюваність команд
– необхідність знання конфігураційної системи
– відсутність вбудованої в систему документації

Рис. 1.12 – Демонстрація використання SSH-клієнта для підключення до OpenWrt роутеру

5

Засоби конфігурації OpenWrt та їх недоліки

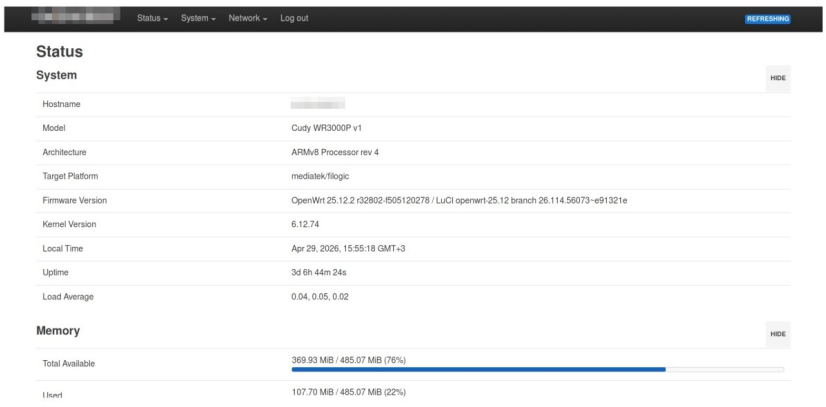


Рисунок 1.6 – Головна сторінка вебінтерфейсу LuCI

LuCI – веб-інтерфейс для налаштувань, використовує та застосовує команди UCI
– не має засобів автоматизації
– більша повторюваність дій
– вищий ризик на помилку або пропуск налаштувань
– інструмент для виконання дій, але не сценаріїв
– орієнтований на ручне налаштування, а не автоматизацію

6

Постановка задачі

До системи автоматизованої конфігурації висуваються такі вимоги:
– використання файлу конфігурації JSON;
– мінімальна прив’язка до синтаксису UCI;
– автоматизація процесу конфігурації;
– мінімізація помилок при виконанні типових сценаріїв;
– різні режими роботи;
– забезпечення гнучкості при застосуванні.

7

Аналіз процесу налаштування UCI

`1. uci set dhcp.guest='dhcp'`	// встановлення значення параметра «guest» у конфігурації «dhcp»
`2. uci set wireless.device0.htmode='HE40'`	// встановлення режиму Wi-Fi 6 (802.11ax) з шириною каналу 40 MHz для пристрою «device0»
`3. uci set firewall.@defaults[0].input='DROP`	// встановлення політики за замовчуванням для вхідного трафіку (INPUT) у «firewall» як «DROP»
`4. uci del wireless.default_radio0`	// видалення секції default_radio0 з конфігурації wireless

Без контексту ці команди:
– мають заплутаний синтаксис;
– потребують знання конфігураційних файлів;
– повторюються в схожих сценаріях.

8

Типові сценарії налаштування

1. Створення точок доступу на пристрої передбачає:
– налаштування бездротових пристроїв;
– створення відповідних точок доступу з прив’язкою до пристроїв;
– налаштування параметрів точок доступу;
– увімкнення бездротових пристроїв;

2. Створення гостьового сегменту та його точок доступу передбачає:
– створення ізольованого віртуального пристрою та інтерфейсу;
– налаштування DHCP-серверу для інтерфейсу;
– створення зони міжмережевого екрану, її налаштування;
– створення правил для роботи DHCP та DNS всередині зони;
– створення та налаштування необхідних точок доступу.

9

Архітектура системи. Зразок з формуванням та виводом команд



– валідатор відповідає за коректність параметрів конфігураційного файлу
– інтерпретатор виконує перетворення параметрів з файлу в команди системи UCI
– команди розміщуються в програмі як масив
– команди з масиву об’єднуються та виводяться в консоль користувача

Рисунок 3.1 – Послідовність дій при перетворенні JSON-файлу в команди

10

Структура конфігураційного файлу

```
"radio0":{
    "band": "2g",
    "mode": "ax",
    "channel": 8,
    "width": 20,
    "aps":[
        {
            "ssid": "Home_2.4",
            "encryption": "wpa2",
            "password": "12345678",
            "disabled": true,
            "isolate": false,
            "random_macaddr": false
        }
    ]
```

Рисунок 3.2 – Зразок секції «wifi» конфігураційного файлу з заданими параметрами бездротового пристрою «radio0» та зразковою точкою доступу

Конфігураційний файл розділено на наступні секції:
– «wifi»
– «guest_wifi»
– «firewall»
– «wan»
– «dns_caching»
– «system»
Секції є незалежними та можуть бути опціональними

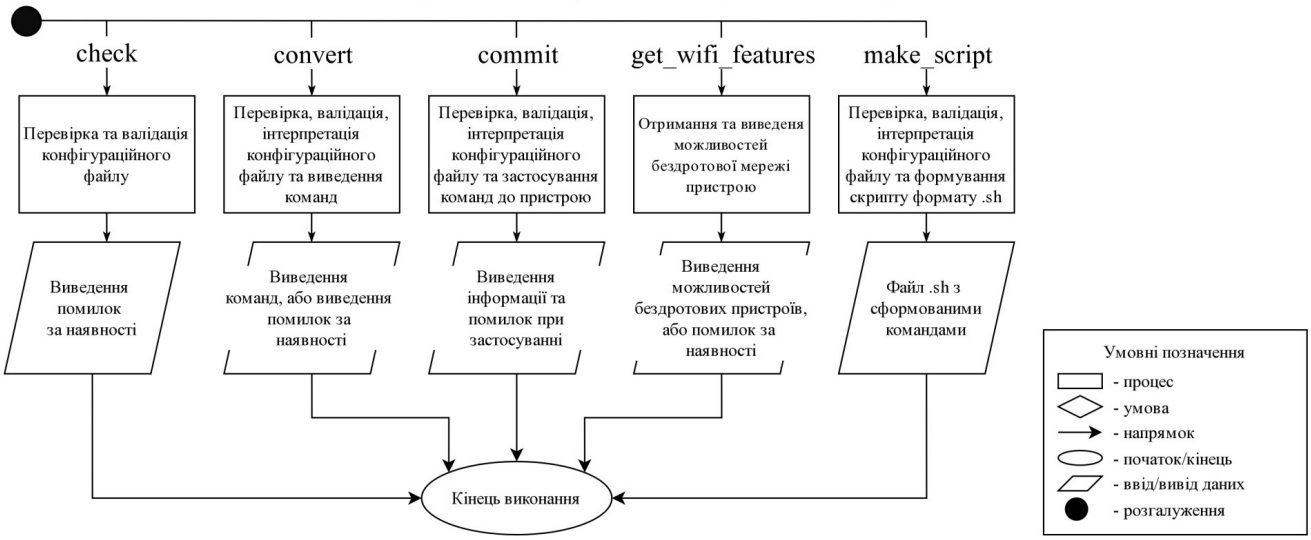
11

Спрощений алгоритм функціонування системи (до розгалуження)



12

Спрощений алгоритм функціонування системи (після розгалуження)



13

Реалізація системи

```
wan = config.get("wan")
if not wan:
    return

dns_servers = wan.get("dns_servers")
if dns_servers:
    commands.append("uci set network.wan.peerdns='0'")
    commands.append("uci del network.wan.dns> /dev/null 2&>1")
    for dns_server in dns_servers:
        commands.append(f"uci add_list network.wan.dns='{dns_server}'")
```

Рисунок 3.11– Фрагмент коду, відповідальний за застосування сторонніх DNS-серверів з конфігураційного файлу до пристрою

– написана мовою Python
– використані модулі: wrtcfg.py, loader.py, validator.py, uci_builder.py, ssh_client.py, wifi_features_parser.py
– SSH-підключення до пристрою виконується за допомогою зовнішньої бібліотеки Python Paramiko
– для отримання апаратних можливостей бездротових пристроїв використовується команда «iw list»

14

Демонстрація роботи

```
"wifi":{
    "devices":{
        "radio0":{
            "band": "2g",
            "mode": "ax",
            "channel": 8,
            "width": 20,
            "aps":[
                {
                    "ssid": "Home_2.4",
                    "encryption": "wpa2",
                    "password": "12345678",
                    "disabled": true,
                    "isolate": false,
                    "random_macaddr": false
                }
            ]
        },
```

Рисунок 3.2 – Зразок секції «wifi» конфігураційного файлу з заданими параметрами бездротового пристрою «radio0» та зразковою точкою доступу

```
$ python wrtcfg.py convert /media/data/university/bach_work/config.json
Config is valid
uci del wireless.default_radio0> /dev/null 2&>1
uci set wireless.radio0.cell_density='0'
uci set wireless.radio0.channel='8'
uci set wireless.radio0.htmode='HE20'
uci del wireless.radio0_main1> /dev/null 2&>1
uci set wireless.radio0_main1='wifi-iface'
uci set wireless.radio0_main1.network='lan'
uci set wireless.radio0_main1.device='radio0'
uci set wireless.radio0_main1.mode='ap'
uci set wireless.radio0_main1.ssid='Home_2.4'
uci set wireless.radio0_main1.encryption='psk2'
uci set wireless.radio0_main1.key='12345678'
uci set wireless.radio0_main1.disabled='1'
uci del wireless.default_radio1> /dev/null 2&>1
uci set wireless.radio1.cell_density='0'
uci set wireless.radio1.channel='44'
uci set wireless.radio1.htmode='HE80'
uci del wireless.radio1_main1> /dev/null 2&>1
uci set wireless.radio1_main1='wifi-iface'
uci set wireless.radio1_main1.network='lan'
uci set wireless.radio1_main1.device='radio1'
uci set wireless.radio1_main1.mode='ap'
uci set wireless.radio1_main1.ssid='Home_5'
uci set wireless.radio1_main1.encryption='psk2'
uci set wireless.radio1_main1.key='12345678'
uci set wireless.radio1_main1.disabled='1'
```

Рис. 3.14 – Виклик програми, перевірка конфігурації та отримані команди з конфігурацій секції «wifi» для пристроїв «radio0» та «radio1»

15

Ефективність розробленої системи

– автоматична генерація команд;
– різні режими використання;
– зрозумілий синтаксис конфігураційного файлу;
– швидке налаштування;
– менша кількість помилок;
– повторне використання конфігурації на різних платформах;
– можливість масштабування.

16

ВИСНОВКИ

У кваліфікаційній роботі було вирішено актуальне завдання автоматизації процесу конфігурації маршрутизаторів на базі операційної системи OpenWrt із використанням центрального конфігураційного файлу. Актуальність теми зумовлена складністю ручного налаштування пристроїв, необхідність використання спеціалізованих знань та високою ймовірністю виникнення помилок при роботі з командним інтерфейсом.

Перший розділ роботи окреслює загальні концепції конфігурації маршрутизаторів, до яких належать теоретичні основи функціонування маршрутизаторів, їх роль у сучасних мережах, та проаналізовано основні підходи до конфігурації. Особливу увагу приділено операційній системі OpenWrt та її засобам налаштування, зокрема системі UCI. Проведений аналіз дозволив виявити недоліки існуючих підходів, серед яких складність використання, відсутність уніфікованих механізмів автоматизації та обмежена придатність наявних рішень для OpenWrt. На основі цього було сформульовано проблемні питання та обґрунтовано доцільність розробки автоматизованої системи конфігурації.

17

ПРОДОВЖЕННЯ ВИСНОВКІВ

У другому розділі проведено аналіз процесу налаштування маршрутизаторів вбудованими інструментами OpenWrt. Було розглянуто типові сценарії конфігурації, такі як створення бездротових точок доступу та організація ізольованого гостьового сегменту. Отримані результати продемонстрували, що ручне налаштування пристрою потребує виконання значної кількості команд та розуміння їх синтаксису, та є чутливим до помилок. На основі проведеного аналізу сформовано функціональні та технічні вимоги до автоматизованої системи, які враховують необхідність спрощення процесу налаштування, підвищення його надійності та гнучкості.

У третьому розділі розроблено автоматизовану систему конфігурації маршрутизаторів. Було визначено архітектуру програмного забезпечення, структуру конфігураційного файлу та алгоритм роботи системи. Реалізація виконана у вигляді програми командного рядку мовою програмування Python. Для забезпечення необхідного функціоналу програми були використані внутрішні та зовнішні модулі мови Python. Розроблена система забезпечує перевірку та валідацію конфігураційного файлу, перетворення його у команди системи UCI, а також їх застосування до маршрутизатора через протокол SSH. Додатково реалізовано функціонал формування скриптів та отримання інформації про можливості бездротових пристроїв.

18

ПРОДОВЖЕННЯ ВИСНОВКІВ

Практичне застосування розробленої системи на реальних маршрутизаторах підтвердило її працездатність та коректність роботи. Використання запропонованого підходу зменшує кількість необхідних дій користувача, скорочує час конфігурації та знижує ймовірність виникнення помилок. Крім того, централізований конфігураційний файл забезпечує можливість повторного використання налаштувань та підвищує зручність адміністрування мережевих пристроїв.

Отже, поставлена мета роботи досягнута, а розроблена автоматизована система може бути використана для спрощення та підвищення ефективності процесу конфігурації маршрутизаторів на базі OpenWrt.

19

РЕЗУЛЬТАТИ ВПРОВАДЖЕННЯ

Бученко І.А., Лясковський Д.О. Розробка системи автоматизованої конфігурації маршрутизаторів на базі OpenWrt. «Сучасні інтелектуальні інформаційні технології в науці та освіті» : VI Всеукраїнська науково-практ. конф., м.Київ, 15 трав. 2026 р. Подано до друку.

20