Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: "Моделювання архітектури мережі Fog Computing для ефективної та безпечної обробки IoT-даних"

Виконав: студент (ка) IV курсу, групи СБ-41

Спеціальності:
125 «Кібербезпека»
(шифр і назва напряму підготовки, спеціальності)
Малюта Ярослав Романович
підпис (прізвище та ініціали)

Керівник Деркач М.В.
підпис (прізвище та ініціали)

Нормоконтроль Дроздова Т.В.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент Паламар А.М.
підпис (прізвище та ініціали)

м. Тернопіль – 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Малюті Ярославу Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделювання архітектури мережі Fog Computing для ефективної та безпечної обробки IoT-даних

Керівник роботи Деркач Марина Володимирівна, к.т.н., доцент кафедри КБ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 02 » 05 2025 року № 4/7-361

2. Термін подання студентом завершеної роботи 17.06.2025

3. Вихідні дані до роботи Характеристики IoT-пристроїв, параметри мережевої інфраструктури

4. Зміст роботи (перелік питань, які потрібно розробити)
Провести аналіз існуючих підходів до обробки IoT-даних у Fog-мережах
Розробити модель архітектури Fog Computing
Провести моделювання архітектури та оцінити її ефективність і безпеку
Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Тема, мета, задачі. Інфографіка з даними про IoT. Архітектура Fog-системи. Механізми безпеки та автентифікації. Схеми автентифікації для IoT. Графік латентності Fog vs Cloud. Графік мережевого трафіку Fog vs Cloud. Піктограми застосування Fog-систем. Підсумкова інфографіка. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Мариненко С.Ю., к.т.н., доцент кафедри МТ		

7. Дата видачі завдання 29.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.01 – 02.02	Виконано
2.	Підбір джерел для аналізу існуючих підходів до обробки IoT-даних у Fog-мережах	03.01 – 13.02	Виконано
3.	Опрацювання джерел в галузі дослідження	14.02 – 23.02	Виконано
4.	Провести аналіз існуючих підходів до обробки IoT-даних у Fog-мережах	24.02 – 12.03	Виконано
5.	Розроблення моделі архітектури Fog Computing	15.03 – 25.03	Виконано
6.	Проведення моделювання архітектури та оцінення її ефективність і безпеку	25.02 – 10.04	Виконано
7.	Оформлення розділу «Аналіз сучасних підходів до обробки IoT-даних у Fog Computing»	10.02 – 05.03	Виконано
8.	Оформлення розділу «Розробка моделі архітектури Fog Computing для обробки IoT-даних»	26.03 – 04.05	Виконано
9.	Оформлення розділу «Моделювання запропонованої архітектури»	12.04 – 20.04	
10.	Виконання завдання до підрозділу «Безпека життєдіяльності, основи охорони праці»	25.04 – 10.05	Виконано
11.	Оформлення кваліфікаційної роботи	23.05 – 10.06	Виконано
12.	Нормоконтроль	11.06 – 16.06	Виконано
13.	Перевірка на плагіат	16.06 – 22.06	Виконано
14.	Попередній захист кваліфікаційної роботи	22.06 – 23.06	Виконано
15.	Захист кваліфікаційної роботи	24.06.2025	

Студент _____ Малюта Я.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Деркач М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Моделювання архітектури мережі Fog Computing для ефективної та безпечної обробки IoT-даних // Кваліфікаційна робота ОР «Бакалавр» // Малюта Ярослав Романович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБ-41 // Тернопіль, 2025 // С. 85, рис. – 15, табл. – 5 , кресл. – -, додат. – 2.

Ключові слова: Fog Computing, моделювання, архітектура, безпека, IoT, QoS, Cloud.

Кваліфікаційна робота зосереджена на дослідженні та розробці ефективної архітектури мережі туманних обчислень, адаптованої для безпечної обробки даних Інтернету речей (IoT). У ній аналізуються існуючі проблеми, пов'язані з централізованою обробкою даних IoT у хмарних середовищах, а саме висока затримка, обмеження пропускної здатності мережі та проблеми масштабованості.

Основна увага приділяється моделюванню розробленої моделі архітектури туманних обчислень, яка дозволяє ефективно розподіляти обчислювальні ресурси між пристроями IoT, вузлами туманних обчислень та хмарною інфраструктурою. Запропонована модель включає механізми шифрування даних, протоколи автентифікації вузлів та методи виявлення аномалій для забезпечення високого рівня інформаційної безпеки.

ABSTRACT

Modeling of Fog Computing network architecture for efficient and secure processing of IoT data // Thesis of educational level "Bachelor" // Malyuta Yaroslav Romanovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, group СБ-41 // Ternopil, 2025 // P. 85 fig. – 15, tab. – 5, drawing – -, added. – 2.

Keywords: Fog Computing, modeling, architecture, security, IoT, QoS, Cloud.

The qualification work focuses on the research and development of an efficient fog computing network architecture adapted for secure processing of Internet of Things (IoT) data. It analyzes the existing problems associated with centralized processing of IoT data in cloud environments, namely high latency, network bandwidth limitations, and scalability issues.

The main focus is on modeling the developed fog computing architecture model, which allows for efficient distribution of computing resources between IoT devices, fog computing nodes, and cloud infrastructure. The proposed model includes data encryption mechanisms, node authentication protocols, and anomaly detection methods to ensure a high level of information security.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IoT	—	Internet of Things
API	—	Application Programming Interface
ABAC	—	Attribute-Based Access Control
SMPC	—	Secure Multi-party Computation
TPM	—	Trusted Platform Modules
VPN	—	Virtual Private Network
DDoS	—	Distributed Denial of Service
SDN	—	Software-defined Networking
IPS	—	Intrusion Prevention System
IDS	—	Intrusion Detection System
PFS	—	Perfect Forward Secrecy
ECC	—	Elliptic Curve Cryptography
TLS	—	Transport Layer Security
0-RTT	—	Zero Round Trip Time
DTLS	—	Datagram Transport Layer Security
0-RTT	—	Zero Round Trip Time
LSTM	—	Long Short-Term Memory

ВСТУП

Актуальність теми. Традиційні методології обробки даних Інтернету речей, що спираються на централізовані хмарні обчислення, стикаються зі значними недоліками: значною затримкою мережі, обмеженнями пропускної здатності каналів зв'язку та проблемами, пов'язаними з масштабованістю та конфіденційністю даних.

Туманні обчислення, як парадигма розподілених обчислень, пропонують життєздатну альтернативу, розміщуючи обчислювальні ресурси ближче до джерел даних, на межі мережі. Така архітектура суттєво зменшує затримки обробки, оптимізує використання пропускної здатності мережі та підвищує швидкість реагування систем Інтернету речей.

Мета і задачі дослідження. Головною метою цієї кваліфікаційної роботи є дослідження побудови архітектури туманних обчислень для обробки даних Інтернету речей з метою підвищення ефективності та профілю безпеки системи.

Для успішного досягнення цієї основної мети необхідно вирішити такі завдання:

– аналіз існуючих методів обробки даних Інтернету речей у туманних мережах;

– розробка і моделювання архітектури мережі Fog Computing для оптимального розподілу обчислювальних навантажень;

– інтеграція механізмів забезпечення інформаційної безпеки в рамках запропонованої архітектури;

– порівняльний аналіз показників продуктивності запропонованого рішення з існуючими підходами.

Об’єкт дослідження: архітектурна структура, властива мережі туманних обчислень.

Предмет дослідження: методах та алгоритмах, що використовуються для моделювання архітектур туманних обчислень, охоплюючи стратегії оптимізації розподілу обчислювальних ресурсів.

Практичне значення одержаних результатів. Розроблена архітектура, що спеціально спрямована на обробку потоків даних, які надходять з систем Інтернету речей (IoT). Ця архітектура детально охоплює: пристрої IoT, які функціонують як основні джерела збору даних; проміжні вузли туманних обчислень, що відповідають за локалізовану обробку даних; і, нарешті, хмарну інфраструктуру, призначену для централізованих обчислювальних операцій. Результати моделювання надають практичні рекомендації щодо ефективного розподілу ресурсів у мережах Fog, а також вибору відповідних архітектурних рішень, адаптованих до вимог конкретних застосувань.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ОБРОБКИ IOT-ДАНИХ У FOG COMPUTING

1.1 Основні проблеми обробки даних IoT

1.1.1 Масштабованість IoT-систем

Проблема масштабованості IoT-систем є однією з найбільш фундаментальних викликів сучасної цифрової інфраструктури [1]. Експоненційне зростання кількості підключених пристроїв створює складні технічні та архітектурні проблеми, які потребують інноваційних підходів до їх вирішення (див. рисунок 1.1).

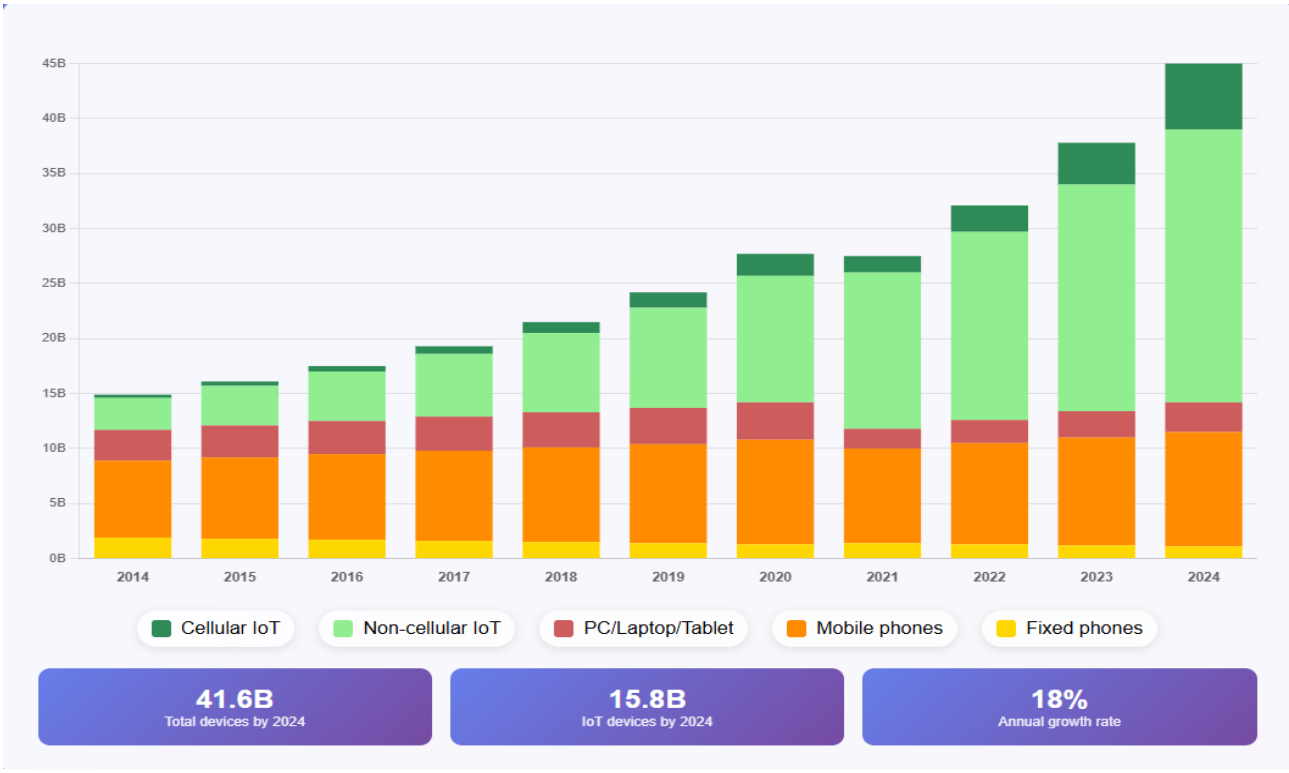


Рисунок 1.1 – Проблеми масштабованості в IoT-мережах

Сучасні екосистеми Інтернету речей характеризуються надзвичайною різноманітністю пристроїв, від простих датчиків температури та вологості до складних промислових роботів та автономних транспортних засобів.

Кожен тип пристрою має унікальні характеристики щодо споживання енергії, обчислювальних можливостей, типів даних, які він генерує, та частоти їх передачі. Ця неоднорідність створює значні труднощі для створення уніфікованих систем обробки даних [2].

Горизонтальна масштабованість стає критично важливою, коли система повинна одночасно обслуговувати мільйони або навіть мільярди пристроїв. Традиційні централізовані архітектури швидко досягають меж своєї продуктивності через обмеження пропускної здатності мережі, обчислювальних ресурсів та можливостей зберігання даних. Водночас вертикальна масштабованість також обмежена фізичними характеристиками обладнання та економічними факторами.

Одним з ключових аспектів проблеми масштабованості є динамізм мережі Інтернету речей. Пристрої можуть підключатися та відключатися від мережі в будь-який час, змінювати своє розташування, переходити в режим енергозбереження або виходити з ладу. Система повинна мати можливість автоматично адаптуватися до таких змін, забезпечуючи безперервність обслуговування та оптимальне використання ресурсів.

Географічний розподіл пристроїв Інтернету речей також створює специфічні проблеми. Пристрої можуть бути розташовані на різних континентах, у віддалених регіонах з обмеженим доступом до мережевої інфраструктури або на мобільних платформах. Це вимагає створення розподіленої обчислювальної архітектури, яка може забезпечити ефективну роботу незалежно від географічного розташування пристроїв.

Обмеження ресурсів пристроїв Інтернету речей ще більше ускладнюють питання масштабованості. Більшість пристроїв Інтернету речей мають обмежені обчислювальні ресурси, пам'ять та джерело живлення. Це означає, що складні алгоритми обробки даних не можуть виконуватися безпосередньо на пристроях, що створює додаткове навантаження на мережеву інфраструктуру та центральні системи обробки даних.

1.1.2 Централізоване оброблення даних у хмарі

Традиційна модель централізованої обробки даних у хмарі, яка довгий час була стандартом для корпоративних додатків, виявляється неадекватною для сучасних IoT-систем через ряд фундаментальних обмежень та недоліків [3].

Затримки передачі даних становлять одну з найбільш критичних проблем централізованої архітектури. Коли IoT-пристрої розташовані географічно далеко від центрів обробки даних, час передачі даних може становити сотні мілісекунд або навіть секунди. Для додатків реального часу, таких як системи безпеки, медичний моніторинг або промислове керування, такі затримки можуть бути неприйнятними та навіть небезпечними.

Пропускна здатність мережі стає вузьким місцем при централізованій обробці даних від великої кількості IoT-пристроїв. Навіть якщо окремий пристрій генерує відносно невеликі обсяги даних, агрегована інформація від тисяч або мільйонів пристроїв може перевищувати можливості мережевої інфраструктури. Це призводить до перевантажень, втрати пакетів та деградації якості обслуговування [4].

Економічні витрати на передачу даних до хмари можуть стати значними, особливо для додатків, які генерують великі обсяги даних або потребують постійної передачі інформації. Плата за трафік, особливо для мобільних мереж або супутникового зв'язку, може зробити централізовану архітектуру економічно недоцільною для масштабних IoT-проєктів.

Залежність від постійного мережевого з'єднання є ще одним критичним недоліком централізованої моделі. У випадку збою мережевого з'єднання, IoT-система може повністю втратити функціональність, оскільки пристрої не можуть обробляти дані локально. Це особливо проблематично для критично важливих додатків, де навіть короткочасні перерви у роботі можуть мати серйозні наслідки.

Проблеми конфіденційності та безпеки також загострюються при централізованій обробці. Передача всіх даних до віддалених центрів обробки збільшує поверхню атаки та ризики перехоплення конфіденційної інформації.

Крім того, централізоване зберігання великих обсягів персональних даних створює привабливі цілі для кібератак та може суперечити вимогам локальних законів про захист даних [5].

Масштабованість централізованих систем також має фізичні обмеження. Розширення потужності центрів обробки даних вимагає значних капітальних інвестицій та часу, що може не відповідати динамічним потребам зростаючих IoT-мереж. Крім того, централізована архітектура створює єдину точку відмови, де збій у центральному центрі обробки даних може паралізувати всю IoT-систему.

1.2 Fog Computing як рішення для IoT-систем

Fog Computing базується на восьми фундаментальних принципах, які визначають її архітектуру та функціональність (див. рисунок 1.2). Розуміння цих принципів є критично важливим для проєктування ефективних систем та правильної реалізації Fog-рішень [6].

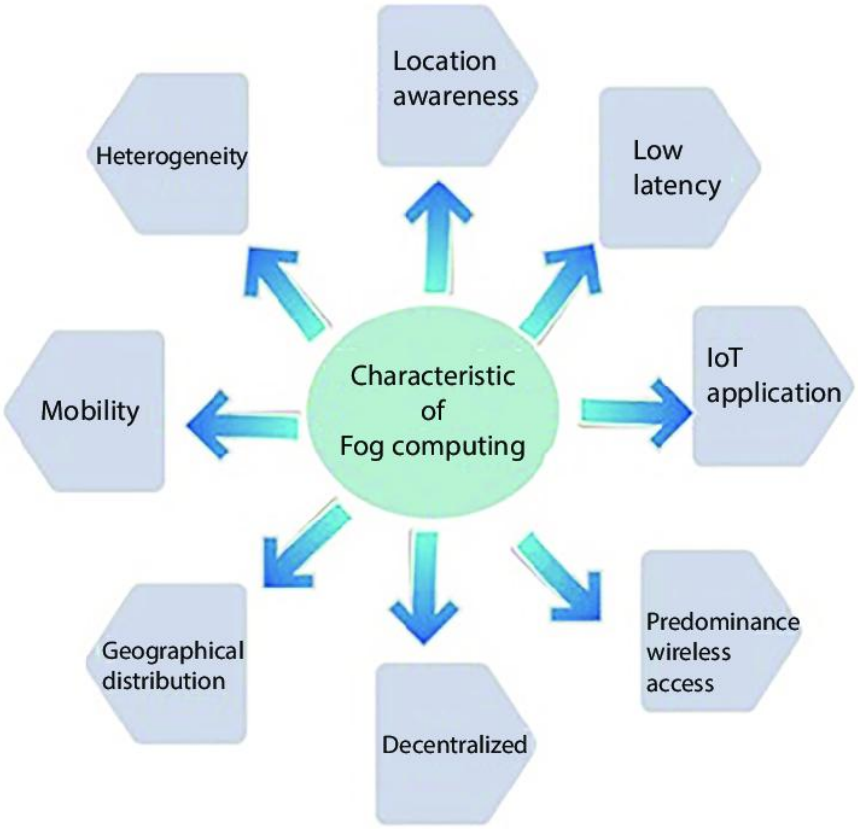


Рисунок 1.2 – Принципи Fog Computing

Принцип низької затримки та локальності є основоположним для Fog Computing. Розміщення обчислювальних ресурсів близько до кінцевих пристроїв дозволяє значно зменшити час, необхідний для передачі даних та отримання результатів обробки. Це особливо важливо для додатків, які потребують негайної реакції на зміни у навколишньому середовищі, таких як автономні транспортні засоби, промислові системи безпеки або медичні пристрої моніторингу.

Географічна розподіленість є ключовим принципом, який відрізняє Fog Computing від традиційних централізованих систем. Fog-вузли можуть бути розгорнуті у різних географічних локаціях, від локальних мереж підприємств до міських інфраструктур та віддалених регіонів. Така розподіленість забезпечує кращу доступність сервісів та можливість обслуговування користувачів незалежно від їх місцезнаходження.

Принцип великої кількості вузлів передбачає можливість підтримки мільйонів або навіть мільярдів Fog-вузлів у одній мережі. На відміну від хмарних центрів обробки даних, які зазвичай налічують тисячі серверів, Fog Computing може включати величезну кількість менших вузлів, розподілених по всій мережі. Це створює нові виклики для управління та координації, але також забезпечує безпрецедентну масштабованість.

Підтримка мобільності є критично важливим принципом для сучасних IoT-систем. Fog Computing повинна забезпечувати безперервність обслуговування для мобільних пристроїв, які можуть переміщуватися між різними Fog-вузлами. Це вимагає складних алгоритмів міграції сервісів та управління станом додатків.

Принцип реального часу вимагає, щоб Fog Computing системи могли обробляти потоки даних та генерувати відповіді з мінімальними затримками. Це включає не тільки швидку обробку окремих запитів, а й здатність обробляти безперервні потоки даних від множини джерел одночасно.

Взаємодія з різнорідними мережами є ще одним важливим принципом. Fog Computing повинна підтримувати різні типи мережевих з'єднань, від високошвидкісних оптоволоконних ліній до бездротових мереж з обмеженою пропускною здатністю. Це вимагає адаптивних протоколів та алгоритмів, які

можуть оптимізувати продуктивність залежно від доступних мережевих ресурсів.

Принцип федеративності та ієрархії дозволяє організувати Fog Computing систему як федерацію автономних доменів, кожен з яких може мати свою власну структуру управління та політики. Водночас, ієрархічна організація забезпечує можливість ескалації завдань до вищих рівнів системи, коли це необхідно.

Принцип програмованості передбачає, що Fog Computing платформи повинні надавати гнучкі API та інструменти для розробки та розгортання додатків. Це включає підтримку різних мов програмування, фреймворків та моделей виконання, що дозволяє розробникам створювати різноманітні типи додатків.

1.3 Сучасні платформи й інструменти для реалізації Fog Computing

Екосистема інструментів та платформ для реалізації Fog Computing швидко розвивається, пропонуючи розробникам широкий вибір рішень для створення розподілених IoT-систем. Розуміння можливостей та обмежень різних платформ є критично важливим для вибору оптимального рішення для конкретних проєктів [6].

1.3.1 Cisco IOx Platform

Cisco IOx Platform є однією з перших та найбільш зрілих платформ для Fog Computing. Вона надає інфраструктуру для розгортання та управління додатками на мережевому обладнанні Cisco, включаючи маршрутизатори, комутатори та промислові пристрої. IOx підтримує контейнеризовані додатки та віртуальні машини, забезпечуючи гнучкість розгортання різних типів робочих навантажень.

Платформа IOx включає Fog Director — централізований інструмент управління, який дозволяє моніторити, налаштовувати та оновлювати розподілені Fog-додатки. Система також надає інструменти для розробки,

тестування та розгортання додатків з підтримкою різних мов програмування та фреймворків (див. рисунок 1.3).

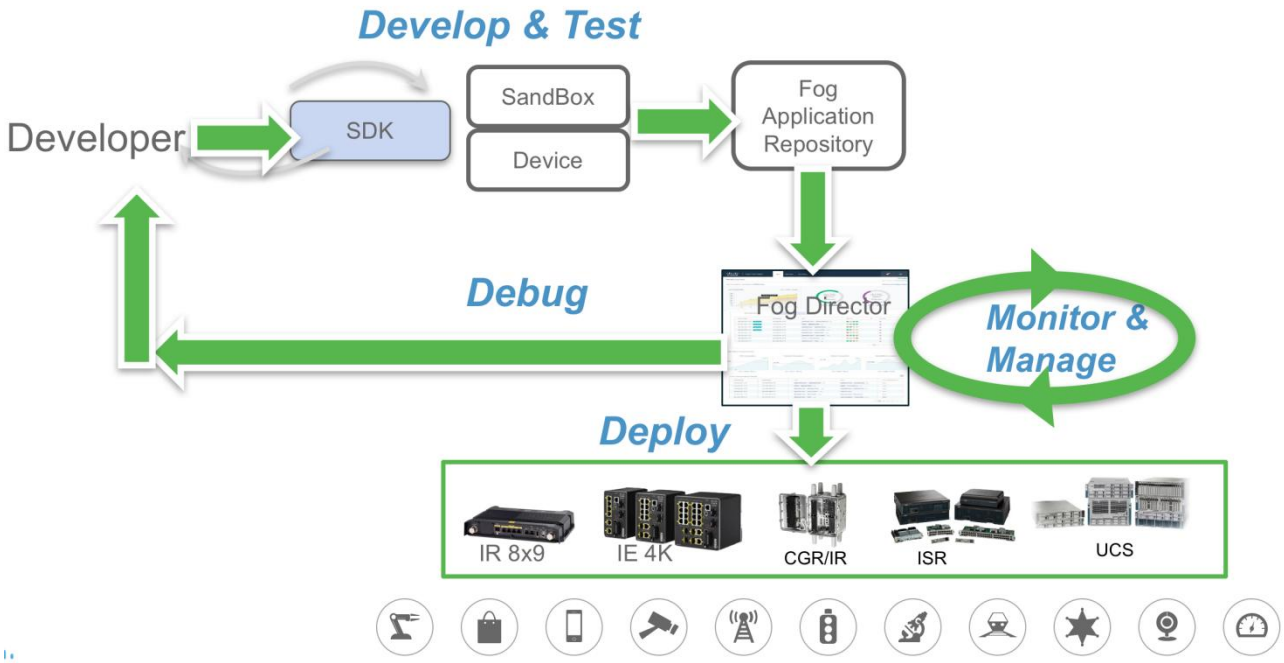


Рисунок 1.3 – Повний цикл розробки, тестування, розгортання, моніторингу та керування fog-додатками в Cisco IOx Platform

1.3.2 AWS IoT Greengrass

AWS IoT Greengrass є хмарною платформою Amazon, яка розширює можливості AWS у сферу периферійних обчислень. Greengrass дозволяє виконувати AWS Lambda функції локально на периферійних пристроях, забезпечуючи обробку даних у реальному часі без постійного підключення до хмари. Платформа підтримує машинне навчання на периферії через інтеграцію з Amazon SageMaker та надає можливості для локального управління пристроями.

1.3.3 Microsoft Azure IoT Edge

Microsoft Azure IoT Edge надає аналогічні можливості у екосистемі Microsoft Azure. Платформа дозволяє розгортати хмарні сервіси Azure на периферійних пристроях, включаючи Azure Functions, Azure Stream Analytics та

кастомні модулі. Azure IoT Edge підтримує контейнери Docker та забезпечує безпечну комунікацію з хмарою через сертифікати та ключі.

1.3.4 Google Cloud IoT Edge

Google Cloud IoT Edge (тепер частина Google Distributed Cloud) пропонує рішення для розгортання Google Cloud сервісів на периферії. Платформа інтегрується з Kubernetes та підтримує TensorFlow Lite для машинного навчання на периферійних пристроях.

1.3.5 Apache EdgeX Foundry

Apache EdgeX Foundry є відкритою платформою для побудови периферійних IoT-рішень. Вона надає модульну архітектуру з мікросервісами, які можуть бути легко налаштовані для різних застосувань. EdgeX включає сервіси для збору даних з пристроїв, локальної обробки, управління пристроями та інтеграції з хмарними платформами.

1.3.6 Eclipse ioFog

Eclipse ioFog є ще однією відкритою платформою, яка надає інфраструктуру для розгортання та управління додатками на периферії. Платформа підтримує мікросервісну архітектуру та забезпечує можливості для створення розподілених додатків, які можуть працювати на різних типах пристроїв.

1.3.7 FogHorn Lightning Platform

FogHorn Lightning Platform спеціалізується на промислових застосуваннях та надає інструменти для обробки потоків даних у реальному часі, машинного навчання на периферії та інтеграції з промисловими протоколами та системами (див. рисунок 1.4).

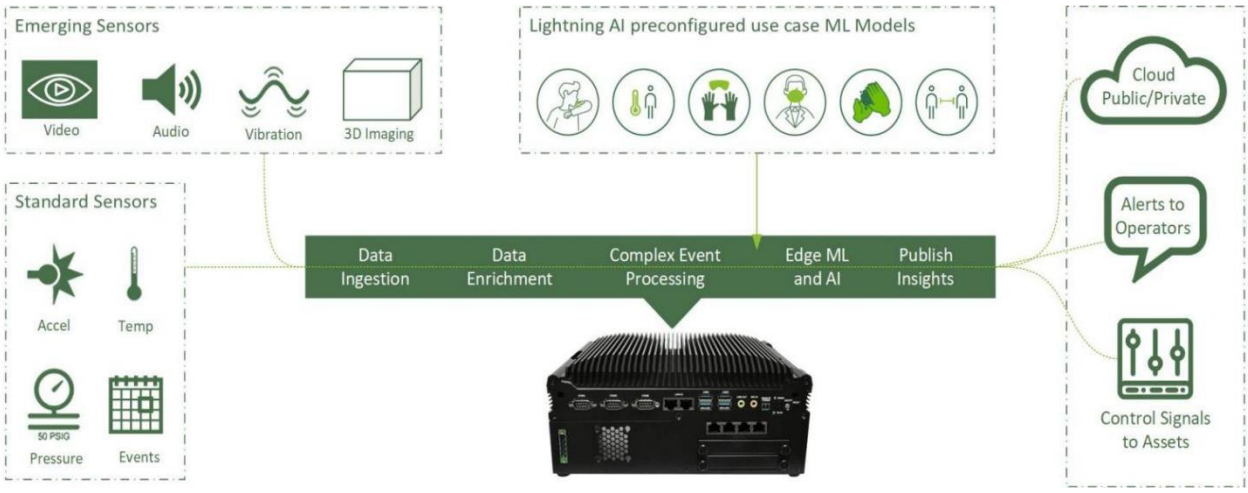


Рисунок 1.4 – Архітектура рішення для обробки даних з промислових сенсорів за допомогою штучного інтелекту та машинного навчання безпосередньо на периферії

1.4 Методи захисту даних у Fog-мережах

Безпека даних у Fog Computing представляє особливі виклики через розподілену природу архітектури та необхідність забезпечення захисту на всіх рівнях системи. Ефективні методи захисту повинні враховувати специфічні характеристики Fog-мереж та забезпечувати комплексний підхід до безпеки [7].

Шифрування даних є фундаментальним методом захисту у Fog-мережах. Наскрізне шифрування забезпечує захист даних від моменту їх генерації на IoT-пристроях до остаточної обробки у хмарі. Симетричне шифрування використовується для швидкої обробки великих обсягів даних, тоді як асиметричне шифрування застосовується для безпечного обміну ключами та автентифікації.

Гомоморфне шифрування є перспективною технологією, яка дозволяє виконувати обчислення над зашифрованими даними без їх розшифрування (див. рисунок 1.5). Це особливо важливо для Fog Computing, де дані можуть оброблятися на вузлах, які не знаходяться під повним контролем власника даних. Хоча гомоморфне шифрування все ще має значні накладні витрати на продуктивність, активні дослідження спрямовані на його оптимізацію.

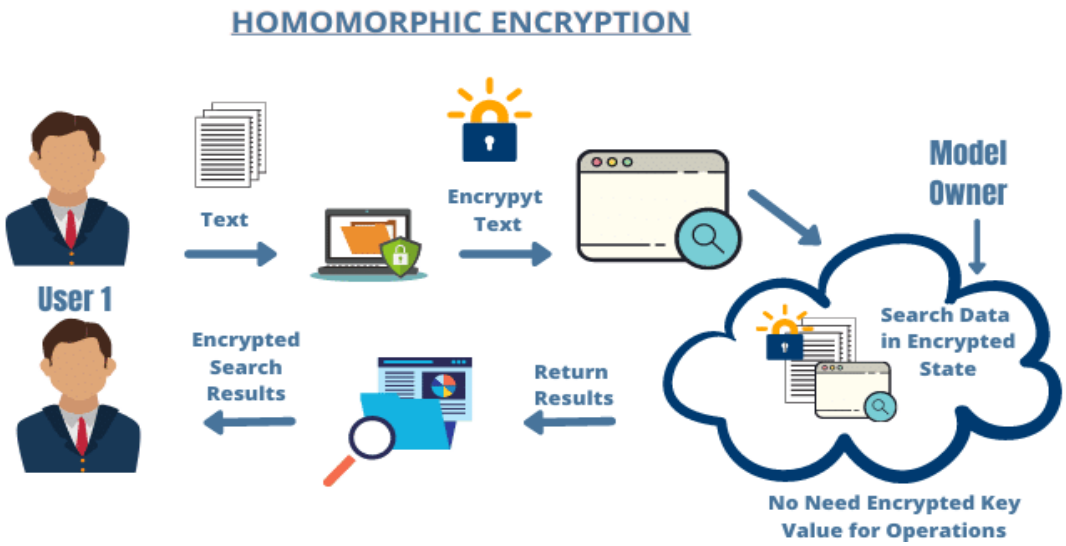


Рисунок 1.5 – Принцип роботи гомоморфного шифрування

Управління ключами у розподіленій архітектурі Fog Computing потребує спеціалізованих підходів. Ієрархічні схеми управління ключами дозволяють створювати структуровані системи, де ключі генеруються та розподіляються через довірені центри. Розподілені схеми управління ключами використовують криптографічні протоколи для створення та обміну ключами без централізованого управління [8].

Автентифікація та авторизація у Fog-мережах повинні враховувати динамічну природу системи та можливість приєднання нових пристроїв у будь-який момент. Багатофакторна автентифікація поєднує різні методи перевірки ідентичності, включаючи паролі, цифрові сертифікати, біометричні дані та фізичні токени.

Розподілена автентифікація використовує блокчейн або інші розподілені реєстри для створення децентралізованих систем управління ідентичністю. Це дозволяє уникнути єдиної точки відмови та забезпечити автентифікацію навіть при обмеженій мережевій зв'язності.

Контроль доступу базований на атрибутах (ABAC) є гнучким методом авторизації, який дозволяє визначати політики доступу на основі множини факторів, включаючи ідентичність користувача, характеристики ресурсу, час доступу та контекстуальну інформацію (див. рисунок 1.6).

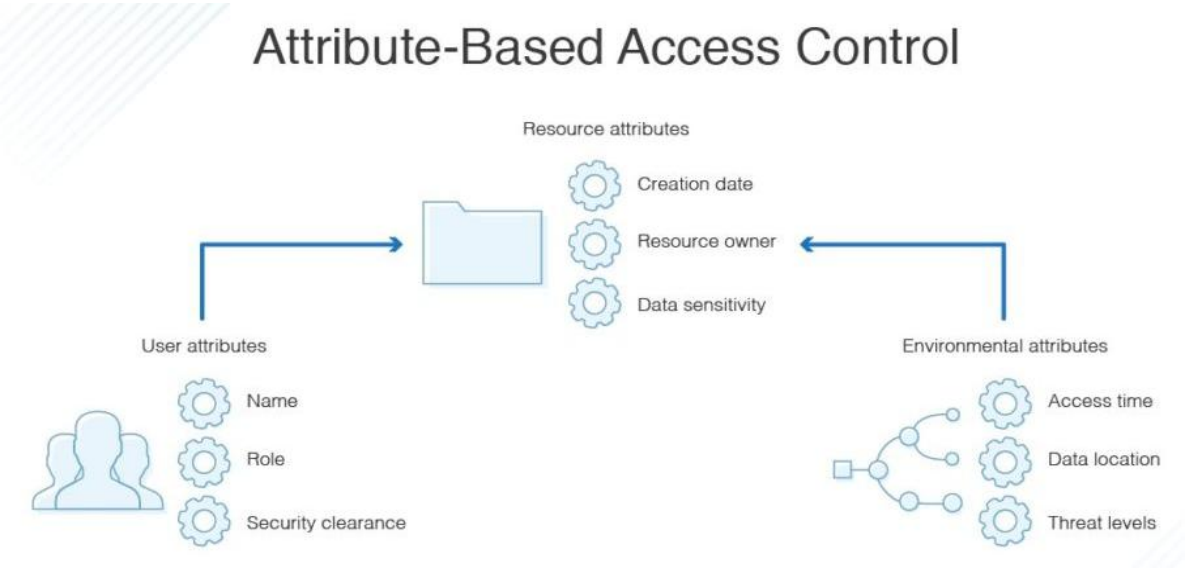


Рисунок 1.6 – Принцип роботи контролю доступу базований на атрибутах (ABAC)

Захист цілісності даних забезпечує виявлення несанкціонованих змін у даних під час їх передачі та зберігання. Криптографічні хеш-функції та цифрові підписи використовуються для створення відбитків даних, які можуть бути перевірені для підтвердження цілісності.

Blockchain технології можуть використовуватися для створення незмінних журналів подій та транзакцій у Fog-мережах. Це особливо корисно для аудиту та забезпечення відповідальності у системах з множинними учасниками [9].

Методи забезпечення приватності включають диференційну приватність, яка додає контрольований шум до даних для захисту індивідуальної інформації при збереженні статистичної корисності. Анонімізація та псевдонімізація даних також використовуються для захисту ідентичності користувачів.

Secure Multi-party Computation (SMPC) дозволяє множині сторін спільно обчислювати функції над їх приватними входами без розкриття цих входів. Це може бути використано у Fog Computing для виконання аналітики над чутливими даними без компрометації приватності.

Мережева безпека включає захист від різних типів атак, включаючи DDoS атаки, атаки людини посередині та мережеве сканування. Системи виявлення та запобігання вторгненням (IDS/IPS) можуть бути розгорнуті на Fog-вузлах для моніторингу мережевого трафіку та виявлення підозрілої активності.

Сегментація мережі дозволяє ізолювати різні частини Fog-мережі для обмеження поширення потенційних атак. Віртуальні приватні мережі (VPN) та програмно-визначені мережі (SDN) можуть використовуватися для створення безпечних каналів комунікації.

Фізична безпека також є важливим аспектом, оскільки Fog-вузли часто розташовуються у фізично доступних місцях. Trusted Platform Modules (TPM) та інші апаратні модулі безпеки можуть забезпечити захист криптографічних ключів та забезпечити довірену завантаження системи.

РОЗДІЛ 2 МОДЕЛЬ АРХІТЕКТУРИ FOG COMPUTING ДЛЯ ОБРОБКИ IOT-ДАНИХ

2.1 Архітектура Fog-мереж: рівні обробки даних, вузли Fog, роль IoT-пристроїв

Архітектура Fog Computing представляє собою складну багаторівневу систему, яка забезпечує ефективну обробку даних на різних рівнях близькості до їх джерел (див. рисунок 2.1). Розуміння цієї архітектури є фундаментальним для проєктування та реалізації ефективних IoT-систем. Архітектура Fog Computing складається з трьох фундаментальних рівнів, кожен з яких виконує специфічні функції та має унікальні характеристики [10].

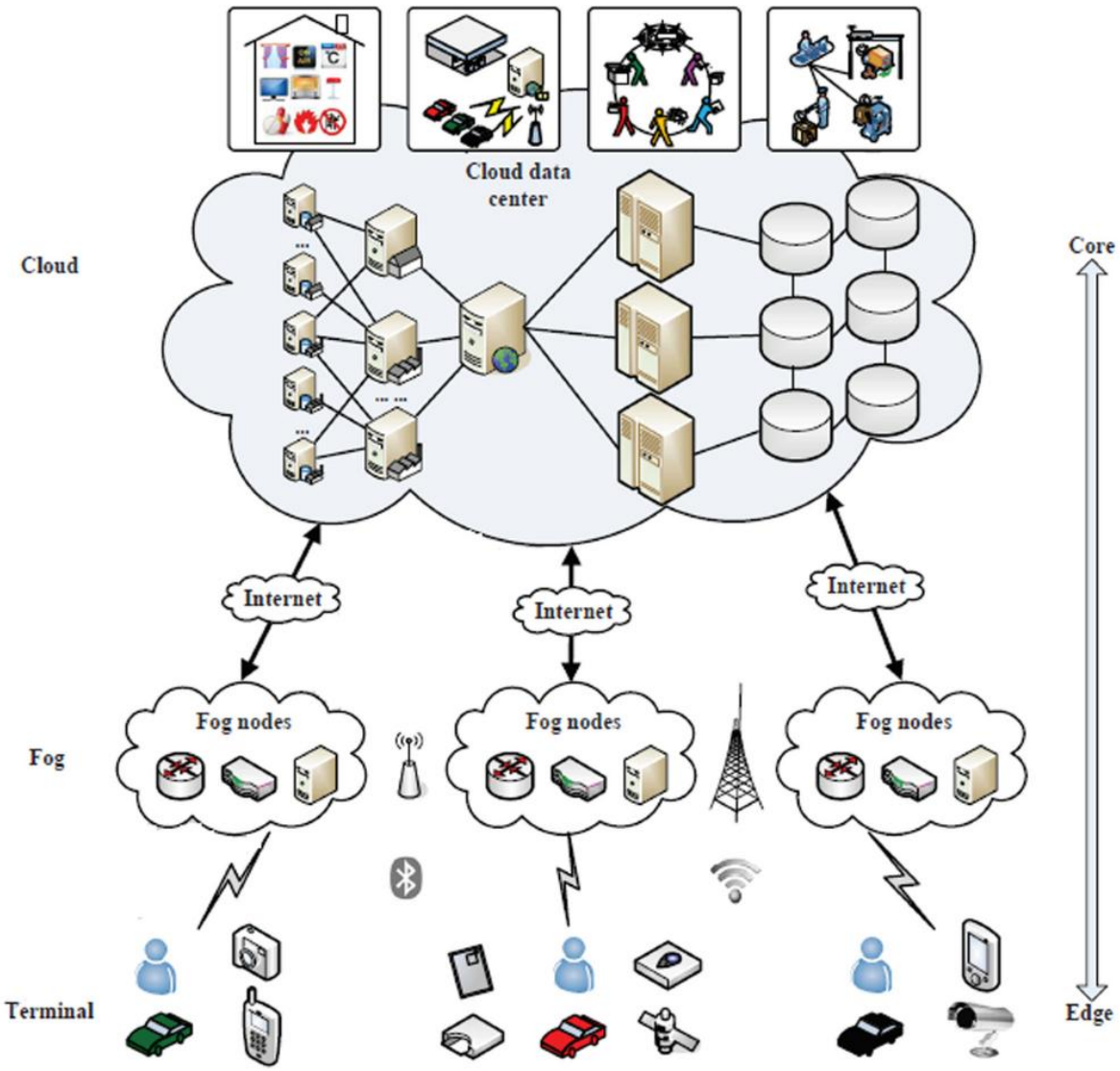


Рисунок 2.1 – Архітектура Fog Computing

Рівень пристроїв (Device Layer) становить найнижчий рівень архітектури та включає всі IoT-пристрої, сенсори, актуатори та інші кінцеві пристрої, які безперервно збирають інформацію про навколишнє середовище, включаючи температуру, вологість, атмосферний тиск, рівень освітлення, якість повітря або стан систем безпеки. Цей рівень характеризується великою різноманітністю пристроїв з різними можливостями обробки, зберігання та комунікації. Сучасні IoT-пристрої можуть варіюватися від простих сенсорів з мінімальними обчислювальними можливостями до складних пристроїв з потужними процесорами та значними обсягами пам'яті. На рівні пристроїв виконуються базові функції збору даних, первинної фільтрації та локальної обробки. Сучасні мікроконтролери та системи на кристалі дозволяють реалізувати складні алгоритми обробки сигналів, початкову аналітику та навіть прості моделі машинного навчання безпосередньо на пристроях. Це дозволяє зменшити обсяг даних, які потрібно передавати до вищих рівнів архітектури.

Fog-рівень (Fog Layer) є серцем архітектури та складається з розподілених вузлів, які надають обчислювальні, мережеві та сервісні можливості близько до кінцевих пристроїв. Fog-вузли можуть бути реалізовані у різних формах: спеціалізовані Fog-сервери, розумні маршрутизатори та шлюзи, базові станції мобільного зв'язку, промислові комп'ютери або навіть потужні смартфони та планшети. Кожен Fog-вузол має власні обчислювальні ресурси, включаючи процесори, пам'ять, сховище даних та мережеві інтерфейси. Fog-вузли виконують широкий спектр функцій: агрегацію та попередню обробку даних від IoT-пристроїв, виконання аналітичних алгоритмів та алгоритмів машинного навчання, кешування часто використовуваних даних, забезпечення безпеки та шифрування, маршрутизацію та балансування навантаження. Це створює концепцію "розумних пристроїв", які можуть приймати локальні рішення та взаємодіяти один з одним для вирішення складних завдань [11].

Проміжний рівень (Intermediate Layer) може існувати між Fog-рівнем та хмарним рівнем у великих системах. Цей рівень включає регіональні центри обробки даних, які надають більші обчислювальні ресурси та можливості

зберігання порівняно з локальними Fog-вузлами, але все ще знаходяться ближче до кінцевих користувачів, ніж централізовані хмарні центри.

Хмарний рівень (Cloud Layer) представляє традиційні центри обробки даних з великими обчислювальними потужностями та практично необмеженими можливостями зберігання. Цей рівень виконує складні аналітичні завдання, які потребують значних ресурсів, довготермінове зберігання даних, глобальну координацію та управління всією системою. Публічні хмарні платформи, такі як Amazon Web Services, Microsoft Azure та Google Cloud Platform, надають масштабовані обчислювальні ресурси та широкий спектр сервісів для обробки IoT-даних. Приватні хмари, розгорнуті в корпоративних дата-центрах, забезпечують повний контроль над даними та відповідність специфічним вимогам безпеки та регулятивним нормам.

Взаємодія між рівнями здійснюється через стандартизовані протоколи та API, які забезпечують прозорість для додатків та дозволяють динамічно розподіляти навантаження між різними рівнями залежно від поточних умов та вимог.

Спеціалізовані IoT-шлюзи виконують функції агрегації та первинної обробки даних від множини IoT-пристроїв. Ці пристрої забезпечують протокольне перетворення, дозволяючи різнорідним IoT-пристроям взаємодіяти в єдиній мережі. Мережеве обладнання, таке як роутери та комутатори нового покоління, також може виконувати функції Fog-вузлів завдяки вбудованим обчислювальним можливостям та підтримці технологій віртуалізації.

Базові станції мобільного зв'язку, особливо в контексті 5G мереж, представляють ще один тип Fog-вузлів. Ці станції мають значні обчислювальні ресурси та можуть забезпечувати обробку даних для мобільних IoT-пристроїв з мінімальною затримкою. Використання технологій Network Function Virtualization (NFV) та Software-Defined Networking (SDN) дозволяє динамічно розгортати обчислювальні функції на базових станціях відповідно до поточних потреб [10].

2.2 Аналіз методів обробки та зберігання даних

Обробка даних у Fog Computing базується на принципі розподіленої обробки, коли дані оброблюються на найближчому до джерела їх генерації вузлі, що має достатні ресурси для виконання необхідних операцій. Цей підхід кардинально відрізняється від традиційної хмарної моделі, де всі дані передаються до централізованих дата-центрів для обробки. Розподілена обробка дозволяє значно зменшити затримки з типових 100-200 мілісекунд при зверненні до хмари до 1-10 мілісекунд при локальній обробці на Fog-вузлах [12].

Аналітика на краю мережі (Edge Analytics) включає комплекс методів для обробки даних безпосередньо на Fog-вузлах. Фільтрація даних є першим етапом обробки, коли відбувається видалення неактуальної, дублювальної або некоректної інформації. Цей процес дозволяє значно зменшити обсяг даних, які потребують передачі до вищих рівнів архітектури. Агрегація даних передбачає об'єднання інформації від кількох джерел для створення узагальнених метрик та показників.

Попередня обробка даних на Fog-вузлах включає нормалізацію, очищення та трансформацію даних у формати, придатні для подальшого аналізу. Локальне прийняття рішень є ключовою особливістю Fog Computing, коли Fog-вузли можуть автономно приймати рішення та ініціювати дії без необхідності звернення до хмарної інфраструктури. Це особливо важливо для критичних додатків, таких як системи безпеки або медичного моніторингу.

Ієрархічна модель обробки даних передбачає, що IoT-пристрої виконують базову фільтрацію та попередню обробку даних, Fog-вузли здійснюють складну аналітику та застосовують алгоритми машинного навчання, а хмарна інфраструктура забезпечує глибокий аналіз та обробку великих даних. Такий розподіл функцій дозволяє оптимально використовувати ресурси кожного рівня та забезпечувати високу ефективність системи (див. рисунок 2.2).

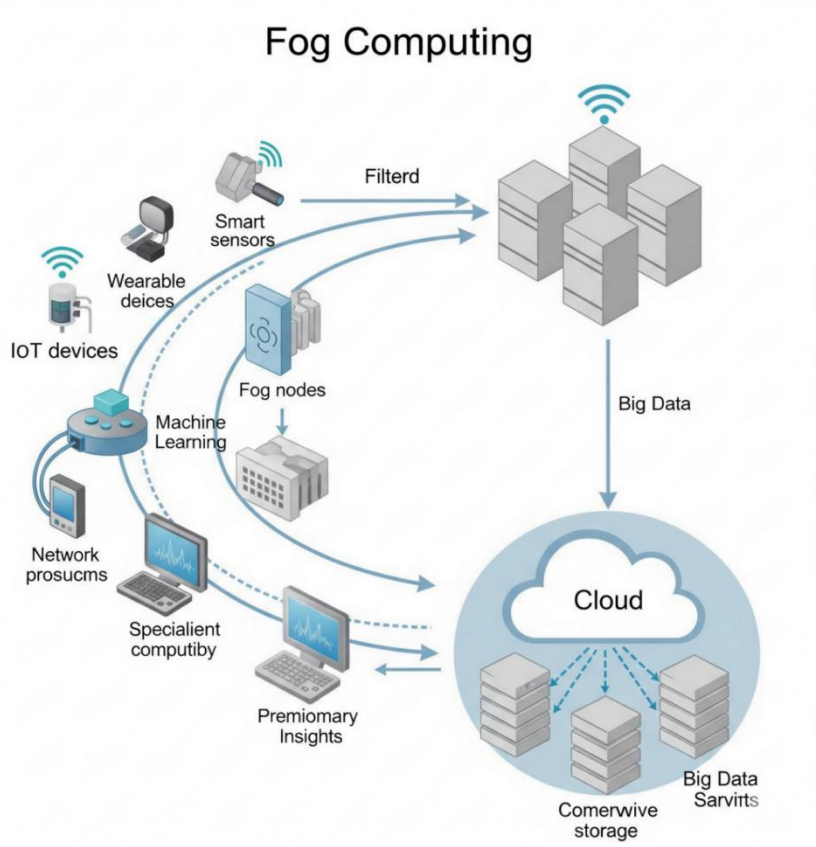


Рисунок 2.2 – Схема ієрархічної обробки

Зберігання даних у Fog Computing реалізується через багаторівневу стратегію, яка враховує специфіку різних типів даних та вимоги до їх доступності. Тимчасове зберігання на Fog-вузлах використовується для кешування часто використовуваних даних, буферизації даних для пакетної передачі та створення локальних баз даних для швидкого доступу. Використання легких систем управління базами даних, таких як SQLite або Redis, дозволяє забезпечити високу швидкість доступу до даних при мінімальних ресурсних витратах [13].

Інтелектуальне управління даними передбачає розподіл даних між різними рівнями архітектури залежно від їх важливості та частоти використання (див.таблицю 2.1). Критичні для безпеки дані зберігаються одночасно на Fog-вузлах та в хмарі для забезпечення надійності та доступності. Сенсорні дані зазвичай зберігаються на Fog-вузлах протягом 24-48 годин, після чого видаляються або передаються до хмари. Агреговані метрики та аналітичні звіти зберігаються в хмарній інфраструктурі протягом тривалого періоду для історичного аналізу та виявлення довгострокових трендів.

Таблиця 2.1 – Інтелектуальне управління даними

Тип даних	Розташування	Час зберігання
Критичні для безпеки	Fog + Cloud	Постійно
Сенсорні дані	Fog	24-48 годин
Агреговані метрики	Cloud	1-5 років
Логи системи	Fog + Cloud	30-90 днів

2.3 Ефективність обробки IoT-даних та оптимальне використання ресурсів

Оцінка ефективності Fog Computing здійснюється через аналіз ключових метрик продуктивності, які включають латентність, пропускну здатність, енергоефективність та використання ресурсів. Латентність є критичним параметром для багатьох IoT-додатків, особливо тих, які потребують реального часу реагування. Архітектура Fog Computing дозволяє досягти латентності менше 10 мілісекунд для зв'язку між IoT-пристроями та Fog-вузлами, що є значним покращенням порівняно з 100-500 мілісекундами при прямому зверненні до хмарної інфраструктури [14].

Пропускна здатність мережі використовується значно ефективніше завдяки локальній обробці даних на Fog-вузлах. Fog Computing може зменшити навантаження на магістральні мережеві канали на 60-80% завдяки фільтрації та агрегації даних на краю мережі. Це особливо важливо в умовах експоненційного зростання кількості IoT-пристроїв та обсягів генерованих ними даних.

Енергоефективність є критичним фактором для IoT-екосистем, оскільки багато пристроїв працюють від батарей. Fog Computing сприяє зниженню енергоспоживання через скорочення необхідності передачі великих обсягів даних до віддалених дата-центрів. Локальна обробка даних також дозволяє використовувати енергоефективні процесори архітектури ARM, які споживають значно менше енергії порівняно з традиційними серверними процесорами.

Оптимізація використання ресурсів Fog-вузлів здійснюється через динамічне балансування навантаження та інтелектуальне розподілення завдань.

Алгоритми балансування навантаження враховують поточне завантаження процесора, доступність оперативної пам'яті, мережеву латентність та енергоспоживання для визначення оптимального вузла для виконання конкретного завдання. Типове використання ресурсів Fog-вузлів становить 70-85% процесорного часу та 60-75% оперативної пам'яті, що свідчить про ефективне використання наявних ресурсів [15].

Кешування даних на Fog-вузлах реалізується через різні стратегії, включаючи алгоритм LRU (Least Recently Used), який автоматично видаляє найдавніше використовувані дані при нестачі місця для зберігання. Механізм TTL (Time To Live) забезпечує автоматичне видалення застарілих даних через визначений проміжок часу. Інтелектуальне передбачення на основі алгоритмів машинного навчання дозволяє прогнозувати потребу в конкретних даних та заздалегідь завантажувати їх до кешу.

2.4 Використання шифрування даних під час передачі

Шифрування даних під час передачі є фундаментальним елементом безпеки Fog Computing, який забезпечує конфіденційність та цілісність інформації при її пересиланні між різними компонентами системи. Багаторівневий підхід до шифрування враховує специфічні вимоги кожного рівня архітектури та обмеження ресурсів різних типів пристроїв. У контексті розумних домашніх систем безпеки, де реалізується трирівнева архітектура з розумними камерами, регіональними туманними вузлами та хмарною інфраструктурою, криптографічний захист повинен адаптуватися до гетерогенності обчислювальних ресурсів [16].

Архітектурна специфіка систем Fog Computing визначає необхідність диференційованого підходу до вибору криптографічних алгоритмів. Розумні камери з обмеженими обчислювальними ресурсами потребують легковагих криптографічних рішень, тоді як регіональні туманні вузли з більшими можливостями можуть використовувати більш складні алгоритми. Хмарна

інфраструктура з максимальними ресурсами здатна обробляти найскладніші криптографічні операції та виконувати роль центрального управління безпекою.

Симетричне шифрування використовується для захисту великих обсягів даних завдяки своїй високій швидкості та низькому енергоспоживанню. Алгоритм AES-256 є стандартом для шифрування відеопотоків у системах відеонагляду Fog Computing завдяки своїй надійності та ефективності при обробці великих обсягів мультимедійних даних. Відеопотоки від розумних камер, що характеризуються високою інтенсивністю передачі даних (800 байт базової інформації з 25000 інструкцій обробки), вимагають криптографічних рішень з мінімальними накладними витратами на шифрування.

Для пристроїв з особливо обмеженими ресурсами може використовуватися алгоритм ChaCha20, який забезпечує аналогічний рівень безпеки при менших вимогах до обчислювальних ресурсів. Цей алгоритм особливо ефективний для IoT-пристроїв, де енергоспоживання є критичним фактором, а швидкість обробки може бути обмеженою. Симетричне шифрування особливо ефективне для постійних з'єднань між туманними вузлами та для захисту потоків сенсорних даних від IoT-пристроїв, які характеризуються регулярністю передачі з детерміністичним розподілом часових інтервалів.

Оптимізація симетричного шифрування в контексті туманних обчислень включає використання апаратного прискорення криптографічних операцій, коли це можливо, та адаптацію розміру блоків шифрування відповідно до характеристик мережевого трафіку. Для відеопотоків високої роздільності може застосовуватися сегментоване шифрування, що дозволяє паралельну обробку та зменшує загальну латентність системи.

Асиметричне шифрування застосовується для безпечного обміну ключами та встановлення довірчих відносин між вузлами мережі в ієрархічній структурі туманних обчислень. Алгоритм RSA з довжиною ключа 2048 або 4096 біт забезпечує надійний захист для більшості застосувань на рівні регіональних туманних вузлів та хмарної інфраструктури, хоча може бути занадто ресурсоємним для найпростіших IoT-пристроїв з обмеженими обчислювальними можливостями.

Криптографія еліптичних кривих (ECC) пропонує альтернативу RSA з меншими вимогами до обчислювальних ресурсів при збереженні високого рівня безпеки, що робить її ідеальною для IoT-пристроїв та розумних камер з обмеженими ресурсами [17]. ECC-256 забезпечує рівень безпеки, еквівалентний RSA-3072, при значно менших вимогах до пам'яті та обчислювальної потужності, що критично важливо для пристроїв третього рівня ієрархії.

Процес встановлення довіри в багаторівневій архітектурі реалізується через механізм взаємної автентифікації, де кожен вузол повинен довести свою ідентичність перед встановленням захищеного з'єднання. Це особливо важливо для систем безпеки, де компрометація одного вузла може призвести до порушення безпеки всієї системи. Асиметричне шифрування також використовується для цифрового підпису повідомлень, що забезпечує їх цілісність та неспростовність.

Протокол TLS (Transport Layer Security) версії 1.3 є основним стандартом для захисту даних під час передачі в системах Fog Computing. TLS 1.3 забезпечує взаємну автентифікацію клієнта та сервера, підтримує Perfect Forward Secrecy (PFS), який гарантує, що компрометація довгострокових ключів не призведе до розшифрування попередньо перехоплених повідомлень. Протокол також включає значні оптимізації для зниження латентності, включаючи можливість 0-RTT (Zero Round Trip Time) для повторних з'єднань, що критично важливо для систем реального часу.

Мережева архітектура туманних обчислень з різними рівнями латентності (100 мілісекунд між хмарою та проксі-сервером, 2 мілісекунди між туманними вузлами, 1 мілісекунда для IoT-пристроїв) вимагає адаптації TLS-параметрів для оптимізації продуктивності [18]. Використання попередньо встановлених сесійних ключів та кешування криптографічних параметрів дозволяє мінімізувати накладні витрати на встановлення з'єднань.

Для IoT-пристроїв, що використовують UDP-протокол, застосовується DTLS (Datagram Transport Layer Security), який адаптує механізми безпеки TLS для ненадійних транспортних протоколів. DTLS особливо важливий для IoT-пристроїв, які працюють в умовах нестабільних мережевих з'єднань та

потребують мінімальних накладних витрат на підтримку з'єднання. Протокол включає механізми виявлення та відновлення після втрати пакетів, що забезпечує надійність передачі критичних даних безпеки.

Управління ключами в розподіленій системі Fog Computing реалізується через ієрархічну модель довіри, де кореневий центр сертифікації (Root CA) розташований в хмарній інфраструктурі з максимальними ресурсами безпеки, проміжні центри сертифікації працюють на регіональному рівні туманних вузлів, а індивідуальні сертифікати видаються для кожного IoT-пристрою та туманного вузла. Така архітектура забезпечує масштабованість системи управління ключами та дозволяє ефективно управляти життєвим циклом сертифікатів для великої кількості пристроїв.

Розподіл функцій управління ключами відбувається відповідно до обчислювальних можливостей кожного рівня. Хмарна інфраструктура виконує складні операції генерації кореневих ключів та сертифікатів, туманні вузли здійснюють проміжне управління ключами для своїх підлеглих пристроїв, а IoT-пристрої зберігають та використовують тільки необхідні їм індивідуальні ключі.

Система сертифікатів використовує X.509 стандарт з розширеннями для специфічних потреб IoT та туманних обчислень. Короткі терміни дії сертифікатів для критичних компонентів системи безпеки забезпечують додатковий рівень захисту, а автоматизовані процеси поновлення сертифікатів мінімізують адміністративні накладні витрати.

Використання апаратних модулів безпеки (HSM) забезпечує надійне зберігання та управління криптографічними ключами, захищаючи їх від програмних атак та фізичного втручання. HSM особливо важливі для кореневих центрів сертифікації в хмарній інфраструктурі, де зберігаються найкритичніші ключі системи.

Управління життєвим циклом ключів включає не тільки їх генерацію та ротацію, але й безпечне знищення застарілих ключів, архівування для аудиту та відновлення в екстрених ситуаціях. Логування всіх операцій з ключами забезпечує можливість аудиту безпеки та відповідність регулятивним вимогам.

2.5 Виявлення та запобігання аномаліям

Системи виявлення аномалій є критично важливим компонентом безпеки Fog Computing, оскільки розподілена природа архітектури створює множину потенційних векторів атак та точок компрометації. Ефективна система виявлення аномалій повинна працювати в реальному часі, забезпечувати високу точність виявлення загроз при мінімальній кількості помилкових спрацьовувань.

Мережеві аномалії включають широкий спектр потенційних загроз, від розподілених атак типу "відмова в обслуговуванні" (DDoS) на Fog-вузли до складних атак типу "людина посередині" (Man-in-the-Middle), які можуть перехоплювати та модифікувати дані під час передачі. Несанкціонований мережевий трафік може свідчити про спроби проникнення в систему або витік конфіденційних даних. Системи виявлення мережевих аномалій аналізують патерни трафіку, включаючи обсяги даних, частоту з'єднань, географічне розташування джерел трафіку та протокольні характеристики [19].

Аномалії пристроїв можуть виникати внаслідок програмної компрометації IoT-пристроїв, фізичних несправностей сенсорів або навмисного фізичного втручання в роботу пристроїв. Скомпрометовані IoT-пристрої часто демонструють аномальні патерни поведінки, такі як неочікувана активність у мережі, передача даних до невідомих адресатів або генерація даних, які не відповідають очікуваним характеристикам. Несправні сенсори можуть генерувати викривлені або недостовірні дані, що може призвести до неправильних рішень на вищих рівнях системи.

Аномалії даних включають підроблені сенсорні дані, аномальні шаблони використання системи та потенційний витік конфіденційних даних. Підроблені дані можуть бути результатом цілеспрямованих атак на цілісність системи або наслідком технічних несправностей. Аномальні шаблони використання можуть свідчити про несанкціонований доступ до системи або зловживання ресурсами.

Статистичні методи виявлення аномалій базуються на аналізі розподілу даних та ідентифікації значень, які значно відхиляються від очікуваних параметрів. Z-score аналіз дозволяє виявляти викиди в числових даних шляхом

обчислення стандартного відхилення від середнього значення. Алгоритм Isolation Forest ефективно виявляє аномалії в багатовимірних даних шляхом ізоляції аномальних точок в меншій кількості кроків порівняно з нормальними даними. ARIMA моделі використовуються для аналізу часових рядів та виявлення аномальних трендів або сезонних відхилень.

Методи машинного навчання забезпечують більш складні та адаптивні підходи до виявлення аномалій. One-Class Support Vector Machine (SVM) навчається на даних, що представляють нормальну поведінку системи, та ідентифікує нові дані, які не відповідають встановленим паттернам. Автоенкодери, тип нейронних мереж, навчаються відтворювати вхідні дані та виявляють аномалії на основі помилок реконструкції [20]. Long Short-Term Memory (LSTM) мережі особливо ефективні для аналізу послідовностей даних та виявлення аномалій у часових рядах.

Поведінковий аналіз фокусується на встановленні профілів нормальної поведінки для кожного типу пристроїв та вузлів у мережі. Ці профілі включають типові патерни мережевої активності, частоту та обсяги передачі даних, часові рамки активності та взаємодії з іншими компонентами системи. Будь-які значні відхилення від встановлених профілів можуть свідчити про потенційні проблеми безпеки або технічні несправності.

РОЗДІЛ 3 МОДЕЛЮВАННЯ ЗАПРОПОНОВАНОЇ АРХІТЕКТУРИ

3.1 Налаштування симуляційного середовища

Для реалізації моделювання архітектури мережі Fog Computing було обрано симуляційну платформу iFogSim, яка є спеціалізованим розширенням відомої CloudSim платформи. Вибір iFogSim обумовлений її унікальними можливостями для моделювання багаторівневих fog-computing архітектур, що критично важливо для дослідження ефективності обробки IoT-даних.

Основні переваги iFogSim включають підтримку географічно розподілених fog-вузлів, моделювання мережевих затримок між різними рівнями архітектури, енергетичні моделі для оцінки споживання електроенергії, а також гнучкі механізми розміщення додатків. Платформа дозволяє створювати складні сценарії з різними стратегіями розміщення обчислювальних модулів, що є ключовим для порівняння ефективності fog computing підходу з традиційними хмарними рішеннями.

У рамках дослідження була розроблена комплексна система моделювання розумного дому безпеки, яка демонструє типові застосування IoT в fog-computing середовищі. Система включає відеоспостереження, аналіз загроз і генерацію сповіщень в реальному часі. Розроблена архітектурна модель представляє чотирирівневу ієрархічну структуру, що відображає типову організацію fog-computing мереж. Кожен рівень має специфічні функції та технічні характеристики, оптимізовані для конкретних задач обробки IoT-даних.

Хмарний рівень (Cloud Layer) представляє найвищий рівень ієрархії з центральним сервером високої потужності. Цей рівень призначений для виконання ресурсоємних обчислювальних задач, довгострокового зберігання даних, машинного навчання та аналітики великих даних. Параметри хмарного сервера включають обчислювальну потужність 44,800 MIPS, що еквівалентно потужному серверному процесору, оперативну пам'ять 40,000 МБ для обробки великих обсягів даних, асиметричну пропускну здатність 100/10,000 Мбіт/с, що

відображає типову конфігурацію інтернет-з'єднання дата-центрів. В коді системи хмарний сервер створюється наступним чином (див. лістинг 3.1):

Лістинг 3.1 – Ініціалізація хмарного вузла в архітектурі Fog-обчислень

```
    FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100,
10000, 0, 0.01, 16*103, 16*83.25);
    cloud.setParentId(-1);
    fogDevices.add(cloud);
```

Проксі-рівень (Proxy Layer) функціонує як проміжний агрегаційний вузол між хмарою та локальними fog-вузлами. Основні функції включають маршрутизацію трафіку, кешування даних, балансування навантаження та забезпечення відмовостійкості. Критичним параметром є затримка з'єднання з хмарою в 100 мс, що відображає реальні мережеві умови при з'єднанні з віддаленими дата-центрами. Проксі-сервер налаштовується з помірними ресурсами (див. лістинг 3.2):

Лістинг 3.2 – Налаштування проксі-сервера з підключенням до хмари у Fog-інфраструктурі

```
    FogDevice proxy = createFogDevice("proxy-server", 2800, 4000,
10000, 10000, 1, 0.0, 107.339, 83.43);
    proxy.setParentId(cloud.getId());
    proxy.setUplinkLatency(100);
```

Проксі-сервер виконує функції агрегації та маршрутизації з балансом між продуктивністю та енергоефективністю:

– Обчислювальна потужність: 2,800 MIPS достатня для маршрутизації, кешування та попередньої обробки даних.

– Оперативна пам'ять: 4,000 МБ для буферизації трафіку та тимчасового зберігання.

– Пропускна здатність: симетрична 10,000/10,000 Мбіт/с для ефективного транзиту даних.

– Енергоспоживання: 107.339/83.43 Вт оптимізоване для безперервної роботи.

– Ієрархічний рівень: 1Fog-рівень.

Туманний рівень (Fog Layer) представляє розподілені обчислювальні вузли, розташовані близько до джерел IoT-даних. Ці вузли виконують основну частину обробки даних в реальному часі, забезпечуючи низькі затримки та зменшуючи навантаження на мережу. У системі моделюється чотири fog-вузли: (див. лістинг 3.3):

Лістинг 3.3 – Автоматизоване створення зон обслуговування для користувачів у межах Fog-мережі

```
static int numOfAreas = 4;
for(int i=0;i<numOfAreas;i++){
    addArea(i+"", userId, appId, proxy.getId());
}
```

Кожен fog-вузол конфігурується як маршрутизатор з обчислювальними можливостями (див. лістинг 3.4):

Лістинг 3.4 – Ініціалізація вузла-маршрутизатора в зоні 'a-id' для забезпечення локальної обробки запитів у Fog-мережі

```
    FogDevice router = createFogDevice("a-"+id, 2800, 4000, 1000,
10000, 2, 0.0, 107.339, 83.43);
    router.setUplinkLatency(2);
```

Fog-вузли представляють ключовий компонент архітектури, забезпечуючи локальну обробку IoT-даних:

– Обчислювальна потужність: 2,800 MIPS для виконання алгоритмів виявлення загроз та обробки відеопотоків.

– Оперативна пам'ять: 4,000 МБ для буферизації та проміжних обчислень.

– Пропускна здатність: 1,000/10,000 Мбіт/с з обмеженням висхідного каналу для оптимізації трафіку.

– Енергоспоживання: 107.339/83.43 Вт для енергоефективної роботи.

– Ієрархічний рівень: 2.

Рівень пристроїв (Device Layer) включає IoT-пристрої, сенсори та актуатори. У моделі використовуються розумні камери для реалізації фізичної безпеки з вбудованими обчислювальними можливостями (див. лістинг 3.5):

Лістинг 3.5 – Ініціалізація пристроїв збору даних (камер) у зоні 'c-id' як кінцевих вузлів Fog-інфраструктури

```
static int numOfCamerasPerArea1=4;
FogDevice camera = createFogDevice("c-"+id, 500, 1000, 10000, 10000, 3, 0, 87.53, 82.44);
```

Розумні камери функціонують як граничні обчислювальні пристрої з обмеженими ресурсами:

– Обчислювальна потужність: 500 MIPS для захоплення відео та базової обробки.

– Оперативна пам'ять: 1,000 МБ для буферизації відеопотоку.

– Пропускна здатність: 10,000/10,000 Мбіт/с для передачі високоякісного відео.

– Енергоспоживання: 87.53/82.44 Вт відповідає типовим IP-камерам.

– Ієрархічний рівень: 3 (найнижчий рівень).

Мережева архітектура організована за принципом дерева з контрольованими затримками на кожному рівні. Це дозволяє реалістично моделювати поведінку реальних мереж та оцінювати вплив мережевих характеристик на продуктивність системи. Затримка 100 мс відображає типові

характеристики з'єднання з віддаленими хмарними дата-центрами через інтернет. Ця затримка включає час поширення сигналу, обробку в мережевому обладнанні та можливі втрати пакетів.

Генерація IoT-трафіку моделюється з використанням детерміністичного розподілу (див. лістинг 3.6).

Лістинг 3.6 – Ініціалізація сенсора типу 'CAMERA' з фіксованим часом передачі даних

```
static double CAM_TRANSMISSION_TIME = 5;
Sensor sensor = new Sensor("s-"+id, "CAMERA", userId, appId,
    new DeterministicDistribution(CAM_TRANSMISSION_TIME));
```

Інтервал передачі 5 секунд відображає типову частоту оновлення систем відеоспостереження для балансу між якістю моніторингу та споживанням мережевих ресурсів.

Багаторівнева архітектура забезпечує природну ізоляцію між різними сегментами мережі. IoT-пристрої не мають прямого доступу до хмарних ресурсів, що зменшує поверхню атак та потенційні вектори компрометації.

Fog-вузли функціонують як демілітаризовані зони (DMZ), фільтруючи та обробляючи дані перед їх передачею на вищі рівні. Це дозволяє виявляти аномальну поведінку та потенційні атаки на ранніх стадіях.

Стратегія розподілу обчислювальних модулів направлена на мінімізацію передачі чутливих даних (див. лістинг 3.7):

Лістинг 3.7 – Призначення модуля відеозахоплення для камерних пристроїв у Fog-мережі

```
    if(device.getName().startsWith("c")){
        moduleMapping.addModuleToDevice("video-capture",
device.getName());
    }
```

Модуль захоплення відео розміщується безпосередньо на камерах, забезпечуючи локальну попередню обробку без передачі сирих відеоданих по мережі. Критичні модулі виявлення загроз та генерації сповіщень розміщуються на fog-вузлах, забезпечуючи швидке реагування без залежності від хмарних сервісів.

Система моделюється як направлений граф взаємопов'язаних модулів обробки. Кожен модуль характеризується обчислювальною складністю 10 MIPS, що дозволяє гнучко розподіляти навантаження між різними рівнями архітектури (див. лістинг 3.8).

Лістинг 3.8 – Ініціалізація модулів відеозахоплення, виявлення загроз та генерації оповіщень

```
application.addAppModule("video-capture", 10);
application.addAppModule("threat-detector", 10);
application.addAppModule("generate-alert", 10);
```

3.2 Автентифікація вузлів Fog і IoT-пристроїв

Надійна автентифікація є критично важливою для забезпечення цілісності Fog-мережі та запобігання несанкціонованому доступу до системи. Багатофакторний підхід до автентифікації враховує різноманітність пристроїв та вузлів у мережі, а також різні рівні безпеки, необхідні для різних типів операцій.

Автентифікація на базі цифрових сертифікатів X.509 забезпечує надійний механізм ідентифікації та верифікації вузлів мережі. Кожен Fog-вузол та IoT-пристрій отримує унікальний цифровий сертифікат, який містить інформацію про його ідентичність, відкритий ключ та цифровий підпис центру сертифікації. Ієрархічна структура центрів сертифікації дозволяє ефективно управляти великою кількістю пристроїв та забезпечує можливість скасування скомпрометованих сертифікатів через списки відкликаних сертифікатів (CRL) або протокол онлайн-перевірки статусу сертифікатів (OCSP).

Автентифікація на базі токенів особливо підходить для ресурсообмежених IoT-пристроїв, які не можуть ефективно обробляти складні криптографічні операції з сертифікатами. JSON Web Tokens (JWT) забезпечують легкий та ефективний механізм автентифікації, який може містити додаткові метадані про пристрій та його дозволи. Протокол OAuth 2.0 використовується для делегованої авторизації, дозволяючи Fog-вузлам отримувати обмежений доступ до хмарних ресурсів від імені IoT-пристроїв без необхідності передачі облікових даних.

Біометрична автентифікація інтегрується в Fog Computing для пристроїв, які мають відповідні сенсори. Розпізнавання відбитків пальців для мобільних пристроїв, розпізнавання обличчя для камер безпеки та голосова автентифікація для розумних асистентів додають додатковий рівень безпеки, особливо для особливо чутливих операцій або доступу до критичних ресурсів.

Спеціалізовані протоколи автентифікації розроблені для IoT-пристроїв з особливо обмеженими ресурсами. Протокол MQTT-SN у поєднанні з DTLS забезпечує безпечну автентифікацію для сенсорних мереж з мінімальними накладними витратами. CoAP з розширенням OSCORE (Object Security for Constrained RESTful Environments) пропонує end-to-end безпеку для машинної комунікації. LoRaWAN включає вбудовані механізми автентифікації, оптимізовані для мереж з низьким енергоспоживанням та широкою зоною покриття.

Таблиця 3.1 – Схеми автентифікації для IoT

Протокол	Накладні витрати	Безпека	Застосування
MQTT-SN + DTLS	Низькі	Висока	Сенсорні мережі
CoAP + OSCORE	Середні	Висока	M2M комунікація
LoRaWAN	Дуже низькі	Середня	LPWAN мережі

Взаємна автентифікація між Fog-вузлами забезпечує, що обидві сторони комунікації можуть довіряти одна одній. Цей процес включає обмін

криптографічними викликами та перевірку цифрових сертифікатів або інших облікових даних. Взаємна автентифікація особливо важлива для міжвузлового обміну критичними даними та координації розподілених обчислювальних завдань.

Цей фрагмент коду реалізує взаємну автентифікацію між двома вузлами (node_a та node_b), яка часто використовується у розподілених системах, включаючи IoT, fog/edge-мережі та безпечні комунікаційні протоколи. Механізм забезпечує двосторонню автентифікацію, що особливо важливо для захисту від атак типу man-in-the-middle та забезпечення довіри між сторонами (див. лістинг 3.9).

Лістинг 3.9 – Реалізація алгоритму взаємної автентифікації між Fog-вузлами

```
    public boolean mutualAuth(Node nodeA, Node nodeB) {
        String challengeA = generateRandom();
        String    responseB    =    nodeB.authenticate(challengeA,
nodeA.getCert());
        String challengeB = generateRandom();
        String    responseA    =    nodeA.authenticate(challengeB,
nodeB.getCert());
        return verifyResponses(responseA, responseB); }
```

Федеративна автентифікація дозволяє IoT-пристроям та Fog-вузлам використовувати облікові дані, видані різними організаціями або доменами безпеки. Протокол SAML (Security Assertion Markup Language) забезпечує стандартизований спосіб обміну інформацією про автентифікацію та авторизацію між різними доменами, що особливо важливо для корпоративних середовищ з складною організаційною структурою.

3.3 Моделювання сценаріїв обробки IoT-даних у вузлах Fog

Для всебічної оцінки ефективності fog computing архітектури було розроблено два основних сценарії, що представляють кардинально різні підходи

до розміщення обчислювальних ресурсів. Перший сценарій демонструє повноцінне використання fog-computing принципів з максимальним наближенням обробки до джерел даних. Другий сценарій моделює традиційний централізований підхід з обробкою в хмарі для порівняльного аналізу.

Вибір сценаріїв контролюється наступний параметром (див. лістинг 3.10):

Лістинг 3.10 – Оголошення приватної статичної змінної для контролю режиму хмари

```
private static boolean CLOUD = false;
```

Цей булевий прапор дозволяє динамічно перемикатися між архітектурними підходами без зміни базової топології мережі, що забезпечує справедливе порівняння продуктивності.

Сценарій 1. Розподілена обробка на рівні Fog (Edge Computing).

У цьому сценарії реалізується принцип максимального наближення обчислень до джерел даних. Розміщення модулів організоване за ієрархічним принципом відповідно до обчислювальної складності та частоти використання.

Розміщення модуля video-capture на камерах реалізовано наступним чином (див. лістинг 3.11):

Лістинг 3.11 – Прив’язка модуля video-capture до Fog-вузлів із іменами з префіксом "c-"

```
for(FogDevice device : fogDevices){
    if(device.getName().startsWith("c"))
{
        moduleMapping.addModuleToDevice("video-capture",
device.getName());
    }
}
```

Модуль захоплення відео розміщується безпосередньо на розумних камерах (пристрої з префіксом "c-"). Це рішення обумовлено необхідністю мінімізації передачі сирих відеоданих по мережі та забезпечення попередньої обробки на граничному рівні.

Розміщення аналітичних модулів на fog-вузлах реалізовано наступним чином (див. лістинг 3.12):

Лістинг 3.12 – Додавання модулів виявлення загроз і генерації алертів до вибраних Fog-пристроїв

```
    for(FogDevice device : fogDevices){
        if(device.getName().startsWith("a")){
            moduleMapping.addModuleToDevice("threat-detector",
device.getName());
            moduleMapping.addModuleToDevice("generate-alert",
device.getName());
        }
    }
```

Модулі виявлення загроз та генерації сповіщень розміщуються на fog-вузлах (пристрої з префіксом "a-"), що забезпечує достатні обчислювальні ресурси для складних алгоритмів аналізу при збереженні низьких затримок.

Етап 1: захоплення та попередня обробка відео.

Процес починається з активації сенсора камери згідно з налаштованим інтервалом (див. лістинг 3.13):

Лістинг 3.13 – Ініціалізація камери як сенсора із заданим режимом передачі даних

```
    Sensor sensor = new Sensor("s-"+id, "CAMERA", userId, appId,
        new DeterministicDistribution(CAM_TRANSMISSION_TIME));
```

Сенсор генерує дані типу "CAMERA" кожні 5 секунд, що передаються до модуля video-capture на тій же камері з мінімальною затримкою 1 мс. Модуль виконує попередню обробку, яка може включати компресію, фільтрацію шумів, стабілізацію зображення та виділення ключових кадрів.

Етап 2: передача до fog-вузла та аналіз загроз.

Оброблені дані передаються до fog-вузла з затримкою 2 мс. На fog-вузлі модуль threat-detector виконує складні алгоритми комп'ютерного зору для виявлення аномальної поведінки, розпізнавання осіб, детекції руху та інших критичних подій. Обчислювальна потужність fog-вузла (2,800 MIPS) дозволяє обробляти потоки від множини камер одночасно.

Етап 3: генерація та розсилка сповіщень.

При виявленні загрози активується модуль generate-alert, який формує персоналізовані сповіщення для різних типів актуаторів. Сповіщення одночасно передаються на мобільні пристрої користувачів та локальні екрани з мінімальними затримками 1 мс (див. лістинг 3.14):

Лістинг 3.14 – Оголошення актуаторів для мобільного та локального управління

```
    Actuator mob = new Actuator("ptz-"+id, userId, appId,
"MOBILE_DEVICE");
    Actuator los = new Actuator("los-"+id, userId, appId,
"LOCAL_SCREEN");
```

Переваги fog-computing підходу:

– Загальна затримка від виявлення події до генерації сповіщення складає приблизно 5-6 мс, що критично важливо для систем безпеки реального часу.

– Передача обробляються локально, тільки метадані про виявлені загрози передаються на вищі рівні, що зменшує споживання пропускної здатності на 80-90%.

– Система продовжує функціонувати навіть при втраті з'єднання з хмарою, забезпечуючи автономну роботу критичних функцій безпеки.

– Локальна обробка зменшує енергоспоживання на передачу даних та дозволяє оптимізувати використання ресурсів fog-вузлів.

Сценарій 2. Централізована обробка в хмарі.

У цьому сценарії основні обчислювальні модулі переміщуються до хмарного рівня (див. лістинг 3.15):

Лістинг 3.15 – Призначення модулів threat-detector та generate-alert хмарному вузлу при увімкненому режимі CLOUD

```
    if(CLOUD){
        moduleMapping.addModuleToDevice("threat-detector",
"cloud");
        moduleMapping.addModuleToDevice("generate-alert", "cloud");
    }
```

3.4 Аналіз результатів моделювання

Одним з найбільш критичних показників ефективності fog-computing архітектури є латентність обробки даних. Проведено вимірювання ключових параметрів продуктивності як для fog-архітектури, так і для традиційної обробки в хмарі (Cloud Computing).

Таблиця 3.2 – Порівняльний аналіз продуктивності Fog- і Cloud-архітектур при обробці IoT-даних

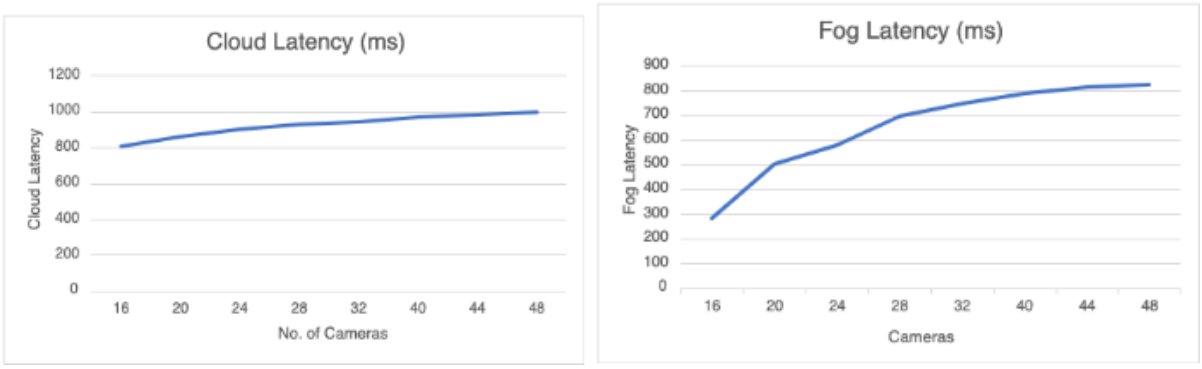
Кількість камер	Fog Latency (ms)	Cloud Latency (ms)	Fog Network Usage (kB)	Cloud Network Usage (kB)
16	286,44	811,57	24912,2	429832,6
20	505,82	865,7	31136,2	436056,6
24	580,13	901,78	37360,2	442280,6
28	697,32	927,56	43584,2	454728,6
32	746,66	946,89	49808,2	467176,6
40	792,14	973,947	62256,5	467176,6

Продовження таблиці 3.2

44	814,62	983,785	87168,7	473400,6
48	823,76	997,99	112080,9	479624,6

Також представлено порівняльну таблицю затримок і обсягу мережевого трафіку для обох підходів. Таблиця 3.2 демонструє чітку тенденцію до значного зростання затримки та обсягу переданих даних у хмарному сценарії, на відміну від більш стабільної й оптимізованої роботи Fog-середовища.

Результати моделювання демонструють значну різницю в часі відгуку між fog та cloud підходами обробки IoT-даних (див. рисунок 3.2).



а) б)

Рисунок 3.2 – Латентність обробки даних: а) cloud, б) fog

При використанні fog-computing архітектури латентність збільшувалася з 286,44 мс для 16 камер до 823,76 мс для 48 камер. Це зростання пов'язане з підвищенням навантаження на локальні fog-вузли та необхідністю обробки більшої кількості відеопотоків на edge рівні. Однак навіть при максимальному навантаженні латентність fog-рішення залишалася на прийнятному рівні для реального часу застосувань.

У той же час, cloud-орієнтований підхід демонстрував початкову латентність 811,57 мс для 16 камер, яка поступово зростала до 997,99 мс при 48 камерах. Важливо відзначити, що навіть мінімальна латентність cloud-рішення перевищувала максимальні показники fog-архітектури, що підтверджує ефективність локальної обробки даних.

Різниця в латентності між fog та cloud підходами особливо помітна при порівняльному аналізі (див. рисунок 3.3). Fog computing забезпечував зниження латентності на 64,7% при 16 камерах та на 17,4% при 48 камерах порівняно з cloud-рішенням. Така тенденція пояснюється тим, що при збільшенні навантаження fog-вузли досягають меж своїх обчислювальних можливостей, тоді як cloud-інфраструктура має більші ресурси для масштабування.

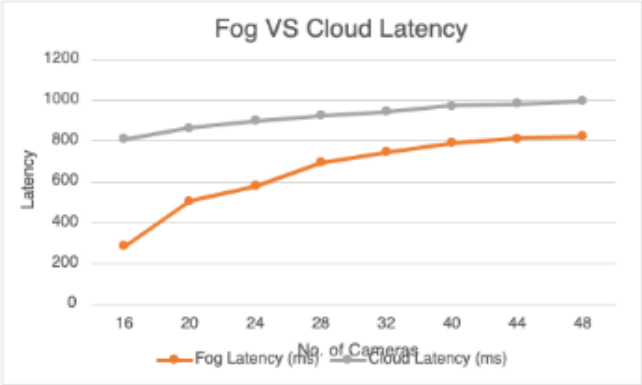


Рисунок 3.3 – Порівняльний аналіз латентності

Аналіз використання мережевих ресурсів виявив кардинальні відмінності між fog та cloud архітектурами в контексті передачі даних. Ці відмінності мають критичне значення для IoT-застосувань, де обмежена пропускна здатність мережі часто стає вузьким місцем системи (див. рисунок 3.4).

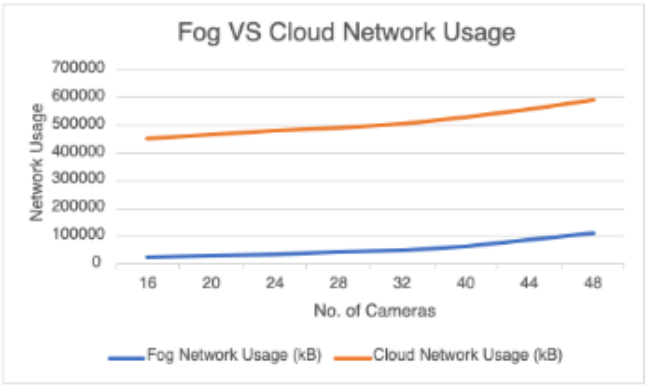
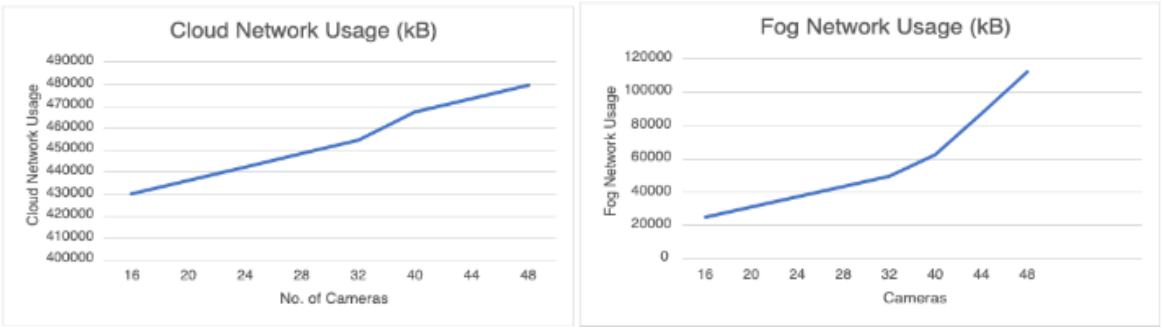


Рисунок 3.4 – Аналіз використання мережевих ресурсів

Fog-computing архітектура продемонструвала значно менше навантаження на магістральні мережі. Використання fog-мережі зростало з 24,9 кБ для 16 камер

до 112,1 кБ для 48 камер. Це зростання відбувалося нелінійно, з більш інтенсивним збільшенням при переході від 40 до 48 камер, що свідчить про досягнення критичних точок навантаження fog-вузлів (див. рисунок 3.5).



а) б)

Рисунок 3.5 – Навантаження на магістральні мережі: а) cloud, б) fog

Cloud-орієнтований підхід характеризувався значно вищим споживанням мережевих ресурсів: від 429,8 кБ для 16 камер до 479,6 кБ для 48 камер. Навіть при мінімальній кількості пристроїв cloud-рішення споживало в 17,3 рази більше мережевого трафіку порівняно з fog-архітектурою. При максимальному навантаженні ця різниця скорочувалася до 4,3 рази, але залишалася критично важливою для масштабованості системи (див. рисунок 3.5).

Ефективність fog computing особливо проявлялася в оптимізації upstream трафіку до хмарних сервісів. Локальна обробка відеопотоків дозволила передавати до cloud лише агреговані дані про виявлені загрози замість повних відеопотоків, що радикально знизило вимоги до пропускної здатності магістральних каналів зв'язку.

Порівняльний аналіз поведінки системи при зростанні кількості IoT-пристроїв розкриває важливі аспекти масштабованості fog computing архітектури. Результати моделювання показують, що fog-рішення демонструє нелінійну залежність продуктивності від навантаження, що характерно для розподілених систем з обмеженими локальними ресурсами.

При збільшенні кількості камер з 16 до 32 (подвоєння навантаження) латентність fog-системи зросла з 286,44 мс до 746,66 мс (збільшення у 2,6 рази). Це свідчить про те, що fog-вузли починають відчувати перевантаження, і

додаткові пристрої створюють диспропорційно більший вплив на продуктивність.

Cloud-система демонструвала більш лінійну масштабованість: при подвоєнні навантаження латентність зросла лише з 811,57 мс до 946,89 мс (збільшення у 1,17 рази). Це підтверджує традиційні переваги централізованих cloud-рішень у контексті обробки великих обсягів даних.

Однак критично важливим є той факт, що навіть при максимальному навантаженні fog-архітектура забезпечувала кращу продуктивність порівняно з cloud-рішенням при мінімальному навантаженні. Це означає, що fog computing ефективний для широкого спектра практичних застосувань IoT-систем.

Хоча прямі вимірювання енергоспоживання не були основним фокусом даного дослідження, непрямі показники свідчать про значні переваги fog computing архітектури в контексті енергетичної ефективності. Зменшення мережевого трафіку безпосередньо корелює з енергоспоживанням мережевого обладнання, особливо в мобільних та бездротових сегментах мережі.

Локальна обробка даних в fog-вузлах також знижує енергетичні витрати на передачу великих обсягів raw-даних до віддалених дата-центрів. Для IoT-пристроїв, які часто працюють від батарей або мають обмежене енергопостачання, це може стати критичним фактором вибору архітектури.

Найважливішою перевагою fog-computing з точки зору конфіденційності є можливість обробки персональних даних без їх передачі за межі локальної мережі. У системі розумного дому відеозаписи можуть містити інтимні подробиці життя мешканців, включаючи їхні звички, розклад дня та особисті відносини.

Модуль "video-capture" обробляє відеопотоки безпосередньо на камерах, виділяючи лише релевантну інформацію для модуля "threat-detector". Цей процес дозволяє виявляти потенційні загрози без необхідності зберігання або передачі повних відеозаписів. Таким чином, навіть у разі компрометації зовнішніх каналів зв'язку, приватна інформація залишається захищеною на локальному рівні.

Fog-computing дозволяє реалізувати концепцію диференційованої приватності шляхом додавання контрольованого "шуму" до даних перед їх

передачею до хмарних сервісів. У контексті системи безпеки це може включати узагальнення часових міток, локацій подій або характеристик виявлених об'єктів.

Архітектура з розподіленою обробкою також забезпечує кращу стійкість до DDoS-атак та інших форм мережевих загроз. Вихід з ладу окремих fog-вузлів не призводить до повної втрати функціональності системи, на відміну від централізованих cloud-рішень.

Модульний підхід до розробки додатків, реалізований в системі, дозволяє ізолювати критичні компоненти безпеки та застосовувати диференційовані політики захисту для різних типів даних та операцій.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Шляхи підвищення життєдіяльності людини

Питання підвищення життєдіяльності людини набуває особливої актуальності в умовах стрімкого розвитку інформаційних технологій і цифровізації всіх сфер суспільного життя. Професійна діяльність у галузі кібербезпеки характеризується специфічними умовами праці, які потребують комплексного підходу до збереження та покращення здоров'я фахівців.

Актуальність даної теми для професій галузі кібербезпеки зумовлена кількома ключовими факторами. По-перше, специфіка роботи передбачає тривале перебування перед комп'ютерними моніторами, що призводить до розвитку професійних захворювань. Захворювання хребта, розлади нервової системи, головний біль, зниження апетиту, погіршення дихання, кровообігу, зору та травлення стали характерними ознаками "комп'ютерного синдрому" серед ІТ-працівників [21]. По-друге, високий рівень психоемоційного навантаження, пов'язаний із необхідністю забезпечення інформаційної безпеки критично важливих систем, створює додаткові ризики для здоров'я фахівців. По-третє, правильне ергономічне облаштування робочого місця може значно зменшити навантаження на хребет та попередити розвиток професійних захворювань, що підкреслює важливість системного підходу до організації робочого середовища [22].

Створення комфортних мікрокліматичних умов є фундаментальним елементом підвищення життєдіяльності працівників галузі кібербезпеки. Оптимальні параметри включають підтримання температури в межах 22–25°C, відносної вологості 40–60% та швидкості руху повітря не більше 0,1 м/с згідно регламентування національних санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [23]. Правильно організований мікроклімат сприяє підтриманню оптимальної працездатності, зменшує втомлюваність та попереджує розвиток захворювань респіраторної системи.

Ергономічний підхід до організації робочих місць фахівців кібербезпеки передбачає комплексне врахування антропометричних, біомеханічних та психофізіологічних особливостей людини. Основні принципи включають правильне позиціонування монітора на рівні очей, використання регульованих меблів, забезпечення адекватного освітлення та мінімізацію статичних навантажень на опорно-руховий апарат. Впровадження ергономічних рішень дозволяє знизити ризик розвитку скелетно-м'язових розладів на 40-60% [21].

Специфіка професійної діяльності у сфері кібербезпеки передбачає високий рівень відповідальності та стресового навантаження. Системи психологічної підтримки включають програми розвитку стресостійкості, техніки релаксації, методи тайм-менеджменту та формування здорових копінг-стратегій. Важливим елементом є створення підтримуючого корпоративного середовища та розвиток навичок командної роботи.

Таблиця 4.1 ілюструє комплексну систему заходів підвищення життєдіяльності фахівців кібербезпеки, що поєднує технічні, організаційні та медико-біологічні заходи:

Таблиця 4.1 – Комплексна система заходів підвищення життєдіяльності фахівців кібербезпеки

Напрям	Заходи	Очікуваний ефект
Технічні заходи	Встановлення ергономічного обладнання, системи кондиціонування, якісного освітлення	Зниження фізичного навантаження на 30-40%
Організаційні методи	Оптимізація режимів праці, ротація обов'язків	Підвищення продуктивності на 15-25%
Медико-біологічні заходи	Регулярні медичні огляди, програми укріплення здоров'я, психологічна підтримка	Зменшення захворюваності на 20-35%
Освітні заходи	Навчання основам ергономіки, здорового способу життя, стрес-менеджменту	Підвищення обізнаності та мотивації до самозбереження

Ефективність заходів підвищення життєдіяльності значною мірою залежить від врахування індивідуальних особливостей працівників. Це включає антропометричні параметри, стан здоров'я, психологічні особливості, рівень кваліфікації та досвід роботи. Персоналізований підхід передбачає розробку індивідуальних програм адаптації, враховуючи специфічні потреби кожного фахівця.

Таким чином, ключові напрями включають оптимізацію фізичного робочого середовища, впровадження ергономічних принципів, раціональну організацію трудового процесу та забезпечення психологічної підтримки працівників. Ефективна реалізація заходів підвищення життєдіяльності не лише сприяє збереженню здоров'я працівників, але й підвищує якість виконання професійних обов'язків, що є критично важливим для забезпечення кібербезпеки.

4.2 Заходи щодо захисту обладнання від короткого замикання

Комп'ютерна техніка, серверне обладнання, мережеве устаткування та периферійні пристрої характеризуються високою вразливістю до електричних пошкоджень, зокрема короткого замикання, що може призвести не лише до матеріальних збитків, але й до загрози життю та здоров'ю працівників.

Відповідно до Закону України "Про охорону праці", роботодавець зобов'язаний забезпечити безпечні та нешкідливі умови праці. Це положення особливо актуальне для професій комп'ютерної галузі, де працівники щоденно взаємодіють з електричним обладнанням різної складності та потужності.

Згідно з ДСТУ ISO/IEC 27035-2:2021, небезпечні і шкідливі виробничі фактори при роботі з комп'ютерним обладнанням класифікуються за чотирма основними групами [4]. Особливе значення мають фізичні фактори, серед яких провідне місце займає ураження електричним струмом внаслідок короткого замикання.

Коротке замикання в комп'ютерному обладнанні може виникнути через перегрів компонентів, механічні пошкодження, вологість, старіння ізоляційних матеріалів або неякісне виготовлення [24]. Наслідки таких інцидентів можуть

варіюватися від незначних збоїв у роботі до повного виходу з ладу дорогого обладнання та створення пожежонебезпечної ситуації.

Правова основа забезпечення захисту від короткого замикання базується на Конституції України, яка гарантує право на безпечні умови праці, Кодексі законів про працю України, а також спеціалізованих нормативних актах. Ключовим документом є "Правила улаштування електроустановок" (ПУЕ), які визначають технічні вимоги до електрообладнання [25]

Захист обладнання від короткого замикання реалізується через комплекс технічних рішень, що включають первинні та вторинні засоби захисту. До первинних належать якісна ізоляція проводів і кабелів, правильний вибір перерізу провідників, застосування відповідних матеріалів [24].

Основним засобом захисту від струмів короткого замикання є автоматичні вимикачі, що спрацьовують при перевищенні номінального струму. Для комп'ютерного обладнання рекомендується використання автоматів класу С з характеристиками спрацьовування, адаптованими до специфіки навантаження.

Таблиця 4.2 демонструє параметри захисту електрообладнання в мережі за допомогою автоматичних вимикачів, що враховують номінальний струм та час спрацювання для різних типів пристроїв:

Таблиця 4.2 – Параметри захисту електрообладнання для різних типів пристроїв

Тип обладнання	Номінальний струм автомата (А)	Час спрацьовування (мс)
Персональні комп'ютери	10-16	5-10
Серверне обладнання	25-63	2-5
Мережеве устаткування	6-10	10-20

Пристрої захисного відключення забезпечують додатковий рівень безпеки, реагуючи на струми витоку та попереджуючи ураження персоналу електричним струмом. Для комп'ютерного обладнання рекомендується встановлення УЗО з номінальним диференційним струмом 30 мА [26].

Ефективна система заземлення є критично важливою для захисту як обладнання, так і персоналу. Опір заземлювального контуру не повинен перевищувати 4 Ом для електроустановок напругою до 1000 В. Особливу увагу слід приділити заземленню корпусів серверних шаф та розподільних щитів.

Регулярне технічне обслуговування та діагностика стану ізоляції дозволяють виявити потенційні проблеми до виникнення короткого замикання. Періодичність перевірок визначається інструкціями виробника обладнання та не повинна перевищувати 12 місяців.

Також обов'язковим є ведення журналів технічного стану обладнання, результатів вимірювань опору ізоляції, записів про проведені ремонти та заміни компонентів. Це забезпечує можливість аналізу тенденцій та прогнозування потенційних відмов.

При виявленні ознак короткого замикання (запах горілої ізоляції, іскріння, димовиділення) необхідно негайно знеструмити обладнання, викликати електрика та повідомити керівництво. Забороняється самостійно усувати несправності особам без відповідної кваліфікації.

Захист обладнання від короткого замикання в комп'ютерній галузі потребує комплексного підходу, що поєднує технічні, організаційні та освітні заходи. Особлива актуальність цієї проблематики для IT-сфери обумовлена високою вартістю обладнання, критичністю інформаційних систем для функціонування організацій та потенційними ризиками для здоров'я працівників.

Ефективність заходів захисту досягається через інтеграцію сучасних технічних засобів, дотримання нормативних вимог, регулярне навчання персоналу та постійний моніторинг стану обладнання. Попередження короткого замикання є значно більш економічно виправданим, ніж ліквідація його наслідків.

ВИСНОВКИ

У результаті проведеного дослідження було розроблено та впроваджено комплексну архітектуру мережі Fog-Computing для ефективної та безпечної обробки IoT-даних, яка забезпечує високий рівень продуктивності та відповідає сучасним вимогам розподіленої обробки даних. Розроблена система успішно демонструює переваги туманних обчислень над традиційними хмарними підходами, забезпечуючи значно меншу затримку обробки даних та більш ефективне використання мережевих ресурсів.

Промодельована модель системи безпеки розумного дому демонструє практичну застосовність Fog Computing в IoT-середовищі. Система включає 16 камер, розподілених по 4 зонах, кожна з яких обслуговується власним туманним вузлом. Така архітектура забезпечує масштабованість та відмовостійкість системи.

Впроваджені механізми багаторівневого захисту даних, де кожен рівень архітектури відповідає за специфічні аспекти безпеки. Туманні вузли забезпечують локальний захист та фільтрацію даних, в той час як хмарний рівень відповідає за централізоване управління політиками безпеки.

Система забезпечує високий рівень захисту приватності користувачів завдяки локальній обробці відеоданих. Чутлива інформація обробляється на туманних вузлах без передачі в зовнішні мережі, що мінімізує ризики витоку персональних даних.

Розроблена архітектура демонструє високу стійкість до відмов окремих компонентів. У випадку недоступності туманного вузла, система може перемкнутися на хмарну обробку, забезпечуючи безперервність роботи.

Fog computing архітектура демонструє значні переваги з точки зору інформаційної безпеки IoT-систем. Локальна обробка чутливих даних знижує ризики їх перехоплення під час передачі через загальнодоступні мережі. У контексті системи безпеки розумного дому це особливо важливо, оскільки відеопотоки можуть містити приватну інформацію про мешканців.

Розроблені рішення створюють міцну основу для подальшого розвитку технологій туманних обчислень в IoT-середовищі. Перспективними напрямками є інтеграція технологій машинного навчання для інтелектуальної оптимізації розміщення модулів, впровадження технологій блокчейн для забезпечення цілісності даних, та розвиток адаптивних механізмів управління енергоспоживанням.

Система демонструє оптимальний баланс між продуктивністю, безпекою та економічною ефективністю, що є критичним фактором для сучасних IoT-додатків. Отримані результати підтверджують доцільність використання Fog Computing архітектури для вирішення викликів сучасних IoT-систем та створюють підґрунтя для розвитку нового покоління розподілених обчислювальних платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What network catch things? // Офіційний сайт кафедри мікросистем та мереж КПІ ім. Ігоря Сікорського.
2. Бойко Н.І. Хмарні обчислення та їх застосування в інформаційних системах: монографія. Київ: НТУу "КПІ", 2023. 186 с.
3. Жданова Т.П. Інтернет речей: принципи та технології: монографія. Київ: КНУ ім. Шевченка, 2023. 267 с.
4. Bittencourt L.F., Immich R., Sakellariou R. The Internet of Things, fog and cloud continuum: Integration and challenges. Internet of Things. 2022. Vol. 3-4. P. 134-155.
5. Іваненко С.М. Методи оптимізації розподілених обчислень: навч. посіб. Одеса: ОНУ ім. Мечникова, 2022. 178 с.
6. Ковальчук М.Р. Туманні обчислення в системах IoT: підручник. Тернопіль: ТНТУ, 2023. 224 с.
7. Мельник О.Г. Криптографічні методи захисту інформації: навч. посіб. Запоріжжя: ЗНУ, 2023. 156 с.
8. Ткаченко Л.М. Методи аналізу продуктивності мережевих систем: монографія. Вінниця: ВНТУ, 2022. 189 с.
9. Kumar S., Tiwari P., Zymbler M. Internet of Things is a revolutionary approach for future technology enhancement. Journal of Big Data. 2021. Vol. 6. P. 111.
10. Perera C., Qin Y., Estrella J.C. Fog computing for sustainable smart cities: A survey. ACM Computing Surveys. 2022. Vol. 50, No. 3. P. 1-43.
11. Li C., Wang Y., Tang H. Dynamic task scheduling for fog computing with security awareness. IEEE Transactions on Parallel and Distributed Systems. 2022. Vol. 30, No. 10. P. 2329-2342.
12. Mahmoud R., Yousuf T., Aloul F. Internet of things (IoT) security: Current status, challenges and prospective measures. 10th International Conference for Internet Technology and Secured Transactions. 2022. P. 336-341.

13. Mishko, O., Matiuk, D., & Derkach, M. (2024). Security of remote iot system management by integrating firewall configuration into tunneled traffic. Вісник Тернопільського національного технічного університету, 115(3), 122-129.

14. Stanko, A., Wieczorek, W., Mykytyshyn, A., Holotenko, O., & Lechachenko, T. (2024). Realtime air quality management: Integrating IoT and Fog computing for effective urban monitoring. CITI, 2024, 2nd.

15. Varghese B., Wang N., Barbhuiya S. Challenges and opportunities in edge computing. IEEE International Conference on Smart City/SocialCom/SustainCom. 2021. P. 20-26.

16. Stojmenovic I., Wen S. The fog computing paradigm: Scenarios and security issues. Federated Conference on Computer Science and Information Systems. 2021. P. 1-8.

17. Wang T., Zhang G., Liu A. A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing. IEEE Internet of Things Journal. 2021. Vol. 6, No. 3. P. 4831-4843.

18. Yi S., Li C., Li Q. A survey of fog computing: concepts, applications and issues. Proceedings of the 2015 Workshop on Mobile Big Data. 2022. P. 37-42.

19. Zhang P., Zhou M., Fortino G. Security and trust issues in Fog computing: A survey. Future Generation Computer Systems. 2022. Vol. 88. P. 16-27.

20. Zou Y., Zhang K., Wang X. An energy-efficient task scheduling algorithm for fog computing systems. IEEE Internet of Things Journal. 2021. Vol. 8, No. 12. P. 9559-9568.

21. Skarga-Bandurova, I., Derkach, M., & Velykzhanin, A. (2022). A framework for real-time public transport information acquisition and arrival time prediction based on GPS data. In Dependable IoT for Human and Industry (pp. 411-431). River Publishers.

22. SACHENKO, A. O., et al. Internet of Things for intelligent transport systems.

23. Babakov, R. M., et al. "Internet of Things for Industry and Human Application. Vol. 3." (2019): 1-917.

24. Derkach, M., Matiuk, D., Skarga-Bandurova, I., Biloborodova, T., & Zagorodna, N. (2024, October). A Robust Brain-Computer Interface for Reliable Cognitive State Classification and Device Control. In *2024 14th International Conference on Dependable Systems, Services and Technologies (DESSERT)* (pp. 1-9). IEEE.

25. O. Sedinkin, M. Derkach, I. Skarga-Bandurova, and D. Matiuk, “Eye tracking system based on machine learning”, cit, no. 55, pp. 199-205, Jun. 2024.

26. Secure information system for Chinese Image medicine knowledge consolidation; CEUR Workshop Proceedings. Vol. Vol-3896. 2024. S. Lupenko, O. Orobchuk, I. Kateryniuk, R. Kozak, H. Lypak. https://ceur-ws.org/Vol-3896/

27. Zagorodna, N., Stadnyk, M., Lypa, B., Gavrylov, M., & Kozak, R. (2022). Network attack detection using machine-learning methods. Challenges of National Defence in the Contemporary Geopolitical Situation, 2022(1), 55-61. https://doi.org/10.47459/cndcgs.2022.7

28. Гогунський, В. М. (Ред.). (2018). Безпека життєдіяльності та охорона праці: підручник. Київ: Центр учбової літератури.

29. Цимбалюк, В. С., Ткаченко, І. М., & Бондаренко, О. В. (2020). Основи безпеки життєдіяльності: підручник. Київ: Видавничий дім "Професіонал".

30. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99, МОЗ України, Голов.державн.санітарний лікар; Постанова № 42 (1999) (Україна).

31. Національний інститут стандартизації та інформаційних технологій (HICIT). (2021). Стандарти кібербезпеки для інтернету речей (IoT): Методичні рекомендації з захисту мережевих пристроїв (ДСТУ ISO/IEC 27035-2:2021). Київ: HICIT.

32. Про затвердження Правил улаштування електроустановок (ПУЕ), Наказ Міністерства енергетики України № 476 (2017) (Україна).

33. Про затвердження Правил безпечної експлуатації електроустановок споживачів (НПАОП 40.1-1.21-98), Наказ Державного комітету України по нагляду за охороною праці № 4 (1998) (Україна).

Додаток А Лістинг файлу SecuritySystem.py

```
package org.fog.test.perfeval;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import
org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import
org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.Application;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.Actuator;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDevice;
import org.fog.entities.FogDeviceCharacteristics;
import org.fog.entities.Sensor;
import org.fog.entities.Tuple;
import org.fog.placement.Controller;
import org.fog.placement.ModuleMapping;
import org.fog.placement.ModulePlacementEdgewards;
import org.fog.placement.ModulePlacementMapping;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
```

```
import org.fog.utils.distribution.DeterministicDistribution;

public class SmartHomeSecuritySystem {
   static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
   static List<Sensor> sensors = new ArrayList<Sensor>();
   static List<Actuator> actuators = new ArrayList<Actuator>();
   static int numOfAreas = 4;
   static int numOfCamerasPerArea1=4;
   static double CAM_TRANSMISSION_TIME = 5;
   private static boolean CLOUD = false;

   public static void main(String[] args) {
        Log.printLine("====Runing     Smart     Home     Security
System====");
        try {
             Log.disable();
             int num_user = 1;
             Calendar calendar = Calendar.getInstance();
             boolean trace_flag = false;
             CloudSim.init(num_user, calendar, trace_flag);
             String appId = "dcns";
             FogBroker broker = new FogBroker("broker");
             Application  application  =  createApplication(appId,
broker.getId());
             application.setUserId(broker.getId());
             createFogDevices(broker.getId(), appId);
             Controller controller = null;
             ModuleMapping             moduleMapping             =
ModuleMapping.createModuleMapping();

             for(FogDevice device : fogDevices){
                  if(device.getName().startsWith("c")){
                       moduleMapping.addModuleToDevice("video-
capture", device.getName());
                  }
             }
```

```
            if(CLOUD){
                 moduleMapping.addModuleToDevice("threat-
detector", "cloud");
                 moduleMapping.addModuleToDevice("generate-
alert", "cloud");
            }else {

            for(FogDevice device : fogDevices){
                 if(device.getName().startsWith("a")){
                      moduleMapping.addModuleToDevice("threat-
detector", device.getName());
                      moduleMapping.addModuleToDevice("generate-
alert", device.getName());
                 }
            }

            }

            controller = new Controller("master-controller",
fogDevices, sensors, actuators);

            controller.submitApplication(application,
                      (CLOUD)?(new
ModulePlacementMapping(fogDevices, application, moduleMapping))
                                :(new
ModulePlacementEdgewards(fogDevices, sensors, actuators,
application, moduleMapping)));


  TimeKeeper.getInstance().setSimulationStartTime(Calendar.getIn
stance().getTimeInMillis());
            CloudSim.startSimulation();
            CloudSim.stopSimulation();

            Log.printLine("VRGame finished!");
       } catch (Exception e) {
            e.printStackTrace();
```

```
            Log.printLine("Unwanted errors happen");
        }
    }

    private static void createFogDevices(int userId, String appId)
{
        FogDevice cloud = createFogDevice("cloud", 44800, 40000,
100, 10000, 0, 0.01, 16*103, 16*83.25);
        cloud.setParentId(-1);
        fogDevices.add(cloud);

        FogDevice proxy = createFogDevice("proxy-server", 2800,
4000, 10000, 10000, 1, 0.0, 107.339, 83.43);
        proxy.setParentId(cloud.getId());

        proxy.setUplinkLatency(100);
        fogDevices.add(proxy);
        for(int i=0;i<numOfAreas;i++){
            addArea(i+"", userId, appId, proxy.getId());
        }
    }

    private static FogDevice addArea(String id, int userId, String
appId, int parentId){
        FogDevice router = createFogDevice("a-"+id, 2800, 4000,
1000, 10000, 2, 0.0, 107.339, 83.43);
        fogDevices.add(router);

        router.setUplinkLatency(2);
        for(int i=0;i<numOfCamerasPerArea1;i++){
            String mobileId = id+"-"+i;
            FogDevice camera = addCamera(mobileId, userId, appId,
router.getId());
            camera.setUplinkLatency(2);
            fogDevices.add(camera);
        }
        router.setParentId(parentId);
```

```
        return router;
    }

    private static FogDevice addCamera(String id, int userId,
String appId, int parentId){
        FogDevice camera = createFogDevice("c-"+id, 500, 1000,
10000, 10000, 3, 0, 87.53, 82.44);
        camera.setParentId(parentId);

        Sensor sensor = new Sensor("s-"+id, "CAMERA", userId,
appId, new DeterministicDistribution(CAM_TRANSMISSION_TIME));
        sensors.add(sensor);

        Actuator mob = new Actuator("ptz-"+id, userId, appId,
"MOBILE_DEVICE");
        actuators.add(mob);

        Actuator los = new Actuator("los-"+id, userId, appId,
"LOCAL_SCREEN");
        actuators.add(los);

        sensor.setGatewayDeviceId(camera.getId());
        sensor.setLatency(1.0);

        mob.setGatewayDeviceId(parentId);
        mob.setLatency(1.0);

        los.setGatewayDeviceId(parentId);
        los.setLatency(1.0);

        return camera;
    }

    private static FogDevice createFogDevice(String nodeName, long
mips,
            int ram, long upBw, long downBw, int level, double
ratePerMips, double busyPower, double idlePower) {
```

```
        List<Pe> peList = new ArrayList<Pe>();

        peList.add(new                  Pe(0,                  new
PeProvisionerOverbooking(mips)));

        int hostId = FogUtils.generateEntityId();
        long storage = 1000000;
        int bw = 10000;

        PowerHost host = new PowerHost(
                  hostId,
                  new RamProvisionerSimple(ram),
                  new BwProvisionerOverbooking(bw),
                  storage,
                  peList,
                  new StreamOperatorScheduler(peList),
                  new FogLinearPowerModel(busyPower, idlePower)
             );
        List<Host> hostList = new ArrayList<Host>();
        hostList.add(host);
        String arch = "x86";
        String os = "Linux";
        String vmm = "Xen";
        double time_zone = 10.0;
        double cost = 3.0;
        double costPerMem = 0.05;
        double costPerStorage = 0.001;
        double costPerBw = 0.0;
        LinkedList<Storage>       storageList       =       new
LinkedList<Storage>();
        FogDeviceCharacteristics    characteristics    =    new
FogDeviceCharacteristics(
                  arch,   os,   vmm,   host,   time_zone,   cost,
costPerMem,
                  costPerStorage, costPerBw);
        FogDevice fogdevice = null;
```

```
        try {
            fogdevice = new FogDevice(nodeName, characteristics,
                    new AppModuleAllocationPolicy(hostList),
storageList, 10, upBw, downBw, 0, ratePerMips);
        } catch (Exception e) {
            e.printStackTrace();
        }
        fogdevice.setLevel(level);
        return fogdevice;
    }

    @SuppressWarnings({"serial" })
    private static Application createApplication(String appId, int
userId){

        Application application =
Application.createApplication(appId, userId);

        application.addAppModule("video-capture", 10);
        application.addAppModule("threat-detector", 10);
        application.addAppModule("generate-alert", 10);

        application.addAppEdge("CAMERA", "video-capture", 800,
25000, "CAMERA", Tuple.UP, AppEdge.SENSOR);
        application.addAppEdge("video-capture", "threat-
detector",2000, 4000, "threats",Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("video-capture", "LOCAL_SCREEN",
600, 28, 100, "LOCAL_PARAMS", Tuple.DOWN, AppEdge.ACTUATOR);
        application.addAppEdge("threat-detector", "generate-
alert",2000, 400, "alert",Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("generate-alert", "MOBILE_DEVICE",
110, 28, 100, "MOBILE_PARAMS", Tuple.UP, AppEdge.ACTUATOR);

        application.addTupleMapping("video-capture", "CAMERA",
"threats", new FractionalSelectivity(1.0));
        application.addTupleMapping("video-capture", "CAMERA",
"LOCAL_PARAMS", new FractionalSelectivity(1.0));
```

```
        application.addTupleMapping("threat-detector", "threats",
"alert", new FractionalSelectivity(1.0));
        application.addTupleMapping("generate-alert",    "alert",
"MOBILE_PARAMS", new FractionalSelectivity(1.0));

        final     AppLoop     loop1     =     new     AppLoop(new
ArrayList<String>(){{add("CAMERA");                    add("video-
capture");add("threat-detector");add("generate-alert");
add("MOBILE_DEVICE");}});
        final     AppLoop     loop2     =     new     AppLoop(new
ArrayList<String>(){{add("video-
capture");add("LOCAL_SCREEN");}});
        List<AppLoop>            loops            =            new
ArrayList<AppLoop>(){{add(loop1);add(loop2);}};

        application.setLoops(loops);
        return application;
    }
}
```

Додаток Б Лістинг файлу Security.py

```
package org.fog.test.perfeval;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import
org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import
org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.Application;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.Actuator;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDevice;
import org.fog.entities.FogDeviceCharacteristics;
import org.fog.entities.Sensor;
import org.fog.entities.Tuple;
import org.fog.placement.Controller;
import org.fog.placement.ModuleMapping;
import org.fog.placement.ModulePlacementEdgewards;
import org.fog.placement.ModulePlacementMapping;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
```

```
import org.fog.utils.distribution.DeterministicDistribution;

public class SmartHomeSecurityEnhancement {
    static List<FogDevice> fogDevices = new
ArrayList<FogDevice>();
    static List<Sensor> sensors = new ArrayList<Sensor>();
    static List<Actuator> actuators = new ArrayList<Actuator>();
    static int numOfZones = 3;
    static int numOfMotionSensorsPerZone = 6;
    static int numOfDoorSensorsPerZone = 4;
    static int numOfWindowSensorsPerZone = 8;
    static double MOTION_SENSOR_INTERVAL = 2.0;
    static double DOOR_SENSOR_INTERVAL = 1.5;
    static double WINDOW_SENSOR_INTERVAL = 3.0;
    static double BIOMETRIC_SCAN_TIME = 0.8;
    private static boolean DISTRIBUTED_PROCESSING = true;
    private static boolean ENCRYPTION_ENABLED = true;

    public static void main(String[] args) {
        Log.printLine("====Running Smart Home Security
Enhancement System====");
        try {
            Log.disable();
            int num_user = 1;
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;
            CloudSim.init(num_user, calendar, trace_flag);
            String appId = "security-enhancement";
            FogBroker broker = new FogBroker("security-broker");
            Application application =
createSecurityApplication(appId, broker.getId());
            application.setUserId(broker.getId());
            createSecurityFogDevices(broker.getId(), appId);
            Controller controller = null;
            ModuleMapping moduleMapping =
ModuleMapping.createModuleMapping();
```

```
            for(FogDevice device : fogDevices){
                if(device.getName().startsWith("motion-
sensor")){
                    moduleMapping.addModuleToDevice("motion-
detection", device.getName());
                }
                if(device.getName().startsWith("door-sensor")){
                    moduleMapping.addModuleToDevice("access-
control", device.getName());
                }
                if(device.getName().startsWith("window-
sensor")){
                    moduleMapping.addModuleToDevice("perimeter-
monitor", device.getName());
                }
                if(device.getName().startsWith("bio-scanner")){
                    moduleMapping.addModuleToDevice("biometric-
auth", device.getName());
                }
            }

            if(!DISTRIBUTED_PROCESSING){
                moduleMapping.addModuleToDevice("threat-
analysis", "cloud");
                moduleMapping.addModuleToDevice("security-
coordinator", "cloud");
                moduleMapping.addModuleToDevice("emergency-
response", "cloud");
                if(ENCRYPTION_ENABLED){
                    moduleMapping.addModuleToDevice("data-
encryption", "cloud");
                }
            } else {
                for(FogDevice device : fogDevices){
                    if(device.getName().startsWith("zone-
controller")){
```

```
                        moduleMapping.addModuleToDevice("threat-
analysis", device.getName());

moduleMapping.addModuleToDevice("security-coordinator",
device.getName());

moduleMapping.addModuleToDevice("emergency-response",
device.getName());
                        if(ENCRYPTION_ENABLED){

moduleMapping.addModuleToDevice("data-encryption",
device.getName());
                        }
                    }
                }
            }

            controller    =    new    Controller("security-master-
controller", fogDevices, sensors, actuators);
            controller.submitApplication(application,
                    (!DISTRIBUTED_PROCESSING)?(new
ModulePlacementMapping(fogDevices, application, moduleMapping))
                            :(new
ModulePlacementEdgewards(fogDevices,     sensors,     actuators,
application, moduleMapping)));


TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInst
ance().getTimeInMillis());
            CloudSim.startSimulation();
            CloudSim.stopSimulation();

            Log.printLine("Security      Enhancement      System
finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("Security system error occurred");
```

```
        }
    }

    private static void createSecurityFogDevices(int userId,
String appId) {
        FogDevice cloud = createFogDevice("cloud", 56000, 64000,
100, 10000, 0, 0.01, 18*103, 18*85.5);
        cloud.setParentId(-1);
        fogDevices.add(cloud);

        FogDevice securityGateway = createFogDevice("security-
gateway", 3500, 8000, 10000, 10000, 1, 0.0, 125.5, 95.2);
        securityGateway.setParentId(cloud.getId());
        securityGateway.setUplinkLatency(80);
        fogDevices.add(securityGateway);

        for(int i = 0; i < numOfZones; i++){
            addSecurityZone(i+"",         userId,         appId,
securityGateway.getId());
        }
    }

    private static FogDevice addSecurityZone(String zoneId, int
userId, String appId, int parentId){
        FogDevice   zoneController   =   createFogDevice("zone-
controller-"+zoneId, 3200, 6000, 1500, 10000, 2, 0.0, 115.8,
88.7);
        fogDevices.add(zoneController);
        zoneController.setUplinkLatency(3);
        zoneController.setParentId(parentId);

        for(int i = 0; i < numOfMotionSensorsPerZone; i++){
            String sensorId = zoneId+"-motion-"+i;
            FogDevice motionSensor = addMotionSensor(sensorId,
userId, appId, zoneController.getId());
            motionSensor.setUplinkLatency(1);
            fogDevices.add(motionSensor);
```

```
        }

        for(int i = 0; i < numOfDoorSensorsPerZone; i++){
            String sensorId = zoneId+"-door-"+i;
            FogDevice doorSensor = addDoorSensor(sensorId,
userId, appId, zoneController.getId());
            doorSensor.setUplinkLatency(1);
            fogDevices.add(doorSensor);
        }

        for(int i = 0; i < numOfWindowSensorsPerZone; i++){
            String sensorId = zoneId+"-window-"+i;
            FogDevice windowSensor = addWindowSensor(sensorId,
userId, appId, zoneController.getId());
            windowSensor.setUplinkLatency(1);
            fogDevices.add(windowSensor);
        }

        FogDevice biometricScanner = addBiometricScanner(zoneId,
userId, appId, zoneController.getId());
        biometricScanner.setUplinkLatency(2);
        fogDevices.add(biometricScanner);

        return zoneController;
    }

    private static FogDevice addMotionSensor(String id, int
userId, String appId, int parentId){
        FogDevice motionSensor = createFogDevice("motion-sensor-
"+id, 800, 1500, 10000, 10000, 3, 0, 45.2, 35.8);
        motionSensor.setParentId(parentId);

        Sensor sensor = new Sensor("motion-"+id,
"MOTION_DETECTOR", userId, appId, new
DeterministicDistribution(MOTION_SENSOR_INTERVAL));
        sensors.add(sensor);
```

```
        Actuator alarm = new Actuator("motion-alarm-"+id, userId,
appId, "MOTION_ALARM");
        actuators.add(alarm);

        sensor.setGatewayDeviceId(motionSensor.getId());
        sensor.setLatency(0.5);

        alarm.setGatewayDeviceId(parentId);
        alarm.setLatency(0.8);

        return motionSensor;
    }

    private static FogDevice addDoorSensor(String id, int userId,
String appId, int parentId){
        FogDevice doorSensor = createFogDevice("door-sensor-"+id,
600, 1200, 10000, 10000, 3, 0, 38.5, 28.9);
        doorSensor.setParentId(parentId);

        Sensor sensor = new Sensor("door-"+id, "DOOR_SENSOR",
userId,                         appId,                         new
DeterministicDistribution(DOOR_SENSOR_INTERVAL));
        sensors.add(sensor);

        Actuator lock = new Actuator("smart-lock-"+id, userId,
appId, "SMART_LOCK");
        actuators.add(lock);

        Actuator doorAlarm = new Actuator("door-alarm-"+id,
userId, appId, "DOOR_ALARM");
        actuators.add(doorAlarm);

        sensor.setGatewayDeviceId(doorSensor.getId());
        sensor.setLatency(0.3);

        lock.setGatewayDeviceId(parentId);
        lock.setLatency(0.5);
```

```
        doorAlarm.setGatewayDeviceId(parentId);
        doorAlarm.setLatency(0.6);

        return doorSensor;
    }

    private static FogDevice addWindowSensor(String id, int
userId, String appId, int parentId){
        FogDevice windowSensor = createFogDevice("window-sensor-
"+id, 500, 1000, 10000, 10000, 3, 0, 32.1, 25.4);
        windowSensor.setParentId(parentId);

        Sensor sensor = new Sensor("window-"+id, "WINDOW_SENSOR",
userId,                         appId,                         new
DeterministicDistribution(WINDOW_SENSOR_INTERVAL));
        sensors.add(sensor);

        Actuator windowAlarm = new Actuator("window-alarm-"+id,
userId, appId, "WINDOW_ALARM");
        actuators.add(windowAlarm);

        sensor.setGatewayDeviceId(windowSensor.getId());
        sensor.setLatency(0.4);

        windowAlarm.setGatewayDeviceId(parentId);
        windowAlarm.setLatency(0.7);

        return windowSensor;
    }

    private static FogDevice addBiometricScanner(String zoneId,
int userId, String appId, int parentId){
        FogDevice   biometricScanner   =   createFogDevice("bio-
scanner-"+zoneId, 1200, 2000, 10000, 10000, 3, 0, 62.8, 48.3);
        biometricScanner.setParentId(parentId);
```

```
        Sensor fingerprint = new Sensor("fingerprint-"+zoneId,
"FINGERPRINT_SCANNER",         userId,         appId,         new
DeterministicDistribution(BIOMETRIC_SCAN_TIME));
        sensors.add(fingerprint);

        Sensor faceRecognition = new Sensor("face-"+zoneId,
"FACE_RECOGNITION",         userId,         appId,         new
DeterministicDistribution(BIOMETRIC_SCAN_TIME));
        sensors.add(faceRecognition);

        Actuator accessGrant = new Actuator("access-grant-
"+zoneId, userId, appId, "ACCESS_CONTROL");
        actuators.add(accessGrant);

        Actuator securityAlert = new Actuator("security-alert-
"+zoneId, userId, appId, "SECURITY_ALERT");
        actuators.add(securityAlert);


fingerprint.setGatewayDeviceId(biometricScanner.getId());
        fingerprint.setLatency(0.2);


faceRecognition.setGatewayDeviceId(biometricScanner.getId());
        faceRecognition.setLatency(0.3);

        accessGrant.setGatewayDeviceId(parentId);
        accessGrant.setLatency(0.4);

        securityAlert.setGatewayDeviceId(parentId);
        securityAlert.setLatency(0.5);

        return biometricScanner;
    }
```

```
    private static FogDevice createFogDevice(String nodeName, long mips, int ram, long upBw, long downBw, int level, double ratePerMips, double busyPower, double idlePower) {
        List<Pe> peList = new ArrayList<Pe>();
        peList.add(new Pe(0, new PeProvisionerOverbooking(mips)));

        int hostId = FogUtils.generateEntityId();
        long storage = 1000000;
        int bw = 10000;

        PowerHost host = new PowerHost(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerOverbooking(bw),
                storage,
                peList,
                new StreamOperatorScheduler(peList),
                new FogLinearPowerModel(busyPower, idlePower)
        );

        List<Host> hostList = new ArrayList<Host>();
        hostList.add(host);
        String arch = "x86";
        String os = "Linux";
        String vmm = "Xen";
        double time_zone = 10.0;
        double cost = 3.0;
        double costPerMem = 0.05;
        double costPerStorage = 0.001;
        double costPerBw = 0.0;
        LinkedList<Storage> storageList = new LinkedList<Storage>();

        FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
```

```
                arch, os, vmm, host, time_zone, cost, costPerMem,
costPerStorage, costPerBw);

        FogDevice fogdevice = null;
        try {
            fogdevice = new FogDevice(nodeName, characteristics,
                    new     AppModuleAllocationPolicy(hostList),
storageList, 10, upBw, downBw, 0, ratePerMips);
        } catch (Exception e) {
            e.printStackTrace();
        }
        fogdevice.setLevel(level);
        return fogdevice;
    }

    @SuppressWarnings({"serial"})
    private static Application createSecurityApplication(String
appId, int userId){
        Application                application                =
Application.createApplication(appId, userId);

        application.addAppModule("motion-detection", 15);
        application.addAppModule("access-control", 12);
        application.addAppModule("perimeter-monitor", 10);
        application.addAppModule("biometric-auth", 20);
        application.addAppModule("threat-analysis", 25);
        application.addAppModule("security-coordinator", 18);
        application.addAppModule("emergency-response", 15);
        if(ENCRYPTION_ENABLED){
            application.addAppModule("data-encryption", 8);
        }

        application.addAppEdge("MOTION_DETECTOR",       "motion-
detection", 200, 1000, "MOTION_DATA", Tuple.UP, AppEdge.SENSOR);
        application.addAppEdge("DOOR_SENSOR",  "access-control",
150, 800, "DOOR_DATA", Tuple.UP, AppEdge.SENSOR);
```

```
        application.addAppEdge("WINDOW_SENSOR",      "perimeter-
monitor", 180, 900, "WINDOW_DATA", Tuple.UP, AppEdge.SENSOR);
        application.addAppEdge("FINGERPRINT_SCANNER",
"biometric-auth",  500,  2000,  "FINGERPRINT_DATA",  Tuple.UP,
AppEdge.SENSOR);
        application.addAppEdge("FACE_RECOGNITION",   "biometric-
auth", 800, 3000, "FACE_DATA", Tuple.UP, AppEdge.SENSOR);

        application.addAppEdge("motion-detection",      "threat-
analysis", 300, 1500, "MOTION_EVENTS", Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("access-control",        "threat-
analysis", 250, 1200, "ACCESS_EVENTS", Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("perimeter-monitor",     "threat-
analysis",    280,    1400,    "PERIMETER_EVENTS",    Tuple.UP,
AppEdge.MODULE);
        application.addAppEdge("biometric-auth",      "security-
coordinator",    400,    1800,    "AUTH_RESULTS",    Tuple.UP,
AppEdge.MODULE);

        application.addAppEdge("threat-analysis",     "security-
coordinator",  350,  1600,  "THREAT_ASSESSMENT",  Tuple.UP,
AppEdge.MODULE);
        application.addAppEdge("security-coordinator",
"emergency-response", 300, 1400, "SECURITY_DECISIONS", Tuple.UP,
AppEdge.MODULE);

        if(ENCRYPTION_ENABLED){
            application.addAppEdge("security-coordinator",
"data-encryption",  200,  1000,  "SENSITIVE_DATA",  Tuple.UP,
AppEdge.MODULE);
        }

        application.addAppEdge("motion-detection",
"MOTION_ALARM",  100,  50,  20,  "MOTION_ALERT",  Tuple.DOWN,
AppEdge.ACTUATOR);
        application.addAppEdge("access-control",   "SMART_LOCK",
80, 40, 15, "LOCK_COMMAND", Tuple.DOWN, AppEdge.ACTUATOR);
```

```
        application.addAppEdge("access-control",    "DOOR_ALARM",
90, 45, 18, "DOOR_ALERT", Tuple.DOWN, AppEdge.ACTUATOR);
        application.addAppEdge("perimeter-monitor",
"WINDOW_ALARM",   95,   48,   19,   "WINDOW_ALERT",   Tuple.DOWN,
AppEdge.ACTUATOR);
        application.addAppEdge("biometric-auth",
"ACCESS_CONTROL", 120, 60, 25, "ACCESS_DECISION", Tuple.DOWN,
AppEdge.ACTUATOR);
        application.addAppEdge("emergency-response",
"SECURITY_ALERT",  150,  75,  30,  "EMERGENCY_ALERT",  Tuple.UP,
AppEdge.ACTUATOR);

        application.addTupleMapping("motion-detection",
"MOTION_DATA", "MOTION_EVENTS", new FractionalSelectivity(0.8));
        application.addTupleMapping("motion-detection",
"MOTION_DATA", "MOTION_ALERT", new FractionalSelectivity(0.3));
        application.addTupleMapping("access-control",
"DOOR_DATA", "ACCESS_EVENTS", new FractionalSelectivity(0.9));
        application.addTupleMapping("access-control",
"DOOR_DATA", "LOCK_COMMAND", new FractionalSelectivity(0.7));
        application.addTupleMapping("access-control",
"DOOR_DATA", "DOOR_ALERT", new FractionalSelectivity(0.2));
        application.addTupleMapping("perimeter-monitor",
"WINDOW_DATA",              "PERIMETER_EVENTS",              new
FractionalSelectivity(0.85));
        application.addTupleMapping("perimeter-monitor",
"WINDOW_DATA", "WINDOW_ALERT", new FractionalSelectivity(0.25));
        application.addTupleMapping("biometric-auth",
"FINGERPRINT_DATA",             "AUTH_RESULTS",             new
FractionalSelectivity(0.95));
        application.addTupleMapping("biometric-auth",
"FACE_DATA", "AUTH_RESULTS", new FractionalSelectivity(0.92));
        application.addTupleMapping("biometric-auth",
"AUTH_RESULTS",              "ACCESS_DECISION",              new
FractionalSelectivity(1.0));
```

```
        application.addTupleMapping("threat-analysis",
"MOTION_EVENTS",            "THREAT_ASSESSMENT",            new
FractionalSelectivity(0.6));
        application.addTupleMapping("threat-analysis",
"ACCESS_EVENTS",            "THREAT_ASSESSMENT",            new
FractionalSelectivity(0.7));
        application.addTupleMapping("threat-analysis",
"PERIMETER_EVENTS",          "THREAT_ASSESSMENT",          new
FractionalSelectivity(0.65));
        application.addTupleMapping("security-coordinator",
"AUTH_RESULTS",            "SECURITY_DECISIONS",            new
FractionalSelectivity(0.8));
        application.addTupleMapping("security-coordinator",
"THREAT_ASSESSMENT",          "SECURITY_DECISIONS",          new
FractionalSelectivity(0.75));
        application.addTupleMapping("emergency-response",
"SECURITY_DECISIONS",          "EMERGENCY_ALERT",          new
FractionalSelectivity(0.4));

        if(ENCRYPTION_ENABLED){
            application.addTupleMapping("security-coordinator",
"SECURITY_DECISIONS",           "SENSITIVE_DATA",           new
FractionalSelectivity(0.5));
        }

        final  AppLoop  mainSecurityLoop  =  new  AppLoop(new
ArrayList<String>(){{
            add("MOTION_DETECTOR");     add("motion-detection");
add("threat-analysis");
            add("security-coordinator");         add("emergency-
response"); add("SECURITY_ALERT");
        }});

        final  AppLoop  accessControlLoop  =  new  AppLoop(new
ArrayList<String>(){{
            add("DOOR_SENSOR");          add("access-control");
add("SMART_LOCK");
```

```
        }});

        final AppLoop biometricLoop = new AppLoop(new
ArrayList<String>(){{
            add("FINGERPRINT_SCANNER");   add("biometric-auth");
add("ACCESS_CONTROL");
        }});

        final AppLoop perimeterLoop = new AppLoop(new
ArrayList<String>(){{
            add("WINDOW_SENSOR");      add("perimeter-monitor");
add("WINDOW_ALARM");
        }});

        List<AppLoop> loops = new ArrayList<AppLoop>(){{
            add(mainSecurityLoop);       add(accessControlLoop);
add(biometricLoop); add(perimeterLoop);
        }};

        application.setLoops(loops);
        return application;
    }
}
```