Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти

на тему

“Програмне забезпечення системи міжз’єднань центрів обробки даних”

КбПЗ_2025

Виконав здобувач вищої освіти
IV курсу, групи КІ-22-МБ
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Дроздов С.І.
«____» _____ 2025 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Якименко Н.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Дроздову Сергію Ігоревичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи міжз’єднань центрів обробки даних*

2. Керівник роботи *Якименко Наталія Миколаївна, канд. фіз.-мат. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 48-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи міжз’єднань центрів обробки даних*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

КПІ-2025

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

КбПЗ_2025

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Якименко Н.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Дроздов С.І.
(прізвище та ініціали)

АНОТАЦІЯ

Дроздов С.І. Програмне забезпечення системи міжз'єднань центрів обробки даних. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи міжз'єднань центрів обробки даних.

Метою розробки є програмне забезпечення системи міжз'єднань центрів обробки даних.

Результат роботи – програмна реалізація системи міжз'єднань центрів обробки даних.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, міжз'єднання центрів обробки даних

КОПЗ_2025

ABSTRACT

Drozdov S.I. Software for the data center interconnection system. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the data center interconnection system.

The purpose of the development is software for the data center interconnection system.

The result of the work is the software implementation of the data center interconnection system.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with the software are provided.

The program can be used on a PC with Windows 10/11.

The program was developed in the Python environment.

Keywords: computer engineering, data center interconnection

КОПЗ_2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи..... 5
1.2 Область застосування..... 5
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 7
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти..... 7
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 21
2.3 Розгорнута постановка завдання 23
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 25
3.1 Опис функціонування системи 25
3.2 Розробка структурної схеми..... 28
3.3 Розробка функціональної схеми 33
3.4 Розробка діаграми процесів..... 38
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 40
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 40
4.2 Захист розробленого програмного забезпечення..... 51
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 56
6 ОСНОВНІ ВИСНОВКИ..... 62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64

КОПЗ – 2025

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

КВ	–	коефіцієнт варіації
КЗ	–	канал зв'язку
ПС	–	програмна середа
СеМО	–	експонентна мережа масового обслуговування
СМО	–	система масового обслуговування
СПД	–	система передачі даних

КбПЗ_2025

ВСТУП

Актуальність теми. Хоча завдання по забезпеченню міжз'єднань центрів обробки даних (DCI) з'явилася разом із самими ЦОДами, ринок відповідних рішень став бурхливо розвиватися лише в останні роки – у міру розширення використання хмарних додатків, контенту й сервісів.

Наша пропозиція дозволить операторам мереж і ЦОДів прискорити розгортання, знизити експлуатаційні витрати, а також підвищити гнучкість і ефективність відповідно до вимог парадигми Web Scale.

Ключове рішення для DCI, реалізує єдину платформу для комутації, спектрального мультиплексування DWDM, передачі даних і фотоники. У ній втілений цілий ряд успішних технологічних інновацій, зокрема когерентна обробка оптичних сигналів. Платформа представлена безліччю форм-факторів, створених з урахуванням різних потреб, і забезпечує транспорт DWDM 10G/40G/100G на будь-якій відстані (від міських до підводних мереж).

Програмувальна багатотерабітна когерентна платформа пакетної DWDM-передачі забезпечує економічну агрегацію й комутацію з підтримкою пакетної передачі, при цьому завдяки підтримці оптики 100G DWDM можливо міжз'єднання сучасних ЦОДів, розташованих один від іншого на відстані до 1000 км. За рахунок дворазового зниження вимог до живлення й простору значно скорочуються експлуатаційні витрати, пов'язані з енергозабезпеченням і орендою площі. А автоматичне виділення ресурсів (ZTP) дозволяє відмовитися від виконання операцій вручну при розгортанні системи.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи міжз'єднань центрів обробки даних.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем міжз'єднань центрів обробки даних.

КОПЗ-2025

– Дослідження системи міжз’єднань центрів обробки даних.

– Програмна реалізація системи міжз’єднань центрів обробки даних.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі міжз’єднань центрів обробки даних.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи міжз’єднань центрів обробки даних, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

КбПЗ_2025

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Спеціально для завдань DCI масштабу Web Scale розроблена нарощувана система забезпечення міжз'єднань. Вона опирається на дві основні ідеї – рентабельність WaveLogic і IT-функціонал Web Scale. Waveserver розрахована на модульне розгортання в стійках і стеках, при цьому один компактний модуль форм-фактора 1RU забезпечує високу, до 400G, пропускну здатність із боку клієнта й ще 400G для зовнішнього підключення.

Система Waveserver дозволяє розширити IT-функціонала серверів до рівня мережних функцій, а її ключовими характеристиками є програмуємість і відкритість: користувачі й розроблювачі можуть створювати, тестувати і набудовувати додатка за допомогою Emulation Cloud – відкритого середовища розробки й проведення випробувань.

1.2 Область застосування

Областю застосування є Центри обробки даних (ЦОД, Дата-центри). Дата центр – усе ще рідке поняття для українських компаній. І якщо серед IT-спеціалістів його роль питань не викликає, то починаючи від менеджерів до власників компаній інших областей виникає маса питань. Питання дуже різномасті – від того, що таке Дата-центр, як улаштований процес його роботи й властиво навіщо він потрібний.

КПЗ_2025

Так що ж це таке Дата-центр? Звичайно можна звернутися до Вікіпедії й прочитати, що: Дата-центр (від англ. data center), або центр (зберігання й) обробки даних (ЦОД/ЦХОД) – це спеціалізований будинок для розміщення (хостинга) серверного й мережного встаткування. Це ясно для згаданих вище все тих же IT-спеціалістів. А як же бути таким людям, які не входять у касту «інопланетян», як же пояснити їм?

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи міжз'єднань центрів обробки даних, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

КбПЗ_2025

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Потреба в багатоволоконій з’єднуваності з низькими втратами в сучасному центрі обробки даних

На даному етапі менеджери ЦОД найбільш стурбовані, серед іншого, питанням ресурсів внесених оптичних втрат, особливо в сучасних об'ємних середовищах віртуалізованих серверів з магістральними міжкомутаторними розгортаннями на більші відстані 40- і 100-гігабітних мереж Ethernet (Gb) для створення мереж зв'язку й мереж зберігання даних (SAN). Ресурси втрат варто враховувати на ранніх етапах конструювання будь-якого центра обробки даних; запобігання перевищення ресурсу втрат істотно важливо для належної передачі оптичних сигналів даних між комутаторами без високої частоти появи помилкових бітів і погіршення показників продуктивності. Керрі Хігбі (CarrieHigbie), директор по глобальних рішеннях ЦОД і експерт компанії Siemon по послугах у сфері мережної інфраструктури, дає рекомендації відносно кількості й типу з'єднань із низькими втратами, що підходять для широкого діапазону застосувань і конфігурацій.

Одним із ключових і найважливіших факторів, які привели до росту заклопотаності проблемою балансу втрат волокна, є перехід на швидкість передачі даних 10, 40 і 100 Гб/с у мережах Ethernet, а також з 8 Гб/с на 16 Гб/с і 32 Гб/с у мережах зберігання даних з оптоволокономи каналами. У міру зростання пропускної здатності й швидкості передачі дані вимоги до внесених втрат стають більше строгими, чим коли-небудь,. Якщо довжина й тип оптоволоконого кабелю, а також кількість з'єднувачів і зростків сприяють

КОПЗ_2025

втратам на лінії, менеджерам ЦОД необхідно вираховувати кожну точку й сегмент з'єднання у своїх оптоволоконих каналах.

Динамічні втрати архітектур

Традиційні **трирівневі архітектури з комутатором рівня 3** широко застосовуються в центрах обробки даних; з обліком багатокомутаторних рівнів і швидкостей оптоволоконих магістралей 10 гігабіт у секунду (Гб/с) відстань і швидкість передачі даних між комутаторами залишилися досить малими для того, щоб більшість центрів обробки даних зберігали два або більше з'єднувачі без перевищення ресурсів оптичних втрат лінії. Проте, традиційні трирівневі архітектури вже не є ідеальним рішенням для великих віртуалізованих центрів обробки даних.

Багато центрів обробки даних переходять на **архітектуру з комутаційними матрицями, що** використовує тільки два комутаторних рівні з меншими міжкомутаторними стрибками. Комутаційні матриці забезпечують більше низьку затримку й більшу пропускну здатність між будь-якими двома точками завдяки перевагам фізичних з'єднань кабельних укладань (порт до порту) комутаторів у порівнянні з висхідними каналами зв'язку від комутатора нижнього рівня до комутатора більше високого рівня.

Комутаційні матриці «стовщеного дерева», також називані архітектурою «leaf and spine», є одними з найпоширеніших комутаційних матриць, використовуваних у сучасних віртуалізованих центрах обробки даних. Архітектура «стовщеного дерева» складається з міжкомпонентних комутаторів («spine»), розміщених у головній зоні розподілу (MDA), і комутаторів доступу («leaf»), розміщених у горизонтальній зоні розподілу (HDA), або EDA, кожний з яких приєднується або підключається висхідним образом до кожному міжмережному комутатору в мережі, звичайно за допомогою оптоволокна.

Хоча архітектура «стовщеного дерева» використовує доступ до заснованого на стандартах оптоволоконим кабельним системам для встановлення великої кількості активних з'єднань між меншим числом комутаторів, такі нові

КПЗ-2025

конструктивні рішення центрів обробки даних найчастіше приводять до збільшення відстаней між міжмережними комутаторами й комутаторами доступу.

Для забезпечення керованості й гнучкості, зручності монтажу й модернізації, а також обмеження доступу до ключових комутаторів багато керівників центрів обробки даних мають потребу в прокладанні численних сполучених пар для підтримки **пунктів реалізації або зручних зон оптоволоконої комутації** (крос-з'єднання).

Зручні зони комутації включають використання оптоволоконих комутаційних панелей, які дзеркально відображають порти міжмережних комутаторів і за допомогою постійних або фіксованих з'єднань підключаються до оптоволоконим комутаційних панелей, що дзеркально відображають порти комутаторів доступу. Такі панелі можна поміщати в окремих шафах, що забезпечує захист і недоторканність комутаторів. Вони також забезпечують спрощені переміщення, додавання й зміни завдяки формуванню конфігурації «кожний з усіма», у якій будь-який комутаційний порт можна приєднати до будь-якого іншого комутаційного порту, просто змінивши положення з'єднань оптоволоконих перемичок у зоні комутації.

Як ідеальне рішення для більших ЦОД або при розподілі оптоволокна на кілька функціональних зон або областей використання крос-з'єднань між міжмережними й/або комутаторами доступу може також дозволити одноразовий монтаж постійної кабельної системи з більшою кількістю волокон від MDA до HDA. Це дозволяє використовувати магістральні оптоволоконі кабельні системи для різних цілей (створення мереж або систем зберігання даних) без безлічі змін, а також спрощувати процес додавання нових комутаторів доступу й устаткування в ЦОД. Використання таких коштовних крос-з'єднань створює додаткові комутаційні порти з подальшими втратами в оптоволоконому каналі. Надалі стандартні значення внесених втрат MPO/MTP можуть створити для керівників ЦОД ризик перевищення ресурсів оптичних втрат лінії, найчастіше перешкоджаючи застосуванню крос-з'єднань і вимагаючи подальшого

КПЗ-2025

використання довгих сегментів оптоволокна, що істотно обмежує гнучкість і ускладнює внесення змін і модернізацію.

Принципи з'єднань для зниження втрат

З'єднувачі MPO або MTP швидко входять у звичайну практику для міжкомутаторних з'єднань за рахунок їхніх претермінованих переваг «plug and play» і простоти нарощування швидкостей з 10 до 40 і 100 Гбіт. На жаль, внесені втрати типового модуля MPO/MTP можуть перешкоджати наявності більш ніж двох сполучених з'єднань в оптоволоконому каналі, що істотно обмежує гнучкість конструкції й керування центром обробки даних. Як удосконалена альтернатива з'єднувачі MPO/MTP з низькими втратами, на відміну від з'єднувачів зі стандартними втратами, більш ефективно підтримують гнучкість множинних сполучених з'єднань у широкому діапазоні відстаней і конфігурацій поряд з дотриманням ресурсу втрат.

З'єднувачі MTP з низькими втратами можуть зменшити втрати до 0,20 дБ, а з'єднувачі LC і SC з низькими втратами – до 0,15 дБ (звичайно 0,1 дБ). Для 10-гігабітних мереж Ethernet оптоволоконі перемички LC з низькими втратами забезпечують втрати 0,15 дб (звичайно 0,1 дб), тоді як модулі MTP-LC або MTP-SC «plug and play» з низькими втратами забезпечують втрати 0,35 дб (звичайно 0,25 дб). Для 40- і 100-гігабітних мереж Ethernet пластини для наскрізних адаптерів MTP-MTP з низькими втратами й оптоволоконі перемички MTP забезпечують втрати 0,2 дб.

Такі знижені показники втрат дають керівникам ЦОД можливість розгорнути більше комутаційних портів, дозволяючи використовувати пункти реалізації або крос-з'єднання, які істотно підвищують гнучкість конфігурування.

Як показано в таблиці 1, застосування з'єднань із низькими втратами дозволяє використовувати чотири модулі MTP-LC в 10-гігабітному каналі Ethernet OM3 або OM4 у порівнянні всього із двома при з'єднаннях зі стандартними втратами. З'єднання з низькими втратами дозволяють використовувати вісім підключень MTP-MTP в 100-метровому 40/ 100-

КОПЗ-2025

гігабітному каналі Ethernet по OM3 у порівнянні із усього чотирма підключеннями зі стандартними втратами, а також п'ятьома підключеннями в 150-метровому 40/ 100-гігабітном каналі Ethernet по оптоволокну OM4 у порівнянні із усього двома підключеннями зі стандартними втратами.

Для розгортання крос-з'єднань між міжмережними комутаторами й комутаторами доступу потрібен мінімум чотири підключення залежно від конфігурації. Тому крос-з'єднання в оптичному каналі значної довжини просто неможливі без з'єднань із низькими втратами.

Крім створення більшої кількості підключень в 10-, 40- і 100-гігабітних каналах Ethernet, з'єднання з низькими втратами забезпечують такої ж переваги для розгортання оптоволоконих каналів у мережах зберігання даних. Наприклад, прокладання 150-метрового оптоволоконого каналу на 8 Гбіт/с (GFC) дозволяє використовувати максимум чотири модулі MTP-LC з низькими втратами, тоді як компоненти зі стандартними втратами дозволяють використовувати тільки два модулі. Використання з'єднань із низькими втратами для розгортання крос-з'єднань у такий спосіб спрощує зміна підключень сервера від мережного висхідного порту до порту системи зберігання даних і навпаки шляхом простої зміни перемички в кросі-з'єднанні.

При сучасній згладженій комутаційній архітектурі й зростаючих витратах внаслідок внесених втрат оптичних ліній оптоволоконі з'єднання з низькими втратами забезпечують більше число точок підключення в мережах Ethernet і оптоволоконих каналах у ЦОД.

При необхідності перегляду витрат на втрати на ранніх етапах конструювання яких-небудь центрів обробки даних керівники ЦОД можуть використовувати оптоволоконі з'єднання з низькими втратами для забезпечення більшої кількості підключень у мережах 10, 40 і 100 Gb або в системах зберігання даних 8, 16 і 32 GFC. З'єднання з низькими втратами можуть відповідати вимогам зниження витрат на втрати й забезпечувати гнучкість у широкому діапазоні відстаней для конфігурацій з перспективою на майбутнє.

КБПЗ-2025

Пошук і усунення мережних неполадок на встаткуванні Cisco UCS Fabric Interconnect

При аналізі підключення Cisco UCS до мережі важливо розуміти, яким образом пакети передаються безпосередньо між елементами UCS. Через специфічне поводження Cisco UCS Fabric Interconnect у порівнянні зі звичайними комутаторами й з Cisco Nexus пошук можливих помилок конфігурації й мережних неполадок буде трохи відрізнятися від стандартного набору процедур на класичному встаткуванні рівня доступу.

Ключові терміни в рамках даної роботи:

– Fabric Interconnect (FI) – комутатори, використовувані одночасно як мережні пристрої при передачі LAN і SAN трафіку, так і як засіб керування мережними й серверними налаштуваннями Cisco UCS за допомогою Cisco UCS Manager. Комутатори FI не обмінюються трафіком серверів прямо – з'єднання між FI необхідні для синхронізації параметрів Active і Subordinate процесів UCS Manager.

– I/O Module (IOM/FEX) – модулі розширення, установлювані в UCS 5108 chassis для забезпечення зв’язності між мережними адаптерами серверів і FI. Два модулі IOM підключаються без перетинання до двох FI для створення двох незалежних шляхів для проходження трафіку від мережних адаптерів.

– vNIC – віртуальні мережні адаптери, презентуємі операційній системі, установленої на сервері. Кількість даних адаптерів буде визначатися типом фізичного мережного адаптера в сервері й кількістю фізичних з'єднань між модулями IOM і комутаторами FI.

– vEth – віртуальний Ethernet інтерфейс на FI, що створюється для кожного vNIC (на одному із двох FI, A або B, якщо не була обрана опція Fabric Failover, і на обох, якщо опція застосована). Фізичні інтерфейси FI, звернені убік модулів IOM виступають як високошвидкісні канали прийому трафіку множини карт vNIC. FI "передає" трафік на відповідний vEth після прочитання мітки VNTag, що встановлюється у фрейми при передачі між IOM і FI.

КПЗ-2025

Cisco UCS Fabric Interconnect є сполучною ланкою між серверами й іншим мережним устаткуванням центра обробки даних, а також систем зберігання даних у деяких варіантах підключення. Однак, маючи зовнішню подібність із комутаторами Cisco Nexus, FI за замовчуванням передає дані між своїми інтерфейсами не як традиційний комутатор. Такий режим роботи одержав назву End-host mode (EHM), і його рекомендується використовувати в більшості варіантів підключення Cisco UCS. У режимі EHM передача фреймів від мережних адаптерів серверів через FI на вихідні інтерфейси йде шляхом процедури, називаної піннінг (Pinning). Кожному віртуальному мережному адаптеру при створенні призначається шлях через один із двох FI (Fabric A або Fabric B). Трафік даного vNIC направляється на відповідному обраному шляху модуль IOM, від якого по одному з фізичних каналів потік досягає FI.

Для пересилання фрейму на відповідний Uplink інтерфейс комутатор FI звертається до політики піннінгу для карти vNIC, від якої даний трафік надійшов. У правилі піннінгу або заданий конкретний інтерфейс комутатора FI (Static pinning), або правило визначене як динамічне. В останньому випадку FI вибирає один з доступних Uplink інтерфейсів, на якому дозволений до передачі відповідний VLAN (Dynamic pinning). При здійсненні пересилання ніяких змін, за винятком віддалення VNTag, із фреймом не виробляється.

Вищестояще встаткування, наприклад, комутатор, що працює в класичному режимі, сприймає фрейми як вступники безпосередньо від мережних адаптерів серверів і записує їх MAC-адреси в таблицю комутації.

Відповідний трафік убік конкретного vNIC повинен повернутися на той же інтерфейс FI, через який політика піннінгу відправила оригінальні фрейми. Фрейми, що порушили це правило, дропаються.

Перевага подібного принципу пересилання фреймів полягає в тому, що EHM виключає необхідність використання STP на ділянці від FI до вищестоящих комутаторів, при цьому такі механізми, як Port-Channel, як і раніше можна використовувати. Тому що FI завжди направляють трафік у строго певні Uplink

КОПЗ_2025

інтерфейси, а також завдяки реалізованому на комутаторах FI механізму усунення повторів і зациклень фреймів (Deja Vu check), поява петель комутації виключається.

При необхідності FI може бути переведений у режим класичної комутації (Switching mode), але в такому випадку стає обов'язковим використання STP для усунення можливих петель комутації й правила піннінгу більше не застосовуються.

Пошук шляху проходження трафіку по MAC-адресі

У випадку виникнення проблем із продуктивністю UCS або при пошуку неполадок на рівні мережі однієї із частих завдань на стадії збору попередньої інформації стає пошук шляху, по якому трафік від конкретної MAC-адреси (vNIC) іде через модулі IOM і комутатори FI на вищестояще встаткування.

Припустимо ситуацію, у якій необхідно встановити місце розташування (шасі й слот із сервером) MAC-адреси 00:11:22:33:44:55, а також шлях, по якому трафік від даного MAC-адреси надходить на вищестояще встаткування. Процедура розбивається на кілька етапів, послідовність яких може варіюватися.

Необхідно визначити:

1. FI, що обслуговує дану MAC-адресу (vNIC).
2. Cluster# connect nxos a
3. Cluster-A(nxos)# show mac-address address 0011.2233.4455
4. VLAN MAC Address Type Age Port
5. -------+------------+------+-----+----
6. 100 0011.2233.4455 dynamic 10 veth901
7. Cluster# connect nxos b
8. Cluster-B(nxos)# show mac-address address 0011.2233.4455
9. Total MAC Addresses: 0
10. Місцезнаходження сервера з даною картою vNIC (номер шасі й слот із сервером)
11. Cluster-A(nxos)# show interface veth901
12. vethernet901 is up
13. Bound Interface is Ethernet1/1/1
14. У даному виводі присутній запис про деякий інтерфейс Ehternet1/1/1, номер якого насправді вказує позицію сервера (перша цифра – номер шасі, третя – номер слоту з потрібним сервером).
15. Uplink на FI, через який трафік цього vNIC залишає UCS.

КОПЗ_2025

```
16.    Cluster-A(nxos)# show pinning interface veth901
17.    SIF Interface  Sticky  Pinned Border interface
18.    ----------+ ------+--------------
19.    veth901  No  Eth1/5
20.    Інтерфейс FI, на який цей трафік надходить від модуля IOM.
21.    Cluster-A(nxos)# show fex <chassis#> detail
22.    …
23.    Fex Port  State  Fabric Port
24.    Eth1/1/1  Up  Eth1/11
25.    …
26.    Uplink інтерфейс IOM-a.
27.    Cluster-A(nxos)# show interface fex-fabric
28.      Fabric  Fabric  Fex
29.    Fex  Port  Port State  Uplink
30.    1  Eth1/11  Active  1
31.    Інтерфейс модуля IOM убік карти vNIC. Опціонально можна також
вивести діаграму підключення в GUI і CLI UCS Manager.
32.    Cluster# connect iom <chassis#>
33.    fex-1# show platform software woodside sts (для FI 62xx і IOM 220x)
34.    Вивід даної команди відобразить в CLI діаграму портів IOM залежно
від його моделі з розташуванням серверів згідно їх слотам у шасі й відповідні їм
внутрішні інтерфейси модуля IOM.
```

В умовах стабільної роботи Cisco UCS є потреба в створенні якогось статистичного викладення, ціль якої – зафіксувати нормальні показники середньої кількості трафіку, що надходить через інтерфейси FI. Частина даних цифр можна зібрати з GUI Cisco UCS Manager, але в даній статті буде звернена увага на одержання статистики з CLI з налаштуванням аварійних повідомлень і коригувальних дій при перевищенні заданих певних порогів частоти помилок на інтерфейсах.

Розглянуті команди досить тривіальні, але все-таки вимагають певної уважності й, як і будь-яке налаштування Cisco UCS, деякої попередньої роботи.

Вище була розглянута процедура пошуку інтерфейсів елементів Cisco UCS (**vNIC -> HIF -> IOM -> NIF -> FI Port -> FI -> vEth -> Uplink Port**), результати якої тепер можна використовувати в інших командах CLI, де буде потрібно вказувати номера потрібних інтерфейсів.

КПІЗ_2025

Існує набір CLI команд, що дозволяють вивчити поводження трафіку на мережних адаптерах vNIC. Для цього буде потрібно зайти в режим роботи з конкретним сервером і потім запустити MCP (Master Control Program).

```
Temp-A# connect adapter 1/1/1
adapter 1/1/1 # connect
No entry for terminal type "dumb";
using dumb terminal settings.
adapter 1/1/1 (top):1# attach-mcp
No entry for terminal type "dumb";
using dumb terminal settings.
```

Потім, використовуючи команду vnic, можна знайти відповідність між символічними іменами мережних адаптерів vNIC і логічними інтерфейсами LIF, якими вони представлені в системі.

```
adapter 1/1/1 (mcp):1# vnic
vnic id  : internal id of vnic, use for other vnic cmds
vnic name/mac  : ucsm provisioned name (-n) or mac address (-m)
vnic type  : enet=ethernet, enet_pt=dynamic ethernet, fc=fcoe
vnic h:bb:dd.f : host pci bus/device/function id
vnic state  : state of vnic
lif  : internal logical if id, use for other lif/vif cmds
lif state  : state of lif
vif uif  : bound uplink 0 or 1, =:primary, -:secondary, >:current
vif ucsm  : ucsm id for this vif
vif idx  : switch id for this vif
vif vlan  : default vlan for traffic
vif state  : state of vif
-----------------------------------
  v n i c  l i f  v i f
id  name  type  h:bb:dd.f state lif state uif  ucsm  idx vlan state
-----------------------------------
 13 vnic_1  enet  0:06:00.0 UP  2 UP  =>0  723  18  2 UP
 14 vnic_2  enet  0:07:00.0 UP  3 UP  =>1  724  20  2 UP
 15 vnic_3  fc  0:08:00.0 UP  4 UP  =>0  725  19  101 UP
 16 vnic_4  fc  0:09:00.0 UP  5 UP  =>1  726  21  102 UP
```

Мережному адаптеру vnic_1 у цьому випадку співвіднесений LIF 2 і VIF 723. Команда lifstats <lif#>дозволить тепер вивчити статистику по даному адаптері за різними показниками, таким як кількість фреймів різного типу й розміру, що пройшли через vNIC.

КПЗ_2025

```
adapter 1/1/1 (mcp):2# lifstats 2
  DELTA  TOTAL DESCRIPTION
  121331817  121331817 Tx unicast frames without error
  1171244  1171244 Tx multicast frames without error
  468170  468170 Tx broadcast frames without error
  566065244188  566065244188 Tx unicast bytes without error
  87872199  87872199 Tx multicast bytes without error
  263183943  263183943 Tx broadcast bytes without error
  49882405  49882405 Tx TSO frames
  97236693  97236693 Rx unicast frames without error
  7443238  7443238 Rx multicast frames without error
  2463357  2463357 Rx broadcast frames without error
  80214585415  80214585415 Rx unicast bytes without error
  648886997  648886997 Rx multicast bytes without error
  828622859  828622859 Rx broadcast bytes without error
  17250998  17250998 Rx frames len == 64
  13535952  13535952 Rx frames 64 < len <= 127
  58449896  58449896 Rx frames 128 <= len <= 255
  5047808  5047808 Rx frames 256 <= len <= 511
  1504377  1504377 Rx frames 512 <= len <= 1023
  85483  85483 Rx frames 1024 <= len <= 1518
  11268774  11268774 Rx frames len > 1518
  2.758Mbps  Tx rate
  406.487kbps  Rx rate
```

Стовпець DELTA показує різницю в значеннях по рядках між даним і попереднім уведенням цієї команди.

Наприкінці минулої частини була згадана команда show platform software woodside sts для комутаторів Fabric Interconnect 62xx. Дана команда дозволяє відобразити фізичну картину підключення внутрішніх і зовнішніх інтерфейсів модулів IOM.

У цей момент підключені тільки сервера blade1 – blade4 і в IOM 1 (fex-1) задіяні внутрішні інтерфейси HIF 15, 19, 23, 27 і 31 (символ ‘|' під номером HIF).

Статистику по цим HIF інтерфейсах можна побачити у виводі команди show platform software woodside rate.

Унизу таблиці представлені тільки задіяні блейд серверами перераховані вище інтерфейси HIF. Дані по кожному HIF можна уточнити, використовуючи команду show platform software woodside rmon 0 hi<hif#>

КБПЗ_2025

Тут у першу чергу варто звернути увагу на рядки **TX_PKTTOTAL RX_PACKETTOTAL** і **TX_PAUSE RX_PAUSE**. Перша, мабуть, повідомляє кількість прийнятих і відданих пакетів, друга – кількість переданих і прийнятих Pause-фреймів, велика кількість яких натякає на затори на шляху трафіку.

Для одержання інформації про частоту втрат пакетів на інтерфейсах модуля IOM, у тому числі для різних значень Class of Service можна виконати команду show platform software woodside loss.

Механіка даної команди така, що вводити її буде потрібно двічі й у виводі буде відображена статистика тільки для тих інтерфейсів, які постраждали від втрат пакетів за час між уведеннями команди.

Повідомлення про перевищення порогів припустимого рівня помилок

Для одержання повідомлень про те, що частота помилок на інтерфейсах у системі UCS виходить за межі заданих середніх або абсолютних значень, можна скористатися спеціальними граничними політиками **Statistics Threshold Policy** в GUI Cisco UCS Manager.

Дані політики можуть бути застосовані до наступних компонентів UCS:

– Інтерфейси Uplink Ethernet.

– Серверні інтерфейси, шасі й комутатори FI.

– Сервери і їхні елементи.

– Інтерфейси Fibre Channel.

Політики здатні генерувати аварійні повідомлення у випадку перевищення порогів і, в окремих випадках, на час виключати інтерфейси.

У розділі GUI **Admin->Stats Management->Collection Policies** для декількох типів елементів (adapter, chassis, fex, host, port, server) можна задати інтервал часу збору статистики (**Collection**) і інтервал часу генерації звітів (**Reporting**).

КОПЗ_2025

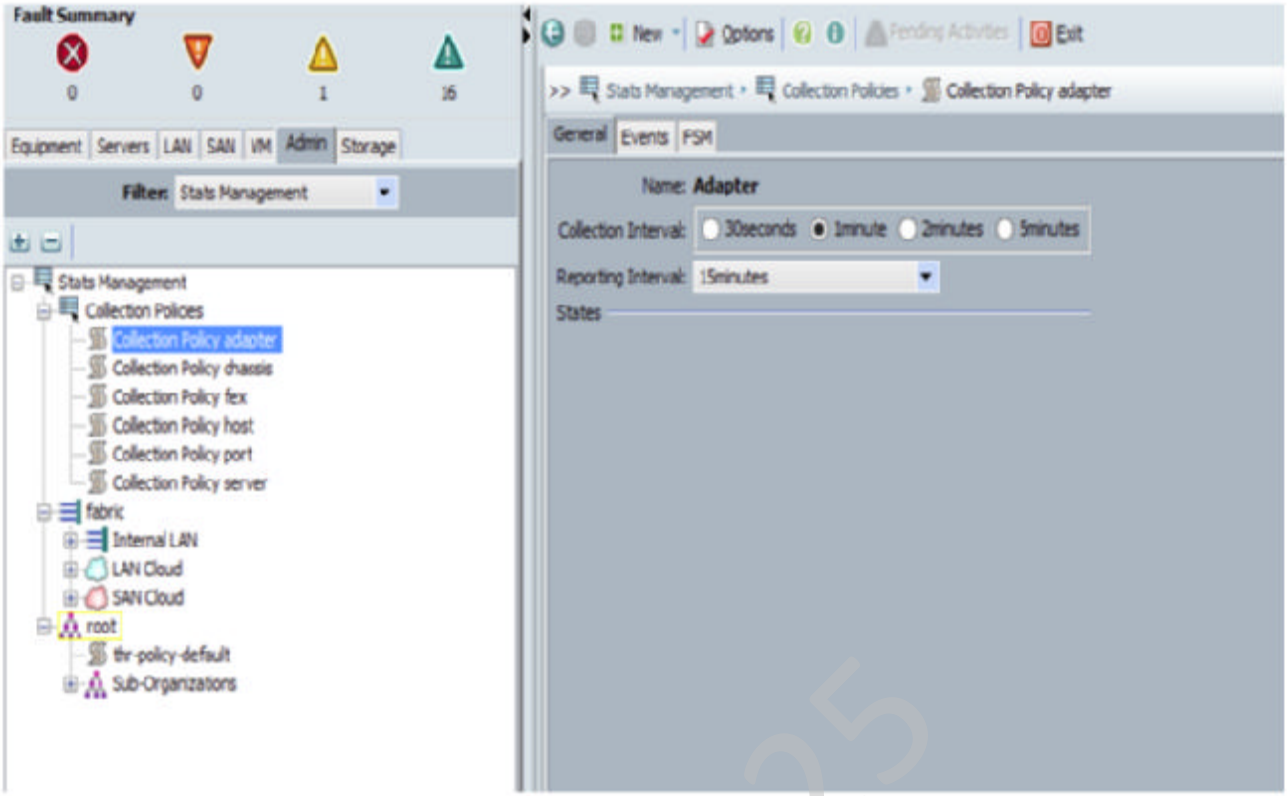


Рисунок 2.1 – Collection Policies

Приклад Threshold Policy можна побачити в розділі **LAN->Internal LAN->Threshold Policies**. У меню, що **випадає, Stats** Class можна вибрати категорію подій, для яких будуть визначатися пороги спрацьовування.

Потім у пункті **Threshold Definition** необхідно додати одне або кілька правил, що задають діапазон значень лічильника помилок і рівень аварійного повідомлення, з яким ця подія буде відображатися в Faults,Events розділі (Critical, Major і т.д.)

У меню **Property Type** можна вибрати конкретний тип помилок. Max буде означати максимальне значення за останній звітний вимірювальний інтервал (**Reporting Policy**), Min, відповідно, мінімальне значення, а Delta – саме свіже значення лічильника за поточний вимірювальний інтервал збору статистики (**Collection Interval**). Іншими словами, Delta – це саме актуальне значення лічильника. **Normal Value** показує очікуване значення лічильника й тільки перевищення його на деяку величину буде викликати поява аварійної події. У

КПЗ – 2025

прикладі на скріншоті **Normal Value** має значення **0.0**, а **Alarm Trigger (Above Normal Value)** для рівня критичності Major – **10** і7. Це означає, що якщо лічильник виявиться в значенні 0.0+10 у сучасний момент часу (через Delta), те з'явиться аварійне повідомлення. Якщо значення впаде нижче 0.0+7, подія буде переведено в статус cleared. Ненульове значення **Normal Value** має сенс, якщо як лічильник коштує, наприклад, **Ether Tx Stats Total Packets Delta Avg**, що не характеризує якусь помилку, а вказує середня кількість пакетів на передачу.

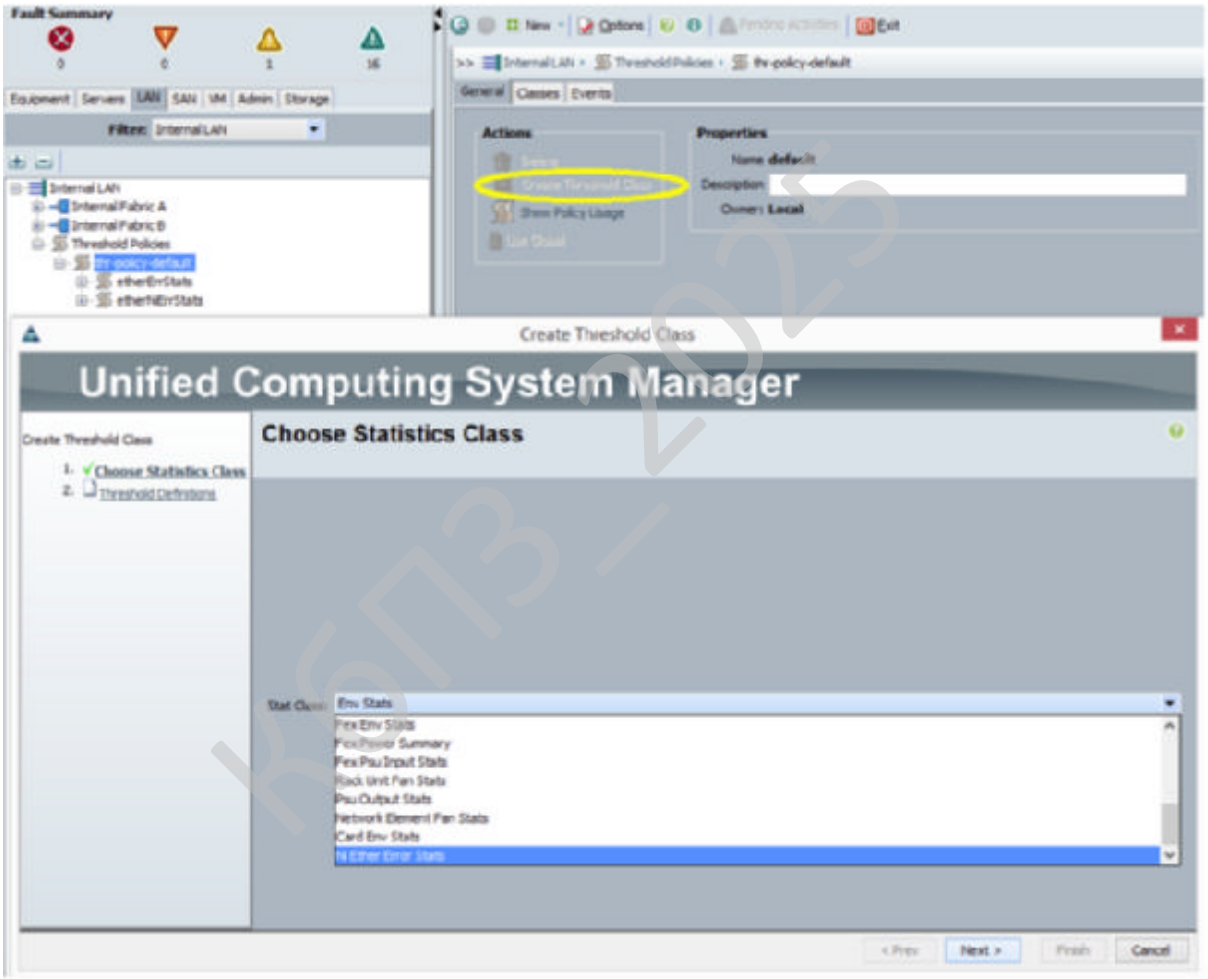


Рисунок 2.2 – Threshold Policies

Аж унизу вікна налаштування є опціональне поле, що дозволяє виключити відповідний інтерфейс FI у випадку помилки на час **Auto Recovery Time** (за замовчуванням – 10 хвилин).

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Як мова програмування обрана Python. Python – високорівнева мова програмування загального призначення з акцентом на продуктивність розроблювача й читаність коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, у тому числі структурне, об'єктно-орієнтоване, функціональне, імперативне й аспектно-орієнтоване. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточні обчислень і зручні високорівневі структури даних. Код у Python організовується у функції й класи, які можуть поєднуватися в модулі (які у свою чергу можуть бути об'єднані в пакети).

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється вільно під дуже ліберальною ліцензією, що дозволяє використовувати його без обмежень у будь-яких застосунках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM і інших. Проект PyPy пропонує реалізацію Python на самому Python, що зменшує витрати на зміни мови й постановку експериментів над новими можливостями.

Python – мова програмування, що активно розвивається, нові версії (з додаванням/зміною язикових властивостей) виходять приблизно раз у два з половиною року. Внаслідок цього й деяких інших причин на Python відсутні ANSI, ISO або інші офіційні стандарти, їхня роль виконує CPython.

Python портований і працює майже на всіх відомих платформах – від КПК до мейнфреймів. Існують порти під Microsoft Windows, практично всі варіанти UNIX (включаючи FreeBSD і Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, AS/400 і навіть OS/390, Symbian і Android.

КПЗ–2025

При цьому, на відміну від багатьох портуємих систем, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM/DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java – Jython, що дозволяє інтерпретаторові виконуватися на будь-якій системі, що підтримує Java, при цьому класи Java можуть безпосередньо використовуватися з Python й навіть бути написаними на Python. Також кілька проектів забезпечують інтеграцію із платформою Microsoft .NET, основні з яких – IronPython і Python.Net.

Python підтримує динамічну типізацію, тобто тип змінної визначається тільки під час виконання. Тому замість «присвоювання значення змінної» краще говорити про «зв'язування значення з деяким ім'ям». У Python є убудовані типи: бульові, рядки, Unicode-рядки, цілі числа довільної точності, числа із плаваючої коми, комплексні числа й деякі інші. З колекцій Python підтримує кортежі (*tuples*), списки, словники (асоціативні масиви) і, починаючи з версії 2.4, безлічі. Всі значення в Python є об'єктами, у тому числі функції, методи, модулі, класи.

Додати новий тип можна або написавши клас (class), або визначивши новий тип у модулі розширення (наприклад, написаному мовою C). Система класів підтримує спадкування (одиночне й множинне) і метапрограмування. Можливе спадкування від більшості убудованих типів і типів розширень.

Всі об'єкти діляться на посилальні й атомарні. До атомарного ставляться int, long, complex і деякі інші. При присвоюванні атомарних об'єктів копіюється їхнє значення, у той час як для посилальних копіюється тільки покажчик на об'єкт, таким чином, обидві змінні після присвоювання використовують те саме значення. Посилальні об'єкти бувають змінювані й незмінні. Наприклад, рядки й кортежі є незмінними, а списки, словники й багато інших об'єктів – змінюваними. Кортеж у Python є, по суті, незмінним списком. У багатьох випадках кортежі працюють швидше списків, тому якщо ви не плануєте змінювати послідовність, то краще використовувати саме їх.

КПЗ-2025

Мова має чіткий і послідовний синтаксис, продуману модульність й масштабованість, завдяки чому вихідний код написаних на Python програм легко читаємий.

Python – стабільна й розповсюджена мова. Він використовується в багатьох проектах і в різних якостях: як основна мова програмування або для створення розширень і інтеграції застосунків. На Python реалізоване велика кількість проектів, також він активно використовується для створення прототипів майбутніх програм. Python використовується в багатьох великих компаніях.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи міжз'єднань центрів обробки даних.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

КОПЗ-2025

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

КбПЗ_2025

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Всі переваги відкритості й програмуємості Waveserver проявляються, коли в оператора ЦОДа виникає потреба створити власний додаток. Всі необхідні для цього інструменти є в Emulation Cloud – відкритому середовищу розробки додатків.

Emulation Cloud розміщається в хмарному середовищі. надає всі необхідні для розробки додатків Web Scale інструменти, функції й підтримку (включаючи зразки коду, документацію, онлайн-керівництва й навчальні матеріали). Оператори ЦОДів можуть використовувати Emulation Cloud для створення унікальних систем, що набудовуються, що відповідають конкретним вимогам експлуатації.

Крім того, це середовище допомагає експериментувати й випробовувати нові сервісні моделі з набагато більше низьким рівнем початкових і кінцевих витрат. Нарешті, створювані з її допомогою інструменти допомагають швидше вирішувати завдання, пов'язані з обслуговуванням мереж, побудованих на основі встаткування різних постачальників, а також багаторівневих мереж.

Реалізується цілеспрямований підхід до автоматизації послуг для клієнтів фізичних і віртуальних доменів: від створення до регулювання й доставки. Однойменне ПЗ знижує експлуатаційні витрати за рахунок абстрагування від складності мережі й створення простору для інновацій, а також надає значна конкурентна перевага завдяки програмуємості послуг. Створюване ним рішення розраховане на чотири основних сценарії використання: багатодомені регулювання послуг (MDSO), регулювання NFV, відкрита мережна операційна система комерційного рівня (ONOS), а також контроль і керування SDN.

КОПЗ–2025

MDSO має на увазі комплексну автоматизацію й регулювання послуг. Використовуючи відкриті API-нтерфейси й шаблони на основі моделей, Blue Planet інтегрує сторонні контролери SDN, системи керування елементами/мережами й платформи для адміністрування й регулювання послуг, для яких використовуються як фізичні, так і віртуальні ресурси. MDSO усуває проблему розрізненості ресурсів, абстрагуючись від складності нижчележачих доменів.

Регулювання послуг NFV забезпечує масштабоване керування й регулювання віртуальних мережних функцій (VNF) і ресурсів ЦОДів. На відміну від закритих рішень NFV багатьох великих постачальників, Blue Planet застосовує відкритий і незалежний від постачальника підхід, що дозволяє мережним операторам використовувати будь-які VNF і інфраструктури NFV (NFVI), оптимізуючи визначення й створення інноваційних послуг на базі NFV.

Відкрита мережна операційна система (ONOS) – перша в галузі версія ПЗ SDN з відкритим вихідним кодом підвищеної надійності для комерційного використання. Ця програмна платформа SDN підтримується багатьма глобальними операторами, включаючи AT&T, NTT Communications, SK Telecom і China Unicom. ВР ONOS доповнює можливості Blue Planet для багатодоменого регулювання, а також регулювання послуг NFV.

Blue Planet – перший і найпоширеніший у галузі контролер SDN із системою керування для мереж операторів. Він забезпечує автоматизацію, керування й контроль багаторівневої мережної інфраструктури на базі рішень різних виробників. Поєднуючи компоненти на рівні оператора й керування мережею (повну систему FCAPS) з розширеним контролем і програмуємістю SDN, Blue Planet дозволяє швидше й з більшою рентабельністю надавати послуги в багаторівневих мережах, побудованих із застосуванням рішень різних постачальників.

ONOS поставляється з повним комплектом ПЗ й послуг, а крім того, надає технічну підтримку. Тим самим ми сподіваємося зміцнити в наших клієнтах

КОПЗ-2025

упевненість у тім, що вони вибрали правильне сполучення рішень, що дозволяють спростити впровадження технологій віртуалізації мережі.

Насамперед рішення Blue Planet ONOS орієнтоване на сценарій використання CORD, у ході якого здійснюється трансформація центрального телефонного вузла в центр обробки даних. При цьому воно доповнює й розширює можливості Blue Planet по багатодоменому регулюванню послуг і регулюванню послуг NFV з метою контролю комутаційних матриць і керування ними на базі систем власного складання в центральних офісах.

І найважливіше: Blue Planet ONOS дозволяє постачальникам послуг використовувати досвід, накопичений співтовариством розроблювачів ПЗ з відкритим вихідним кодом, а також можливості обслуговування й технічної підтримки, щоб забезпечити успішне комерційне розгортання. Всі поліпшення й виправлення, внесені в ONOS, будуть передаватися в співтовариство розроблювачів ПЗ з відкритим кодом.

Рішення являє собою компактну інтелектуальну платформу CPE з підтримкою 10Gb і можливістю інтеграції з віртуальними мережними функціями. У першу чергу це рішення призначене для постачальників послуг.

На платформі можна розмістити трохи VNF, які, у свою чергу, дозволяють операторові сформувати цілий портфель додаткових керованих послуг для гнучкого розгортання віртуальних функцій, таких як оптимізація WAN і віртуальний корпоративний маршрутизатор.

Віртуальні функції стають усе більше й більше популярними, а модель реалізації поступово стає більше гнучкою. Постачальники послуг можуть одержати доступ до нових джерел доходу, розгорнувши ці важливі мережні функції на граничних ділянках мережі, причому найчастіше на території замовників. Брандмауери, корпоративні маршрутизатори й оптимізація WAN, по загальному визнанню, стають набагато більше рентабельними при інтеграції у віртуальну платформу.

КОПІЯ 2025

Попит на пропускну здатність каналів зв'язку росте з кожним роком усе швидше й швидше. Тепер проблема полягає в тім, що для збільшення пропускної здатності по звичці вдосконалюється спочатку апаратне забезпечення.

Однак, по даним Gartner, в 2017 році глобальний мобільний трафік даних досяг 52 млн терабайт, тобто збільшився на 60% у порівнянні з 2016 роком. Починається новий етап, коли навіть темпів відновлення комп'ютерного встаткування відповідно до закону Мура виявляється недостатньо для задоволення зростаючих потреб. Тому необхідно знайти нові інструменти, які дозволять задовольняти постійно зростаючий попит.

Для розвитку й просування необхідно, щоб пропускна здатність наявних мереж стала більше адаптивної, а самі мережі були перетворені в програмувальні платформи. Інфраструктура повинна бути гнучкою, динамічною й здатною підтримувати роботу в реальному часі, як сучасні смартфони. Тобто потрібно перейти до створення мереж на вимогу. Топологія мережі, можливості підключення, клас послуг і якість обслуговування повинні дозволити надавати послуги по запиті, що набудовуються відповідно до потреб кінцевого користувача.

Ключем до успіху є більше інтенсивне використання програмних ресурсів. У розвитку мереж наступила дуже важлива й цікава епоха. З появою рішень для програмно обумовлених мереж (SDN) і віртуалізації мережних функцій (NFV) мережа перестала бути всього лише набором розрізнених фрагментів устаткування – тепер апаратне забезпечення поставляється в комплекті з ПЗ, створюючи програмувальні рішення, які дозволять нам випередити темпи

3.2 Розробка структурної схеми

Коли компанія дорастає до певного розміру й одного ЦОДа їй стає мало, відразу виникає маса питань, як далі розвивати мережну інфраструктуру. Дійсно, як розширити границі існуючого ЦОД, щоб він прозоро забезпечував існуючі

КПІЗ_2025

сервіси на віддалених площадках? Робити великий L2 домен, щоб не було проблем з віртуалізацією або поєднувати площадки по третьому рівні? Якщо робити інфраструктуру ієрархічної, то як обійти обмеження існуючих стандартів (802.1q) і що буде в цьому випадку з безпекою? А як, при цьому, забезпечити надійну передачу конвергентного трафіку (e.g. FCo) між площадками? І всім цим ще необхідно злагоджено управляти.

Стійкий «тренд» останнього часу на віртуалізацію й побудова хмарних інфраструктур однозначно показує, що переважніше інших з багатьох причин є варіант із об'єднанням площадок ЦОД по другому (L2) рівню. Однак відразу виникає питання, яку технологію для цього використовувати? Очевидно, що будувати зараз розподілений L2 домен на основі STP, як мінімум, не раціонально. З існуючих альтернатив – TRILL, PBB/SPB, FabricPath (proprietary), MPLS/VPLS, dark fiber – варіант із використанням для DCI технології VPLS є, з одного боку, самим зрілій і перевіреним на практиці, з іншого боку – гнучким і богатим по функціональності. Про нього далі й поговоримо докладно.

Щоб не ускладнювати приклад, візьмемо два географічно рознесених ЦОДа. Отже, завдання – прозоро для віртуального середовища (читай – VM) об'єднати в ЦОД дві площадки, щоб при цьому:

– прозоро пропускати трафік рівня 2 (L2) по всій мережі ЦОД;

– уникнути блокованих каналів і неефективного використання смуги пропущення;

– максимально спростити мережне керування;

– забезпечити необхідну відказостійкость і захист від різного роду збоїв;

– забезпечити простоту нарощування портової ємності й, у цілому, масштабованість мережі;

– можливість поступового нарощування портової ємності додаванням комутаторів у стек у міру необхідності;

Для рішення цього завдання на кожній площадці встановлюємо по два комутатора HP серії 5800 і поєднуємо їх в один віртуальний комутатор (для

КБПЗ-2025

надійності, якщо не критично, те можна цей крок пропустити). Робиться це просто, комутатори поєднуються через стандартні 10G порти, далі IRF-стек налаштовується.

Потім площадки фізично з'єднуються 10G інтерфейсам (думаємо, що між площадками ЦОД лежить оптика) і інтерфейси збираються в LACP-агрегат.

На другій площадці віртуальний комутатор налаштовується аналогічно. При цьому відзначимо, що оптика, що з'єднує площадки, «приземляється» на фізично різні комутатори. Це робиться на випадок падіння одного з комутаторів у стеці, щоб трафік автоматично переходив на другий комутатор/канал. Т.е., одержуємо таку картину:

Потім, щоб прозоро «прокидати» між площадками VLAN-И (трафік другого рівня), піднімаємо на цих комутаторах VPLS сервіс. Для цього спочатку набудовуємо MPLS і LDP сервіси так, щоб LSP (Label Switched Path) «сигналились» автоматично.

Піднімає в MPLS-ядрі OSPF, щоб не морочитися зі статичними маршрутами й комутатори на площадках бачили один одного по IP (і «кладемо» в OSPF LoopBack-).

Після цього, набудовуємо на обох площадках сигналізацію VPLS. При цьому питання, використовувати варіант Компелла або Мартіні – це більше питання релігії. На встаткуванні працюють варіанта. На LDP (Мартіні) налаштовується L2 MPLS VPN, потім створюється VPLS Instance (VSI) і приєднується до інтерфейсу.

З BGP більш складно, плюс до всьому потрібно ще підняти BGP Peering між площадками, настроїти в ньому VPLS family і встановити параметри для VPLS VSI (vpn-target і route-distinguisher).

КБПЗ_2025

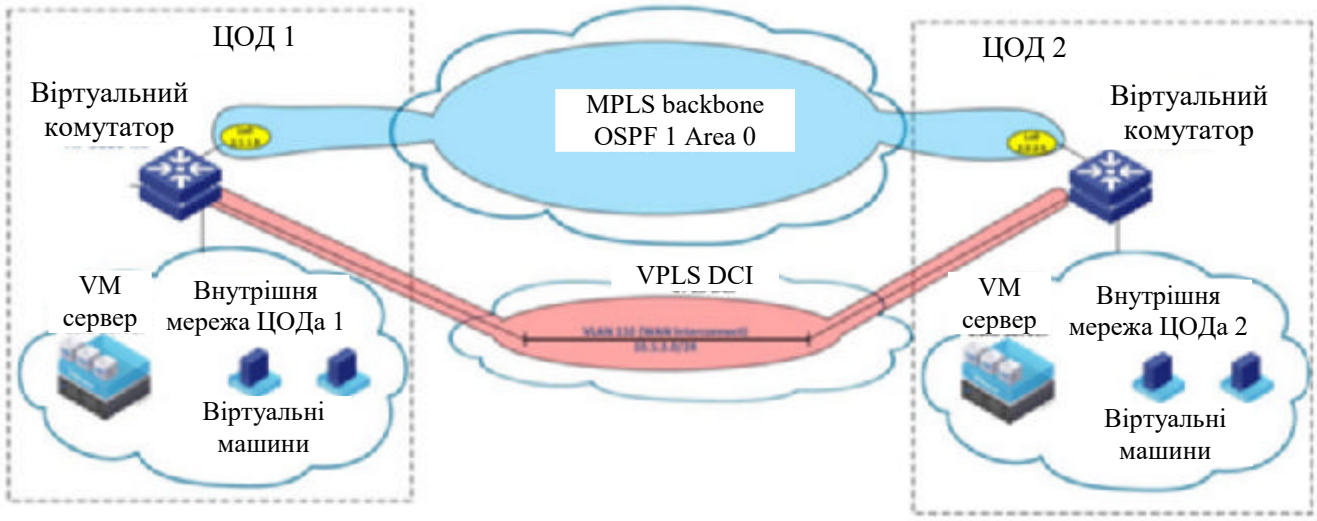


Рисунок 3.1 – Структурна схема системи

У такій схемі трафік другого рівня прозоро пробрасується в VLAN-ах між площадками, а віртуальні ресурси й додатки можуть безперешкодно мігрувати з однієї площадки на іншу, начебто вони розташовані в одному місці.

У цьому спрощеному прикладі не зовсім очевидно, навіщо піднімати VPLS між двома площадками? Здається, можна просто прокинути trunk і VLAN-и стануть доступні на обох площадках... Однак, крім завдань керування трафіком, мережною безпекою і якістю сервісу, які забезпечує MPLS/VPLS, як тільки в L2 домені з'явиться ще одна площадка, (або ще одне redundant з'єднання), виникне необхідність управляти доступом до середовища й захищатися від петель. І робити це з STP зараз, як мінімум, поганий тон.

VPLS

VPLS (англ. **Virtual Private LAN Service** – сервіс віртуальної приватної мережі) – один зі способів організації зв'язку точка-багатоточка на базі IP / MPLS мереж, що дозволяє поєднувати географічно розділені об'єкти/офіси в єдину мережу.

VM

Віртуальна машина виконує деякий машинно-незалежний код (наприклад, байт-код, щойшов код, р-код) або машинний код реального процесора. Крім

КПІЗ–2025

процесора, ВМ може емулювати роботу як окремих компонентів апаратного забезпечення, так і цілого реального комп'ютера (включаючи BIOS, оперативну пам'ять, жорсткий диск і інші периферійні пристрої). В останньому випадку у ВМ, як і на реальний комп'ютер, можна встановлювати операційні системи (наприклад, Windows можна запускати у віртуальній машині під Linux або навпаки).

IRF-стек

Технологія IRF (Intelligent Resilient Framework) – це пропрієтарна технологія від компанії HP, що дозволяє забезпечити стек до дев'яти комутаторів і 1024 портів 1Gb /10Gb

Надійність і безперебійна робота комп'ютерної мережі – одна з найважливіших складових успішного бізнесу. Щоб убезпечити мережа від зупинки роботи системні адміністратори резервують (дублюють) всі її компоненти (комутатори, роутери, канали підключення).

У зв'язку із цим мережа розростається, що веде до деяких проблем:

– При відмові роботи частини мережі час перемикання на резервні канали може становити кілька секунд, що критично для роботи більшості додатків. Наприклад у банківській сфері, де обрив банківської операції може вилитися в більші фінансові витрати й недовіру до банку.

– Неефективне використання смуги пропущення (оскільки між двома комутаторами може бути кілька фізичних шляхів, пакети передаються тільки по одному активному з'єднанню)

– Складності конфігурування комутаторів і усунення збоїв через неправильні параметри конфігурації одного з комутаторів

Компанія HP створила унікальне рішення, що допомогло уникнути цих проблем.

BGP

BGP (англ. Border Gateway Protocol, протокол граничного шлюзу) – динамічний протокол маршрутизації. Ставиться до класу протоколів маршрутизації зовнішнього шлюзу (англ. EGP – External Gateway Protocol).

КПІЗ_2025

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2. Представимо опис розробленого програмного комплексу для системи забезпечення міжз'єднань центрів обробки даних повідомлень на основі аналітичного й імітаційного моделювання.

Завдання проектування мережі складається у визначенні пропускних здатностей КЗ із урахуванням неоднорідності трафіку, різноманіття топологій, алгоритмів маршрутизації, варіантів розміщення прикладних програм і наборів даних по вузлах мережі, способів взаємодії користувачів мережі.

Аналітична модель системи забезпечення міжз'єднань центрів обробки даних будується у вигляді розімкнутої СеМО для будь-якої топології мережі, що задається аналітично або графічно. На основі аналітичної моделі вирішується завдання визначення пропускних здатностей КЗ у розподілених системи забезпечення міжз'єднань центрів обробки даних при обмеженнях на час доставки пакетів або на вартість мережі.

Програма дозволяє розрахувати характеристики системи забезпечення міжз'єднань центрів обробки даних, такі як:

– пропускні здатності КЗ;

– час затримки пакетів при передачі по кожному каналу й у мережі в цілому;

– завантаження кожного КЗ;

– інтенсивності потоків пакетів у каналах зв'язку;

– імовірності передачі пакетів від користувачів у мережу, між каналами й з мережі до користувачів.

КБПЗ – 2025

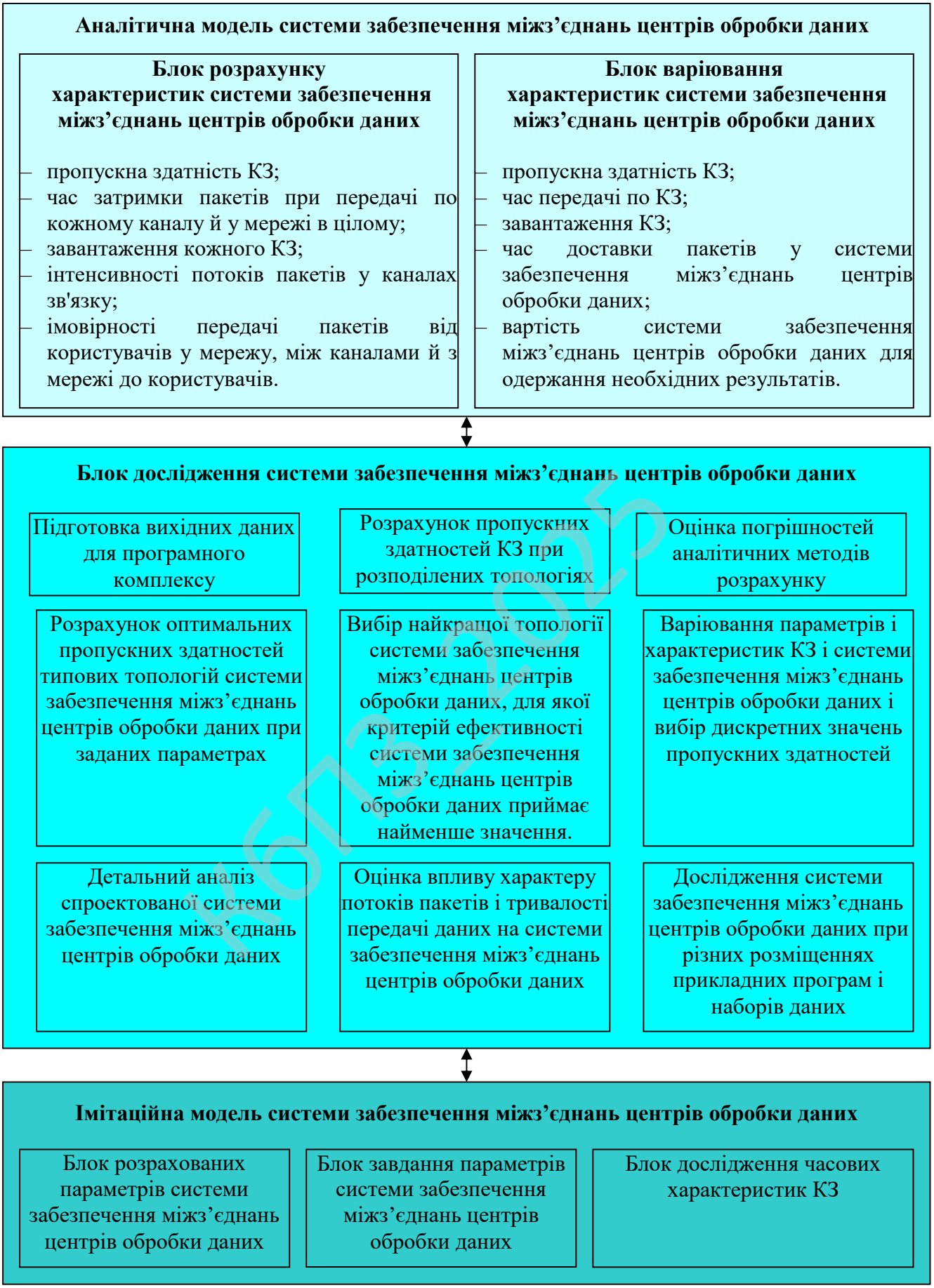


Рисунок 3.2 – Функціональна схема системи

КбПЗ 2025

Програма дозволяє варіювати отримані характеристики, такі як:

– пропускні здатності;

– час передачі по КЗ;

– завантаження КЗ;

– час доставки пакетів у системи забезпечення міжз'єднань центрів обробки даних;

– вартість системи забезпечення міжз'єднань центрів обробки даних для одержання необхідних результатів.

Крім того, вона дозволяє вибрати значення пропускних здатностей з дискретного ряду значень, знайдених із традиційної оптимізації безперервних значень пропускних здатностей.

Імітаційна модель системи забезпечення міжз'єднань центрів обробки даних генерується автоматично на основі розрахунку характеристик аналітичної моделі у вигляді розімкнутої експонентної СеМО. Отримані характеристики аналітичної моделі використовуються як параметри імітаційної моделі системи забезпечення міжз'єднань центрів обробки даних, що представляє собою розімкнуту СеМО, вузли якої відповідають каналам зв'язку, а заявки – пакетам у системи забезпечення міжз'єднань центрів обробки даних. Число джерел заявок в імітаційній моделі дорівнює числу вузлів у системи забезпечення міжз'єднань центрів обробки даних, при цьому інтенсивності надходження заявок у моделі визначаються як зовнішні інтенсивності пакетів від користувачів до вузлів системи забезпечення міжз'єднань центрів обробки даних відповідно. Аналітична модель, реалізована в програмі, дозволяє розрахувати наступні параметри для імітаційної моделі.

1. Імовірність передачі пакетів від користувача j до каналу k.

2. Імовірність передачі пакетів від каналу k до каналу h.

3. Імовірність передачі пакетів від каналу k до користувача j.

4. Середній інтервал часу між вступниками пакетами від користувача j у системи забезпечення міжз'єднань центрів обробки даних.

КПЗ–2025

5. Середній час передач пакетів у каналах зв'язку *k*.

Крім **розрахованих параметрів** для імітаційної моделі системи забезпечення міжз'єднань центрів обробки даних додатково необхідно **задати наступні параметри**:

а) закони розподілів інтервалів часу між вступниками пакетами від користувачів у системи забезпечення міжз'єднань центрів обробки даних із коефіцієнтами варіації цих розподілів;

б) закони розподілів часу передачі пакетів у КЗ із коефіцієнтами варіації цих розподілів.

Відзначимо, що імітаційна модель системи забезпечення міжз'єднань центрів обробки даних призначена для детального аналізу характеристик функціонування мережі, спроектованої в процесі аналітичного моделювання. При цьому, у випадку відмінності реального характеру процесів надходження пакетів у мережу або передачі пакетів по каналах зв'язку від експонентного, в імітаційній моделі передбачена можливість варіювання законів розподілу часу передачі (довжин обслуговування) пакетів у кожному із КЗ, а також законів розподілу інтервалів часу між вступними в мережу пакетами. Як такі закони в роботі використовувалися наступні розподіли: експонентний, детермінований; гіпоекспоненційний різного порядку й, відповідно, з різними коефіцієнтами варіації; рівномірний; експонентний з ненульовими зсувами; гіперекспонентний; Гамма-розподіл.

Аналогічно **розроблений засіб дослідження часових характеристик каналу зв'язку** шляхом генерування імітаційної моделі каналу зв'язку у вигляді СМО типу G/G/1 з можливістю зміни завантаження каналу й варіювання законів розподілу інтервалів часу між пакетами й часу передачі пакетів по каналі. Даний засіб дозволяє одержати значення часових характеристик каналу зв'язку й оцінити погрішність аналітичних методів розрахунку характеристик каналу зв'язку.

КБПЗ-2025

Сформульовано **методику дослідження розподіленої системи забезпечення міжз'єднань центрів обробки даних** із неоднорідним трафіком, що містить наступні етапи.

1. Підготовка вихідних даних для програмного комплексу:

– кількість вузлів мережі і їхнє взаємне розташування;

– навантаження системи забезпечення міжз'єднань центрів обробки даних, створювана користувачами при роботі із прикладними програмами, наборами даних і в процесі обміну повідомлень;

– обмеження на середній час доставки пакетів або на вартість системи забезпечення міжз'єднань центрів обробки даних;

– максимальна довжина пакета;

– вибір типу каналів і завдання вартісних коефіцієнтів КЗ.

2. Розрахунок оптимальних пропускних здатностей типових топологій системи забезпечення міжз'єднань центрів обробки даних при заданих наступних параметрах:

– типова топологія: зірка, кільце, дерево, повнозв'язна;

– модель взаємодії користувачів мережі: RDA (Remote Data Access), DBS (DataBase Server) і AS (Application Server);

– розподіл прикладних програм і наборів даних по вузлах системи забезпечення міжз'єднань центрів обробки даних;

– метод маршрутизації.

Після завдання цих параметрів розраховуються оптимальні значення пропускних здатностей КЗ, час передачі пакетів і завантаження каналів.

3. Розрахунок пропускних здатностей КЗ при розподілених топологіях.

4. Вибір найкращої топології системи забезпечення міжз'єднань центрів обробки даних, для якої обрана залежно від постановки завдання як критерій ефективності характеристика системи забезпечення міжз'єднань центрів обробки даних (середній час доставки пакетів або вартість системи забезпечення міжз'єднань центрів обробки даних) приймає найменше значення.

КПІ – 2025

5. Варіювання параметрів і характеристик КЗ і системи забезпечення міжз'єднань центрів обробки даних і вибір дискретних значень пропускних здатностей.

6. Оцінка погрішностей аналітичних методів розрахунку характеристик КЗ.

7. Оцінка впливу характеру потоків пакетів і тривалості передачі даних на характеристики системи забезпечення міжз'єднань центрів обробки даних:

– вплив третього моменту розподілу вхідного потоку пакетів на характеристики КЗ;

– вплив довжини пакетів на характеристики каналів зв'язку й системи забезпечення міжз'єднань центрів обробки даних;

– вплив законів розподілів трафіку в мережах і часі передачі пакетів у КЗ на характеристики функціонування системи забезпечення міжз'єднань центрів обробки даних.

8. Дослідження системи забезпечення міжз'єднань центрів обробки даних при різних розміщеннях прикладних програм і наборів даних.

9. Детальний аналіз спроектованої системи забезпечення міжз'єднань центрів обробки даних, що припускає варіювання параметрів системи забезпечення міжз'єднань центрів обробки даних, у тому числі: довжини запитів і відповідей прикладних програм і наборів даних, імовірність передачі по основному шляху, способи маршрутизації й т.д.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

КОПЗ_2025

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма взаємодії процесів

використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів: Процеси які являють собою трансформацію даних в рамках описуваної системи; Сховища даних (репозиторії); Зовнішні по відношенню до системи сутності; Потоки даних між елементами трьох попередніх типів.

Рисунок 3.3 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

КПЗ-2025

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо використані мережні підходи для реалізації програми, а саме архітектуру клієнт-сервер яка є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

КПЗ-2025

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

– прикладний рівень, який реалізує основну логіку ПЗ і на якому здійснюється необхідна обробка інформації;

– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка ПЗ та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови

КОПЗ-2025

розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка ПЗ. Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення –звичайний браузер. Це позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає).

КОПЗ–2025

Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних програм, крім PHP, широко застосовуються Java, Perl, Python, Delphi.

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

КПЗ-2025

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

КПЗ - 2025

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю забезпечення міжз’єднань центрів обробки даних.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

КОПЗ-2025

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ.

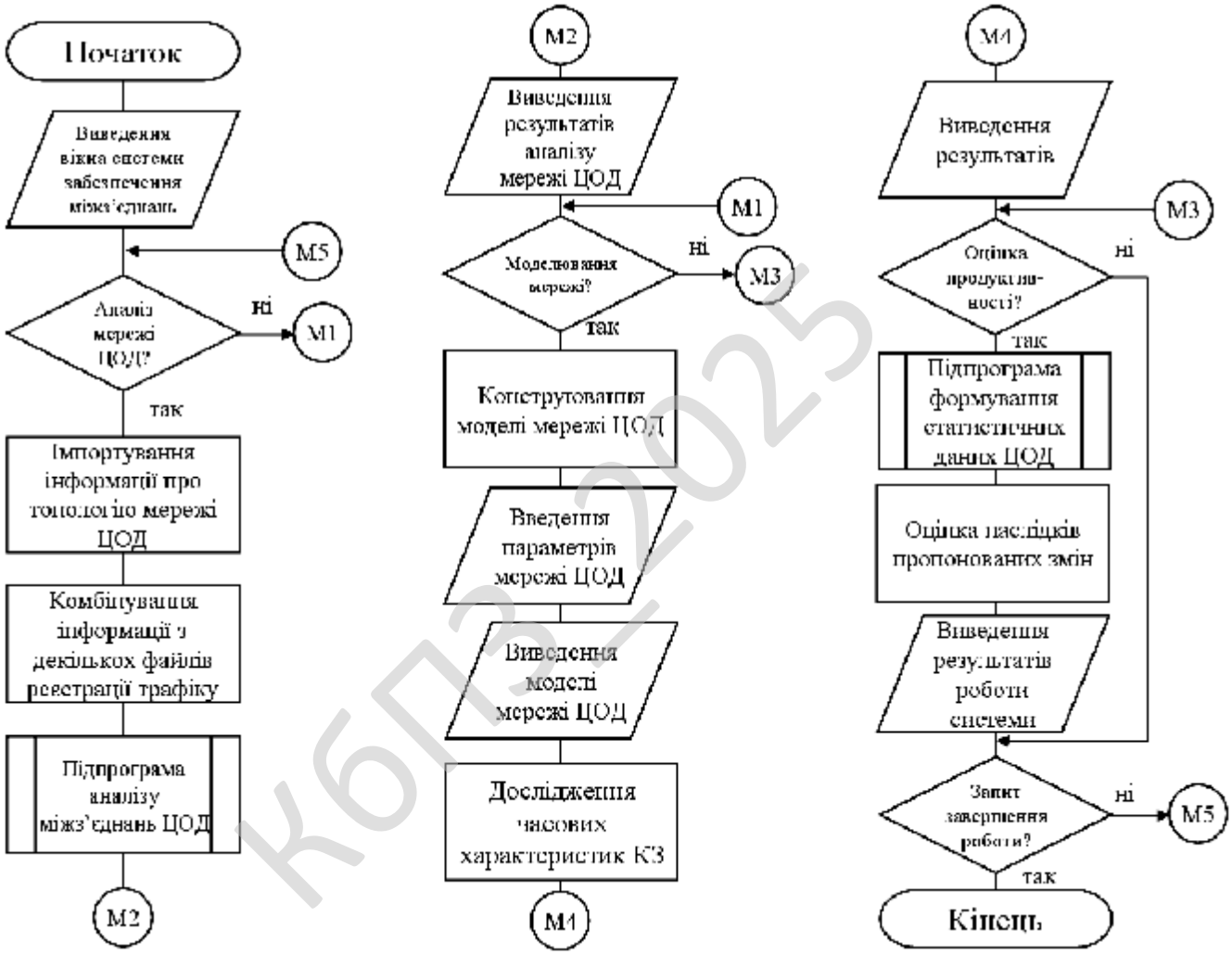


Рисунок 4.1 – Блок-схема основної програми

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

КПЗ-2025

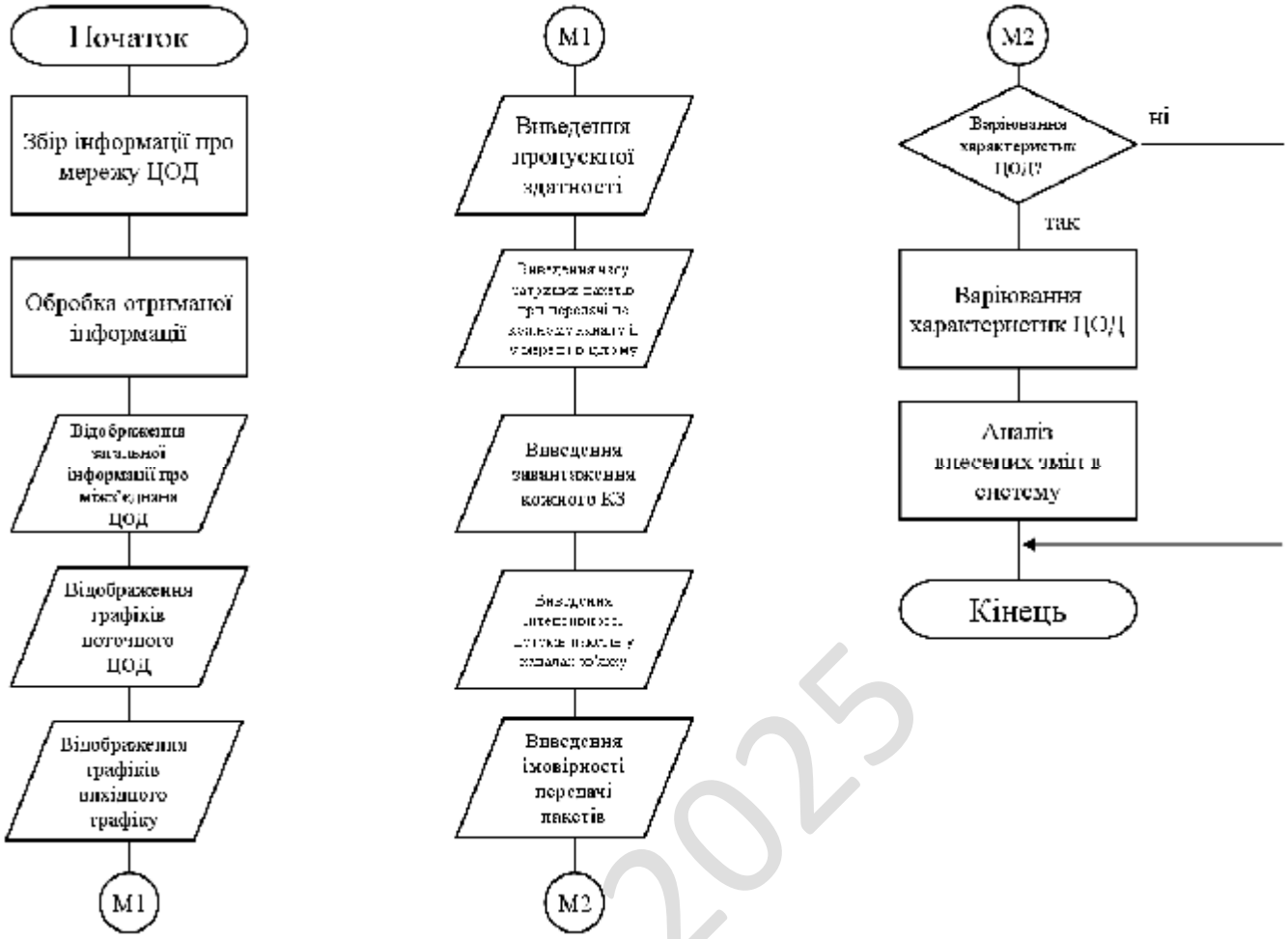


Рисунок 4.2 – Блок-схема роботи підпрограми

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: Діаграма класів; Діаграма компонент; Діаграма об'єктів; Діаграма розгортання.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

КБПЗ - 2025

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати

КПЗ-2025

діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Діаграма компонент в UML це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

Модуль програмного забезпечення може бути представлено в якості компоненти. Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми.

КОПЗ_2025

Діаграма компонент відображає лише структурні характеристики, для відображення окремих екземплярів компонент слід використовувати діаграму розгортування.

Компоненти об'єднуються разом використовуючи структурні зв'язки (assembly connector) щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

Структурна взаємодія – «зв'язок двох компонент, який передбачає, що один з них надає послуги, потрібні іншому компоненту».

При використанні діаграми компонент щоб показати внутрішню структуру компонента, клієнтські та серверні інтерфейси можуть утворювати пряме з'єднання з внутрішніми. Таке з'єднання називається з'єднанням делегації.

Діаграма об'єктів в UML це діаграма, що відображає об'єкти та їх зв'язки в певний момент часу. Діаграма об'єктів може розглядатись як окремий випадок діаграми класів, на якій можуть бути представлені як класи, так і екземпляри (об'єкти) класів. Схожою за змістом є діаграма взаємодії (collaboration diagram).

Діаграми об'єктів не мають власної нотації. Оскільки діаграми класів можуть відображати об'єкти, то діаграма класів, на якій відображено лише об'єкти, та не відображено класи, може вважатись діаграмою об'єктів.

Діаграма об'єктів відображає об'єкти та зв'язки в певний момент роботи програми. Об'єкти можуть містити інформацію про власні значення а не про описання. Для відображення загальних шаблонів об'єктів та зв'язків, що можуть багаторазово створюватись під час роботи програми, слід використовувати діаграму взаємодії, яка може відображати характеристики об'єктів та зв'язків. Екземпляр діаграми взаємодії створює діаграму об'єктів.

Діаграма об'єктів не відображає еволюцію системи під час роботи. Натомість, слід використовувати діаграми взаємодії з повідомленнями, або діаграми послідовності.

Діаграма розгортання (deployment diagram) це діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та

КПЗ-2025

об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Serpent – симетричний блочний алгоритм шифрування, розроблений Россом Андерсоном, Елі Біхамом та Ларсом Кнудсеном. Алгоритм був одним з фіналістів 2-го етапу конкурсу AES. Як і інші алгоритми, які брали участь у конкурсі AES, Serpent має розмір блоку 128 біт і можливі довжини ключа 128, 192 або 256 біт. Алгоритм являє собою 32-раундовий шифр на основі SP-мережі, і працює з блоком з чотирьох 32-бітових слів. Serpent був розроблений так, що всі операції можуть бути виконані паралельно, використовуючи 32-а 1-бітних «потоків».

При розробці Serpent використовувався консервативніший підхід до безпеки, ніж у інших фіналістів AES, проектувальники шифру вважали, що 16 раундів достатньо, щоб протистояти відомим видам криптоаналізу, але збільшили число раундів до 32, щоб алгоритм міг краще протистояти ще не відомим методам криптоаналізу.

Шифр Serpent не запатентований і є громадським надбанням.

Алгоритм створювався під гаслом «криптографічний алгоритм 21 століття» для участі в конкурсі AES. При створенні нового алгоритму Serpent його автори дотримувалися консервативних поглядів на проектування, що підтверджується первісним рішенням про використання таблиць підстановки з відомого багато років раніше алгоритму шифрування DES, який протягом

КПЗ-2025

довгого часу вивчався провідними фахівцями в області криптографії та захисту інформації і чиї властивості і особливості були добре відомі науковому світу. Одночасно з цим до нового алгоритму міг бути застосований вичерпний аналіз, вже розроблений для DES. Не використовувалися нові, неперевірені і невипробувані технології при створенні шифру, який у разі прийняття був би використаний для захисту величезних масивів фінансових транзакцій та урядової інформації. Основною вимогою до учасників конкурсу AES було те, що алгоритм-претендент повинен бути швидшим, ніж 3DES, і надавати як мінімум такий же рівень безпеки: він повинен мати блок даних довжиною 128 біт і ключ завдовжки 256 біт. 16-раундовий Serpent був би таким же надійним, як 3DES, але в два рази швидшим. Однак, автори вирішили, що для більшої надійності варто збільшити кількість раундів до 32. Це зробило шифр таким же швидким, як DES, і набагато надійнішим, ніж 3DES.

Структура алгоритму

Алгоритм Serpent є SP-мережею, у котрій весь блок даних довжиною 128 біт на кожному раунді розбивається на 4 слова довжиною 32 біти. Всі значення, що використовуються при шифруванні є бітовими потоками. Бітові індекси змінюють значення від 0 до 31 для 32-бітових слів, від 0 до 127 – для 128-бітових блоків та від 0 до 255 для 256-бітових ключів тощо. Для внутрішніх обчислень всі біти величин представлені в прямому порядку (little-endian).

Serpent шифрує відкритий текст P довжиною 128 біт в шифротекст C довжиною таких же 128 біт за 32 раунд за допомогою 33 подключів K{0},...,K{32}, довжиною 128 біт. Довжина використовуваного ключа може приймати різні значення, але для конкретики зафіксуємо їх довжину в 128, 192 або 256 біт. Короткі ключі довжиною менше 256 біт доповнюються до повної довжини в 256 біт.

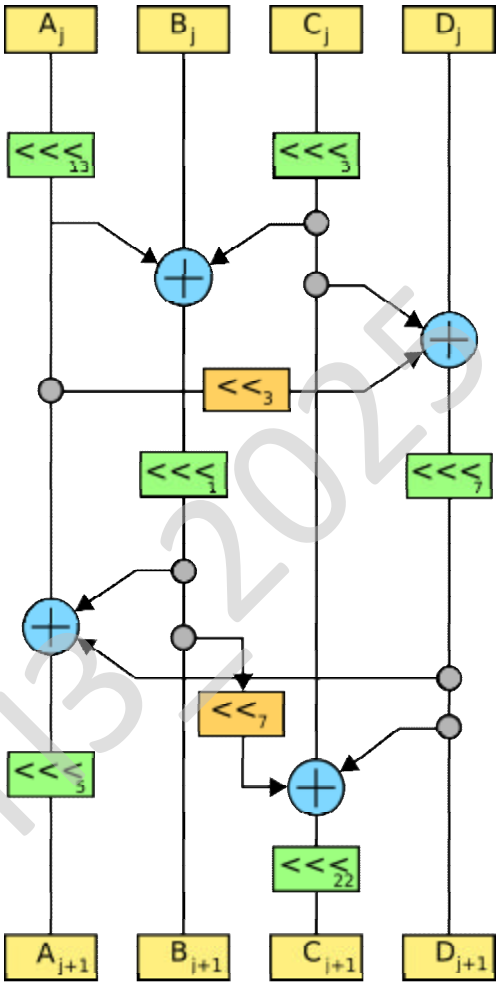
Шифрування складається з наступних основних кроків:

– Початкова перестановка.

КПЗ–2025

– 32 раунд, кожен з яких складається з операції змішування з 128-бітовим ключем (побітове логічне виключаюче «або»), таблична заміна (S-box) і лінійне перетворення. В останньому раунді лінійне перетворення замінюється додатковим накладанням ключа.

– Кінцева перестановка.



КБПЗ 2025

Рисунок 4.3 – Структура алгоритму Serpent

Початкова і кінцева перестановки не мають будь-якої криптографічного значущості. Вони використовуються для спрощення оптимізованої реалізації алгоритму і підвищення обчислювальної ефективності.

Розширення ключа

Як і інші алгоритми, що брали участь в конкурсі AES, Serpent має можливі довжини ключа 128, 192 або 256 біт. «Неповний» ключ довжиною менше 256 біт доповнюється за наступним правилом: додається одиничний біт справа, за ним слід стільки нульових бітів, щоб довжина ключа стала дорівнює 256 бітам.

Початкова перестановка IP

Дана перестановка IP задається таблицею, де вказується позиція, на яку перейде відповідний біт (наприклад, біт 1 перейде на 32 позицію):

S-бокси (таблиці замін)

В алгоритмі Serpent таблиці замін є 4-бітовими перестановками з властивостями стійкості до диференціального криптоаналізу, до лінійного криптоаналізу і такою властивістю, що порядок вихідних біт, як функції вхідних повинен бути максимальний, тобто бути рівним 3.

Таблиця підстановки генерується з відомих і добре вивчених таблиць для алгоритму DES в ітераційному процесі, поки не будуть отримані бажані диференціальні й лінійні властивості. Таким чином, створюється 8 таблиць підстановки.

Лінійне перетворення LT

Лінійне перетворення LTзадається таблицею, де біти перераховані від 0 до 127 (наприклад, вихідний 2 біт утворений 2, 9, 15, 30, 76, 84, 126 битами, складеними за модулем 2) . В кожному рядку описується 4 вихідних біти, які разом складають вхідні дані на одну таблицю замін в наступному раунді. Варто зазначити, що даний набір являє собою таблицю IP(LT(FP(x))), де LT і є те лінійне перетворення.

Таблиця зворотного лінійного перетворення, яке використовується при розшифровці ILT.

Кінцева перестановка FP

Дана перестановка є зворотною до початкової, тобто $FP=IP^{-1}$ і задається наступною таблицею.

КПЗ_2025

Ефективна реалізація алгоритму

Бажання авторів зробити алгоритм саме таким, яким він є стає зрозумілим при розгляді його ефективної низькорівневої реалізації.

Serpent був створений таким чином, щоб всі операції в процесі шифрування і розшифрування одного блоку могли бути виконані паралельно в 32 потоках. До того ж низькорівневий опис алгоритму набагато простішій, ніж стандартний опис. Ніяких початкових і кінцевих перестановок не потрібно.

Шифрування складається з 32 раундів. Відкритий текст є першими проміжними даними $B_0 = P$. Потім виконується 32 раунди, кожен і-й раунд складається з:

– Змішування з ключем. Проводиться побітове виключаюче «або» проміжних даних B_i з ключем довжиною 128 біт.

– Застосування таблиць підстановки. Вхідні дані довжиною 128 біт поділяються на 4 слова по 32 біта. Таблиця підстановки, реалізована послідовністю логічних операцій (як якщо це було б реалізовано апаратно), застосовується до цих 4 словам. В результаті виходить 4 вихідних слова. Таким чином, центральний процесор виконує підстановку по 32 копій таблиці одночасно.

– Лінійне перетворення. 32-бітові слова перетворюються заданим порядком.

Першою причиною вибору такого лінійного перетворення є максимізація лавинного ефекту. Такі таблиці підстановки мають властивість, що зміна кожного вхідного біта призведе до зміни 2 вихідних бітів. Таким чином, кожен вхідний біт відкритого тексту вже через 3 раунди впливає на всі вихідні біти. Аналогічно кожен біт ключа впливає на результат шифрування.

Друга причина полягає в простоті перетворення. Воно може бути реалізоване на будь-якому сучасному процесорі з мінімальними витратами.

КБПЗ–2025

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ забезпечення міжз'єднань центрів обробки даних яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Файл; Аналізатор мережі; Моделювання мережі; Оцінка продуктивності; Параметри; Довідка.

– Розділу обрання груп у деревовидному вигляді.

– Розділу мережного інтерфейсу.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

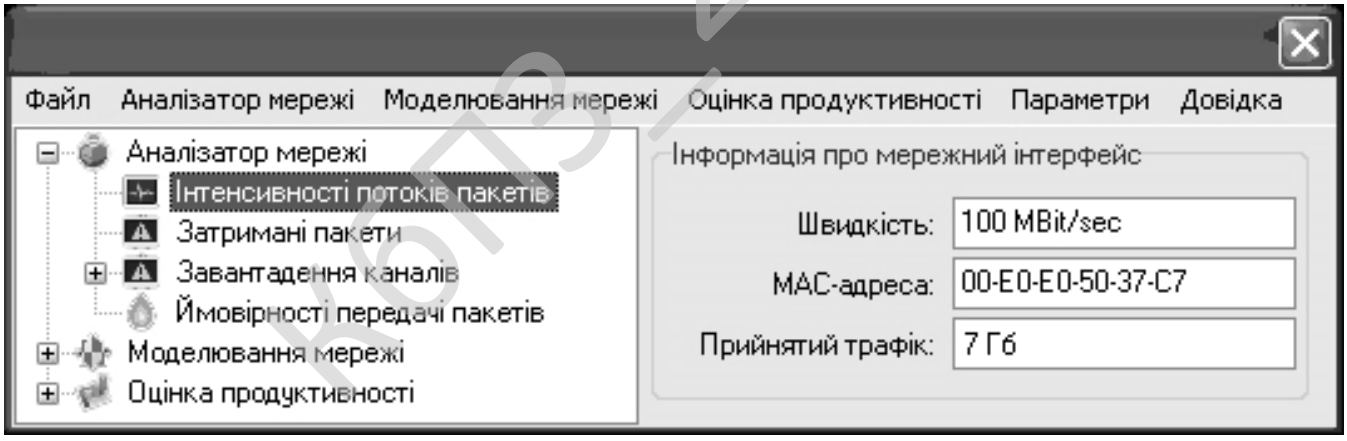


Рисунок 5.1 – Головне вікно ПЗ

Спеціально для завдань DCI масштабу Web Scale розроблена нарощувана система забезпечення міжз'єднань. Вона опирається на дві основні ідеї – рентабельність WaveLogic і IT-функціонал Web Scale.

КПІ – 2025

Waveserver розрахована на модульне розгортання в стійках і стеках, при цьому один компактний модуль форм-фактора 1RU забезпечує високу, до 400G, пропускну здатність із боку клієнта й ще 400G для зовнішнього підключення.

Система Waveserver дозволяє розширити IT-функціонала серверів до рівня мережних функцій, а її ключовими характеристиками є програмуємість і відкритість: користувачі й розроблювачі можуть створювати, тестувати і набудовувати додатка за допомогою Emulation Cloud – відкритого середовища розробки й проведення випробувань.

Розроблена програма має дуже простий і інтуїтивно зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом.

Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача.

Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

КПЗ-2025

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

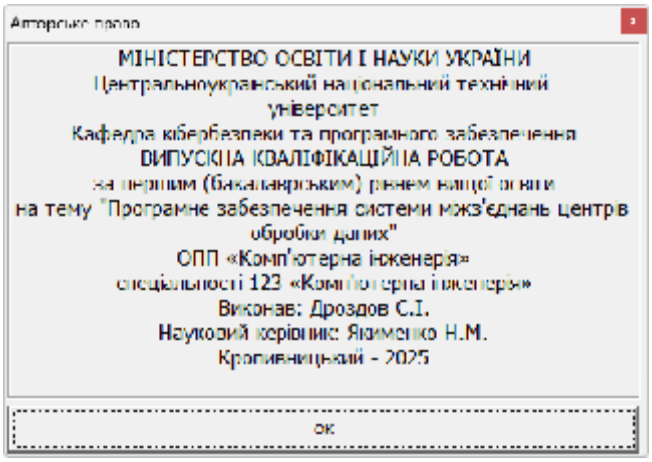


Рисунок 5.2 – Авторське право

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Віддалення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати IT рішення для автоматизації операцій усередині саме цих

КбПЗ_2025

процедур. Таким чином, розроблене ІТ рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

КПЗ-2025

Проводилось тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Обрано умови розповсюдження – Shareware. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому

КПЗ – 2025

випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

КбПЗ_2025

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи міжз'єднань центрів обробки даних.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем міжз'єднань центрів обробки даних.

– Досліджена система міжз'єднань центрів обробки даних.

– На основі отриманих результатів досліджень створена програмна реалізація системи міжз'єднань центрів обробки даних.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання міжз'єднань центрів обробки даних.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи міжз'єднань центрів обробки даних. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне

КПЗ-2025

програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Serpent.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

КбПЗ_2025

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оліфер В.Г. Комп'ютерні мережі. Принципи, технології, протоколи. Підручник / В.Г. Оліфер, Н.А.Оліфер. – [5-е вид.]. – 2016. – 944 с.

2. Е. Таненбаум, Д. Уезеролл «Комп'ютерні мережі». – [5-е вид.]. – 2016. – 960 с.

3. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.

4. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.

5. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

6. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

7. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

8. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

9. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

10. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

11. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

КОПІЯ 2025

12. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

13. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

14. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

15. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

16. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

17. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

18. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

КОПЗ-2025

19. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

20. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

21. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645.

22. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». *International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019*; Odessa; Ukraine; 9-13 September 2019. P.22-28.

23. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

24. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

25. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

КОПЗ – 2025

26. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

27. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

28. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

29. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

30. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

31. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

32. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

КБПЗ_2025

33. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

34. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

35. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

36. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

37. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

38. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

39. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного

КОПЗ – 2025

процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

40. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

41. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

42. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

43. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

44. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

45. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

46. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

КБПЗ-2025

47. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

48. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

49. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

50. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

КбПЗ_2025

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

КОПЗ-2025

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи міжз’єднань центрів обробки даних.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 48-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи міжз’єднань центрів обробки даних.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

КБПЗ_2025

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи міжз’єднань центрів обробки даних;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

КПЗ_2025

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

КбПЗ – 2025

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 70 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

КБПЗ_2025

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 4.06.2025 р.

КбПЗ_2025

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Якименко Н.М.

Програмне забезпечення системи міжз’єднань центрів обробки даних

КбПЗ_2025

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 19

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#import threading module for concurrent operations
import threading
#import time module for time related operations
import time
#import random module for randomness in simulation
import random
#import logging module for logging events
import logging

#configure logging settings
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

#Define the DataCenter class to represent a data processing center
class DataCenter:
#Constructor for DataCenter with id, name, and IP address parameters
    def __init__(self, id, name, ip):
#Initialize unique identifier for the data center
        self.id = id
#Initialize the human-readable name of the data center
        self.name = name
#Initialize the IP address for the data center
        self.ip = ip
#Initialize the current load of the data center as zero
        self.load = 0
#Set the default operational status to Active
        self.status = "Active"
#Initialize an empty list to record the history of events
        self.history = []

#Method to update the load value for the data center
    def update_load(self, new_load):
#Log the load update event with previous and new load values
        logger.info("Updating load for DataCenter %s from %d to %d", self.name,
self.load, new_load)
#Set the new load value
        self.load = new_load
#Record the load update event in the history with a timestamp
        self.history.append("Load updated to {} at {}".format(new_load,
time.ctime()))
#Return the updated load value
        return self.load

#Method to check and return the current operational status of the data center
    def check_status(self):
#Log the status check operation for the data center
        logger.info("Checking status for DataCenter %s: %s", self.name,
self.status)
#Record the status check in the history
        self.history.append("Status checked: {} at {}".format(self.status,
time.ctime()))
#Return the current status value
        return self.status

#Method to process an incoming query and simulate a response
    def process_query(self, query):
```

КОПІЯ 2025

```
#Log the receipt of a new query at the data center
        logger.info("DataCenter %s processing query: %s", self.name, query)
#Simulate processing delay with a random sleep duration
        time.sleep(random.uniform(0.1, 0.5))
#Decide randomly if the query processing is successful
        if random.random() < 0.9:
#Formulate a successful response message
            response = "Response from {} for query: {}".format(self.name, query)
#Log the successful processing of the query
            logger.info("Query processed successfully at %s", self.name)
#Record the successful event in the history log
            self.history.append("Query processed successfully at {} at
{}".format(self.name, time.ctime()))
#Return the successful response message
            return response
        else:
#Formulate an error response message due to processing failure
            response = "Error processing query at {}".format(self.name)
#Log the error encountered during query processing
            logger.error("Query processing failed at %s", self.name)

#Record the failure event in the history log
            self.history.append("Query processing failed at {} at
{}".format(self.name, time.ctime()))
#Return the error response message
            return response

#Method to record a custom event message into the data center's history log
    def log_event(self, event_message):
#Log the custom event message for auditing purposes
        logger.info("DataCenter %s event: %s", self.name, event_message)
#Append the event message along with a timestamp to the history list
        self.history.append("{} at {}".format(event_message, time.ctime()))
#Return after logging the event
        return

#Define the DataCenterNetwork class to manage multiple data centers and their
interactions
class DataCenterNetwork:
#Constructor for DataCenterNetwork initializes the network container and
synchronization lock
    def __init__(self):
#Initialize an empty dictionary to store DataCenter objects keyed by their id
        self.data_centers = {}
#Initialize an empty list to record the history of network events and queries
        self.query_history = []
#Create a threading lock to ensure thread-safe operations on the network
        self.lock = threading.Lock()

#Method to add a new DataCenter to the network
    def add_data_center(self, data_center):
#Acquire the lock for thread safety during the addition process
        with self.lock:

#Insert the DataCenter object into the dictionary using its id as the key
            self.data_centers[data_center.id] = data_center
#Log the successful addition of the data center
            logger.info("Added DataCenter: %s", data_center.name)

#Record the addition event in the network's history log
            self.query_history.append("DataCenter {} added at
{}".format(data_center.name, time.ctime()))
```

КОПЗ-2025

```
#Method to remove an existing DataCenter from the network using its id
    def remove_data_center(self, dc_id):

#Acquire the lock to ensure safe removal operation
        with self.lock:
#Check if the specified data center id exists in the network dictionary
            if dc_id in self.data_centers:
#Retrieve the name of the data center to be removed
                name = self.data_centers[dc_id].name
#Delete the data center entry from the dictionary
                del self.data_centers[dc_id]

#Log the removal event of the data center
                logger.info("Removed DataCenter: %s", name)
#Record the removal event in the network's history log
                self.query_history.append("DataCenter {} removed at
{}".format(name, time.ctime()))
#Return a confirmation message indicating successful removal
                return "DataCenter {} removed.".format(name)
            else:
#Log an error message if the data center id is not found in the network
                logger.error("DataCenter with id %s not found.", dc_id)

#Return an error message indicating the failure to remove due to non-existence
                return "DataCenter with id {} not found.".format(dc_id)

#Method to route a specific query to a designated DataCenter using its id
    def route_query(self, dc_id, query):
#Acquire the lock to safely access the network's data centers
        with self.lock:

#Verify if the target data center id exists in the network dictionary
            if dc_id in self.data_centers:
#Log the routing operation for the query
                logger.info("Routing query to DataCenter id: %s", dc_id)
#Record the routing event in the network history
                self.query_history.append("Query routed to {} at
{}".format(self.data_centers[dc_id].name, time.ctime()))

#Delegate the processing of the query to the DataCenter's process_query method
                response = self.data_centers[dc_id].process_query(query)
#Return the response received from the DataCenter
                return response
            else:
#Log an error message if the target data center id does not exist
                logger.error("DataCenter with id %s not found for routing
query.", dc_id)

#Return an error message indicating the absence of the specified data center
                return "DataCenter with id {} not found.".format(dc_id)

#Method to broadcast a single query to all DataCenters in the network
simultaneously
    def broadcast_query(self, query):
#Initialize an empty list to collect responses from all data centers
        responses = []
#Acquire the lock to iterate safely over the data centers
        with self.lock:

#Loop through each DataCenter object in the network dictionary
            for dc in self.data_centers.values():
```

КОПІЯ 2025

```
#Log the broadcasting action for the current data center
                logger.info("Broadcasting query to DataCenter: %s", dc.name)
#Record the broadcasting event in the network history log
                self.query_history.append("Broadcast query sent to {} at
{}".format(dc.name, time.ctime()))
#Invoke the process_query method for the current data center and capture its
response
                response = dc.process_query(query)
#Append the received response to the responses list
                responses.append(response)
#Return the complete list of responses after broadcasting the query
        return responses

#Method to simulate network traffic by sending a series of random queries to
random DataCenters
    def simulate_traffic(self, num_queries):

#Log the start of network traffic simulation with the specified number of
queries
        logger.info("Simulating network traffic with %d queries", num_queries)

#Loop over the range of queries to simulate each individual query
        for i in range(num_queries):

#Acquire the lock to safely select a DataCenter from the network
            with self.lock:
#If there are no DataCenters available, exit the loop early
                if not self.data_centers:
                    break
#Randomly choose a data center id from the available keys in the network
dictionary
                dc_id = random.choice(list(self.data_centers.keys()))
#Generate a unique query string using the loop index and a random integer
            query = "Query_{}_{}".format(i, random.randint(1, 100))

#Log the simulation of the query being sent to the selected DataCenter
            logger.info("Simulating query: %s to DataCenter id: %s", query,
dc_id)
#Route the generated query to the selected DataCenter and capture the response
            response = self.route_query(dc_id, query)
#Log the response obtained from processing the query
            logger.info("Received response: %s", response)

#Pause briefly to simulate network delay between queries
            time.sleep(random.uniform(0.05, 0.2))

#Method to monitor the status of all DataCenters in the network and compile a
status report
    def monitor_network(self):
#Log the commencement of network monitoring
        logger.info("Monitoring network status")
#Initialize an empty dictionary to hold the status report for each DataCenter
        status_report = {}

#Acquire the lock to safely iterate over the network's DataCenters
        with self.lock:
#Loop through each DataCenter in the network dictionary
            for dc_id, dc in self.data_centers.items():
#Retrieve the current status of the DataCenter using its check_status method
                status = dc.check_status()
```

КОТВ-2025

```
#Store the retrieved status in the status report dictionary keyed by DataCenter
id
                status_report[dc_id] = status
#Record the status check event in the network history log
                self.query_history.append("Status of {}: {} at
{}".format(dc.name, status, time.ctime()))
#Return the complete status report after monitoring all DataCenters
        return status_report

#Method to retrieve the complete history log of network events and queries
    def get_network_history(self):
#Log the retrieval process for the network history
        logger.info("Retrieving network history")
#Return the query history list containing all recorded events
        return self.query_history

#Function to simulate external queries arriving at the network system from
outside sources
def simulate_external_queries(network, num_queries):
#Log the start of the external queries simulation
    logger.info("Starting simulation of external queries")

#Loop through the specified number of external queries
    for i in range(num_queries):
#Generate a unique external query string using the loop index and a random four-
digit number
        query = "External_Query_{}_{}".format(i, random.randint(1000, 9999))

#Acquire the network lock to safely select a DataCenter for the query
        with network.lock:
#If no DataCenters are available in the network, skip the current iteration
            if not network.data_centers:
                continue

#Randomly choose a DataCenter id from the available keys in the network
dictionary
            dc_id = random.choice(list(network.data_centers.keys()))
#Log the external query being directed to the selected DataCenter
        logger.info("External query %s directed to DataCenter id: %s", query,
dc_id)
#Route the external query to the chosen DataCenter and capture its response
        response = network.route_query(dc_id, query)
#Log the response received from processing the external query
        logger.info("External query response: %s", response)

#Pause briefly to simulate network latency for external queries
        time.sleep(random.uniform(0.1, 0.3))

#Function to perform scheduled maintenance on all DataCenters in the network
def scheduled_maintenance(network):
#Log the initiation of scheduled maintenance across the network
    logger.info("Starting scheduled maintenance on network")
#Acquire the network lock to ensure safe maintenance operations
    with network.lock:
#Loop through each DataCenter in the network dictionary for maintenance
        for dc in network.data_centers.values():

#Log the commencement of maintenance on the current DataCenter
            logger.info("Performing maintenance on DataCenter: %s", dc.name)

#Simulate maintenance by updating the DataCenter load with a random small value
            dc.update_load(random.randint(1, 10))
```

КОПЗ_2025

```
#Record the completion of maintenance in the DataCenter's event history
            dc.log_event("Maintenance completed")
#Pause briefly to simulate the time taken for maintenance on each DataCenter
            time.sleep(0.1)

#Function to dynamically scale network resources based on the current load
across DataCenters
def dynamic_scaling(network):
#Log the start of dynamic scaling evaluation for the network
    logger.info("Starting dynamic scaling of network resources")
#Initialize a variable to accumulate the total load across the network
    total_load = 0
#Acquire the network lock to safely iterate over DataCenters
    with network.lock:
#Loop through each DataCenter in the network dictionary to sum up their load
values
        for dc in network.data_centers.values():
#Add the current DataCenter's load to the total load
            total_load += dc.load
#Calculate the average load across all DataCenters in the network
        average_load = total_load / len(network.data_centers) if
network.data_centers else 0
#Log the computed average load value
        logger.info("Average load across network: %.2f", average_load)

#Check if the average load exceeds a defined threshold indicating high network
usage
    if average_load > 70:
#Log that the network is under high load and scaling is necessary
        logger.info("High average load detected, scaling up resources")
#Determine a new unique id for the additional DataCenter
        new_id = max(network.data_centers.keys()) + 1 if network.data_centers
else 1
#Instantiate a new DataCenter with generated parameters
        new_dc = DataCenter(new_id, "DataCenter_{}".format(new_id),
"192.168.1.{}".format(new_id))

#Add the newly created DataCenter to the network
        network.add_data_center(new_dc)
#Log the addition of the new DataCenter due to scaling requirements
        logger.info("Added new DataCenter: %s due to high load", new_dc.name)
    else:
#Log that the current network load is within acceptable limits and no scaling is
performed
        logger.info("Network load is within acceptable limits, no scaling
performed")

#Function to simulate data replication across all DataCenters in the network
def simulate_data_replication(network, replication_factor):

#Log the start of the data replication process with the specified replication
factor
    logger.info("Starting data replication with factor %d", replication_factor)
#Acquire the network lock to ensure safe replication operations
    with network.lock:

#Loop through each DataCenter in the network for data replication
        for dc in network.data_centers.values():

#Perform replication for the defined number of times per DataCenter
            for r in range(replication_factor):
```

KCTB_2025

```
#Log each replication step for the current DataCenter
                logger.info("Replicating data for %s: replication %d", dc.name,
r + 1)
#Record the completion of each replication step in the DataCenter's event
history
                dc.log_event("Replication {} completed".format(r + 1))
#Pause briefly to simulate the time taken for each replication step
                time.sleep(0.05)
#Log the completion of the data replication process
    logger.info("Data replication completed")

#Function to check the integrity of the network by verifying the status of all
DataCenters
def check_network_integrity(network):
#Log the commencement of network integrity verification
    logger.info("Checking network integrity")

#Initialize an empty list to collect any DataCenters with issues
    issues = []
#Acquire the network lock to safely iterate over DataCenters
    with network.lock:

#Loop through each DataCenter in the network dictionary
        for dc in network.data_centers.values():
#If the DataCenter status is not Active, record it as an issue
            if dc.status != "Active":
                issues.append(dc.name)

#Check if any issues were found and log accordingly
    if issues:
        logger.warning("Integrity issues found in: %s", issues)
    else:
        logger.info("No integrity issues detected")
#Return the list of DataCenters with integrity issues
    return issues

#Main function to initialize the network, add DataCenters, and run all
simulations
def main():

#Log the start of the DataCenter Network Simulation
    logger.info("Starting DataCenter Network Simulation")
#Create an instance of the DataCenterNetwork
    network = DataCenterNetwork()

#Add initial DataCenters to the network in a loop
    for i in range(1, 6):
#Instantiate a new DataCenter with a unique id, name, and IP address
        dc = DataCenter(i, "DataCenter_{}".format(i), "192.168.1.{}".format(i))
#Add the newly created DataCenter to the network
        network.add_data_center(dc)

#Log the initialization of the DataCenter
        logger.info("Initialized DataCenter: %s", dc.name)
#Extend the network with additional DataCenters in a second loop
    for i in range(6, 11):
#Instantiate a new DataCenter for network extension
        dc = DataCenter(i, "DataCenter_{}".format(i), "192.168.1.{}".format(i))

#Set an initial random load for the extended DataCenter
        dc.update_load(random.randint(5, 50))
```

КОПЗ - 2025

```
#Add the extended DataCenter to the network
        network.add_data_center(dc)
#Log the extension of the network with the new DataCenter
        logger.info("Extended network with DataCenter: %s", dc.name)
#Create and start a thread to simulate external queries arriving at the network
    external_thread = threading.Thread(target=simulate_external_queries,
args=(network, 20))
    external_thread.start()

#Create and start a thread to simulate internal network traffic
    traffic_thread = threading.Thread(target=network.simulate_traffic,
args=(30,))
    traffic_thread.start()

#Create and start a thread to perform scheduled maintenance on the DataCenters
    maintenance_thread = threading.Thread(target=scheduled_maintenance,
args=(network,))
    maintenance_thread.start()
#Wait for the external queries thread to complete its execution
    external_thread.join()

#Wait for the network traffic simulation thread to complete its execution
    traffic_thread.join()
#Wait for the scheduled maintenance thread to complete its execution
    maintenance_thread.join()

#Simulate dynamic scaling based on current network load
    dynamic_scaling(network)
#Simulate data replication across the network with a replication factor of 3
    simulate_data_replication(network, 3)
#Check the overall integrity of the network after all operations
    issues = check_network_integrity(network)

#Monitor the final status of all DataCenters in the network
    final_status = network.monitor_network()
#Log the final network status report
    logger.info("Final Network Status: %s", final_status)

#Retrieve the complete history of network events and queries
    history = network.get_network_history()

#Log the network history for review
    logger.info("Network Query History: %s", history)

#Log the completion of the DataCenter Network Simulation
    logger.info("DataCenter Network Simulation Completed")


#Entry point check to ensure main function runs when script is executed
if __name__ == "__main__":
#Invoke the main function to start the simulation process
    main()
#End of source code for DataCenter interconnection system
```

RGTB-2025

Файл check_status.py

```
import threading
import time
import random
import logging
from flask import Flask, render_template_string, request, jsonify, redirect,
url_for, session
import numpy as np

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

class DataCenter:
    def __init__(self, dc_id, name, ip):
        self.id = dc_id
        self.name = name
        self.ip = ip
        self.load = random.randint(1, 50)
        self.status = "Active"
        self.history = []

    def update_load(self, new_load):
        self.load = new_load
        self.history.append(("load", new_load, time.ctime()))
        return self.load

    def check_status(self):
        self.history.append(("status", self.status, time.ctime()))
        return self.status

    def process_query(self, query):
        time.sleep(random.uniform(0.1, 0.3))
        if random.random() < 0.9:
            result = "Processed: " + query
            self.history.append(("query", query, "success", time.ctime()))
            return result
        else:
            self.history.append(("query", query, "failure", time.ctime()))
            return "Error"

    def get_history(self):
        return self.history

class DataCenterNetwork:
    def __init__(self):
        self.data_centers = {}
        self.lock = threading.Lock()
    def add_data_center(self, dc):
        with self.lock:
            self.data_centers[dc.id] = dc
    def remove_data_center(self, dc_id):
        with self.lock:
            if dc_id in self.data_centers:
                del self.data_centers[dc_id]
                return True
            return False

    def get_statuses(self):
        statuses = {}
        with self.lock:
            for dc_id, dc in self.data_centers.items():
```

КБПЗ_2025

```
                statuses[dc_id] = {"name": dc.name, "ip": dc.ip, "load":
dc.load, "status": dc.status}
        return statuses
    def simulate_traffic(self, num_queries):
        for i in range(num_queries):
            with self.lock:
                if not self.data_centers:
                    break
                dc = random.choice(list(self.data_centers.values()))
            query = "query_" + str(i) + "_" + str(random.randint(1, 100))
            dc.process_query(query)
            time.sleep(random.uniform(0.05, 0.2))
    def get_all_history(self):
        all_history = {}
        with self.lock:
            for dc_id, dc in self.data_centers.items():
                all_history[dc_id] = dc.get_history()
        return all_history

class User:
    def __init__(self, username, password, role):
        self.username = username
        self.password = password
        self.role = role

class UserManager:
    def __init__(self):
        self.users = {}
        self.lock = threading.Lock()
        self.create_user("admin", "adminpass", "admin")
        self.create_user("operator", "operatorpass", "operator")
        self.create_user("viewer", "viewerpass", "viewer")

    def create_user(self, username, password, role):
        with self.lock:
            self.users[username] = User(username, password, role)
    def validate_user(self, username, password):
        with self.lock:
            if username in self.users and self.users[username].password ==
password:
                return self.users[username]
            return None
    def change_password(self, username, new_password):
        with self.lock:
            if username in self.users:
                self.users[username].password = new_password
                return True
            return False
    def get_user_role(self, username):
        with self.lock:
            if username in self.users:
                return self.users[username].role
            return None

class AuditLogger:
    def __init__(self, filename):
        self.filename = filename
        self.lock = threading.Lock()
    def log(self, event_type, details):
        with self.lock:
            with open(self.filename, "a") as f:
                f.write(f"{time.ctime()} | {event_type} | {details}\n")
```

КОТЗ_2025

```
    def get_logs(self):
        with self.lock:
            with open(self.filename, "r") as f:
                return f.read()

class LoadPredictor:
    def __init__(self):
        self.history = []
    def add_data_point(self, load):
        self.history.append((time.time(), load))
    def predict(self, future_seconds):
        if len(self.history) < 2:
            return None
        times = np.array([point[0] for point in self.history])
        loads = np.array([point[1] for point in self.history])
        A = np.vstack([times, np.ones(len(times))]).T
        m, c = np.linalg.lstsq(A, loads, rcond=None)[0]
        future_time = time.time() + future_seconds
        prediction = m * future_time + c
        return prediction

app = Flask(__name__)
app.secret_key = "secret_key_for_session"
network = DataCenterNetwork()
user_manager = UserManager()
audit_logger = AuditLogger("audit.log")
load_predictor = LoadPredictor()

for i in range(1, 11):
    dc = DataCenter(i, "DC_" + str(i), "192.168.0." + str(i))
    network.add_data_center(dc)

dashboard_template = """
<!DOCTYPE html>
<html>
<head>
<title>Моніторинг дата-центрів</title>
</head>
<body>
<h1>Статус дата-центрів</h1>
<table border="1">
<tr>
<th>ID</th>
<th>Назва</th>
<th>IP</th>
<th>Навантаження</th>
<th>Статус</th>
</tr>
{% for dc_id, info in statuses.items() %}
<tr>
<td>{{ dc_id }}</td>
<td>{{ info.name }}</td>
<td>{{ info.ip }}</td>
<td>{{ info.load }}</td>
<td>{{ info.status }}</td>
</tr>
{% endfor %}
</table>
<br>
{% if prediction is not none %}
```

КБПЗ_2025

```
<h2>Прогноз навантаження через 60 секунд: {{ prediction }}</h2>
{% endif %}
<a href="/audit">Переглянути аудиторський журнал</a><br>
<a href="/logout">Вихід</a>
</body>
</html>
"""

login_template = """
<!DOCTYPE html>
<html>
<head>
<title>Вхід</title>
</head>
<body>
<h1>Вхід</h1>
<form method="post" action="/login">
Користувач: <input type="text" name="username"><br>
Пароль: <input type="password" name="password"><br>
<input type="submit" value="Увійти">
</form>
</body>
</html>
"""

audit_template = """
<!DOCTYPE html>
<html>
<head>
<title>Аудиторський журнал</title>
</head>
<body>
<h1>Аудиторський журнал</h1>
<pre>{{ logs }}</pre>
<a href="/">Повернутися</a>
</body>
</html>
"""

@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form.get("username")
        password = request.form.get("password")
        user = user_manager.validate_user(username, password)

        if user:
            session["username"] = username
            audit_logger.log("login", f"{username} logged in")
            return redirect(url_for("dashboard"))
        else:
            return redirect(url_for("login"))
    return render_template_string(login_template)

@app.route("/logout")
def logout():
    username = session.get("username", "unknown")
    audit_logger.log("logout", f"{username} logged out")
    session.pop("username", None)
    return redirect(url_for("login"))
```

КбПЗ_2025

```
@app.route("/")
def dashboard():
    if "username" not in session:
        return redirect(url_for("login"))
    statuses = network.get_statuses()
    for dc in network.data_centers.values():
        load_predictor.add_data_point(dc.load)
    prediction = load_predictor.predict(60)
    return render_template_string(dashboard_template, statuses=statuses,
prediction=prediction)

@app.route("/api/status", methods=["GET"])
def api_status():
    if "username" not in session:
        return jsonify({"error": "Unauthorized"}), 401
    return jsonify(network.get_statuses())

@app.route("/api/update_load", methods=["POST"])
def api_update_load():
    if "username" not in session:
        return jsonify({"error": "Unauthorized"}), 401
    data = request.get_json()
    dc_id = data.get("dc_id")
    new_load = data.get("new_load")
    if dc_id in network.data_centers:
        dc = network.data_centers[dc_id]
        dc.update_load(new_load)
        audit_logger.log("update_load", f"DC {dc_id} load updated to {new_load}
by {session['username']}")
        return jsonify({"success": True})
    return jsonify({"error": "DataCenter not found"}), 404

@app.route("/api/replicate", methods=["POST"])
def api_replicate():
    if "username" not in session:
        return jsonify({"error": "Unauthorized"}), 401
    data = request.get_json()
    replication_factor = data.get("replication_factor", 1)
    for dc in network.data_centers.values():
        for i in range(replication_factor):
            dc.history.append(("replication", i+1, time.ctime()))
    audit_logger.log("replication", f"Replication performed with factor
{replication_factor} by {session['username']}")
    return jsonify({"success": True})


@app.route("/audit")
def audit():
    if "username" not in session:
        return redirect(url_for("login"))
    logs = audit_logger.get_logs()
    return render_template_string(audit_template, logs=logs)


@app.route("/api/scale", methods=["POST"])
def api_scale():
    if "username" not in session:
        return jsonify({"error": "Unauthorized"}), 401
    current_loads = [dc.load for dc in network.data_centers.values()]
    avg_load = sum(current_loads) / len(current_loads) if current_loads else 0
    if avg_load > 70:
        new_id = max(network.data_centers.keys()) + 1 if network.data_centers
else 1
```

КОПІЯ 2025

```
        new_dc = DataCenter(new_id, "DC_" + str(new_id), "192.168.0." +
str(new_id))
        network.add_data_center(new_dc)
        audit_logger.log("scale", f"New DataCenter DC_{new_id} added due to high
average load {avg_load}")
        return jsonify({"scaled": True, "new_dc": {"id": new_dc.id, "name":
new_dc.name, "ip": new_dc.ip}})
    return jsonify({"scaled": False, "avg_load": avg_load})

def background_traffic():
    while True:
        network.simulate_traffic(5)
        time.sleep(5)

if __name__ == "__main__":
    threading.Thread(target=background_traffic, daemon=True).start()
    app.run(host="0.0.0.0", port=5000)
```

КбПЗ_2025

Файл basicConfig.py

```
import threading
import time
import random
import logging
import re
import csv
import os
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger("ExtendedSystem")
class DataCenter:
    def __init__(self, dc_id, name, ip):
        self.id = dc_id
        self.name = name
        self.ip = ip
        self.load = random.randint(1,100)
        self.status = "Active"
        self.history = []
    def update_load(self, new_load):
        self.load = new_load
        self.history.append(("update_load", new_load, time.ctime()))
    def set_status(self, new_status):
        self.status = new_status
        self.history.append(("set_status", new_status, time.ctime()))
    def get_status(self):
        return self.status
    def process_query(self, query):
        time.sleep(random.uniform(0.1, 0.3))
        self.history.append(("process_query", query, time.ctime()))
        return "Result from " + self.name + " for query: " + query
class DataCenterNetwork:
    def __init__(self):
        self.data_centers = {}
        self.lock = threading.Lock()
    def add_data_center(self, dc):
        with self.lock:
            self.data_centers[dc.id] = dc
    def remove_data_center(self, dc_id):
        with self.lock:
            if dc_id in self.data_centers:
                del self.data_centers[dc_id]
    def get_all_data_centers(self):
        with self.lock:
            return list(self.data_centers.values())
    def simulate_queries(self, num):
        results = []
        with self.lock:
            dcs = list(self.data_centers.values())
        for i in range(num):
            if not dcs:
                break
            dc = random.choice(dcs)
            query = "simulated_query_" + str(i)
            res = dc.process_query(query)
            results.append((dc.id, res))
        return results
```

ИБП3_2025

```
class FailureRecoveryManager:
    def __init__(self, network):
        self.network = network
        self.recovery_interval = 5
        self.running = True
        self.thread = threading.Thread(target=self.monitor_failures)
        self.thread.daemon = True
        self.thread.start()
    def monitor_failures(self):
        while self.running:
            dcs = self.network.get_all_data_centers()
            for dc in dcs:
                if dc.get_status() != "Active":
                    self.recover(dc)
            time.sleep(self.recovery_interval)
    def recover(self, dc):
        time.sleep(1)
        dc.set_status("Active")
        logger.info("Recovered DataCenter %s", dc.name)
    def stop(self):
        self.running = False
        self.thread.join()

class SecurityThreatDetector:
    def __init__(self):
        self.threat_patterns = [r"DROP\s+TABLE", r";--", r"OR\s+1=1",
r"<script>"]
        self.threat_log = []
    def analyze_query(self, query):
        for pattern in self.threat_patterns:
            if re.search(pattern, query, re.IGNORECASE):
                self.threat_log.append((query, pattern, time.ctime()))
                logger.warning("Threat detected in query: %s matches %s", query,
pattern)
                return True
        return False
    def get_threat_log(self):
        return self.threat_log

class PerformanceAnalytics:
    def __init__(self, network):
        self.network = network
        self.report_data = []
    def collect_data(self):
        dcs = self.network.get_all_data_centers()
        for dc in dcs:
            record = {"id": dc.id, "name": dc.name, "load": dc.load, "status":
dc.status, "timestamp": time.ctime()}
            self.report_data.append(record)
    def generate_report(self, filename="performance_report.csv"):
        fieldnames = ["id", "name", "load", "status", "timestamp"]
        with open(filename, "w", newline="") as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for record in self.report_data:
                writer.writerow(record)
        logger.info("Performance report generated in %s", filename)
    def reset_data(self):
        self.report_data = []


class DynamicIPManager:
    def __init__(self, network):
```

КОПІЯ 2025

```
        self.network = network
        self.ip_pool = set()
        for i in range(1, 255):
            self.ip_pool.add("10.0.0." + str(i))
        self.assigned_ips = {}
    def assign_ip(self, dc):
        if dc.id in self.assigned_ips:
            return self.assigned_ips[dc.id]
        if self.ip_pool:
            ip = self.ip_pool.pop()
            self.assigned_ips[dc.id] = ip
            dc.ip = ip
            logger.info("Assigned IP %s to DataCenter %s", ip, dc.name)
            return ip
        return None
    def release_ip(self, dc):
        if dc.id in self.assigned_ips:
            ip = self.assigned_ips.pop(dc.id)
            self.ip_pool.add(ip)
            dc.ip = "0.0.0.0"
            logger.info("Released IP %s from DataCenter %s", ip, dc.name)
    def reassign_ip(self, dc):
        self.release_ip(dc)
        return self.assign_ip(dc)
    def update_all_ips(self):
        dcs = self.network.get_all_data_centers()
        for dc in dcs:
            self.assign_ip(dc)

class NetworkDelaySimulator:
    def __init__(self, loss_probability=0.1, min_delay=0.05, max_delay=0.2):
        self.loss_probability = loss_probability
        self.min_delay = min_delay
        self.max_delay = max_delay
    def simulate(self, func, *args, **kwargs):
        delay = random.uniform(self.min_delay, self.max_delay)
        time.sleep(delay)
        if random.random() < self.loss_probability:
            logger.warning("Packet loss simulated")
            return None
        return func(*args, **kwargs)

def main():
    network = DataCenterNetwork()
    for i in range(1, 11):
        dc = DataCenter(i, "DC_" + str(i), "10.0.0." + str(i))
        network.add_data_center(dc)
    ip_manager = DynamicIPManager(network)
    ip_manager.update_all_ips()
    failure_manager = FailureRecoveryManager(network)
    threat_detector = SecurityThreatDetector()
    analytics = PerformanceAnalytics(network)
    delay_simulator = NetworkDelaySimulator()
    def simulate_failures():
        while True:
            dcs = network.get_all_data_centers()
            if dcs:
                dc = random.choice(dcs)
                if random.random() < 0.2:
                    dc.set_status("Failed")
                    logger.info("DataCenter %s has failed", dc.name)
            time.sleep(3)
```

RGTB_2025

```
    failure_thread = threading.Thread(target=simulate_failures)
    failure_thread.daemon = True
    failure_thread.start()
    def simulate_queries():
        while True:
            queries = ["SELECT * FROM users", "DROP TABLE students;--", "Normal
query", "OR 1=1", "Regular operation", "<script>alert('xss')</script>"]
            query = random.choice(queries)
            if not threat_detector.analyze_query(query):
                result = delay_simulator.simulate(lambda q:
random.choice(["Success", "Timeout", "Error"]), query)
                logger.info("Query: %s, Result: %s", query, result)
            time.sleep(2)
    query_thread = threading.Thread(target=simulate_queries)
    query_thread.daemon = True
    query_thread.start()
    def generate_reports():
        while True:
            analytics.reset_data()
            analytics.collect_data()
            analytics.generate_report("performance_report_" +
str(int(time.time())) + ".csv")
            time.sleep(10)
    report_thread = threading.Thread(target=generate_reports)
    report_thread.daemon = True
    report_thread.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        failure_manager.stop()
        logger.info("Shutting down system")
if __name__ == "__main__":
    main()
```

КбПЗ_2025