

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
“___” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи балансування навантаження
у хмарі на основі Application Load Balancer та Network Load
Balancer”

Виконав здобувач вищої освіти
IV курсу, групи КІ-23-МБ
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Костюков В.С.
«___» _____ 2026 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Якименко Н.М.
«___» _____ 2026 р.
Рецензент _____

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
« 27 » лютого 2026 року

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

Костюкову Валентину Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer*

2. Керівник роботи *Якименко Наталія Миколаївна, кандидат фізико-математичних наук, доцент*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 126-02 від 27.02.2026 року

3. Строк подання студентом роботи до захисту 23.05.2026 р.

4. Мета та завдання випускної кваліфікаційної роботи: *Метою роботи є розробка програмного забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer*

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

7. Дата видачі завдання « 27 » лютого 2026 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	10.03.2026 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2026 р.	
3.	Розробка моделі компонента	20.03.2026 р.	
4.	Розробка структур даних	25.03.2026 р.	
5.	Розробка алгоритмів зв'язку та відображення	30.03.2026 р.	
6.	Програмування алгоритмів	10.04.2026 р.	
7.	Оформлення ПЗ	17.04.2026 р.	
8.	Попередній захист роботи	23.05.2026 р.	

Дата видачі завдання
« 27 » лютого 2026 р.

Підпис керівника

Якименко Н.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 27 » лютого 2026 р.

Підпис здобувача

Костюков В.С.
(прізвище та ініціали)

АНОТАЦІЯ

Костюков В.С. Програмне забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2026.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

Метою розробки є програмне забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

Результат роботи – програмна реалізація системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

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, Application Load Balancer, Network Load Balancer

ABSTRACT

Kostiukov V.S. Software for a cloud load balancing system based on Application Load Balancer and Network Load Balancer. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for a cloud load balancing system based on Application Load Balancer and Network Load Balancer.

The purpose of the development is software for a cloud load balancing system based on Application Load Balancer and Network Load Balancer.

The result of the work is a software implementation of a cloud load balancing system based on Application Load Balancer and Network Load Balancer.

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 software are provided.

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

The program is developed in the Python environment.

Keywords: computer engineering, Application Load Balancer, Network Load Balancer

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	9
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.....	9
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування.....	23
2.3 Розгорнута постановка завдання	24
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ	25
3.1 Опис функціонування системи	25
3.2 Розробка структурної схеми.....	39
3.3 Розробка функціональної схеми	46
3.4 Розробка діаграми процесів.....	55
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ.....	58
4.1 Розробка блок-схем та опис алгоритмів функціонування системи.....	58
4.2 Захист розробленого програмного забезпечення.....	72
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ	77
6 ОСНОВНІ ВИСНОВКИ.....	80
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	82

					ВКРБ-123.26.0031.00.00.ПЗ			
Вим.	Арк.	№ докум.	Підп.	Дата	Програмне забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer	Лім.	Аркуш	Аркушів
Розроб.	Костюков В.С.					Б	1	89
Перев.	Якименко Н.М.							
Н.контр.	Коваленко А.С.					ЦНТУ КІ-23-МБ		
Затв.	Смірнов О.А.							

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

EOM	–	електронно-обчислювальна машина
ALB	–	Application Load Balancer
ATM	–	Asynchronous Transfer Mode – асинхронний спосіб передачі даних
AWS	–	Amazon Web Services
BER	–	Bit Error Rate – імовірність ушкодження одного біта
CBWFQ	–	class based weighted fair queueing
DiffServ	–	Differentiated Services – механізм який залежно від вимог до якості обслуговування записує в IP заголовки пакетів спеціальні мітки
DSCP	–	DiffServ CoSde Point
ICMP	–	Internet CoSntrol Message Protocol – міжмережевий протокол управляючих повідомлень
ISP	–	Internet Service Provider – провайдер
LLQ	–	low latency queueing
MPLS-TE	–	MultIProtocol Label Switching Traffic Engineering
NLB	–	Network Load Balancer
PQ	–	priority queueing
RIO	–	RED with Input Output
RTT	–	Round TrIP Time – час між відправкою запиту та отриманням відповіді
QoS	–	Quality of Service – якість обслуговування
WFQ	–	Weighted Fair Queuing – механізм планування пакетних потоків даних
WRED	–	Weighted random early detection – взвішане значення довжини черги, у якості фактора, визначаючого імовірність відкидання пакета

ВСТУП

Актуальність теми. Application Load Balancer (ALB) і Network Load Balancer (NLB) – це балансувальники навантаження другого покоління від комерційної публічної хмарної платформи Amazon Web Services (AWS). Розуміння того, на що здатні ці нові продукти і як вони працюють, дозволить вам повноцінно скористатися всіма перевагами їх нової функціональності. Балансувальник навантаження ухвалює вхідний мережевий трафік від клієнта й, ґрунтуючись на деяких критеріях цього трафіку, відправляє ці повідомлення на один з декількох бекенд-серверів. В офіційній документації хмарної платформи AWS для Application Load Balancer і Network Load Balancer застосунок ALB називається балансувальником навантаження «рівня 7», а NLB іменують балансувальником навантаження «рівня 4». Ці рівні, як можна легко догадатися, є прямим відсиланням до мережевої моделі OSI. Таким чином, знання того шляху, яке мережеве з'єднання проходить через мережу, дає нам розуміння, що балансувальник навантаження – це щось більше, ніж просто пари ліній між клієнтом і балансувальником навантаження, а також між балансувальником навантаження й бекенд-сервером. Ці лінії приховують складну мережеву взаємодію. У дійсності, клієнт відправляє своє повідомлення вниз по стеку рівнів мережевої моделі OSI; коли це повідомлення досягає балансувальника навантаження, воно переміщається назад нагору по стеку рівнів; а потім, впливаючи далі по мережевому шляхові від балансувальника навантаження до сервера призначення, повідомлення проходить повторну трансформацію – опускаючись відразу до нижнього фізичного рівня, а потім знову піднімаючись нагору до прикладного рівня.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		3

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

- Огляд існуючих систем балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.
- Дослідження системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.
- Програмна реалізація системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		4

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

«Рівень» балансувальника навантаження безпосередньо вказує на те, до якого рівня мережевої моделі OSI повинне бути трансформоване мережеве повідомлення, отримане по мережі на фізичному рівні, щоб балансувальник навантаження зміг його обробити й відправити назад по мережі до обраного їх кінцевому місцю призначення. Тобто, повідомлення після обробки балансувальником навантаження знову трансформується вниз по стеку до фізичного рівня, а потім проходить зворотне перетворення до прикладного рівня вже на стороні сервера. Інакше кажучи, для того, щоб Application Load Balancer, який є балансувальником навантаження «рівня 7», зміг прочитати HTTP-запит і визначити, куди цей запит повинен бути спрямований, мережеве повідомлення перед цим піддається трансформації до прикладного рівня мережевої моделі OSI. Аналогічним образом, балансувальник навантаження «рівня 4» (у цьому випадку мова йде про Network Load Balancer) зчитує заголовну інформацію, що втримується в TCP-пакеті, щоб визначити, куди направити це повідомлення, але інша інформація, що втримується в цих пакетах, йому для ухвалення рішення не потрібна. Тому для Network Load Balancer потрібна трансформація мережевого повідомлення до транспортного рівня мережевої моделі OSI

1.2 Область застосування

Областю застосування є хмара. Хмара – це віртуальне середовище, у якому можна запускати віртуальні комп'ютери (сервери), до яких забезпечений дистанційний доступ. Фізично, воно складається з апаратної частини (потужних «залізних» комп'ютерів) і віртуалізуючого програмного забезпечення (гіпервізора). Взагалі, «хмара» – не дуже-те комп'ютерний термін. Їм стали

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		5

користуватися, тому що інтернет у схемах комп'ютерних мереж і систем часто позначали й позначають зображенням хмари. Хмара (інтернет) у таких схемах відіграє роль каналу зв'язки, по якому користувачі одержують доступ до якихось комп'ютерів або забезпечуваних ними послугам.

Потім маркетологи придумали вважати хмарою не тільки канал зв'язки, але й комп'ютери з послугами, доступ до яких забезпечується через інтернет.

Далі виникла от яка обставина. Комп'ютерні ресурси повинні бути не тільки доступними, але й надійними.

Бажаний рівень надійності можна забезпечити застосуванням спеціалізованого встаткування (яке як правило, дороге), його дублюванням і резервуванням. Однак відразу виникає питання ефективності використання цього дорогого встаткування.

Справа в тому, що фактичний рівень навантаження на комп'ютерні системи досить складно прогнозувати. Щоб уникнути перевантаження, комп'ютерні системи часто проектують із дуже більшим запасом продуктивності, з надзвичайно надлишковим її запасом. Часом середній рівень фактичного завантаження серверів не перевищує 20%.

Сучасним способом підвищення ефективності використання комп'ютерного встаткування є його віртуалізація.

Історично операційні системи були досить тісно пов'язані з апаратною частиною комп'ютерів, тобто на кожний новий комп'ютер насамперед потрібно було встановити якусь операційну систему. Потім виникла технологія програмної емуляції апаратної частини комп'ютера, яка забезпечила прошарок між «залізом» і операційною системою.

Цей прошарок дозволив переносити операційну систему з одного фізичного комп'ютера на іншій без її переустановки. Головна умова такого переносу – наявність на іншому комп'ютері такої ж або сумісної «прошарку».

Однак «прошарок» дозволяє не тільки переносити операційну систему, але й розміщати на одному потужному «залізному» комп'ютері кілька екземплярів операційних систем (віртуальних машин).

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		6

Це відкриває нові можливості. Відомо, що заміна в якійсь інформаційній системі однієї великої машини декількома невеликими однотипними комп'ютерами може підвищити надійність або продуктивність цієї системи, у цілому.

Тепер ми підійшли до більш точного визначення терміна «хмара».

Хмара – це місце в інтернеті, де користувачам надаються обчислювальні ресурси віртуальних комп'ютерів. Це місце визначається доменними іменем або IP-адресою.

Очевидно, що й зв'язок з віртуальними комп'ютерами здійснюється також через інтернет, у режимі дистанційного доступу.

Таким чином, дві головні відмітні ознаки хмари:

- використання віртуальних комп'ютерів;
- доступ до них через інтернет.

У змісті прикладного використання віртуального комп'ютера його цілком можна сприймати як звичайний «залізний» комп'ютер, який установлений десь далеко, і звертання до якого відбувається по мережі.

Типи хмар

Коли обговорюють комп'ютерні хмари, найчастіше до них додаються прикметники «публічне» або «частка». У технологічному відношенні публічна хмара мало відрізняється від приватної хмари, і навпаки.

Відмінності між ними зводяться до того, хто володіє інфраструктурою, і хто нею управляє.

Публічна хмара

Якщо власник хмари надає можливість усім бажаючим, що й оплатили послугу одержати в цій хмарі один або кілька віртуальних серверів, ця хмара – публічна.

Гарним прикладом публічної хмари служить хмара 1cloud. У ньому реалізований принцип самообслуговування: будь-який зареєстрований користувач може сам – без звернення до служби підтримки – створити потрібне йому число віртуальних серверів або віртуальних мереж.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		7

Приватна хмара

Якщо якесь підприємство, використовуючи хмарні технології, побудувала хмара на своєму встаткуванні для своїх внутрішньокорпоративних потреб, така хмара називається часткам. Устаткування приватної хмари може розміщатися як на «території» його власника, так і в центрі обробки даних (ЦОД).

У технологічному плані приватна хмара – це таке ж хмара, що й публічне, але використовуване винятково усередині одного підприємства або організації.

Порівняння типів хмар

Як уже було відзначено, відмінності між публічним і часткою хмарами, в основному, пов'язані з володінням устаткуванням і зонами відповідальності (керування).

Гібридна інфраструктура

Конкретна інформаційна система може бути побудована з використанням елементів, розташованих як у частці (корпоративному) хмарі, так і в публічній хмарі.

Вона може бути побудована з декількох публічних хмар або з декількох приватних хмар, а також з будь-якої їхньої комбінації.

Принципова ознака гібридної хмарної інфраструктури – застосування в одній інформаційній системі елементів, розташованих у декількох хмарах (неважливо, однакового або різних типів).

Часто гібридною хмарою називають просто одночасне спільне використання хмар різного типу.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		8

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

У роботі системи необхідно періодично контролювати збереження складу серверів і оптимальності їх завантаження, виконуючи для цього опитування значень завантаження процесора, а при істотних розбіжностях у завантаженні (більш 10%) між окремими серверами – виконувати повторне обчислення оптимальних пропорцій. Таке ж дія виконується й у випадках порушень у роботі серверів (відсутність відповідей від агентів SNMP) або підключення нового сервера до системи. При збереженні состава вузлів і несуттєвій розбіжності завантаження змінювати правила балансування не потрібно.

Розробляється програмний модуль-балансувальник, що запускається в якості сервісу на DNS-сервері з підтримкою BIND. У якості вихідних даних модуль балансування одержує ім'я зони, для якої потрібно виконати балансування. Після старту виконується опитування DNS-сервера з метою одержання актуального списку IP-адрес, які зазначені в якості носіїв ресурсів для даної зони, а також поточних правил розподілу запитів. Задається інтервал, достатній для ухвалення рішення про вплив установлених пропорцій розподілу на ступінь завантаженості серверної платформи (за замовчуванням – 10 хвилин). Потім проводиться Snmp-Опитування з метою одержання актуальних відомостей про завантаження серверної платформи.

Обчисливши пропорції в продуктивності вузлів відповідно до раніше наведеного алгоритму, балансувальник формує новий файл конфігурації зони. У даному файлі втримується послідовний список IP-адрес вузлів, який буде використовувати DNS-сервер при циклічному переборі. Пропорції розподілу

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		9

навантаження задаються частотою розташування IP-адреси вузла в цьому списку. Після цього виконуються дії по перезавантаженню файлу зони в DNS-сервері. При наступній роботі балансувальник виконує періодичний Snmp-Опитування серверів і при наявності подій, що вимагають зміни пропорцій у розподілі навантаження, робить дії по їхнім обчисленню, формуванні нового файлу зони і його перезавантаженню. Такими подіями вважаються відсутність Snmp-Відповіді (ознака порушення роботи сервера) або дисбаланс у завантаженні процесора більш 10%. При відсутності подібних подій файл у зоні не обновляється.

Важливий вибір періодичності запитів SNMP від балансувальника. З одного боку, часті запити SNMP дозволяють оперативно відреагувати на ситуації виходу з ладу сервера застосунків або виникнення перекосів у завантаженні, але з іншого боку – обробка SNMP на сервері застосунків починає займати значну частку процесорного часу.

Коли потужності сервера вже не вистачає, встає питання, яким шляхом іти далі. Апгрейд часто не дає пропорційного приросту й до того ж не забезпечує необхідної відказостійкості. Тому самим вірним кроком буде установка другого сервера, який візьме на себе частину навантаження. Залишається вибрати застосунок, який буде забезпечувати балансування.

Із чого вибирати?

Застосунки для балансування мережевого навантаження тільки на перший погляд виглядають однаково. Насправді в процесі бере участь безліч технологій і алгоритмів, тому знайти два однакові продукти неможливо. Неочевидними особливостями може бути підтримка певних протоколів (наприклад, тільки HTTP або будь-які), є й безліч інших параметрів.

Крім того, системи балансування навантаження можуть перенаправляти трафік клієнтів на вибраний сервер декількома способами. Самі популярні: трансляція мережевих адрес (Network Address Translation) і відсилення через шлюз TCP (TCP gateway). У першому випадку балансувальник на льоту підмінює в пакеті IP-адреси, чим приховує IP сервера від клієнта й навпаки. Якщо ж IP

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		10

клієнта потрібно кінцевому застосунку для статистики або будь-яких інших операцій, його звичайно зберігають в HTTP-заголовок X-forwarded-for. При використанні іншого протоколу слід переконатися, що подібна можливість реалізована. У випадку TCP gateway балансувальник уміє управляти трафіком на L4 (транспортному) рівні й навіть на рівні застосунку (L7). Для цього він установлює з'єднання й дивиться усередину пакета. Звичайно клієнт і застосунок обмінюються інформацією через балансувальник. Але останнім часом стає все більш популярною конфігурація сервера із прямим поверненням (Direct Server Return, DSR) коли відповідь від сервера йде до клієнта прямо, а не через пристрій балансування. Використання DSR зменшує навантаження на балансувальник, але не дозволяє використовувати куки й розшифровувати SSL. Даний спосіб на порядок швидше, чим використання NAT-Балансування, і дозволяє сервісам бачити реальні IP-адреси клієнтів.

Також у системах можна зустріти різні методи балансування. Розберемося із призначенням деяких з них. У налаштуваннях продуктів вони можуть мати відмінні назви або свої особливості в реалізації, але часто їх суть одна.

Найпростіший – Round Robin DNS, це спеціальний DNS-сервер, що містить трохи A-записів і їх вага (опціонально), що й видає при запиті клієнтів різні IP-адреси. Мінуси очевидні. Він абсолютно не володіє інформацією про поточне завантаження й стані бекендів, не враховує попередні підключення клієнтів (небагато згладжує ситуацію DNS-кеш).

Є аналогічний за назвою алгоритм, але реалізований засобами самого балансувальника – Round Robin. Усі клієнти рівномірно розподіляються по бекендам, і звичайно які-небудь інші параметри не враховуються. Алгоритм розподілу по вазі (Round Robin Weighted) ураховує значення параметра Weight, зазначеного для кожного сервера. Проставивши більшу вагу для могутнішого сервера, ми зможемо направити до нього більше клієнтів. Трохи інакше діє розподіл по пріоритету. У цьому алгоритмі працює тільки сервер з більшим пріоритетом, інші підключаються, як правило, тільки у випадку його відмови.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		11

Цей варіант дозволяє будувати кластер з одним активним вузлом, наприклад коли другий сервер виконує іншу роль і тільки підстраховує перший. Ще один варіант – Least Connection (Least Session) – з'єднання ухвалює сервер, що обслуговує найменшу кількість з'єднань (сесій), але з'єднання можуть бути різні (користувач активний або пасивний) і, відповідно, давати різне навантаження на сервер. А от алгоритм Least Bandwidth урахує дійсне завантаження мережі.

Hash sticky client – клієнт прив'язується до одному серверу, для цього в спеціальну таблицю міститься геш-рядок, що вказує на його IP. Можливі варіанти. Клієнт завжди йде до одному серверу, а у випадку його виходу підключення неможливе. Або, коли не відповідає «рідний», він з'єднується з іншими системами. Доступність бекендів визначається двома: активний (keepalives, балансувальник сам опитує сервери) і пасивний (In-Band, контролюються поточні з'єднання, відповіді сервісу).

Balanceng

Проект номер один у списку – Balanceng, є розвитком Open Source застосунку Balance, але поширяється вже під подвійною ліцензією (комерційної й безкоштовної Free Basic License). У безкоштовній версії можна підключати один віртуальний сервер і два вузли, чого з урахуванням можливостей досить, щоб без проблем упоратися із середньою, а іноді й великим навантаженням. Являє собою розв'язок для балансування IP-навантаження, що підтримує IPv6, пропонує кілька методів управління вибору бекенда (Round Robin, Random, Weighted Random, Least Session, Least Bandwidth, Hash, Agent і Randomized Agent).

У продукті використаний оригінальний движок, що працює на Layer 2 (Ethernet), балансування ведеться на основі IP-адреси клієнта, без прив'язки до портів працювати може з будь-яким сервісом. Підтримує DNS GSLB (Global Server Load-Balancing) і конфігурацію сервера із прямим поверненням Direct Server Return (DSR), коли відповідь від сервера йде до клієнта прямо, а не через пристрій балансування. Містить агент, що настроюється, перевірки UDP, підтримує VRRP для установки високодоступних конфігурацій на багатьох

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		12

вузлах. Вбудовані механізми дозволяють зробити захвата й збереження пакетів за допомогою rsnr для подальшого дослідження. Передбачено кілька варіантів перевірки працездатності кінцевих систем: агент, ping, TCP Open, скрипт і інші інструменти начебто wget.

```
modules      vrrp,arp,ping,hc,maser,slb,tnat,nat,rt
set          ipforwarding 1
interface 1 {
    name eth0
    access raw
}
interface 2 {
    name eth1
    access raw
}
register     interfaces 1,2
enable      interface 1,2
vrrp        {
    vrid 71
    priority 200
    network 1
}
network     1 {
    name "DMZ Network"
    addr 172.16.1.0
    mask 255.255.255.0
    real 172.16.1.251
    virt 172.16.1.253
    nat outside
    interface 1
}
network     2 {
    name "LN Network"
    addr 10.1.1.0
    mask 255.255.255.0
    real 10.1.1.252
    virt 10.1.1.254
    nat inside
    interface 2
}
```

Рисунок 2.1 – Конфігураційний файл Balanceng

Можливе резервування балансувальника з реплікацією NAT-станів між основним і резервним вузлами, клієнт при перепідключенні приєднує до того ж серверу. Для збереження сесії використовується IP-адреса клієнта й порт призначення. Підтримується Linux bonding. Усі таблиці зберігаються в ОЗП, але вимоги невеликі, для 4 мільйонів сесій досить 512 Мб пам'яті.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		13

Може працювати в Linux (з використанням сокетів API PF_PACKET) і SPARC/Intel Solaris (Streams/DLPI API). Для установки пропонуються rpm (Red Hat RHEL 6 / Centos) і deb (Debian/Ubuntu) пакети й тарбал для інших дистрибутивів. Також доступний готовий образ для віртуальної машини (на базі Ubuntu 8.04), що дозволяє швидко розгорнути потрібну функціональність. Під час наочування будуть показані всі паролі для входу. Агент (bngagent) поставляється з відкритим вихідним кодом і підтримує Linux, Solaris, OS X, HP-UX і інші.

Який-небудь інтерфейс для налаштування не передбачений, усі установки проводяться за допомогою конфігураційного файлу /etc/bng.conf. У принципі, складним його назвати не можна, особливо враховуючи, що на сайті проекту доступно більш десятка готових прикладів, часто потрібно лише вибрати найбільш підходящий і відредагувати під себе.

Нароху

Нароху – балансувальник навантаження й проксі-сервер рівня застосунків Layer7 для TCP і HTTP. Проект починався як дуже простий HTTP-проксі Webroute, але поступово обростив новими можливостями. Особливістю Нароху є використання для регулювання з'єднань cookie і контенту, він вбудовує cookie і перевіряє вміст пакета при підключенні. Аналіз пакетів на Layer7 дозволяє фільтрувати несанкціонований трафік і протоколи. Перевіряючи HTTP-запит за допомогою регулярних виражень, можна задати будь-які правила для вибору сервера, наприклад реалізуючи ACL і геолокацію. Якщо буде потреба обробки IP-клієнта на кінцевому сервері можемо зберегти його в X-forwarded-for. Движок стабільний, безпечний, оптимізований, у результаті Нароху здатний одночасно обробляти велика кількість підключень (20 тисяч у секунду й більш).

Обробкою всіх підключень займається один процес, за рахунок оптимізації й планувальника, що забезпечує велику кількість одночасних з'єднань на дуже високих швидкостях. Такі системи не занадто добре масштабуються на багатопроцесорних системах, але не хворіють блокуваннями й обмеженнями пам'яті. Але тут ми не побачимо просунутих функцій начебто підтримки SSL і

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		14

keep-alive – за твердженням розроблювачів, вони ускладнюють і «утяжеляють» процес. Відказостійкість Наргоху-сервера досягається через використання демона keeralived, перевіряючого його працездатність, синхронізація з резервним сервером (протокол VRRP) забезпечує дублювання.

Надає логування з'єднань і різноманітні звіти (статистика виводиться на окремому порту). Орієнтований у першу чергу на веб-сайти, але використовується й в інших сценаріях, наприклад у якості балансувальника для multi-master серверів Mysql. Підтримується фільтрація користувачів і підключення до того ж серверу (для RDP і HTTP), автентифікація HTTP.

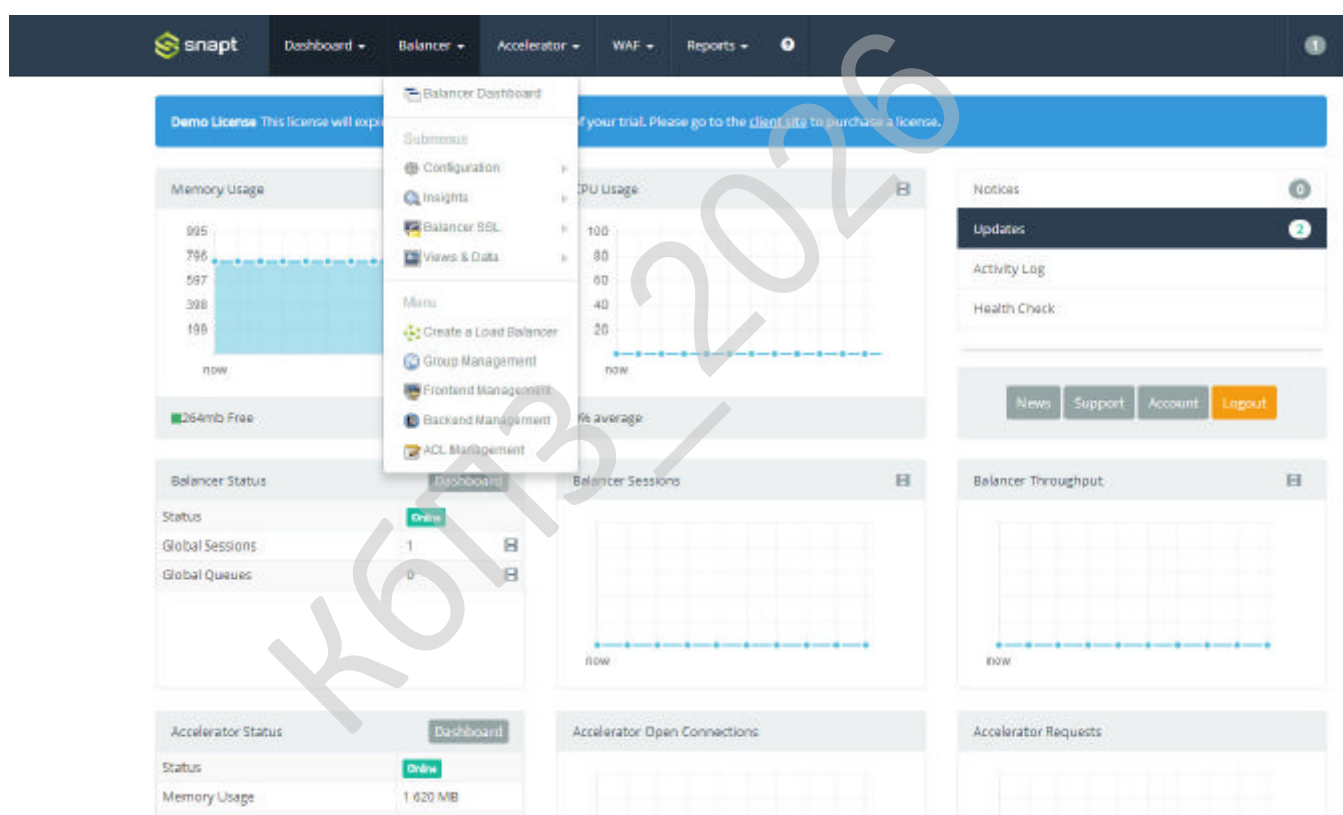


Рисунок 2.2 – Веб-інтерфейс Snapt для налаштування Наргоху

Наргоху у якості балансувальника використовується в декількох компаніях зі списку Fortune 500: Amazon RDS, Github, Stack Overflow, Server Fault, Twitter...

Можливості розширюються за допомогою аддонів, їхній повний список є на сайті. Усі налаштування проводяться за допомогою конфігураційного файлу (на сайті є приклади). Крім того, відомі два інтерфейси сторонніх розроблювачів: комерційний веб Snart і вільний NATop.

Працює на декілька архітектурах x86, x86_64, Alpha, SPARC, MIPS, PARISC. Офіційно підтримує Linux 2.6.32+ (рекомендується для максимальної продуктивності) і 2.4, Solaris 8-10, FreeBSD, OpenBSD. Установка й налаштування тривіальне, хоча пакет у репозиторіях не є присутнім. Проект пропонує вихідний код під ліцензією GPL v2 і готові бінарники під Linux/x86 Glibc 2.2 і Solaris8/Sparc.

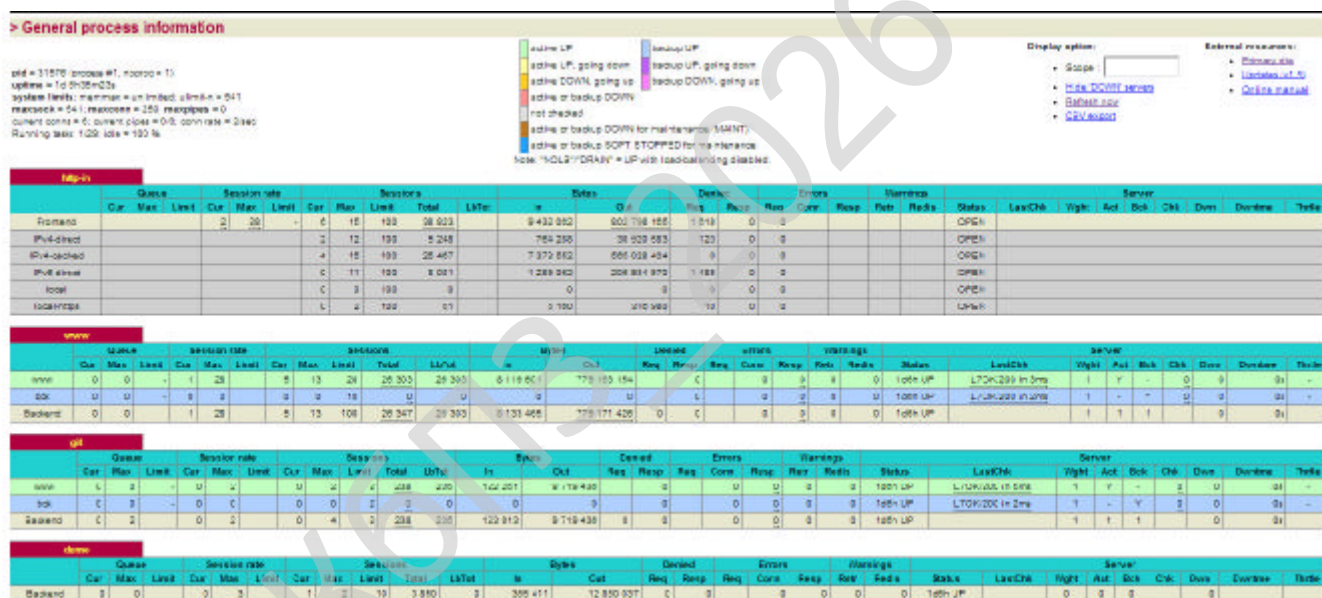


Рисунок 2.3 – Статистика Наргоху

Round – проксі й балансування HTTP і HTTPS

Первісна мета проекту Round – розподіл навантаження між декількома серверами Zore, у підсумку вийшов вузьконаправлений інструмент, що представляє собою зворотний проксі й балансувальник навантаження для HTTP і HTTPS.

Балансування проводиться по стану сесії й іншим параметрам (URL, автентифікації, cookies, HTTP headers). Повна підтримка Webdav. На відміну від Narproxy обробляє SSL. Розроблений в It-компанії безпекою, що займається, що також позначилося на можливостях продукту. Особливістю є наявність базових функцій Web Application Firewall, Pound уміє контролювати коректність заголовків HTTP і HTTPS, відкидаючи неправильні. За замовчуванням пропускаються всі запити, але можна створити шаблони URL і тип запитів (стандартні, розширені, RPC і Webdav), які будуть перевірятися на відповідність. За результатами вибирається кінцевий сервер або з'єднання блокується. Дизайн споконвічно передбачає мінімальне втручання в трафік (наприклад, вбудовування cookie), але може прописувати X-forwarded-for для передачі на бекенд сервера IP-адреси користувача.

Підтримує IPv6, може перекидати IPv6 клієнтів до серверів IPv4. Інформація про сеанс зберігається, і клієнт надалі підключається до свого сервера.

Зі специфіки – можлива не тільки відправлення з'єднання до бекенду, але й редирект на інший URL.

Pound не вимагає багато ресурсів, примітно, що крім як за зчитуванням SSL-сертифікатів демон не звертається до харду. Може бути запущено в chroot і використовувати setuid/setgid. Яких-небудь вбудованих механізмів відказостійкості немає. Перевірка працездатності бекендів проводиться завжди по HTTP.

На процесорі рівня Pentium D дозволяє досягтися приблизно 600–800 HTTP- і 200-300 HTTPs-з'єднань у секунду. Тому ковзан Pound – невеликі проекти з упором на доступність, безпеку й більший контроль над трафіком. Для більш високого навантаження самі розроблювачі Pound рекомендують скористатися іншими застосунками.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		17

```

Терминал - msc[root@exa... Терминал - user@example: ~
Терминал - user@example: ~
Файл Правка Вид Терминал Переход Справка
user@example:~$ cat /etc/pound/pound.cfg
## Minimal sample pound.cfg
##
## see pound(8) for details

#####
## global options:

User          "www-data"
Group         "www-data"
#RootJail     "/chroot/pound"

## Logging: (goes to syslog by default)
##      0      no logging
##      1      normal
##      2      extended
##      3      Apache-style (common log format)
LogLevel      1

## check backend every X secs:
Alive         30

## use hardware-acceleration card supported by openssl(1):
#SSLEngine    "<hw>"

# poundctl control socket
Control       "/var/run/pound/poundctl.socket"

#####
## listen, redirect and ... to:

## redirect all requests on port 8080 ("ListenHTTP") to the local webserver (see "Service" below):
ListenHTTP
    Address 127.0.0.1
    Port    8080

```

Рисунок 2.4 – Конфігураційний файл Pound

Установка й налаштування не представляють більших складностей, хоча й проводяться за допомогою конфігураційних файлів (документація дуже докладна). Офіційно був протестований на Linux, Solaris і OpenBSD. Проект пропонує тільки вихідні тексти, у репозиторіях SUSE, Debian і Ubuntu можна знайти готові пакети, крім цього, на сайті є посилання для Red Hat і готового дистрибутива, зібраного на базі FreeBSD.

Crossroads

Crossroads забезпечує балансування навантаження до будь-яких TCP-сервісам, тому підходить не тільки для HTTP(S), але й для SMTP, SSH, баз даних і інших. У звичайному варіанті він просто гарантує підключення до бекенду, не вникаючи у внутрішності, і забезпечує в цьому режимі максимальну

продуктивність. Один сервер може обслуговувати кілька сайтів з різними бекендами й протоколами. Але є спеціальний режим HTTP mode, при якому обробляються й при необхідності міняються заголовки сеансу. Це забезпечує можливість перенаправку користувача на той же сервер (session stickiness) і збереження IP клієнта в X-Forwarded-For. Сам по собі HTTP mode і особливо модифікація пакета впливає на продуктивність (втрати до 30%).

За замовчуванням клієнт перенаправляється на сервер, що ухвалює менше підключень, але при необхідності алгоритм легко змінити на Round Robin, Least-Connections і First Available або визначитися зовнішньою програмою або скриптом. Клієнтів можна закріплювати за певним сервером (Hash sticky client), жорстко або з можливістю підключення до іншого сервера, якщо «свій» не відповідає. При поновленні роботи бекенда він автоматично включається в роботу. Можливе керування доступом до Crossroads на основі IP-адреси (allow і deny).

У версії 2.x усі підключення обслуговує один процес, що працює в просторі користувача, це позитивно позначилося на швидкості роботи й на керуванні. Може працювати як stand-alone демон або запускатися через inetd. Зручно, що всі налаштування й команди можна віддавати на лету (за допомогою утиліти xr), зміна параметрів не вимагає перезапуску Crossroads. Наприклад, настроїмо балансування для двох веб-сайтів і серверів Mysql.

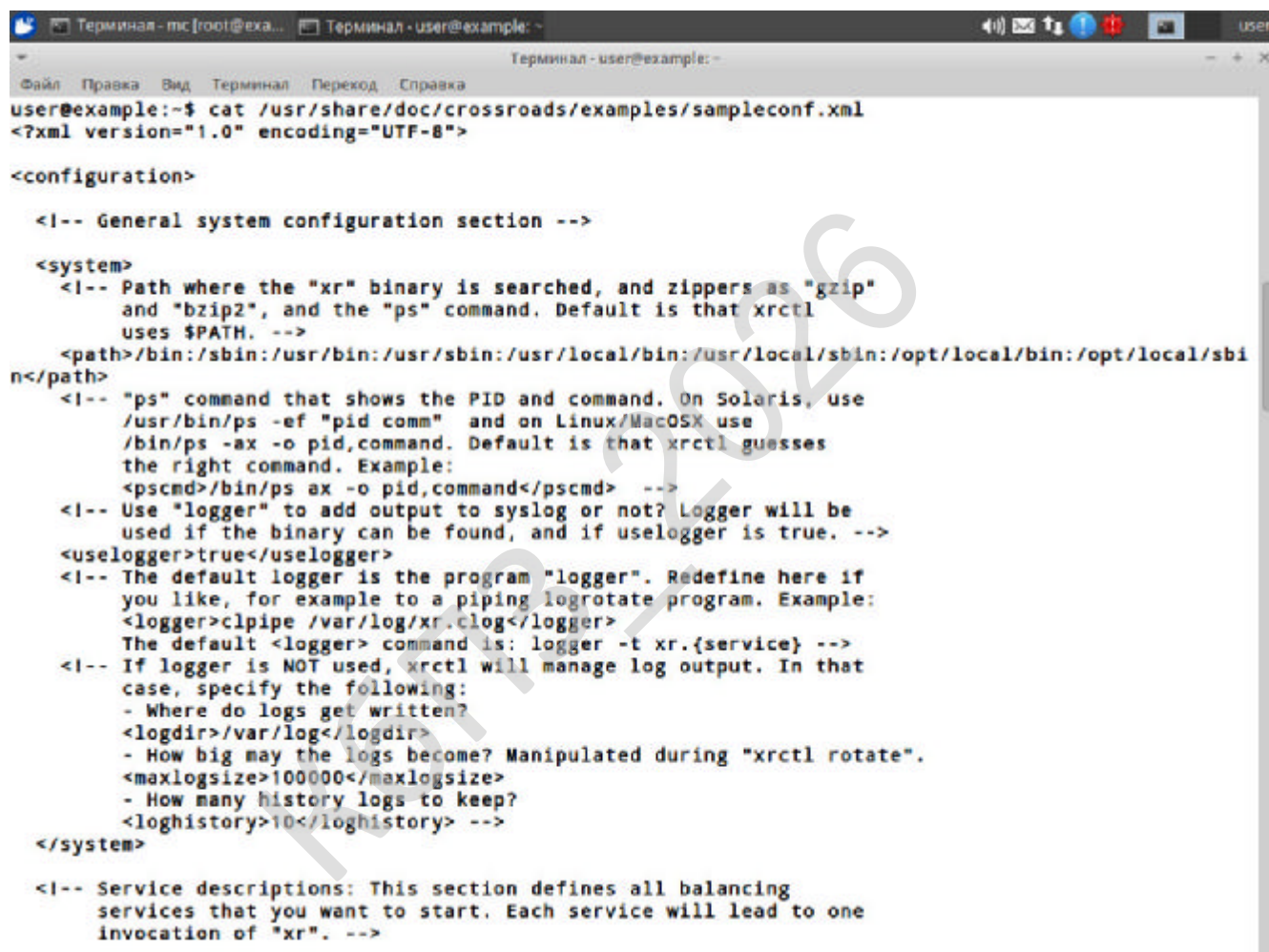
```
# /usr/sbin/xr --verbose -S --server HTTP:0:80 --host-match www.host1.com -  
-backend 192.168.1.10:80 --backend 192.168.1.20:80 --host-match www.host2.com --  
backend 192.168.2.10:80 --backend 192.168.2.20:80 2>&1 >> /var/log/xr.log  
# /usr/sbin/xr --verbose --server TCP:0:3306 --backend 192.168.1.1:3306 --  
backend 192.168.1.2:3306 2>&1 >> /var/log/xr.log &
```

Статистика виводиться за допомогою веб-інтерфейсу, порт якого задається разом із ключем -W (-web-interface). Також з його допомогою можна змінити три параметри: максимальна кількість з'єднань для Crossroads і для бекендів, вага бекендів.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		19

Для зручності всі їх записують в окремий файл, який і вказують при завантаженні, або використовують спеціальний конфігураційний файл у форматі XML (/etc/xrctl.xml, у поставці кілька готових прикладів).

Може бути запущений на будь-якій Posix-системі – Linux, OS X, Solaris. Проект пропонує вихідні тексти (складання проблем не викликає), готові пакети доступні в репозиторіях основних дистрибутивів.



```
user@example:~$ cat /usr/share/doc/crossroads/examples/sampleconf.xml
<?xml version="1.0" encoding="UTF-8">

<configuration>

  <!-- General system configuration section -->

  <system>
    <!-- Path where the "xr" binary is searched, and zippers as "gzip"
    and "bzip2", and the "ps" command. Default is that xrctl
    uses $PATH. -->
    <path>/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/opt/local/bin:/opt/local/sbin</path>
    <!-- "ps" command that shows the PID and command. On Solaris, use
    /usr/bin/ps -ef "pid comm" and on Linux/MacOSX use
    /bin/ps -ax -o pid,command. Default is that xrctl guesses
    the right command. Example:
    <pscmd>/bin/ps ax -o pid,command</pscmd> -->
    <!-- Use "logger" to add output to syslog or not? Logger will be
    used if the binary can be found, and if uselogger is true. -->
    <uselogger>true</uselogger>
    <!-- The default logger is the program "logger". Redefine here if
    you like, for example to a piping logrotate program. Example:
    <logger>clpipe /var/log/xr.clog</logger>
    The default <logger> command is: logger -t xr.{service} -->
    <!-- If logger is NOT used, xrctl will manage log output. In that
    case, specify the following:
    - Where do logs get written?
    <logdir>/var/log</logdir>
    - How big may the logs become? Manipulated during "xrctl rotate".
    <maxlogsize>100000</maxlogsize>
    - How many history logs to keep?
    <loghistory>10</loghistory> -->
  </system>

  <!-- Service descriptions: This section defines all balancing
  services that you want to start. Each service will lead to one
  invocation of "xr". -->
```

Рисунок 2.5 – У поставці Crossroads кілька готових конфігураційних файлів

Zen Load Balancer

Застосунок, що трохи відрізняється від інших учасників огляду. Розроблювачі справедливо вирішили, що під балансування виділяється окремий

сервер, а тому краще відразу надати готовий дистрибутив, щоб спростити розгортання, підвищити продуктивність і безпека. Основою Zen Load Balancer є оптимізована й урізана версія Debian 6. Підтримується балансування на Layer 4 для протоколів TCP, UDP і на Layer 7 для HTTP/HTTPS. Спеціальний режим L4xNAT дозволяє настроїти балансування на транспортному рівні без обліку номерів портів (фактично без прив'язки до сервісу), у цьому випадку Zen просто пропускає через себе весь трафік, розподіляючи його по серверах. Є в Zen і фішка – режим DATALINK. У цьому випадку Zen виступає в якості шлюзу за замовчуванням, забезпечуючи рівномірне навантаження на канал і резервування при підключенні до декількох провайдерів (балансування на Layer 3).

Реалізовано кілька алгоритмів вибору бекенда: Round Robin, по вазі (Weight), по пріоритету (Priority) або підключення до певного сервера (Hash sticky client). Можливе повернення користувача у відкриту сесію, він визначається декількома способами (по IP-адресі, cookie, запитуваному URL, по заголовкові HTTP). Передбачене збереження IP в X-forwarded-for.

Технологія Farmguardian дозволяє визначати доступність сервісів за допомогою спеціального скрипта й розподіляти по них підключення. Zen може виконувати функцію SSL-проксі, шифруючи потік до клієнта, на ділянці Zen – бекенд дані йдуть у відкритому виді. Підтримується VLAN.

З інших особливостей можна назвати просту систему резервування й відновлення конфігурації, роботу декількох балансувальників у кластері, систему моніторингу й виводу різної статистики, у тому числі й у вигляді наочних графіків.

І головне – усі налаштування проводяться за допомогою веб-інтерфейсу (за замовчуванням на HTTPS/444, логін/пароль – admin/admin). Пропонується дві версії. Безкоштовна тільки в x 86-варіанті й без технічної підтримки. Її можливостей цілком вистачає для більшості мереж середнього розміру. Платна версія надається вже в x 64-складанню. Розгорнути готову систему дуже просто, на сайті доступна вся необхідна інформація з налаштувань (на англійському).

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		21



Рисунок 2.6 – Усі налаштування Zen Load Balancer проводяться через веб-інтерфейс

Висновок

Як бачимо, вибрати є із чого. Кожний продукт має свої сильні й слабкі сторони, знаючи які легко визначитися, який з них підходить для певної ситуації. Так, Нарошу підкуповує продуктивністю, Balanceng – функціональністю, простотою в налаштуваннях. Round відрізняється можливістю розбирання HTTP, а Crossroads – гнучкістю. Якщо все-таки потрібний веб-інтерфейс, то альтернативи Zen Load Balancer ні, ну якщо тільки не хочеться платити за Snart для Нарошу.

Хмарні сервіси для балансування навантаження

З ростом популярності хмарних сервісів усе більше систем розміщуються на зовнішніх майданчиках, утворюючи гібридні хмари. Більшість розроблювачів пропонує до всього іншого й можливість балансування навантаження, це дуже зручно, тому що не вимагає від клієнта установки додаткового встаткування й готове до застосування майже відразу. Хоча можливостей по керуванню такий

спосіб дає, як правило, менше. Усі основні гравці вже запропонували свої розв'язки. Наприклад, Amazon Route 53, що є частиною Amazon Web Services, що й надає балансування за допомогою Round Robin DNS. Вартість послуги невелика й становить один долар на місяць, плюс 0,50 долара за перший мільйон запитів і 0,25 за кожний наступний. Передбачений зручний API для редагування, додавання й видалення записів.

Також потрібна функціональність доступна в спеціалізованих хмарних продуктах – WAF або застосунках для захисту від DDoS. Наприклад, сервіс Akamai Global Traffic Management Akamai.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – це потужна мова програмування, яка проста у вивченні. Він має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис і динамічна типізація Python разом з його інтерпретованим характером роблять його ідеальною мовою для створення сценаріїв і швидкої розробки додатків у багатьох сферах на більшості платформ.

Інтерпретатор Python і обширна стандартна бібліотека доступні у вихідному або двійковому вигляді для всіх основних платформ на веб-сайті Python <https://www.python.org/> і можуть вільно поширюватися. Цей же сайт також містить дистрибутиви та вказівники на багато безкоштовних сторонніх модулів Python, програм і інструментів, а також додаткову документацію.

Інтерпретатор Python легко розширюється за допомогою нових функцій і типів даних, реалізованих у C або C++ (або інших мовах, які можна викликати з C). Python також підходить як мова розширення для налаштовуваних програм.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		23

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускню кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

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;

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) сформулювати висновки про виконаний обсяг робіт та одержані результати.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		24

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Як працюють балансувальники навантаження Application Load Balancer і Network Load Balancer в Amazon Web Services (AWS)

Application Load Balancer (ALB) і Network Load Balancer (NLB) – це балансувальники навантаження другого покоління від комерційної публічної хмарної платформи Amazon Web Services (AWS). Розуміння того, на що здатні ці нові продукти і як вони працюють, дозволить вам повноцінно скористатися всіма перевагами їх нової функціональності.

Що таке балансувальник навантаження?

Балансувальник навантаження ухвалює вхідний мережевий трафік від клієнта й, ґрунтуючись на деяких критеріях цього трафіку, відправляє ці повідомлення на один з декількох бекенд-серверів (дивитесь Рисунок 3.1).

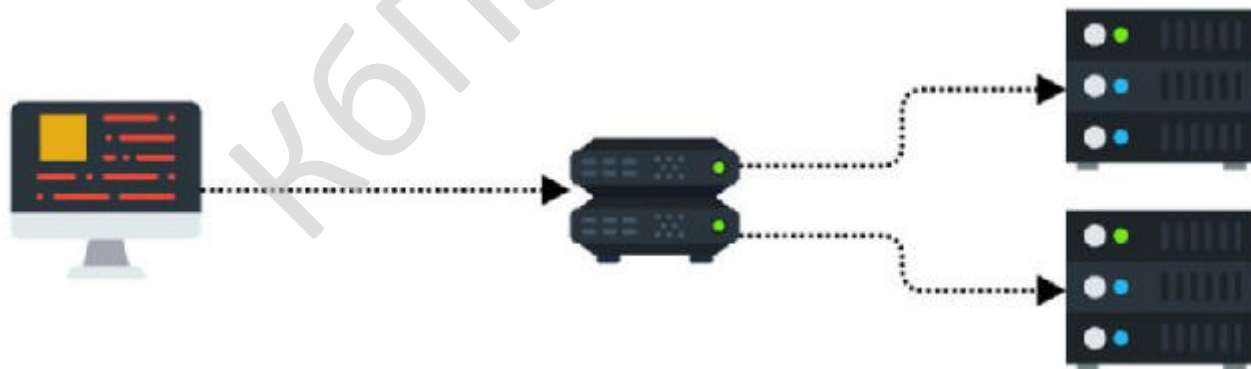


Рисунок 3.1 – Балансувальники навантаження розподіляють вхідний мережевий трафік від клієнта між декількома бекенд-серверами

Балансувальники навантаження є ключовим елементом для створення чудових сучасних інтернет-застосунків, оскільки вони забезпечать вашому застосунку наступні переваги:

– Резервування. Сервер застосунків може вийти з ладу, але доти, поки залишається хоча б один сервер застосунків, балансувальник навантаження може направляти трафік клієнта на, що залишилися в робочому стані сервера застосунків.

– Масштабованість. Так, наприклад, два сервери, що працюють за балансувальником навантаження, дозволять обробляти вдвічі більший трафік від клієнтів. Інакше кажучи, у міру збільшення вашого мережевого трафіку, балансувальники навантаження дозволять вам легко додавати усе більше й більше бекенд-серверів.

Відзначимо також, що при виводі на ринок нових продуктів другого покоління компанія AWS не забрала балансувальник навантаження минулого покоління із продуктової лінійки, а перейменувала його в Classic Load Balancer, і позиціонує його для роботи з додатками, які були побудовані в мережі EC2 Classic.

У чому різниця між Application Load Balancer і Network Load Balancer?

В офіційній документації хмарної платформи AWS для Application Load Balancer і Network Load Balancer застосунок ALB називається балансувальником навантаження «рівня 7», а NLB іменують балансувальником навантаження «рівня 4». Ці рівні, як можна легко догадатися, є прямим відсиланням до мережевої моделі OSI (детальніше проілюстровано на рисунку 3.2).

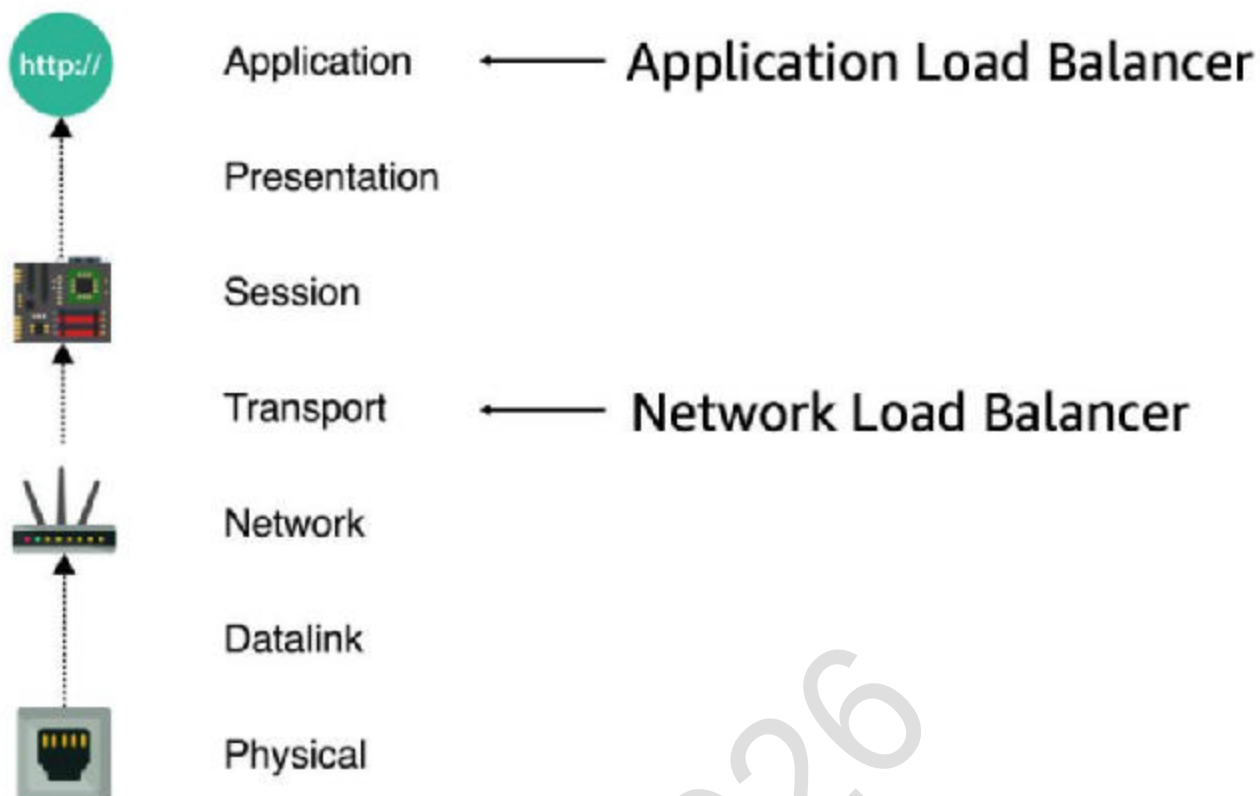


Рисунок 3.2 – Application Load Balancer є балансувальником навантаження «рівня 7», а Network Load Balancer – балансувальником навантаження «рівня 4»

Модель OSI класифікує різні операції, пов'язані з мережевим обміном даними між однією комп'ютерною програмою на одному комп'ютері й іншої комп'ютерною програмою на іншому комп'ютері. Повне пояснення мережевої моделі OSI може зайняти цілу книгу, але нижче наведена ілюстрація (дивитися Рисунок 3.3) може наочно продемонструвати рух мережевого трафіку вниз і нагору по мережевій моделі OSI. Ця загальновідома інформація допоможе нам краще зрозуміти, які трансформації відбуваються з мережевим трафіком, коли він досягає балансувальника навантаження.

Якщо не вдаватися в деталі, то з'єднання між клієнтом і сервером відбувається в такий спосіб:

- На прикладному рівні веб-браузер створює HTTP-запит, що представляє собою невеликий текстовий документ, що описує, який ресурс браузер прагне одержати з веб-сервера.

- Для забезпечення безпечного з'єднання між браузером і сервером веб-запит шифрується з використанням криптографічного протоколу транспортного рівня TLS (або SSL). Ці процес використовує відкрит ключ сервер для перетворення корисного навантаження (payload) HTTP у, що нечитасьмий фрагмент зашифрованих двійкового коду.

- Декількома рівнями нижче, на транспортному рівні, зашифроване корисне навантаження розділяється на пакети TCP. Кожний пакет є частиною корисного навантаження HTTP, «обгорненої» у метадані, такі як вихідний IP-адреса, звідки був відправлений TCP-пакет, і IP-адреса призначення, куди повинен бути доставлений цей пакет.

- Фізичний рівень ухвалює неопрацьовані цифрові одинички й нулі, які є TCP-пакетами у двійковій виставі, і перетворює їх в аналоговий сигнал, такий як електричний імпульс на мідному проведенні, світловий імпульс в оптоволоконному кабелі або радіохвиля в повітрі. На іншому кінці з'єднання інший пристрій перетворює цей аналоговий сигнал назад у цифрові «1» і «0».

- Мережевий трафік починає свій шлях назад по мережевій моделі OSI, коли одинички й нулі інтерпретуються назад у TCP-пакети, які потім знову збираються у вихідне корисне навантаження зашифрованих даних.

- Сервер використовує свій закритий ключ для дешифрування зашифрованої за допомогою TLS (або SSL) корисного навантаження назад у вихідний текстовий документ HTTP-запиту.

- Сервер використовує незашифрований HTTP-запит, щоб визначити, який ресурс необхідно відправити назад по мережі.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		29

Таким чином, знання того шляху, яке мережеве з'єднання проходить через мережу, дає нам розуміння, що балансувальник навантаження – це щось більше, ніж просто пари ліній між клієнтом і балансувальником навантаження, а також між балансувальником навантаження й бекенд-сервером, як це було показано на спрощеній схемі роботи балансувальника навантаження на рисунку 3.1. Ці лінії приховують складне мережеве взаємодії. У дійсності, клієнт відправляє своє повідомлення вниз по стеку рівнів мережевої моделі OSI; коли це повідомлення досягає балансувальника навантаження, воно переміщається назад нагору по стеку рівнів; а потім, впливаючи далі по мережевому шляхові від балансувальника навантаження до сервера призначення, повідомлення проходить повторну трансформацію – опускаючись відразу до нижнього фізичного рівня, а потім знову піднімаючись нагору до прикладного рівня.

Властиво, «рівень» балансувальника навантаження безпосередньо вказує на те, до якого рівня мережевої моделі OSI повинне бути трансформоване мережеве повідомлення, отримане по мережі на фізичному рівні, щоб балансувальник навантаження зміг його обробити й відправити назад по мережі до обраного їх кінцевому місцю призначення. Тобто, повідомлення після обробки балансувальником навантаження знову трансформується вниз по стеку до фізичного рівня, а потім проходить зворотне перетворення до прикладного рівня вже на стороні сервера.

Інакше кажучи, для того, щоб Application Load Balancer, який є балансувальником навантаження «рівня 7», зміг прочитати HTTP-запит і визначити, куди цей запит повинен бути спрямований, мережеве повідомлення перед цим піддається трансформації до прикладного рівня мережевої моделі OSI (детальніше дивитися на рисунку 3.4).

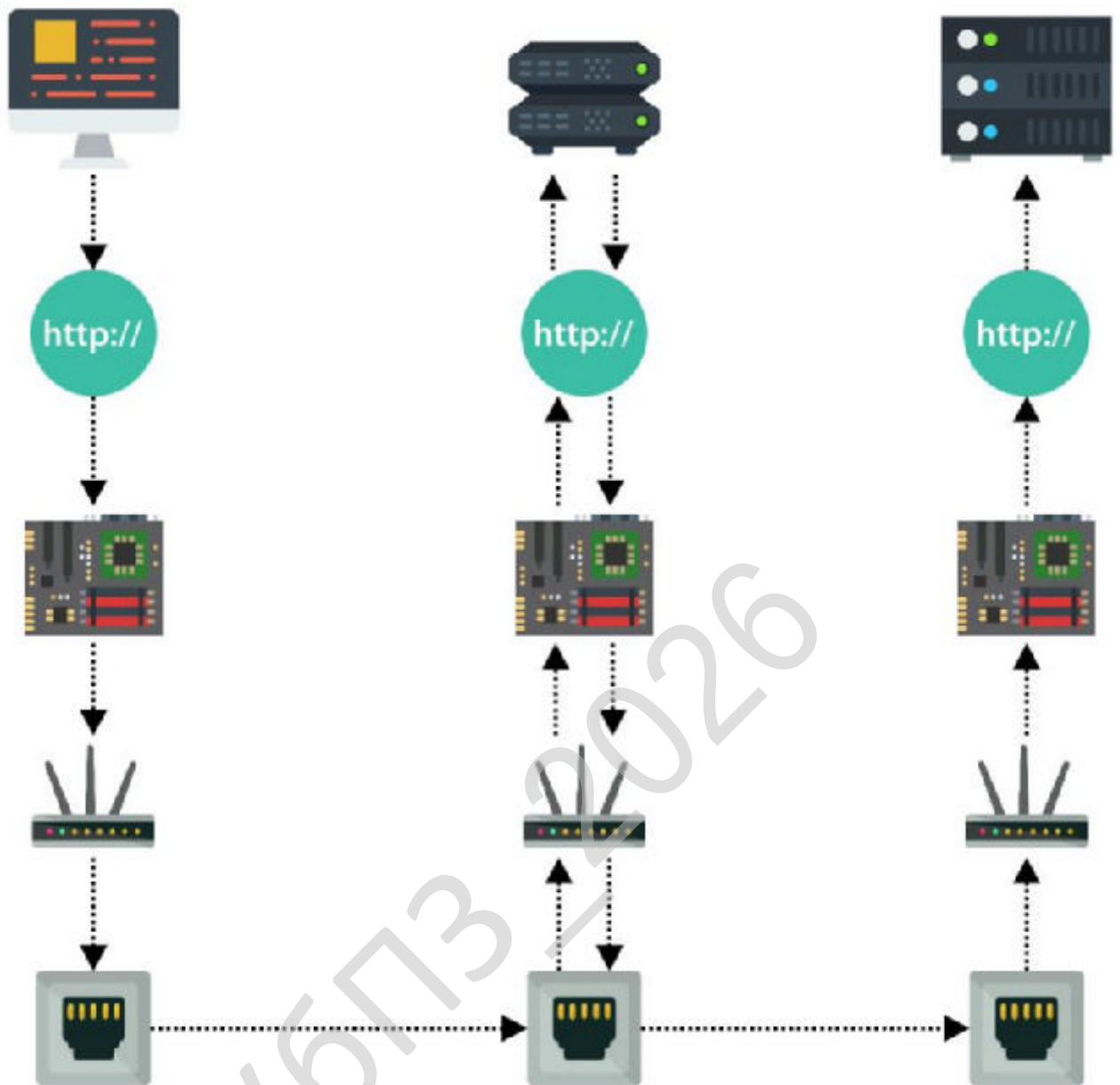


Рисунок 3.4 – Для того щоб балансувальник навантаження «рівня 7» (Application Load Balancer) зміг прочитати HTTP-запит і визначити, куди цей запит повинен бути спрямований, мережеве повідомлення повинне перед цим бути трансформоване аж до прикладного рівня мережевої моделі OSI

Аналогічним образом, балансувальник навантаження «рівня 4» (у цьому випадку мова йде про Network Load Balancer) зчитує заголовну інформацію, що втримується в TCP-пакеті, щоб визначити, куди направити це повідомлення, але інша інформація, що втримується в цих пакетах, йому для ухвалення рішення не

потрібна. Тому для Network Load Balancer потрібна трансформація мережевого повідомлення до транспортного рівня мережевої моделі OSI (детальніше дивитися на рисунку 3.5).

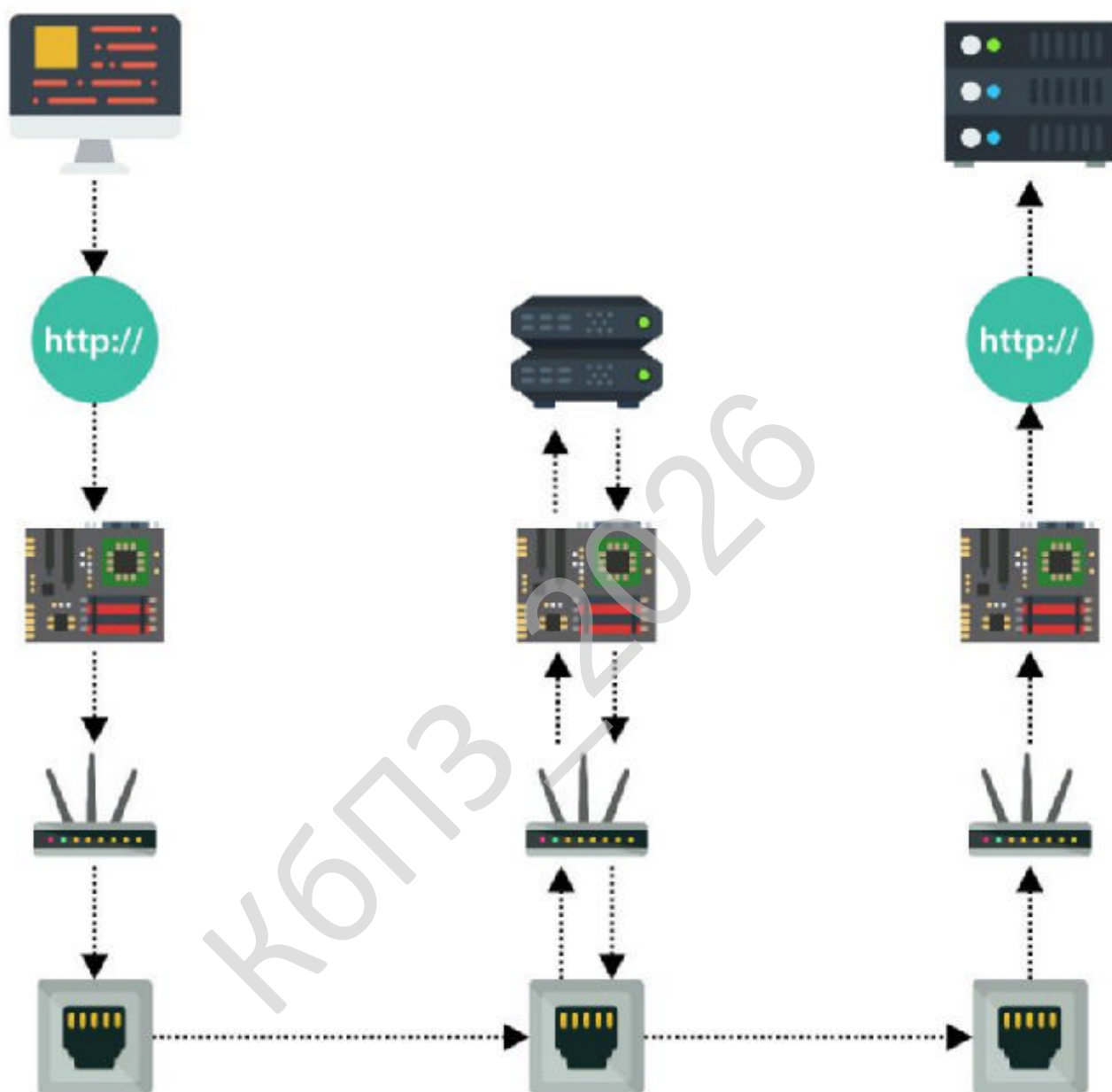


Рисунок 3.5 – Для ухвалення рішення балансувальнику навантаження «рівня 4» (Network Load Balancer) потрібна трансформація мережевого повідомлення до транспортного рівня мережевої моделі OSI

Основні особливості роботи Application Load Balancer і Network Load Balancer

Рівень балансувальника навантаження, який ви використовуєте для свого застосунку, безпосередньо впливає на те, що ви можете реалізувати за допомогою цього балансувальника навантаження, а також робить різні балансувальники навантаження краще підходящими під конкретні шаблони проектування застосунків.

Робота з SSL/TLS

Існує кілька підходів до реалізації шифрування SSL/TLS для мережевого трафіку між клієнтом і веб-сервером. Приміром, одним з розповсюджених застосувань для балансувальника навантаження «рівня 7» (у цьому випадку мова йде про Application Load Balancer) є зниження використання ресурсів веб-сервера (дивіться Рисунок 3.6). При такому підході «важка робота» по шифруванню й дешифруванню мережевого трафіку лягає на балансувальник навантаження. Таким чином, сервер[^]-серверові-веб-серверу, який розташований за Application Load Balancer, необхідно буде обробляти тільки незашифровані HTTP-запити, тому що ці запити вже були розшифровані балансувальником навантаження.

Проте, існують застосунки, для яких по регулятивних вимогах або міркуваннях безпеки необхідно в обов'язковому порядку здійснювати шифрування даних під час передачі повідомлень по мережі. Використання для таких застосунків балансувальника навантаження «рівня 7» тільки побільшає завантаження ваших ресурсів (дивіться Рисунок 3.7). Application Load Balancer працює на прикладному рівні, тому він змушений буде відразу розшифрувати HTTP-запит для перевірки його заголовків, а потім знову зашифрувати запит, щоб відправити його до місця призначення. Після цього вашому сервер[^]-серверові-веб-серверу прийде розшифрувати це мережеве повідомлення знову, щоб прочитати його. Це приведе не тільки до появи додаткової латентності за рахунок дублювання шифрування й дешифрування мережевого трафіку, але й створить додаткові погрози безпеки, тому що в цьому випадку ваш закритий ключ повинен буде зберігатися як на рівні балансувальника навантаження, так і на рівні веб-сервера.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		33

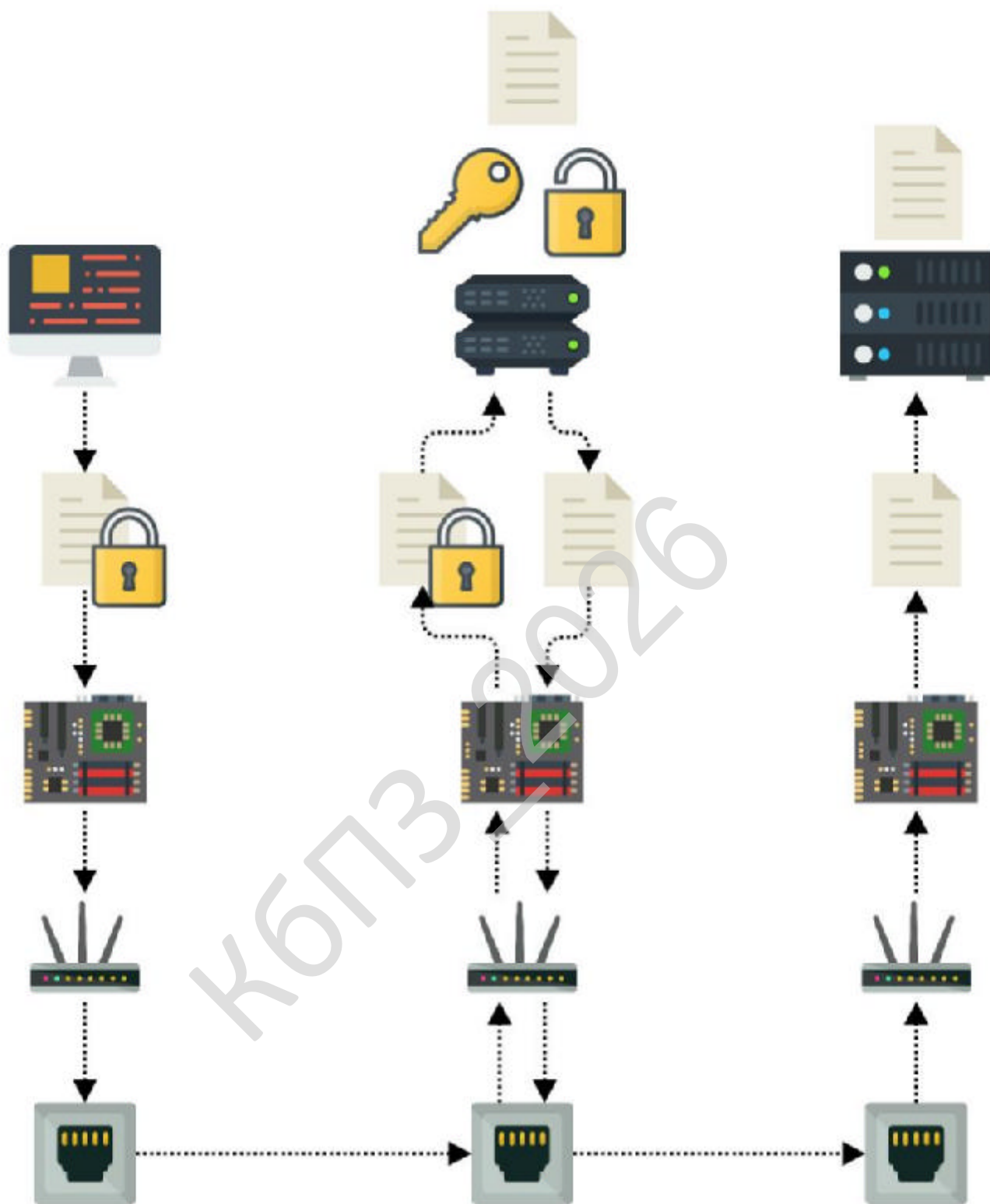


Рисунок 3.6 – Підхід до реалізації шифрування SSL/TLS для мережевого трафіку між клієнтом і веб-сервером, при якому «важка робота» по шифруванню й дешифруванню мережевого трафіку лягає на балансувальник навантаження Application Load Balancer

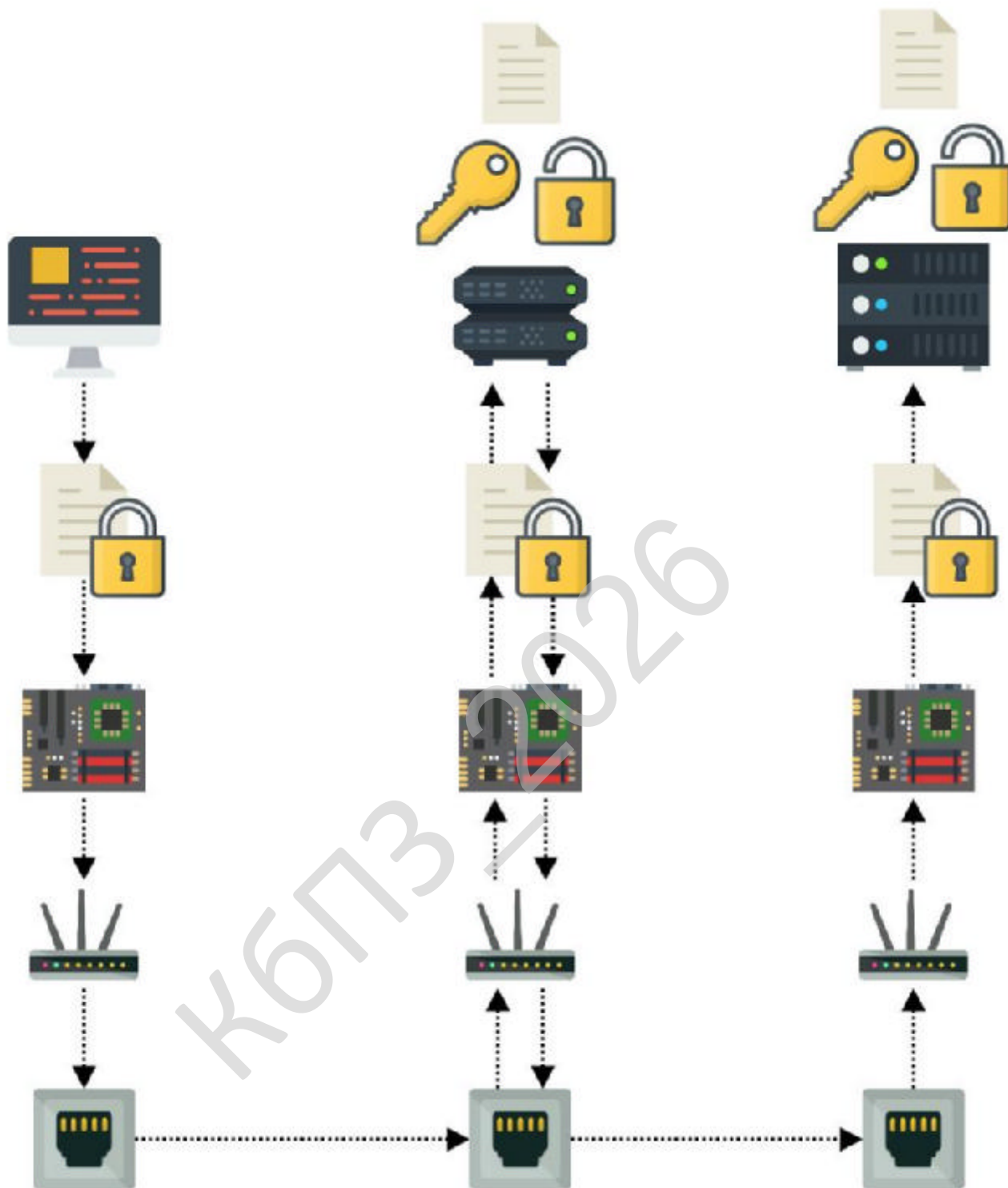


Рисунок 3.7 – Використання Application Load Balancer для застосунків, яким потрібна реалізація наскрізного шифрування, є недоцільним, тому що це приводить до появи додаткової латентності й створює нові погрози безпеки.

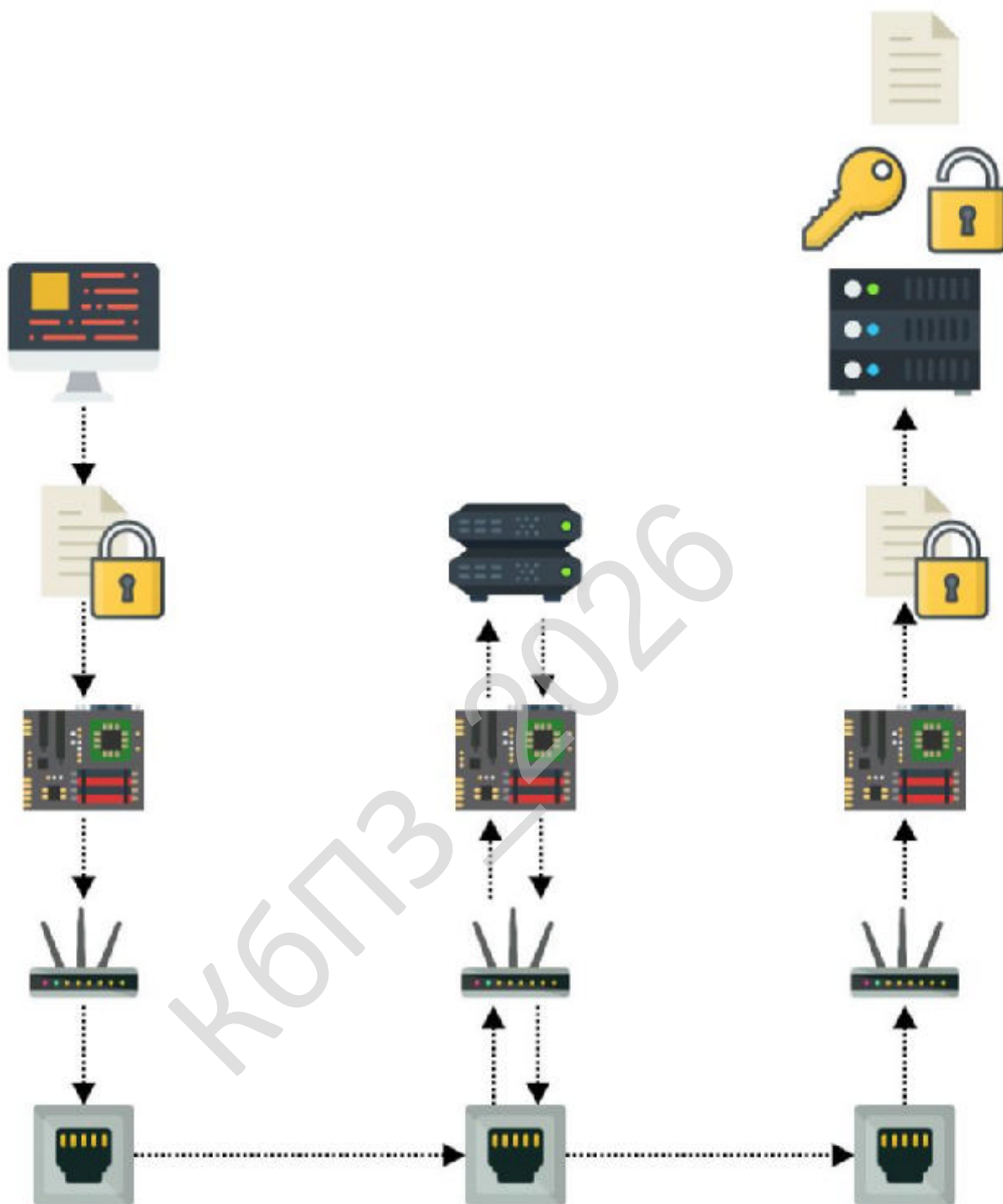


Рисунок 3.8 – Використання Network Load Balancer для застосунків, для яких потрібне здійснення шифрування даних під час мережевої передачі, зменшить латентність застосунку, а також забезпечить його дійсно наскрізним шифруванням між клієнтом і веб-сервером

Таким чином, якщо вашому застосунку потрібна реалізація наскрізного шифрування, те більш доцільно буде використовувати Network Load Balancer, який працює на транспортному рівні, а не балансувальник навантаження «рівня 7» (дивіться Рисунок 3.8). При такому дизайні застосунку корисне навантаження не буде розшифровуватися на рівні балансувальника навантаження. Замість цього Network Load Balancer просто буде направляє TCP-пакети на ваш веб-сервер. Це зменшить латентність застосунку, а також забезпечить його дійсно наскрізним шифруванням між клієнтом і веб-сервером.

Функціональність Host-Based Routing і Path-Based Routing для HTTP-запитів

Якщо наскрізне шифрування не потрібно, то ви можете використовувати одні із самих потужних функцій балансувальника навантаження Application Load Balancer: Host-Based Routing (Маршрутизація на основі хоста) і Path-Based Routing (Маршрутизація на основі шляху). При такій конфігурації балансувальник навантаження «рівня 7» використовує правила, які перевіряють HTTP-заголовки вхідного запиту. Якщо шлях запиту збігається з одним шаблоном, тоді Application Load Balancer направляє запит на один комп'ютер, але якщо він збігається з іншим шаблоном, то цей запит направляється на інший комп'ютер (дивіться Рисунок 3.9). Це може бути надзвичайно корисно для мікросервісних або інших розподілених архітектур, де у вас є один загальнодоступний API, але кілька компонентів включають різні секції API.

Варто окремо підкреслити, що функціональність «Маршрутизації на основі хоста» і «Маршрутизація на основі шляху» доступна тільки при використанні балансувальника навантаження «рівня 7», тому що для цієї можливості необхідна детальна інформація з HTTP-запиту. Network Load Balancer оперує тільки заголовками TCP-пакетів для своєї роботи, тому він не може читати заголовки HTTP-запитів, як це робить Application Load Balancer.

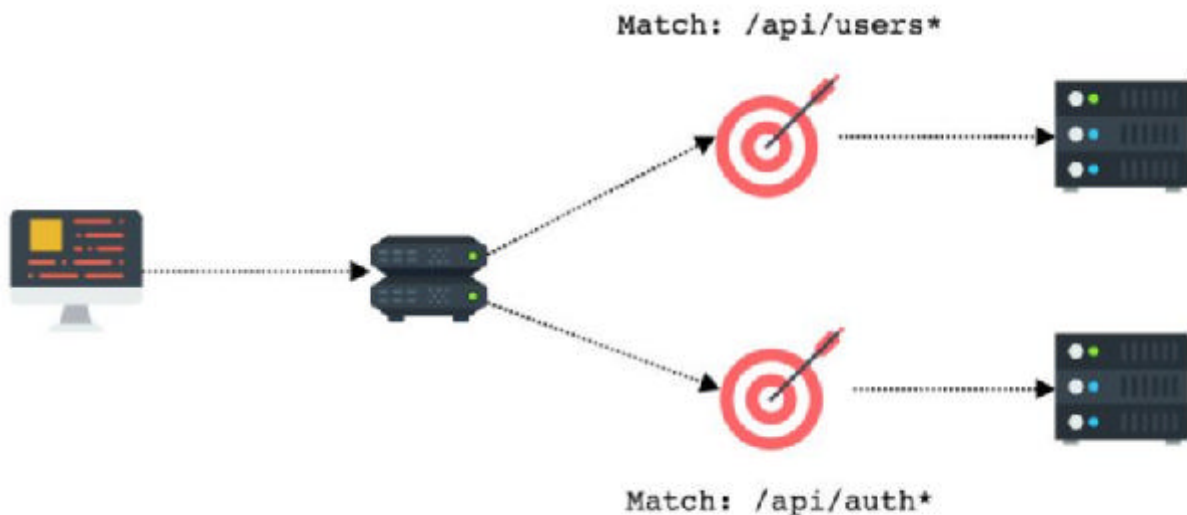


Рисунок 3.9 – Функції Host-Based Routing і Path-Based Routing – це один із самих потужних інструментаріїв в арсеналі розроблювача сучасних веб-застосунків

Протоколи прикладного рівня, відмінні від HTTP

Не кожний веб-застосунок побудований на основі протоколу HTTP. Наприклад, багатокористувацька гра в реальному часі може бути реалізована на основі власного спрощеного протоколу прикладного рівня поверх TCP (або навіть UDP) для зв'язку між ігровим клієнтом і ігровим сервером. Для такої взаємодії Application Load Balancer не буде працювати, оскільки цей балансувальник навантаження очікує, що на прикладному рівні весь мережевий трафік буде HTTP.

У цьому випадку варто використовувати Network Load Balancer, оскільки він може направляти TCP-пакети від ігрового клієнта до ігрового сервера. Крім того, для ігор у реальному часі не менш важливим буде й те, що балансувальник навантаження «рівня 4» також сприяє зменшенню латентності застосунку, тому що мережеві повідомлення не будуть проходити всю трансформацію від фізичного рівня до прикладного й назад, коли будуть оброблятися цим балансувальником навантаження.

3.2 Розробка структурної схеми

Application Load Balancer (ALB) і Network Load Balancer (NLB) – це балансувальники навантаження другого покоління від комерційної публічної хмарної платформи Amazon Web Services (AWS). У цьому розділі ми розглянемо, як повноцінно скористатися всіма перевагами їх нової функціональності, такий, приміром, як маршрутизація трафіку на ваші контейнерні застосунки Docker, що працюють у хмарі.

Ще однієї із самих цікавих і корисних нових функцій, доступних як для Application Load Balancer, так і Network Load Balancer, є можливість відправляти мережевий трафік на динамічні порти. Для балансувальника навантаження минулого покоління традиційна схема роботи з портами виглядала так: на кожному екземплярі використовуваної групи бекенд-серверів, розташованої за балансувальником навантаження, запускалася ідентична копія застосунку, що працює на добре всьому відомому статичному номері порту. При цьому завдання балансувальника навантаження минулого покоління обмежувалося тільки рівномірним розподілом будь-яких отриманих запитів по тому самому номеру порту між групою бекенд-серверів.

Традиційна схема роботи з портами для балансувальників навантаження минулого покоління обмежувалися тільки рівномірним розподілом будь-яких отриманих запитів по тому самому номеру порту між групою бекенд-серверів.

Однак, з настанням ери контейнерних застосунків Docker і початком активного використання стратегій упакування в контейнери (bin packing) для запуску декількох застосунків на одному інстансі Amazon EC2, змінився й підхід до організації багатьох архітектур програмного забезпечення. У цьому випадку один інстанс Amazon EC2 може працювати з декількома контейнерами, кожному з яких був випадковим образом призначений. Ці номери портів не є статичними. Контейнерам застосунку може бути призначений інший номер порту після зупинки їх роботи й перезапуску.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		39

При новому підході до організації архітектури застосунків один інстанс Amazon EC2 може працювати з декількома контейнерами, кожному з яких був випадковим образом призначений номер динамічного порту.

Як Application Load Balancer, так і Network Load Balancer підтримують цей новий шаблон динамічних портів, використовуючи функціональність «Цільова група» («Target Group»). Цей ресурс хмарної платформи AWS дозволяє відслідковувати список портів, які ухвалюють мережевий трафік у кожному інстансі, і, відповідно, надає балансувальнику навантаження можливість рівномірно розподіляти трафік між портами.

Таким чином, у вищенаведеному прикладі, Network Load Balancer 40 % мережевого трафіку відправить першому інстансу, у якому контейнери розміщені на двох портах, а інші 60 % мережевого трафіку будуть відправлені іншому інстансу, у якого є три відкриті порти.

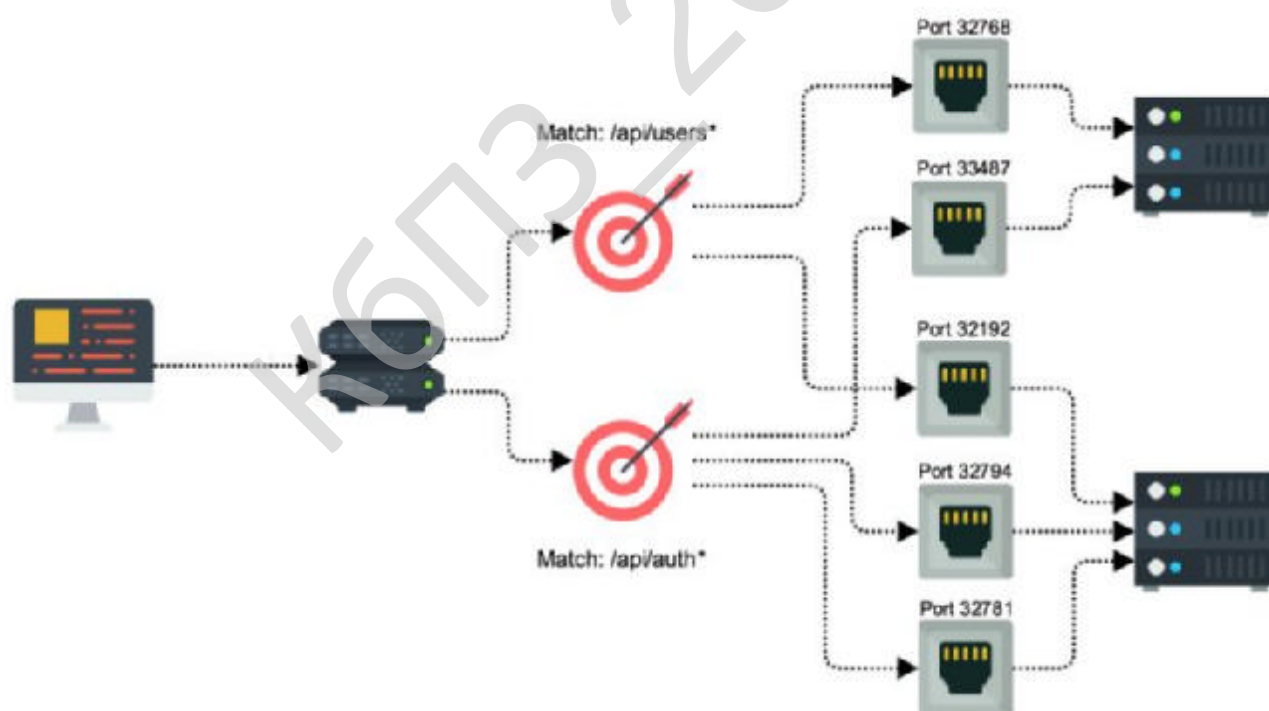


Рисунок 3.10 – Структурна схема системи

У свою чергу, Application Load Balancer, який здійснює маршрутизацію на основі HTTP на прикладному рівні мережевої моделі OSI, при подібній

конфігурації буде в одній цільовій групі по 50 % усього мережевого трафіку, який відповідає шаблону шляху «/api/users*», розподіляти між двома динамічними портами, а для іншої цільової групи із шаблоном шляху «/api/auth*» по третині мережевого трафіку буде розподілено між трьома іншими портами (дивіться рисунок 3.10).

Інтеграція Application Load Balancer і Network Load Balancer з Amazon EC2 Container Service

Завдяки підтримці динамічних портів ALB і NLB ідеально підходять для маршрутизації мережевого трафіку в сервісах керування контейнерами. Існує безшовна інтеграція між балансувальниками навантаження другого покоління й сервісом керування контейнерами Amazon EC2 Container Service (ECS).

ECS – це керована система оркестровки для платформи хмарних обчислень Amazon Web Services, використовувана для розгортання й експлуатації контейнерних застосунків Docker на безлічі інстансів. Вона призначена для забезпечення простого способу підключення широкої екосистеми сервісів AWS до контейнерів.

Розгортання контейнерного застосунку за Application Load Balancer або Network Load Balancer починається із самого застосунку. Візьмемо, приміром, простої веб-застосунок на Node.js, яке підключається до порту 8081 і очікує весь трафік, який зробить на цей порт (дивіться рисунок 3.11). Якби цей застосунок розгортався без контейнерів, воно було б запущене на декілька інстансах, і балансувальник навантаження направляв би весь трафік на статичний порт 8081 у кожному інстансе.

Але, оскільки ми впроваджуємо цей застосунок в Docker-контейнер, у нас з'являється можливість використовувати функціональність Docker для зіставлення портів усередині контейнера з різними портами на хосте, на якому запущений даний контейнер. Це досягається за допомогою простого налаштування, яке передається в Docker при запуску контейнера. В Amazon EC2 Container Service ми можемо використовувати ресурс, який називається

«Призначення завдання» («Task Definition»), за допомогою якого й призначається необхідна нам конфігурація. «Task Definition» – це полегшений варіант документа з метаданими, який повідомляє сервіс керування контейнерами ECS інформацію про те, з якими параметрами запускати ваш Docker-контейнер. Приклад «Task Definition», який ми можемо використовувати для запуску веб-застосунку на Node.js, описаного ледве вище, представлено на рисунку 3.12.

```
const app = require('koa')();
const router = require('koa-router')();

router.get('/', function *() {
  this.body = 'Ready to receive requests';
});

app.use(router.routes());
app.use(router.allowedMethods());

app.listen(8081);
```

Рисунок 3.11 – Приклад простого веб-застосунку на Node.js, яке підключається до порту 8081 і очікує весь трафік, який зробить на цей порт

Як ви можете бачити, порт контейнера зазначений як 8081. Це той же порт, до якого прив'язаний код застосунку. Однак порт хоста встановлений на нуль. Це говорить програмному забезпеченню Docker, що нам не потрібний певний статичний порт на хосте. Замість цього Docker може використовувати випадковий динамічний порт на хосте й перенаправляти трафік, який надходить на цей порт, на порт 8081 усередині контейнера.

При такій конфігурації кілька екземплярів контейнера можуть бути розгорнуті на одному комп'ютері.

Як ми можемо бачити, будь-який мережевий трафік, що надходить на порт 32768, направляється на порт 8081 в одному контейнері, а будь-який мережевий

трафік, що надходить на порт 33487, направляється на порт 8081 в іншому контейнері.

```
{
  "family": "api",
  "containerDefinitions": [
    {
      "name": "api",
      "image": "000000000000.dkr.ecr.us-east-2.amazonaws.com/api:v1",
      "cpu": 256,
      "memory": 256,
      "portMappings": [
        {
          "hostPort": 0,
          "containerPort": 8081,
          "protocol": "tcp"
        }
      ]
    }
  ]
}
```

Рисунок 3.12 – Приклад «Task Definition» для запуску веб-застосунку на Node.js, «упакованого» в Docker-контейнері

Таким чином, ресурс «Task Definition» з конфігурацією перенаправку динамічних портів може бути використаний для запуску сервісів в EC2 Container Service (дивитися рисунок 3.13).

Кожний сервіс ECS, підключений до балансувальнику навантаження, використовує функціональність «Цільова група», яка підтримує автоматичну синхронізацію при запуску нового завдання або її зміні. Цільова група надає список кожного інстанса й усіх портів, які ухвалюють мережевий трафік на цьому інстансе (дивитися рисунок 3.14).

Service : api

Cluster **production** Status **ACTIVE**

Task Definition **api6**

Desired count 2

Pending count 0

Running count 2

Service Role **api**

Details Tasks Events Auto Scaling Deployments Metrics

Load Balancing

Target Group Name	Container Name	Container Port
api	api	8081

Рисунок 3.13 – Використання ресурсу «Task Definition» з конфігурацією перенаправлення динамічних портів для запуску сервісів в EC2 Container Service

Target group: **api**



Description Targets Health checks Monitoring Tags

The load balancer starts routing requests to a newly registered target as soon as the registration process completes and the target passes the initial health checks. If demand on your targets increases, you can register additional targets. If demand on your targets decreases, you can deregister targets.

Edit

Registered targets

Instance ID	Name	Port	Availability Zone	Status
i-04d959e3ae7ab14ec		32768	us-east-2b	healthy ⓘ
i-04d959e3ae7ab14ec		33487	us-east-2b	healthy ⓘ

Рисунок 3.14 – Ресурс «Цільова група», який використовує кожний сервіс ECS, підключений до балансувальнику навантаження, надає список кожного інстанса й усіх портів, які ухвалюють мережевий трафік на цьому інстансе

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		44

Application Load Balancer або Network Load Balancer, підключений до «Цільової групи», буде використовувати цей список, щоб зробити вибір інстанса й порту, на який буде відправлений мережевий трафік.

ALB або NLB використовують «Цільову групу», щоб вибрати інстанс і порт, на який буде відправлений мережевий трафік.

Таким чином, увесь процес маршрутизації мережевого трафіку в сервісах керування контейнерами виглядає в такий спосіб:

1. Клієнтський застосунок ініціює нове з'єднання з балансувальником навантаження.

2. Балансувальник навантаження одержує мережевий трафік і вибирає «мета» з «Цільової групи», прикріпленої до балансувальнику навантаження. Ця «Цільова група» була зконфігурована сервісом керування контейнерами Amazon EC2 Container Service і містить у собі список доступних інстансів і портів.

3. Балансувальник навантаження відправляє мережевий трафік на обраний інстанс і порт.

4. Docker ухвалює мережевий трафік і перенаправляє його на зконфігурований порт усередині потрібного контейнера.

5. Ваш застосунок, що працює усередині контейнера, одержує мережевий трафік через порт, до якого воно прив'язане.

Як Application Load Balancer, так і Network Load Balancer спроектовані з нуля для сучасної парадигми конфігурації динамічних портів, які звичайно використовуються при розгортанні сучасних контейнеризованих застосунків. Вибір правильного балансувальника навантаження буде залежати від конкретних потреб вашого застосунку, таких як: чи є мережевий трафік HTTP чи ні, чи потрібне вам наскрізне шифрування SSL/TLS, а також чи прагнете ви маршрутизувати мережевий трафік на основі хоста й шляхи чи ні.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.15. З рисунку видно, що розроблена система складається з наступних функціональних частин:

- Блок моделювання завантаженості системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.
- Блок моніторингу системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.
- Блок інтерфейсу користувача.
- Блок призначення пріоритетів.
- Блок організації та обслуговування черг.
- Блок управління навантаженням.
- Блок формування трафіка.
- Блок визначення параметрів QoS.
- Блок примусового завдання параметрів QoS.
- Блок дослідження можливостей механізмів WRED.
- Блок дослідження можливостей механізмів WFQ.

Розглянемо ці блоки більш детально.

Блок моделювання завантаженості системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer

Призначений для моделювання завантаженості системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer з визначеним трафіком та заданими параметрами якості обслуговування (QoS).

Блок моніторингу системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer

Призначений для аналізу поточного стану системи балансування

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		46

навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

Блок інтерфейсу користувача

Призначений для реалізації взаємодії користувача, або дослідника з системою.

Блок призначення пріоритетів

Нарівні з нарощуванням апаратного забезпечення мережі для реалізації QoS застосовуються й засоби типу завдання пріоритетів даних і організації черг. Маршрутизатори підтримують ці механізми протягом багатьох років, як і деякі з нових комутаторів для каналів Gigabit Ethernet. Однак ПЗ для керування мережею за заданими правилами, яких необхідно для практичного втілення цієї технології, поки не розроблено.

Способи пріоритезації даних можна умовно підрозділити на явні й неявні.

При неявному призначенні пріоритетів маршрутизатор або комутатор автоматично привласнює послугам відповідні рівні, виходячи із заданих адміністратором мережі критеріїв (наприклад, типу додатка для застосовуваного протоколу передачі або адреси джерела). Кожний вхідний пакет аналізується (фільтрується) на відповідність цим критеріям. Механізм неявної пріоритезації підтримують практично всі маршрутизатори.

Деякі комутатори теж здатні задавати пріоритети, але мають обмежений набір функцій. Так, комутатори можуть забезпечувати пріоритезацію даних по типу віртуальної локальної мережі, адресі джерела або адресата, але не використовують інформацію більш високого рівня (протокол передачі або тип додатка). Розроблювальні в цей час системи керування мережею за заданими правилами дозволять реалізувати більше зроблені схеми пріоритезації даних при роботі з такими комутаторами. При явній пріоритезації даних користувач або додаток запитує певний рівень служби, а комутатор або маршрутизатор намагається задовольнити запит. Імовірно, самим популярним механізмом явної пріоритезації стане протокол IP Precedence (протокол старшинства), що одержав

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		47

другу назву IP TOS (IP Type Of Service), – один з розділів четвертої версії протоколу IP.

IP TOS резервує спеціальне поле в заголовку пакета, де можуть бути зазначені ознаки QoS, що визначають час затримки, швидкість пропущення й рівень надійності передачі пакета. Однак знайдеться небагато популярних додатків – за винятком мультимедійного ПЗ, – у які реалізована підтримка протоколу IP TOS. Зараз розробляється протокол резервування ресурсів RSVP, що передбачає більше складний, чим в IP TOS, механізм передачі від додатка до маршрутизатору запиту на гарантовану якість послуг. Як і IP TOS, протокол RSVP поки не одержав широкої підтримки розроблювачів – він реалізований лише в окремих типах маршрутизаторів. Поширення RSVP стримується через те, що не вирішені деякі питання, пов'язані із сумісністю різних мереж. До того ж застосування RSVP значно збільшує навантаження на маршрутизатори й може привести до зниження швидкодії цих пристроїв. Видимо, у доступному для огляду майбутньому неявна пріоритезація, не потребуюча серйозних обчислювальних потужностей маршрутизатора, залишиться більше популярною, чим явна. Крім того, при явному завданні пріоритетів значно ускладнюється керування мережею. Кінцеві користувачі, швидше за все, будуть набудовувати своє програмне забезпечення на запит найвищого з можливих рівнів послуг. Відповідно, адміністраторові мережі прийдеться розробляти правила керування користувачами й, можливо, навіть побудовувати служби з гарантованою якістю для кожного користувача окремо.

Блок організації та обслуговування черг

Після того як переданим по мережі даним призначені відповідні пріоритети (за допомогою явних або неявних методів), потрібно визначити порядок передачі цих даних, задавши алгоритм обслуговування черг із необхідною якістю (рівнем QoS). По суті, черги являють собою області пам'яті комутатора або маршрутизатора, у яких групуються пакети з однаковими пріоритетами передачі. Алгоритм обслуговування черги визначає порядок, у

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		48

якому відбувається передача пакетів, що зберігаються в ній. Зміст застосування всіх алгоритмів зводиться до того, щоб забезпечити найкраще обслуговування трафіку з більш високим пріоритетом за умови, що й пакету з низьким пріоритетом гарантується відповідна увага.

При використанні способів завдання явних і неявних пріоритетів алгоритм обробки черг визначає порядок їхнього обслуговування. Відповідно до цього алгоритму на кожні два пакети, переданих у мережу із черги 1 (з високим пріоритетом) доводиться по одному пакету із черг 2 і 3. Пакети з однаковими пріоритетами передаються за принципом FIFO (“першим прийшов – першим вийшов”).

Якщо в мережі виникає перевантаження, служба черг не гарантує своєчасного досягнення пункту призначення найбільш важливими даними. Гарантується лише те, що ці пакети будуть передані раніше, ніж ті, що мають більш низький пріоритет.

Сучасні служби QoS вирішують таке завдання за рахунок резервування смуги пропускання. Кожній із черг (або їхніх груп) виділяється заздалегідь задана величина смуги пропускання, що гарантує певну смугу пропускання для черги з більш високим пріоритетом. Для критичних ситуацій, коли обсяг даних у черзі перевищує розміри смуги пропускання, в алгоритмах обслуговування звичайно передбачається передача трафіку з високим пріоритетом на смугу пропускання, “приналежну” чергам з низьким пріоритетом, і навпаки. Найпростіші алгоритми обслуговують кожну чергу за принципом FIFO. При цьому передача кадрів великого розміру, що мають високий пріоритет, може приводити до затримок трафіку іншого додатка з настільки ж високим пріоритетом, але меншим обсягом.

У більш складних алгоритмах уживає спроба “справедливої” обробки черг. Наприклад, алгоритм рівномірного пропорційного (або зваженого) обслуговування (WFQ – Weighted Fair Queuing), розроблений компанією Cisco, підрозділяє додатки на потребуючі великої й малої ширини смуги пропускання, а сама смуга пропускання розподіляється між всіма додатками нарівно. Слід

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		49

зазначити, що основні виробники маршрутизаторів самі розробляють алгоритми обслуговування черг і використовують для їхнього опису власну термінологію.

Істотним недоліком сучасних маршрутизаторів і комутаторів є те, що вони підтримують мале число черг. Найчастіше виробники організують служби QoS, що використовують чотири черги, хоча чим більше черг, тим більше різних пріоритетів можна привласнити переданим пакетам і тим “справедливіше” розподілити смугу пропускання між додатками. Наприклад, адміністратор у стані задати пріоритети таким чином, щоб перевага при передачі віддавалося пакетам, адресованим на більше віддалені вузли.

Блок управління навантаженням

Служба QoS дає можливість використовувати для керування мережею системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer два важливих механізми – керування в умовах перевантаження й запобігання перевантажень. Перший з них дозволяє кінцевій станції відразу знижувати швидкість передачі даних, коли в мережі починається втрата пакетів. У протоколах TCP/IP і SNA цей механізм підтримується вже протягом декількох років. І хоча сам по собі він не гарантує якості передачі, при його використанні разом з механізмом запобігання перевантажень результати виявляються набагато кращими. У мережах TCP/IP механізм запобігання перевантажень застосовується досить давно, але лише в останні роки він стає стандартом “де-факто” для маршрутизаторів телекомунікаційних мереж і Internet.

Стандартним способом запобігання перевантажень у мережі стало застосування механізму випадкового виділення пакетів (Random Early Detection, RED). При заповненні черг вище певної критичної оцінки цей механізм змушує маршрутизатор вибирати із черги за випадковим законом деякі пакети й “втрачати” їх. Швидкість передачі даних станціями-відправниками знижується, що й дозволяє уникнути переповнення черги.

Механізм пропорційного випадкового виділення пакетів – WRED (Weighted RED) – можна вважати наступною, більше зробленою “версією” RED.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		50

Він передбачає, що вибір пакетів, які повинні “втратитися”, буде відбуватися з обліком їх пріоритезації згідно IP TOS.

Блок формування трафіка

Формування трафіку – це загальний термін, яким прийнято позначати різні способи маніпулювання даними для підвищення якості їхньої передачі. Один із таких способів – сегментація пакетів. У мережах системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer гарантовано високий рівень QoS досягається в тому числі й за рахунок малого розміру переданих пакетів (осередків – у термінології системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer). Максимальний час затримки при передачі будь-якого пакета системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer – це час передачі одного осередку.

Запозичаючи корисні механізми технології системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer, виробники маршрутизаторів і комутаторів починають забезпечувати у своїх продуктах можливість сегментації пакетів. Деякі пристрої, призначені для мереж frame relay, сегментують пакети, передані по каналах глобальних мереж, щоб гарантувати конкретний час передачі й мінімізувати затримки.

Ще один спосіб формування трафіку – його “вирівнювання”. Для таких протоколів, як наприклад, AppleTalk, характерна нерівномірна передача пакетів, що часом приводить до появи в мережі послідовностей або ланцюжків пакетів, а отже – до її перевантаження. Процедура вирівнювання трафіку дозволяє розчленувати ланцюжки шляхом розміщення пакетів у буфері перед їхньою передачею в мережу. Для забезпечення більше рівномірної передачі даних можна також вирівнювати трафік кінцевих вузлів мережі.

Блок визначення параметрів QoS

Призначений для визначення існуючих параметрів якості обслуговування (QoS). До них відносяться:

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		51

– Bandwidth (BW) – смуга пропускання, описує номінальну пропускну здатність середовища передачі інформації, визначає ширину каналу. Вимірюється в bit/s (bps), kbit/s (kbps), mbit/s (mbps).

– Delay – затримка при передачі пакета.

– Jitter – коливання (варіація) затримки при передачі пакетів.

– Packet Loss – втрати пакетів. Визначає кількість пакетів, що відкидаються мережею під час передачі.

Блок примусового завдання параметрів QoS

Призначений для примусового завдання одного, або декількох параметрів якості обслуговування (QoS). До них відносяться:

– Bandwidth (BW) – смуга пропускання, описує номінальну пропускну здатність середовища передачі інформації, визначає ширину каналу. Вимірюється в bit/s (bps), kbit/s (kbps), mbit/s (mbps).

– Delay – затримка при передачі пакета.

– Jitter – коливання (варіація) затримки при передачі пакетів.

– Packet Loss – втрати пакетів. Визначає кількість пакетів, що відкидаються мережею під час передачі.

Блок дослідження можливостей механізмів WRED

Одним з методів QoS, призначених для забезпечення необхідних вимог до різних потоків даних – запобігання перевантажень (congestion avoidance). Він заснований на обмеженні розмір черги, сигналізуючи джерелам даних про необхідність зменшити швидкість передачі інформації (WRED – Weighted random early detection).

Блок дослідження можливостей механізмів WFQ

Другим з методів QoS, призначених для забезпечення необхідних вимог до різних потоків даних – керування перевантаженням (congestion management). Він заснований на присвоєнні квот і пріоритетів потокам, і у випадку перевантаження, потоки одержують якість, обмежену їхньою квотою й пріоритетом (WFQ – Weighted Fair Queuing).

Існує не занадто багато способів розрахунків показників QoS у системи

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		52

балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer. Найпростіший з них – збільшення смуги пропускання системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer за рахунок нарощування апаратних можливостей устаткування системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer. Можна використовувати й такі прийоми, як завдання пріоритетів даних, організація черг, запобігання перевантажень і формування трафіку. Керування мережею за заданими правилами в перспективі повинне об'єднати всі ці способи в єдину автоматизовану систему, що буде гарантувати якість послуг абсолютно на всіх ділянках мережі.

Збільшення апаратної потужності, безсумнівно, є найбільш ефективним засобом реалізації QoS у системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer. Тиск із боку конкурентів, необхідність підвищення ефективності виробництва, поява нових технологій, що дозволяють оснащувати спеціалізовані мікросхеми (ASIC) найрізноманітнішими функціями, – все це змушує постачальників комутаційного встаткування для локальних мереж викидати на ринок усе більше швидкодіючі пристрої за цінами, порівнянним з вартістю моделей колишнього покоління.

Малоймовірно, що в доступному для огляду майбутньому даний підхід до підтримки QoS у мережах системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer перестане бути пріоритетним. Оскільки в мережах системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer вдається забезпечити гарантовану якість послуг, не прибігаючи до дорогої модернізації усього встаткування й серйозних змін у системі керування мережею, мережні адміністратори будуть звертати увагу й на програмні засоби, що дозволяють реалізувати QoS.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		53

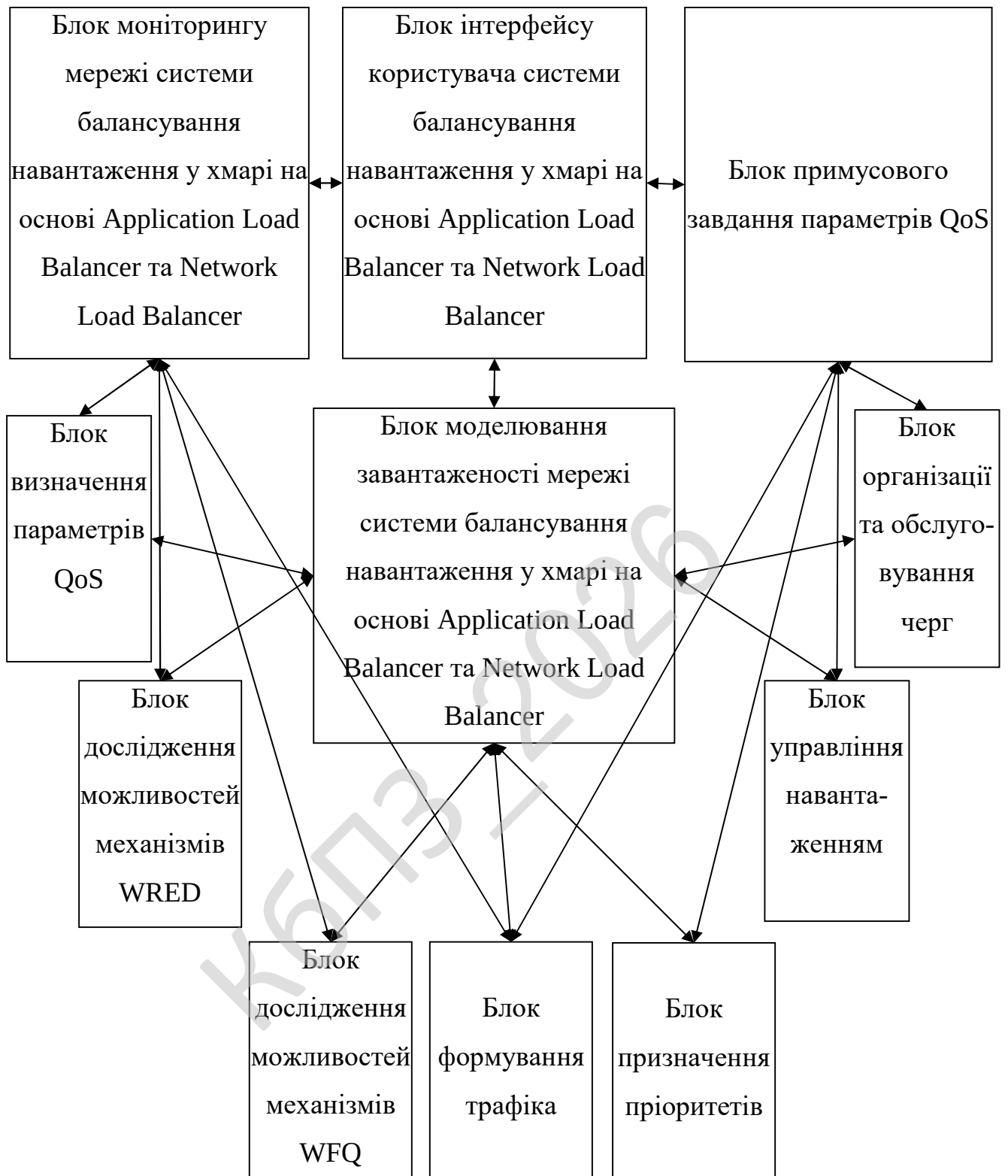


Рисунок 3.15 – Функціональна схема системи

Отже, найбільше поширення, швидше за все, одержить комбінований підхід. Деякі виробники висловлюються на його користь, затверджуючи, що

найкраще збільшувати пропускну здатність мережі не прямо, а за рахунок інтелектуальних можливостей устаткування, що має засоби розрахунків показників QoS. Правда, виробники мережних пристроїв навряд чи можуть бути об'єктивними в цьому питанні, тому що вони зацікавлені в збуті тих самих продуктів, які підтримують гарантовану якість послуг.

У глобальних мережах нарощування апаратних потужностей використовується рідше. Звичайно, зниження вартості смуги пропускання зробило б передачу даних по глобальних мережах доступною для більше широкого кола користувачів (і навіть трохи знизило б актуальність впровадження гарантованої якості послуг). Але в найближчому майбутньому вартість смуги пропускання в глобальних мережах буде залишатися досить високою, тому й нарощування апаратної потужності не стане настільки популярним, як у мережах системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

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

3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.16.

Розроблена діаграма взаємодії процесів використовується для представлення та візуалізації процесів обробки даних тобто структурного проектування бакалаврської роботи.

Основні складові елементи діаграми взаємодії процесів це потоки даних:

- Репозиторії, потік сховища даних.
- Потоки зовнішні по відношенню до системи сутності.
- Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Потоки даних гібридні між елементами трьох попередніх типів.

Відповідно до документації основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.



Рисунок 3.16 – Діаграма взаємодії процесів

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Таким чином у результаті після розгляду, вищеописаної системи, схеми структурної, функціональної, діаграми взаємодії процесів перейдемо до опису та розгляду блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

К6ПЗ-2026

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		57

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1.

З рисунку видно, що після запуску програми спочатку відбувається вивід основного вікна програми. Потім здійснюється наступна послідовність дій.

- Виведення вікна системи балансування навантаження у хмарі.
- Сканування та виведення наявного обладнання.
- Підпрограма оптимізації Application Load Balancer.
- Виведення результатів роботи.
- Моніторинг стану мережі системи.
- Формування пакету поточних значень.
- Виведення пакету результатів.
- Аналіз статистики навантаження мережі (Network Load Balancer).
- Формування поточного звіту.
- Виведення результатів аналізу.
- Розподіл потоку даних системи.

Робота підпрограми (рисунок 4.2):

- Ініціалізація динамічних структур та підключення файлі звітності.
- Редагування таблиці відстаней.
- Пошук найкоротшого шляху з використанням Application Load Balancer.
- Побудова таблиці відстаней.
- Редагування таблиці топології.
- Перевірка кількості рядків в таблиці топології.

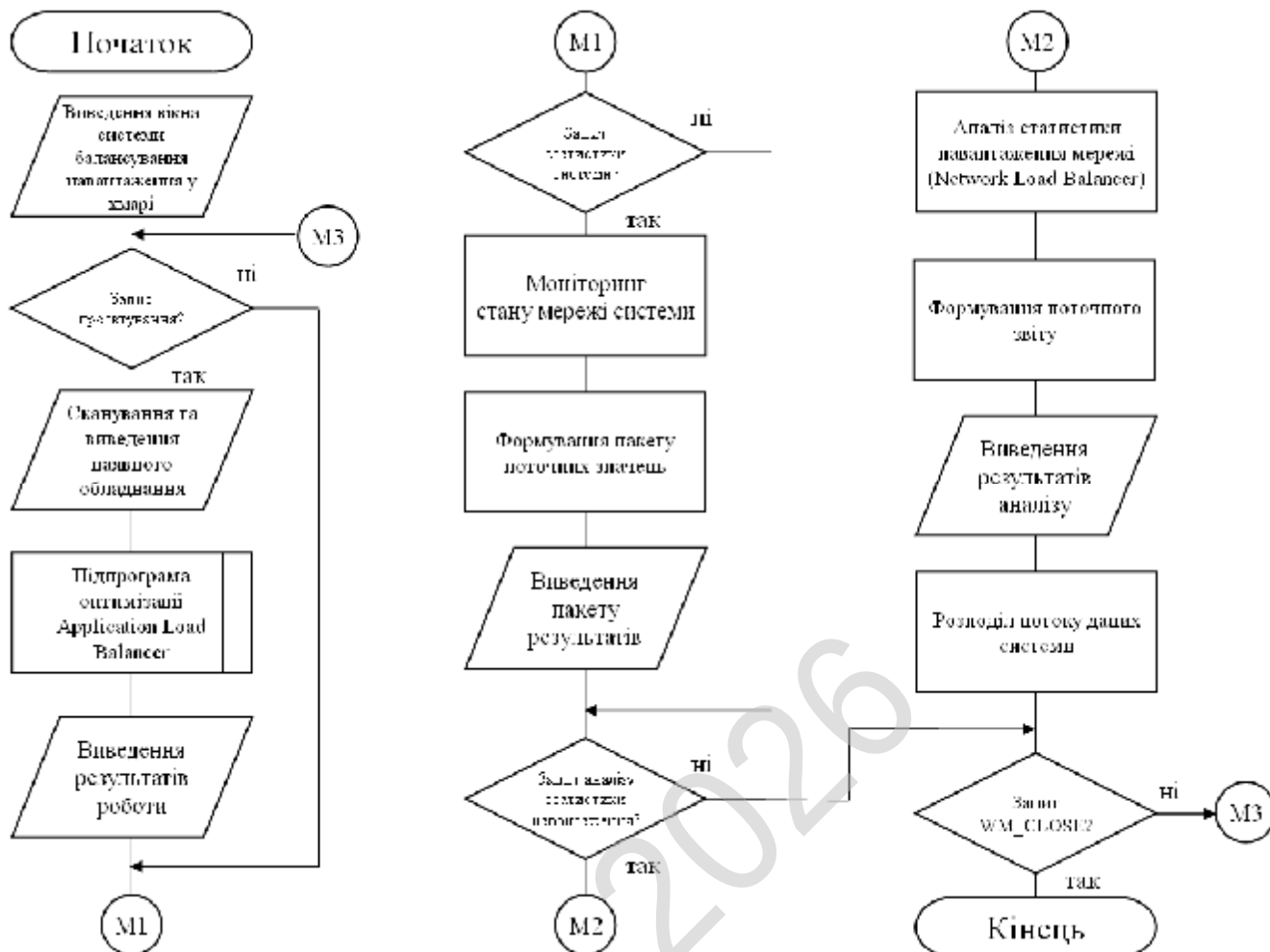


Рисунок 4.1 – Блок-схема основної програми

- Формування множини вузлів, найближчих до маршрутизатора.
- Редагування таблиці топологій.
- Оновлення таблиці відстаней.
- Формування базової множини шляхів передачі даних.

Розглянемо визначення API що активно використовувалось під час роботи. Це прикладний програмний інтерфейс (Application Programming Interface, API) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

Спрощено – це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного

забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

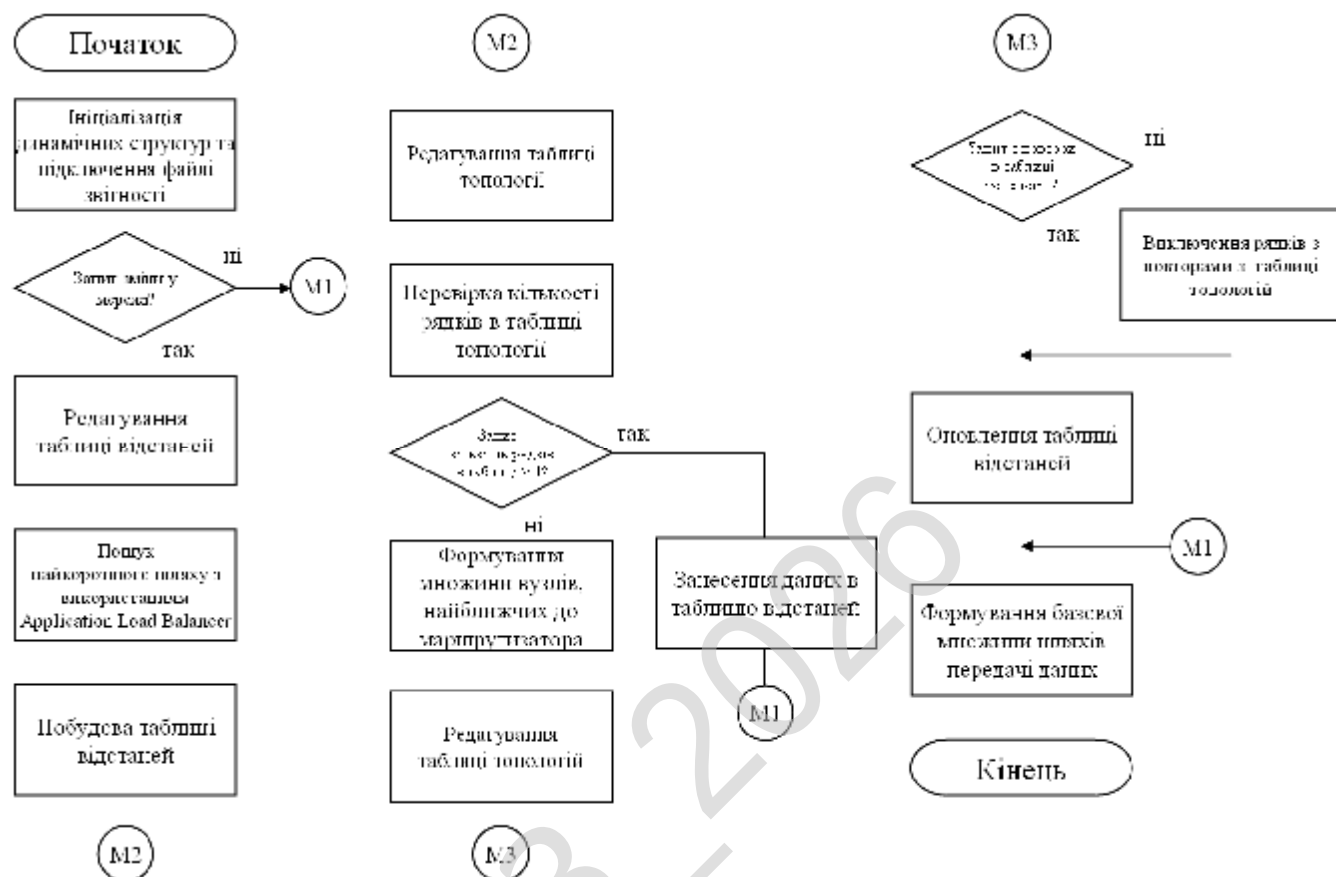


Рисунок 4.2 – Блок-схема роботи підпрограми

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій, наприклад для малювання вікна чи іконок на екрані. API є абстрактним поняттям – програмне забезпечення, що пропонує деякий API, часто називають реалізацією (implementation) даного API. У багатьох випадках API є частиною набору розробки програмного забезпечення, водночас, набір розробки може включати як API, так і інші інструменти/апаратне забезпечення, отже ці два терміни не є взаємозамінювані.

Високорівневі API часто програють у гнучкості. Виконання деяких функцій нижчого рівня стає набагато складнішим, або навіть неможливим.

В об'єктно-орієнтованих мовах, прикладний програмний інтерфейс зазвичай включає в себе опис набору визначень класу, з набором форм поведінки, пов'язаних з цими класами. Це абстрактне поняття пов'язане з реальними функціями, які надані або надаватимуться, класами, які реалізуються в методах класу.

Прикладний програмний інтерфейс в даному випадку можна розглядати як сукупність всіх методів, які публічно доступні в класах (зазвичай званий інтерфейс класу). Це означає, що прикладний програмний інтерфейс вказує методи, за допомогою яких взаємодіє з об'єктами, отриманими з визначень класів і обробляє їх.

У більш загальному плані можна визначити Прикладний Програмний Інтерфейс як сукупність усіх видів об'єктів, які можна вивести з визначення класу, і пов'язаних з ними можливих варіантів поведінки.

Наприклад: клас, що представляє Stack, може просто виставити публічно два методи Push() (для додавання нового елемента в стек) і Pop() (для вилучення останнього пункту, ідеально розташований на вершині стека).

У цьому випадку Прикладний Програмний Інтерфейс може бути інтерпретованим як два методи pop() і push(), або, більш широко, використовується варіант, коли можна використовувати елемент типу Stack, який реалізує поведінку стека, надаючи йому можливість для додавання / видалення елементів з вершини. Друга інтерпретація видається більш доречною в дусі об'єктно-орієнтованого підходу.

Якість документації, пов'язаної з Прикладним Програмним Інтерфейсом, є часто ключовим фактором, що визначає його успішність з точки зору простоти використання.

ППІ, як правило, пов'язаний із бібліотеками програмного забезпечення: ППІ описує і вказує очікувану поведінку в той час, як бібліотека є фактичною реалізацією даного набору правил. Один ППІ може мати декілька реалізацій (або жодної, будучи абстрактним) у вигляді різних бібліотек, які мають такий же

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		61

інтерфейс. Прикладний програмний інтерфейс також може бути пов'язаним з платформами програмування: платформа може бути заснована на кількох бібліотеках реалізує декілька інтерфейсів ППІ, але на відміну від звичайного використання ППІ, доступ до поведінки вбудований в платформу опосередкований шляхом розширення його змісту новими класами і вставлений в саму платформу. Крім того, загальний потік управління програми може бути під контролем абонента.

Прикладний програмний інтерфейс може бути також реалізацією протоколу.

Коли ППІ реалізує протокол, він може бути заснованим на проксі-методах віддалених викликів, що засновані на протоколі зв'язку. Роль ППІ може заключатися саме в тому, щоб приховати деталі транспортного протоколу. Наприклад: RMI є ППІ, який реалізує протокол або JRMP IIOP як RMI-IIOP.

Протоколи, як правило, розподіляються між різними технологіями і зазвичай дозволяють різним технологіям обмінюватися інформацією, діючи як абстракція між двома світами. ППІ, як правило, є специфічним для конкретної технології: звідси, інтерфейси даної мови не можуть бути використані на інших мовах, якщо виклики функції не будуть перетворені з конкретної адаптації бібліотеки.

Деякі мови, серед яких такі, що працюють на віртуальних машинах (наприклад: мови, сумісні з NET CLI середовища CLR і JVM сумісних мов у віртуальній машині Java) можуть ділитися програмними інтерфейсами.

У цьому випадку віртуальна машина дозволяє мові взаємодії завдяки спільному знаменнику віртуальної машини, що абстрагується від конкретної мови, використовувати проміжний байт-код і його мову.

При використанні прикладного програмного інтерфейсу в контексті веб-розробки, як правило, ППІ визначається набором повідомлень запиту HTTP, також визначається структура повідомлень-відповідей, зазвичай у розширенні мови розмітки XML або в форматі об'єктного запису JavaScript (JSON). У той час

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		62

як прикладний програмний інтерфейс у Web історично був практично синонімом для веб-служби, останнім часом тенденція змінилась (так званий Web 2.0) на відхід від Simple Object Access Protocol (SOAP) на основі веб-сервісів і сервіс-орієнтованої архітектури (SOA) на більш прямі передачі репрезентативного стану (REST) стилів веб-ресурсів та ресурсів-орієнтованої архітектури (ROA).

Частина цієї тенденції пов'язана з рухом Семантичного веб-ресурсу до Опису Платформ (RDF), Концепції розвитку веб-технологій інженерних онтологій. Прикладні програмні інтерфейси у Web, що дозволяють комбінувати декількома прикладними програмними інтерфейсами в нові додатки називають гібридними.

Розглянемо використану архітектуру клієнт-сервер. Це 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. 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

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 браузером.

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 інсталиувати спеціальні програми (хоча інколи така необхідність все-таки виникає).

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		65

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).

Опис системи

Програмне забезпечення виконує керування балансуванням навантаження у хмарній інфраструктурі. Система розподіляє HTTP та HTTPS запити через Application Load Balancer, а TCP та UDP трафік через Network Load Balancer. Програма оцінює стан серверів, кількість активних з'єднань, затримку, помилки та доступність цільових груп.

Архітектура системи.

Архітектура складається з модуля приймання запитів, модуля Application Load Balancer, модуля Network Load Balancer, модуля перевірки стану серверів, модуля моніторингу та модуля формування звіту. Усі модулі працюють в одній логічній системі та передають дані між собою через об'єкти Python.

Основні модулі

1. Модуль цільових серверів зберігає параметри серверів і стан їх доступності.
2. Модуль Application Load Balancer розподіляє прикладні запити за шляхом сервісу.
3. Модуль Network Load Balancer розподіляє мережевий трафік за портом і протоколом.
4. Модуль перевірки стану визначає здорові та проблемні сервери.
5. Модуль моніторингу збирає статистику запитів, помилок і затримки з модулем звітності формує короткий підсумок роботи системи.

Розглянемо основні функції у вихідному коді ПЗ

Система приймає запити користувачів, визначає тип балансувальника, обирає доступний сервер, контролює стан цільових груп, фіксує помилки, рахує навантаження та формує звіт для адміністратора хмарної інфраструктури.

```
from dataclasses import dataclass, field
from typing import Dict, List, Optional
import random
@dataclass
class TargetServer:
    name: str
```

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		67

```

        address: str
        port: int
        protocol: str
        active_connections: int = 0
        healthy: bool = True
        response_time_ms: int = 30
        error_count: int = 0

    @dataclass
    class ClientRequest:
        client_ip: str
        path: str
        protocol: str
        port: int

    @dataclass
    class BalanceResult:
        balancer_type: str
        target_name: str
        target_address: str
        status: str
        response_time_ms: int

    @dataclass
    class MonitoringReport:
        total_requests: int = 0
        successful_requests: int = 0
        failed_requests: int = 0
        alb_requests: int = 0
        nlb_requests: int = 0
        events: List[str] = field(default_factory=list)

#Опис функціоналу
#Клас реалізує прикладний балансувальник для HTTP та HTTPS запитів
    class ApplicationLoadBalancer:
        def __init__(self, targets: Dict[str, List[TargetServer]]):
            self.targets = targets
            self.round_robin_index: Dict[str, int] = {}

        def route_request(self, request: ClientRequest) ->
Optional[TargetServer]:
            service_key = self.define_service(request.path)
            available_targets = self.get_healthy_targets(service_key)
            if not available_targets:
                return None
            current_index = self.round_robin_index.get(service_key, 0)
            selected = available_targets[current_index %

```

```

len(available_targets)]
        self.round_robin_index[service_key] = current_index + 1
        return selected

    def define_service(self, path: str) -> str:
        if path.startswith("/api"):
            return "api"
        if path.startswith("/media"):
            return "media"
        return "web"

    def get_healthy_targets(self, service_key: str) -> List[TargetServer]:
        return [target for target in self.targets.get(service_key, []) if
target.healthy]

#Опис функціоналу
#Клас реалізує мережевий балансувальник для TCP та UDP трафіку
class NetworkLoadBalancer:
    def __init__(self, targets: List[TargetServer]):
        self.targets = targets
    def route_request(self, request: ClientRequest) ->
Optional[TargetServer]:
        available_targets = [
            target for target in self.targets
            if target.healthy and target.protocol == request.protocol and
target.port == request.port
        ]
        if not available_targets:
            return None
        return min(available_targets, key=lambda target:
target.active_connections)

#Опис функціоналу
#Клас об'єднує ALB NLB моніторинг і перевірку стану серверів
class CloudLoadBalancingSystem:
    def __init__(self):
        self.report = MonitoringReport()
        self.alb = ApplicationLoadBalancer({
            "web": [
                TargetServer("web-server-1", "10.0.1.11", 443, "HTTPS"),
                TargetServer("web-server-2", "10.0.1.12", 443, "HTTPS")
            ],
            "api": [
                TargetServer("api-server-1", "10.0.2.21", 443, "HTTPS"),
                TargetServer("api-server-2", "10.0.2.22", 443, "HTTPS")
            ],
        })

```

```

        "media": [
            TargetServer("media-server-1", "10.0.3.31", 443, "HTTPS")
        ]
    })
    self.nlb = NetworkLoadBalancer([
        TargetServer("tcp-service-1", "10.0.4.41", 3306, "TCP"),
        TargetServer("tcp-service-2", "10.0.4.42", 3306, "TCP"),
        TargetServer("udp-service-1", "10.0.5.51", 53, "UDP")
    ])

    def process_request(self, request: ClientRequest) -> BalanceResult:
        self.report.total_requests += 1
        if request.protocol in ["HTTP", "HTTPS"]:
            self.report.alb_requests += 1
            target = self.alb.route_request(request)
            balancer_type = "Application Load Balancer"
        else:
            self.report.nlb_requests += 1
            target = self.nlb.route_request(request)
            balancer_type = "Network Load Balancer"
        if target is None:
            self.report.failed_requests += 1
        self.report.events.append("Запит не має доступної цільової групи")
        return BalanceResult(balancer_type, "Не призначено", "0.0.0.0", "Помилка", 0)
        target.active_connections += 1
        response_time = self.simulate_response(target)
        target.active_connections -= 1
        if response_time > 120:
            target.error_count += 1
            self.report.failed_requests += 1
        self.report.events.append(f"Висока затримка на сервері {target.name}")
        status = "Попередження"
    else:
        self.report.successful_requests += 1
        status = "Успішно"
    return BalanceResult(balancer_type, target.name,
                        target.address, status, response_time)

    def simulate_response(self, target: TargetServer) -> int:
        dynamic_delay = random.randint(5, 90)
    return target.response_time_ms+dynamic_delay+target.active_connections * 2

    def health_check(self) -> None:
        all_targets = []
        for group in self.alb.targets.values():

```

```

        all_targets.extend(group)
    all_targets.extend(self.nlb.targets)
    for target in all_targets:
        if target.error_count >= 3 or target.response_time_ms > 150:
            target.healthy = False
self.report.events.append(f"Сервер {target.name} позначено як проблемний")
        else:
            target.healthy = True
def generate_summary(self) -> str:
    return (
        f"Усього запитів {self.report.total_requests}\n"
        f"Успішних запитів {self.report.successful_requests}\n"
        f"Помилкових запитів {self.report.failed_requests}\n"
        f"Запитів через ALB {self.report.alb_requests}\n"
        f"Запитів через NLB {self.report.nlb_requests}\n"
        f"Події системи {len(self.report.events)}"
    )
#Опис функціоналу
#Функція демонструє роботу системи балансування навантаження
def main() -> None:
    system = CloudLoadBalancingSystem()
    requests = [
        ClientRequest("192.168.1.10", "/", "HTTPS", 443),
        ClientRequest("192.168.1.11", "/api/users", "HTTPS", 443),
        ClientRequest("192.168.1.12", "/media/video", "HTTPS", 443),
        ClientRequest("192.168.1.13", "", "TCP", 3306),
        ClientRequest("192.168.1.14", "", "UDP", 53)
    ]
    for request in requests:
        result = system.process_request(request)
        print(result)
    system.health_check()
    print(system.generate_summary())

if __name__ == "__main__":
    main()

```

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 Кнудсенем. 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 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 невикробувані 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

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		72

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).

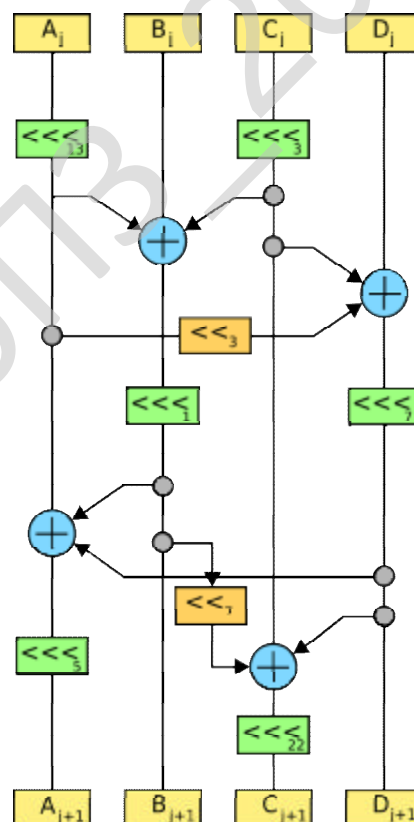


Рисунок 4.3 – Структура алгоритму Serpent

Serpent шифрує відкритий текст P довжиною 128 біт в шифротекст C довжиною таких же 128 біт за 32 раунд за допомогою 33 підключів $K\{0\}, \dots, K\{32\}$, 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.
- 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.

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 заміни)

В алгоритмі Serpent таблиці заміни 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 LT задається 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 заміни 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.

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 простіший, ніж стандартний опис. Ніяких початкових і кінцевих перестановок не потрібно.

Шифрування складається з 32 раундів. Відкритий текст є першими проміжними даними $V_0 = P$. Потім виконується 32 раунди, кожен i -й раунд складається з:

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

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		75

– 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.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		76

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. З рисунку головного вікна можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

- Верхнього меню: Файл; Налаштування; Довідка.
- Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші: Журнал системи дій; Налаштування.
- Розділу обрання групи дій: Обрання IP адреси; Налаштування.
- Розділу виведення результату роботи системи у вигляді вікна виведення результатів.

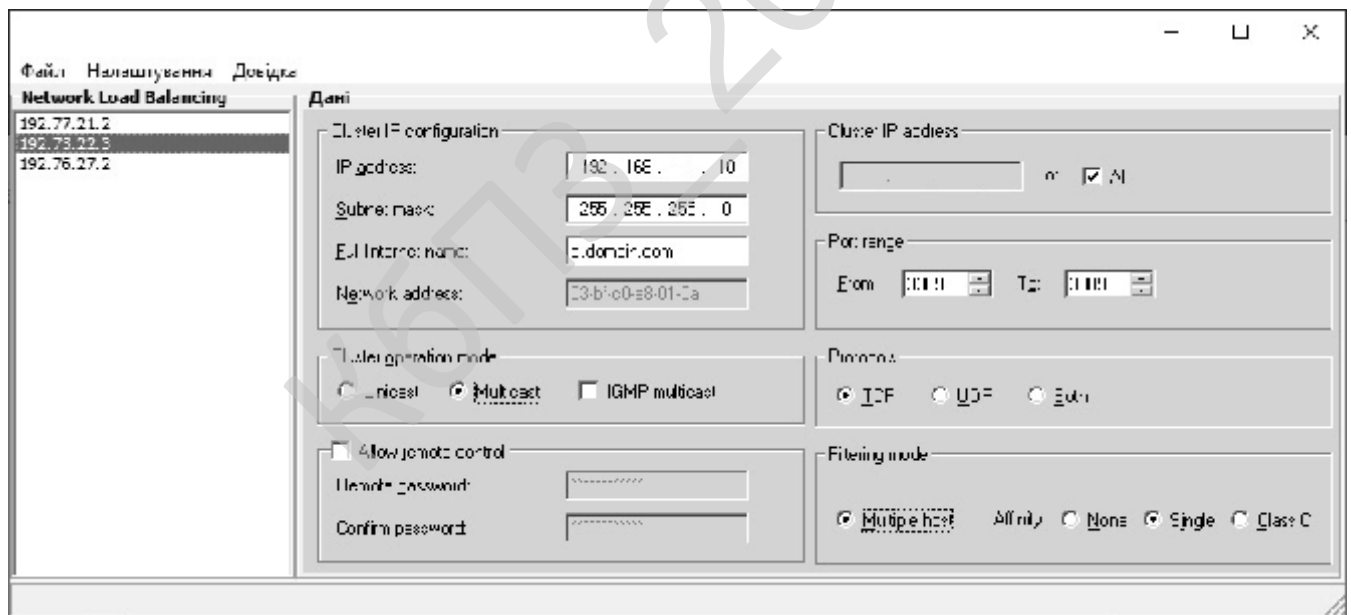


Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено отримані статистичні дані оформлені у вигляді веб звіту.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		77

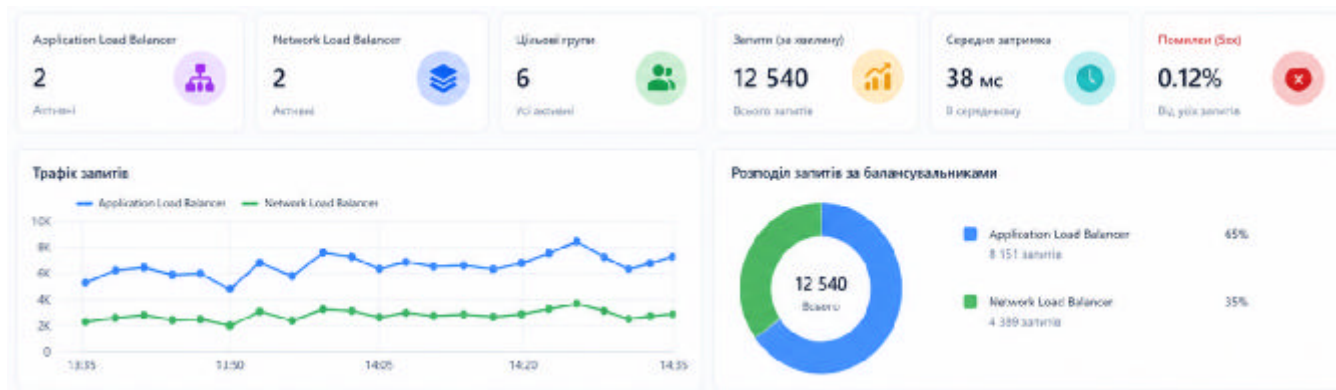


Рисунок 5.2 – Статистичні дані системи

На рисунку 5.3 зображено авторські дані розробленого програмного забезпечення. Розроблена програма має дуже простий і 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.

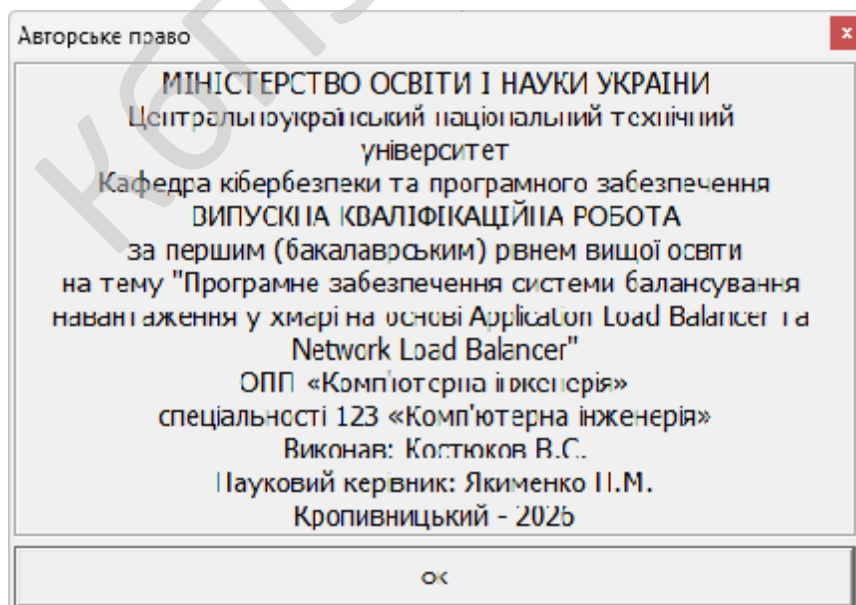


Рисунок 5.2 – Авторське право

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.

Таким чином у результаті вищерозглянутого можна стверджувати що розроблено інтерфейс системи у відповідності з вибраною метою роботи. Система містить максимальний необхідний набір функцій придатних для виконання будь-яких дій для забезпечення повноцінної роботи програми. Далі розглянемо висновки та використані літературні джерела.

КБПЗ-2026

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		79

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

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

Рішення завдання полягало у вирішенні наступних задач:

- Був проведений огляд існуючих систем балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.
- Досліджена система балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.
- На основі отриманих результатів досліджень створена програмна реалізація системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer.

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 створенні 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

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		80

системи балансування навантаження у хмарі на основі Application Load Balancer та Network Load Balancer. 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 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.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докum.	Підпис	Дата		81

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Е. Таненбаум, Д. Уезеролл «Комп'ютерні мережі». – [5-е вид.]. – 2016. – 960 с.
2. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.
3. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.
4. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.
5. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.
6. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.
7. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.
8. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп'ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». *Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка» (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.)*. Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.
9. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». *International Review on Modelling and Simulations* 18 (1), 2025. pp. 32-42.
10. Смірнова Т.В., Коноплицька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		82

технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка*. 2024. №4(24), С. 6-27.

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

12. 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.

13. 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.

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

15. 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.

16. 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.

17. 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.

18. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		83

телекомунікаційних систем». *Проблеми інформатизації та управління*, № 2(70). 2022. С. 28-37.

19. 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.

20. 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.

21. 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.

22. 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.

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

24. 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.

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		84

25. 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.
26. 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.
27. 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.
28. 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.
29. 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.
30. Smirnov O. Kuznetsov A., Zaichenko 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.
31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. Smirnov A.A., Kuznetsov 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.

40. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

41. 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.

42. 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.

43. Смірнов О.А., Коноплицька-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.

44. 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.

45. Смірнов О.А., Дреєва Г.М., Дреєв 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.

46. Смірнов О.А., Коноплицька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних

					ВКРБ-123.26.0031.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		87

систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

47. 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).

48. Смірнов О.А., Дреєва 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

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

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

51. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Коноплицька-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.

52. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

53. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

54. Смірнов О.А., Смірнов С.А., Дідик А.К., Дреєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". - Випуск 5 (142). - Х.: ХУПС - 2016. - С. 148-152.

55. Смірнов О.А., Смірнов С.А. Дідик А.К., Дреєв О.М. Моделі системи нейромережових експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.

56. Смірнов О.А., Смірнов С.А., Дідик А.К., Дреєв А.М. Спосіб контролю ліній зв'язку телекомунікаційної системи антивірусу. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. - 2016. - С. 121-127.

57. Смірнов О.А., Смірнов С.А., Дідик А.К. Метод безпечної маршрутизації метаданих у хмарні антивірусні системи. Системи озброєння та військова техніка. - Випуск 2 (46) - Х.: ХУПС - 2016. - С. 146-149.