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
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-22М-2
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Дібрівний О.С.
«____» _____ 2023 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Дібрівному Олександру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі*

2. Керівник роботи *Петренюк Володимир Ілліч, канд. фіз.-мат. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 33-13 від 04.08.2023 року

3. Строк подання студентом роботи до захисту *10.12.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі*

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

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2023	14.11.2023
Охорона праці	Оришака О.В.	06.10.2023	16.11.2023

7. Дата видачі завдання « 6 » вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
« 6 » вересня 2023 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2023 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Дібрівний О.С. Дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Метою розробки є дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Об'єктом дослідження є процес динамічного перерозподілу трафіку вузлів мережі у хмарі.

Предметом дослідження є методи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Методи дослідження базуються на методах хмарних технологій, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

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 архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Delphi 10.4.1.

Ключові слова: комп'ютерні науки, трафік

ABSTRACT

Dibrivnyi O.S. Research and software implementation of a system of dynamic traffic redistribution of network nodes in the cloud. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the system of dynamic redistribution of traffic of network nodes in the cloud.

The purpose of the development is research and software implementation of a system of dynamic traffic redistribution of network nodes in the cloud.

The object of research is the process of dynamic traffic redistribution of network nodes in the cloud.

The subject of research is the methods of dynamic redistribution of traffic of network nodes in the cloud.

Research methods are based on methods of cloud technologies, methods of mathematical statistics, methods of software development.

The result of the work is a software implementation of a system of dynamic traffic redistribution of network nodes in the cloud.

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 tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Delphi 10.4.1 environment.

Keywords: computer science, traffic

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ДПТМ	–	динамічний перерозподіл трафіку мережі
ЕЦП	–	електронний цифровий підпис
ІВК	–	інфраструктура відкритих ключів
ІТ	–	інформаційні технології
ЛОМ	–	локальна обчислювальна мережа
ПЗ	–	програмне забезпечення
СКЗІ	–	система комплексного захисту інформації
УК	–	управляючі компоненти
УЦ	–	удостовірюючий центр
IGMP	–	Internet Group Management Protocol
NLB	–	Network Load Balancing, балансування мережного навантаження
PKI	–	Public Key Infrastructure
VPN	–	віртуальна приватна мережа

ВСТУП

Актуальність теми. На сьогоднішній день темпи розвитку галузі телекомунікацій є одними із самих стрімких. Поряд зі зниженням темпів росту клієнтської бази операторів зв'язку, спостерігається ріст трафіку за рахунок впровадження нових технологій і збільшення частки послуг на базі IP-технологій. З огляду на зазначені тенденції, оператори зв'язку, впроваджують нові послуги, що приводить до переходу телекомунікаційних мереж до мультисервісності.

У свою чергу це накладає деякі обмеження на функціонування телекомунікаційних мереж. Виникає необхідність виконання вимог якості обслуговування – Quality of Service (QoS), які для різних класів трафіку найчастіше не тільки відрізняються, але й суперечать один одному. Для одночасного забезпечення різних вимог QoS у систему зв'язку потрібно впроваджувати системи управління трафіком, які у свою чергу повинні враховувати особливості різних класів трафіку й забезпечувати ефективний перерозподіл ресурсів мережі.

У цей час для користувачів мультисервісных мереж все більший інтерес представляють такі служби, як відеоконференц-зв'язок, доступ до web-служб. Але методологія забезпечення вимог по якості обслуговування різнорідного трафіку не є до кінця вирішеною.

Для рішення цієї проблеми необхідне впровадження класів обслуговування для різних видів трафіків, а також систему управління трафіком, що буде забезпечувати задані класи обслуговування для різних видів трафіку шляхом перерозподілу мережного ресурсу. У даний момент тема управління трафіком є дуже актуальною, так як поки що немає єдиного підходу до рішення цієї проблеми.

Одним з варіантів забезпечення ефективної передачі трафіку з підтримкою параметрів QoS (Quality of Service) є використання технології багатопротокольної

комутації міток MPLS (MultiProtocol Label Switching). Це реалізується за допомогою технологій трафіку інжинірингу Traffic Engineering (TE) за рахунок використання механізмів збалансованого завантаження ресурсів мережі, вибору оптимального маршруту проходження трафіку, використання процедур резервування й розподіли завантаження мережі, балансування трафіку й механізмів запобігання перевантажень.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

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

– Огляд існуючих систем динамічного перерозподілу трафіку вузлів мережі у хмарі.

– Дослідження системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

– Програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Об'єктом дослідження є процес динамічного перерозподілу трафіку вузлів мережі у хмарі.

Предметом дослідження є методи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Методи дослідження базуються на методах хмарних технологій, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод динамічного перерозподілу трафіку вузлів мережі у хмарі.

– Розроблено вітчизняний продукт динамічного перерозподілу трафіку вузлів мережі у хмарі, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі динамічного перерозподілу трафіку вузлів мережі у хмарі.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2023, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №14.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для реалізації програмного забезпечення динамічного перерозподілу трафіку вузлів мережі для забезпечення потрібного рівня Quality of Service (QoS). Відповідність до стандартів є лише однією з багатьох вимог, що пред'являються до сучасних мереж. Важливішим є виконання мережею певного набору послуг, наприклад, надання доступу до файлових архівів або веб-сторінок публічних Internet-сайтів, обмін електронною поштою в межах підприємства або в глобальних масштабах, інтерактивний обмін голосовими повідомленнями IP-телефонії тощо. Решта вимог – продуктивність, надійність, сумісність, керованість, захищеність, розширюваність і масштабованість – пов'язані з якістю виконання цього основного завдання.

І хоча всі перераховані вище вимоги є важливими, часто поняття «Якість обслуговування» (Quality of Service, QoS) комп'ютерної мережі трактується вужче: воно містить лише дві важливі характеристики мережі – продуктивність і надійність.

Швидкість передачі трафіку

Середня швидкість обчислюється шляхом ділення загального об'єму переданих даних на час їх передачі, зазвичай обирається достатньо тривалий проміжок часу – година, день або тиждень.

Миттєва швидкість відрізняється від середньої тим, що для усереднення вибирається дуже маленький проміжок часу – наприклад, 10 мс або 1 с.

Максимальна швидкість – це найбільша швидкість передачі, що зафіксована протягом періоду спостереження. Зазвичай, при проектуванні, налаштуванні і оптимізації мережі використовуються такі показники, як середня і максимальна швидкість. Середня швидкість, з якою обробляє трафік окремий елемент мережі або мережа в цілому, дозволяє оцінити роботу мережі впродовж

тривалого часу, протягом якого піки і спади інтенсивності трафіку компенсують один одного. Максимальна швидкість дозволяє оцінити, як мережа буде долати пікові навантаження, які є характерними для особливих періодів роботи, наприклад в ранкові години, коли співробітники підприємства майже одночасно реєструються в мережі і звертаються до розподілених файлів чи баз даних.

Зазвичай, при визначенні швидкісних характеристик певного сегменту чи пристрою в переданих даних не виділяється трафік певного користувача, застосування або комп'ютера – обчислюється загальний об'єм переданої інформації. Проте, для точної оцінки якості обслуговування така деталізація є бажаною, і системи керування мережами дозволяють її виконувати.

Продуктивність

Потенційно висока продуктивність – це одна з основних переваг розподілених систем, до яких відносяться комп'ютерні мережі. Ця властивість забезпечується принциповою можливістю розподілу робіт.

Основні характеристики продуктивності мережі:

– Час реакції мережі.

– Швидкість передачі трафіку.

– Пропускна здатність.

– Затримка передачі і варіанти затримки передачі.

– Час реакції мережі

В загальному випадку час реакції визначається як інтервал між відправленням запиту користувача до мережної служби і отриманням відповіді на нього.

Час реакції мережі є інтегральною характеристикою продуктивності мережі з погляду користувача. Саме цю характеристику має на увазі користувач, коли говорить: «Сьогодні мережа працює поволі».

Значення цього показника залежить від типу служби, до якої звертається користувач, від того, який користувач і до якого сервера звертається, а також від поточного стану елементів мережі – завантаженості сервера та сегментів, комутаторів і маршрутизаторів, через які проходить запит.

Тому, варто усереднювати цей показник по користувачах, серверах і годинах доби (від чого в значній мірі залежить завантаження мережі).

Час реакції мережі зазвичай складається з кількох складових:

– Час підготовки запитів на клієнтському комп'ютері.

– Час передачі запитів між клієнтом і сервером через сегменти мережі і проміжне комунікаційне устаткування.

– Час обробки запитів на сервері.

– Час передачі відповідей від сервера до клієнта.

– Час обробки отриманих від сервера відповідей на клієнтському комп'ютері.

Очевидно, що розкладання часу реакції на складові користувачу не є цікавим, головним є кінцевий результат. Проте, для мережного фахівця дуже важливим є виділення з загального часу реакції складових, що відповідають етапам власне мережної обробки даних, – передачі даних від клієнта до сервера через сегменти мережі і комунікаційне обладнання.

Знання мережних складових часу реакції дозволяє оцінити продуктивність окремих елементів мережі, виявити вузькі місця і за необхідності виконати модернізацію мережі для підвищення її загальної продуктивності.

Пропускна здатність

Пропускна здатність – це максимально можлива швидкість обробки трафіку, що визначена стандартом технології, на якій побудована мережа. Пропускна здатність відображає максимально можливий об'єм даних, що передається по мережі або її частині в одиницю часу.

Пропускна здатність не є характеристикою, призначеною для користувача, оскільки вона свідчить про швидкість виконання внутрішніх операцій мережі – передачі пакетів даних між вузлами мережі через різні комунікаційні пристрої. Вона безпосередньо характеризує якість виконання основної функції мережі – транспортування повідомлень, тому її частіше використовують при аналізі продуктивності мережі, ніж час реакції або швидкість.

Пропускна здатність вимірюється або в бітах в секунду, або в пакетах в секунду.

Пропускна здатність мережі залежить як від характеристик фізичного середовища передачі (мідний кабель, оптичне волокно, скручена пара) так і від наявного способу передачі даних (технологія Ethernet, FastEthernet, ATM). Пропускна здатність часто використовується як характеристика не стільки мережі, скільки власне технології, на якій побудована мережа.

На відміну від часу реакції або швидкості передачі трафіку пропускна здатність не залежить від завантаженості мережі і має постійне значення, що визначається використаними в мережі технологіями.

На різних ділянках гетерогенної мережі, де використовується кілька різних технологій, пропускна здатність може бути різною. Для аналізу і налаштування мережі корисно знати дані про пропускну здатність окремих її елементів.

Важливо відзначити, що із-за послідовного характеру передачі даних різними елементами мережі загальна пропускна здатність будь-якого складеного шляху в мережі буде мінімальною з пропускних здатностей елементів маршруту. Для підвищення пропускної здатності складеного шляху необхідно в першу чергу звернути увагу на найповільніші елементи.

Іноді корисно оперувати загальною пропускною здатністю мережі, яка визначається як середня кількість інформації, що передається між всіма вузлами мережі за одиницю часу. Цей показник характеризує якість мережі в цілому, не розкладаючи його по окремих сегментах або пристроях.

Затримка передачі

Затримка передачі визначається як час між моментом надходження даних на вхід мережного пристрою або частини мережі та моментом появи їх на виході цього пристрою.

Цей параметр продуктивності за сенсом є близьким до часу реакції мережі, але відрізняється тим, що завжди характеризує лише мережні етапи обробки даних, без затримок обробки кінцевими вузлами мережі.

Звичайну якість мережі характеризують величинами максимальної затримки передачі і варіантом затримки. Не всі типи трафіку є чутливими до затримок передачі, оскільки, зазвичай затримки не перевищують сотень мілісекунд, рідше – кількох секунд.

Затримки пакетів, що зумовлені файловою службою, службою електронної пошти або службою друку, мало впливають на якість цих служб з погляду користувача мережі.

З іншого боку, затримки пакетів, що містять голосові або відеодані, можуть призводити до значного зниження якості наданої користувачу інформації – виникає ефект «відлуння», неможливість розібрати деякі слова, вібрації зображення тощо.

Всі вказані характеристики продуктивності мережі є достатньо незалежними. Тоді як пропускна здатність мережі є постійною величиною, швидкість передачі трафіку може коливатися в залежності від завантаження мережі, не перевищуючи встановлених меж пропускної здатності.

1.2 Область застосування

Областю застосування є комп'ютерна мережа. Комп'ютерна мережа – це система розподіленої обробки інформації між комп'ютерами за допомогою засобів зв'язку. Комп'ютерна мережа являє собою сукупність територіально рознесених комп'ютерів, здатних обмінюватися між собою повідомленнями через середовище передачі даних. Передача інформації між комп'ютерами відбувається за допомогою електричних сигналів, які бувають цифровими та аналоговими. У комп'ютері використовуються цифрові сигнали у двійковому вигляді, а під час передачі інформації по мережі – аналогові (хвильові). Частота аналогового сигналу – це кількість виникнень хвилі у задану одиницю часу. Аналогові сигнали також використовуються модеми, які двійковий нуль перетворюють у сигнал низької частоти, а одиницю – високої частоти.

Комп’ютери підключаються до мережі через вузли комутації. Вузли комутації з’єднуються між собою канали зв’язку. Вузли комутації разом з каналами зв’язку утворюють середовище передачі даних. Комп’ютери, підключені до мережі, у літературі називають вузлами, абонентськими пунктами чи робочими станціями. Комп’ютери, що виконують функції керування мережею чи надають які-небудь мережеві послуги, називаються серверами. Комп’ютери, що користуються послугами серверів, називаються клієнтами.

Кожен комп’ютер, підключений до мережі, має ім’я (адресу). Комп’ютерні мережі можуть обмінюватися між собою інформацією у вигляді повідомлень. Природа цих повідомлень може бути різна (лист, програма, книга і т.д.). У загальному випадку повідомлення по шляху до абонента-одержувача проходить декілька вузлів комутації. Кожний з них, аналізуючи адресу одержувача в повідомленні і володіючи інформацією про конфігурацією мережі, вибирає канал зв’язку для наступного пересилання повідомлення. Таким чином, повідомлення “подорожує” по мережі, поки не досягає абонента-одержувача.

Для підключення до мережі комп’ютери повинні мати:

– апаратні засоби, що з’єднують комп’ютери із середовищем передачі даних;

– мережеве програмне забезпечення, за допомогою якого здійснюється доступ до послуг мережі.

У світі існують тисячі різноманітних комп’ютерних мереж. Найбільш істотними ознаками, що визначають тип мережі, є ступінь територіального розосередження, топологія і застосовані методи комутації.

По ступеню розосередження комп’ютерні мережі поділяються на локальні, регіональні і глобальні.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Бувають ситуації, коли необхідно мати статистику прийнятої й переданої інформації й контролювати кількість трафіку. Для таких випадків і були придумані програми для виміру, контролю й обмеження інтернет-трафіку. Серед них є платні й безкоштовні, одні справляються зі своїми завданнями краще, а інші небагато гірше. У кожної з них є свої плюси й мінуси.

NetWorx

Для початку розглянемо невелику програму обліку трафіку – NetWorx. Вона дозволяє вести облік вхідні й вихідного трафіку, робить моніторинг швидкості інтернет-підключення, може працювати з кабельним і бездротовим підключеннями. Поширюється безкоштовно і є можливість вибрати російську мову.

Для зміни налаштувань і керування NetWorx, необхідно кликнути правою кнопкою мишки по значку в треї

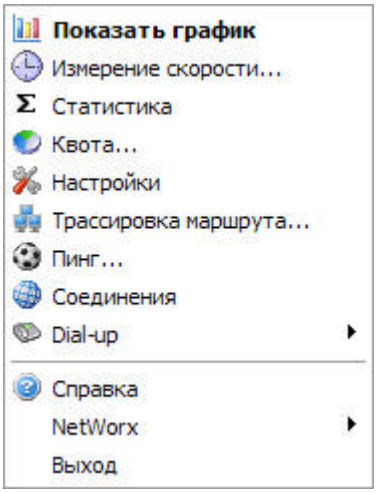

Рисунок 2.1 – Інтерфейс користувача програми NetWorx

Вибравши розділ «Налаштування» відкриється кілька вкладок, що дозволяють зробити додаткові зміни. На вкладці «Загальні» можна включити або виключити автоматичне відновлення, настроїти виведену інформацію в треї, одиниці виміру й множник, а також вибрати підключення, за яким необхідно вести спостереження.

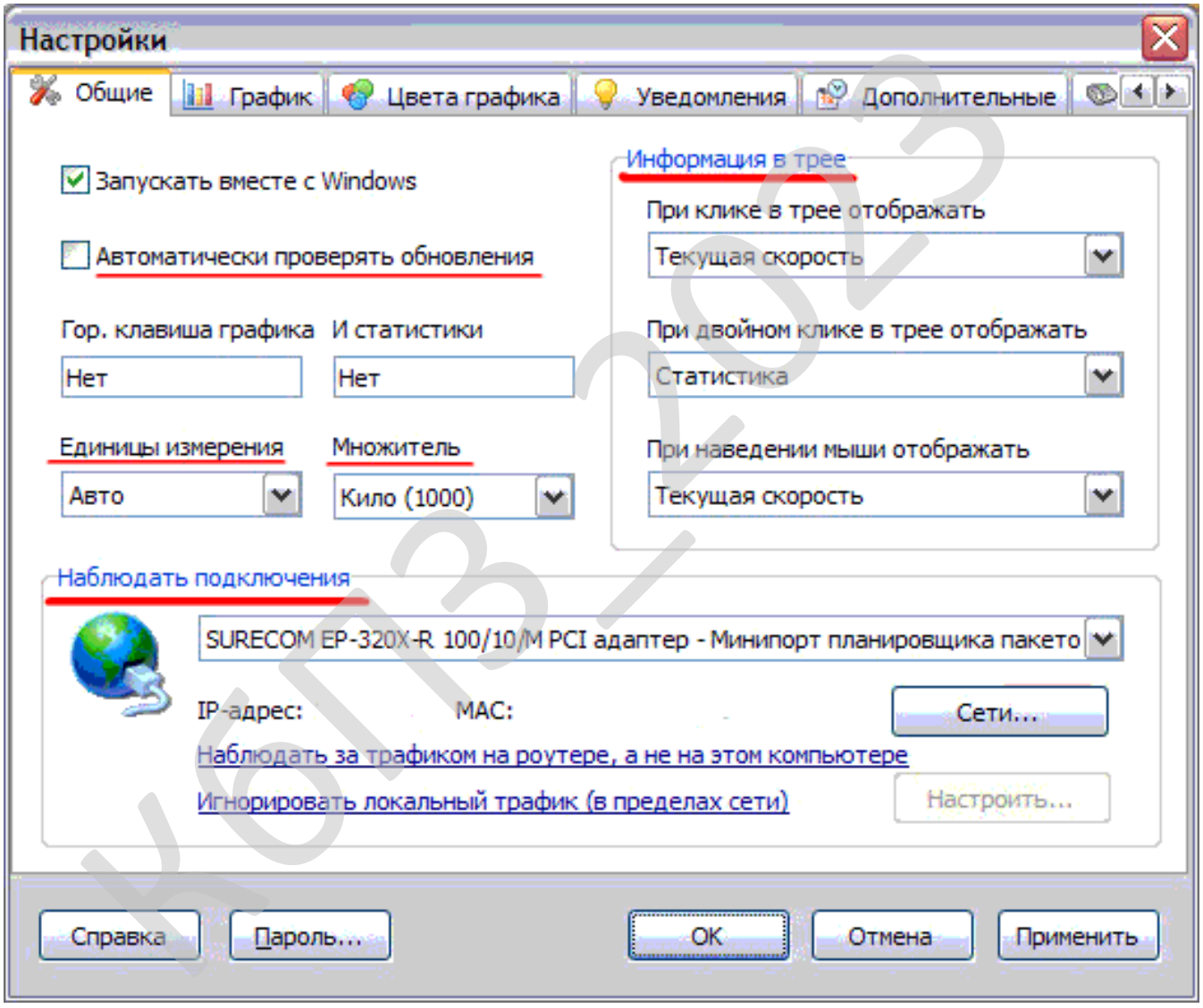

Рисунок 2.2 – Інтерфейс користувача програми NetWorx: закладка «Загальні»

«Повідомлення» – дозволяє настроїти повідомлення й дії цієї програми.

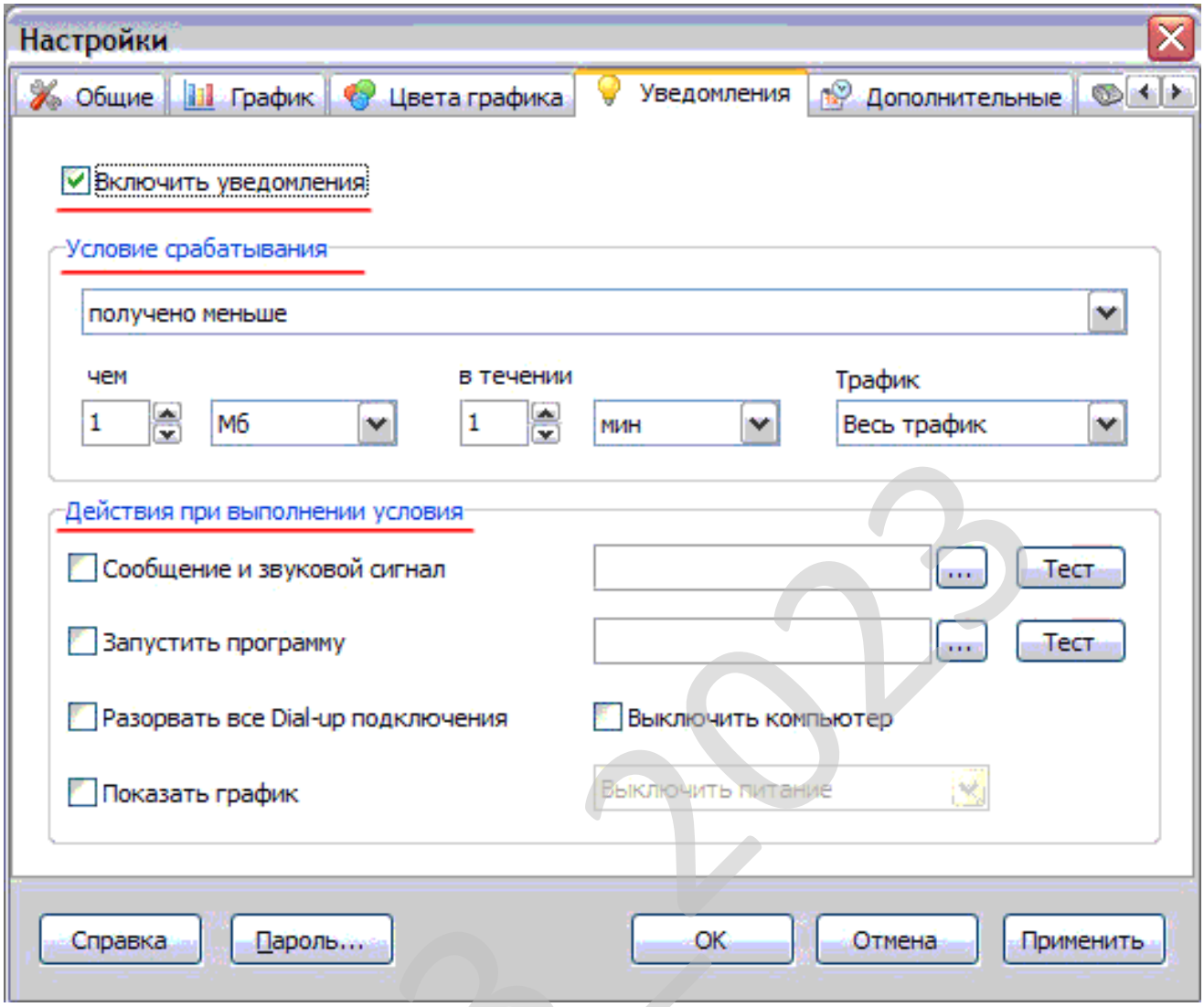

Рисунок 2.3 – Інтерфейс користувача програми NetWorx: закладка «Повідомлення»

Розділ «З'єднання», відображає всі додатки, що вимагають підключення до Internet і дозволяє контролювати їхню мережну активність.

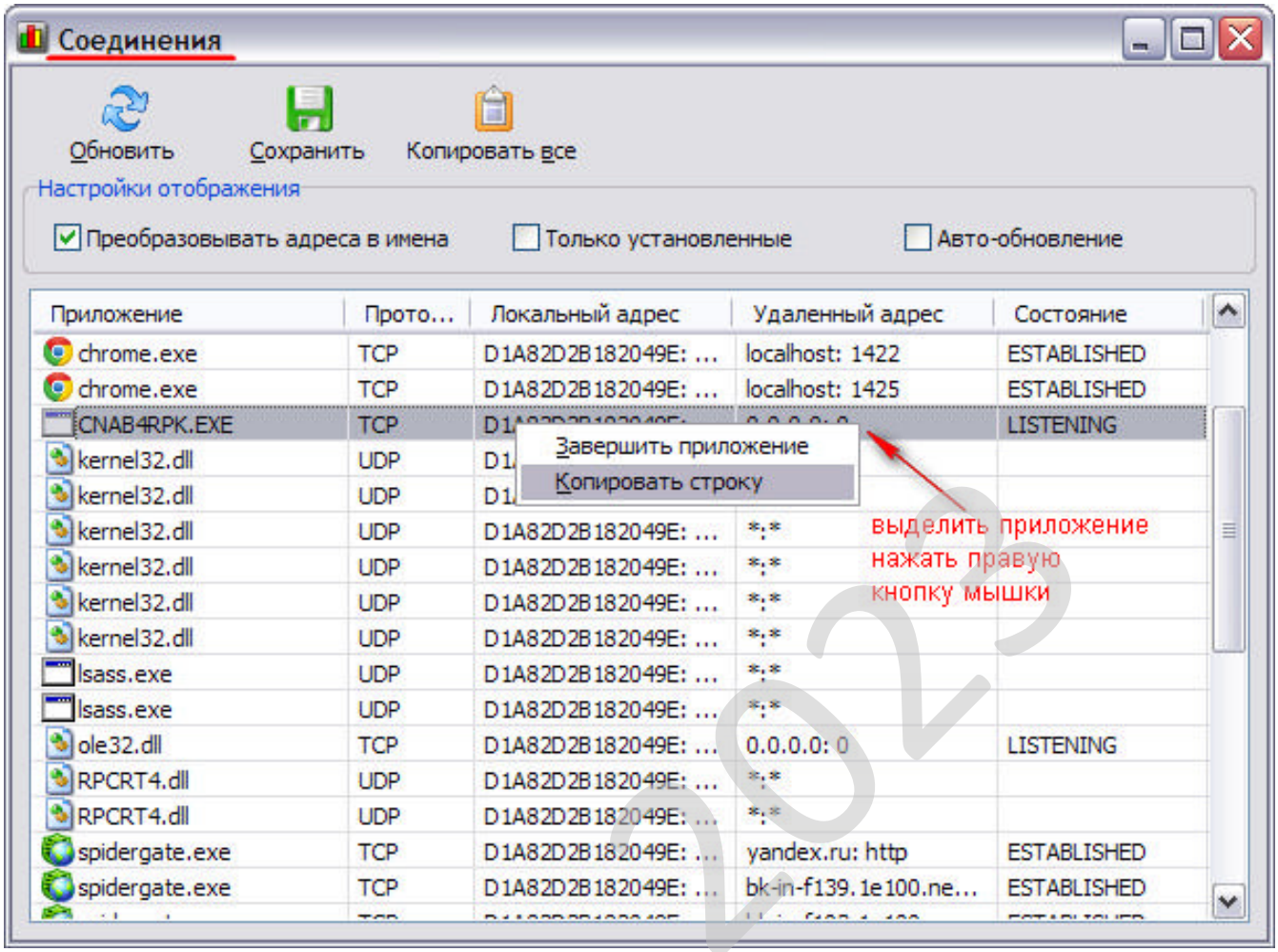

Рисунок 2.4 – Інтерфейс користувача програми NetWorx: закладка «З'єднання»

Налаштування програми NetWorx нескладні й розібратися в них особливої праці не складе. Скачати цю утиліту можна за адресою: http://www.softperfect.com/products/networx.

DUTraffic

Наступна програма обліку трафіку, у цьому невеликому огляді, буде – DUTraffic. Вона є безкоштовною й при установці можна вибрати російський інтерфейс.

Але навіть якщо Ви цього не зробите на початку інсталяції, це можна зробити, зайшовши в «Параметри».

У категорію «Основні» і вибрати мову потім.

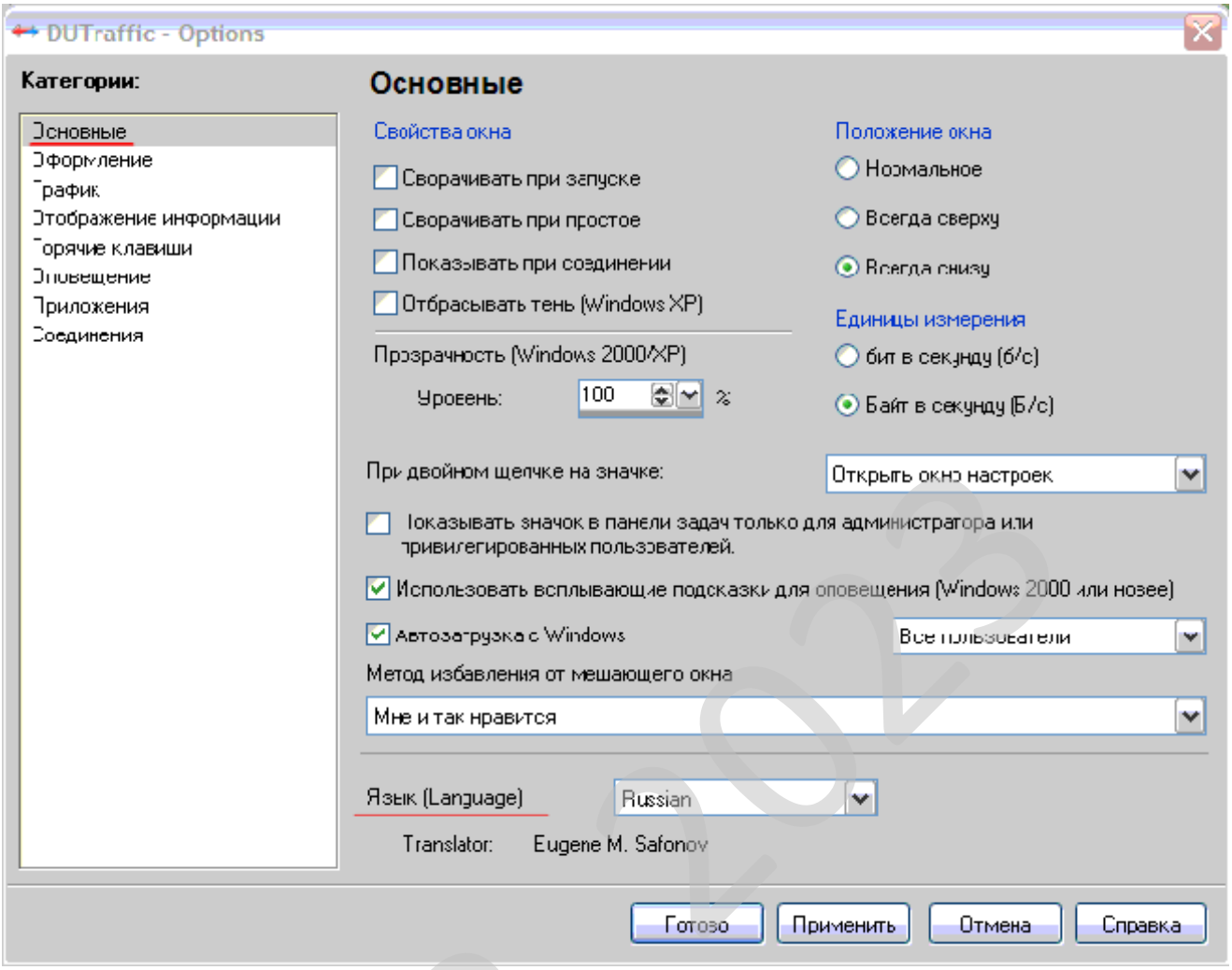

Рисунок 2.5 – Інтерфейс користувача програми DUTraffic

Програма дозволяє підраховувати час, проведений в Інтернеті й скільки на це Ви витратили грошей. Це можливо настроїти, нажавши на «З'єднання».

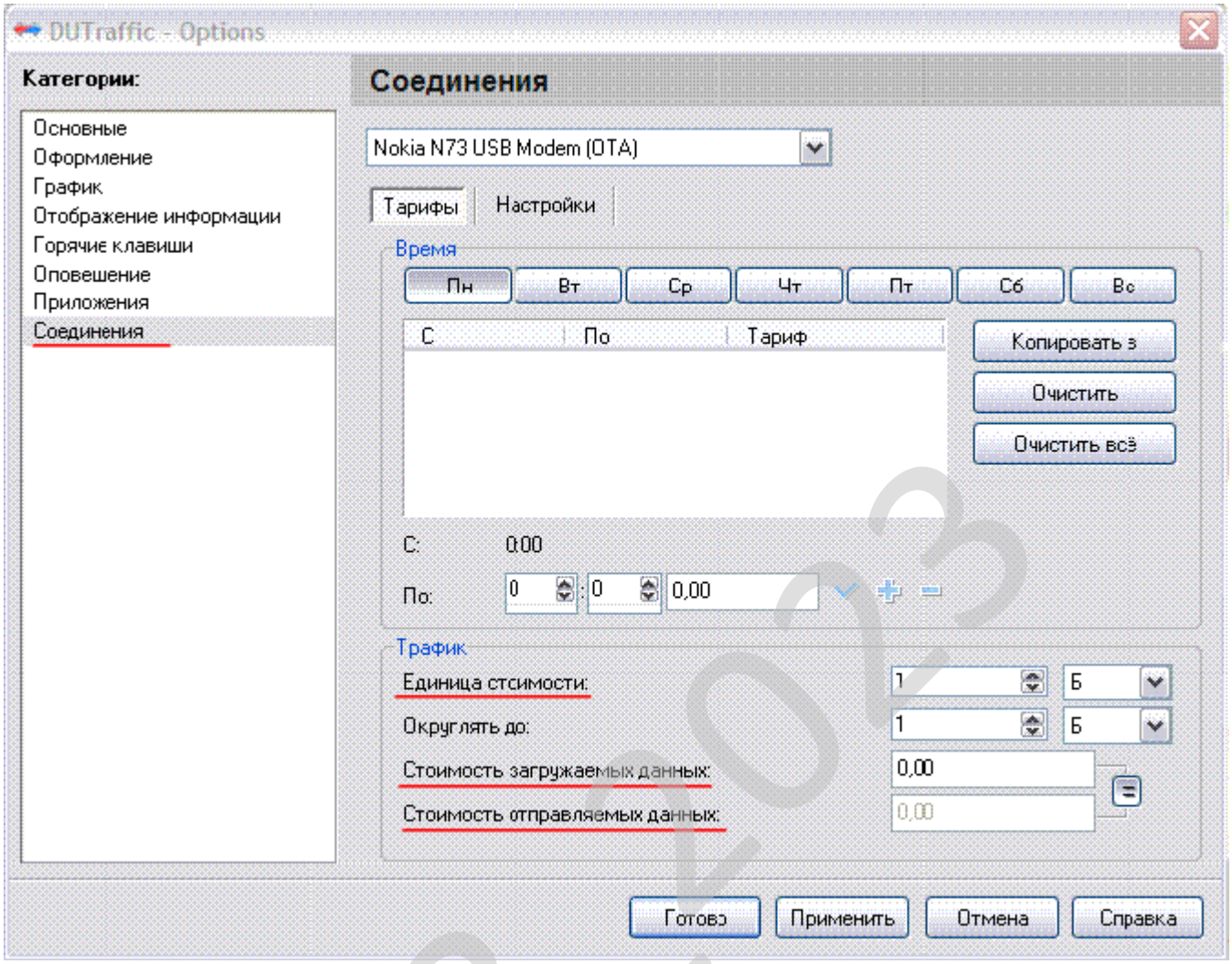

Рисунок 2.6 – Інтерфейс користувача програми DUTraffic: закладка «З'єднання»

Так само можна настроїти відтворення звуку, показати спливаюче повідомлення або розірвати з'єднання з мережею, при тих умовах, які ви задасте в категорії «Оповіщення». Крім цього DUTraffic дозволяє запускати додатка після установки з'єднання або закривати його після розриву зв'язку. Це налаштування доступне в «Додатках».

Woobind Network Meter

Пропоную Вам ще одну програму обліку трафіку – Woobind Network Meter. Вона призначена для обліку трафіку при підключенні за допомогою Dial Up або через локальну мережу. Споконвічно була розроблена для виходу в

Інтернет через GPRS, CDMA або 3 G-Модем. Woobind Network Meter може вести облік грошей при помегабайтній оплаті за трафік. Налаштування цієї функції виробляється в меню «Тарифи». Тут можна вказати вартість 1МБ даних, у яких грошових одиницях ведеться облік і вибрати напрямок що враховується трафіку.

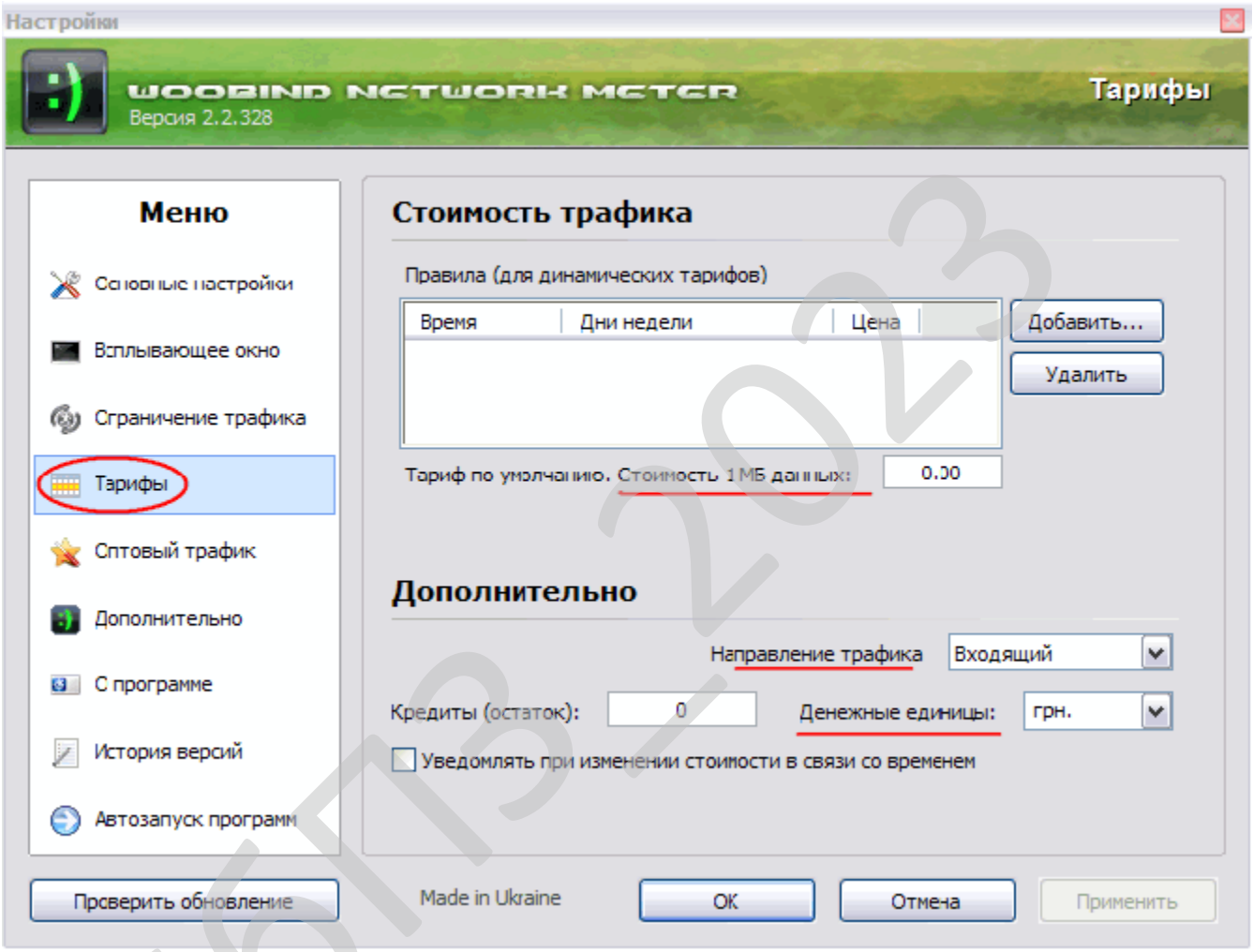

Рисунок 2.7 – Інтерфейс користувача програми Woobind Network Meter

Налаштування обмеження трафіку й включення режиму повідомлення виробляються в меню «Обмеження трафіку».

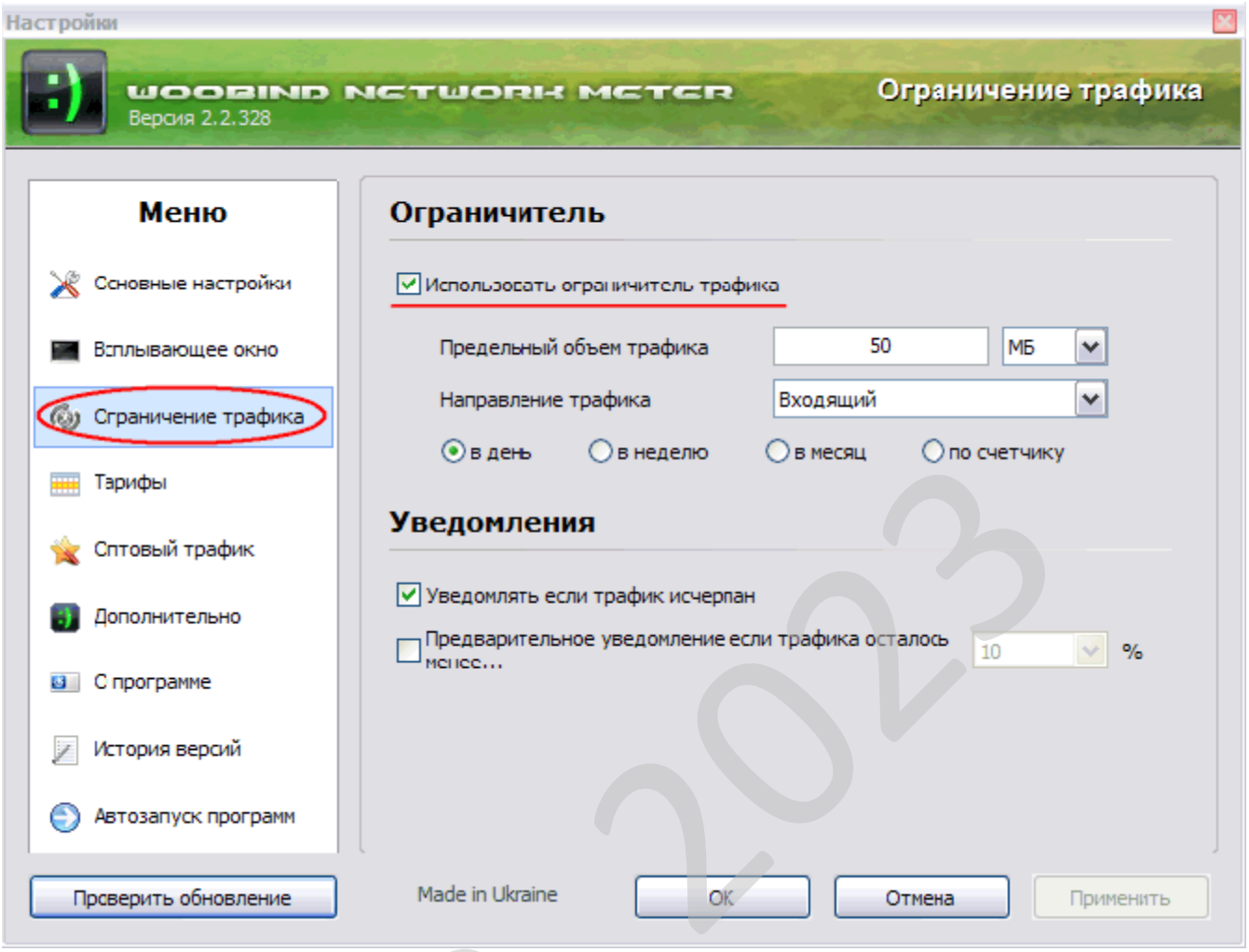

Рисунок 2.8 – Інтерфейс користувача програми Woobind Network Meter: закладка «Обмеження трафіку»

Ця програма веде статистику по прийнятим і переданим даним за день, тиждень і місяць. Одержати доступ до цієї інформації можна кликнувши лівою кнопкою мишки по іконці, розташованої в треї й вибравши внизу вкладку «Report». Програма Woobind Network Meter підтримує російську мову, поширюється безкоштовно й доступна за адресою: www.woobind.org.ua

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

− Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент. Це

розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

– Велика кількість поліпшень STL від Dinkumware.

– Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.

– Поліпшена підтримка Cmake.

– Велика кількість виправлень для підвищення стабільності і якості.

– Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне

забезпечення, яке призначено для системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) 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) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У цей час усе більше уваги приділяється питанням управління трафіком у мережах [1, 2, 3]. Існує безліч публікацій на тему оптимізації трафіку за допомогою технологій MPLS. Ця технологія постійно вдосконалюється в напрямку адаптації до умов передачі трафіку в мережах, забезпечуючи підтримку QoS. У рішенні цих завдань неоціненний внесок внесли такі дослідники як Вишневський В.М., Awduche D.O., Malcolm J.,Agogbua J., McManus J., M.S.Garey, D.S.Johnson, G.Cornuejols, M.L.Fisher, B.Fortz. R.Widyono,і ін. У статті [4] розглянуті проблеми управління трафіком в IP-мережах і засоби технології MPLS, їх вирішують.

Існуючі публікації, присвячені оптимізації мереж IP / MPLS [5, 6] мають складну практичну реалізацію й звичайно призначені для мереж на етапі проектування. У роботі [7] представлений варіант оптимізації мережі із застосуванням імітаційного моделювання.

Принцип роботи технології MPLS

MPLS (MultiProtocol Label Switching) – це технологія швидкої комутації пакетів у багатопротокольних мережах, заснована на використанні міток. MPLS розробляється й позиціонується як спосіб побудови високошвидкісних IP-магістралей, однак область її застосування не обмежується протоколом IP, а поширюється на трафік будь-якого маршрутизуємого мережного протоколу.

В основі MPLS лежить принцип обміну міток [8]. Будь-який переданий пакет асоціюється з тим або іншим класом мережного рівня (Forwarding Equivalence Class, FEC), кожний з яких ідентифікується певною міткою. Значення мітки унікально лише для ділянки шляху між сусідніми вузлами мережі MPLS, які називаються маршрутизаторами, комутуючими по мітках (Label Switching

Router, LSR). Мітка передається в складі будь-якого пакета, причому спосіб її прив'язки до пакета залежить від використовуваної технології канального рівня.

Маршрутизатор LSR одержує топологічну інформацію про мережу, беручи участь у роботі алгоритму маршрутизації – OSPF, BGP, IS-IS. Потім він починає взаємодіяти із сусідніми маршрутизаторами, розподіляючи мітки, які надалі будуть застосовуватися для комутації.

Розподіл міток між LSR приводить до встановлення усередині домену MPLS шляхів з комутацією по мітках (Label Switching Path, LSP). Кожний маршрутизатор LSR містить таблицю, що ставить у відповідність парі «вхідний інтерфейс, вхідна мітка» трійку «префікс адреси одержувача, вихідний інтерфейс, вихідна мітка». Одержуючи пакет, LSR по номері інтерфейсу, на який прийшов пакет, і за значенням прив'язаної до пакета мітки визначає для нього вихідний інтерфейс. Старе значення мітки заміняється новим, що містилось у полі «вихідна мітка» таблиці, і пакет відправляється до наступного пристрою на шляху LSP.

Вся операція вимагає лише одноразової ідентифікації значень полів в одному рядку таблиці. Це займає набагато менше часу, ніж порівняння IP-адреси відправника з найбільш довгим адресним префіксом у таблиці маршрутизації, що використовується при традиційній маршрутизації.

Мережа MPLS ділиться на дві функціонально різні області – ядро й гранична область. Ядро утворять пристрою, мінімальною вимогою до яких є підтримка MPLS і участь у процесі маршрутизації трафіку для того протоколу, що комутирується за допомогою MPLS. Маршрутизатори ядра займаються тільки комутацією.

Всі функції класифікації пакетів по різним FEC, а також реалізацію таких додаткових сервісів, як фільтрація, явна маршрутизація, вирівнювання навантаження й управління трафіком, беруть на себе граничні LSR. У результаті інтенсивні обчислення доводяться на граничну область, а високопродуктивна комутація виконується в ядрі, що дозволяє оптимізувати конфігурацію пристроїв MPLS залежно від їхнього місця розташування в мережі.

Таким чином, головна особливість MPLS – відділення процесу комутації пакета від аналізу IP-адрес у його заголовку, що відкриває ряд привабливих можливостей. Очевидним наслідком описаного підходу є той факт, що черговий сегмент LSP може не збігатися із черговим сегментом маршруту, що був би обраний при традиційній маршрутизації [9].

Оскільки на встановлення відповідності пакетів певним класам FEC можуть впливати не тільки IP-адреси, але й інші параметри, неважко реалізувати, наприклад, призначення різних LSP пакетам, що ставляться до різних потоків RSVP або маючим різні пріоритети обслуговування.

Кожний із класів FEC обробляється окремо від інших – не тільки тому, що для нього будується свій шлях LSP, але й у змісті доступу до загальних ресурсів (смузі пропущення каналу й буферному простору). У результаті технологія MPLS дозволяє дуже ефективно підтримувати необхідну якість обслуговування, не порушуючи наданих користувачеві гарантій. Застосування в LSR таких механізмів управління буферізацією і чергами, як WRED, WFQ або CBWFQ, дає можливість операторові мережі MPLS контролювати розподіл ресурсів і ізолювати трафік окремих користувачів. Так як для різних класів будується свій шлях залежно від смуги пропущення й завантаження каналу, то пакети прихожі на один маршрутизатор, але які мають різний клас обслуговування підуть по різному шляху.

Використання маршруту, що задається явно, у мережі MPLS вільно від недоліків стандартної IP-маршрутизації від джерела, оскільки вся інформація про маршрут утримується в мітці й пакету не потрібно нести адреси проміжних вузлів, що поліпшує управління розподілом навантаження в мережі. На граничному маршрутизаторі відбувається визначення класу обслуговування для пакета, що прийшов на нього. Відповідно до цього класу обслуговування, пакету призначається мітка. І відповідно до цієї мітки визначається наступний маршрутизатор, на який піде цей пакет. Там мітка переназначиться й піде на наступний маршрутизатор. Так буде відбувається доти, поки пакет не прийде на граничний маршрутизатор призначення. Там мітка вийде й піде до користувача.

Управління трафіком в MPLS-мережі

При оптимізації управління трафіком у мережах MPLS (MultiProtocol Label Switching), важливу роль грає технологія інжинірингу трафіку (Traffic Engineering, TE). В основу технології інжинірингу трафіку покладені ідеї балансування використання різнорідних мережних ресурсів – інформаційних, буферних і канальних. Ефективність технології Traffic Engineering підтверджується тим, що багато мережних засобів управління трафіком удосконалюються на її принципах, підтвердженням тому є протоколи резервування ресурсів RSVP-TE і LDP-TE, протоколи маршрутизації IS-IS-TE, OSPF-TE. Одним з найефективніших є підхід, заснований на балансуванні черг на принципах технології інжинірингу трафіку (Traffic Engineering Queues) [10, 11], запропонованого для управління чергами в MPLS-мережах.

Модель балансування черг із підтримкою Traffic Engineering Queues

Вважається, що число окремих трафіків або агрегированных по класах або пріоритетам потоків відомо й дорівнює M. Максимальне число черг на мережному вузлі також фіксоване й дорівнює N [9, 12].

Крім того, a_i (i = 1, M) – інтенсивність трафіку i-го класу, що надходить на обслуговування мережним вузлом; b_j (j = 1, N) – частина пропускної здатності вихідного КМ, що виділена j -й черзі (j = 1, N).

У ході управління чергами необхідно виконати умову відсутності перевантаження каналу зв'язку:

$$\sum_{j=1}^{N} b_j \le b\,, \tag{3.1}$$

де b – пропускна здатність вихідного КМ.

Крім того, з метою запобігання перевантаження мережного вузла необхідно забезпечити виконання наступної умови:

$$\sum_{i=1}^{M} a_i \le b\,. \tag{3.2}$$

Виконання умови (3.2) визначає необхідність превентивного обмеження інтенсивності сумарного (агрегованого) потоку пакетів, що надходять на мережний вузол, щоб вона не перевищувала пропускну здатність вихідного каналу зв'язку. Додати динамічний характер процесу обслуговування черг у рамках пропонованої моделі можна шляхом введення керуючої змінної x_{ij}, під якою малася на увазі частка i-го трафіку, що надходить для обслуговування в j-ю чергу. Відповідно до фізичного змісту хij мають місце наступні умови:

$$x_{ij} \in \{0,1\} \quad (i = \overline{1,M},\ j = \overline{1,N}) \tag{3.3}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad (i = \overline{1,M}), \tag{3.4}$$

$$\sum_{i=1}^{M} a_i x_{ij} \le b_j \quad (j = \overline{1,N}). \tag{3.5}$$

Виконання умови (3.4) гарантує відсутність втрат пакетів на розглянутому мережному вузлі.

Умова (3.5) уводиться для запобігання перевантаження пропускної здатності КМ, виділюваної для передачі пакетів тої або іншої черги мережного вузла в процесі управління. За аналогією з моделлю, розглянутої в роботах [13, 14], як шуканий вектор виберемо вектор:

$$\bar{x} = \begin{bmatrix} x_{ij} \\ -- \\ b_j \end{bmatrix} \quad (i = \overline{1,M};\ j = \overline{1,N}), \tag{3.6}$$

у ході розрахунку якого вдається забезпечити погодженість у рішенні завдань обслуговування черг і динамічного розподілу за ними пропускної здатності вихідного каналу зв'язку.

Умови запобігання перевантаження черг на вузлі MPLS-мережі

У ході виконання умов (3.5) через випадковий і нестаціонарний характер мережного трафіку на вузлі виникають черги й пов'язані з ними затримки пакетів. Для кожної черги визначається її поточна завантаженість і максимальна ємність.

Також умова (3.5) доповнюється умовами запобігання перевантаження черг по їхній довжині.

У загальному виді шукані умови будуть мати вигляд:

$$\overline{n}_j \le n_j^{\max} \quad (j = \overline{1,N}), \tag{3.7}$$

і завдання тепер зводиться лише до вибору (обґрунтуванню) аналітичного вираження для розрахунку середньої довжини черги в процесі обслуговування.

Формулювання оптимізаційного завдання по обслуговуванню черг на вузлі MPLS-мережі з підтримкою Traffic Engineering Queues

У зв'язку з тим, що, у загальному випадку, вибір керуючих змінних x_{ij} і b_j у рамках обмежень (3.1), (3.3), (3.4) і (3.7) можна зробити безліччю випадків, то доцільно завдання, пов'язану з розрахунком вектора (3.6), сформулювати у вигляді оптимізаційной. Основною вимогою до цільової функції є облік фізики процесів, що протікають на вузлі, обслуговування пакетів (3.1) – (3.7), а також відповідність одержуваних рішень принципам концепції Traffic Engineering Queues, що стосується забезпечення збалансованого завантаження буферних ресурсів. Із цією метою вище викладену модель важливо доповнити наступними умовами:

$$f(p_j, d_j) \cdot n_j \le \alpha, \quad (j = \overline{1,N}),$$

де α – верхній динамічно керована межа завантаженості черг на вузлі ТКС, $f(p_j)$ – деяка функція від характеристик j-го потоку.

Як правило, чим менше довжина пакета, тим «більш якісний» обслуговується потік, тому що невеликими пакетами передається трафік реального часу, що дуже чутливий до затримок. Потік з більше високим пріоритетом повинен традиційно [12] обслуговуватися краще, ніж трафік з низьким пріоритетом. Як функція характеристик потоку можна використовувати наступне вираження:

$$f(p_j, d_j) = \frac{p_j}{v \cdot d_j}, \quad (j = \overline{1,N}),$$

де v – деякий нормувальний коефіцієнт, що повинен згладжувати розходження в порядку значень пріоритету й довжини пакета в байтах [9].

Якщо в одній черзі обслуговуються потоки з різними значеннями довжини й (або) пріоритету пакета, то в цьому вираженні доцільно використовувати їхні усереднені значення.

У випадку, якщо кількість формованих черг перевершує число потоків трафіку, те завдання розподілу потоків по чергах стає тривіальної, через відсутність дефіциту черг. Тому розмірність шуканого вектора (3.6) можна значно знизити, тому що змінні x_{ij} ($i = 1,\ M\ ;\ j = 1,\ N$) розраховувати немає необхідності, а необхідне балансування черг можна буде забезпечувати за рахунок обчислення лише змінних b_j ($j = 1, N$).

3.2 Розробка структурної схеми

Структурна схема розробленої системи зображена на рисунку 3.1. У термінології комп'ютерних мереж, динамічний перерозподіл трафіку мережі – розподіл процесу виконання завдань між декількома серверами мережі з метою оптимізації використання ресурсів і скорочення часу обчислення.

Динамічний перерозподіл трафіку мережі може бути використаний для розширення можливостей ферми серверів, що складається більш ніж з одного сервера. Вона також може дозволити продовжувати роботу навіть в умовах, коли декілька серверів вийшли з ладу. Завдяки цьому забезпечується ріст відказостійкості, за рахунок установки пристроїв у кластері.

Динамічний перерозподіл трафіку мережі, що доводяться на декілька серверів, дозволяє уникнути такої ситуації, коли передані по мережі WEB пакети лавиною обрушуються на один сервер, у той час як інші простоюють без справи. Для розподілу навантаження між WEB-серверами звичайно використовується функція DNS, іменована циклічною вибіркою (round-robin feature), що передбачає можливість кругової передачі IP-адреси будь-якого WEB-сервера, що становить

сайт, будь-якому клієнтові; у підсумку всі WEB-сервери стають рівною мірою завантажені трафіком. Однак цей механізм недостатньо ефективний у тих середовищах, де можливості апаратних і програмних компонентів окремих WEB-серверів нерівнозначні. Однак з погляду процедури циклічної вибірки служби DNS між цими системами немає ніякої різниці; більше того, дана функція "не має подання" про доступність того або іншого WEB-сервера, оскільки не може визначити, працездатні комп'ютер або вийшов з ладу.

Не дуже давно постачальники налагодили випуск систем динамічного перерозподілу трафіку мережі – програмних продуктів, які вирівнюють навантаження, розподіляючи її по декількох серверах. Крім того, вони підвищують відказостійкість WEB-серверів: у випадку відмови однієї машини направляють пакети даних на інший сервер або сайт. Таким чином, час очікування скорочується, а число неопрацьованих запитів зводиться до мінімуму. Системи динамічного перерозподілу трафіку мережі можна використовувати як при наявності лише одного WEB-сайту, так і при роботі із цілим рядом вузлів. Одержавши подання про те, що таке системи динамічного перерозподілу трафіку мережі і як вони працюють, можна визначити найбільш важливі їхні характеристики, які варто враховувати при виборі засобу вирівнювання навантаження.

Система динамічного перерозподілу трафіку мережі WEB-серверів – це інструментальний засіб, призначений для переадресації клієнтських запитів на найменш завантажений або найбільш підходящий WEB-сервер із групи машин, на яких зберігаються дзеркальні копії інформаційного ресурсу. Клієнт не підозрює про те, що звертається до цілої групи серверів: всі вони представляються йому у вигляді якогось єдиного віртуального сервера. Припустимо що ми обслуговуємо один WEB-сайт і маємо при цьому два WEB-сервери. Представляючи наш сайт користувачам Internet, система динамічного перерозподілу трафіку мережі використовує ім'я віртуального комп'ютера, а також віртуальна IP-адреса (VIP-адресу – скажемо). Щоб зв'язати ім'я віртуальної

системи й відповідна VIP-адреса із двома нашими WEB-серверами, ми повинні опублікувати ім'я системи і її VIP-адресу на сервері DNS. Система динамічного перерозподілу трафіку мережі постійно контролює навантаження й ступінь готовності кожного з WEB-серверів. Коли на вузол заглядає відвідувач, його запит надходить не на один з WEB-серверів, а в систему динамічний перерозподіл трафіку мережі. Ця система й ухвалює рішення щодо того, на який сервер направити запит. При цьому вона керується такими критеріями, як завантаження кожного підопічного сервера, а також дотримує умов і правила, сформульовані адміністратором. Потім система динамічного перерозподілу трафіку мережі направляє запит клієнта відповідному серверу (як правило, воно ж направляє відповідь сервера клієнтові, але це залежить від конкретної реалізації).

Системи динамічного перерозподілу трафіку мережі можуть до того ж забезпечувати вирівнювання навантажень декількох WEB-сайтів. Застосування декількох сайтів дозволяє розміщати сервери-"дублери" (дзеркальні WEB-сервери) ближче до відвідувачів сайту й скорочувати затримки при обміні інформацією між сайтом і клієнтами. Крім того, при наявності декількох WEB-сайтів з'являється можливість рівномірно розподіляти навантаження між ними, а також забезпечувати високий ступінь їхньої готовності й відказостійкість у випадку порушень у роботі сайту (будь те через збої в системі енергопостачання або через втрату зв'язку з Internet в обчислювальному центрі). При наявності декількох сайтів, всі системи балансування навантажень на всіх сайтах мають одне загальне ім'я віртуальної системи, але різні VIP-адреси. Зв'язок всіх систем балансування з локальними WEB-серверами встановлюються так само, як і у випадку з одним сайтом. Поряд з моніторингом навантажень локальних серверів системи балансування обмінюються зі своїми "колегами" на інших сайтах інформацією про конфігурацію й завантаження; при цьому перевіряється й ступінь готовності сайту. У підсумку всі системи балансування мають у своєму розпорядженні загальну картину розподілу навантажень і готовності до роботи

різних вузлів. При наявності декількох сайтів системи балансування навантажень часто паралельно виконують ще одне завдання: беруть на себе роль серверів DNS, що обслуговують ім'я віртуальної системи. Одержавши через DNS сигнал від клієнта, що вказало дане ім'я, система балансування повертає клієнтові VIP-адреса сайту, найбільш підходящого з урахуванням поточного рівня навантаження, ступеня далекості від клієнта й інших параметрів. Потім клієнт автоматично одержує доступ до цього вузла.

Система динамічного перерозподілу трафіку мережі постійно відслідковує рівень навантаження й стан довірених їй WEB-серверів для того, щоб на підставі зібраної інформації в будь-який момент надати клієнтові доступ до того сервера, що зможе щонайкраще відповісти на його запит. При цьому використовується два методи контролю: зовнішній моніторинг і внутрішній моніторинг.

При проведенні зовнішнього моніторингу система динамічного перерозподілу трафіку мережі розраховує час відгуку сервера, для чого направляє на сервер запит і замірить час відповіді. Найпростіша техніка виконання зовнішнього моніторингу сервера припускає використання ping-тестів за протоколом керування повідомленнями Internet Control Message Protocol (ICMP). Ці тести дозволяють системі переконатися в готовності сервера до роботи й довідатися, скільки часу необхідно для передачі інформації із сервера на систему балансування й назад. Якщо система динамічного перерозподілу трафіку мережі не одержує відгуку від сервера після декількох послідовних запитів, уважається, що даний сервер недоступний. Як правило, адміністратори підключають WEB-сервери безпосередньо до системи динамічного перерозподілу трафіку мережі, тому якщо час, затрачуваний на передачу, занадто велике, система містить, що сервер працює з великим навантаженням.

Тут, однак, треба відзначити, що в ході ping-тестів сервера за протоколом ICMP діагностується тільки стек протоколів IP; описаний метод не дає подання про стан стека TCP, що використовується протоколом передачі гіпертексту HTTP. Щоб переконатися в правильності функціонування стека TCP сервера,

засіб динамічний перерозподіл трафіку мережі вживає спробу встановити з'єднання за протоколом TCP, для чого потрібно здійснити обмін, що складається із трьох етапів, що підтверджують повідомленнями. Робиться це так. Спочатку засіб динамічний перерозподіл трафіку мережі направляє серверу TCP-пакет, у якому значення біта SYN установлено рівним 1. Якщо після цього система балансування одержує від сервера TCP-пакет, у якому значення біта SYN дорівнює 1, а значення біта ACK теж установлено рівним 1, вона направляє серверу другий TCP-пакет зі значенням біта SYN рівним 0 і значенням біта ACK рівним 1. Якщо обмін підтверджувальними повідомленнями завершився успішно, виходить, TCP-стек сервера функціонує нормально. По завершенні такого обміну засіб динамічний перерозподіл трафіку мережі негайно розриває з'єднання із сервером, щоб виключити непродуктивне використання його ресурсів. Якість TCP-з'єднання із сервером оцінюється системою по такому показнику, як час, необхідне для виконання всіх трьох етапів обміну підтверджувальними повідомленнями.

Поряд з тестуванням стеків протоколів, кращі засоби динамічного перерозподілу трафіку мережі можуть забезпечувати моніторинг часу відгуку й готовності як самого WEB-сервера, так і встановлених на ньому додатків ще одним способом: на сервер направляється запит за протоколом HTTP на одержання інформаційних матеріалів або адреси URL. Час відгуку визначається системою динамічного перерозподілу трафіку мережі як час із моменту відправлення запиту на надання інформації до моменту одержання коду повернення.

Вибір сервера

З урахуванням інформації, отриманої в ході зовнішнього й внутрішнього моніторингу серверів, система динамічного перерозподілу трафіку мережі може виділити сервер, що здатний краще інших упоратися з обробкою клієнтського запиту. Якщо робочі характеристики апаратних і програмних компонентів всіх серверів пула рівнозначні, можна настроїти систему динамічний перерозподіл

трафіку мережі так, щоб вона виділяла сервер для обробки чергового запиту за принципом кругового списку, з огляду на при цьому стан сервера. Могутнішому серверу буде діставатися більше число запитів. Інакше кажучи, мова йде про той же принцип кругового списку, тільки з урахуванням вагових коефіцієнтів.

Сучасні системи динамічного перерозподілу трафіку мережі дозволяють адміністраторові визначати правила вибору сервера за своїм розсудом. Можна, приміром, включити в ці правила такі критерії, як коефіцієнт завантаження ЦП і пам'яті, число відкритих з'єднань TCP і кількість пакетів, що надходять на мережну інтерфейсну плату того або іншого сервера. Формула, по якій система балансування визначає рівень завантаженості серверів, може виглядати приблизно так: (10 * рівень використання ЦП) + (3 * рівень використання пам'яті) + (6 * число відкритих TCP-з'єднань) + (3 * число переданих пакетів) = завантаження сервера. При одержанні запиту від клієнта система динамічного перерозподілу трафіку мережі розраховує по цій формулі навантаження кожного сервера й направляє запит серверу з найменшим навантаженням.

У деяких випадках після того, як засіб балансування призначає сервер для виконання запиту клієнта й клієнт установлює первісне з'єднання, для забезпечення виконання прикладної програми необхідно, щоб система балансування постійно підтримувала трафік між клієнтом і сервером. Таке з'єднання називається постійним. Припустимо, відвідувач електронної книгарні вибирає покупку й відкладає у віртуальний кошик три книги. Якщо сервер, що обробляє запит клієнта, кешує інформацію про вміст кошика локально, система динамічного перерозподілу трафіку мережі вже не може виділити для наступних обмінів із клієнтом інший сервер, навіть якщо того вимагає співвідношення, що змінилося, навантажень серверів вузла. Адже при такій переадресації дані про вміст віртуального кошика клієнта будуть загублені, тому що новий сервер попросту не має цих відомостей. Отже, система балансування повинна фіксувати інформацію про те, з яким сервером зв'язується кожний клієнт, і зберігати її в пам'яті протягом певного часу (час зберігання вказує адміністратор з

урахуванням таких факторів, як звичайне поводження клієнтів і особливості прикладної програми). При активізації функції постійних (стійких) з'єднань вона буде увесь час скасовувати інші правила динамічного перерозподілу трафіку мережі.

При організації стійкого з'єднання основне завдання системи полягає в тому, щоб ідентифікувати клієнта й зв'язати відповідний ідентифікатор із сервером-одержувачем запиту. Як правило, системи динамічного перерозподілу трафіку мережі використовують як ідентифікатор клієнта, застосовувану їм IP-адресу відправника. Але вся справа в тому, що адреса відправника не обов'язково збігається з реальною IP-адресою клієнта. Багато компаній і провайдери, намагаючись удержати WEB-трафік під контролем і сховати від сторонніх очей IP-адреси своїх користувачів, установлюють сервери-посередники. На щастя, цю проблему можна вирішити, якщо скористатися системою динамічного перерозподілу трафіку мережі, оснащеної засобами ідентифікації IP-адрес відправників і номерів портів TCP. Подібні системи здатні пізнавати клієнтів навіть у тому випадку, коли останні виходять в Internet через той самий proxy-сервер. Така ідентифікація можлива тому, що кожне TCP-з'єднання має унікальна IP-адресу відправника й номер порту TCP. Ще один спосіб ідентифікації клієнта, що проводить захищений сеанс зв'язку за протоколом HTTP, полягає в тому, щоб зафіксувати ідентифікаційний номер сеансу зв'язку користувача за протоколом Secure Sockets Layer (SSL). Протокол SSL призначає кожному встановленому сеансу зв'язку спеціальний ідентифікатор, а прикладні програми для віртуальних магазинів часто користуються цим протоколом. Найсучасніший засіб підтримки стійких з'єднань – це розповсюджувані по мережі WEB cookie-файли. Нагадаю, що ці файли містять як відомості про клієнта, так і інші дані (наприклад, про те, з яким сервером клієнт зв'язувався востаннє). Аналіз умісту cookie-файлів допомагає системі динамічного перерозподілу трафіку мережі ідентифікувати клієнтів і підбирати для них найбільш підходящий сервер. У число постачальників систем динамічний перерозподіл трафіку мережі, оснащених

засобами роботи з cookie-файлами, входять такі компанії, як Alteon WEBSystems, ArrowPoint Communications, F5 Networks і Resonate.

Нарешті, існує ще один спосіб вибору сервера – так зване безпосереднє зв'язування (immediate binding). Відповідно до цього методу системи динамічного перерозподілу трафіку мережі підбирають сервер для клієнта й направляють запит на нього в той самий момент, коли система одержує від клієнта пакет TCP SYN. При цьому система балансування вибирає сервер, керуючись заданими правилами розподілу навантаження серверів, а також IP-адресою, що втримується в отриманому від клієнта пакеті TCP SYN. Цей метод забезпечує висока швидкодія, але треба сказати, що при його застосуванні система балансування просто не встигає проаналізувати іншу інформацію: зокрема, ідентифікатор сеансу зв'язку за протоколом SSL, уміст cookie-файлу, адреса URL і дані прикладної програми. Щоб одержати більше докладні відомості про клієнта й, відповідно, точніше вибрати для нього сервер, системі динамічного перерозподілу трафіку мережі потрібен час для аналізу інформації рівня додатка. При виборі сервера по методу відкладеного зв'язування система динамічного перерозподілу трафіку мережі ухвалює рішення щодо призначенні сервера лише по завершенні трьохетапного обміну підтверджувальними повідомленнями й після установки з'єднання між нею й клієнтом. Система може враховувати вміст публікуемих матеріалів, якщо досліджує інформацію рівня прикладної програми до вибору сервера для клієнта.

Переадресація трафіку

Системи динамічного перерозподілу трафіку мережі можуть перенаправляти трафік клієнтів на вибраний сервер декількома способами: по методу трансляції адрес із керуванням доступом до середовища передачі (media access control (MAC) address translation, MAT), по методу трансляції мережних адрес (Network Address Translation, NAT), або – при використанні відкладеного зв'язування – за допомогою механізму шлюзу TCP (TCP gateway). Розглянемо, як

реалізується кожний із цих методів перенапрямку трафіка засобами вирівнювання навантаження.

MAT. Цей метод може бути реалізований системою динамічного перерозподілу трафіку мережі при тій умові, що кожний WEB-сервер поряд зі своїм фізичним IP-адресою використовує в якості інтерфейсного адресу зворотного зв'язка (loopback interface address) VIP-адреси системи балансування. Одержавши від клієнта пакет і призначивши йому відповідний сервер, система балансування заміняє в цьому пакеті MAC-адреса одержувача MAC-адресою відповідного сервера, після чого направляє пакет на виділений сервер. У пакеті втримується IP-адреса клієнта, так що для прямої відповіді клієнтові сервер використовує в якості IP-адреси одержувача первісна IP-адреса клієнта. Однак у якості IP-адреси відправника сервер указує VIP-адресу системи динамічного перерозподілу трафіку мережі, як якби трафік надходив клієнтові саме від її. Таким чином, що випливає пакет від клієнта направляється не тому, що відповів, серверу, а системі динамічного перерозподілу трафіку мережі.

NAT. При використанні цього методу система балансування направляє отриманий від клієнта пакет призначеному серверу лише після того, як виконає над пакетом кілька операцій: по-перше, вона заміщає в пакеті адресу одержувача (тобто власна VIP-адреси) IP-адресою призначеного сервера, а по-друге, – міняє IP-адресу відправника на свій VIP-адресу. Даний метод дозволяє приховувати від клієнтів IP-адреси WEB-серверів, так що останні можуть використовувати будь-які IP-адреси, у тому числі й частки. При цьому WEB-сервери не обов'язково повинні бути безпосередньо з'єднані із системою балансування (інакше кажучи, входити в той самий сегмент ЛОМ); досить, щоб вони могли встановлювати з'єднання по протоколах статичної або мережної маршрутизації.

Шлюз TCP. При установці безпосереднього (негайного) зв'язування системи динамічного перерозподілу трафіку мережі можуть направляти трафік по методах MAT або NAT на рівнях 2 або 3. Але якщо мова йде про відкладене зв'язування, системи балансування повинні управляти трафіком на рівні TCP і на

більше високих рівнях. Відкладене зв'язування припускає, що система динамічного перерозподілу трафіку мережі й клієнт установлюють з'єднання за протоколом TCP так, щоб система могла одержати дані додатки ще до призначення сервера. Потім засіб балансування встановлює TCP-з'єднання із призначеним сервером і передає йому клієнтський запит. Далі система балансування передає клієнтові відповідь сервера, для чого знов-таки використовується TCP-з'єднання "система балансування – клієнт". Описана функція й називається шлюзом TCP. Фахівці компанії Resonate реалізують її в системі динамічного перерозподілу трафіку мережі за допомогою спеціального агента (цей агент установлюється на сервері, що забезпечує пряме TCP-з'єднання між клієнтом і сервером, що виступає в ролі засобу балансування). По термінології, запропонованою компанією-постачальником, дана реалізація йменується системою із транзитом TCP-з'єднань (TCP connection hop).

Вибір сайту й керування трафіком на глобальному рівні

У тих випадках, коли інформаційні ресурси розміщаються на декількох дзеркальних вузлах, системи динамічного перерозподілу трафіку мережі (іменовані також глобальними системами динамічного перерозподілу трафіку мережі) визначають підходящий для клієнта вузол за допомогою вже описаних механізмів вибору сервера. Крім того, як критерій вибору сайту глобальна система балансування може використовувати такий показник, як відстань між сайтом і клієнтом (виражене в кількості транзитних ділянок і в тривалості мережної затримки). При визначенні найбільш підходящого сайту система балансування часто направляє трафік клієнта на відповідний вузол за допомогою інтелектуальної функції DNS.

Крім методу динамічного призначення клієнтові того або іншого вузла системи динамічного перерозподілу трафіку мережі можуть використовувати для зв'язування конкретних клієнтів з конкретними сайтами метод статичного призначення (static mapping method).

Архітектура служби динамічного перерозподілу трафіку мережі мережі

Для досягнення максимальної пропускної здатності й відказостійкості служба ДПТМ використовує повністю розподілену програмну архітектуру. На всіх вузлах кластера паралельно виконуються однакові драйвери служби ДПТМ. Ці драйвери поєднують всі вузли в єдину мережу для обробки вхідного потоку даних, що надходять на основну IP-адресу кластера (і на додаткові IP-адреси багатомережних вузлів). Для кожного окремого вузла драйвер виконує функції фільтра між драйвером мережного адаптера й стеком протоколів TCP/IP, дозволяючи розподіляти потік даних, одержуваних вузлом. Таким чином, що надходять запити розділяються й розподіляються між вузлами кластера.

Служба ДПТМ функціонує як мережний драйвер, розташований у мережній моделі нижче високорівневих протоколів додатків, таких як HTTP і FTP.

Така архітектура дозволяє домогтися максимальної пропускної здатності за рахунок використання широкомовної підмережі для доставки даних, що надходять, на всі вузли кластера, що дозволяє обійтися без маршрутизації вхідних пакетів. Оскільки фільтрація непотрібних пакетів працює швидше, ніж маршрутизація (при якій необхідно одержати, перевірити, перезаписати й повторно відправити кожний пакет), при використанні служби ДПТМ досягається більше висока пропускна здатність мережі в порівнянні з рішеннями на основі диспетчеризації. При росту швидкості роботи сервера й мережі пропорційно росте й продуктивність; у такий спосіб усувається залежність від продуктивності апаратних рішень для розподілу навантаження на основі маршрутизації. Наприклад, у гігабітних мережах служба ДПТМ демонструє пропускну здатність до 250 Мб/с.

Іншою ключовою перевагою повністю розподіленої архітектури служби ДПТМ є чудові показники відказостійкості (N-1) для кластера з N вузлами. Навпроти, у рішеннях на основі диспетчеризації обов'язково є центральний елемент, що є «вузьким місцем» системи, для усунення якого необхідно використовувати резервний диспетчер, забезпечуючи лише односпрямоване

переміщення навантаження при збої. Такий захист від збою менш ефективний у порівнянні з повністю розподіленою архітектурою.

В архітектурі служби ДПТМ для одночасної доставки даних, що надходять, на кожний вузол кластера використовується концентратор і/або комутатор підмережі. Проте, такий підхід веде до збільшення навантаження на комутатори й вимагає додаткових ресурсів пропускної здатності портів. Звичайно це не впливає на більшість широко використовуваних додатків (наприклад, WEB-служби й мультимедіа-мовлення), оскільки вхідні дані становлять дуже невелику частку загального потоку даних у мережі. Проте, якщо пропускна здатність лінії зв'язку до комутатора з боку клієнта значно вище пропускної здатності каналу з боку сервера, що входять дані можуть становити досить значну частину загального потоку даних. Та ж проблема виникає й при підключенні декількох кластерів до одного комутатора, коли для окремих кластерів не настроєні віртуальні локальні мережі LAN.

Повністю конвеєрний механізм служби ДПТМ при надходженні пакетів одночасно передає їх у стек протоколів TCP/IP і одержує пакети від драйвера мережного адаптера. Це знижує загальний час обробки потоку даних і затримку, оскільки стек TCP/IP може обробляти пакет одночасно з одержанням драйвером NDIS наступного пакета. Крім того, потрібно менше ресурсів для координації операцій стека TCP/IP і драйвера, а також, у більшості випадків, у пам'яті не створюються додаткові копії пакетів даних. При відправленні пакетів служба ДПТМ також забезпечує підвищену пропускну здатність, малий час затримки й відсутність накладних витрат продуктивності за рахунок збільшення числа пакетів TCP/IP, які можуть бути відправлені за один виклик NDIS. Для досягнення настільки високої продуктивності служба ДПТМ використовує пул буферів і дескрипторів пакетів, використовуваний для конвеєрних операцій зі стеком TCP/IP і драйвером NDIS.

На структурній схемі, яка зображена на рисунку 3.1, показано, що система, структурно, складається з наступних блоків:

– Блок динамічного перерозподілу трафіку мережі.

– Блок формування звітів.

– Блок формування сигналів керування перерозподілу трафіку мережі.

– Блок збирання статистичного навантаження на вузли мережі.

– Блок аналізу статистичного навантаження.

Для рішення завдання вибору числа й розміщення вузлів мережі, представленої у вигляді інформаційно-обчислювального комплексу, запропоновано використовувати евристичні алгоритми, засновані на методах локальної оптимізації й теорії масового обслуговування. Альтернативний підхід до рішення завдання дискретного математичного програмування, заснований на теорії графів, неприйнятний через високу обчислювальну складність при великій кількості вузлів у мережі. Наприклад, у мережі з 10 вузлів існує 2^{45} варіантів розташування ліній зв'язку, включаючи безліч тривіальних випадків. Якщо припустити, що аналіз кожного варіанта становить 1 секунду, то на дослідження буде потрібно більш ніж $9*10^8$ років.

На змістовному рівні завдання побудови оптимальної мережі формулюється в такий спосіб. Виходячи із заданих значень інтенсивності запитів абонентів з урахуванням вводимих допущень і обмежень визначити оптимальні за критерієм мінімуму наведених витрат структурні параметри мережі: число й розміщення програмно-апаратних модулів вузлів мережі у пунктах мережі, співвіднести групи абонентів з обслуговуючими їх вузлом мережі, ємністю вузла мережі і каналів зв'язку. При цьому повинні дотримуватися обмеження на якість обслуговування: середній час обробки повідомлення й імовірність своєчасного обслуговування не повинні перевищувати граничних значень.

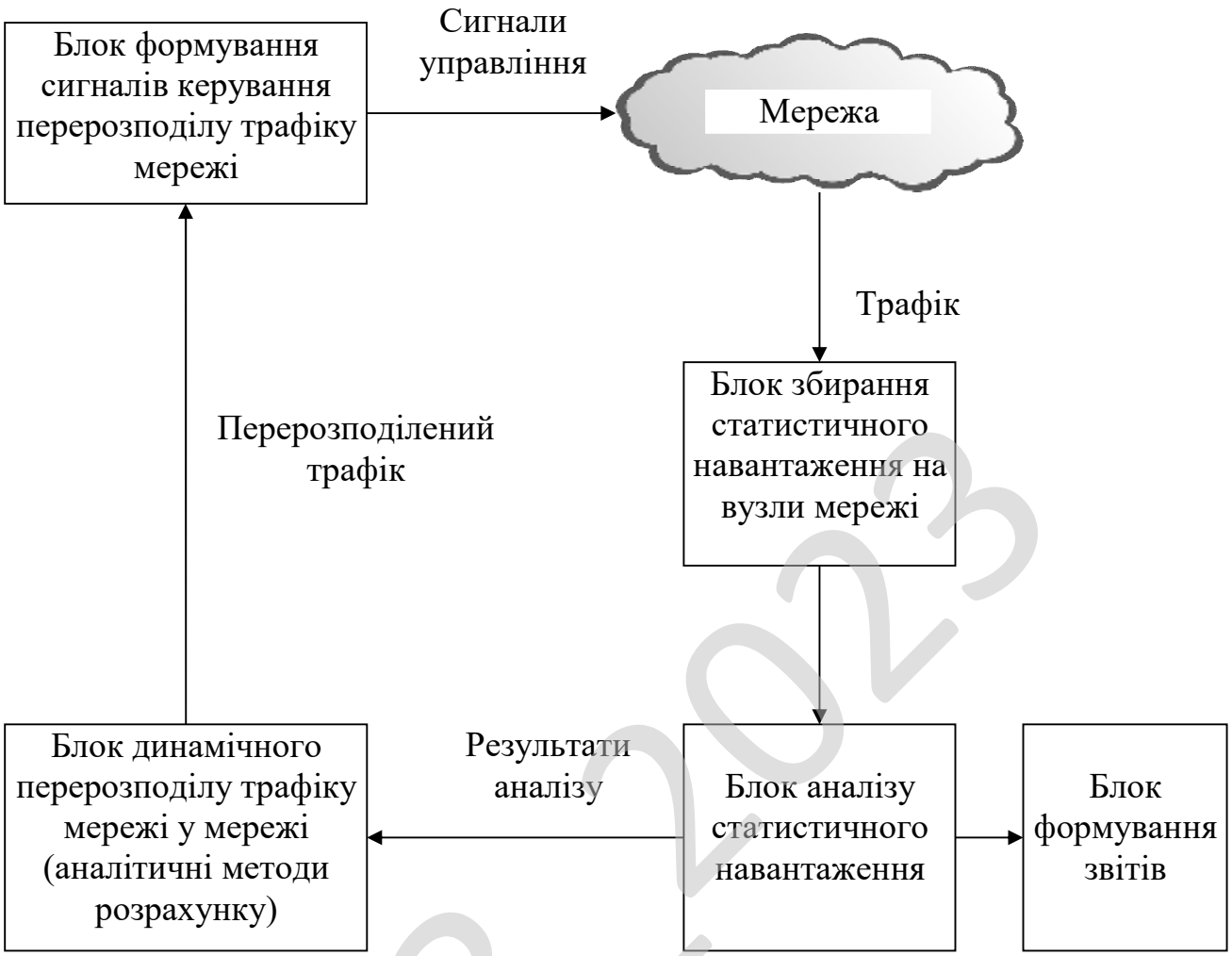

Рисунок 3.1 – Структурна схема системи

Як критерій оптимізації обрані наведені витрати на канал зв'язку й вузол мережі.

Як модель, що інтерпретує інформаційний потік у мережі з УЦ, розглядається дві мережі масового обслуговування (СеМО):

– багатофазна СеМО з відмовами й повторними викликами;

– багатофазна СеМО комбінованою комутацією.

Завдання пошуку оптимальної структури мережі успішно вирішуються з використанням евристичних алгоритмів, заснованих на методах локальної оптимізації зі спрямованим перебором варіантів структури мережі.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2. Створена функціональна модель мережі дозволяє вирішити наступні завдання:

– сформувати наочне візуальне подання взаємодії основних процесів, що відбуваються у мережі;

– зробити чітке визначення ролі системи поліпшення функціонування в загальній системі;

– точно визначити вхідні, вихідні й керуючі впливи на підсистему динамічного перерозподіл трафіку мережі, що надалі дозволяє провести аналітичне й імітаційне моделювання інформаційного потоку;

– сценарій динамічного перерозподілу трафіку мережі, розроблений у ході виконання магістерської роботи, є основою алгоритму імітаційного моделювання.

З рисунку видно, що розроблена система складається з наступних частин:

– Блок формування матриці транзитних вузлів.

– Блок формування сигналів для елементів мережі.

– Згенеровані звіти по поточному завантаженню елементів мережі.

– Блок розрахунку вектору значень інтегрального критерію оптимізації.

– Блок ініціалізації граничних значень параметрів моделювання.

– Блок формування матриці динамічний перерозподіл трафіку мережі.

Інтернет-технології прийняли широке поширення, і використовуються як основа побудови корпоративних і критично важливих додатків, таких як WEB-вузли, вузли потокового мультимедіа-віщання й сервери віртуальних приватних мереж. Служба динамічного перерозподілу трафіку мережі (ДПТМ) є оптимальним і ефективним рішенням, що забезпечує масштабованість і високу відказостійкість таких додатків як в Інтернеті, так і в інтрамережах. Служба ДПТМ дозволяє системним адміністраторам створювати кластери, що включають до 32 вузлів, між якими будуть розподілятися вступні від клієнтів запити. При цьому з погляду клієнтів кластер нічим не відрізняється від звичайного сервера; серверні додатки також не вимагають адаптації для роботі в кластері.

Рисунок 3.2 – Функціональна схема системи

Служба ДПТМ постачена всіма необхідним адміністраторам способами керування, у тому числі можливістю (після уведення пароля) віддалено управляти кластером з будь-якого комп'ютера в мережі. Крім того, адміністратори мають можливість набудовувати кластер під спеціальні завдання, управляючи потоком даних на рівні портів. Вузли додаються й виключаються із кластера без припинення обслуговування. Крім того, програмне забезпечення на вузлах кластера можна обновляти без припинення обробки клієнтських запитів.

Служба ДПТМ використовує для розподілу навантаження між вузлами повністю розподілений алгоритм. На відміну від рішень на основі диспетчеризації така архітектура забезпечує високу продуктивність і низькі накладні витрати ресурсів на розподіл потоку запитів від клієнтів. Крім того, для

цієї архітектури характерна висока відказостійкість (N-1) при числі вузлів N. Всі ці характеристики досягаються без необхідності використовувати спеціальні апаратні або програмні рішення.

Тести продуктивності демонструють, що використання програмної служби ДПТМ дає низькі накладні витрати на обробку потоку даних і чудові можливості масштабування продуктивності, обмежені тільки пропускною здатністю підмережі.

Служба ДПТМ демонструє пропускну здатність більше 200 Мбіт/с у реальних рішеннях по обслуговуванню електронної торгівлі з більш ніж 800 млн. запитів протягом дня.

Служба ДПТМ періодично розсилає ритмічні повідомлення, призначені для інформування кожного члена кластера про наявність інших вузлів. Збій будь-якого вузла виявляється протягом п'яти секунд, а відновлення обслуговування клієнтів виконується протягом десяти секунд.

Як при відключенні працюючого вузла, так і при додаванні нового вузла в кластер навантаження автоматично й прозоро перерозподіляється між членами кластера.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. Після початку роботи розробленого ПЗ ми потрапляємо до інтерфейсу ПЗ (головне вікно програми), далі можна провести моніторинг вузлів мережі та топологічне формування мережі з проведенням поточної оптимізації зі спрямованим перебором варіантів структури мережі та створенням вузлів.

Провести аналіз трафіку вузлів мережі та у наступному переглянути результати аналізу з формуванням звіту. Та провести динамічний перерозподіл трафіку вузлів мережі з формуванням сигналів керування навантаженням, проведенням переадресації трафіку, розподілу потоку даних кластера.

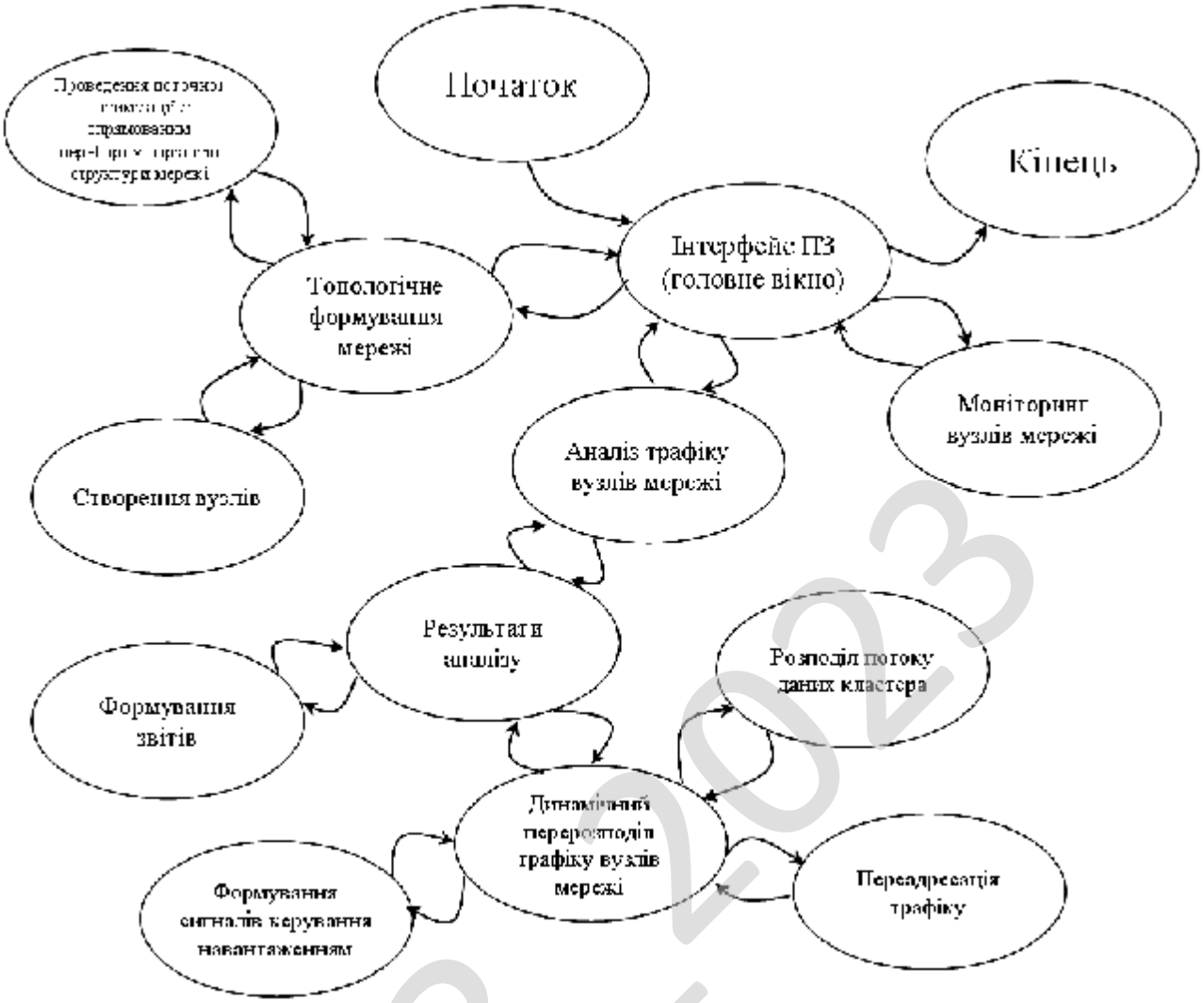

Рисунок 3.3 – Діаграма взаємодії процесів

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

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ 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 відбувається:

– Виведення головного вікна ПЗ.

– Запит сканування мережі?

– Підпрограма перерозподілу трафіку вузлів мережі.

– Підпрограма оптимізації шляхів передачі даних.

– Виведення динамічного перерозподілу трафіку вузлів мережі.

– Збір статистики роботи вузлів мережі?

– Сканування стану наявних серверів.

– Збереження результатів у БД результатів.

– Виведення статистичних результатів.

– Запит аналізу статистики навантаження?

– Аналіз статистики навантаження та формування звіту.

– Виведення результатів аналізу статистики навантаження.

– Балансування навантаження у мережі.

– Розподіл потоку даних та переадресація трафіку.

– Системне повідомлення WM_CLOSE?

Виконання підпрограми перерозподілу трафіку вузлів мережі зображена на рисунку 4.2.

З рисунку видно, що після запуску програми відбувається:

– Формування початкових значень вузлів мережі.

– Запит пошуку та змін у мережі?

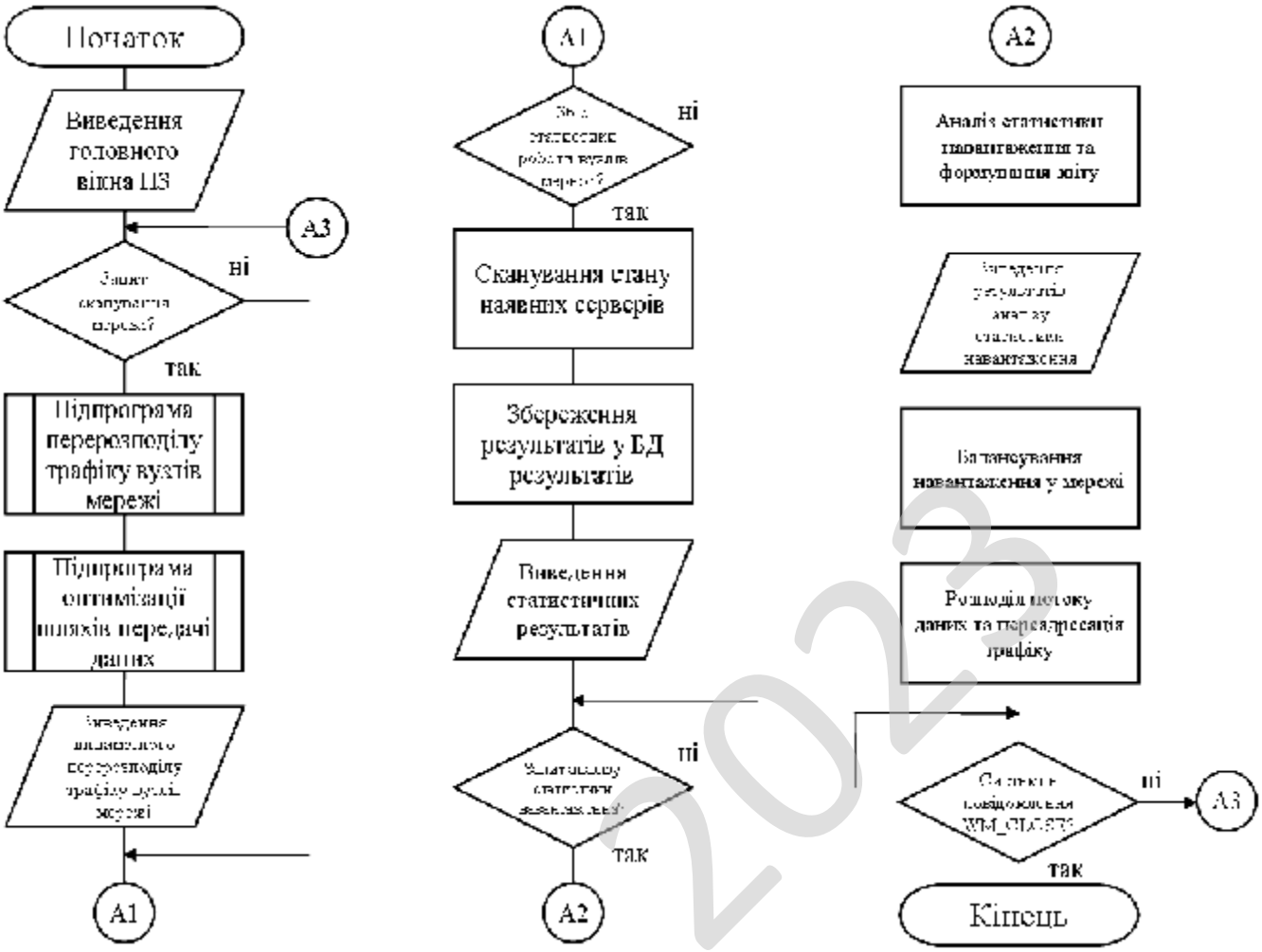

Рисунок 4.1 – Блок-схема основної програми

– Оновлення таблиці відстаней.

– Пошук найкоротших шляхів.

– Сканування та побудова таблиці відстаней.

– Переформатування таблиці топології.

– Сканування рядків в таблиці топології.

– Запит кількість рядків > 1?

– Формування множини вузлів, найближчих до роутера.

– Оновлення таблиці топології вузлів мережі.

– Знайдені повтори у таблиці топологій?

– Оновлення та застосування таблиці відстаней.

– Формування записів шляхів вузлів мережі.

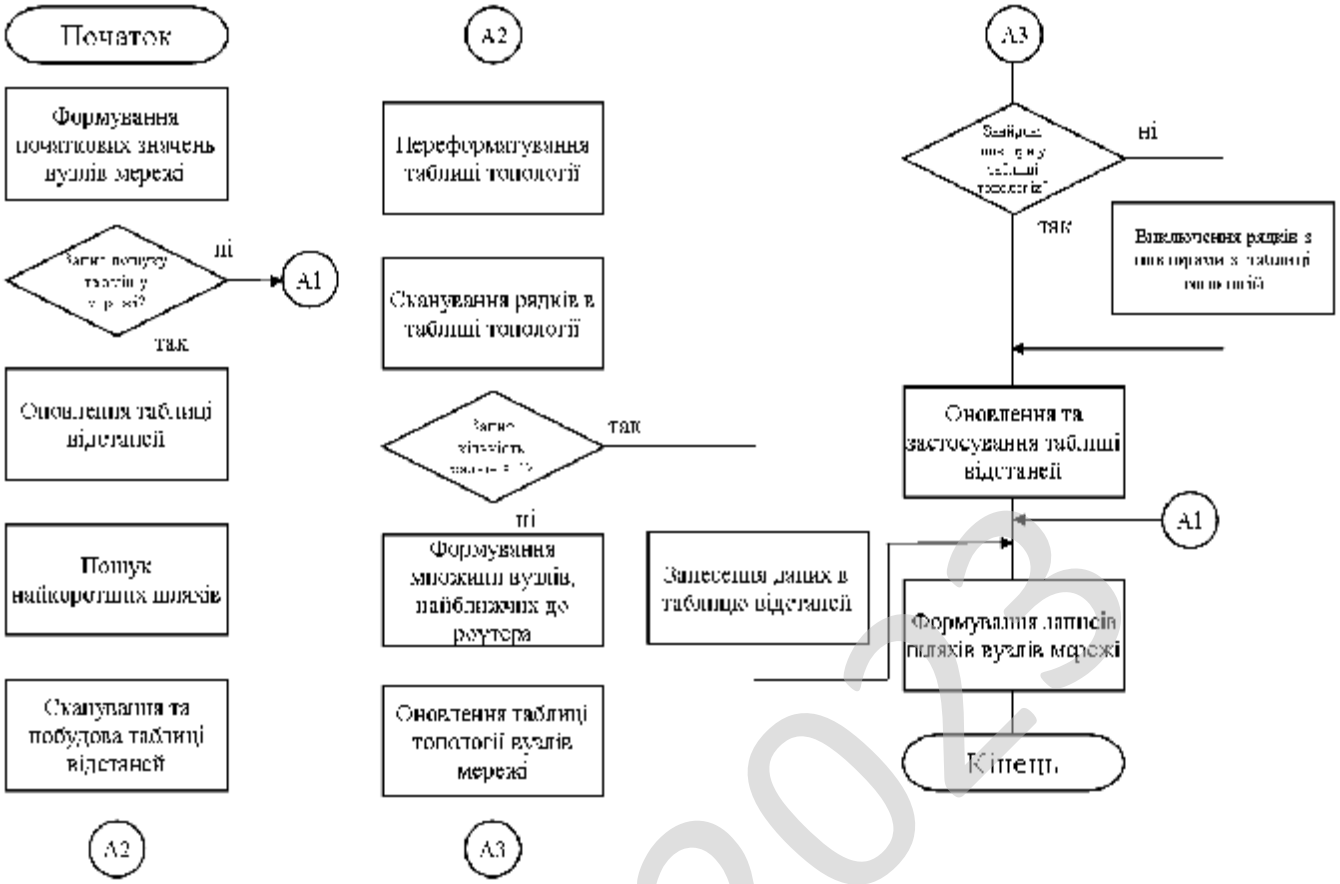

Рисунок 4.2 – Блок-схема підпрограми перерозподілу трафіку вузлів мережі зображена

Виконання підпрограми оптимізації шляхів передачі даних зображена на рисунку 4.3.

З рисунку видно, що після запуску програми відбувається:

– Обрання критерію оптимізації та оновлення.

– Перевірка та виключення маршрутів, що не задовольняють поставленим вимогам.

– Проведення моніторингу вхідного трафіку та розподілення інформаційного потоку.

– Запит швидкість передачі даних оптимальна?

– Формування даних динамічного перерозподілу трафіку вузлів мережі.

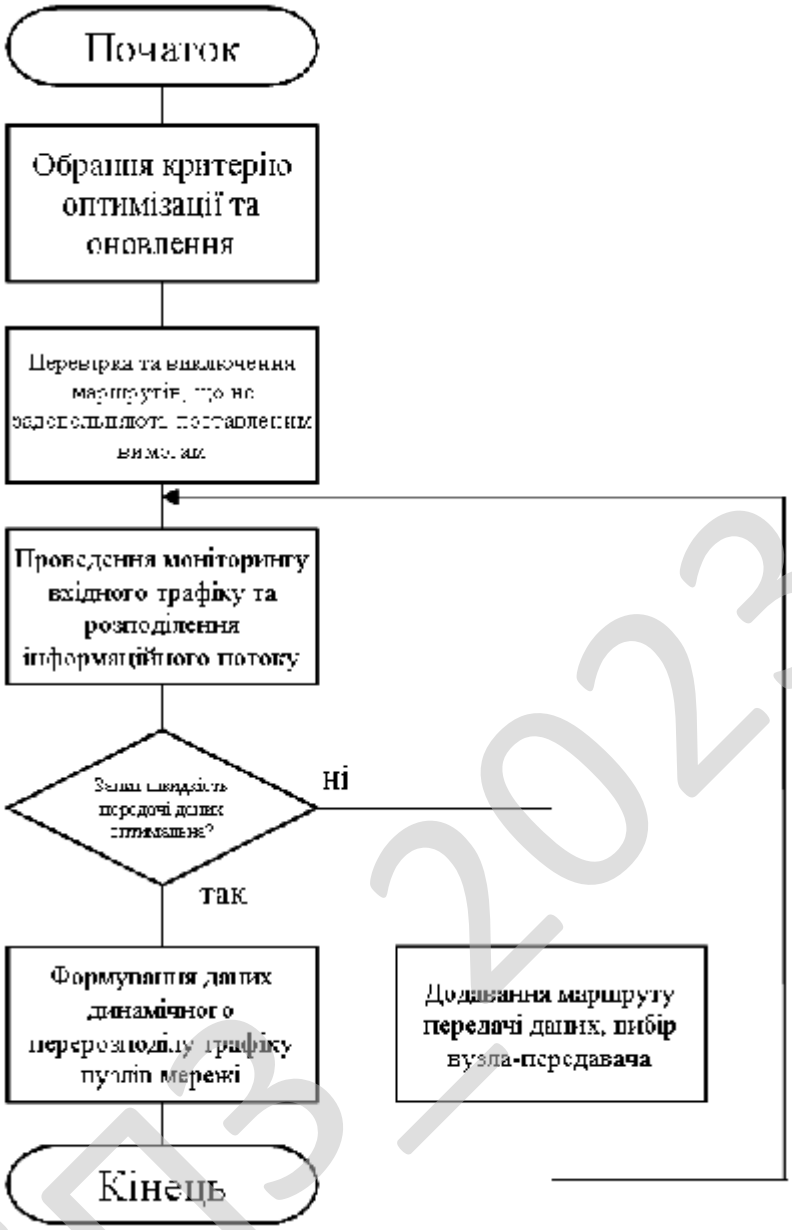

Рисунок 4.3 – Блок-схема підпрограми оптимізації шляхів передачі даних

Опис алгоритмів функціонування системи

Наведемо частину коду, який реалізує вище перераховані дії. Визначимо типи даних та константи, які потрібні для роботи програмного продукту.

```
const
  WellKnownPorts: array[1..32] of TWellKnownPort
  = (
    ( Prt: 0; Srv:    ' RESRVED'   ),
{Зарезервовано}
    ( Prt: 7; Srv:    ' ECHO    '  ),
```

```
{Пінгування    }
    ( Prt: 9; Srv:   ' DISCARD'  ),
    ( Prt: 13; Srv:  ' DAYTIME'  ),
    ( Prt: 17; Srv:  ' QOTD   '  ),
{Показчик на день}
    ( Prt: 19; Srv:  ' CHARGEN'  ),
{Генератор символів}
    ( Prt: 20; Srv:  ' FTPDATA'  ),
{ Протокол File Transfer Protocol – дані}
    ( Prt: 21; Srv:  ' FTPCTRL'  ),
{ Протокол File Transfer Protocol – управління}
    ( Prt: 22; Srv:  ' SSH    '  ),
    ( Prt: 23; Srv:  ' TELNET '  ),
    ( Prt: 25; Srv:  ' SMTP   '  ),
{ Протокол Simple Mail Transfer Protocol}
    ( Prt: 37; Srv:  ' TIME   '  ),
{ Протокол Time Protocol }
    ( Prt: 43; Srv:  ' WHOIS  '  ),
{ WHO IS сервіс  }
    ( Prt: 53; Srv:  ' DNS    '  ),
{ Domain Name Service }
    ( Prt: 67; Srv:  ' BOOTPS '  ),
{ BOOTP сервер }
    ( Prt: 68; Srv:  ' BOOTPC '  ),
{ BOOTP клієнт }
    ( Prt: 69; Srv:  ' TFTP   '  ),
{ Стандартний  FTP  }
    ( Prt: 70; Srv:  ' GOPHER '  ),
{ Протокол Gopher      }
    ( Prt: 79; Srv:  ' FINGER '  ),
{ Протокол Finger      }
    ( Prt: 80; Srv:  ' HTTP   '  ),
{ Протокол HTTP        }
    ( Prt: 88; Srv:  ' KERBROS'  ),
{ Протокол Kerberos    }
    ( Prt: 109; Srv: ' POP2   '  ),
{ Протокол Post Office Protocol Version 2 }
    ( Prt: 110; Srv: ' POP3   '  ),
{ Протокол Post Office Protocol Version 3 }
    ( Prt: 111; Srv: ' SUN_RPC'  ),
{ SUN віддалений виклик функцій }
    ( Prt: 119; Srv: ' NNTP   '  ),
```

```
{ Протокол Network News Transfer Protocol }
    ( Prt: 123; Srv: ' NTP    '  ),
{ Протокол Network Time protocol          }
    ( Prt: 135; Srv: ' DCOMRPC'  ),
{ Локальний сервіce              }
    ( Prt: 137; Srv: ' NBNAME '  ),
{ NETBIOS сервіс імен          }
    ( Prt: 138; Srv: ' NBDGRAM'  ),
{ NETBIOS сервіс датаграм     }
    ( Prt: 139; Srv: ' NBSESS '  ),
{ NETBIOS сервіс сесій        }
    ( Prt: 143; Srv: ' IMAP   '  ),
{ Протокол Internet Message Access Protocol }
    ( Prt: 161; Srv: ' SNMP   '  ),
{ Протокол Simple Netw. Management Protocol }
    ( Prt: 169; Srv: ' SEND   '  )
    );
  const
  ICMP_ERROR_BASE = 11000;
  IcmpErr : array[1..22] of string =
  (
   ' IP_BUFFER_TOO_SMALL' ,' IP_DEST_NET_UNREACHABLE' , '
IP_DEST_HOST_UNREACHABLE' ,
   ' IP_PROTOCOL_UNREACHABLE' , ' IP_DEST_PORT_UNREACHABLE' , '
IP_NO_RESOURCES' ,
   ' IP_BAD_OPTION' ,' IP_HARDWARE_ERROR' , ' IP_PACKET_TOO_BIG' , '
IP_REQUEST_TIMED_OUT' ,
   ' IP_BAD_REQUEST' ,' IP_BAD_ROUTE' , ' IP_TTL_EXPIRED_TRANSIT' ,
   ' IP_TTL_EXPIRED_REASSEM' ,' IP_PARAMETER_PROBLEM' , ' IP_SOURCE_QUENCH' ,
   ' IP_OPTION_TOO_BIG' , ' IP_BAD_DESTINATION' ,' IP_ADDRESS_DELETED' ,
   ' IP_SPEC_MTU_CHANGE' , ' IP_MTU_CHANGE' , ' IP_UNLOAD'
  );
  ARPEntryType  : array[1..4] of string = ( ' Інший' , ' Несправний' ,
    ' Динамічний' , ' Статичний'
    );
  TCPConnState  :
    array[1..12] of string =
 ( ' CLOSED' , ' READ' , ' SYN_SENT' ,
    ' SYN_RCVD' , ' ESTABLISHED' , ' FIN_WAIT1' ,
    ' FIN_WAIT2' , ' WAIT' , ' CLOSING' ,
    ' LAST_ACK' , ' WAIT' , ' DELETE_TCB'    );
  TCPToAlgo     : array[1..4] of string =
```

```
    ( ' Const.Timeout' , ' MIL-STD-1778' ,
    ' Van Jacobson' , ' Other'  );
  IPForwTypes   : array[1..4] of string =
    ( ' other' , ' invalid' , ' local' , ' remote'  );
  IPForwProtos  : array[1..18] of string =
    ( ' OTHER' , ' LOCAL' , ' NETMGMT' , ' ICMP' , ' EGP' ,
    ' GGP' , ' HELO' , ' RIP' , ' IS_IS' , ' ES_IS' ,
    ' CISCO' , ' BBN' , ' OSPF' , ' BGP' , ' BOOTP' ,
    ' AUTO_STAT' , ' STATIC' , ' NOT_DOD'  );
type
// для IpHlpNetworkParams
  TNetworkParams = record
    HostName: string ;
    DomainName: string ;
    CurrentDnsServer: string ;
    DnsServerTot: integer ;
    DnsServerNames: array [0..9] of string ;
    NodeType: UINT;
    ScopeID: string ;
    EnableRouting: UINT;
    EnableProxy: UINT;
    EnableDNS: UINT;
  end;
  TIfRows = array of TMibIfRow ;
// динамічний масив колонок
```

Наведемо код классу TNetworkWorkgroup. Це клас, який створює список всіх комп' ютерів в робочій групі. Цей клас є нащадком класу TStrings і повністю сумісний з іншим нащадками цього класу.

Об'єкти цього класу записуються у властивості об'єктів класу TNetworkNeiborhood. Клас TNetworkNeiborhood містить об' єкти і властивості робочих груп.

```
TNetworkWorkgroup = class (TStringObjectList);
{TNetworkNeiborhood – клас, який створює список всіх робочих груп в мережі}
  TNetworkNeighborhood = class (TStringObjectList)
  private
    function CreatePIDL(Size: Integer): PItemIDList;
    procedure DisposePIDL(ID: PItemIDList);
    function NextPIDL(IDList: PItemIDList): PItemIDList;
    function GetPIDLSize(IDList: PItemIDList): Integer;
```

```
    function CopyPIDL(IDList: PItemIDList): PItemIDList;
    procedure StripLastID(IDList: PItemIDList);
    function GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
    class function GetDisplayName(ShellFolder: IShellFolder; PIDL:
PItemIDList): TString;
    function OriginFolder:  IShellFolder;
    function OriginFolderNT: IShellFolder;
    class function EnumObjects(ShellFolder: IShellFolder): IEnumIDList;
    class procedure ParseFolder(Folder: IShellFolder; Items:
TStringObjectList; StorePIDLs: Boolean = False);
    class procedure ParseFolderEx(Folder: IShellFolder; Items: TStrings);
    function FreeRefObj(Index: Integer; var Obj: TStringObject): Integer;
    function GetWorkgroup(Name: TString): TNetworkWorkgroup;
  public
    { Процедура Оновлення шукає всі доступні робочі групи в мережі }
    procedure Refresh;
    { містить списки всіх комп'ютерів в  мережі}
    property Workgroup[Name: TString]: TNetworkWorkgroup read GetWorkgroup;

{ Функція FindComputer шукає комп'ютер зі вказаним ім'ям і повертає ім'я
робочої групи де його знайдено, або порожню строку
якщо комп'ютера з таким іменем у мережі немає }

function FindComputer(Name: TString): TString;
{ Процедура ListComputers копіює список всіх комп'ютерів в мережі
у об'єкті TStrings}
procedure ListComputers(Strings: TStrings);
{ Процедура ListNetwork копіює відсортований в алфавітному порядку список
всіх робочих груп і комп'ютерів в мережі.
Робочі групи мають ' TObject(1)' у відповідному елементі властивості
об'єктів, а комп'ютери – ' nil'}
    procedure ListNetwork(Strings: TStrings);
    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;
    constructor Create;
  end;
```

Наведемо частину коду, який реалізує вище перераховані дії.

```
procedure TForm1.ClickMeClick(Sender: TObject);
var
 N: TNetworkNeighborhood;
 List: TStringList;
 i, j: Integer;
 ListViewItem: TListItem;
 WorkgroupNode, ComputerNode: TTreeNode;
begin
 Screen.Cursor:=crHourGlass;
 try
  // Ініціалізація об'єктів та сканування мережі
  N:=TNetworkNeighborhood.Create;
  try
   // Отримання списку всіх комп'ютерів в мережі
   Memo1.Lines.Clear;
   N.ListComputers(Memo1.Lines);
   // Отримання списку всіх робочих груп і комп'ютерів в мережі,
   // відсортованих в алфавітному порядку
   List:=TStringList.Create;
   ListView1.Items.Clear;
   try
    N.ListNetwork(List);
    for i:=0 to List.Count - 1 do begin
     ListViewItem:=ListView1.Items.Add;
     ListViewItem.Caption:=List[i];
     ListViewItem.ImageIndex:=Integer(List.Objects[i]);
    end;
   finally
    List.Free;
   end;
   // Побудова дерева робочих груп і комп'ютерів в мережі
   TreeView1.Items.Clear;
   for i:=0 to N.Count - 1 do begin
    WorkgroupNode:=TreeView1.Items.Add(nil, N[i]);
    WorkgroupNode.ImageIndex:=1;
    WorkgroupNode.SelectedIndex:=1;
    for j:=0 to (N.Objects[i] as TStrings).Count - 1 do begin
     ComputerNode:=TreeView1.Items.AddChild(WorkgroupNode, (N.Objects[i] as
TStrings).Strings[j]);
     ComputerNode.ImageIndex:=0;
```

```
      end;
     end;
    // Отримання IP адрес комп'ютерів
    GetIPAddresses(N, Memo2.Lines);
    finally
     N.Free;
    end;
    TreeView1.FullExpand;
   finally
    Screen.Cursor:=crDefault;
   end;
  end;
```

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Threefish – в криптографії симетричний блоковий криптоалгоритм, розроблений автором Blowfish та Twofish, американським криптографом Брюсом Шнайером 2008 року для використання в хеш-функції Skein і як універсальну заміну наявним блоковим шифрам. Основними принципами розробки шифру були: мінімальне використання пам'яті, необхідна для використання в хеш-функції стійкість до атак, простота реалізації та оптимізація під 64-розрядні процесори.

Структура алгоритму

Threefish має дуже просту структуру і може бути використаний для заміни алгоритмів блочного шифрування, бувши швидким і гнучким шифром, що працюють в довільному режимі шифрування. Threefish S-блоки не використовує, заснований на комбінації інструкцій виключаючого або, складання і циклічного зсуву.

Як і AES, шифр реалізований у вигляді підстановочно-перестановочної мережі на оборотних операціях, не будучи шифром мережі Фейстел.

Алгоритм передбачає використання tweak-значення, свого роду вектора ініціалізації, дозволяючи змінювати таким чином значення виходу, без зміни

ключа, що має позитивний ефект як для реалізації нових режимів шифрування, так і на криптостойкости алгоритму.

Як результат думки авторів, що кілька складних раундів часто гірше ніж застосування великого числа простих раундів, алгоритм має нетрадиційно велику кількість раундів – 72 або 80 при ключі 1024 біт, проте, за заявою творців, його швидкісні характеристики випереджають AES приблизно вдвічі. Варто зауважити, що через 64-бітної структури шифру, дана заява має місцево лише на 64-розрядної архітектури. Тому, Threefish, як і Skein [1], заснований на ньому, на 32-розрядних процесорах показує значно гірші результати ніж на «рідному» обладнанні.

Ядром шифру є проста функція «MIX», перетворювальна два 64-бітових беззнакових числа, в процесі якої відбувається складання, циклічний зсув (ROL / ROR), і додавання по модулю 2 (XOR) .

Нижче представлений код MIX-функції для Threefish-1024 [2]:

```
<syntaxhighlight lang="C">
// Константи для циклічного зсуву
int R16 [8] [8] = {
  {55, 43, 37, 40, 16, 22, 38, 12},
  {25, 25, 46, 13, 14, 13, 52, 57},
  {33, 8, 18, 57, 21, 12, 32, 54},
  {34, 43, 25, 60, 44, 9, 59, 34},
  {28, 7, 47, 48, 51, 9, 35, 41},
  {17, 6, 18, 25, 43, 42, 40, 15},
  {58, 7, 32, 45, 19, 18, 2, 56},
  {47, 49, 27, 58, 37, 48, 53, 56},
};
// D – раунд, j – індекс в таблиці циклічного зсуву
void mix (int j, int d) {
  usigned long long rotl;
  y [0] = x [0] + x [1];
  rotl = R16 [d% 8] [j];
  y [1] = (x [1] << rotl) | (x [1] >> (64 – rotl));
  y [1] ^ = y [0];
}
</Source>
```

Процедура розшифрування обернена процедурі зашифрування і містить зворотну функцію DEMIX.

Кожен з 72 раундів Threefish-256 і Threefish-512 має чотири MIX перетворення, Threefish-1024 – вісім звернень до MIX функції.

Безпека

За заявою авторів, алгоритм має більш високий рівень безпеки, ніж AES. Існує атака на 25 з 72 раундів Threefish, в той час як для AES – на 6 з 10. Threefish має показник фактора безпеки 2.9, в свою чергу, AES всього 1.7 [3]

Для досягнення повної дифузії, шифру Threefish-256 досить 9 раундів, Threefish-512 – 10 раундів і Threefish-1024 – 11 раундів. Виходячи з цього, 72 і 80 раундів відповідно в середньому, забезпечать кращі результати, ніж існуючі шифри. [4]

У той же час, алгоритм має набагато простішу структуру і функцію перетворення, проте виконання 72-80 раундів, на думку дослідників, забезпечує необхідну стійкість. Вживаний розмір ключа від 256 до 1024 біт зводить нанівець можливість повного перебору паролів при так званій атаці грубою силою (brute force attack) на сучасному обладнанні.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. З нього видно, що інтерфейс користувача програми складається з таких логічних блоків:

– Меню: Мережа; Налаштування; Довідка.

– Вузли мережі № 1 – № N (де N кількість вузлів що використовується).

– Поточні дії програми.

– Функції: Формування сигналів керування перерозподілу трафіку мережі; Статистичне навантаження на вузли мережі; Аналіз статистичного навантаження; Налаштування; Звіт роботи.

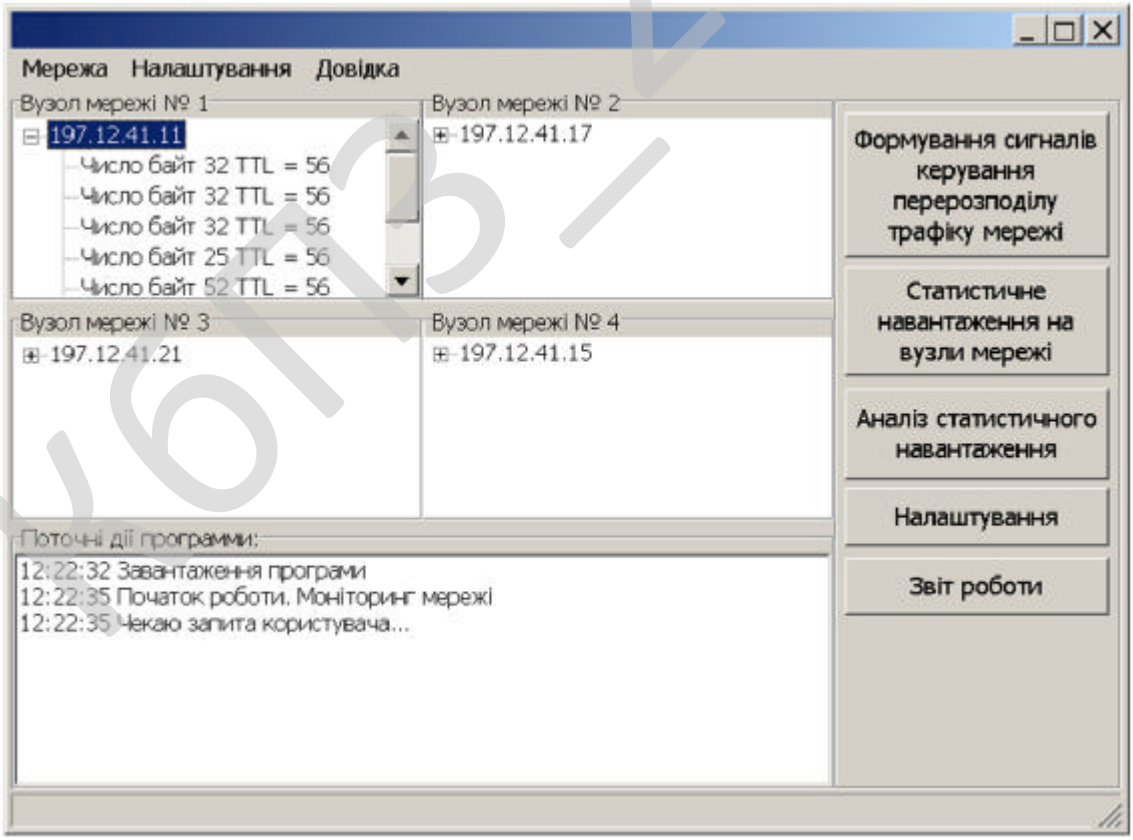

Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено форму авторського права, де відображені дані розробника. Було обрано Shareware умову розповсюдження.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (неповнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно.

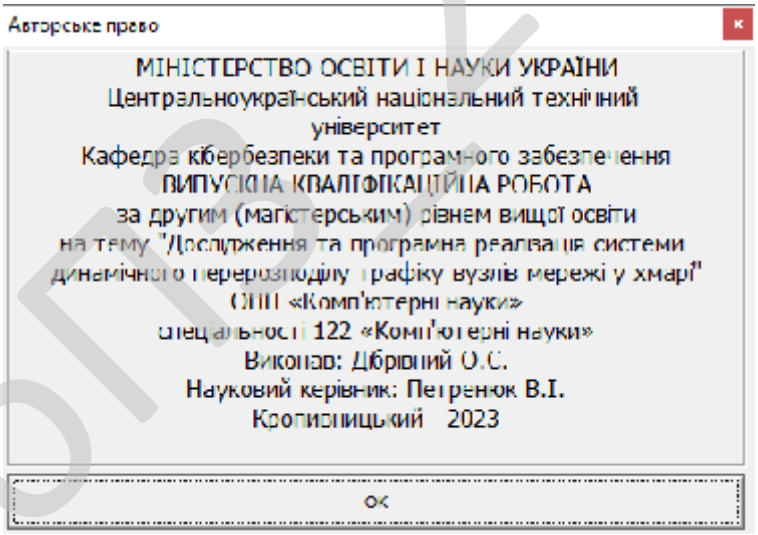

Рисунок 5.2 – Довідка

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Метою розробки є дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Об'єктом дослідження є процес динамічного перерозподілу трафіку вузлів мережі у хмарі.

Предметом дослідження є методи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Методи дослідження базуються на методах хмарних технологій, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод динамічного перерозподілу трафіку вузлів мережі у хмарі.

– Розроблено вітчизняний продукт динамічного перерозподілу трафіку вузлів мережі у хмарі, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці).

В магістерській роботі було проведене дослідження та розроблене програмне забезпечення системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	130
3. Запланований термін розробки, днів	Fpq	48 (2 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження табл. 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	_	2
13. Рівень спрацьованості колективу (1-6)	_	2
14. Ступінь вимірності процесів (1-6)	_	3
15. Необхідна надійність програмного забезпечення (1-6)	_	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	_	2
17. Складність кінцевого програмного продукту (1-6)	_	2
18. Необхідний рівень забезпечення повторного використання (1-6)	_	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	_	2
20. Вимоги до швидкодії ПП (1-6)	_	2
21. Обмеження на розміри основного сховища даних (1-6)	_	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	_	2
23. Професійний рівень аналітиків (1-6)	_	2
24. Професійний рівень програмістів (1-6)	_	2
25. Постійність складу команди розробників (1-6)	_	2
26. Досвід розробки додатків (1-6)	_	2
27. Досвід роботи з обчислювальною платформою (1-6)	_	2

Продовження табл. 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	13000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B \qquad (7.1)$$

де A – коефіцієнт Боема, A=2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001\sum W_i \qquad (7.2)$$

де W_i – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПВ_j, \qquad (7.3)$$

де $ПВj$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6{,}78\cdot(0{,}88\cdot0{,}93\cdot0{,}88\cdot0{,}91\cdot0{,}95\cdot1\cdot1\cdot0{,}87\cdot1{,}22\cdot1{,}16\cdot1{,}1\cdot1{,}1\cdot$$
$$\cdot1{,}12\cdot1{,}1\cdot1\cdot1{,}1\cdot1{,}1)=9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3CT_{уточн}^{0{,}33+0{,}2(B-1{,}01)}S, \qquad (7.4)$$

де С – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot2{,}66\cdot9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot53=89 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	89	Ф 7.1-7.4
Впровадження	13	Д13
Всього	130	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів,

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{130 \cdot 1}{48\text{-}5} = 3 \text{ ставки}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	8	720	12
Монітор	60	8	480	8
Клавіатура	30	8	240	4
Маніпулятор «мишка»	30	8	240	4
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	2	60	1
Кабельні господарства ЛОМ на 1 м.п.	2,5	200	500	8,33
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	42,99

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{43 \cdot 2}{1,2} = 72 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=72/(48·8)=0,2 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів – електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	Кількість штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server, серверу доступу АДСЛ (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,2	0,1
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,2	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,2	
	Забезпечення цілодобової роботи зв'язку клієнтів д мережі Інтернет	0,2	
Всього		0,8	

Продовження табл. 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,4
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	1	
	Контроль взаєморозрахунків з постачальниками	0,2	
Всього		3,2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,2	0,1
	Створення графічних і стилістичних елементів сайту	0,2	
	Оформлення банерів і промо-сторінок	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		0,8	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	0,2	0,1
	Верстка друкованих видань	0,2	
	Додрукова підготовка макетів	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		0,8	

Складемо штатний розклад виконавців:

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (ІТ-менеджер)	1	13311	26622
Продакт-менеджер	0,4	13000	10400
Інженер-програміст	3	14000	84000
Інженер-електронщик	0,2	13000	5200
Інженер-системотехнік	0,1	13000	2600
Адміністратор мережі	0,1	13000	2600
Системний програміст	0,1	13000	2600
Дизайнер WEB	0,1	13000	2600
Інженер-верстальник	0,1	13000	2600
Бухгалтер-економіст	0,1	14000	2800
Всього за період розробки	$R_{сп}$=5,2	-	$Ф_{роб}$=142022

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{142022}{5,2 \cdot 48} = 569 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9$$

де $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць.

$S_у$ – питома площа на одне робоче місце, $м^2$,

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 400…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$I_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались за прайсом фірми Supercomp за 30.10.23 – джерело https://supercomp.kiev.ua/

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		12721
Системний блок		7721
Процесор	AMD Ryzen 3 4100 (100-100000510BOX) AM4, 4 ядра, 8 потоків, 3.8 GHz, 4.0 GHz, TDP – 65 Вт, 7nm	-
Системна плата	ASUS PRIME A520M-A II/CSM, DDR4, 128 ГБ, 4800 MHz, LAN – 1 Гбіт/с, D-Sub (VGA), DisplayPort, HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX	-
Відеокарта	Radeon R7 350 2Gb Afox PCI-Express 3.0, 2 ГБ, GDDR5, 128 Bit	-
Жорсткий диск	SSD 2.5" 240GB Mibrand (MI2.5SSD/SP240GB) Серія – Spider, 240 GB, 3D TLC NAND, 2.5", SATA III (6Gb/s)	-
Оперативна пам’ять	DDR4 8GB 2400 MHz Patriot (PSD48G240081)	-
DVD-привод	DVD±RW ASUS DRW-24B5ST Black Bulk	-
Корпус	Vinga CS104B-500W Класичний, Miditower, ATX, Micro – ATX, Mini – ITX, PSU – 500 Вт, Слотів – 7, додатково – Back 1x12 см, Front 1x12 см, GPU – до 270 мм, PSU – 200 мм, 180 x 410 x 410 мм	-
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	-
інше	Клавіатура, мишка	Подаруно

Продовження таблиці 7.5

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Монітор	Монітор BenQ GL2450HM Black	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійног живлення	Powercom BNT-600AP USB	1400

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межа до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складем таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	8	12721	10176,8	111944,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	133576,3

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	137536	-	-
Всього по групі	137536	30	41260,8
Нематеріальні активи			
4. Нематеріальні активи	13000	25	3250
Група 5, 6			
5. Вимірювальні пристрої	9031	25	2257,75
6. Транспортні засоби	121875	20	24375
7. Господарський інвентар	28000	25	7000
Всього по групі	158906	-	33632,75
Разом	$К_р$ = 1717442		$А_р$ = 148543,55

Примітка: вартість автомобіля взята за даними електронного ресурсу https://auto.ria.com/uk/auto_citroen_berlingo_pass _33583056.html і складає 121875 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де $N_е$ – Кількість екземплярів програм, шт.

$$З_о = 569 \cdot 130/130 = 569 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 569 \cdot 10 \cdot 0{,}01 = 57 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 22(569+57) = 138 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом Нг=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 569 \cdot 15 \cdot 0{,}01 = 85 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картриджів, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 1/3 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=210 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 210 \cdot 1/3 = 70 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 38):

$$З_{М2} = \sum Ц_д\,, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 26 грн./шт., DVD-R box – 26 грн./шт.

$$З_{М2} = 38 \cdot 26 = 922 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574 + 558 + 570 = 1702 \text{ грн.}$$

$$З_М = (70 + 922 + 1702)/130 = 21 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

де $Н_п$ – норматив витрат на освоєння нових мов програмування, %

$$О_п = 569 \cdot 15 \cdot 0{,}01 = 85 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 130 прим.)

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де Ар – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = 148544 \cdot 2/(130 \cdot 12) = 190 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$C_п = З_о + З_д + C_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$C_п = 569+57+138+85+21+85+190 = 1145 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($P_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де $P_с$ – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 50 \cdot 1145 = 572{,}5 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	569
2. Додаткова зарплата виконавців	$З_д$	57
3. Відрахування на соціальні потреби	$C_{оц}$	138
4. Загальногосподарські витрати	$Г_{осп}$	85
5. Витрати на матеріали	$З_М$	21
6. Освоєння нових операційних систем, мов програмування	$О_п$	85

Продовження таблиці 7.9

1	*2*	3
7. Амортизація основних фондів	$А_м$	190
8. Повна собівартість програмного забезпечення	$С_п$	1145
9. Плановий прибуток	$П_р$	572,5
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	1717,5
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	*ПДВ*	343,5
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	*Ц*	2061

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	2061
Всього капітальних витрат	–	2061

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування системи	$З_р$	26840	14762
2. Витрати на амортизацію	$З_{ам}$	0	515,25
Всього витрат за рік	$І$	26840	15277,25

Витрати на технічне обслуговування і підтримку працездатності системи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \qquad (7.23)$$

де $Т_р$ – кількість годин обслуговування за рік, *год.*,

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 200 годин на рік до 110 годин на рік, тому витрати на технічне обслуговування зменшилися з

$$З_{р\,баз} = 200 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 26840 \text{ грн.}$$

до

$$З_{р\,нов} = 110 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 14762 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	2061	–	515,25
Всього відрахувань	-	–	2061	–	515,25

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.24)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (1717,5 - 1145) \cdot 130 - (0,05 \cdot 1408000 + 0,3 \cdot 137536 +$$
$$+0,2 \cdot 121875 + 0,25 \cdot 37031 + 0,25 \cdot 13000) \cdot 2/12 = 49668 \; грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.25)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{310442}{(1717,5 - 1145) \cdot 130 \cdot 12 / 2} = 0,7 \; років$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	130
2. Повна собівартість розробленої програми	Грн.	1145
3. Ціна розробленої програми	Грн.	1717,5
4. Плановий прибуток від реалізації розробленої програми	Грн.	572,5
5. Рентабельність програмної продукції	%	50
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1717442
7. Загальний прибуток від реалізації програмної продукції	Грн.	74425
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	49668
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,7
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	2061
11.Величина економічного ефекту у користувача програмної продукції	Грн.	11048
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Роки	0,18

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \qquad (7.26)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $К_б$, $К_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (26840\text{-}15277)\text{-}0{,}25 \cdot 2061 = 11048 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{К_н - К_б}{I_б - I_н} \tag{7.27}$$

$$T_{сп} = \frac{2061}{26840 - 15277} = 0{,}18 \text{ років}$$

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Законом України “Про охорону праці” [1] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207 [4], який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2].

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням [2]. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [2], та «Правила охорони праці під час експлуатації електронно-обчислювальних машин» НПАОП 0.00-1.28-10,

Умови праці програміста вулючають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення впливу комп'ютера на організм програміста визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста,

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машин (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98 [2].

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

– ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.

– ризик виникнення пожежі;

– негативний вплив на органи зору людини;

– ризики ураження електричним струмом;

– недостатня, або надмірна освітленість робочого місця;

– електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

– несприятливі мікрокліматичні умови;

– нервово-емоційна напруженість праці;

– інтелектуальні навантаження;

– монотонність праці;

– невідповідність ергономічних показників робочого місця діючим вимогам;

– шум;

– статичні навантаження на кістково-м’язовий апарат.

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	4
Довжина	5
Висота	3

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6,6
Обсяг, V	$м^3$	не менше 20.0	20

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин) [2].

У зазначеному приміщенні працює 2 людей. За даними, які наведено у табл. 8.1 та табл. 8.2, можна зробити висновок, що площа приміщення у розрахунку на одно робоче місце програміста відповідають нормативним вимогам (Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про

затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», а об'єм – ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2] та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови Головного державного санітарного лікаря України [5], робота, яка виконується в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря у приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22,5-23	45-60	0,11
Тепла	23-25	50-70	0,1	23-25	52-70	0,1

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер HP Laser 107a, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5 – 28 – 2006 р можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному

висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов’язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору).

8.5 Розрахункова частина

Завдання: розрахувати *штучне освітлення робочого приміщення.*

Початкові дані: ширина *робочого* приміщення: 4 м.; довжина – 5 м.; висота – 3 м.

Розрахунок штучного освітлення проводемо за методом коефіцієнта використання світлового потоку.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F=ESKZ/n,$$

де: *F* – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення (у нашому випадку *S*=3,5 х 3,5 = 11,9 $м^2$);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін **($\rho_{стін.}$)** і стелі **($\rho_{стелі}$),** значення коефіцієнтів дорівнюють $\rho_{стін}$ = **50% і $\rho_{стелі}$ = 50%** *[6]***.**

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де: S – площа приміщення, $S = 11{,}9\ м^2$;

h – розрахункова висота підвісу, $h = 3\ м$;

A – ширина приміщення, $A = 3{,}5\ м$;

B – довжина приміщення, $B = 3{,}5\ м$.

Підставимо всі значення у формулу та визначемо індекса приміщення: $i=0{,}57$.

Знаючи індекс приміщення, за знаходимо $n = 0.29$ *(з табличних даних коефіцієнтів використання світлового потоку (n) світильників відповідного типу [6])*. Підставимо всі значення у формулу, визначемо світловий потік: $F=21521$ *Лм.*

Для штучного освітлення приміщення використовуються *LED панель MAXUS ASSISTANCE PRO 80W 5000K WHITE (M1052480531)*, світловий потік яких $F_л = 8000$ Лм.

Число світильників визначається по формулі:

$$N=F/F_л$$

де: F – світловий потік,

$F_л$ – світловий потік одного *світильника*.

$$N= 21521/ 8000=2{,}7\ шт.$$

Приймаємо необхідну кількість світильників 3 шт.

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок захисного штучного освітлення. Розроблено заходи з охорони праці.

Список використаних джерел інформації

1. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

2. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2.007-98. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

3. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

4. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

5. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. – Кіровоград: КІСМ, 1997. – 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358 (дата звернення: 16.06.2023).

6. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

7. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : "Колос", 1973. – 238 с.

8. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький: ЦНТУ, 2022. – 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення: 16.06.2023).

9. Методичні рекомендації до виконання розділу «Заходи з охорони праці та техніки безпеки» у магістерській дисертації / Л.Д. Третякова; М-во освіти і науки України, Національний технічний університет україни «Київський політехнічний інститут» – Київ, КПІ, 2014. – 26 с. – Режим доступу до ресурсу: http://surl.li/dhulo (дата звернення: 16.06.2023).

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів динамічного перерозподілу трафіку вузлів мережі у хмарі.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем динамічного перерозподілу трафіку вузлів мережі у хмарі.

– Досліджена система динамічного перерозподілу трафіку вузлів мережі у хмарі.

– На основі отриманих результатів досліджень створена програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання динамічного перерозподілу трафіку вузлів мережі у хмарі.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

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

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

Програма реалізована на мові високого рівня Delphi 10.4.1. Дана мова програмування дозволяє найбільш ефективно обробляти дані. 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 Threefish.

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.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 11048 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,18 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дібрівний О.С. Дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

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

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

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

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

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

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

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

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

10. 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.

11. 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.

12. 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.

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

14. 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.

15. 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.

16. 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.

17. 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.

18. 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.

19. 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.

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

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

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

30. 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.

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

32. 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.

33. 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.

34. 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.

35. 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

36. 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.

37. 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.

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

39. 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.

40. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга

Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

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

42. 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).

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

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

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

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

47. 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.

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

49. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв 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.

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

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

52. 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.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 33-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи динамічного перерозподілу трафіку вузлів мережі у хмарі;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Delphi 10.4.1.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 105 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2023 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 19.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Петренюк В.І.

Дослідження та програмна реалізація системи динамічного перерозподілу трафіку вузлів мережі у хмарі

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2023 року

Файл Networks.pas – робота з мережею

```
unit Networks;

interface

uses Windows, ShellAPI, ShlObj, ActiveX, ComObj, Dim, Classes, SysUtils,
WinSock;

type
  ComputerFound = class (Exception);
  ECannotFindNetwork = class (Exception);

  TStringObject = class (TObject)
  private
    FValue: TString;
    FTag: Integer;
    FData: Pointer;
    FRefObj: TObject;
    procedure SetValue(const Value: TString);
    procedure SetData(const Value: Pointer);
    procedure SetRefObj(const Value: TObject);
    procedure SetTag(const Value: Integer);
  public
    property Value: TString read FValue write SetValue;
    property RefObj: TObject read FRefObj write SetRefObj;
    property Tag: Integer read FTag write SetTag;
    property Data: Pointer read FData write SetData;
  end;

  TStringObjectArray = class (TDynamicArray)
  private
    function GetObject(Index: Integer): TStringObject;
    function GetData(Index: Integer): Pointer;
    function GetRefObj(Index: Integer): TObject;
    function GetTag(Index: Integer): Integer;
    function GetValue(Index: Integer): TString;
    procedure SetData(Index: Integer; const Value: Pointer);
    procedure SetRefObj(Index: Integer; const Value: TObject);
    procedure SetTag(Index: Integer; const Value: Integer);
    procedure SetValue(Index: Integer; const Value: TString);

    function FreeObject(Index: Integer; var Obj: TStringObject): Integer;

    procedure FreeItem(Index: Integer);
    procedure CreateItem(Index: Integer);

  protected
    procedure SetCount(const NewCount: Cardinal); override;
  public
    function Add: Integer; override;
    procedure Insert(Index: Integer); override;
    procedure Delete(Index: Integer); override;
    function AddItem(const Item): Integer; override;
    procedure InsertItem(Index: Integer; const Item); override;
    procedure DeleteItem(Index: Integer; out Item); override;
    property Value[Index: Integer]: TString read GetValue write SetValue;
default;
    property RefObj[Index: Integer]: TObject read GetRefObj write SetRefObj;
    property Tag[Index: Integer]: Integer read GetTag write SetTag;
    property Data[Index: Integer]: Pointer read GetData write SetData;
    constructor Create;
    destructor Destroy; override;
  end;

  TStringObjectList = class (TStrings)
```

```
  private
    FArray: TStringObjectArray;
    function GetData(Index: Integer): Pointer;
    function GetTag(Index: Integer): Integer;
    procedure SetData(Index: Integer; const Value: Pointer);
    procedure SetTag(Index: Integer; const Value: Integer);
  protected
    function Get(Index: Integer): string; override;
    function GetCount: Integer; override;
    function GetObject(Index: Integer): TObject; override;
    procedure Put(Index: Integer; const S: string); override;
    procedure PutObject(Index: Integer; AObject: TObject); override;

  public
    property Data[Index: Integer]: Pointer read GetData write SetData;
    property Tag[Index: Integer]: Integer read GetTag write SetTag;

    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Exchange(Index1, Index2: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;


    constructor Create;
    destructor Destroy; override;
  end;

  {TNetworkWorkgroup - клас, який створює список всіх комп' ютерів в робочій
групі. Цей клас є нащадком класу TStrings і повністю сумісний з іншим нащадками
цього класу.  Об' єкти цього класу записуються у властивості об' єктів класу
TNetworkNeiborhood. Клас TNetworkNeiborhood містить об' єкти і властивості
робочих груп. }

  TNetworkWorkgroup = class (TStringObjectList);


  {TNetworkNeiborhood - клас, який створює список всіх робочих груп в мережі}
  TNetworkNeighborhood = class (TStringObjectList)
  private
    function CreatePIDL(Size: Integer): PItemIDList;
    procedure DisposePIDL(ID: PItemIDList);
    function NextPIDL(IDList: PItemIDList): PItemIDList;
    function GetPIDLSize(IDList: PItemIDList): Integer;
    function CopyPIDL(IDList: PItemIDList): PItemIDList;
    procedure StripLastID(IDList: PItemIDList);
    function GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
    class function GetDisplayName(ShellFolder: IShellFolder; PIDL: PItemIDList):
TString;
    function OriginFolder:  IShellFolder;
    function OriginFolderNT: IShellFolder;
    class function EnumObjects(ShellFolder: IShellFolder): IEnumIDList;
    class procedure ParseFolder(Folder: IShellFolder; Items: TStringObjectList;
StorePIDLs: Boolean = False);
    class procedure ParseFolderEx(Folder: IShellFolder; Items: TStrings);

    function FreeRefObj(Index: Integer; var Obj: TStringObject): Integer;
    function GetWorkgroup(Name: TString): TNetworkWorkgroup;

  public
    { Процедура Оновлення шукає всі доступні робочі групи в мережі }

    procedure Refresh;

    { містить списки всіх комп' ютерів в  мережі}

    property Workgroup[Name: TString]: TNetworkWorkgroup read GetWorkgroup;
```

```
   { Функція FindComputer шукає комп’ ютер зі вказаним ім’ ям і повертає ім’ я
робочої групи де його знайдено, або порожню строку
     якщо комп’ ютера з таким іменем у мережі немає }

    function FindComputer(Name: TString): TString;

    { Процедура ListComputers копіює список всіх комп’ ютерів в мережі
      у об’ єкті TStrings}
    procedure ListComputers(Strings: TStrings);

    { Процедура ListNetwork копіює відсортований в алфавітному порядку список
всіх робочих груп і комп’ ютерів в мережі.
     Робочі групи мають ‘ TObject(1)’  у відповідному елементі властивості об’
єктів, а комп’ ютери - ‘ nil’ }

    procedure ListNetwork(Strings: TStrings);

    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;
    constructor Create;
  end;

{ Функція GetIPAddress отримує IP адресу комп’ ютера або сервера.
  Параметр NetworkName конкретизує ім’ я комп’ ютера або  сервера.
  Ця функція повертає адреси IP у форматі XXX.XXX.XXX.XXX у разі успішного
виконання, ‘ Error’  - коли неможливо ініціалізувати, ‘ Unknown’  – коли
параметр NetworkName посилається на неіснуючий комп’ ютер або на комп’ ютер  без
встановленого протоколу TCP/IP }

  function GetIPAddress(NetworkName: TString): TString;


{ GetIPAddresses отримує IP адреси всіх доступних комп’ ютерів мережі }

procedure GetIPAddresses(Network: TNetworkNeighborhood; List: TStrings);



{ Функція EnumSharedResources перераховує загальні ресурси мережі. Параметр
ComputerName конкретизує
  ім’ я комп’ ютера. }

function EnumSharedResources(ComputerName: TString; List: TStrings): Boolean;


implementation

uses NetConst;

{ TStringObject }

procedure TStringObject.SetData(const Value: Pointer);
begin
  FData := Value;
end;

procedure TStringObject.SetRefObj(const Value: TObject);
begin
  FRefObj := Value;
end;

procedure TStringObject.SetTag(const Value: Integer);
begin
  FTag := Value;
end;

procedure TStringObject.SetValue(const Value: TString);
```

```
begin
  FValue := Value;
end;

{ TStringObjectArray }

function TStringObjectArray.Add: Integer;
begin
 Result:=inherited Add;
 CreateItem(Result);
end;

function TStringObjectArray.AddItem(const Item): Integer;
begin
 Result:=Add;
end;

constructor TStringObjectArray.Create;
begin
 inherited Create(0, SizeOf(TStringObject));
end;

procedure TStringObjectArray.CreateItem(Index: Integer);
var
 P: ^TStringObject;
begin
 P:=GetItemPtr(Index);
 P^:=TStringObject.Create;
end;

procedure TStringObjectArray.Delete(Index: Integer);
begin
 FreeItem(Index);
 inherited;
end;

procedure TStringObjectArray.DeleteItem(Index: Integer; out Item);
begin
 Delete(Index);
end;

destructor TStringObjectArray.Destroy;
begin
 ForEach(Integer(Self), @TStringObjectArray.FreeObject);
 inherited;
end;

procedure TStringObjectArray.FreeItem(Index: Integer);
var
 P: ^TStringObject;
begin
 P:=GetItemPtr(Index);
 FreeAndNil(P^);
end;

function TStringObjectArray.FreeObject(Index: Integer;
  var Obj: TStringObject): Integer;
begin
 FreeAndNil(Obj);
 Result:=0;
end;

function TStringObjectArray.GetData(Index: Integer): Pointer;
begin
 Result:=GetObject(Index).Data;
end;

function TStringObjectArray.GetObject(Index: Integer): TStringObject;
begin
```

```
 GetItem(Index, Result);
end;

function TStringObjectArray.GetRefObj(Index: Integer): TObject;
begin
 Result:=GetObject(Index).RefObj;
end;

function TStringObjectArray.GetTag(Index: Integer): Integer;
begin
 Result:=GetObject(Index).Tag;
end;

function TStringObjectArray.GetValue(Index: Integer): TString;
begin
 Result:=GetObject(Index).Value;
end;

procedure TStringObjectArray.Insert(Index: Integer);
begin
 inherited;
 CreateItem(Index);
end;

procedure TStringObjectArray.InsertItem(Index: Integer; const Item);
begin
 Insert(Index);
end;

procedure TStringObjectArray.SetCount(const NewCount: Cardinal);
var
 i, OldCount: Integer;
begin
 OldCount:=Count;
 if NewCount > Count then begin
  inherited SetCount(NewCount);
  for i:=OldCount to NewCount - 1 do CreateItem(i);
 end else if NewCount < Count then begin
  for i:=NewCount to OldCount - 1 do FreeItem(i);
  inherited SetCount(NewCount);
 end;
end;

procedure TStringObjectArray.SetData(Index: Integer; const Value: Pointer);
begin
 GetObject(Index).Data:=Value;
end;

procedure TStringObjectArray.SetRefObj(Index: Integer;
  const Value: TObject);
begin
 GetObject(Index).RefObj:=Value;
end;

procedure TStringObjectArray.SetTag(Index: Integer; const Value: Integer);
begin
 GetObject(Index).Tag:=Value;
end;

procedure TStringObjectArray.SetValue(Index: Integer; const Value: TString);
begin
 GetObject(Index).Value:=Value;
end;

{ TStringObjectList }

function TStringObjectList.Add(const S: string): Integer;
begin
 Result:=FArray.Add;
```

```
 FArray.Value[Result]:=S;
end;

procedure TStringObjectList.Clear;
begin
 FArray.Count:=0;
end;

constructor TStringObjectList.Create;
begin
 inherited Create;
 FArray:=TStringObjectArray.Create;
end;

procedure TStringObjectList.Delete(Index: Integer);
begin
 FArray.Delete(Index);
end;

destructor TStringObjectList.Destroy;
begin
 FArray.Free;
 inherited;
end;

procedure TStringObjectList.Exchange(Index1, Index2: Integer);
begin
 FArray.Swap(Index1, Index2);
end;

function TStringObjectList.Get(Index: Integer): string;
begin
 Result:=FArray.Value[Index];
end;

function TStringObjectList.GetCount: Integer;
begin
 Result:=FArray.Count;
end;

function TStringObjectList.GetData(Index: Integer): Pointer;
begin
 Result:=FArray.Data[Index];
end;

function TStringObjectList.GetObject(Index: Integer): TObject;
begin
 Result:=FArray.RefObj[Index];
end;

function TStringObjectList.GetTag(Index: Integer): Integer;
begin
 Result:=FArray.Tag[Index];
end;

procedure TStringObjectList.Insert(Index: Integer; const S: string);
begin
 FArray.Insert(Index);
 FArray.Value[Index]:=S;
end;

procedure TStringObjectList.Put(Index: Integer; const S: string);
begin
 FArray.Value[Index]:=S;
end;

procedure TStringObjectList.PutObject(Index: Integer; AObject: TObject);
begin
 FArray.RefObj[Index]:=AObject;
```

```
end;

procedure TStringObjectList.SetData(Index: Integer; const Value: Pointer);
begin
 FArray.Data[Index]:=Value;
end;

procedure TStringObjectList.SetTag(Index: Integer; const Value: Integer);
begin
 FArray.Tag[Index]:=Value;
end;

{ TNetworkNeighborhood }

function TNetworkNeighborhood.Add(const S: string): Integer;
begin
 Result:=inherited Add(S);
 Objects[Result]:=TNetworkWorkgroup.Create;
end;

procedure TNetworkNeighborhood.Clear;
begin
 FArray.ForEach(Integer(Self), @TNetworkNeighborhood.FreeRefObj);
 inherited;
end;

function TNetworkNeighborhood.CopyPIDL(IDList: PItemIDList): PItemIDList;
var
 Size: Integer;
begin
 Size := GetPIDLSize(IDList);
 Result := CreatePIDL(Size);
 if Assigned(Result) then CopyMemory(Result, IDList, Size);
end;

constructor TNetworkNeighborhood.Create;
begin
 inherited Create;
 Refresh;
end;

function TNetworkNeighborhood.CreatePIDL(Size: Integer): PItemIDList;
var
 Malloc: IMalloc;
 HR: HResult;
begin
 Result := nil;
 HR := SHGetMalloc(Malloc);
 if Failed(HR) then  Exit;
 try
  Result := Malloc.Alloc(Size);
  if Assigned(Result) then FillChar(Result^, Size, 0);
 finally
 end;
end;

procedure TNetworkNeighborhood.Delete(Index: Integer);
begin
end;

procedure TNetworkNeighborhood.DisposePIDL(ID: PItemIDList);
var
 Malloc: IMalloc;
begin
 if ID = nil then Exit;
 OLECheck(SHGetMalloc(Malloc));
 Malloc.Free(ID);
end;
```

```
class function TNetworkNeighborhood.EnumObjects(
  ShellFolder: IShellFolder): IEnumIDList;
const
 Flags = SHCONTF_FOLDERS or SHCONTF_NONFOLDERS or SHCONTF_INCLUDEHIDDEN;
begin
 ShellFolder.EnumObjects(0, Flags, Result);
end;

function TNetworkNeighborhood.FindComputer(Name: TString): TString;
var
 i, j: Integer;
 List: TNetworkWorkgroup;
 S: TString;
begin
 Result:=' ' ;
 try
  for i:=0 to Count - 1 do begin
   List:=Objects[i] as TNetworkWorkgroup;
   for j:=0 to List.Count - 1 do begin
    S:=List[j];
    CleanUp(S);
    if EqualText(Name, S) then begin
     Result:=Strings[i];
     raise ComputerFound.Create(' ' );
    end;
   end;
  end;
 except
  if not (ExceptObject is ComputerFound) then raise;
 end;
end;

function TNetworkNeighborhood.FreeRefObj(Index: Integer;
  var Obj: TStringObject): Integer;
begin
 FreeAndNil(Obj.FRefObj);
 Result:=0;
end;

class function TNetworkNeighborhood.GetDisplayName(ShellFolder: IShellFolder;
  PIDL: PItemIDList): TString;
var
 StrRet: TStrRet;
 P: PChar;
begin
 Result := ' ' ;
 ShellFolder.GetDisplayNameOf(PIDL, SHGDN_NORMAL, StrRet);
 case StrRet.uType of
  STRRET_CSTR: SetString(Result, StrRet.cStr, lStrLen(StrRet.cStr));
  STRRET_OFFSET: begin
   P := @PIDL.mkid.abID[StrRet.uOffset - SizeOf(PIDL.mkid.cb)];
   SetString(Result, P, PIDL.mkid.cb - StrRet.uOffset);
  end;
  STRRET_WSTR: Result := StrRet.pOleStr;
 end;
 CleanUp(Result, True);
end;

function TNetworkNeighborhood.GetPIDLSize(IDList: PItemIDList): Integer;
begin
 Result := 0;
 if Assigned(IDList) then begin
  Result := SizeOf(IDList^.mkid.cb);
  while IDList^.mkid.cb <> 0 do begin
   Result := Result + IDList^.mkid.cb;
   IDList := NextPIDL(IDList);
  end;
 end;
end;
```

```
function TNetworkNeighborhood.GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
var
 Temp: PItemIDList;
begin
 Temp := CopyPIDL(PIDL);
 if Assigned(Temp) then StripLastID(Temp);
 if Temp.mkid.cb <> 0 then Result:=Temp else Result:=nil;
end;

function TNetworkNeighborhood.GetWorkgroup(Name: TString): TNetworkWorkgroup;
var
 Index: Integer;
begin
 Index:=IndexOf(Name);
 if Index<>-1 then Result:=Objects[Index] as TNetworkWorkgroup else Result:=nil;
end;

procedure TNetworkNeighborhood.Insert(Index: Integer; const S: string);
begin
end;

procedure TNetworkNeighborhood.ListComputers(Strings: TStrings);
var
 i, j: integer;
 L: TNetworkWorkgroup;
 S: TString;
begin
 Strings.BeginUpdate;
 try
  Strings.Clear;
  for i:=0 to Count - 1 do begin
   L:=Objects[i] as TNetworkWorkgroup;
   for j:=0 to L.Count - 1 do begin
    S:=L[j];
    CleanUp(S);
    Strings.Add(S);
   end;
  end;
 finally
  Strings.EndUpdate;
 end;
end;

procedure TNetworkNeighborhood.ListNetwork(Strings: TStrings);
var
 List: TStringList;
 i: Integer;
begin
 List:=TStringList.Create;
 try
  List.AddStrings(Self);
  for i:=0 to List.Count - 1 do List.Objects[i]:=TObject(1);
  for i:=0 to Count - 1 do begin
   List.AddStrings(Objects[i] as TStrings);
  end;
  for i:=Count to List.Count - 1 do List.Objects[i]:=nil;
  List.Sort;
  Strings.Assign(List);
 finally
  List.Free;
 end;
end;

function TNetworkNeighborhood.NextPIDL(IDList: PItemIDList): PItemIDList;
begin
 Result := IDList;
 Inc(PChar(Result), IDList^.mkid.cb);
end;
```

```
function TNetworkNeighborhood.OriginFolder: IShellFolder;
var
 Desktop: IShellFolder;
 S: TString;
 P: PWideChar;
 Len, Flags: LongWord;
 Machine, Workgroup, Network: PItemIDList;
begin
 S:=' \\' +GetComputerName;
 Len:=Length(S);
 P:=StringToOleStr(S);
 Flags:=0;
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Workgroup:=GetPrevPIDL(Machine);
 try
   Network:=GetPrevPIDL(Workgroup);
  try
   Desktop.BindToObject(Network, nil, IShellFolder, Pointer(Result));
  finally
   DisposePIDL(Network);
  end;
 finally
  DisposePIDL(Workgroup);
 end;
end;

function TNetworkNeighborhood.OriginFolderNT: IShellFolder;
var
 Desktop: IShellFolder;
 S: TString; W: WideString; P: PWideChar;
 Len, Flags: LongWord;
 Machine, Workgroup, Network: PItemIDList;
 NetShell: IShellFolder;
 Enum: IEnumIDList;
 ID: PItemIDList;
begin
 S:=' \\' +GetComputerName;
 Len:=Length(S);
 W:=S; P:=PWideChar(W);
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Workgroup:=GetPrevPIDL(Machine);
 Network:=GetPrevPIDL(Workgroup);
 Desktop.BindToObject(Network, nil, IShellFolder, NetShell);
 Enum:=EnumObjects(NetShell);
 Enum.Next(1, ID, Flags);
 NetShell.BindToObject(ID, nil, IShellFolder, Pointer(Result));
 DisposePIDL(Network);
 DisposePIDL(Workgroup);
end;

class procedure TNetworkNeighborhood.ParseFolder(Folder: IShellFolder;
  Items: TStringObjectList; StorePIDLs: Boolean);
var
 ID: PItemiDList;
 EnumList: IEnumIDList;
 NumIDs: LongWord;
 S: TString;
 Index: Integer;
begin
 Items.BeginUpdate;
 try
  Items.Clear;
  EnumList:=EnumObjects(Folder);
  if Assigned(EnumList) then while EnumList.Next(1, ID, NumIDs) = S_OK do begin
   S:=GetDisplayName(Folder, ID);
   Index:=Items.Add(S);
```

```
   if StorePIDLs then Items.Data[Index]:=ID;
  end;
 finally
  Items.EndUpdate;
 end;
end;

class procedure TNetworkNeighborhood.ParseFolderEx(Folder: IShellFolder;
  Items: TStrings);
var
 ID: PItemiDList;
 EnumList: IEnumIDList;
 NumIDs: LongWord;
 S: TString;
begin
 Items.BeginUpdate;
 try
  Items.Clear;
  EnumList:=EnumObjects(Folder);
  if Assigned(EnumList) then while EnumList.Next(1, ID, NumIDs) = S_OK do begin
   S:=GetDisplayName(Folder, ID);
   Items.Add(S);
  end;
 finally
  Items.EndUpdate;
 end;
end;

procedure TNetworkNeighborhood.Refresh;
var
 Network: IShellFolder;
 Workgroup: IShellFolder;
 i: Integer;
begin
 try
  if WinNT and (not Win2K) then Network:=OriginFolderNT else
   Network:=OriginFolder;
  ParseFolder(Network, Self, True);
  for i:=0 to Count - 1 do begin
   Network.BindToObject(PItemIDList(Data[i]), nil, IShellFolder, Workgroup);
   ParseFolder(Workgroup, Objects[i] as TStringObjectList, False);
   Workgroup:=nil;
  end;
 except
  raise ECannotFindNetwork.Create(SCannotFindNetwork);
 end;
end;

procedure TNetworkNeighborhood.StripLastID(IDList: PItemIDList);
var
 MarkerID: PItemIDList;
begin
 MarkerID := IDList;
 if Assigned(IDList) then begin
  while IDList.mkid.cb <> 0 do begin
   MarkerID := IDList;
   IDList := NextPIDL(IDList);
  end;
  MarkerID.mkid.cb := 0;
 end;
end;


procedure GetIPAddresses(Network: TNetworkNeighborhood; List: TStrings);
var
 Error: DWORD;
 HostEntry: PHostEnt;
 Data: WSAData;
 Address: In_Addr;
```

```
 i: Integer;
 TmpList: TStringList;
 S: TString;
begin
{ List.BeginUpdate;
 try}
  List.Clear;
  Error:=WSAStartup(MakeWord(1, 1), Data);
  if Error = 0 then begin
   TmpList:=TStringList.Create;
   try
    Network.ListComputers(TmpList);
    for i:=0 to TmpList.Count - 1 do begin
     HostEntry:=gethostbyname(PChar(TmpList[i]));
     Error:=GetLastError;
     if Error <> 0 then S:=' Unknown'  else begin
      Address:=PInAddr(HostEntry^.h_addr_list^)^;
      S:=inet_ntoa(Address);
     end;
     List.Add(Format(' %s [%s]' , [TmpList[i], S]));
    end;
   finally
    TmpList.Free;
   end;
  end else begin
   List.Add(' Error' );
  end;
{ finally
  List.EndUpdate;
 end;}
end;


function GetShellFolder(ComputerName: TString): IShellFolder;
var
 S: TString;
 W: WideString;
 P: PWideChar;
 Desktop: IShellFolder;
 Len, Flags: LongWord;
 Machine: PItemIDList;
begin
 S:=ComputerName;
 if Pos(' \\' , S) <> 1 then S:=' \\' +S;
 Len:=Length(S);
 W:=S;
 P:=@W[1];
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Desktop.BindToObject(Machine, nil, IShellFolder, Pointer(Result));
end;

function EnumSharedResources(ComputerName: TString; List: TStrings): Boolean;
var
 ShellFolder: IShellFolder;
begin
 ShellFolder:=GetShellFolder(ComputerName);
 Result:=Assigned(ShellFolder);
 if Result then TNetworkNeighborhood.ParseFolderEx(ShellFolder, List);
end;



function GetIPAddress(NetworkName: TString): TString;
var
 Error: DWORD;
 HostEntry: PHostEnt;
 Data: WSAData;
 Address: In_Addr;
```

```
begin
 Error:=WSAStartup(MakeWord(1, 1), Data);
 if Error = 0 then begin
  HostEntry:=gethostbyname(PChar(NetworkName));
  Error:=GetLastError();
  if Error = 0 then begin
   Address:=PInAddr(HostEntry^.h_addr_list^)^;
   Result:=inet_ntoa(Address);
  end else begin
   Result:=‘ Unknown’ ;
  end;
 end else begin
  Result:=‘ Error’ ;
 end;
 WSACleanup();
end;

end.
```

Основна програма

Файл Main.pas основної програми

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Dim, Networks, StdCtrls, ComCtrls, ImgList, ShellAPI, ShlObj, IpHlpApi,
IPtraff,
  ExtCtrls, About, Buttons;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    ClickMe: TButton;
    TreeView1: TTreeView;
    ImageList1: TImageList;
    ListView1: TListView;
    Memo2: TMemo;
    Button1: TButton;
    Label1: TLabel;
    Edit1: TEdit;
    Memo3: TMemo;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    MemoTR: TMemo;
    Timer1: TTimer;
    Label7: TLabel;
    Button2: TButton;
    Button3: TButton;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    MemoX: TMemo;
    Label8: TLabel;
    procedure ClickMeClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);

  private
    { Private declarations }
     procedure DOIpStuff;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
```

```
function GetShellFolder(CompName: TString): IShellFolder;
var
 S: TString;
 W: WideString;
 P: PWideChar;
 Desktop: IShellFolder;
 Len, Flags: LongWord;
 Machine: PItemIDList;
begin
 S:=CompName;
 if Pos(‘ \\’ , S) <> 1 then S:=‘ \\’ +S;
 Len:=Length(S);
 W:=S;
 P:=@W[1];
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Desktop.BindToObject(Machine, nil, IShellFolder, Pointer(Result));
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 Memo3.Lines.Clear;
 Screen.Cursor:=crHourGlass;
 try
  if not EnumSharedResources(Edit1.Text, Memo3.Lines)
   then raise Exception.Create(‘ Невірне ім"я комп"ютера’ );
 finally
  Screen.Cursor:=crDefault;
 end;
end;



procedure TForm1.ClickMeClick(Sender: TObject);
var
 N: TNetworkNeighborhood;
 List: TStringList;
 i, j: Integer;
 ListViewItem: TListItem;
 WorkgroupNode, ComputerNode: TTreeNode;
begin
 Screen.Cursor:=crHourGlass;
 try
  // Ініціалізація об’ єктів та сканування мережі
  N:=TNetworkNeighborhood.Create;
  try
   // Отримання списку всіх комп’ ютерів в мережі
   Memo1.Lines.Clear;
   N.ListComputers(Memo1.Lines);

   // Отримання списку всіх робочих груп і комп’ ютерів в мережі, відсортованих
в алфавітному порядку
   List:=TStringList.Create;
   ListView1.Items.Clear;
   try
    N.ListNetwork(List);
    for i:=0 to List.Count - 1 do begin
     ListViewItem:=ListView1.Items.Add;
     ListViewItem.Caption:=List[i];
     ListViewItem.ImageIndex:=Integer(List.Objects[i]);
    end;
   finally
    List.Free;
   end;
```

```
   // Побудова дерева робочих груп і комп’ютерів в мережі
   TreeView1.Items.Clear;
   for i:=0 to N.Count - 1 do begin
    WorkgroupNode:=TreeView1.Items.Add(nil, N[i]);
    WorkgroupNode.ImageIndex:=1;
    WorkgroupNode.SelectedIndex:=1;
    for j:=0 to (N.Objects[i] as TStrings).Count - 1 do begin
     ComputerNode:=TreeView1.Items.AddChild(WorkgroupNode, (N.Objects[i] as
TStrings).Strings[j]);
     ComputerNode.ImageIndex:=0;
    end;
   end;

  // Отримання IP адрес комп’ютерів
  GetIPAddresses(N, Memo2.Lines);

  finally
   N.Free;
  end;
  TreeView1.FullExpand;

 finally
  Screen.Cursor:=crDefault;
 end;
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
 Edit1.Text:=GetComputerName;

  if LoadIpHlp then
  begin
      DOIpStuff;
      Timer1.Enabled := true;
  end;

end;


procedure TForm1.Timer1Timer(Sender: TObject);
begin

   Timer1.Enabled := false;
   DoIPStuff;
   Timer1.Enabled := true;

end;


procedure TForm1.DOIpStuff;
begin

 Get_TCPTable( MemoX.Lines );
 Get_UDPTable( MemoTR.Lines );

end;


procedure TForm1.Button2Click(Sender: TObject);
begin

Form2.Show;

end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin

Form1.Close;

end;


procedure TForm1.SpeedButton2Click(Sender: TObject);
begin

Form1.Close;

end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin

Form2.Show;

end;


end.
```

Файл IPtraff.pas – відслідковування та динамічний перерозподіл трафіку

```
unit IPtraff;

interface

uses
  Windows, Messages, SysUtils, Classes, Dialogs, IpHlpApi;

const
  NULL_IP       = '   0.  0.  0.  0' ;


type
  TWellKnownPort = record
    Prt: DWORD;
    Srv: string[20];
  end;


const

  WellKnownPorts: array[1..32] of TWellKnownPort
  = (
//    ( Prt: 0; Srv:   ' RESRVED'  ),     {Зарезервовано}
    ( Prt: 7; Srv:   ' ECHO   '  ),     {Пінгування    }
    ( Prt: 9; Srv:   ' DISCARD'  ),
    ( Prt: 13; Srv:  ' DAYTIME'  ),
    ( Prt: 17; Srv:  ' QOTD   '  ),     {Показчик на день}
    ( Prt: 19; Srv:  ' CHARGEN'  ),     {Генератор символів}
    ( Prt: 20; Srv:  ' FTPDATA'  ),     { Протокол File Transfer Protocol -
дані}
    ( Prt: 21; Srv:  ' FTPCTRL'  ),     { Протокол File Transfer Protocol -
управління}
    ( Prt: 22; Srv:  ' SSH    '  ),
    ( Prt: 23; Srv:  ' TELNET '  ),
    ( Prt: 25; Srv:  ' SMTP   '  ),     { Протокол Simple Mail Transfer
Protocol}
    ( Prt: 37; Srv:  ' TIME   '  ),     { Протокол Time Protocol }
    ( Prt: 43; Srv:  ' WHOIS  '  ),     { WHO IS сервіс  }
    ( Prt: 53; Srv:  ' DNS    '  ),     { Domain Name Service }
    ( Prt: 67; Srv:  ' BOOTPS '  ),     { BOOTP сервер }
    ( Prt: 68; Srv:  ' BOOTPC '  ),     { BOOTP клієнт }
    ( Prt: 69; Srv:  ' TFTP   '  ),     { Стандартний  FTP  }
    ( Prt: 70; Srv:  ' GOPHER '  ),     { Протокол Gopher       }
    ( Prt: 79; Srv:  ' FINGER '  ),     { Протокол Finger       }
    ( Prt: 80; Srv:  ' HTTP   '  ),     { Протокол HTTP         }
    ( Prt: 88; Srv:  ' KERBROS'  ),     { Протокол Kerberos     }
    ( Prt: 109; Srv: ' POP2   '  ),     { Протокол Post Office Protocol Version
2 }
    ( Prt: 110; Srv: ' POP3   '  ),     { Протокол Post Office Protocol Version
3 }
    ( Prt: 111; Srv: ' SUN_RPC'  ),     { SUN віддалений виклик функцій }
    ( Prt: 119; Srv: ' NNTP   '  ),     { Протокол Network News Transfer
Protocol }
    ( Prt: 123; Srv: ' NTP    '  ),     { Протокол Network Time protocol
}
    ( Prt: 135; Srv: ' DCOMRPC'  ),     { Локальний сервісе             }
    ( Prt: 137; Srv: ' NBNAME '  ),     { NETBIOS сервіс імен          }
    ( Prt: 138; Srv: ' NBDGRAM'  ),     { NETBIOS сервіс датаграм     }
    ( Prt: 139; Srv: ' NBSESS '  ),     { NETBIOS сервіс сесій        }
    ( Prt: 143; Srv: ' IMAP   '  ),     { Протокол Internet Message Access
Protocol }
    ( Prt: 161; Srv: ' SNMP   '  ),     { Протокол Simple Netw. Management
Protocol }
    ( Prt: 169; Srv: ' SEND   '  )
    );
```

```
  const
  ICMP_ERROR_BASE = 11000;
  IcmpErr : array[1..22] of string =
  (
   ' IP_BUFFER_TOO_SMALL',' IP_DEST_NET_UNREACHABLE' , '
IP_DEST_HOST_UNREACHABLE' ,
   ' IP_PROTOCOL_UNREACHABLE' , ' IP_DEST_PORT_UNREACHABLE' , ' IP_NO_RESOURCES'
,
   ' IP_BAD_OPTION',' IP_HARDWARE_ERROR' , ' IP_PACKET_TOO_BIG' , '
IP_REQUEST_TIMED_OUT' ,
   ' IP_BAD_REQUEST',' IP_BAD_ROUTE' , ' IP_TTL_EXPIRED_TRANSIT' ,
   ' IP_TTL_EXPIRED_REASSEM',' IP_PARAMETER_PROBLEM' , ' IP_SOURCE_QUENCH' ,
   ' IP_OPTION_TOO_BIG' , ' IP_BAD_DESTINATION',' IP_ADDRESS_DELETED' ,
   ' IP_SPEC_MTU_CHANGE' , ' IP_MTU_CHANGE' , ' IP_UNLOAD'
  );


  ARPEntryType  : array[1..4] of string = ( ' Інший' , ' Несправний' ,
    ' Динамічний' , ' Статичний'
    );
  TCPConnState  :
    array[1..12] of string =
 ( ' CLOSED' , ' READ' , ' SYN_SENT' ,
    ' SYN_RCVD' , ' ESTABLISHED' , ' FIN_WAIT1' ,
    ' FIN_WAIT2' , ' WAIT' , ' CLOSING' ,
    ' LAST_ACK' , ' WAIT' , ' DELETE_TCB'    );

  TCPToAlgo     : array[1..4] of string =
    ( ' Const.Timeout' , ' MIL-STD-1778' ,
    ' Van Jacobson' , ' Other'  );

  IPForwTypes   : array[1..4] of string =
    ( ' other' , ' invalid' , ' local' , ' remote'  );

  IPForwProtos  : array[1..18] of string =
    ( ' OTHER' , ' LOCAL' , ' NETMGMT' , ' ICMP' , ' EGP' ,
    ' GGP' , ' HELO' , ' RIP' , ' IS_IS' , ' ES_IS' ,
    ' CISCO' , ' BBN' , ' OSPF' , ' BGP' , ' BOOTP' ,
    ' AUTO_STAT' , ' STATIC' , ' NOT_DOD'  );

type

// для IpHlpNetworkParams
  TNetworkParams = record
    HostName: string ;
    DomainName: string ;
    CurrentDnsServer: string ;
    DnsServerTot: integer ;
    DnsServerNames: array [0..9] of string ;
    NodeType: UINT;
    ScopeID: string ;
    EnableRouting: UINT;
    EnableProxy: UINT;
    EnableDNS: UINT;
  end;

  TIfRows = array of TMibIfRow ; // динамічний масив колонок

// для IpHlpAdaptersInfo
  TAdaptorInfo = record
    AdapterName: string ;
    Description: string ;
    MacAddress: string ;
    Index: DWORD;
    aType: UINT;
    DHCPEnabled: UINT;
    CurrIPAddress: string ;
    CurrIPMask: string ;
    IPAddressTot: integer ;
```

```
    IPAddressList: array of string ;
    IPMaskList: array of string ;
    GatewayTot: integer ;
    GatewayList: array of string ;
    DHCPTot: integer ;
    DHCPServer: array of string ;
    HaveWINS: BOOL;
    PrimWINSTot: integer ;
    PrimWINSServer: array of string ;
    SecWINSTot: integer ;
    SecWINSServer: array of string ;
    LeaseObtained: LongInt ;
    LeaseExpires: LongInt;
  end ;

  TAdaptorRows = array of TAdaptorInfo ;



function IpHlpAdaptersInfo(var AdpTot: integer;var AdpRows: TAdaptorRows):
integer ;
procedure Get_AdaptersInfo( List: TStrings );
function IpHlpNetworkParams (var NetworkParams: TNetworkParams): integer ;
procedure Get_NetworkParams( List: TStrings );
procedure Get_ARPTable( List: TStrings );
procedure Get_TCPTable( List: TStrings );
procedure Get_TCPStatistics( List: TStrings );
function IpHlpTCPStatistics (var TCPStats: TMibTCPStats): integer ;
procedure Get_UDPTable( List: TStrings );
procedure Get_UDPStatistics( List: TStrings );
function IpHlpUdpStatistics (UdpStats: TMibUDPStats): integer ;
procedure Get_IPAddrTable( List: TStrings );
procedure Get_IPForwardTable( List: TStrings );
procedure Get_IPStatistics( List: TStrings );
function IpHlpIPStatistics (var IPStats: TMibIPStats): integer ;
function Get_RTTAndHopCount( IPAddr: DWORD; MaxHops: Longint;
  var RTT: longint; var HopCount: longint ): integer;
procedure Get_ICMPStats( ICMPIn, ICMPOut: TStrings );
function IpHlpIfTable(var IfTot: integer; var IfRows: TIfRows): integer ;
procedure Get_IfTable( List: TStrings );
function IpHlpIfEntry(Index: integer; var IfRow: TMibIfRow): integer ;
procedure Get_RecentDestIPs( List: TStrings );

// утіліти перетворення
function MacAddr2Str( MacAddr: TMacAddress; size: integer ): string;
function IpAddr2Str( IPAddr: DWORD ): string;
function Str2IpAddr( IPStr: string ): DWORD;
function Port2Str( nwoPort: DWORD ): string;
function Port2Wrd( nwoPort: DWORD ): DWORD;
function Port2Svc( Port: DWORD ): string;
function ICMPErr2Str( ICMPErrCode: DWORD) : string;

implementation

var
  RecentIPs     : TStringList;


{ перетворення токена у рядок, потім рядок знищується }
function NextToken( var s: string; Separator: char ): string;
var
  Sep_Pos       : byte;
begin
  Result := ' ' ;
  if length( s ) > 0 then begin
    Sep_Pos := pos( Separator, s );
    if Sep_Pos > 0 then begin
      Result := copy( s, 1, Pred( Sep_Pos ) );
```

```
      Delete( s, 1, Sep_Pos );
    end
    else begin
      Result := s;
      s := ' ' ;
    end;
  end;
end;


{ пертворення MAC-адреси до ww-xx-yy-zz рядка }
function MacAddr2Str( MacAddr: TMacAddress; size: integer ): string;
var
  i             : integer;
begin
  if Size = 0 then
  begin
    Result := ' 00-00-00-00-00-00' ;
    EXIT;
  end
  else Result := ' ' ;
  //
  for i := 1 to Size do
    Result := Result + IntToHex( MacAddr[i], 2 ) + ' -' ;
  Delete( Result, Length( Result ), 1 );
end;


{ перетворення  IP-адреси  в мережний байт типу DWORD }
function IpAddr2Str( IPAddr: DWORD ): string;
var
  i             : integer;
begin
  Result := ' ' ;
  for i := 1 to 4 do
  begin
    Result := Result + Format( ' %3d.' , [IPAddr and $FF] );
    IPAddr := IPAddr shr 8;
  end;
  Delete( Result, Length( Result ), 1 );
end;


{ перетворення  крапкову десяткову IP-адресу  в мережний байт типу  DWORD}
function Str2IpAddr( IPStr: string ): DWORD;
var
  i             : integer;
  Num           : DWORD;
begin
  Result := 0;
  for i := 1 to 4 do
  try
    Num := ( StrToInt( NextToken( IPStr, ' .'  ) ) ) shl 24;
    Result := ( Result shr 8 ) or Num;
  except
    Result := 0;
  end;

end;


{ перетворення  номер порту в мережний байт типу  DWORD }
function Port2Wrd( nwoPort: DWORD ): DWORD;
begin
  Result := Swap( WORD( nwoPort ) );
end;


{ перетворення  номера порту в мережний байт типу  string }
```

```
function Port2Str( nwoPort: DWORD ): string;
begin
  Result := IntToStr( Port2Wrd( nwoPort ) );
end;


{ перетворення  відомого номера порту у сервіс ID }
function Port2Svc( Port: DWORD ): string;
var
  i             : integer;
begin
  Result := Format( ‘ %4d’ , [Port] ); // у випадку якщо порт не знайдено
  for i := Low( WellKnownPorts ) to High( WellKnownPorts ) do
    if Port = WellKnownPorts[i].Prt then
    begin
      Result := WellKnownPorts[i].Srv;
      BREAK;
    end;
end;


{ фіксуємо параметри мережі }

procedure Get_NetworkParams( List: TStrings );
var
    NetworkParams: TNetworkParams ;
    I, ErrorCode: integer ;
begin
    if not Assigned( List ) then EXIT;
    List.Clear;
    ErrorCode := IpHlpNetworkParams (NetworkParams) ;
    if ErrorCode <> 0 then
    begin
        List.Add (SysErrorMessage (ErrorCode));
        exit;
    end ;
    with NetworkParams do
    begin
        List.Add( ‘ HOSTNAME          : ‘  + HostName );
        List.Add( ‘ DOMAIN            : ‘  + DomainName );
        List.Add( ‘ NETBIOS NODE TYPE : ‘  + NETBIOSTypes[NodeType] );
        List.Add( ‘ DHCP SCOPE        : ‘  + ScopeID );
        List.Add( ‘ ROUTING ENABLED   : ‘  + IntToStr( EnableRouting ) );
        List.Add( ‘ PROXY   ENABLED   : ‘  + IntToStr( EnableProxy ) );
        List.Add( ‘ DNS     ENABLED   : ‘  + IntToStr( EnableDNS ) );
        if DnsServerTot <> 0 then
        begin
            for I := 0 to Pred (DnsServerTot) do
                List.Add( ‘ DNS SERVER ADDR   : ‘  + DnsServerNames [I] ) ;
        end ;
    end ;
end ;

//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */ /
function IpHlpNetworkParams (var NetworkParams: TNetworkParams): integer ;
var
  FixedInfo     : PTFixedInfo;         //
  InfoSize      : Longint;
  PDnsServer    : PTIP_ADDR_STRING ;   //
begin
    InfoSize := 0 ;   //
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetNetworkParams( Nil, @InfoSize );  //
    if result <> ERROR_BUFFER_OVERFLOW then exit ; //
    GetMem (FixedInfo, InfoSize) ;                       //
    try
    result := GetNetworkParams( FixedInfo, @InfoSize );   //
    if result <> ERROR_SUCCESS then exit ;
```

```
    NetworkParams.DnsServerTot := 0 ;
    with FixedInfo^ do
    begin
        NetworkParams.HostName := trim (HostName) ;
        NetworkParams.DomainName := trim (DomainName) ;
        NetworkParams.ScopeId := trim (ScopeID) ;
        NetworkParams.NodeType := NodeType ;
        NetworkParams.EnableRouting := EnableRouting ;
        NetworkParams.EnableProxy := EnableProxy ;
        NetworkParams.EnableDNS := EnableDNS ;
        NetworkParams.DnsServerNames [0] := DNSServerList.IPAddress ;  //
        if NetworkParams.DnsServerNames [0] <> ' '  then
                                        NetworkParams.DnsServerTot := 1 ;
        PDnsServer := DnsServerList.Next;
        while PDnsServer <> Nil do
        begin
            NetworkParams.DnsServerNames [NetworkParams.DnsServerTot] :=
                                        PDnsServer^.IPAddress ;  //
            inc (NetworkParams.DnsServerTot) ;
            if NetworkParams.DnsServerTot >=
                          Length (NetworkParams.DnsServerNames) then exit ;
            PDnsServer := PDnsServer.Next ;
        end;
    end ;
    finally
       FreeMem (FixedInfo) ;                  //
    end ;
end;

//---------------------------------------------------------------------------

function ICMPErr2Str( ICMPErrCode: DWORD) : string;
begin
   Result := ' UnknownError : '  + IntToStr( ICMPErrCode );
   dec( ICMPErrCode, ICMP_ERROR_BASE );
   if ICMPErrCode in [Low(ICMpErr)..High(ICMPErr)] then
     Result := ICMPErr[ ICMPErrCode];
end;


//---------------------------------------------------------------------------

// включаємо байти вводу/виводу для кожного адаптеру

function IpHlpIfTable(var IfTot: integer; var IfRows: TIfRows): integer ;
var
  I,
  TableSize   : integer;
  pBuf, pNext : PChar;
begin
  result := ERROR_NOT_SUPPORTED ;
  if NOT LoadIpHlp then exit ;
  SetLength (IfRows, 0) ;
  IfTot := 0 ; //
  TableSize := 0;
   //  перший виклик: беремо необхідний розмір пам' яті
  result := GetIfTable (Nil, @TableSize, false) ;  //
  if result <> ERROR_INSUFFICIENT_BUFFER then exit ;
  GetMem( pBuf, TableSize );
  try
      FillChar (pBuf^, TableSize, #0);  // очищуємо буфер, з W98 не потрібно

   // беремо показчик на таблицю
      result := GetIfTable (PTMibIfTable (pBuf), @TableSize, false) ;
      if result <> NO_ERROR then exit ;
      IfTot := PTMibIfTable (pBuf)^.dwNumEntries ;
      if IfTot = 0 then exit ;
      SetLength (IfRows, IfTot) ;
      pNext := pBuf + SizeOf(IfTot) ;
```

```
      for i := 0 to Pred (IfTot) do
      begin
         IfRows [i] := PTMibIfRow (pNext )^ ;
         inc (pNext, SizeOf (TMibIfRow)) ;
      end;
  finally
      FreeMem (pBuf) ;
  end ;
end;

procedure Get_IfTable( List: TStrings );
var
  IfRows        : TIfRows ;
  Error, I      : integer;
  NumEntries    : integer;
  sDescr, sIfName: string ;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  SetLength (IfRows, 0) ;
  Error := IpHlpIfTable (NumEntries, IfRows) ;
  if (Error <> 0) then
      List.Add( SysErrorMessage( GetLastError ) )
  else if NumEntries = 0 then
      List.Add( ' Даних немає.'  )
  else
  begin
      for I := 0 to Pred (NumEntries) do
      begin
          with IfRows [I] do
          begin
             if wszName [1] = #0 then
                 sIfName := ' '
             else
                 sIfName := WideCharToString (@wszName) ;  // перетворюємо
Unicode у string
             sIfName := trim (sIfName) ;
             sDescr := bDescr ;
             sDescr := trim (sDescr);
             List.Add (Format (
               ' %0.8x - %3d - %16s - %8d - %12d - %2d - %2d - %10d - %10d  -
%-s - %-s' ,
              [dwIndex, dwType, MacAddr2Str( TMacAddress( bPhysAddr ),
              dwPhysAddrLen ), dwMTU, dwSpeed, dwAdminStatus,
              dwOPerStatus, Int64 (dwInOctets), Int64 (dwOutOctets),  //
counters are 32-bit
              sIfName, sDescr] )  //  , додаємо введення/вивід
             );
          end;
      end ;
  end ;
  SetLength (IfRows, 0) ;  // звільняємо пам' ять
end ;

function IpHlpIfEntry(Index: integer; var IfRow: TMibIfRow): integer ;
begin
  result := ERROR_NOT_SUPPORTED ;
  if NOT LoadIpHlp then exit ;
  FillChar (IfRow, SizeOf (TMibIfRow), #0);  // очищуємо буфер, з W98 не
потрібно
  IfRow.dwIndex := Index ;
  result := GetIfEntry (@IfRow) ;
end ;

//---------------------------------------------------------------------------
{ інформація про мережні адаптери }

function IpHlpAdaptersInfo(var AdpTot: integer; var AdpRows: TAdaptorRows):
integer ;
```

```
var
  BufLen        : DWORD;
  AdapterInfo   : PTIP_ADAPTER_INFO;
  PIpAddr       : PTIP_ADDR_STRING;
  PBuf          : PCHAR ;
  I             : integer ;
begin
  SetLength (AdpRows, 4) ;
  AdpTot := 0 ;
  BufLen := 0 ;
  result := GetAdaptersInfo( Nil, @BufLen );
  if (result <> ERROR_INSUFFICIENT_BUFFER) and (result = NO_ERROR) then exit ;
  GetMem( pBuf, BufLen );
  try
      FillChar (pBuf^, BufLen, #0);  // очищуємо буфер
      result := GetAdaptersInfo( PTIP_ADAPTER_INFO (PBuf), @BufLen );
      if result = NO_ERROR then
      begin
         AdapterInfo := PTIP_ADAPTER_INFO (PBuf) ;
         while ( AdapterInfo <> nil ) do
         begin
            AdpRows [AdpTot].IPAddressTot := 0 ;
            SetLength (AdpRows [AdpTot].IPAddressList, 2) ;
            SetLength (AdpRows [AdpTot].IPMaskList, 2) ;
            AdpRows [AdpTot].GatewayTot := 0 ;
            SetLength (AdpRows [AdpTot].GatewayList, 2) ;
            AdpRows [AdpTot].DHCPTot := 0 ;
            SetLength (AdpRows [AdpTot].DHCPServer, 2) ;
            AdpRows [AdpTot].PrimWINSTot := 0 ;
            SetLength (AdpRows [AdpTot].PrimWINSServer, 2) ;
            AdpRows [AdpTot].SecWINSTot := 0 ;
            SetLength (AdpRows [AdpTot].SecWINSServer, 2) ;
            AdpRows [AdpTot].CurrIPAddress := NULL_IP;
            AdpRows [AdpTot].CurrIPMask := NULL_IP;
            AdpRows [AdpTot].AdapterName := Trim( string(
AdapterInfo^.AdapterName ) );
            AdpRows [AdpTot].Description := Trim( string(
AdapterInfo^.Description ) );
            AdpRows [AdpTot].MacAddress := MacAddr2Str( TMacAddress(
                                  AdapterInfo^.Address ),
AdapterInfo^.AddressLength ) ;
            AdpRows [AdpTot].Index := AdapterInfo^.Index ;
            AdpRows [AdpTot].aType := AdapterInfo^.aType ;
            AdpRows [AdpTot].DHCPEnabled := AdapterInfo^.DHCPEnabled ;
            if AdapterInfo^.CurrentIPAddress <> Nil then
            begin
                AdpRows [AdpTot].CurrIPAddress :=
AdapterInfo^.CurrentIPAddress.IpAddress ;
                AdpRows [AdpTot].CurrIPMask :=
AdapterInfo^.CurrentIPAddress.IpMask ;
            end ;

        // беремо список IP адрес та масок для IPAddressList
            I := 0 ;
            PIpAddr := @AdapterInfo^.IPAddressList ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].IPAddressList [I] := PIpAddr.IpAddress ;
                AdpRows [AdpTot].IPMaskList [I] := PIpAddr.IpMask ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].IPAddressList) <= I then
                begin
                     SetLength (AdpRows [AdpTot].IPAddressList, I * 2) ;
                     SetLength (AdpRows [AdpTot].IPMaskList, I * 2) ;
                end ;
            end ;
            AdpRows [AdpTot].IPAddressTot := I ;
```

```
        // беремо список IP адрес для GatewayList
            I := 0 ;
            PIpAddr := @AdapterInfo^.GatewayList ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].GatewayList [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].GatewayList) <= I then
                            SetLength (AdpRows [AdpTot].GatewayList, I * 2) ;
            end ;
            AdpRows [AdpTot].GatewayTot := I ;

        // беремо список IP адрес для GatewayList
            I := 0 ;
            PIpAddr := @AdapterInfo^.DHCPServer ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].DHCPServer [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].DHCPServer) <= I then
                            SetLength (AdpRows [AdpTot].DHCPServer, I * 2) ;
            end ;
            AdpRows [AdpTot].DHCPTot := I ;

        // беремо список IP адрес для PrimaryWINSServer
            I := 0 ;
            PIpAddr := @AdapterInfo^.PrimaryWINSServer ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].PrimWINSServer [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].PrimWINSServer) <= I then
                            SetLength (AdpRows [AdpTot].PrimWINSServer, I * 2)
;
            end ;
            AdpRows [AdpTot].PrimWINSTot := I ;

       // беремо список IP адрес для SecondaryWINSServer
            I := 0 ;
            PIpAddr := @AdapterInfo^.SecondaryWINSServer ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].SecWINSServer [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].SecWINSServer) <= I then
                            SetLength (AdpRows [AdpTot].SecWINSServer, I * 2) ;
            end ;
            AdpRows [AdpTot].SecWINSTot := I ;

            AdpRows [AdpTot].LeaseObtained := AdapterInfo^.LeaseObtained ;
            AdpRows [AdpTot].LeaseExpires := AdapterInfo^.LeaseExpires ;

            inc (AdpTot) ;
            if Length (AdpRows) <= AdpTot then
                        SetLength (AdpRows, AdpTot * 2) ;  // more memory
            AdapterInfo := AdapterInfo^.Next;
         end ;
         SetLength (AdpRows, AdpTot) ;
      end ;
  finally
      FreeMem( pBuf );
  end ;
end ;

procedure Get_AdaptersInfo( List: TStrings );
```

```
var
  AdpTot: integer;
  AdpRows: TAdaptorRows ;
  Error: DWORD ;
  I: integer ;
  //J: integer ;  jpt
  //S: string ;        id.
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  SetLength (AdpRows, 0) ;
  AdpTot := 0 ;
  Error := IpHlpAdaptersInfo(AdpTot, AdpRows) ;
  if (Error <> 0) then
      List.Add( SysErrorMessage( GetLastError ) )
  else if AdpTot = 0 then
      List.Add( ' Даних немає.'  )
  else
  begin
      for I := 0 to Pred (AdpTot) do
      begin
        with AdpRows [I] do
        begin
            //List.Add(AdapterName + ' -'  + Description ); // jpt : не корисне
            List.Add( Format(' %8.8x - %6s - %16s - %2d - %16s - %16s - %16s' ,
                [Index, AdaptTypes[aType], MacAddress, DHCPEnabled,
                GatewayList [0], DHCPServer [0], PrimWINSServer [0]] ) );
            {if IPAddressTot <> 0 then    // jpt : not useful
            begin
                S := ' '  ;
                for J := 0 to Pred (IPAddressTot) do
                        S := S + IPAddressList [J] + ' /'  + IPMaskList [J] + '
- '  ;
                List.Add(IntToStr (IPAddressTot) + '  IP Addresse(s): '  + S);
            end ;
            List.Add( '    '  ); }
        end ;
      end ;
  end ;
  SetLength (AdpRows, 0) ;
end ;

//---------------------------------------------------------------------------
{ зчитуємо кількість раундів, та час звертання до IP }
function Get_RTTAndHopCount( IPAddr: DWORD; MaxHops: Longint; var RTT: Longint;
  var HopCount: Longint ): integer;
begin
  if not GetRTTAndHopCount( IPAddr, @HopCount, MaxHops, @RTT ) then
  begin
    Result := GetLastError;
    RTT := -1; //
    HopCount := -1;
  end
  else
    Result := NO_ERROR;
end;

//---------------------------------------------------------------------------
{ ARP-таблиця – список відношень між віддаленими IP та віддаленими MAC-адресами.

}
procedure Get_ARPTable( List: TStrings );
var
  IPNetRow      : TMibIPNetRow;
  TableSize     : DWORD;
  NumEntries    : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
```

```
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  //  перший виклик: беремо довжину таблиці
  TableSize := 0;
  ErrorCode := GetIPNetTable( Nil, @TableSize, false );   //
  //
  if ErrorCode = ERROR_NO_DATA then
  begin
    List.Add( '  ARP-кеш пустий.'  );
    EXIT;
  end;
  // беремо таблицю
  GetMem( pBuf, TableSize );
  NumEntries := 0 ;
  try
  ErrorCode := GetIpNetTable( PTMIBIPNetTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMIBIPNetTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then //.
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        IPNetRow := PTMIBIPNetRow( PBuf )^;
        with IPNetRow do
          List.Add( Format( ' %8x - %12s - %16s - %10s' ,
                            [dwIndex, MacAddr2Str( bPhysAddr, dwPhysAddrLen ),
                            IPAddr2Str( dwAddr ), ARPEntryType[dwType]
                            ]));
        inc( pBuf, SizeOf( IPNetRow ) );
      end;
    end
    else
      List.Add( '  ARP-кеш пустий.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );

  // we _must_ restore pointer!
  finally
      dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( IPNetRow ) );
      FreeMem( pBuf );
  end ;
end;


//-----------------------------------------------------------------------------
procedure Get_TCPTable( List: TStrings );
var
  TCPRow        : TMIBTCPRow;
  i,
    NumEntries  : integer;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  DestIP        : string;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  RecentIPs.Clear;
  //  перший виклик : беремо розмір таблиці
  TableSize := 0;
  NumEntries := 0 ;
  ErrorCode := GetTCPTable(Nil, @TableSize, false );  //
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;
```

```
  // беремо розмір пам’ яти, який потрібно та здійснюємо виклик знову
  GetMem( pBuf, TableSize );
  // беремо таблицю
  ErrorCode := GetTCPTable( PTMIBTCPTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin

    NumEntries := PTMIBTCPTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        TCPRow := PTMIBTCPRow( pBuf )^; // беремо наступний запис
        with TCPRow do
        begin
          if dwRemoteAddr = 0 then
            dwRemotePort := 0;
          DestIP := IPAddr2Str( dwRemoteAddr );
          List.Add(
            Format( ' %15s : %-7s -> %15s : %-7s -> %-16s’ ,
            [IpAddr2Str( dwLocalAddr ),
            Port2Svc( Port2Wrd( dwLocalPort ) ),
              DestIP,
              Port2Svc( Port2Wrd( dwRemotePort ) ),
              TCPConnState[dwState]
              ] ) );
         //
            if (not ( dwRemoteAddr = 0 ))
            and ( RecentIps.IndexOf(DestIP) = -1 ) then
               RecentIPs.Add( DestIP );
        end;
        inc( pBuf, SizeOf( TMIBTCPRow ) );
      end;
    end;
  end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( TMibTCPRow ) );
  FreeMem( pBuf );
end;

//---------------------------------------------------------------------------
procedure Get_TCPStatistics( List: TStrings );
var
  TCPStats     : TMibTCPStats;
  ErrorCode    : DWORD;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  if NOT LoadIpHlp then exit ;
  ErrorCode := GetTCPStatistics( @TCPStats );
  if ErrorCode = NO_ERROR then
    with TCPStats do
    begin
      List.Add( ' Алгоритм повторної передачі: '  + TCPToAlgo[dwRTOAlgorithm] );
      List.Add( ' Мінімальний час виведення         : '  + IntToStr( dwRTOMin )
+ '  ms’  );
      List.Add( ' Максимальний час виведення         : '  + IntToStr( dwRTOMax )
+ '  ms’  );
      List.Add( ' Максимальне число підключень : '  + IntToStr( dwRTOAlgorithm )
);
      List.Add( ' Активні підключення             : '  + IntToStr( dwActiveOpens
) );
      List.Add( ' Пасивні підключення            : '  + IntToStr( dwPassiveOpens
) );
      List.Add( ' Невдала спроба відкриття     : '  + IntToStr( dwAttemptFails )
);
      List.Add( ' Відновлений зв'язок   : '  + IntToStr( dwEstabResets ) );
```

```
      List.Add( ' Поточний зв'язок .: '  + IntToStr( dwCurrEstab ) );
      List.Add( ' Отримано сегментів        : '  + IntToStr( dwInSegs ) );
      List.Add( ' Відправлено сегментів            : '  + IntToStr( dwOutSegs )
);
      List.Add( ' Ретрансльовано сегментів   : '  + IntToStr( dwReTransSegs ) );
      List.Add( ' Помилки входження          : '  + IntToStr( dwInErrs ) );
      List.Add( ' Скидання виведення          : '  + IntToStr( dwOutRsts ) );
      List.Add( ' Сукупні зв'язки   : '  + IntToStr( dwNumConns ) );
    end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
end;

function IpHlpTCPStatistics (var TCPStats: TMibTCPStats): integer ;
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetTCPStatistics( @TCPStats );
end;

//-----------------------------------------------------------------------------
procedure Get_UDPTable( List: TStrings );
var
  UDPRow        : TMIBUDPRow;
  i,
    NumEntries  : integer;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;

  //  перший виклик : беремо розмір таблиці
  TableSize := 0;
  NumEntries := 0 ;
  ErrorCode := GetUDPTable(Nil, @TableSize, false );
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  // беремо потрібний розмір пам'яті, викликаємо знову
  GetMem( pBuf, TableSize );

  // беремо таблицю
  ErrorCode := GetUDPTable( PTMIBUDPTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMIBUDPTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        UDPRow := PTMIBUDPRow( pBuf )^; // беремо наступний запис
        with UDPRow do
          List.Add( Format( ' %15s : %-6s' ,
            [IpAddr2Str( dwLocalAddr ),
            Port2Svc( Port2Wrd( dwLocalPort ) )
              ] ) );
        inc( pBuf, SizeOf( TMIBUDPRow ) );
      end;
    end
    else
      List.Add( ' Даних немає.'  );
  end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( TMibUDPRow ) );
  FreeMem( pBuf );
```

```
end;

//---------------------------------------------------------------------------
procedure Get_IPAddrTable( List: TStrings );
var
  IPAddrRow     : TMibIPAddrRow;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
  NumEntries    : DWORD;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  TableSize := 0; ;
  NumEntries := 0 ;
  //  перший виклик: беремо довжину таблиці
  ErrorCode := GetIpAddrTable(Nil, @TableSize, true );  //
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  GetMem( pBuf, TableSize );
  // беремо таблицю
  ErrorCode := GetIpAddrTable( PTMibIPAddrTable( pBuf ), @TableSize, true );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMibIPAddrTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) );
      for i := 1 to NumEntries do
      begin
        IPAddrRow := PTMIBIPAddrRow( pBuf )^;
        with IPAddrRow do
          List.Add( Format( ' %8.8x | %15s | %15s | %15s | %8.8d' ,
            [dwIndex,
            IPAddr2Str( dwAddr ),
              IPAddr2Str( dwMask ),
              IPAddr2Str( dwBCastAddr ),
              dwReasmSize
              ] ) );
        inc( pBuf, SizeOf( TMIBIPAddrRow ) );
      end;
    end
    else
      List.Add( ' Даних немає.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );

  // we must restore pointer!
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( IPAddrRow ) );
  FreeMem( pBuf );
end;

//---------------------------------------------------------------------------
{ беремо дані з таблиці маршрутизатора Cisco}
procedure Get_IPForwardTable( List: TStrings );
var
  IPForwRow     : TMibIPForwardRow;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
  NumEntries    : DWORD;
begin

  if not Assigned( List ) then EXIT;
  List.Clear;
```

```
  TableSize := 0;

  //  перший виклик: беремо довжину таблиці
  NumEntries := 0 ;
  ErrorCode := GetIpForwardTable(Nil, @TableSize, true);
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  // беремо таблицю
  GetMem( pBuf, TableSize );
  ErrorCode := GetIpForwardTable( PTMibIPForwardTable( pBuf ), @TableSize,
true);
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMibIPForwardTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) );
      for i := 1 to NumEntries do
      begin
        IPForwRow := PTMibIPForwardRow( pBuf )^;
        with IPForwRow do
        begin
          if (dwForwardType < 1)
          or (dwForwardType > 4) then
                   dwForwardType := 1 ;   //  , враховуємо погані значення
          List.Add( Format(
            ' %15s | %15s | %15s | %8.8x | %7s |   %5.5d |  %7s |   %2.2d' ,
            [IPAddr2Str( dwForwardDest ),
            IPAddr2Str( dwForwardMask ),
              IPAddr2Str( dwForwardNextHop ),
              dwForwardIFIndex,
              IPForwTypes[dwForwardType],
              dwForwardNextHopAS,
              IPForwProtos[dwForwardProto],
              dwForwardMetric1
              ] ) );
        end ;
        inc( pBuf, SizeOf( TMibIPForwardRow ) );
      end;
    end
    else
      List.Add( ' Даних немає.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( TMibIPForwardRow ) );
  FreeMem( pBuf );
end;

//-----------------------------------------------------------------------------
procedure Get_IPStatistics( List: TStrings );
var
  IPStats       : TMibIPStats;
  ErrorCode     : integer;
begin
  if not Assigned( List ) then EXIT;
  if NOT LoadIpHlp then exit ;
  ErrorCode := GetIPStatistics( @IPStats );
  if ErrorCode = NO_ERROR then
  begin
    List.Clear;
    with IPStats do
    begin
      if dwForwarding = 1 then
        List.add( ' Розблоковане пересилання      : '  + ' Так'  )
      else
        List.add( ' Розблоковане пересилання      : '  + ' Ні'  );
      List.add( ' Вбудований TTL             : '  + inttostr( dwDefaultTTL ) );
```

```
      List.add( ' Отримані датаграми      : '  + inttostr( dwInReceives ) );
      List.add( ' Помилка заголовку     (в)  : '  + inttostr( dwInHdrErrors ) );
      List.add( ' Адреса помилкова    (в)  : '  + inttostr( dwInAddrErrors ) );
      List.add( ' Датаграма переслана     : '  + inttostr( dwForwDatagrams ) );
//
      List.add( ' Невідомий протокол (в)  : '  + inttostr( dwInUnknownProtos )
);
      List.add( ' Датаграма відвергнута     : '  + inttostr( dwInDiscards ) );
      List.add( ' Датаграма встановлена     : '  + inttostr( dwInDelivers ) );
      List.add( ' Запит виведення            : '  + inttostr( dwOutRequests ) );
      List.add( ' Маршрутизація в маршрутизаторі Cisco відвергнута      : '  +
inttostr( dwRoutingDiscards ) );
      List.add( ' Немає маршрутів        (з)  : '  + inttostr( dwOutNoRoutes )
);
      List.add( ' Час виведення перебору      : '  + inttostr( dwReasmTimeOut )
);
      List.add( ' Запити перебору     : '  + inttostr( dwReasmReqds ) );
      List.add( ' Перебори здійснено : '  + inttostr( dwReasmOKs ) );
      List.add( ' Помилка перебору     : '  + inttostr( dwReasmFails ) );
      List.add( ' Фрагментація мережі успішна: '  + inttostr( dwFragOKs ) );
      List.add( ' Помилка фрагментації   : '  + inttostr( dwFragFails ) );
      List.add( ' Датаграми фрагментовано    : '  + inttostr( dwFRagCreates ) );
      List.add( ' Кількість інтерфейсів    : '  + inttostr( dwNumIf ) );
      List.add( ' Кількість IP-адрес  : '  + inttostr( dwNumAddr ) );
      List.add( ' маршрут в таблиці маршрутизації Cisco : '  + inttostr(
dwNumRoutes ) );
    end;
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
end;

function IpHlpIPStatistics (var IPStats: TMibIPStats): integer ;      //
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetIPStatistics( @IPStats );
end ;

//------------------------------------------------------------------------------
procedure Get_UdpStatistics( List: TStrings );
var
  UdpStats      : TMibUDPStats;
  ErrorCode     : integer;
begin
  if not Assigned( List ) then EXIT;
  ErrorCode := GetUDPStatistics( @UdpStats );
  if ErrorCode = NO_ERROR then
  begin
    List.Clear;
    with UDPStats do
    begin
      List.add( ' Datagrams (в)     : '  + inttostr( dwInDatagrams ) );
      List.add( ' Datagrams (з)    : '  + inttostr( dwOutDatagrams ) );
      List.add( ' No Ports           : '  + inttostr( dwNoPorts ) );
      List.add( ' Errors    (в)     : '  + inttostr( dwInErrors ) );
      List.add( ' UDP Listen Ports  : '  + inttostr( dwNumAddrs ) );
    end;
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
end;



//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
function IpHlpUdpStatistics (UdpStats: TMibUDPStats): integer ;     //
begin
```

```
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetUDPStatistics (@UdpStats) ;
end ;



//---------------------------------------------------------------------------

procedure Get_ICMPStats( ICMPIn, ICMPOut: TStrings );

var
  ErrorCode     : DWORD;
  ICMPStats     : PTMibICMPInfo;

begin
  if ( ICMPIn = nil ) or ( ICMPOut = nil ) then EXIT;
  ICMPIn.Clear;
  ICMPOut.Clear;
  New( ICMPStats );
  ErrorCode := GetICMPStatistics( ICMPStats );
  if ErrorCode = NO_ERROR then
  begin
    with ICMPStats.InStats do
    begin
      ICMPIn.Add( ' Отримано повідослень    : ' + IntToStr( dwMsgs ) );
      ICMPIn.Add( ' Помилки ICMP             : ' + IntToStr( dwErrors ) );
      ICMPIn.Add( ' Розташування    : ' + IntToStr( dwDestUnreachs ) );
      ICMPIn.Add( ' Час перевищено        : ' + IntToStr( dwTimeEcxcds ) );
      ICMPIn.Add( ' Проблеми параметрів    : ' + IntToStr( dwParmProbs ) );
      ICMPIn.Add( ' Джерело відключено       : ' + IntToStr( dwSrcQuenchs ) );
      ICMPIn.Add( ' Перенаправлення          : ' + IntToStr( dwRedirects ) );
      ICMPIn.Add( ' Ехо запит        : ' + IntToStr( dwEchos ) );
      ICMPIn.Add( ' Ехо повтор         : ' + IntToStr( dwEchoReps ) );
      ICMPIn.Add( ' Запит мітки часу   : ' + IntToStr( dwTimeStamps ) );
      ICMPIn.Add( ' Повтор мітки часу    : ' + IntToStr( dwTimeStampReps ) );
      ICMPIn.Add( ' Запит маски адреси : ' + IntToStr( dwAddrMasks ) );
      ICMPIn.Add( ' Повтор маски адреси   : ' + IntToStr( dwAddrReps ) );
    end;
     //


    with ICMPStats.OutStats do


 begin
      ICMPOut.Add( ' Повідомлення відправлено       : ' + IntToStr( dwMsgs )
);
      ICMPOut.Add( 'Помилки ICMP            : ' + IntToStr( dwErrors ) );
      ICMPOut.Add( 'Розташування    : ' + IntToStr( dwDestUnreachs ) );
      ICMPOut.Add( 'Час перевищено        : ' + IntToStr( dwTimeEcxcds ) );
      ICMPOut.Add( 'Проблеми параметрів      : ' + IntToStr( dwParmProbs ) );
      ICMPOut.Add( 'Джерело відключено       : ' + IntToStr( dwSrcQuenchs ) );
      ICMPOut.Add( 'Перенаправлення          : ' + IntToStr( dwRedirects ) );
      ICMPOut.Add( 'Ехо запит        : ' + IntToStr( dwEchos ) );
      ICMPOut.Add( 'Ехо повтор         : ' + IntToStr( dwEchoReps ) );
      ICMPOut.Add( 'Запит мітки часу   : ' + IntToStr( dwTimeStamps ) );
      ICMPOut.Add( 'Повтор мітки часу    : ' + IntToStr( dwTimeStampReps ) );
      ICMPOut.Add( 'Запит маски адреси: ' + IntToStr( dwAddrMasks ) );
      ICMPOut.Add( 'Повтор маски адреси   : ' + IntToStr( dwAddrReps ) );
    end;
  end
  else
    IcmpIn.Add( SysErrorMessage( ErrorCode ) );
  Dispose( ICMPStats );
end;


//---------------------------------------------------------------------------
```

```
procedure Get_RecentDestIPs( List: TStrings );
begin
  if Assigned( List ) then
    List.Assign( RecentIPs )
end;

initialization

  RecentIPs := TStringList.Create;

finalization

  RecentIPs.Free;

end.
```

Файл NetConst.pas – ініціалізація констант

```
unit NetConst;



interface

resourcestring

  SShellLinkReadError = ' Помилка  читання' ;
  SShellLinkWriteError = ' Помилка запису' ;
  SShellLinkLoadError = ' Не можу завантажити %s' ;
  SShellLinkSaveError = ' Не можу зберегти %s' ;
  SShellLinkCreateError = ' Не можу ініціалізувати інтерфейс' ;
  SDynArrayIndexError = ' У масиві %s немає елемента з таким індексом (%d)' ;
  SDynArrayCountError = ' Перевищена довжина масиву (%d)' ;
  SSharedMemoryError = ' Не можу створити файл' ;
  SCannotInitTimer = ' Не можу ініціалізувати таймер' ;
  SPrinterIndexError = ' Принтер не доступний (%d)' ;
  SIndicesOutOfRange = ' Недопустимий індекс матриці [%d: %d]' ;
  SRowIndexOutOfRange = ' Недопустиме занчення рядка матриці (%d)' ;
  SColIndexOutOfRange = ' Недопустиме занчення стовбчика матриці (%d)' ;
  SNoAdminRights = ' У Вас немає прав адміністратора' ;

  SFileError = ' Помилка %s файл %s%s' ;
  SFileReading = ' читання' ;
  SFileWriting = ' запис' ;

 SFileError002 = '  - файл не знайдено' ;
  SFileError003 = '  - шлях не знайдено' ;
  SFileError004 = '  - не можу відкрити файл' ;
  SFileError005 = '  - немає доступу' ;
  SFileError014 = '  - не достатньо пам"яті' ;
  SFileError015 = '  - не можу знайти драйвер' ;
  SFileError017 = '  - не можу перемістити файл' ;
  SFileError019 = '  - носій захищений від запису' ;
  SFileError020 = '  - не можу знайти пристрій' ;
  SFileError021 = '  - пристрій не відкривається для читання' ;
  SFileError022 = '  - пристрій не може розпізнати команду' ;
  SFileError025 = '  - вказана область не знайдена' ;
  SFileError026 = '  - пристрій недоступний' ;
  SFileError027 = '  - сектор не знайдено' ;
  SFileError029 = '  - помилка запису на пристрій' ;
  SFileError030 = '  - помилка читання з пристрою' ;
  SFileError032 = '  - файл використовується іншою програмою' ;
  SFileError036 = '  - занадто багато відкритих файлів' ;
  SFileError038 = '  - досягнутий кінець файлу' ;
  SFileError039 = '  - диск переповнений' ;
  SFileError050 = '  - запит не підтримується' ;
  SFileError051 = '  - віддалений комп' ютер недоступний' ;
  SFileError052 = '  - у мережі знайдені ідентичні імена' ;
  SFileError053 = '  - мережний шлях не знайдено' ;
  SFileError054 = '  - мережа зайнята' ;
  SFileError055 = '  - ресурс мережі або пристрій недоступний' ;
  SFileError057 = '  - апаратна помилка в мережному адаптері' ;
  SFileError058 = '  - сервер не в змозі виконувати дану дію' ;
  SFileError059 = '  - помилка у мережі' ;
  SFileError064 = '  - недоступне мережне ім"я' ;
  SFileError065 = '  - немає доступу до мережі' ;
  SFileError066 = '  - невірно вказатий тип мережного ресурсу' ;
  SFileError067 = '  - не знайдене вказане мережне ім"я' ;
  SFileError070 = '  - відключейний сервер мережі' ;
  SFileError082 = '  - не можу створити файл чи каталог' ;
  SFileError112 = '  - не достатньо вільного місця на диску' ;
  SFileError123 = '  - в імені файлу вказано недопустимий символ' ;
  SFileError161 = '  - неправильно вказано шдях' ;
```

```
  SFileError183 = ‘  - файл не існує’ ;

  SCannotSetSize = ‘ Не можу змінити розмір файлу’ ;

  SUnableToCompress = ‘ Не можу заархівувати дані’ ;
  SUnableToDecompress = ‘ Не можу розархівувати дані’ ;

  SCannotFindNetwork = ‘ Не можу знайти  мережу’ ;

implementation

end.
```

Файл About.pas - довідка

```
unit About;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls;

type
  TForm2 = class(TForm)
    Image1: TImage;
    Memo1: TMemo;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.Button1Click(Sender: TObject);
begin
Form2.Close;
end;

end.
```