МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Сільман Богдан Олегович

Програмне забезпечення системи моніторингу SDN

Спеціальність: 123 «Комп’ютерна інженерія»

Освітній ступінь: бакалавр

Науковий керівник:

Коваленко Олександр Володимирович_____

(підпис) (дата)

доктор технічних наук, доцент

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« 11 » січня 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Сільману Богдану Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи моніторингу SDN*
керівник роботи *Коваленко Олександр Володимирович, докт. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу № 204-02 від 28.12.2020 року
2. Строк подання студентом роботи до захисту *22.05.2021 р.*
3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи моніторингу SDN*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

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

6. Дата видачі завдання « 11 » січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2021 р.	
3.	Розробка моделі компонента	20.03.2021 р.	
4.	Розробка структур даних	25.03.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2021 р.	
6.	Програмування алгоритмів	10.04.2021 р.	
7.	Оформлення ПЗ	17.04.2021 р.	
8.	Попередній захист роботи	14.05.2021 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сільман Б.О. Програмне забезпечення системи моніторингу SDN. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній бакалаврській розроблено програмне забезпечення, яке призначено для системи моніторингу SDN.

Метою розробки є програмне забезпечення системи моніторингу SDN.

Результат роботи – програмна реалізація системи моніторингу SDN.

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 XP/Vista/7/8/10.

Програму розроблено в середовищі Embarcadero Delphi.

Ключові слова: комп'ютерна інженерія, моніторинг, SDN

ABSTRACT

Silman B.O. SDN monitoring system software. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this bachelor's qualification the software which is intended for SDN monitoring system is developed.

The purpose of the development is the software of the SDN monitoring system.

The result is a software implementation of the SDN monitoring system.

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

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in the Embarcadero Delphi environment.

Keywords: computer engineering, monitoring, SDN

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ВМ	–	Віртуальна машина
API	–	Прикладні програмні інтерфейси
BYOC	–	Прив'язка різних гаджетів до корпоративної мережі
BYOD	–	Використання співробітниками своїх особистих пристроїв
DSR	–	Direct Server Return
IT	–	Інформаційні технології
NC	–	Network Controller
OpenFlow	–	Протокол комунікації контролерів і комутаторів
RAS	–	Служби дистанційного доступу
RDMA	–	Дистанційний доступ до пам'яті
SDN	–	Програмно-обумовлені мережі
VLAN	–	Віртуальні локальні мережі

ВСТУП

Актуальність теми. Тотальна мобільність, хмарні технології й консьюмеризація IT – це основні тренди, які будуть визначати майбутнє IT уже сьогодні.

Можливість працювати буквально де завгодно суттєво підвищує продуктивність і мотивує свідомих співробітників на багато чого. Сьогодні користувачі можуть одержувати доступ до даних і застосунків з будь-якого місця: в офісі, будинку, в аеропорті, у готелі. Причому використовувати для цього найрізноманітніші пристрої: ноутбук, планшет, смартфон і, використовуючи будь-які технології провідні мереж Ethernet, Wifi або 3G/4G. Корпоративні додатки переносяться з фізичних серверів на віртуальні машини або навіть у хмари.

Хмарні технології в корені міняють основну модель витрат компаній, перетворюючи частину витрат на створення IT-інфраструктуру з капітальних витрат в операційні й допомагають гнучко нарощувати додаткові ресурси або потужності на вимогу або в міру росту бізнесу.

Робочі столи співробітників стають віртуальними, перестаючи бути прив'язаними до чорних ящиків конкретних комп'ютерів. Набагато ефективніше вирішуються питання ліцензування програмного забезпечення і його своєчасного відновлення. Однак при цьому, хмарні технології й створюють певні проблеми, суттєво ускладнюючи життя IT-фахівцям у тих випадках, коли потрібно зрозуміти, із чим зв'язана низька продуктивність сервісів і на чиїй стороні виникла проблема.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи моніторингу SDN.

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

– Огляд існуючих систем моніторингу SDN.

– Дослідження системи моніторингу SDN.

– Програмна реалізація системи моніторингу SDN.

Практична цінність отриманих результатів полягає в т ому, що розроблені алгоритми дозволяють успішно вирішувати задачі моніторингу SDN.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи моніторингу SDN, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Термін «консьюмеризація IT» в основному означає тенденцію використання співробітниками своїх особистих пристроїв (BYOD) для виконання робочих функцій. Для цього відділ IT повинен розв'язати завдання ефективної прив'язки різних гаджетів до корпоративної мережі, у тому числі – і особистих ноутбуків (BYOC). Крім того, при впровадженні BYOD і/або BYOC служба IT буде змушено здійснювати підтримку не тільки корпоративних пристроїв, але й особистих пристроїв. На більшість гаджетів установлені в тому числі й сторонні додатки, які автоматично обновляються у фоні при підключенні до будь-якої мережі, і в такий спосіб будуть конкурувати за IT ресурси нарівні з корпоративними пристроями. На жаль, на даному етапі не всі сучасні застосунки по моніторингові IT-інфраструктури готові надати реальну допомогу й підтримку IT-фахівцям у забезпеченні питання безпеки й продуктивності такого спільного використання.

Ефективне впровадження всіх нових тенденцій у рамках існуючої концепції побудови мереж – завдання не із простих. Адже по суті, мережі протягом останніх років концептуально майже не змінювалися – росли швидкості й з'являлися нові протоколи, але принципи керування й передачі трафіку практично не мінялися (якщо не сказати відверто – ускладнювалися й далеко не завжди це ускладнення приводило до позитивних результатів). Типовий підхід в організації середньостатичної сучасної корпоративної мережі: кожний елемент налаштовується й адмініструється відокремлено, якщо виникає проблема із продуктивністю, то простий пристрій міняється на більш прокачаний. Підтримувати нові технології, враховуючи нереальну швидкість їх появи й розробки, з використанням старих принципів стало практично неможливо.

Наприклад, якщо ми розв'яжемо запустити нову віртуальну машину сьогодні й перенести на неї додаток, то переналаштування списків контролю доступу на всіх пристроях корпоративної мережі може зайняти кілька днів, що неприпустимо.

1.2 Область застосування

Програмно-обумовлені мережі

Концепція програмно-обумовлених мереж (SDN) ґрунтовно міняє принципи функціонування мереж і їх керування. У швидко мінливому сучасному світі саме мережі передачі даних були названі «тонкою ланкою», яке обмежує ріст продуктивності застосунків у міру росту кількості мобільних користувачів, масштабування віртуальних середовищ, формування кластерів для Великих Даних. В SDN мережах завдання комутації трафіку й завдання керування строго розділені. Уся логіка керування централізується й передаються контролеру. Комутатор у концепції SDN – досить примітивний пристрій, який відповідає тільки за перемикання пакетів на підставі дуже простих правил. Контролер SDN управляє всіма комутаторами в мережі й програмує кожний з них для правильної передачі трафіку. Централізація логіки керування дозволяє програмувати мережа як єдине ціле й спростити операційну модель більших корпоративних мереж, які занадто статичні на даний момент і не відповідають сучасному бізнесу, із властивими йому мобільністю користувачів/пристроїв/застосунків, розподілом застосунків між віртуальними машинами й інтенсивним обміном даними.

Незалежне від виробника керування всіма пристроями з єдиного центру суттєво спрощує конфігурацію й експлуатацію мережі. Завдяки контролеру з розширеними API інтерфейсами, уся мережа стає подібною одному великому логічному комутатору. Протокол OpenFlow – один із самих універсальних протоколів комунікації контролерів і комутаторів на сьогоднішній день надає стандартний підхід до програмування таблиць комутації, у яких основним об'єктом є потік даних. Однак у той час як OpenFlow дозволяє контролеру

програмувати комутатори, він не визначає, як контролеру реагувати й відповідати на виклики, пов'язані із хмарами, BYOD, BYOC, віртуалізацією. Застосунок будь-яких проблеми із продуктивністю мережі, та й просто її роботою, покладені на контролер і додатка, які на ньому запущені. Саме набір таких застосунків і відрізняє різні впровадження SDN, а також застосунки різних виробників. Тому під парасолькою SDN на даний момент розвиваються інші технології, які поки прив'язані до того або іншому виробника.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи моніторингу SDN, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Network Controller: програмно-обумовлені мережі в Windows Server 2016

Технологія Microsoft, з області програмно-обумовлених мереж (SDN). Network Controller – це служба керування мережною інфраструктурою в Windows Server 2016.

VLAN. Перші віртуальні мережі з'явилися в 1998 році. Це були локальні віртуальні мережі VLAN, що працюють по протоколу IEEE 802.1Q. Технологія дозволяє розділяти одну L2-мережу на безліч логічних L2-мереж. Але VLAN має обмеження, яке виявилося істотним з ростом кількості мереж: максимальна кількість з'єднань на одній L2-мережі – 4096. Для його подолання виробники почали розробляти нові протоколи з більшою масштабованістю, наприклад: IEEE 802.1ad і IEEE 802.1ah. Ми не будемо зупинятися на них докладно й підемо далі.

VXLAN, STT і NV GRE. В 2011 році компанії VMware, Arista і Cisco випускають протокол VXLAN, що дозволяє створювати до 16 мільйонів логічних L2-сегментів в одній мережі. Паралельно з ними Microsoft створює протокол NVGRE схожої специфікації, який починає розбудовуватися в Windows Server 2012. Компанія Nicira пропонує протокол STT. Питання з обмеженою масштабованістю вирішений, можна створювати як завгодно багато мереж. З ростом мереж інфраструктура стає складною в керуванні, і з'являється необхідність централізованої консолі для налаштування віртуального й фізичного устаткування. Так виникає концепція програмно-обумовлених мереж (SDN) із централізованим керуванням мережною інфраструктурою.

SDN. Підхід до керування мережею за допомогою уніфікованих програмних засобів. Однієї їхніх перших технологію SDN представила компанія Nicira. Після покупки Nicira компанія VMware створює продукт VMware NSX, використовуючи протокол VXLAN.

В 2013 році Microsoft пропонує свою версію програмно-обумовленої мережі в редакції Windows Server 2012R2 на базі протоколу NVGRE. У протоколу NVGRE було кілька недоліків:

– керування віртуальними мережами в Windows Server 2012R2 здійснювалося тільки через Virtual Machine Manager (VMM). Він був крапкою зберігання всіх конфігурацій віртуальної інфраструктури і єдиною крапкою відмови;

– виробники устаткування як основного протоколу використовували VXLAN, і лише деякі впровадили підтримку NVGRE.

У редакції Windows Server 2016 компанія Microsoft представила нову реалізацію програмно-обумовлених мереж уже на базі протоколу VXLAN – Network Controller (NC). Доступна дана служба тільки в редакції Windows Server 2016 Datacenter. Давайте розглянемо особливості Network Controller.

Windows Server 2016: Network Controller

Network Controller – єдина крапка керування й моніторингу для всіх фізичних і віртуальних мереж домена. За допомогою його налаштовується IP-підмережі, VLAN, маршрутизатори й міжмережеві екрани. NC зберігає інформацію про топологію мережі, балансує трафік і задає правила NAT.

Сервіси:

– Virtual Network Management – служба керування віртуалізованими мережами.

– Software Load Balancer Management – служба балансування трафіку й правил NAT.

– Firewall Management – служба налаштування міжмережевих екранів і аркушів контролю доступу до ВМ.

– RAS Gateway Management – служба організації VPN-тунелів.

– iDNS – служба створення віртуальних DNS-серверів.

Мережі:

– Backend Network (provider addresses, PA) – мережа із провайдерскими адресами. Тут створюються віртуальні тунелі й проходить трафік.

– Management Network – мережа керування, через неї проходить трафік між компонентами SDN.

– VM network (customer addresses, CA) – віртуалізована мережа, до якої підключені віртуальні машини.

– Transit Network – транзитна мережа. По ній іде обмін трафіком між BGP і SLB Multiplexer/Gateway. По транзитній мережі публікуються маршрути на BGP.

Служби на хості:

– Azure VFP Switch Extension – розширення свитча Hyper-V. Додає віртуальному свитчу функціонал L 3-комутатора. Містить у собі distributed load balancer(DLB) і Distributed firewall (DFW).

– Distributed load balancer – балансувальникик трафіку.

– Distributed firewall – міжмережевий екран, відповідає за виконання правил доступу до ВМ.

– Distributed router (vSwitch) – віртуальний маршрутизатор.

– NC Host Agent – одержує від NC інформацію про віртуальні мережі й правилах Firewall.

– SLB Host Agent – одержує від NC правила балансування.

Службові віртуальні машини:

– SLB multiplexer (MUX) – віртуальні машини Windows Server з роллю Software Load Balancing. На них утримується інформація про правила балансування трафіку. MUX публікують маршрути на BGP і обробляють вхідні пакети.

– Gateway – віртуальний шлюз.

У цілому принцип архітектури Network Controller такий: вузли контролера розміщаються на окремих ВМ, а служби маршрутизації, балансування трафіку й міжмережевого екрана – на серверах віртуалізації.

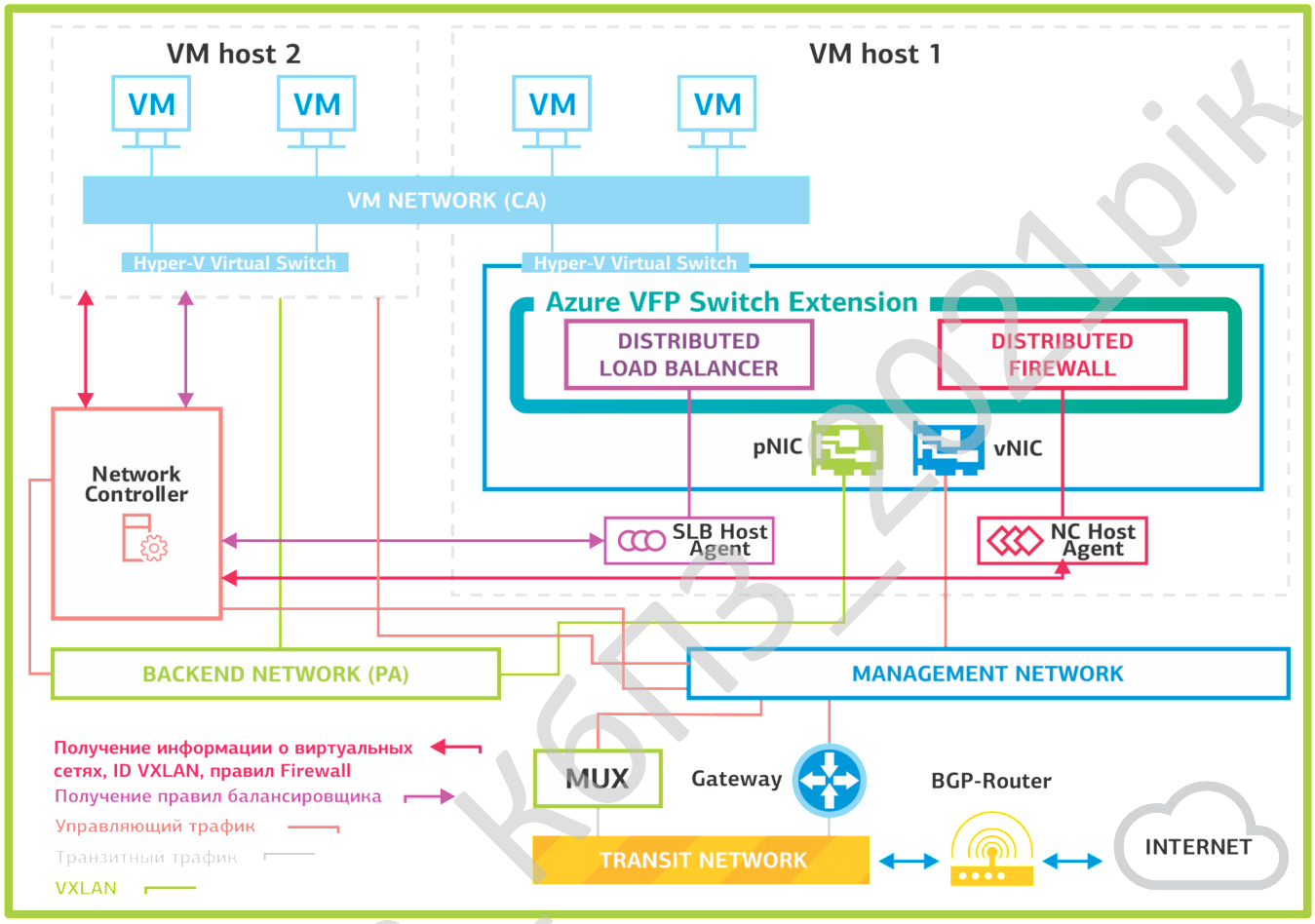

Рисунок 2.1 – Архітектура Network Controller

Network Controller створений на базі програмного комутатора Open vSwitch. Цей комутатор також використовується в програмно-обумовлених мережах VMware NSX і Openstack Neutron. Він підтримує протокол керування OVSDB (Open vSwitch Database), відповідальний за обмін інформацією між мережним устаткуванням.

Розберемо, як працює Network Controller. Сервери, віртуальні машини, віртуальні комутатори гіпервізорів реєструються в NC. Після реєстрації й налаштування сервери встановлюють з'єднання з Network Controller і одержують

від нього всю необхідну інформацію про мережі віртуальних машин, правилах балансування й VPN-тунелях.

Віртуальні мережі на хостах підключені до віртуального комутатора Hyper-V з розширенням Azure VFP Switch Extension. Через нього відбувається керування віртуальними портами, до яких підключені ВМ:

– включення й відключення портів;

– керування смугою пропущення в обидва боки;

– призначення пріоритетів пакетам по класах (COS);

– призначення цілей: ведення статистики й керування ACL на портах віртуального комутатора.

Служби Network Controller

Розберемо докладно, які служби входять в Network Controller і чому вони управляють.

Virtual Network Management. Це служба створення віртуальних мереж і керування ними. Вона управляє віртуальними комутаторами Hyper-V і віртуальними мережними адаптерами на віртуальних машинах. Virtual Network Management містить налаштування віртуальної мережі, які застосовуються іншими службами Network Controller.

Firewall Management. Ця служба спрощує організацію міжмережевого екранування: не потрібно набудовувати міжмережеві екрани на кожної ВМ вручну й запам'ятовувати конфігурації. Network Controller зберігає всі налаштування Access Control Lists (ACL) для віртуальних машин і мереж. Distributed Firewall на хостах запитує в нього інформацію й застосовує правила до потрібних мереж і віртуальним машинам. Налаштування на стороні віртуальних машин не потрібно.

Нижче схема принципу роботи міжмережевого екранування в Network Controller. На ній Northbound – інтерфейс, через який проводиться керування Network Controller з використанням REST API. Southbound служить для зв'язку з мережними пристроями, SLB-мультиплексорами, шлюзами й серверами, для

мережного виявлення й інших служб. Шлюзи визначають, для якої віртуальної мережі потрібно побудувати тунелі. Потім вони пересилають пришедші пакети по тунелю на сервери з віртуальними машинами, підключеними до потрібної віртуальної мережі.

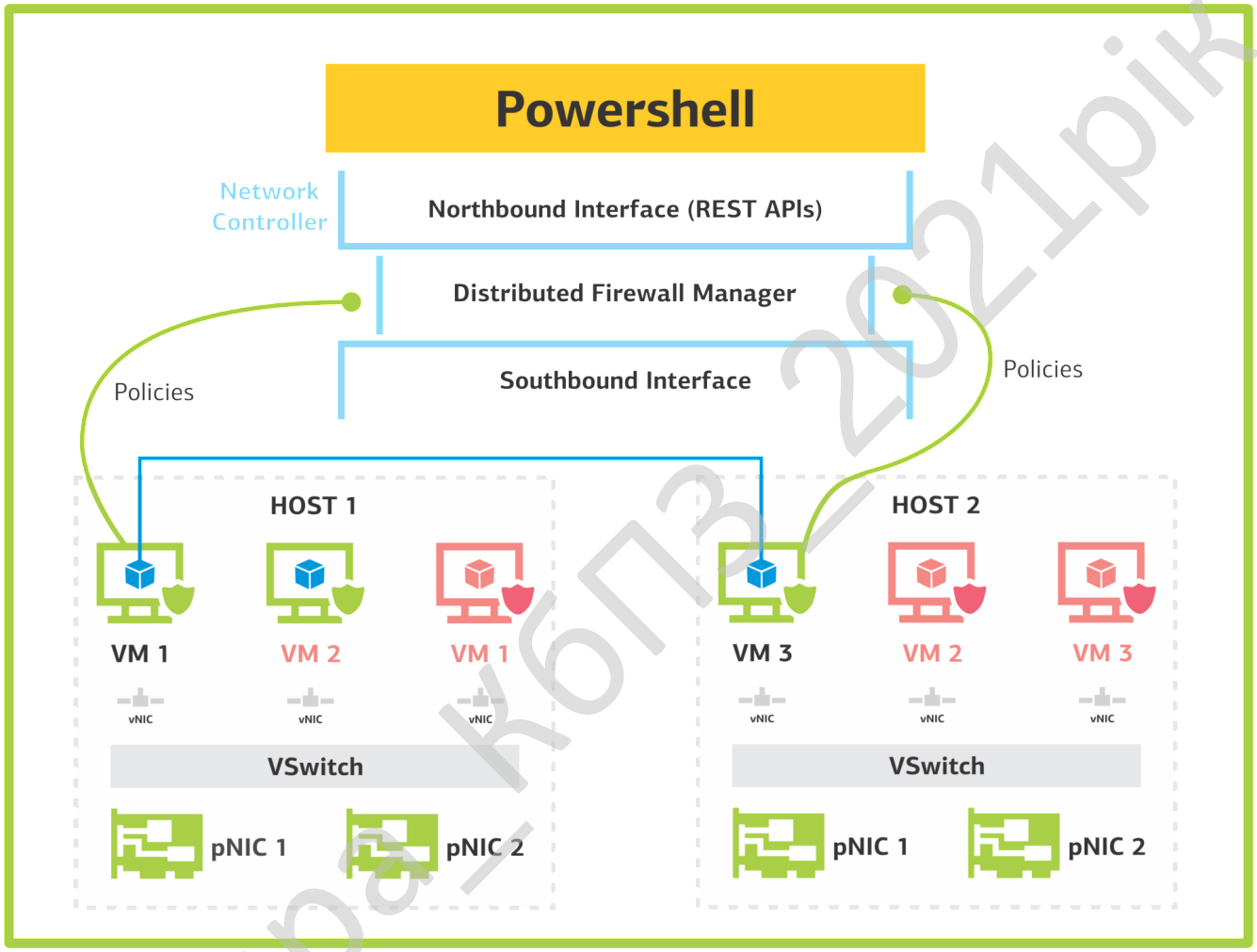

Рисунок 2.2 – Пристрій служби міжмережевого екранування в Network Controller

Software Load Balancer Management. Служба працює на рівні віртуалізованих мереж і балансує трафік на рівні L4. Балансування трафіку відбувається за допомогою службових віртуальних машин SLB Multiplexer (MUX) і BGP-маршрутизаторів. На один NC можна зробити 8 MUX. Від Network

Controller MUX одержують правила маршрутизації. Мультиплексори анонсують маршрути /32 для кожного VIP через BGP-маршрутизатори.

Приклад

1. На BGP-маршрутизатор приходить пакет для певного віртуального IP-адреси.

2. BGP-маршрутизатор перевіряє, який у пакета наступний вузол. У цьому випадку це MUX.

3. По таблицях правил балансування, отриманим від Network Controller, MUX визначає призначення пакета: віртуалізовану мережу, хост і віртуальну машину.

4. Далі MUX формує VXLAN пакет і відправляє його хосту в мережу Backend.

5. Хост ухвалює пакет і передає це віртуальному комутатору (vSwitch), який визначає віртуальну машину для одержання пакета.

6. Пакет декапсулюється, аналізується й відправляється віртуальній машині.

7. Віртуальна машина обробляє пакет. Адреса відправника не змінюється після проходження пакета через MUX. Тому віртуальна машина не бачить за ним MUX балансувальника й зчитує, що пакет прийшов прямо з інтернету. Таке балансування називається прозорої (transparent load balancing).

8. ВМ посилає відповідь.

9. Віртуальний комутатор на хосте визначає, що це відповідь на пакет, пришедший з балансувальника, і відправляє його назад в Інтернет. Причому не по тій же ланцюжкові, а переписує адреса відправника на IP балансувальника й відправляє його відразу в Інтернет. Даний підхід називається Direct Server Return (DSR). Це значно прискорює процес обробки пакетів.

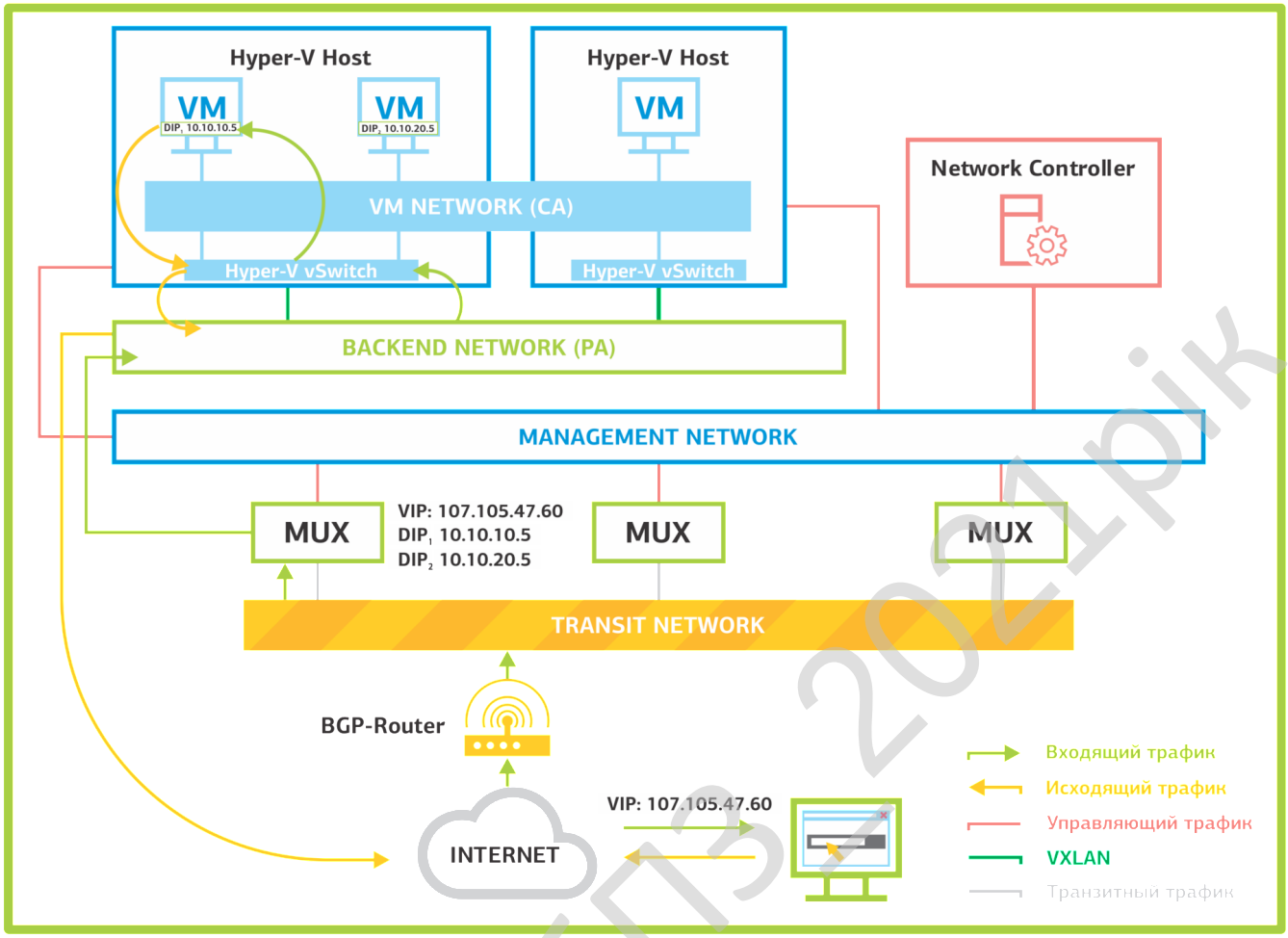

Рисунок 2.3 – Балансування трафіку в Network Controller

В Windows Server 2016 правила NAT тепер також задаються через Software Load Balancer Management. Network Controller знає, для якої мережі організований NAT. MUX містить інформацію про хостах і віртуальних машинах, підключених до мережі, для якої організує NAT.

NAT в Network Controller реалізований шляхом балансування трафіку для групи з одного хоста. На даний момент підтримується балансування тільки TCP- і Udp-трафіку.

RAS Gateway Manager. Ця служба створення VPN-тунелів для зв'язку хмарної інфраструктури з фізичної. В Network Controller можна будувати VPN будь-яким зручним способом:

– через IPsec, який тепер працює й з IKEv1, і з IKEv2;

– створювати SSTP-тунелі;

– створювати GRE-тунелі.

Кінцевою крапкою тунелю виступає віртуальний шлюз Gateway. Він організує тунель, інкапсулює трафік в VXLAN-пакети й пересилає його хостам.

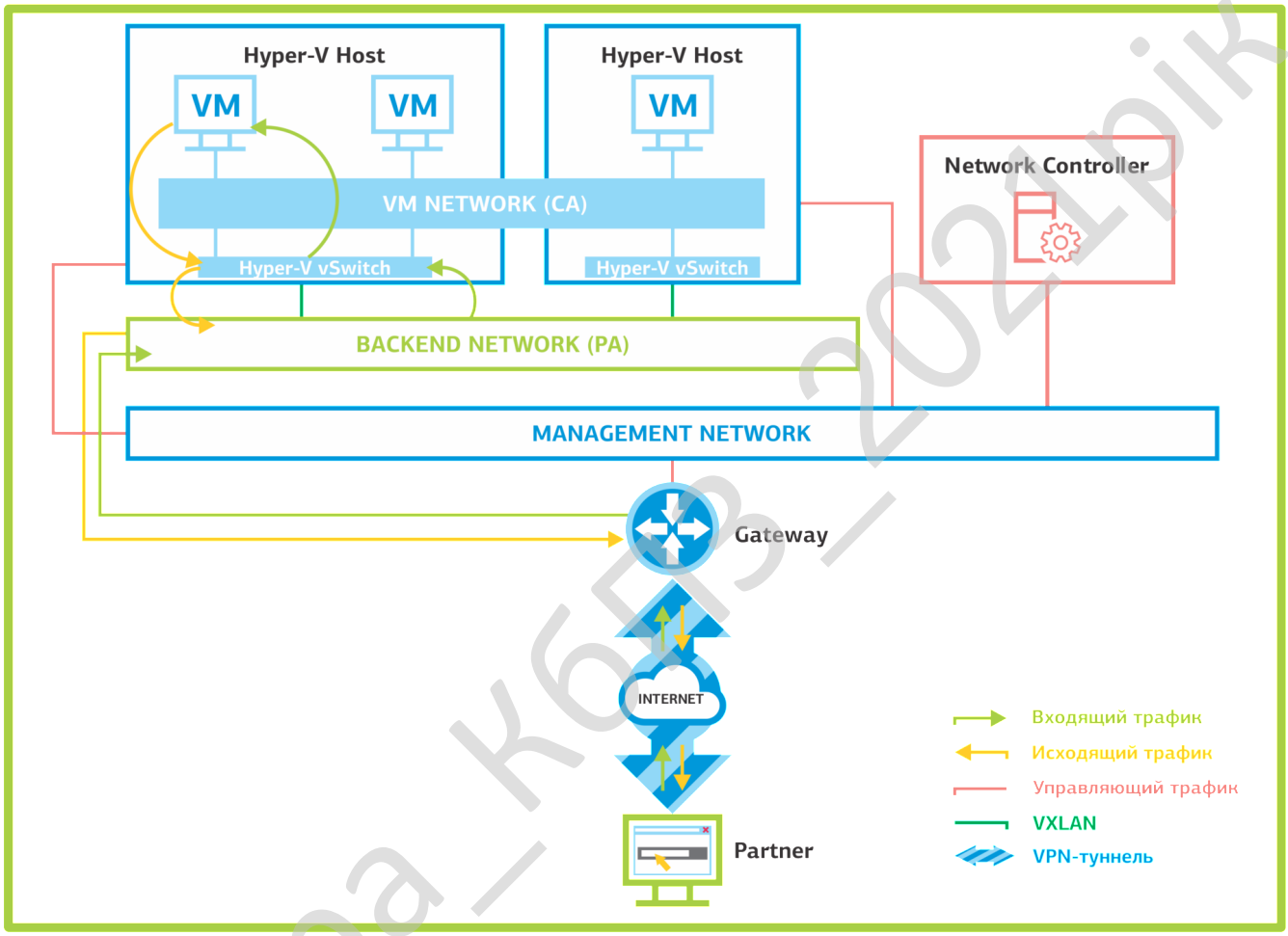

Рисунок 2.4 – Служба створення VPN-тунелів RAG Gateway Management

iDNS: Internal DNS. Служба дозволяє створювати віртуальні DNS-сервера. На даний момент існує як реалізація концепції й підтримує роботу тільки з однієї DNS-зоною, що незручно для клієнтів. Мультитенантний режим буде доданий у наступній редакції Windows Server 2016.

Розглянемо, як працює служба iDNS. Організує DNS-зону, у якій будуть створюватися клієнтські зони. Налаштовується Network Controller: вказується, що

обслуговується DNS-зона й вищі DNS-сервери, які будуть обробляти запити, що приходять від клієнтів.

Після налаштування iDNS Network Controller поширює інформацію з хостів. Служба iDNS proxy на хостах обробляє DNS-запити від клієнтів. Працює це так:

– Клієнт надсилає запит на дозвіл імені.

– iDNS-proxy дивиться, чи обслуговує він запити, що виходять із даної віртуальної мережі. Це визначається по первинному DNS-суфіксу в мережного адаптера. Він повинен збігатися із зоною, що обслуговується iDNS. Якщо і DNS-proxy обслуговує запити із цієї мережі, то відправляє запит вищому DNS-серверу.

– DNS-сервер перевіряє, чи ставиться запит до внутрішніх зон. Якщо клієнт намагається дозволити внутрішнє ім'я, то DNS-сервер його обробляє. Якщо немає – відправляє запит далі.

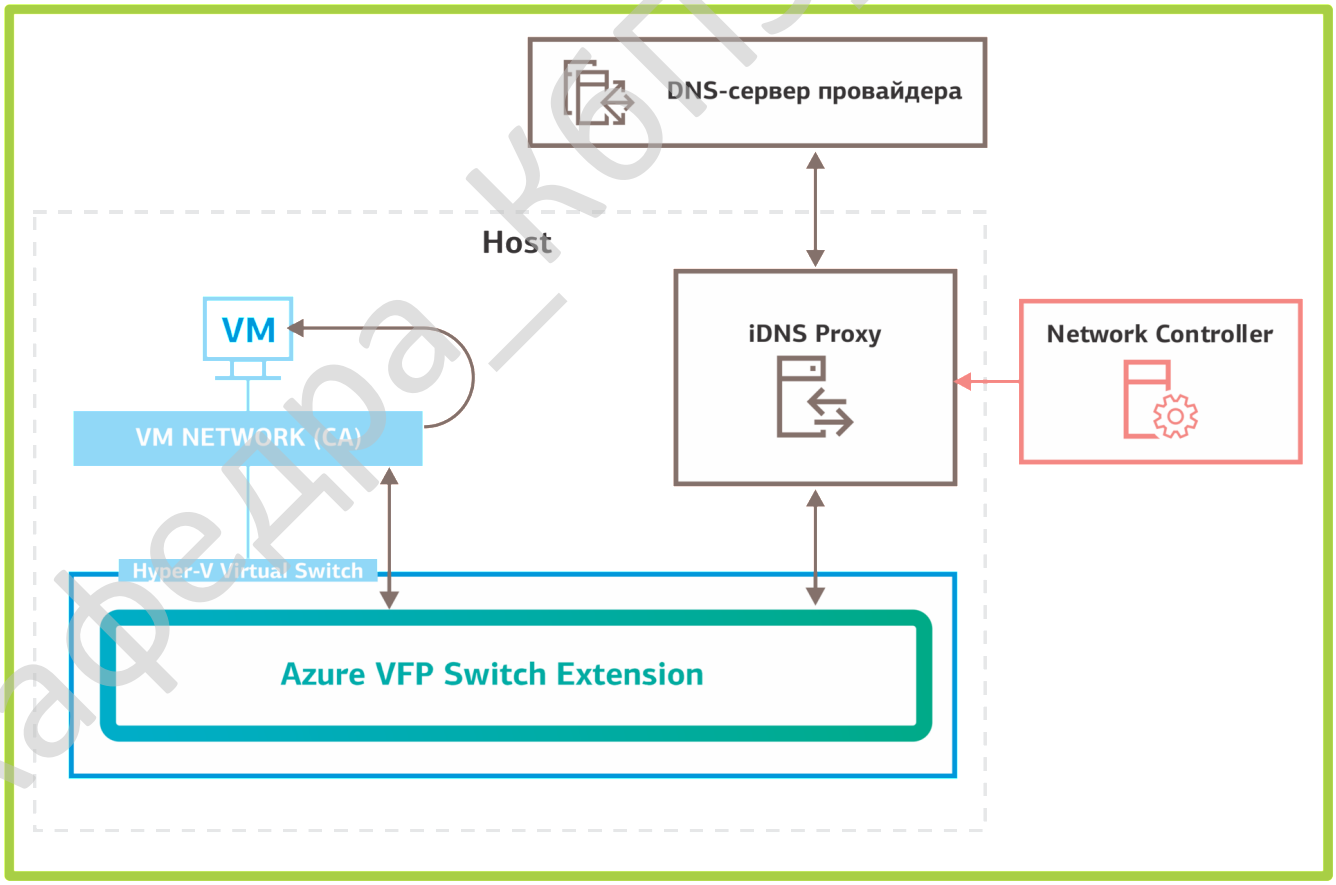

Рисунок 2.4 – Принцип роботи служби iDNS в Network Controller

Приклад

Є віртуальна мережа, для якої доступна служба iDNS. При підключенні віртуальних машин до цієї мережі, у кореневій зоні DNS, зазначеної при налаштуванні, буде створена зона з іменем, що збігаються з ID віртуальної мережі. У цій зоні будуть створюватися А-записи для віртуальних машин.

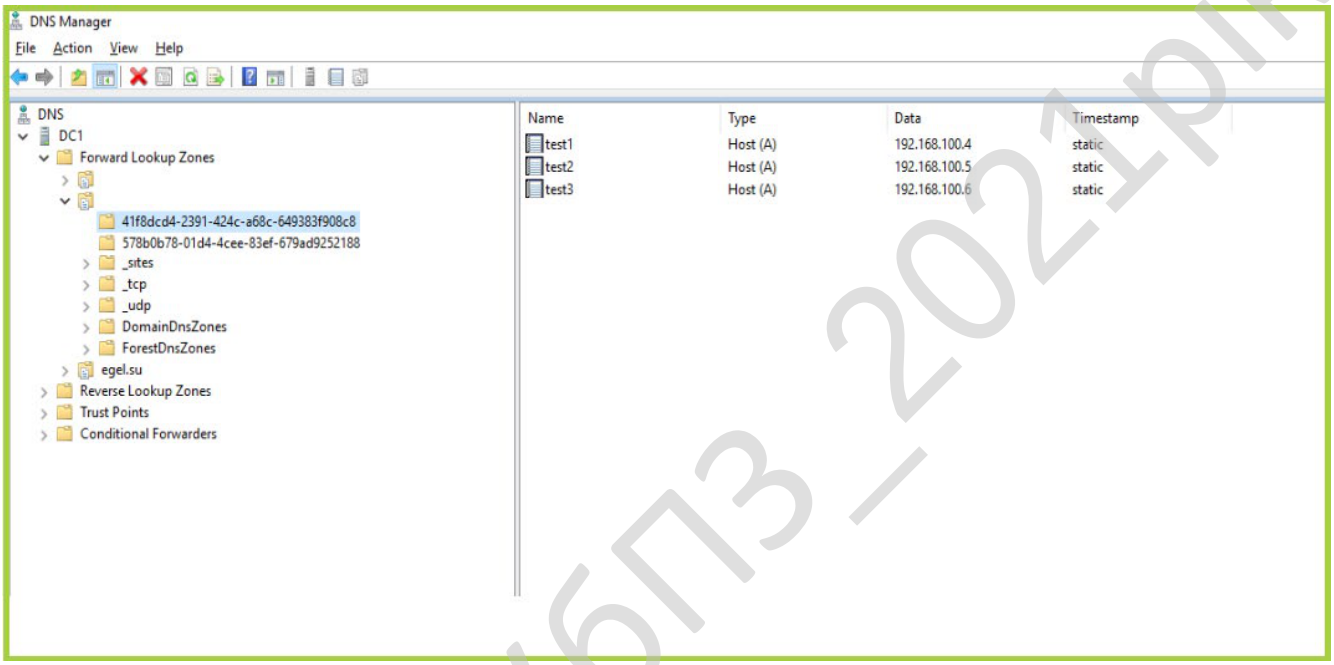

Рисунок 2.5 – А-записи віртуальних машин у службі iDNS

Усі, про що ми докладно розповіли вище, – це служби Network Controller, які ввійшли в офіційний реліз Windows Server 2016. Нам також удалося протестувати служби, що ввійшли в Technical Preview, але вилучені з релізу. Одна з таких служб – Canary Network Diagnostics для моніторингу продуктивності мережі, збору статистики по роботі фізичного й віртуального мережного устаткування й виявлення помилок. Доповнення повинні ввійти в реліз Windows Server 2016R2.

Підрозділи цього розділу містять огляд і технічні відомості про програмно певні мережні технології, які включено в Windows Server 2019 і 2016.

Мережний контролер

Мережний контролер надає централізовану, програмувальну крапку автоматизації для керування, налаштування, моніторингу й усунення неполадок як віртуальної, так і фізичної мережної інфраструктури в центрі обробки даних. За допомогою мережного контролера можна автоматизувати налаштування мережної інфраструктури, а не виконувати налаштування мережних пристроїв і служб вручну.

Мережний контролер – це високодоступний і масштабований сервер, що надає два прикладні програмні інтерфейси (API):

- API підсистеми – дозволяє мережному контролеру взаємодіяти з мережею.
- API обміну – дозволяє взаємодіяти з мережним контролером.

Для керування інфраструктурою фізичної й віртуальної мережі можна використовувати Windows Powershell, API-інтерфейс передачі даних про стан (RESTFUL) або додаток керування.

- віртуальні машини й комутатори Hyper-V;
- фізичні мережні комутатори;
- фізичні мережні маршрутизатори;
- програмні брандмауери;
- VPN-шлюзи, включаючи багатоклієнтські шлюзи служби дистанційного доступу (RAS);
- балансувальникики навантаження.

Віртуалізація мережі Hyper-V

Віртуалізація мережі Hyper-V (HNV) допомагає вам абстрактно використовувати додатки й робочі навантаження з фізичної мережі за допомогою віртуальних мереж. Віртуальні мережі забезпечують необхідну багатокористувацьку ізоляцію в спільно використовуваній фізичній мережній

інфраструктурі, підвищуючи тим самим ефективність використання ресурсів. Щоб забезпечити можливість переносу існуючих інвестицій, можна настроїти віртуальні мережі на існуючих мережних технологіях. Крім того, віртуальні мережі сумісні з віртуальними локальними мережами (VLAN).

Віртуальний комутатор Hyper-V

Віртуальний комутатор Hyper-V – це програмний комутатор рівня 2 Ethernet, доступний у диспетчерові Hyper-V після установки ролі сервера Hyper-V. Комутатор надає програмно керовані й розширювані можливості для підключення віртуальних машин до віртуальних мереж і до фізичної мережі. Крім того, віртуальний комутатор Hyper-V забезпечує примусове застосування політик для рівнів безпеки, ізоляції й обслуговування.

Також можна розгорнути віртуальний комутатор Hyper-V із включеним об'єднанням комутаторів (SET) і дистанційним доступом до пам'яті (RDMA).

Внутрішня служба DNS (iDNS) для SDN

Розміщеним віртуальним машинам і застосункам потрібно, щоб служба DNS взаємодіяла у своїх мережах і із зовнішніми ресурсами в Інтернеті. За допомогою iDNS можна надати клієнтам служби дозволу імен DNS для ізольованих локальних просторів імен і Інтернет-ресурсів.

Віртуалізація мережної функції

Апаратні пристрої, такі як підсистеми балансування навантаження, брандмауери, маршрутизатори й комутатори, усе частіше стають віртуальними пристроями. У корпорації Майкрософт віртуалізовані мережі, комутатори, шлюзи, NAT, підсистеми балансування навантаження й брандмауери. Така "віртуалізація мережних функцій" є природнім етапом розвитку віртуалізації серверів і мереж. Віртуальні пристрої швидко створюються й створюються на новому ринку. Вони продовжують створювати інтерес і одержувати імпульс як у платформах віртуалізації, так і в хмарних службах.

Доступні наступні технології віртуалізації мережних функцій:

– Load Balancer програмного забезпечення (SLB) і перетворення мережних адрес (NAT). Підвищення пропускної здатності за рахунок підтримки прямого повернення сервера, у якому мережний трафік, що вертається, може обходити мультиплексор балансування навантаження. Додаткові відомості див. у розділі Програмне балансування навантаження/(SLB/) для SDN.

– Брандмауер центру обробки даних. Надають деталізовані списки керування доступом (ACL), що дозволяють застосовувати політики брандмауера на рівні інтерфейсу віртуальної машини або на рівні підмережі. Додаткові відомості див. у розділі Загальні відомості про брандмауера центру обробки даних.

– Шлюз RAS для SDN. Направляти мережний трафік між фізичною мережею й мережними ресурсами віртуальної машини незалежно від розташування. Можна маршрутизувати мережний трафік у тому ж фізичному розташуванні або в декількох різних розташуваннях.

Дистанційний доступ до пам'яті (RDMA) і об'єднання впроваджених комутаторів

В Windows Server 2016 можна включити RDMA на мережних адаптерах, прив'язаних до віртуального комутатора Hyper-V, із включеною підтримкою об'єднання комутаторів (SET) або без неї. Це дозволяє використовувати менше мережних адаптерів, якщо ви прагнете використовувати RDMA і задати одночасно.

— Це альтернативний застосунок для об'єднання мережних карт, яке можна використовувати в середовищах, що включають Hyper-V і стек програмно-обумовлених мереж (SDN) в Windows Server 2016. Установите деякі функції об'єднання мережних карт у віртуальний комутатор Hyper-V.

Набір дозволяє групувати один і вісім фізичних мережних адаптерів Ethernet в один або декілька програмних віртуальних мережних адаптерів. Ці віртуальні мережні адаптери забезпечують високу продуктивність і

відтказостійкість у випадку збою мережного адаптера. Установка мережних адаптерів повинна бути виконана на одному фізичному вузлі Hyper-V, який буде розміщений у команді.

Крім того, ви можете використовувати команди Windows Powershell, щоб включити міст центру обробки даних (DCB), створити віртуальний комутатор Hyper-V з віртуальним мережним адаптером RDMA (vNIC) і створити віртуальний комутатор Hyper-V з параметром SET і RDMA vNIC.

Протокол прикордонного шлюзу (BGP)

Протокол BGP (BGP) – це протокол динамічної маршрутизації, який автоматично довідається маршрути між сайтами, що використовують VPN-підключення типу "мережа – мережа". Тому BGP скорочує налаштування маршрутизаторів вручну. При налаштуванні шлюзу RAS BGP дозволяє управляти маршрутизацією мережного трафіку між мережами віртуальних машин і вилученими сайтами клієнтів.

Програмне балансування навантаження для SDN

Постачальники хмарних служб (CSP) і підприємства, які розгортають SDN, можуть використовувати програмне балансування навантаження (SLB) для рівномірного розподілу мережного трафіку клієнтів клієнта й клієнта між ресурсами віртуальної мережі. Windows Server SLB дозволяє розміщати ту саму робоче навантаження на декількох серверах, що забезпечує високий рівень доступності й масштабування.

Контейнери Windows Server

Контейнери Windows Server – це спрощений метод віртуалізації операційної системи, який відокремлює додатки або служби від інших служб, що працюють на тому ж вузлі контейнера. Кожний контейнер має свою операційну систему, процеси, файлову систему, реєстр і IP-адреси, які можна підключатися до віртуальних мереж.

System Center

Розгортання інфраструктури SDN і керування нею за допомогою керування віртуальними машинами (VMM) і Operations Manager. За допомогою VMM ви виконуєте підготовку й адміністрування ресурсів, необхідних для створення й розгортання віртуальних машин і служб у приватних хмарах. За допомогою Operations Manager виконується моніторинг служб, пристроїв і операцій по всьому підприємству для виявлення проблем і оперативного реагування на них.

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 кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи моніторингу SDN.

В процесі розробки кваліфікаційної бакалаврської роботи 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

Найбільш ефективним способом для підключення до мережі й захвата трафіку є розгалужувачі мережного трафіку або Test Access Point (TAP). Дані пристрої дозволяють розв'язати як завдання забезпечення безпеки мережі, так і завдання оптимізації й підвищення продуктивності IT-інфраструктури. У чому переваги розгалужувача трафіку в порівнянні з усіма іншими способами?

Їх багато:

– швидкий доступ до даних без переривання сервісу або додаткових налаштувань на активному мережному устаткуванні;

– при підключенні в розрив каналу трафік для аналізу передається з фізичного рівня без яких-небудь змін, тобто ми бачимо всі помилки на рівні 1 і 2, які звичайно при використанні SPAN портів відкидаються;

– агрегація трафіку з різних SPAN або inline портів і передача його на один порт для аналізу;

– регенерація трафіку для аналізу різними засобами, так як SPAN портів обмежена кількість, а доступ до трафіку потрібний для аналізу різними пристроями;

– створити універсальну підмережу для копіювання трафіку й доставки по мережі для наступного аналізу.

Якщо подивитися на застосунки, представлені на ринку, то їх можна розділити на кілька класів:

– пасивні TAP розгалужувачі – схема 1:1

– агрегируючі TAP розгалужувачі – схема M:1

– що регенерують TAP розгалужувачі – схема 1:M

– брокери мережних пакетів – схема M:M

Пасивні розгалужувачі мережного трафіку включаються в розрив каналу зв'язки й для первісного підключення зажадають переривання сервісу. Розгалужувач має два порти для підключення в розрив каналу зв'язки й два порти для підключення пристрою для аналізу трафіку або моніторингу. Мідний розгалужувач трафіку компанії Gigamon має порт A и порт B для підключення в розрив і два порти моніторингу, на кожний з яких копіюється відповідні потік Monitor A і Monitor B.

Для аналізу трафіку необхідний пристрій, який має два мережні інтерфейси й на своєму рівні може об'єднати ці потоки або все частіше доводиться зустрічати, що після такого пристрою коштує більш функціональне залізо, наприклад, багатопортовий брокер мережних пакетів. Найбільш популярна схема підключення до оптичного магістрального каналу зв'язку включає пасивний оптичний розгалужувач із поділом 70 на 30 і брокер мережних пакетів. У брокер підключають пристрої контент фільтрації, системи виявлення й запобігання вторгнень або системи моніторингу мережного трафіку.

Агрегируючі розгалужувачі працюють за схемою кілька портів в один і дозволяють проаналізувати трафік відразу з декількох сегментів мережі. Трафік звичайно збирається з декількох SPAN портів і подається на пристрій для аналізу. Так як трафік збирається з SPAN портів, т о зрозуміло, що для підключення до мережі переривання сервісів не потрібно. При такому підключенні важливо пам'ятати про пропускну здатність порту моніторингу, тобто якщо ми збираємо трафік з 4-х сегментів мережі, які завантажені більш ніж на 10%, те на порту моніторингу ми можемо одержати перевантаження, а, отже, упустити частина пакетів при підвищенні завантаження каналу.

Регенераційні розгалужувачі трафіку, як ви вже догадалися, дозволяють обійти обмеження кількості доступних SPAN портів, коли, наприклад, на кореневому комутаторі доступний один SPAN порт, а нам необхідно підключити кілька систем для аналізу трафіку. Тоді ідеальний застосунок – регенераційний розгалужувач мережного трафіку. Він підключається в розрив каналу й копіює

трафік на кілька портів, до яких і підключаються всі системи моніторингу безпеки або продуктивності.

У випадку якщо перед вами коштує більш комплексне завдання й необхідно підключитися в розрив каналу й агрегувати і регенерувати трафік, та ще з попередньою фільтрацією трафіку або конверсією швидкості (з 1 Гбіт/сек на 10Гбіт/сек або навпаки) або конверсією середовища передачі (мідь/оптика), то для її застосунків існують брокери мережних пакетів, які мають інтерфейс керування й безліч налаштувань і працюють за схемою M:M. Брокер мережних пакетів дозволяє здійснювати дедуплікацію пакетів, з погляду безпеки видаляти корисне навантаження, обмежуючи аналіз тільки заголовків, а також виконувати балансування навантаження для оптимального завантаження засобів моніторингу.

Завершуючи розділ можна сказати, що TAP розгалужувачі мають один недолік – вони коштують грошей, але при цьому, як жоден інший метод дозволяють одержати повний доступ до мережного трафіку й гнучко набудовувати пристрою виходячи з вартому перед вами завдання – безпека або аналіз продуктивності.

Скільки трафіку губиться при використанні SPAN порту

Компанія Garland Technology провела порівняння між копіюванням трафіку за допомогою SPAN порту й за допомогою розгалужувача трафіку. Виводи невтішні – з використанням SPAN порту ми втрачаємо до 8% трафіку.

3.2 Розробка структурної схеми

Слідом за віртуалізацією серверів і застосунків прийшла черга мереж. Розроблені 25 років тому класичні віртуальні мережі VLAN, що працюють на другому рівні моделі OSI були відмінним застосунком для логічного угруповання пристроїв і керування обміном інформацією між ними. Але даний підхід має обмеження – з його допомогою можна організувати всього 4094 мережі й

неможливо перенести віртуальну машину через границі канального рівня. У такий спосіб виникла модель створення накладених віртуальних мереж поверх існуючої фізичної IT інфраструктури.

У якості найпоширеніших протоколів побудови накладених (оверлейних) мереж можна привести VXLAN (компанія VMware) і NVGRE (компанія Microsoft). Усі протоколи мають на увазі наявність віртуального комутатора на базі гіпервізора й термінованих тунелів у віртуальних вузлах. Що дозволяє будувати логічні мережі на канальному рівні в рамках уже існуючих мереж рівня 3 моделі OSI. Віртуалізація мережі відповідає на виклики мобільності й багатокористувацького використання, але створює додаткові проблеми, пов'язані з питаннями моніторингу, керування й безпеки як для самої фізичної інфраструктури, так і накладених віртуальних мереж. Основні проблеми з яким прийде зіштовхнутися – це моніторинг і контроль над синхронізацією керування мережею (фізичної й накладеної) і керування потоками даних.

Система моніторингу SDN мереж

Застосунки на основі SNMP для моніторингу всієї мережі, швидше за все не будуть затребувані, так як SDN-мережі будуються на основі простих пристроїв комутації й увесь розум зосереджений у контролері, тому основну інформацію через API інтерфейси можна буде легко зняти. А от проблему синхронізації й можливих збоїв у спілкуванні контролер-комутатор прийде вирішувати за допомогою застосунків для аналізу продуктивності сервісів і мережі на основі глибокого аналізу реального трафіку.

Представимо більшу програмно-обумовлену мережу. Усе керування здійснюється контролером, і він повинен бути синхронізований з усіма комутаторами й оперативно обновляти інформацію. На папері виглядає гарно й легко написати, що все буде літати й працювати як треба, але реальне життя складніше віртуальної й проблеми будуть і рости з ростом SDN мережі. Ріст SDN мережі й кількість устаткування буде приводити до збільшення часу для синхронізації контролера й комутаторів при внесенні змін у конфігурацію

мережі. Це може бути пов'язане з різними факторами, такими як затримка між контролером і комутатором, втрати пакетів у каналах зв'язку, устаткування від різних виробників, яке має різні розміри внутрішніх таблиць комутації, ну або як мінімум наявність багів і проблем у програмному або апаратному забезпеченні. Залежно від впровадження, комутатори можуть розсинхронізовуватися з контролером і не поновити зв'язок протягом деякого періоду часу. У такій ситуації передбачити поведінка мережі дуже складно. Для того, щоб мінімізувати такі ситуації необхідно моніторити трафік між комутаторами, щоб переконається, що мережа працює як планувалося в рамках того, що вважається нормальним. Це завдання буде можливо розв'язати шляхом кореляції інформації з потоків даних на рівні комутаторів і налаштувань на рівні контролера. Дана інформація може бути корисна не тільки для застосунків завдань моніторингу продуктивності, але й для забезпечення безпеки SDN мережі. А також може використовуватися як зворотний зв'язок для внесення змін у налаштування через контролер у комутатори для відновлення очікуваного поведінки мережі.

Створення віртуальних мереж і в минулому й у майбутньому пов'язане з додаванням додаткових заголовків у пакети, що робить скрутним аналіз трафіку за допомогою застосунків на коліні (типу ноутбук і аналізатор трафіку). Також не всі існуючі аналізатори можуть коректно відпрацьовувати трафік і видалення заголовків, які ставляться до VXLAN, NVGRE і т.д. У цьому випадку можуть бути корисні брокери мережних пакетів, які оперативно обновляють свої функціональні можливості в частині аналізу нових видів протоколів і інкапсуляцій, у тому числі й для підтримки SDN мереж. Створення й видалення віртуальних каналів і оверлейних мереж відбувається на рівні гіпервізора й на рівні фізичної інфраструктури цей процес буде неможливо проконтролювати, що робить пошук несправностей і керування продуктивністю каналів зв'язки дуже складним. Наприклад, якщо пакет даних був відправлений з однієї віртуальної машини на іншу з використанням VXLAN і не дійшов до адресата, то причини можуть бути: у гіпервізорі; у відправника; у контролері SDN мережі (невірний

маршрут); в одержувача; у базовій фізичній мережі. І нарешті, поділ потоків керування й користувача на рівні накладеної мережі й на рівні фізичної, вимагає контролю з можливістю кореляції, щоб бачити, як одна мережа впливає на іншу.

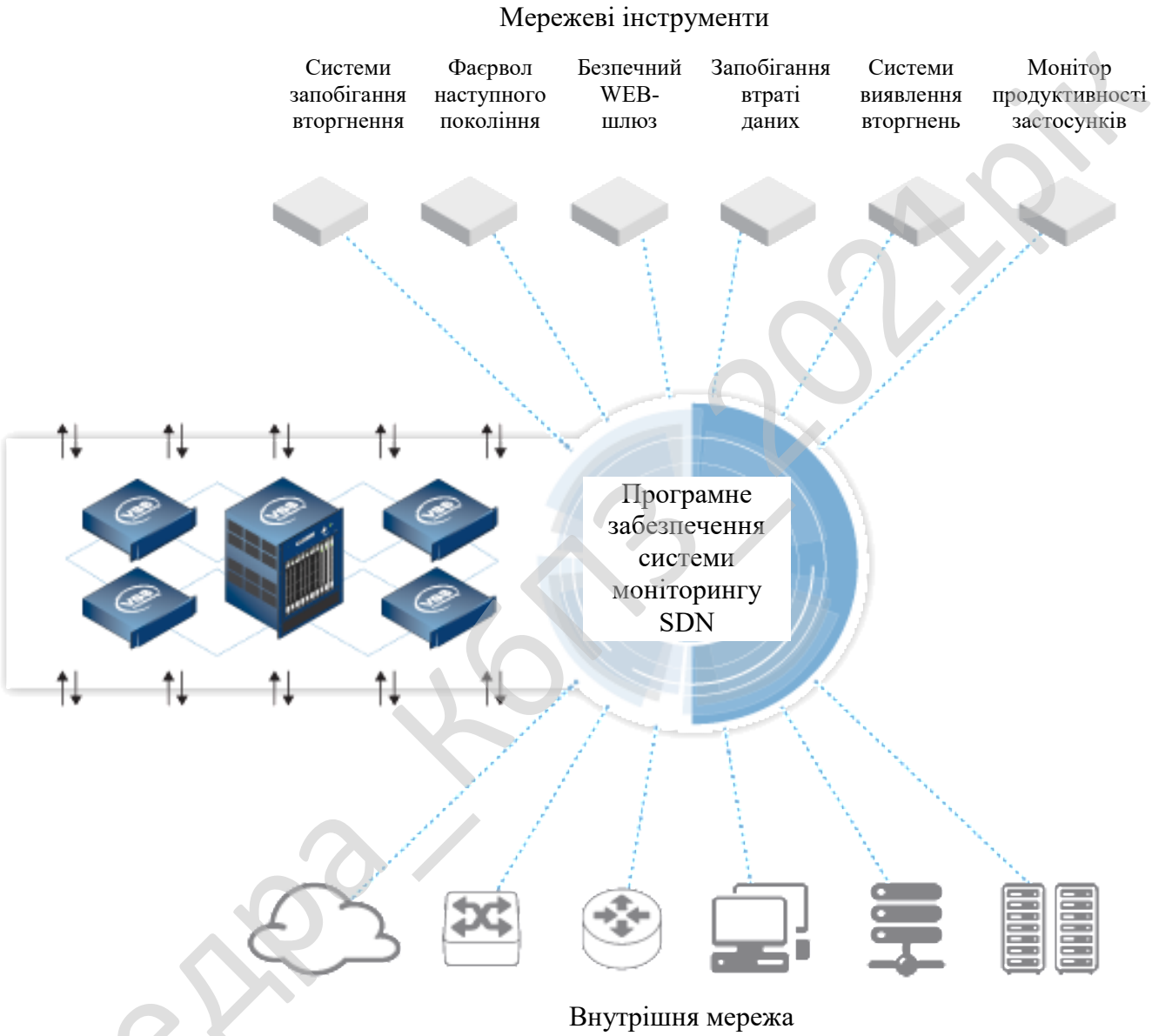

Рисунок 3.1 – Структурна схема системи

Таким чином, для забезпечення ефективного моніторингу мереж SDN і фізичної інфраструктури необхідні сучасні застосунки, які підтримують усі нові протоколи й інкапсуляції. Основна ідеологія SDN мережі – зробити мережу

розумною й недорогою, тому система моніторингу продуктивності й безпеки, також як і керування мережею, повинна бути встановлена в одному місці. Таким чином, усі розподілені системи моніторингу заміняються на мережу знімання інформації за допомогою розгалужувачів трафіку.

Розгалужувачі трафіку встановлюються в мережі й надають доступ до реального трафіку. Далі маркуючи трафік, доставляють його в центр для обробки й аналізу системами моніторингу або безпеки й видачею зворотного зв'язка для коректування налаштувань комутаторів через контролер. Компанія Gigamon дала назву своєї філософії Unified Visibility Fabric, а компанія VSS Monitoring – Unified Visibility Plane:

Але суті це не міняє, і система моніторингу стає відірваною від фізичної інфраструктури мережі, гнучко управляється через єдиний інтерфейс користувача й гарантовано доставляє трафік для його обробки. Таким чином, буде дотримана й економічна складова, яку несе SDN мережі й можливості повного контролю продуктивності, які властиві традиційним мережам.

3.3 Розробка функціональної схеми

Ми розглянули ключові компоненти й принципи роботи рішення SDN. Тепер проаналізуємо практичну цінність рішення для IT і бізнесу на прикладі конкретних сценаріїв.

Розглянемо ряд завдань, типових для IT, і зрівняємо дві умовних мережі: «класичну» і мережа на основі фабрики SDN (далі – мережа SDN).

Під класичною мережею в цьому порівнянні будемо розуміти кампусну мережу з рівнем доступу, що комутирується, й маршрутизуємим ядром. Припустимо, що контроль доступу в такій мережі реалізується за допомогою списків контролю доступу (ACL), а більшість операцій по налаштуванню, пошуку й усуненню несправностей проводиться вручну. Мережа використовує RADIUS-сервер для контролю доступу користувачів.

Під мережею SDN будемо розуміти мережу, побудовану на основі фабрики SDN, що включає в себе мережеву інфраструктуру, контролер DNA Center і сервер контролю доступу ISE.

Як швидко запустити мережу в експлуатацію?

Одне з типових завдань IT пов'язана із запуском в експлуатацію нових частин мережі або мереж на нових площадках, а також з модернізацією вже наявних мереж.

Рішення цього завдання у випадку класичної мережі часто реалізується шляхом складної комбінації ручної праці й різнорідних засобів автоматизації.

З таким підходом складно забезпечити консистентні конфігурації устаткування при впровадженні й особливо в процесі експлуатації, коли неминуче виникають зміни. Процес слабко захищений від помилок внаслідок «людського фактора». І, нарешті, він не дозволяє гнучко адаптуватися до змін бізнесу – у процесі робіт із впровадження, що включають у себе розробку дизайну й конфігурацій, замовлення й фізичну поставку устаткування, стейджинг і впровадження конфігурацій – можуть відбутися зміни в потребах бізнесу й впроваджувана конфігурація виявиться застарілою ще до запуску мережі в експлуатацію.

Здавалося б, достаток засобів централізованого керування й додаткового інструментарію повинне вирішувати цю проблему. Але дані дослідження Cisco, по яких у корпоративних мережах близько 90% змін дотепер виробляється вручну, говорять про зворотнє.

Мережа SDN пропонує інструментарій для оптимізації й автоматизації процесу. DNA Center дозволяє централізовано задати ієрархічну структуру площадок організації, прив'язаних до географічного місця розташування. Для об'єктів ієрархії можна централізовано вказати значення параметрів, наприклад адреси серверів AAA, NTP, DNS, колекторів даних NetFlow, Syslog і SNMP, паролі й т.п. Ці параметри можуть успадковуватися по ієрархії мережами інших площадок, а можуть задаватися індивідуально. Далі, DNA Center дозволяє

створити шаблони конфігурацій для подальшого твердження й впровадження в мережі. Крім того, мережа DNA пропонує інструментарій Plug and Play (PnP) для автоматизованого підключення нових пристроїв у фабрику з можливістю їхньої прив'язки до площадок ієрархії й централізованим розподілом конфігурацій по пристроях мережі.

У результаті мережа SDN забезпечує:

– Значне прискорення розгортання мережі за рахунок автоматизації.

– Зниження ризиків, пов'язаних з «людським фактором», за рахунок автоматизованої підтримки консистентних і актуальних конфігурацій пристроїв.

Як підключати клієнтські пристрої швидко й безпечно?

Тренд цифровізації приводить до значного росту кількості й типів пристроїв, що підключаються до мережі. Це настільні комп'ютери й ноутбуки, смартфони й планшети, принтери, камери відеоспостереження, сенсори й інші пристрої Internet of Things (IoT) і т.д.

Необхідно переконатися не тільки в тому, що план IP-адресації мережі готовий до підключення всіх необхідних пристроїв, але й застосувати відповідні політики безпеки.

Традиційна мережа вимагає значних зусиль для рішення цього завдання. По-перше, оскільки IP-адреси звичайно використовуються не тільки для адресації, але і як критерій для застосування політик, план адресації ускладнюється, а внесення змін в адресацію стає досить скрутним. По-друге, складності реалізації політик часто приводять до того, що внутрішній периметр провідної мережі не захищений, а політики безпеки до пристроїв, що підключаються, не застосовуються або в найкращому разі застосовується базовий функціонал 802.1x.

Мережа SDN має централізований функціонал керування пулами IP-адрес, у т.ч. з інтеграцією із сервісами Infoblox. Це значно спрощує роботу з адресацією мережі. Крім того, і сама архітектура мережевої фабрики сприяє значному спрощенню IP-адресації, тому що дозволяє використовувати адреси тільки по

їхньому цільовому призначенню. Більше немає необхідності «прив'язувати» політики до адрес хостів і підмереж. Крім того, навіть якщо через вимоги застосунків або яких-небудь клієнтських пристроїв потрібна «розтягнута» по кампусу IP-підмережа, оверлей дозволяє вирішити це завдання без порушення дизайну маршрутизуємої опорної мережі, без ризиків широкомовних штормів і інших неприємностей доменів Рівня 2. У результаті мережа SDN готова до підключення максимальної кількості пристроїв різних типів при мінімальних вимогах до кількості підмереж.

Захист внутрішнього периметра мережі реалізується завдяки інтеграції із сервісом ISE і службою каталогів Active Directory, що забезпечують динамічне й гранулярне застосування політик безпеки без прив'язки до IP-адрес.

Таким чином, мережа SDN пропонує:

– Готовність до масового підключення пристроїв різних типів за рахунок простоти керування IP-адресацією, ефективності використання адрес і гнучкого транспорту Рівнів 2 і 3.

– Значну оптимізацію витрат часу й сил на керування IP-адресацією.

– Безпечне впровадження пристроїв за рахунок простоти застосування динамічний, гранулярних політик безпеки без прив'язки до IP-адрес.

Як швидко адаптувати існуючі сервіси й політики до нових вимог бізнесу?

Ще зовсім недавно від мережі було потрібно відносно деяке – забезпечити доступ користувачів в Інтернет, до електронної пошти, до набору застосунків, що працюють у корпоративному ЦОД.

Але вимоги до мережі стрімко ростуть і продовжують рости. Набирає оберти тренд цифровізації. Росте кількість елементів мережевої інфраструктури, для яких необхідно робити провіженінг, налаштування, моніторинг, пошук і усунення несправностей. Стрімко росте кількість підключень до мережі, і за цими підключеннями можуть стояти як користувачі (у випадку комп'ютерів, телефонів, планшетів і т.п.), так і «машини» (системи автоматизації, цифрові вивіски,

сенсори й інші пристрої Internet of Things (IoT). Всім цим пристроям потрібно надати доступ у мережу з необхідним рівнем сервісу й правильних повноважень безпеки.

Просто забезпечити зв'язок між точками А и Б, як правило, уже недостатньо. Необхідні ще й політики. Політики можна класифікувати як транспортні (визначальні можливості взаємодії між областями мережі, вимоги до шляху передачі), політики безпеки (визначальні контроль доступу, автентичність, цілісність, конфіденційність переданих даних), сервісні політики (визначальні обробку потоків трафіку мережевими функціями).

Наприклад, безпечна робота різних категорій користувачів і пристроїв у корпоративній мережі звичайно вимагає належного контролю доступу. У ряді випадків взагалі краща ізоляція їхній друг від друга – наприклад, у випадку доступу в мережі гостьових користувачів, впровадження пристроїв IoT і т.п.

Інші приклади – для телефонів або комп'ютерів з мультимедійними додатками, що працюють у бездротової ЛОМ, може вимагатися роумінг на Рівнях 2 і 3. Міркування вимог застосунків або організації доступу до ресурсів можуть приводити до необхідності присутності однієї й тієї ж IP-підмережі в різних частинах корпоративного кампусу.

Повільне впровадження нових сервісів може означати упущені бізнес-можливості й, в остаточному підсумку, загублену частку ринку. Тому мережеві сервіси й політики, важливі для роботи бізнес-процесів, повинні впроваджуватися чим швидше, тим краще. Виникає питання – як забезпечити запуск нових сервісів і політик, що відповідають вимогам бізнесу, причому зробити це швидко й по всій мережі?

Класична мережа в сучасних умовах вимагає усе більше часу й зусиль для запуску нових сервісів і адаптації до змін у політиках, тому що вона являє собою набір автономних друг від друга пристроїв. Крім того, пристрої класичної мережі можуть мати різні операційні системи, різні способи реалізації потрібного функціонала, різні способи його налаштування в інтерфейсі командного рядка.

Централізовані засоби керування допомагають далеко не завжди. Хоча вони й дають можливості автоматизації багатьох IT-процесів, але створення єдиної, скоординованої політики вимагає функціонала оркестрації, що на практиці відсутній у класичній мережі.

Інша обставина – згодом конфігурації пристроїв більшості корпоративних мереж «обростають» додатковими налаштуваннями, політиками й стають усе більше складними й усе менш консистентними. Це, у свою чергу, утрудняє використання засобів централізованого керування. У підсумку, відповідно до внутрішнього дослідження від 2016 р., близько 90% змін у корпоративних мережах дотепер виробляється вручну. І це, як правило, не тільки довго, але й ризиковано через «людського фактора».

Мережа SDN пропонує більше сучасний підхід на основі технології програмно-визначаємой мережі й контролера DNA Center. Контролер управляє як опорною мережею, так і оверлейною топологією й має цілісну картину всієї мережі. Адміністратор може задати політики, використовуючи графічний інтерфейс DNA Center.

Мережеві сервіси й політики існують не самі по собі, а для виконання певних вимог бізнесу, або « бізнес-намірів». DNA Center побудований навколо ідеї «бізнес-намірів» – адміністратор спочатку задає високорівневі деталі налаштування, логічні групи користувачів, бажані взаємозв'язки між ними, політики обробки трафіку й т.д., а далі контролер програмує мережеву фабрику з метою досягнення бажаного результату. Контролер взаємодіє з мережевою інфраструктурою через south bound APIs – Netconf/YANG, REST, а також CLI.

Крім того, контролер дозволяє абстрагувати корпоративну мережу за допомогою north bound REST API і застосовувати будь-які засоби програмування мережі, що працюють через ці API.

У результаті мережа SDN допомагає забезпечити оперативне впровадження мережевих сервісів і політик і, в остаточному підсумку, швидко виконувати вимоги бізнесу.

Як реалізувати універсальний транспорт Рівнів 3 і 2? Як інтегрувати провідні й бездротові ЛОМ?

Багатство застосунків і сценаріїв використання мережі в різних областях бізнесу, промисловості й т.д. може приводити до потреби у швидкому й надійному транспорті по всьому кампусу не тільки на Рівні 3, але й на Рівні 2.

Прикладами таких сценаріїв можуть бути:

– Промислове підприємство, у якому через зміну потреб виробництва потрібно переміщати устаткування, підключене до провідної мережі, а це устаткування не дозволяє легко змінити використовувану адресацію;

– Університет, у якому студенти, персонал, дослідники переміщаються по кампусу й них необхідний провідний і бездротової доступ до своїх застосунків без зміни підмереж;

– Медична установа, у якому необхідно провести фізичну реконфігурацію лабораторної IT-середовища без змін в адресації;

– Будь-яка організація, у якій вимоги застосунків або сценарію використання приводять до необхідності розподілу єдиного домену Рівня 2 по кампусу або фізичного переміщення пристроїв без зміни IP-адрес.

Крім того, клієнтські пристрої стають усе більше мобільними, можуть переміщатися й підключатися як до провідних, так і бездротовим мережам.

Як же забезпечити універсальний транспорт Рівнів 3 і 2, створити єдине транспортне середовище, що охоплює провідні й бездротові ЛОМ?

Вирішити це завдання в класичній мережі важко. Для цього буде потрібно або прокладати VLAN'и по кампусу, жертвуючи простотою, надійністю й гнучкістю дизайну, або застосовувати традиційні оверлейні топології, наприклад MPLS VPN, що ускладнює мережа й підходить не для всіх організацій.

Інтеграція провідних і бездротових мереж являє собою окремі труднощі. У сучасних кампусних мережах найбільше поширення одержала централізована архітектура бездротових ЛОМ, у якій трафік користувачів БЛОМ тунелюється від точок радіодоступу на контролер і надходить у провідну мережу вже через нього.

Тому трафік провідних і бездротових пристроїв обробляється по-різному, а застосувати до нього єдині політики важко.

Мережа SDN вирішує це завдання завдяки оверлею, невід'ємній властивості фабрики. Інкапсуляція трафіку в пакети VXLAN забезпечує транспорт як Рівня 3, так і Рівні 2. А точки радіодоступу інтегруються у фабрику завдяки технології Fabric Enabled Wireless (FEW). Трафік і провідних, і бездротових користувачів надходить у фабрику через комутатори доступу (Edge Nodes), до нього можна застосовувати однакові, консистентні політики. Кожний комутатор доступу (Edge Node) мережі SDN реалізує control plane на базі протоколу LISP і забезпечує L3 Anycast Gateway. У результаті SDN дозволяє клієнтським пристроям зберігати ті самі IP -адреси, переміщаючись по всьому кампусу.

Таким чином, мережа SDN реалізує:

– Інтегрований, простій у використанні механізм транспорту на Рівнях 3 і 2.

– Інтеграцію провідних і бездротових мереж у єдине транспортне середовище.

– Зниження витрат праці й ризиків, пов'язаних із забезпеченням транспорту Рівнів 3 і 2.

Як сегментувати мережу?

Небувалий зліт активності зловмисників, що спостерігається останнім часом, переводить багато погроз і вектори атак з теоретичної площини в практичну.

Наприклад, зловмисники викрали дані понад 40 млн. платіжні карти в Target, проникнувши в корпоративну мережу через підключену до мережі систему кондиціювання.

Епідемія хробаків-шифрувальників, таких як WannaCry, Petya і т.п., нанесла ще більший збиток по усьому світі.

Можна або запобігти, або істотно знизити ефект від подібних інцидентів шляхом впровадження сегментації мережі. Сегментація мережі дозволяє розділити користувачів мережі на відособлені групи, трафік між яким контролюється або блокується залежно від вимог політики безпеки й поточних обставин.

Рішення завдання сегментації в класичній мережі звичайно досягається створенням віртуальних логічних топологій (наприклад, на базі VLAN, VRF, MPLS VPN і т.п.) і/або застосуванням списків контролю доступу (ACL). Обидва методи вимагають серйозних витрат часу й сил і ці витрати тим більше, ніж динамічніше середовище сегментації. У підсумку зміни політики сегментації реалізуються складно й довго. Крім того, як сегментація за допомогою віртуальних топологій, так і ACL уразлива до проявів «людського фактора», і ця уразливість знову ж тим більше, ніж динамічніше вимоги сегментації. У деяких випадках ці обставини можуть взагалі привести до непрактичності сегментації в класичній мережі.

Мережа SDN пропонує могутніший і гнучкий інструментарій для рішення завдання сегментації. Вона може виконуватися на двох рівнях. Перший рівень – сегментація на віртуальні мережі (Virtual Networks) за допомогою механізму VRF. Другий рівень – сегментація на групи (Scalable Groups) за допомогою технології TrustSec.

Сегментація на базі VRF оптимальна для грубого поділу на віртуальні мережі, состав яких відносно статичний. Наприклад, це можуть бути окремі віртуальні мережі для корпоративних співробітників і інженерних підсистем будинків. Інший приклад – окремі мережі для компаній, що входять в одну групу компаній.

Сегментація на базі TrustSec дуже зручна для динамічних середовищ, у яких часто міняються й состав груп користувачів, і правила їхньої взаємодії. TrustSec дозволяє задавати групи користувачів і правила взаємодії централізовано

на сервері ISE. Далі ці правила поширюються по мережі автоматично й реалізуються мітками SGT і списками контролю доступу на основі міток SGACL.

TrustSec забезпечує дуже високий ступінь автоматизації. Технологія оптимальна для динамічних середовищ і може успішно використовуватися для мікросегментації користувачів на малі групи.

Технологія TrustSec реалізована в широкому спектрі серій устаткування і може застосовуватися незалежно від фабрики SDN. Але крім цього, TrustSec глибоко інтегрована й у фабрику і є однієї з її невід'ємних функцій, якими можна легко скористатися при експлуатації мережі SDN.

Крім того, оскільки мітки SGT передаються в складі заголовків VXLAN, немає необхідності підтримки технології TrustSec на кожному проміжному пристрої фабрики або застосування додаткових протоколів control plane, таких як SXP. Досить реалізації TrustSec на прикордонних пристроях фабрики, що і так є присутнім у них споконвічно.

Таким чином, SDN пропонує елегантне рішення завдання сегментації за рахунок інтеграції технології TrustSec у функціонала фабрики.

У результаті мережа SDN забезпечує:

– Значне зниження ризиків ІБ і простою бізнес-процесів через «людського фактора» завдяки інтегрованим і простим у використанні, але дуже потужним засобам сегментації на базі технологій VRF і TrustSec.

– Значну оптимізацію витрат праці на сегментацію за рахунок автоматизації.

– Значне прискорення реалізації сегментації за рахунок автоматизації.

Як реалізувати цілісну, консистентну, динамічну політику по всій мережі?

Практично будь-яка сучасна мережа має ту або іншу реалізацію політик – політик безпеки, обробки трафіку, якості обслуговування (QoS) і т.п.

Реалізація політик критично важливий, тому що саме вона в остаточному підсумку й забезпечує транспортне середовище з потрібними бізнесу сервісами і якістю.

У класичній мережі політики реалізуються у вигляді комплексу складних у впровадженні й експлуатації налаштувань мережевої інфраструктури.

Мережа SDN пропонує можливість реалізації єдиний, консистентних політик за рахунок засобів оркестрації DNA Center і властивостей фабрики. Крім того, такий функціонал динамічний і тому зручно з погляду адаптації до вимог, що змінюються, бізнесу.

Політика безпеки

Розглянемо реалізацію політики безпеки на прикладі класичної мережі. Контроль доступу – найважливіший компонент політики безпеки – у класичній мережі часто опирається на IP-адреси хостів і підмереж як критерію для ухвалення рішення. Це відбувається тому, що в заголовку IP-пакета немає даних, що дозволяють ідентифікувати користувача, визначити його приналежність до тої або іншої групи. Тому як заміна доводиться використовувати IP-адреси хостів і підмереж.

Політика, реалізована на основі IP-адрес, стає не централізованої, а розподіленої по мережі в конфігураціях пристроїв. З'являються ACL довжиною в тисячі рядків. Користувачі й пристрої групуються по VLAN'ам. Підмережі використовуються для групування користувачів і пристроїв по яких-небудь ознаках. Створюються віртуальні мережеві топології для сегментації трафіку. IP-адреси стають просто перевантажені значеннєвим навантаженням.

У результаті такий підхід до реалізації політики приводить до більших витрат часу й сил, а також підвищеної ймовірності помилки.

Крім того, використання IP-адрес як критерій для застосування політик позбавляє мережу гнучкості: з'являється взаємозв'язок з адресацією, доводиться створювати більше підмереж і VLAN'ов, чим це було б необхідно без обліку вимог політики. Внесення змін у мережу стають усе довше й ризикованіше,

процеси експлуатації, пошуку й усунення несправностей – складніше, а мобільність хостів без застосування додаткових методів, тунелюючих трафік користувача, непрактична, оскільки зміна місця розташування часто приводить до зміни підмережі.

Реалізація політики безпеки на основі IP-адрес працює тим гірше, чим більше динаміки в мережевому середовищі. А такої динаміки звичайно багато й у сучасних мережах згодом стає усе більше.

Мережа SDN дає можливість реалізувати єдину цілісну політику, використовуючи як критерій для застосування політики не IP-адреса, а групу SGT (Scalable Group Tag), інформація про яку інкапсулюється в заголовок VXLAN мережевої фабрики. Призначення користувачів у групу SGT можна робити централізовано й динамічно стосовно бази даних сервера контролю доступу Identity Services Engine (ISE), а також Active Directory. Призначення в групу SGT робиться з урахуванням додаткового контексту, такого як профілювання й posture.

У підсумку мережа SDN забезпечує цілісну, консистентну, динамічну політику по всій мережі, незалежну від IP-адрес.

Політика якості обслуговування (QoS)

Політики якості обслуговування (QoS) у сучасних корпоративних мережах реалізуються, як правило, у рамках моделі DiffServ. Модель припускає класифікацію й маркування трафіку застосунків на границі домену DiffServ з подальшим диференційованим обслуговуванням усередині домену на підставі проведених маркувань.

Повномасштабна реалізація політик QoS в умовах класичної мережі дуже складна. До ключових причин можна віднести, як мінімум, що випливають:

– Складність конфігурації функціонала QoS і налагодження його роботи.

– Складність внесення змін у реалізацію політики QoS.

– Неконсистентність функціонала QoS і (у ряді випадків) помітні розходження в методах його налаштування на устаткуванні, що утворить мережа.

– Складність належної класифікації трафіку через неефективність наявних засобів класифікації (наприклад, на підставі адрес і портів мережевого й транспортного рівнів) або неприступності на наявному устаткуванні ефективних засобів (наприклад, NBAR2, що сполучить механізми DPI, DNS-AS, статистичний і поведінковий аналіз і ряд інших засобів).

веде роботу над функціоналом створення політик QoS в інтерфейсі DNA Center. У результаті з'явиться можливість вибору площадок мережі й впровадження (push) політики QoS на пристрої фабрики.

Централізована оркестрація й автоматизоване впровадження QoS покликані вирішити проблему складності конфігурації й відновлення політики. Застосування єдиної апаратної бази (наприклад, інтегральних мікросхем UADP) на комутаторах фабрики і єдиного програмного забезпечення IOS-XE на комутаторах і маршрутизаторах фабрики значно підвищують консистентність реалізації QoS. Нарешті, елементна база фабрики підтримує механізми NBAR2 класифікації застосунків на додаток до традиційних методів.

Таким чином, мережа SDN пропонує інструментарій для ефективного централізованого впровадження політики QoS у корпоративному середовищі.

Політика обробки трафіку

У корпоративній мережі також виникає потреба застосування політик обробки трафіку, наприклад для вибору конкретного шляху проходження трафіку (traffic engineering) і дзеркалювання потрібного трафіку.

В умовах класичної мережі подібні політики можна реалізувати за допомогою статичних і динамічних засобів, наприклад Policy Based Routing (PBR), MPLS Traffic Engineering (MPLS TE), що підходять методів SPAN і т.п.

Мережа SDN пропонує інтегрований і набагато більше простий у впровадженні й використанні функціонал Path Preference і Traffic Copy Contracts, реалізований засобами DNA Center і фабрики.

Як ефективно усувати інциденти? Як запобігати майбутні інциденти?

З погляду бізнесу мережа повинна забезпечувати рівень доступності і якості обслуговування, необхідний і достатній для оптимальної роботи бізнес-процесів.

Рішення цього завдання в ідеалі повинне проводитися проактивно, попереджаючи інциденти ще до їхнього виникнення. На практиці в класичній мережі це виходить рідко. За даними дослідження Cisco, 85% опитаних компаній практикують реактивний, а не проактивний підхід. Адміністратори перевантажені масою різнорідних даних, що надходять від мережевих пристроїв і обробляємих різних методів за допомогою різних інструментів, але ці дані не пропонують конкретних керівництв до дії. Важко одержати цілісний погляд на мережу, важко локалізувати корінь проблеми. У підсумку навіть реактивне усунення проблем не завжди відбувається оперативно й усе рідше влаштовує бізнес.

Мережа SDN пропонує рішення цього завдання за допомогою функціонала DNA Center. Він забезпечує цілісний погляд на корпоративну мережу й надає виводи про інциденти, пов'язаних з мережею, користувачами й додатками. Ці виводи допомагають ужити конкретних заходів для усунення причини інцидентів.

Також DNA Center проводить аналіз трендів, що відбуваються в мережі, і дає прогнози їхніх наслідків. Ця інформація допомагає впровадити проактивний, а не реактивний підхід до рішення інцидентів.

Крім того, DNA Center одержує потокову телеметрію від елементів мережевої інфраструктури й надає адміністраторові відомості про мережеві події з точністю до секунд.

У результаті мережа SDN надає:

– Ефективну допомогу в рішення інцидентів.

– Допомога в переході від реактивного до проактивному підходу в експлуатації мережі.

– Інструментарій для забезпечення високої доступності бізнес-процесів.

– Зниження ризиків збоїв бізнес-процесів.

Фабрика Software-Defined Access (SDN) являє собою новий підхід до побудови корпоративних мереж і робить великий крок уперед у порівнянні із класичними кампусними мережами. По значимості цей крок можна зрівняти з переходом від комп'ютерів з інтерфейсом командного рядка до комп'ютерів із графічним інтерфейсом користувача.

Фабрика пропонує службі IT нові потужні можливості в області дизайну, впровадження, експлуатації корпоративної мережі, реалізації політик, а також пошуку й усунення несправностей. Ці можливості дозволяють IT виконувати вимоги бізнесу швидко і якісно. З'являється можливість мінімізувати рутинну роботу, виділити більше часу на більше стратегічні, важливі завдання, що мають більше високу цінність для роботодавця, а також дають конкурентні переваги на ринку праці.

Бізнес, у свою чергу, одержує значні переваги у швидкості виконання ініціатив і рішення завдань, що опираються на мережу. В остаточному підсумку це допомагає розширити займану частку ринку, збільшити виторг. Також SDN пропонує бізнесу значне зниження ризиків збоїв бізнес-процесів, пов'язаних з «людським фактором», і принципово новий рівень безпеки інформаційного середовища.

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

– сформувати наочне візуальне подання взаємодії основних процесів, що відбуваються у системи моніторингу SDN;

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

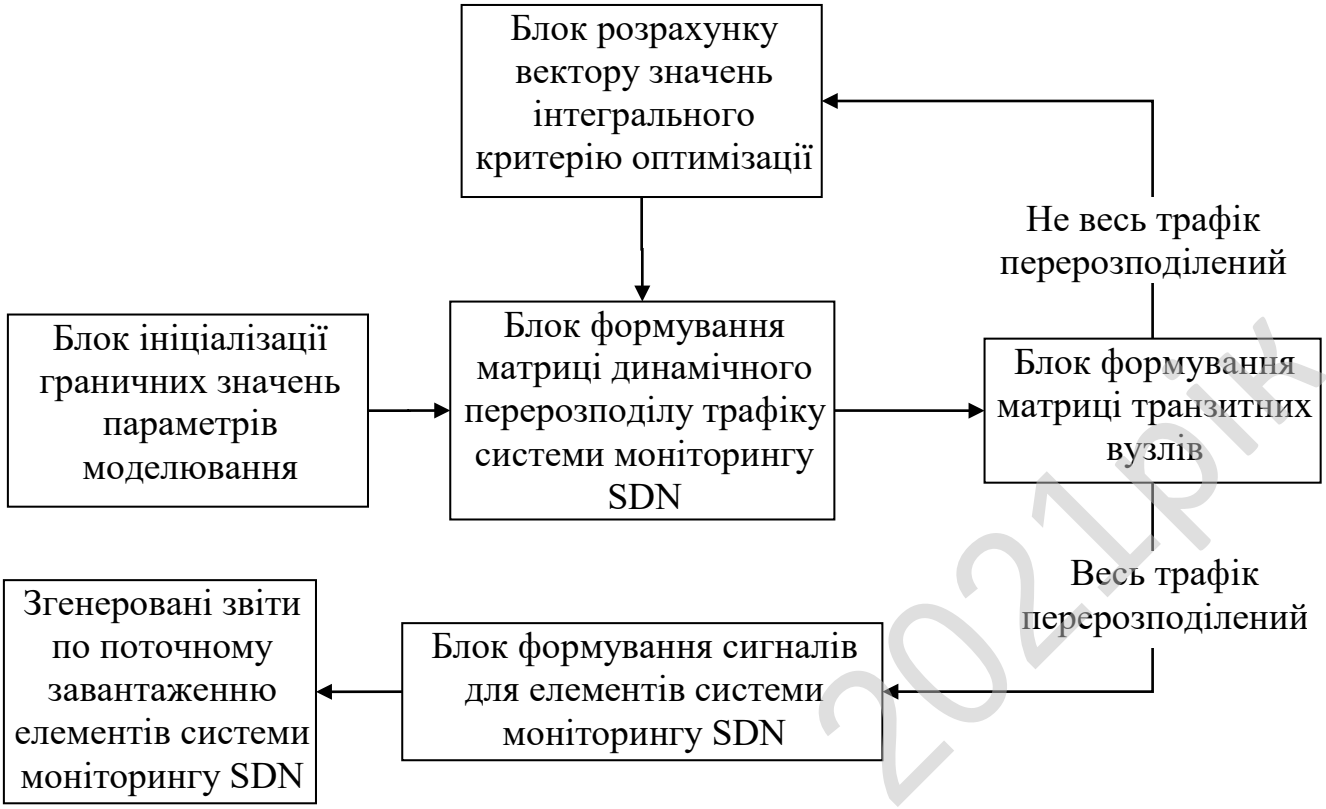

Рисунок 3.2 – Функціональна схема системи

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

– сценарій динамічного перерозподілу трафіку системи моніторингу SDN, розроблений у ході виконання бакалаврської 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 системи моніторингу SDN.

– Згенеровані звіти по поточному завантаженню елементів системи моніторингу SDN.

– Блок розрахунку вектору значень інтегрального критерію оптимізації.

– Блок ініціалізації граничних значень параметрів моделювання.

– Блок формування матриці динамічний перерозподіл трафіку системи моніторингу SDN.

Інтернет-технології прийняли широке поширення, і використовуються як основа побудови корпоративних і критично важливих застосунків, таких як WEB-вузли, вузли потокового мультимедіа-віщання й сервери віртуальних приватних мереж. Служба динамічного перерозподілу трафіку системи моніторингу SDN (ДПТМ SDN) є оптимальним і ефективним рішенням, що забезпечує масштабованість і високу відказостійкість таких застосунків як в Інтернеті, так і в інтрамережах. Служба ДПТМ SDN 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.

Служба ДПТМ SDN постачена всіма необхідним адміністраторам способами керування, у тому числі можливістю (після уведення пароля) віддалено управляти кластером з будь-якого комп'ютера в системи моніторингу SDN. 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 SDN 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 SDN дає низькі накладні витрати на обробку потоку даних і чудові можливості масштабування продуктивності, обмежені тільки пропускною здатністю підсистеми моніторингу SDN.

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

Служба ДПТМ SDN 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.

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

3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3.

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

Основні складові елементи діаграми взаємодії процесів це потоки даних:

– Репозиторії, поток сховища даних.

– Потоки зовнішні по відношенню до системи сутності.

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Потоки даних гібридні між елементами трьох попередніх типів.

Відповідно до документації основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як

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

Рисунок 3.3 – Діаграма взаємодії процесів

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1, як можна побачити система складається з наступних елементів:

– Виведення вікна системи моніторингу SDN.

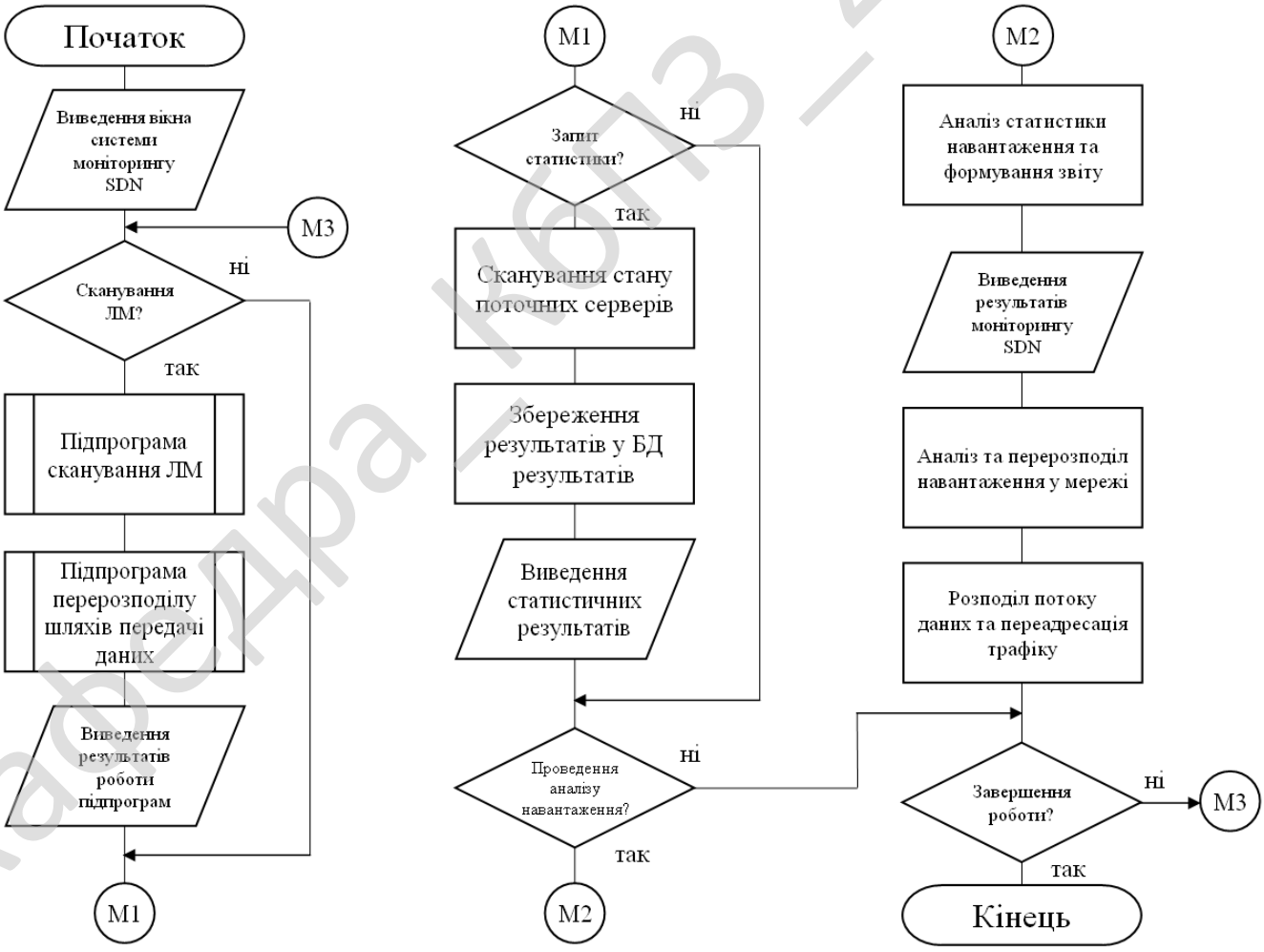

Рисунок 4.1 – Блок-схема основної програми

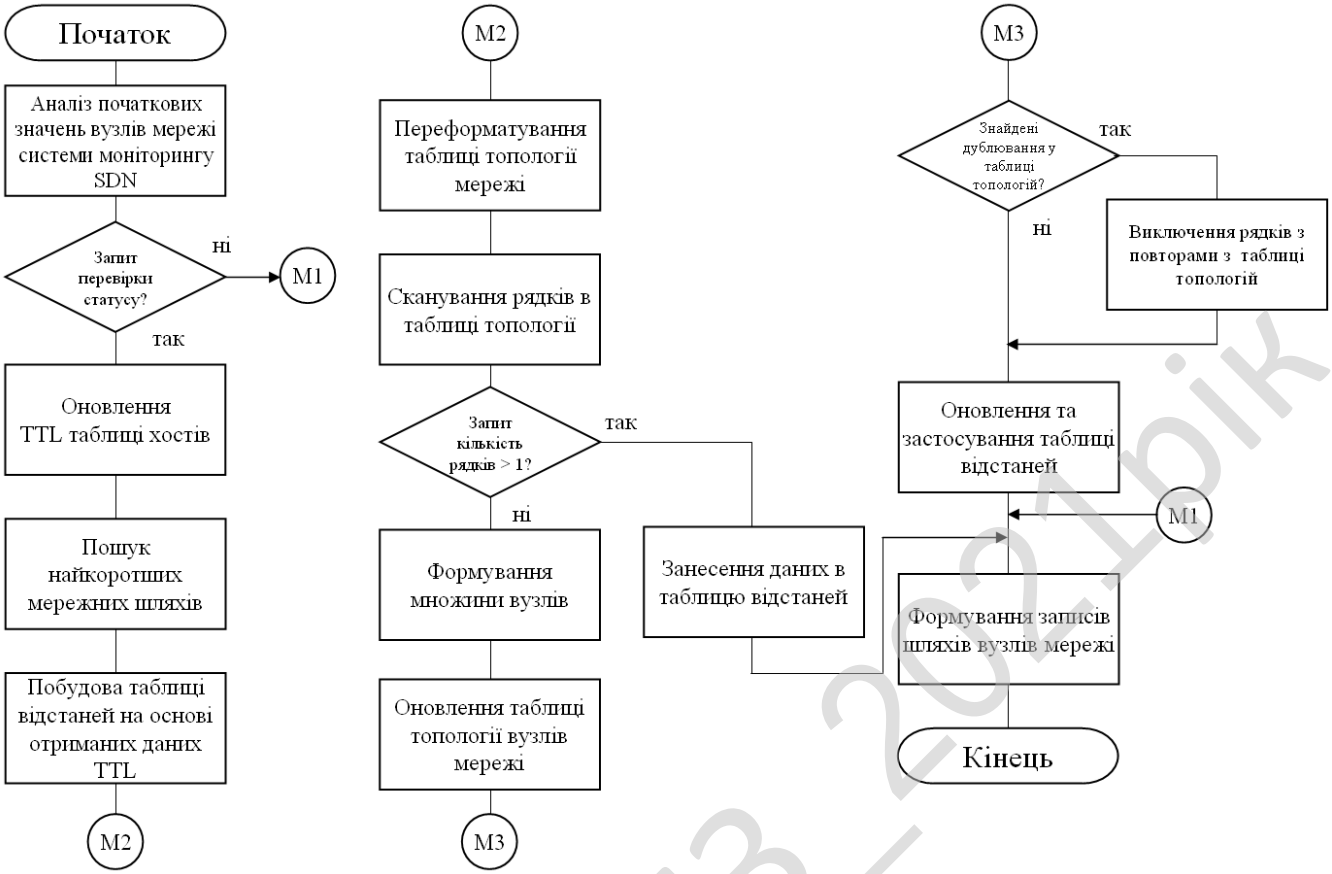

Рисунок 4.2 – Блок-схема роботи підпрограми

– Запит сканування ЛМ.

– Виклик підпрограми сканування ЛМ.

– Виклик підпрограми перерозподілу шляхів передачі даних.

– Виведення результатів роботи підпрограм.

– Запит статистики.

– Сканування стану поточних серверів.

– Збереження результатів у БД результатів.

– Виведення статистичних результатів.

– Запит проведення аналізу навантаження.

– Аналіз статистики навантаження та формування звіту.

– Виведення результатів моніторингу SDN.

– Аналіз та перерозподіл навантаження у мережі.

– Розподіл потоку даних та переадресація трафіку.

– Останній запит основного циклу – завершення роботи.

З рисунку 4.2 видно з яких елементів складається підпрограма, а саме:

– Аналіз початкових значень вузлів мережі системи моніторингу SDN.

– Запит перевірки статусу.

– Оновлення TTL таблиці хостів.

– Пошук найкоротших мережних шляхів.

– Побудова таблиці відстаней на основі отриманих даних TTL.

– Переформатування таблиці топології мережі.

– Сканування рядків в таблиці топології.

– Запит кількості рядків з послідуючою реалізацією.

– Формування множини вузлів.

– Оновлення таблиці топології вузлів мережі.

– Запит знайдення дублювання у таблиці топологій.

– Оновлення та застосування таблиці відстаней.

– Та останній елемент підпрограми формування записів шляхів вузлів мережі.

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 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
```

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
побудованій на базі технології Software-Defined Access}
    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 побудованій
        на базі технології Software-Defined Access
        }
            procedure Refresh;
{ містить списки всіх комп'ютерів в  мережі побудованій на базі технології
Software-Defined Access}
        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
побудованій на базі технології Software-Defined Access немає }
        function FindComputer(Name: TString): TString;
{ Процедура ListComputers копіює список всіх комп'ютерів в мережі побудованій на
базі технології Software-Defined Access
        у об'єкті TStrings}
        procedure ListComputers(Strings: TStrings);
{ Процедура ListNetwork копіює відсортований в алфавітному порядку список всіх
робочих груп і комп'ютерів в мережі побудованій на базі технології Software-
Defined Access. 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;
```

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

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

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

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до

серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

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

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

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

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

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

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

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

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

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

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

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

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

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

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

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

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

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

Redmine – вільне серверне ПЗ для управління проектами та відстежування помилок. До системи входить календар-планувальник та діаграми Ганта для візуального представлення ходу робіт за проектом та строків виконання. Redmine написано на мові Ruby і є ПЗ розробленим з використанням відомого веб-фреймворку Ruby on Rails, що означає легкість в розгортанні системи та її

адаптації під конкретні вимоги. Для кожного проекту можна вести свої вікі та форуми.

Функціональні можливості:

– Ведення декількох проектів.

– Гнучка система доступу з використанням ролей.

– Система відстеження помилок.

– Діаграми Ганта та календар.

– Ведення новин проекту, документів та управління файлами.

– Сповіщення про зміни за допомогою RSS-потоків та електронної пошти.

– Власна Wiki для кожного проекту.

– Форуми для кожного проекту.

– Облік часових витрат.

– Налаштування власних (custom) полів для задач, затрат часу, проектів та користувачів.

– Легка інтеграція із системами керування версіями (SVN, CVS, Git, Mercurial, Bazaar и Darcs).

– Створення записів про помилки на основі отриманих листів

– Підтримка LDAP автентифікації.

– Можливість самореєстрації нових користувачів.

– Багатомовний інтерфейс (у тому числі українська мова).

– Підтримка СКБД: MySQL, PostgreSQL, SQLite.

Діаграма Ганта (*Gantt chart*, також стрічкова діаграма, графік Ганта) – це популярний тип діаграм, який використовується для ілюстрації плану, графіка робіт за будь-яким проектом. Є одним з методів планування та управління проектами.

Діаграма Ганта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі. Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і

тривалості завдання. На деяких діаграмах Ганта також показується залежність між завданнями.

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню, заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні».

Часто діаграма Ганта використовується спільно з таблицею зі списком робіт, рядки якої відповідають окремо взятій задачі, зображеній на діаграмі, а стовпці містять додаткову інформацію про задачу.

Система відстеження помилок Багтрекер – прикладна програма для допомоги розробникам програмного забезпечення (програмістам, тестувальникам тощо) враховувати і контролювати помилки, знайдені у програмах, питання щодо функціональності, рішення та оновлення, побажання користувачів, а також стежити за процесом їх виконання.

Кожному, хто розробляв програмні продукти, добре знайоме співвідношення «20/80» – останні 20 % роботи тривають 80 % часу.

Як це не парадоксально, але нічого дивного в цій пропорції немає, адже саме на завершальній стадії починається тестування проекту, коли виявляються помилки, і що більший проект, то більше буде знайдено помилок.

Водночас досить часто виявляється, що більшість цих помилок були відомі та могли бути виправлені з меншими витратами на попередніх стадіях роботи, але не були вчасно описані, а потім загубилися серед інших важливих завдань.

Отже, система відстеження помилок у найпростішому варіанті – це процес, що включає в себе виявлення помилки, її опис, виправлення і перевірку цього виправлення, тобто процес «стеження» за багом протягом всього як його життєвого циклу, так і життєвого циклу розробки в цілому.

Сукупність інформації про дефект. Головний компонент такої системи – база даних, що містить відомості про виявлені дефекти. Ці відомості можуть включати в себе:

– номер (ідентифікатор) дефекту;

– хто повідомив про дефект;

– дата і час виявлення дефекту;

– версія продукту, в якій виявлено дефект;

– серйозність (критичність) дефекту та пріоритет рішення;

– опис кроків для відтворення дефекту (неправильної поведінки програми);

– відповідальний за усунення дефекту;

– обговорення можливих рішень та їх наслідків;

– поточний стан виправлення дефекту;

– версії продукту, в якій дефект виправлений.

Крім того, розвинені системи надають можливість прикріплювати файли, які допомагають описати проблему, наприклад, дамп пам'яті або скріншот.

Використання. Основна перевага систем відстеження помилок полягає в забезпеченні чітких централізованих оглядів, запитів на розробку (включаючи помилки і виправлення) та їх стан. У корпоративному середовищі, системи відстеження помилок можуть бути використані для генерації звітів по продуктивності програмістів виправлення помилок. Однак, це може іноді приводити до неточних результатів, тому що різні помилки можуть мати різні ступені пріоритету та серйозності, що пов'язано з складністю їх фіксації.

Життєвий цикл дефекту. Як правило, система відстеження помилок використовує той чи інший варіант «життєвого циклу» помилки, стадія якого визначається поточним станом помилки.

Типовий життєвий цикл дефекту:

1. Новий – дефект зареєстрований тестувальником.

2. Призначений – призначений відповідальний за виправлення дефекту.

3. Дозволений – дефект переходить назад у сферу відповідальності тестувальника. Як правило, супроводжується резолюцією, наприклад:

– Виправлено (виправлення включені у версію таку-то).

– Дубль (повторює дефект, що вже знаходиться в роботі).

– Не виправлено (працює відповідно до специфікації, має занадто низький пріоритет, виправлення відкладено до наступної версії тощо).

– «В мене все працює» (запит додаткової інформації про умови, в яких дефект проявляється).

4. Далі тестувальник проводить перевірку виправлення, залежно від чого дефект або знову переходить у стан «Призначений» (якщо він описаний як виправлений, але не виправлений), або у стан «Закрито».

5. Відкрито повторно – дефект знайдено знову в іншій версії.

Система може надавати адміністраторові можливість налаштування користувачі, які можуть переглядати і редагувати помилки залежно від їх стану, переводити їх в інший стан або видаляти.

У корпоративному середовищі, система відстеження помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає достатньо точних результатів через те, що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми прямо не стосується складності її усунення.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Blowfish, який є симетричним алгоритмом шифрування, тобто таким, у якому ключ шифрування дорівнює ключу дешифрування. Він є мережею Фейштеля, у якій кількість ітерацій дорівнює 16. Довжина блоку дорівнює 64 бітам, ключ може мати будь-яку довжину в межах 448 біт. Хоча перед початком будь-якого шифрування виконується складна фаза ініціалізації, саме шифрування даних виконується досить швидко.

Алгоритм призначений в основному для застосунків, у яких ключ міняється нечасто, до того ж існує фаза початкового рукостискання, під час якої

відбувається автентифікація сторін і узгодження загальних параметрів і секретів. При реалізації на 32-бітних мікропроцесорах з більшим кешем даних Blowfish значно швидше DES.

Алгоритм складається із двох частин: розширення ключа й шифрування даних. Розширення ключа перетворює ключ довжиною, принаймні, 448 біт у кілька масивів підключів загальною довжиною 4168 байт.

В основі алгоритму лежить мережа Фейштеля з 16 ітераціями. Кожна ітерація складається з перестановки, що залежить від ключа, і підстановки, що залежить від ключа й даних. Операціями є XOR і додавання 32-бітних слів.

Blowfish використовує велику кількість підключів. Ці ключі повинні бути обчислені заздалегідь, до початку будь-якого шифрування або дешифрування даних. Елементи алгоритму:

1. P – масив, що складається з вісімнадцяти 32-бітних підключів: P_1, P_2, ..., P_{18}.

2. Чотири 32-бітних S-boxes с 256 входами кожний. Перший індекс означає номер S-box, другий індекс – номер входу.

3. $S_{1,0}$, $S_{1,1}$, … $S_{1,255}$;

4. $S_{2,0}$, $S_{2,1}$, … $S_{2,255}$;

5. $S_{3,0}$, $S_{3,1}$, … $S_{3,255}$;

6. $S_{4,0}$, $S_{4,1}$, … $S_{4,255}$;

Шифрування

Входом є 64-бітний елемент даних X, що ділиться на дві 32-бітні половини, X_l і X_r.

$X_\text{l} = X_\text{l}$ XOR P_i

$X_\text{r} = F\,(X_\text{l})$ XOR X_r

Swap X_l and X_r

Функція F

Розділити X_l на чотири 8-бітних елементи A, B, C, D.

$F\,(X_\text{l}) = ((S_{1,A} + S_{2,B} \bmod 2^{32})$ XOR $S_{3,C}) + S_{4,D} \bmod 2^{32}$

Дешифрування відрізняється від шифрування тим, що P_i використовуються у зворотному порядку.

Генерація підключів

Підключи обчислюються з використанням самого алгоритму Blowfish.

1. Ініціалізувати перший *P*-масив і чотири *S*-boxes фіксовані рядки.

2. Виконати операцію XOR P_1 з першими 32 бітами ключа, операцію XOR P_2 із другими 32 бітами ключа й т.д. Повторювати цикл доти, поки весь *P*-масив не буде побітово складний з усіма бітами ключа. Для коротких ключів виконується конкатенація ключа із самим собою.

3. Зашифрувати нульовий рядок алгоритмом Blowfish, використовуючи підключи, описані в пунктах (1) і (2).

4. Замінити P_1 і P_2 виходом, отриманим на кроці (3).

5. Зашифрувати вихід кроку (3), використовуючи алгоритм Blowfish з модифікованими підключами.

6. Замінити P_3 і P_4 виходом, отриманим на кроці (5).

7. Продовжити процес, заміняючи всі елементи *P*-масиву, а потім всі чотири *S*-boxes, виходами відповідним чином модифікованого алгоритму Blowfish.

Для створення всіх підключів потрібна 521 ітерація.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

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

– Функціональних кнопок ПЗ: Формування сигналів; Статистичного навантаження; Аналізу статистичного навантаження; Налаштування; Звіт роботи.

– Верхнього меню: Мережа; Формат; Налаштування; Довідка.

– Розділу обрання вузла мережі.

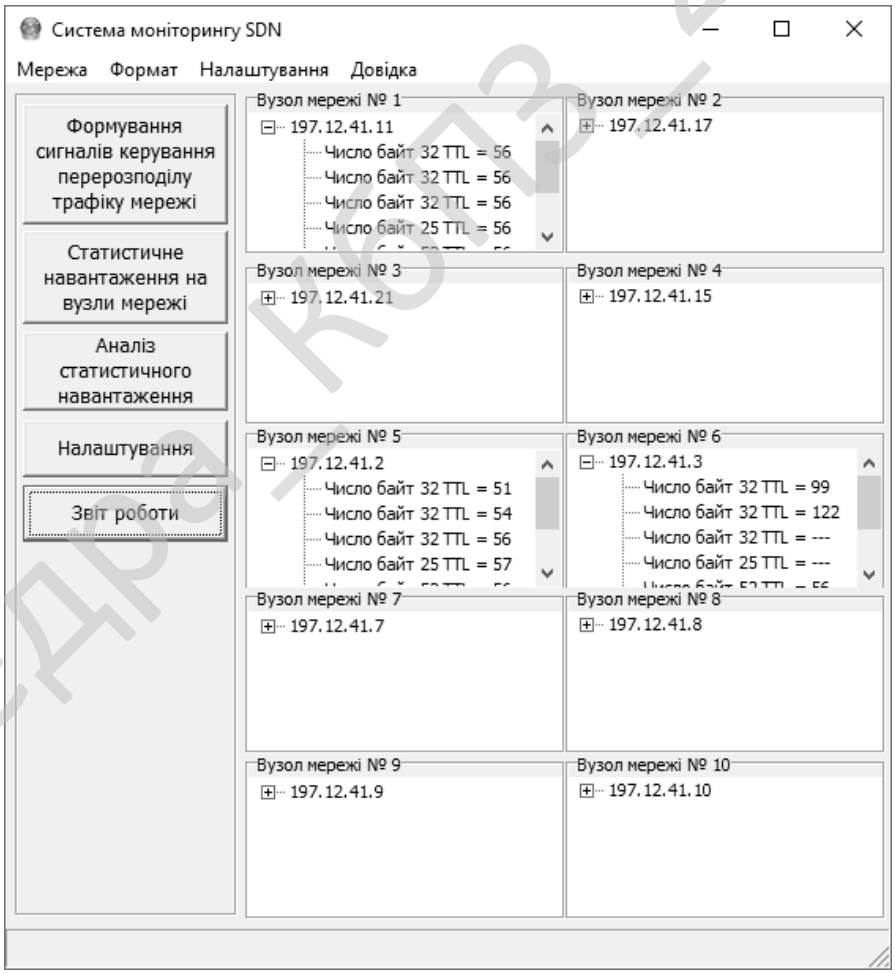

Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення. Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

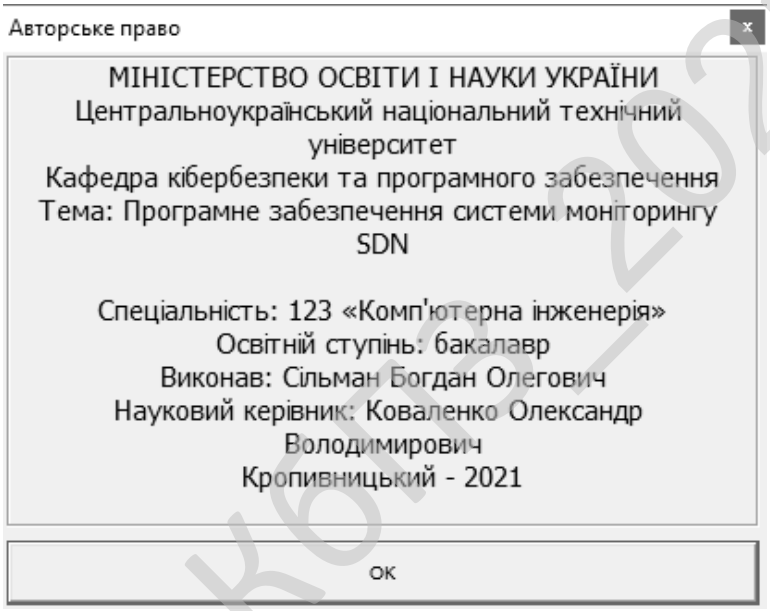

Рисунок 5.2 – Авторське право

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної бакалаврської роботи, призначено для системи моніторингу SDN.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем моніторингу SDN.

– Досліджена система моніторингу SDN.

– На основі отриманих результатів досліджень створена програмна реалізація системи моніторингу SDN.

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання моніторингу SDN.

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 Embarcadero Delphi. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи моніторингу SDN. 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.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Blowfish.

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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Семенов А. Д. Идентификация объектов управления: Учебн. Пособие / А.Д.Семенов, Д.В.Артамонов, А.В.Брюхачев - Пенза: Изд-во Пенз. гос. ун-та, 2003.- 211 с.

2. Семенов С.Г. Анализ методов прогнозирования в телекоммуникационных сетях автоматизированных систем управления / С.Г.Семенов // Збірник наукових праць «Системи управління, навігації та зв'язку», - К.:ЦНДІ навігації і управління, - 2008.-Вип. 2(6) .- С.134-137

3. Семенов С.Г. Математическая модель процесса доставки информационных пакетов в компьютерной сети системы критического применения / С.Г.Семенов, И.В.Ильина // Науково-технічний журнал «Радіоелектронні і комп'ютерні системи» Х.:ХАІ, - 2008.-Вип. 1(28) - С.162-165

4. Семенов С.Г. Оптимизация трафика на основе сбалансированной загрузки информационно-телекоммуникационной сети // Системи обробки інформації. – Х.: ХВУ, 2004. – № 8(36). – С.206-210

5. Семенов С.Г. Сравнительные исследования методов идентификации трафика в телекоммуникационной сети для повышения оперативности передачи данных / С.Г.Семенов, Е.В.Мелешко // Науково-технічний журнал "Прикладная радиоэлектроника" Том 9 №3. - Х: ХНУРЕ. – 2010. – С.444-448.

6. Семенов С.Г. Сравнительные исследования и анализ алгоритмов управления очередями в многопротокольных узлах связи телекоммуникационной сети / С.Г.Семенов, Е.В.Мелешко, В.В.Босько // «Новітні технології – для захисту повітряного простору» Матеріали шостої наукової конференції 14-15 квітня. – Х: ХУПС, 2010.– С.132.

7. Semenov S. The method of processing and identification of telecommunication traffic based on BDS-tests / S. Semenov, A.Smirnov., E.Meleshko // The book of materials International Conference «Statistical Methods of Signal and

Data Processing (SMSDP-2010)» –Kiev, Ukraine, National Aviation University "NAU-Druk" Publishing House, October 13-14, 2010. – С.166-168. – engl.

8. Семенов Ю.А. Сети Інтернет. Архитектура и протоколы / Ю.А. Семенов. – М.: Блик плюс, 1998. – 424 с.

9. Смирнов А.А. Разработка методики оценки среднего времени обслуживания информационных пакетов в телекоммуникационной сети / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи управління, навігації та зв'язку. – Київ: ДП «Центральний науково-дослідний інститут навігації і управління», 2009. – Вип. 2(10). – С.162-165.

10. Смирнов А.А. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

11. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011. – 193-195 с.

12. Современные телекоммуникации. Технологии и экономика / [В.Л. Банкет, О.В. Бондаренко, П.П. Воробьенко и др.]; под ред. С.А. Довгого. – М.: Эко-Трендз, 2003. – 320 с.

13. Смірнова Т.В., Дрєєв О.М., «Інформаційна технологія оптимізації технологічного процесу відновлення та зміцнювання поверхонь валів зі сталі як хмарний сервіс» у *Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка*. – Х. : Вид. Рожко С.Г. 2019. С. 92-107.

14. Т.В.Смірнова, Л.І. Поліщук, «Дослідження хмарних технологій як сервісів для системи інженерних розрахунків» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. С. 106-121.

15. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Експертна система оптимізації процесу відновлення та зміцнення поверхонь деталей типу «вал» електродуговим напиленням», *Системи управління, навігації та зв'язку*, № 2 (54). с. 149-154, 2019.

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

17. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, Є.К. Солових, «Методи оптимізації технологічних процесів відновлення сталевих покриттів», *Shipbuilding & marine infrastructure / Суднобудування і морська інфраструктура* № 1 (11). с. 48-57, 2019.

18. Т. В. Смирнова, А. А. Смирнов, А. Н. Дреев, А. В. Дудан, «Оптимизация технологического процесса восстановления и упрочнения поверхностей с заданными характеристиками в виде облачного сервиса», Вестник Полоцкого государственного университета. Серия В, Промышленность. Прикладные науки. **Республика Беларусь** - 2020. - № 3. - С. 50-61.Режим доступу: http://elib.psu.by:8080/handle/123456789/24988

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

20. Т.В.Смірнова, «Формалізація та реалізація структури технологічного процесу електродугового напилення для оптимізаційної експертної системи», *Технічні науки та технології*. № 1(19). С. 104-113. 2020. (Категорія Б)

21. T. Smirnova, I. Smirnov, A. Lopata, L. Lopata «Improvement of functional properties of gas-thermal coatings by electro-contact treatment», *Problems of Tribology*, Vol. 25, № 1/95, P. 41-48. 2020.

22. Т.В. Смірнова «Формування евристичних правил, бази знань та формалізація структури й правил технологічного процесу для оптимізаційної

хмарної інформаційної системи», *Системи управління, навігації та зв'язку*, № 2 (60). с. 101-104, 2020. (Категорія Б)

23. Т.В. Ворона, Т.И. Ивченко, В.Я. Николайчук, «Повышение износотойкости стальных газотермических покрытий электроконтактной обработкой», *Сучасні енергетичні установки на транспорті, технології та обладнання для їх обслуговування,* Херсон 28-29 вересня 2017 року, Херсонська державна морська академія, 2017, с. 197-198**.**

24. В.Н. Лопата, М.С. Агеев, Т.В. Ворона, «Переход от гальванической технологии к газотермическим технологиям при получении антикоррозионных покрытий», «*Modern questions of production and repair in industry and in transport»* February 10–16, Brno, Czech Republic, 2018, pp. 245-249.

25. А.В. Дудан, М.С. Агеев, Т.В. Ворона, «Переход от гальванической технологии к газотермическим технологиям при получении антикоррозионных покрытий», «*Инновационные технологии в машиностроении*», ИннТехМаш (Новополоцк 19-20 апреля 2018), 2018, с. 186-192.

26. Т.В. Смірнова, О.А. Смірнов, О.М. Дрєєв, С.А. Смірнов, «Використання хмарних експертних систем в сфері інформаційного забезпечення обробки поверхні деталей», *Комп'ютерна інженерія і кібербезпека: досягнення та інновації*, м. Кропивницький. 27-29 листопада, 2018, с. 111-113

27. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов «Захист даних у інформаційній технології відновлення поверхонь деталей у вигляді хмарної платформи як послуги», *Проблеми кібербезпеки інформаційно-телекомунікаційних систем*, м. Київ, 11-12 квітня, 2019, с. 221-224.

28. Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, «Хмарний сервіс інформаційної технології оптимізації технологічного процесу відновлення та зміцнювання поверхонь зі сталі», *Інформаційна безпека та інформаційні технології, Information Security and Information Technologies*, м. Харків, 24-25 квітня, 2019, с. 36

29. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Побудова хмарної експертної системи оптимізації технологічного процесу електродугового напилення сталевих покриттів», *Міжнародний форум з інформаційних систем і технологій INFOS-2019,* м. Харків, 24-27 квітня, 2019, с. 95-98.

30. Т. В. Смирнова, А. А. Смирнов, Е. К. Соловых «Разработка математической модели и программного обеспечения для оптимизации режима электроконтактной обработки газотермических покрытий», *Комплексне забезпечення якості технологічних процесів та систем,* том 2, м. Чернігів, 14 - 16 травня, 2019, с. 78-79.

31. Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, О.А. Смірнов, «Експертна система інформаційної технології оптимізації абстрактного технологічного процесу як хмарного сервісу», *Інформаційні технології: Наука, техніка, технологія, освіта, здоров'я. MicroCAD-2019,* м. Харків, 15-17 травня, 2019, с. 195.

32. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Формування абстрактних експертних систем на основі досліджень відомих експертних систем», *XXI Міжнародний науково-практичний семінар «Комбінаторні конфігурації та їх застосування»*, м. Кропивницький, 17-18 травня, 2019, с. 143-147.

33. Т.В. Смірнова, О.А. Смірнов, «Захист даних у інформаційній технології відновлення поверхонь деталей у вигляді хмарної платформи як послуги», *Інформація, комунікація, суспільство – 2019*, см.т. Чинадієво, 16-18 травня, 2019, с. 25-26

34. Т.В. Смирнова, А.Н. Дреев, А.А. Смирнов, «Информационная технология оптимизации технологического процесса восстановления и упрочнения поверхностей в виде облачного сервиса», *Modern Information, Measurement And Control Systems: Problems And Perspectives (MIMCS 2019)*, Baku, Azerbaijan, 01-02 July, 2019, p. 282.

35. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Огляд відомих експертних систем оптимізації технологічних процесів», *Стратегія якості в промисловості і освіті,*м. Варна, Болгарія, 3-6 червня, 2019, с.442-444

36. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Формалізація та узагальнення інформаційної моделі технологічних операцій зміцнення та відновлення сталевих поверхонь», *Інтелектуальні системи та інформаційні технології*, м.Одеса, 19-24 серпня, 2019, с. 211-213.

37. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Формування інформаційної моделі технологічного процесу». *V Всеукраїнська науково-практична Інтернет-конференція «Електронні та мехатронні системи: теорія, інновації, практика»*, м. Полтава, 8 листопада, 2019, с. 87-91.

38. T.V. Smirnova, M.S. Chernovol, Ageev M. «Study of the spraying process and the influence of its factors on the properties of electric arc spraying coatings». *Materials of the 20th international scientific and technical seminar «Modern questions of production and repair in industry and in transport»*, *Tbilisi, Georgia*, 23-28 *March* 2020, с. 201-205.

39. Т.В.Смірнова, Є.К. Солових, О.А.Смірнов, «Дослідження стандартів забезпечення кібербезпеки хмарних технологій як сервісів», *X міжнародна науково-технічна конференція «ITSEC»*, Єгипет, м. Шарм-ель-Шейх, 19-24 березня, 2020. с. 36.

40. Т.В.Смірнова, Л.І. Поліщук, О.А.Смірнов, «Аналіз хмарних технологій як сервісів», *XIII всеукраїнська науково-практична конференція «Комп'ютерні інтелектуальні системи та мережі (КІСМ-2020)*». м. Кривий Ріг. 24-26 березня, 2020, С. 210-212.

41. Т.В. Смірнова, Л.І. Поліщук, О.М. Дрєєв, «Застосування сервісу CAEaaS як системи інженерних розрахунків з використанням хмарних технологій», *II Міжнародна науково-практична конференція «Інформаційна безпека та інформаційні технології, Information Security and Information Technologies»*, м. Кропивницький, 2-3 квітня, 2020, с. 52.

42. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, Є.К. Солових, «Інформаційна структура технологічного процесу електродугового напилення». *Всеукраїнська науково-технічна конференція «Стан, досягнення та перспективи інформаційних систем і технологій»*, м. Одеса, 21-22 квітня, 2020, с. 184-186.

43. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, Є.К. Солових, «Реалізація інформаційної структури технологічного процесу електродугового напилення», *XXII Міжнародний науково-практичний семінар «Комбінаторні конфігурації та їх застосування»*, м. Кропивницький, 15-16 травня, 2020, с. 158-162.

44. Т.В. Смірнова, А.В. Щербань, Ю.Ю. Моторін, О.А. Смірнов, «Перспективні напрямки забезпечення кібербезпеки сервісу CAEaaS», *VI Всеукраїнська науково-практична конференція «Перспективні напрями захисту інформації»*, м. Одеса, 2-6 вересня 2020, с. 58-59

45. Столлингс В. Современные компьютерные сети / Вильям Столлингс.– СПб.: Питер, 2003. – 778 с.

46. Сэломон Д. Сжатие данных, изображений и звука / Д. Сэломон.– М.: Техносфера, 2004. – 368 с.

47. Таненбаум Э. Компьютерные сети / Эндрю Таненбаум; пер. с англ. А. Леонтьев. – СПб.: Питер, 2002. — 848 с.

48. Телекоммуникационные системы и сети: учебное пособие. В 3 томах / [В.В. Величко, Е.А. Субботин, В.П. Шувалов, А.Ф. Ярославцев]; под ред. В.П. Шувалова. – М.: Горячая линия-Телеком, 2005, т. 3 – 592 с.

49. Уолрэнд Дж. Телекоммуникационные и компьютерные сети / Дж. Уолрэнд. – М.: Постмаркет, 2001. – 480 с.

50. Хайкин С. Нейронные сети: полный курс / С. Хайкин.– М.:Вильямс, 2006. – 1103 с.

51. Хаусли Т. Системы передачи и телеобработки данных: пер. с англ. / Т. Хаусли; под ред. Ю.М. Мартынова.– М.: Радио и связь, 1994. – 452 с.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи моніторингу SDN.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 204-02 від 28.12.2020 року).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи моніторингу SDN.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи 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;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи моніторингу SDN;

– 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 XP/Vista/7/8/10 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10.

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 Embarcadero Delphi.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 83 аркуші.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 5.06.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ Коваленко О.В.

Програмне забезпечення системи моніторингу SDN

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2021 року

Файл Networks.pas – робота з системою моніторингу SDN

```
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 системи моніторингу SDN

```
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 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

// 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      : '  + ' Hi'  );
      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.
```