Київський національний університет імені Тараса Шевченка

Факультет радіофізики, електроніки та комп'ютерних систем

Кафедра комп'ютерної інженерії

Програмний комплекс для організації дистанційного доступу працівника до свого службового комп'ютера у період карантиннихобмежень і складного оперативного становища

Кваліфікаційна робота бакалавра
студентки 4 року навчання
спеціальності123 «Комп`ютерна інженерія»
Ганни КУШНАРЕНКО

_____

Науковий керівник
Доцент кафедри комп'ютерної інженерії
Сергій ЗАГОРОДНЮК

_____

Рецензент
Доцент кафедри геоінформатики ННІ
"Інститут геології"
кандидат ф.-м. наук
Всеволод ДЕМИДОВ

_____

До захисту допускаю
Протокол засідання кафедри від
"___"_____2022 р. №____

Завідувач кафедри
кандидат фіз.-мат. наук, доцент
Юрій БОЙКО

КИЇВ 2022

РЕФЕРАТ

Кваліфікаційна робота за об’ємом складає 47 сторінок, містить 33 ілюстрації, 5 таблиць, 11 джерел посилань.

Об’єктом роботи є процес здійснення дистанційного доступу до віддаленого комп’ютера .

Предметом роботи є розробка та налаштування програмного комплексу для здійснення захищеного віддаленого доступу за протоколом TCP.

Метою роботи є розробка мережевого програмного комплексу для організації дистанційного доступу працівника до свого службового комп'ютера у період карантинних обмежень.

Інструменти виконання: ОС Windows 10, ОС Linux (Ubuntu), Visual Studio 2022 RC, SSH-client Putty, WireShark, Bitvise SSH.

Результати роботи: розроблено мережевий програмний комплекс мовою C# у вигляді служби Windows за допомогою якого можна здійснювати віддалений доступ до комп’ютера.

Ключові слова: IP-АДРЕСА, TCP-ПОРТ, TCP-СОКЕТ, СЛУЖБА, ТУНЕЛЬ, СЕРВЕР, КЛІЄНТ.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ОС – Операційна система

TCP – Transmission Control Protocol

RDP – Remote Desktop Protocol

IANA – Internet Assigned Numbers Authority

FTP – File Transfer Protocol

DNS – Domain Name System

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

ICMP – Internet Control Message Protocol

ARP – Address Resolution Protocol

DHCP – Dynamic Host Configuration Protocol

RFB – Remote Framebuffer

UDP – User Datagram Protocol

OSI – The Open Systems Interconnection model

SSH – Secure SHell — «безпечна оболонка»

IP – Internet Protocol

GUI – Graphical user interface

ВСТУП

Ситуація, коли потрібний терміновий доступ до домашнього комп'ютера або комп'ютера внутрішньої корпоративною мережі, є доволі поширеною в сучасному світі, особливо у період карантинних обмежень. Домашня мережа або локальна мережа підприємства завжди налаштовані як внутрішня приватна мережа NAT-маршрутизатора, що повністю приховує внутрішню інфраструктуру підприємства, установи або приватного помешкання від зовнішнього користувача. Зрозуміло, що отримати прямий доступ до робочого комп'ютера або сервера, що знаходиться в приватній мережі NAT-маршрутизатора, принципово неможливо. Для доступу саме до віддаленого робочого столу існують зовнішні закриті програми (TeamViewer, AnyDesk, тощо), але ці програми, наприклад, не дозволяють приєднатись на довільну TCP-службу, наприклад SSH-сервер або HTTPS-сервер.

Мета випускної кваліфікаційної роботи:

Створити програмний комплекс, що дозволяє авторизованому працівнику установи, підприємства, або власнику приватного житла заздалегідь налаштувати всередині відповідної приватної мережі довільну мережеву службу, доступною за будь-яким протоколом 7 рівня моделі OSI , що передбачає доступ лише по одному TCP-порту (наприклад SSH, HTTP, HTTPS, MySQL, SMTP, VNC, RDP), після чого мати можливість приєднатися до цієї довільної служби, перебуваючи далеко за межами відповідної приватної мережі і у такий спосіб обійти бар'єр NAT-маршрутизатора підприємства, провайдера або мобільного оператора.

Внутрішня або зовнішня частина програмного комплексу повинені бути розроблені для операційних систем Windows та Linux. Відповідно до цієї умови потрібно реалізувати чотири варіанта:

1. Внутрішня частина на ОС Windows, зовнішня частина на ОС Linux;
2. Внутрішня частина на ОС Linux, зовнішня частина на ОС Windows;
3. Обидві частини на ОС Windows;
4. Обидві частини на ОС Linux.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Мережеві порти

Кожен пристрій у мережі TCP/IP повинен мати IP-адресу, яка буде ідентифікувати його в даній мережі. Але оскільки на кожному пристрої з великою вірогідністю буде запущена більш ніж одна програма або служба однієї IP-адреси для декількох застосунків, які будуть обмінюватися інформацією в мережі з іншими хостами, буде недостатньо. Саме через дану потребу в ідентифікації програм та служб, запущених на пристрої, були створені мережеві порти. Тобто, використання портів дозволяє комп'ютерам або пристроям запукати декілька служб або програм [1].

У протоколах транспортного рівня, а саме TCP або UDP порт вважається системним ресурсом, що має номер. На одному хості один порт може бути зайнятий тільки однією програмою або службою та мати номер у межах від 0 до 65536. Також деякі номери портів є зарезервованими для протоколів прикладного рівня від 0 до 1023 , наприклад порт 22 – для протоколу SSH. Про інші зарезервовані протоколи можна дізнатися з таблиці 1.1.

Порти 1025 – 49151 – напівзареєстровані порти, вони можуть бути використані для послуг IANA (Internet Assigned Numbers Authority).

Порти 49152 – 65535 – можуть бути використані клієнтськими програмами.

Порт	Опис
0	зарезервовано
20	Протокол FTP (data)
21	Протокол FTP (codd)
22	Протокол SSH
23	Протокол Telnet
53	Протокол DNS
80	Протокол HTTP
443	Протокол HTTPS
6969	BitTorrent
33433	Traceroute

Таблиця 1.1 Деякі зарезервовані загальновідомі порти [1]

1.2 Мережеві сокети

Мережевий сокет (Network Socket) можна розглядати як кінцевий пункт призначення даних у двосторонній взаємодії хостів у мережі. Даний сокет має власну адресу, що складається з поєднання IP-адреси та номеру порту. Мережний сокет доставляє інформацію, яка надходить з мережі, програмному процесу відповідно до адреси сокету (Рис.1.1). Кожний процес має можливість створювати сокет для прослуховування або на відправку даних та прив’язати його до будь-якого порту в системі [2].

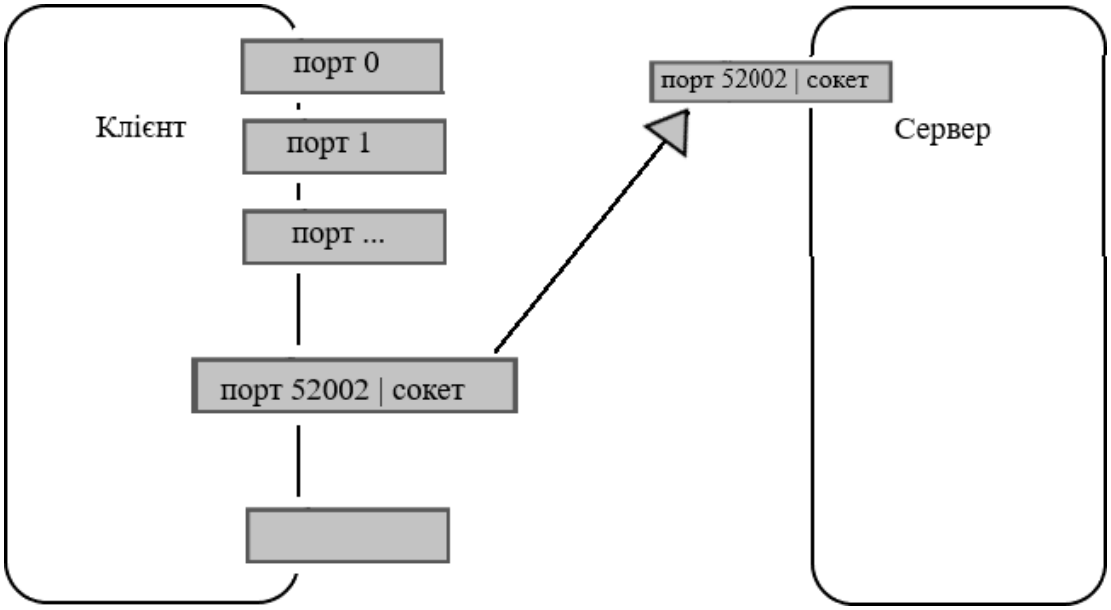


Рис.1.1 Схема роботи сокетів

1.3 Стек протоколів TCP/IP

TCP/IP – це велике сімейство протоколів, назване на честь двох найважливіших членів цього сімейства. Стек протоколів TCP/IP поділяється на 4 рівня, на відміну від моделі OSI яка поділяється на 7 рівнів. Для розуміння співвідношення рівнів протоколів моделі OSI та стеку протоколів TCP/IP можна скористатися наступною таблицею :

Модель OSI	TCP/IP модель	
Прикладний рівень	Прикладний рівень	SMTP, POP3, Telnet, SSH, DNS, HTTPS, HTTP, FTP
Рівень презентації		
Рівень сеансу		
Транспортний рівень	Транспортний рівень	TCP, UDP
Мережевий рівень	Інтернет	IP, ICMP, ARP, DHCP
Рівень даних	Рівень мережевого доступу	Ethernet
Фізичний рівень		

Таблиця 1.2 Співвідношення рівнів моделі OSI та TCP/IP [1]

1.4 Транспортний рівень. Протокол TCP та його особливості

Протоколи транспортного рівня організовують транспортування даних для прикладних процесів, таких як SSH, HTTP, тощо. TCP (Transmission Control Protocol) – найбільш широко використовуваний протокол для передачі даних у мережах, таких як Інтернет наприклад.
Особливості протоколу TCP [2] :

- TCP є надійним протоколом. Це означає те, що відправник сегментів завжди має чітке уявлення про те, чи досягнув сегмент до пункту призначення, чи йому потрібно надсилати сегмент повторно.
- TCP гарантує, що дані досягають цільового призначення в тому ж порядку, в якому вони були надіслані.
- TCP вимагає попередньо встановленого з’єднання між віддаленими хостами для того щоб почати обмінюватися безпосередньо корисними даними.
- TCP забезпечує механізм перевірки помилок та відновлення з’єднання.
- Забезпечує наскрізний зв’язок.
- Забезпечує контроль потоку обміну повідомленнями та якість обслуговування.
- TCP працює в режимі клієнт – сервер.
- Забезпечує повнодуплексний сервер, тобто може виконувати ролі як одержувача, так і відправника.

Як зазначалося вище, зв’язок між двома віддаленими хостами здійснюється за допомогою номерів портів.
Заголовок TCP-сегмента має наступні прапори для керування з’єднанням :

- NS – біт Nonce Sum використовується процесом сигналізації явного сповіщення про перевантаження.
- CWR – коли хост отримує пакет із встановленим бітом ECE, він встановлює Congestion Windows Reduced, щоб підтвердити отримання ECE.
- ECE - Має два значення:
  - Якщо біт SYN чистий до 0, то ECE означає, що IP-пакет має встановлений біт CE (перевантаження).
  - Якщо біт SYN встановлено на 1, ECE означає, що пристрій підтримує ECT.
- URG – вказує, що поле термінового вказівника містить значущі дані і має бути обробленим.
- ACK – вказує, що поле підтвердження має значення. Якщо ACK встановлено до 0, це означає, що пакет не містить жодного підтвердження.
- PSH - Якщо встановлено, це запит до станції-отримувача на PUSH даних (як тільки вони надходять) до програми-отримувача без їх буферизації.
- RST - Прапор скидання має такі особливості:
  - Використовується для відмови від вхідного з’єднання.
  - Використовується для відхилення сегмента.
  - Використовується для перезапуску з'єднання.
- SYN - Цей прапор використовується для встановлення з'єднання між хостами.
- FIN – цей прапор використовується для розблокування з’єднання, і після цього обмін даними більше не відбувається. Оскільки пакети з прапорами SYN і FIN мають порядкові номери, вони обробляються в правильному порядку.

Зв’язок TCP працює згідно моделі клієнт – сервер. Клієнт ініціює з’єднання, а сервер або приймає, або відхиляє його. Тристороннє рукостискання використовується для керування з’єднанням між хостами. Клієнт ініціює з’єднання та надсилає сегмент із порядковим номером. Сервер підтверджує його своїм власним порядковим номером і ACK сегмента клієнта, який на один більше, ніж порядковий номер клієнта. Клієнт після отримання ACK свого сегмента надсилає підтвердження відповіді сервера

(Рис.1.2.). Будь-який із серверів і клієнта можуть надіслати сегмент TCP з прапором FIN, встановленим на 1. Коли одержувач відповідає на нього, підтверджуючи FIN, цей напрямок TCP-зв'язку закривається, а з'єднання розривається [2].

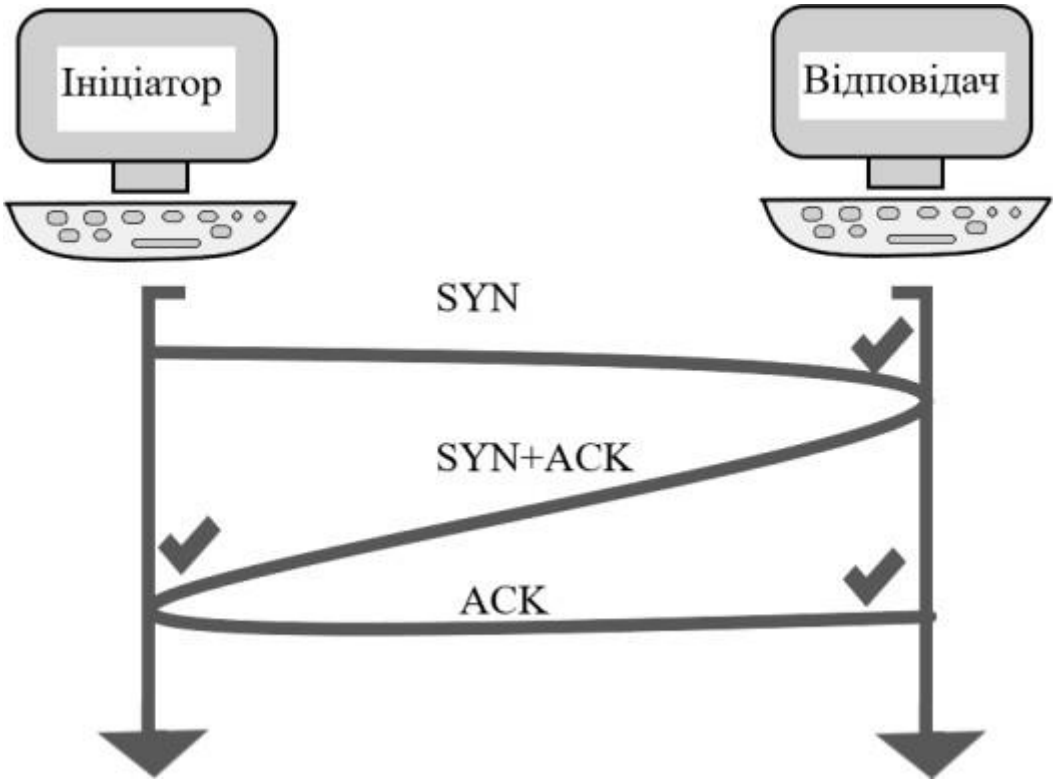


Рис.1.2. Встановлення з’єднання між клієнтом та сервером

1.5 Мережевий рівень та його взаємодія з транспортним рівнем

Основний протокол мережевого рівня стеку TCP/IP – IP-протокол. Головне завдання даного протоколу – це маршрутизація пакетів між різнотипними комп’ютерними мережами. Для цього протокол використовує таблицю маршрутизації та фрагментує або дефрагментує пакети с даними за потребою. Даний протокол може забезпечити доставку даних між пристроями, але слід враховувати, що на одному пристрої може бути запущено декілька програм та служб. Тому не достатньо ідентифікувати тільки адресу пристрою в мережі з допомогою IP-адреси, треба ще розрізняти програми-одержувачі між собою, а дану функцію не можливо виконати за допомогою протоколів мережевого рівня. Більш того, пакети мережевого рівня розглядаються пристроєм як окремий та незалежний блок даних, тому

стає складним забезпечити цілісність даних, що відправляються з одного хоста на інший.[6] Таким чином, якщо розглядати протоколи транспортного рівня як надбудову над IP-протоколом, то можна зробити висновок, що протоколи транспортного рівня виконують покриття даних функцій та вирішують попередньо поставлені задачі про передачу даних відповідним програмам на одному пристрою та забезпечення цілісності даних, що передаються. Але гарантовану цілісність даних може забезпечити лише протокол TCP, не UDP [1]. Дану надбудову над мережевим рівнем можна уявити, якщо розглянути роботу додатка за TCP протоколом (Рис.1.3.)

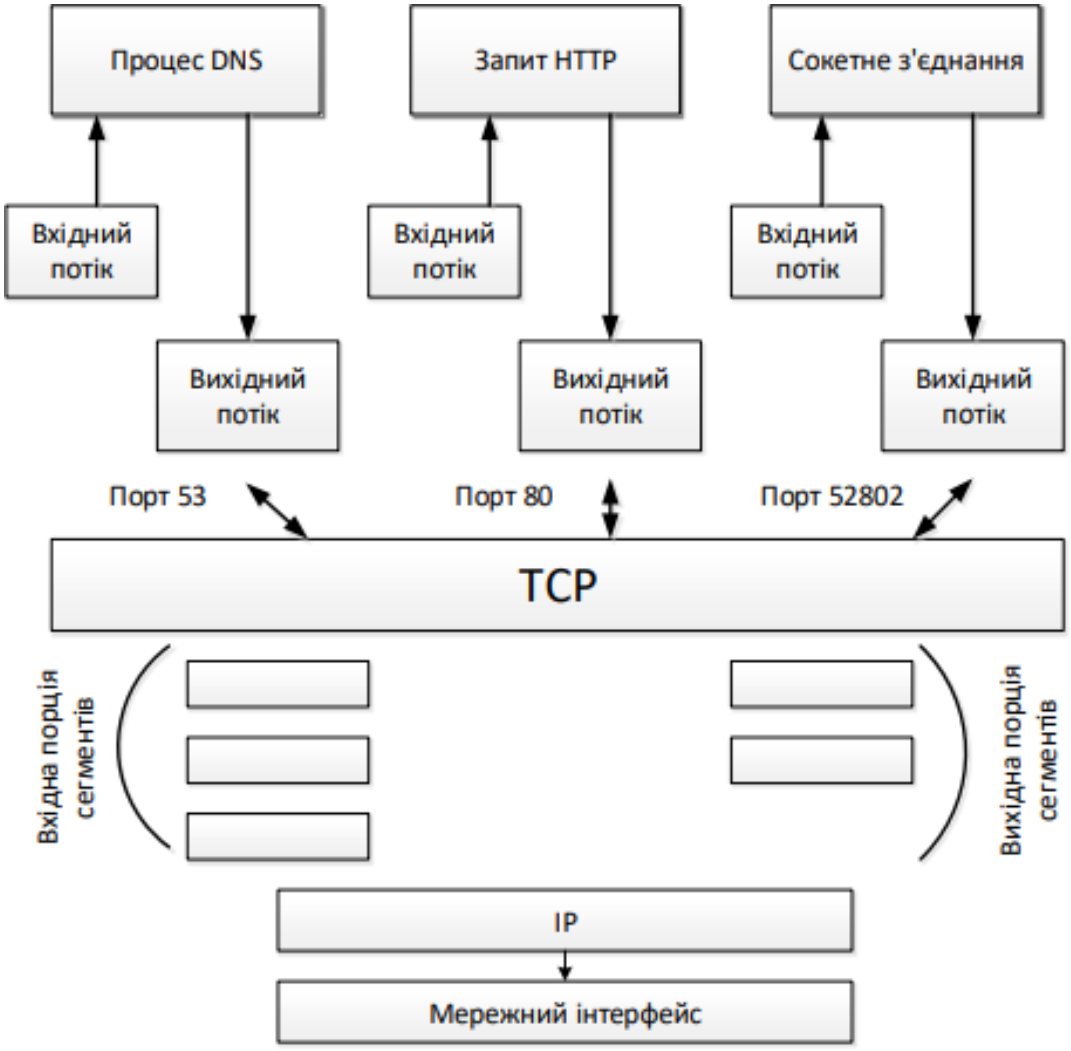


Рис.1.3. Робота додатка за TCP протоколом

1.6 Протоколи віддаленого керування комп’ютером

Віддалене управління - це процес передачі даних між комп’ютерами або серверами через мережу. Подібно до того, як браузеру потрібен протокол HTTP (протокол передачі гіпертексту) для перегляду веб-сторінок, віддалене керування також потребує підтримки спеціальних протоколів. Важливо зауважити що дані протоколи – є протоколами прикладного рівня за моделлю TCP/IP.
На даний момент існує 4 широко використовувані протоколи віддаленого керування:

1) Протокол RDP - протокол віддаленого робочого столу, більшість систем Windows підтримують цей протокол за замовчуванням, а керування віддаленим робочим столом у системах Windows базується на цьому протоколі. За замовчуванням, RDP використовує порт TCP 3389.
2) Протокол RFB - графічний протокол віддаленого керування, інструмент віддаленого керування VNC заснований на цьому протоколі. За замовчуванням RFB використовує діапазон TCP-портів з 5900 до 5906.
3) Telnet - протокол віддаленого керування інтерфейсом командного рядка. Майже всі операційні системи підтримують цей протокол за замовчуванням. Особливістю цього протоколу є те, що він використовує метод передачі чистого тексту під час передачі даних, тобто не шифрує дані. Telnet використовує порт TCP 23.
4) Протокол SSH - протокол віддаленого керування інтерфейсом командного рядка, майже всі операційні системи підтримують цей протокол за замовчуванням. На відміну від Telnet, цей протокол шифрує та стискає дані під час передачі даних, тому використання цього протоколу для передачі даних є безпечним і швидким. Для підключення по протоколу SSH використовується порт TCP 22.

Буде доцільно порівняти протоколи RDP та RFB, а також Telnet і SSH між собою, бо RDP як і RFB надає доступ керування віддаленим комп’ютером через графічний інтерфейс, натомість Telnet і SSH – через інтерфейс командного рядка.

Почнемо з RDP та RFB:

Як протокол RDP, так і протокол RFB дозволяють користувачам отримувати доступ до віддалених систем через графічний інтерфейс користувача, але протокол RFB спрямований на передавання зображення, а протокол RDP має тенденцію передавати команди:

- протокол RFB намалює вікно у відеопам'яті на стороні сервера, а потім передає зображення клієнту, клієнту потрібно лише розшифрувати та відобразити отримане зображення
- RDP передасть роботу зі створення зображення клієнту, а сервер повинен внести відповідні налаштування на основі можливостей відображення клієнта.

Таким чином, для виконання тієї ж операції обсяг даних, що передаються за допомогою протоколу RFB, буде більшим, ніж RDP. Більш того, RDP має більш жорсткі вимоги до клієнта, ніж RFB. RFB підходить для «тонких клієнтів», а RDP - для мереж низької швидкості.

Далі порівняємо Telnet та SSH:

Протокол Telnet і протокол SSH є протоколами віддаленого керування командного рядка, мають спільне поле застосування та часто використовуються для віддаленого доступу до серверів. У порівнянні з протоколом Telnet, протокол SSH шифрує дані під час надсилання даних, і передача даних є більш безпечною. Тому протокол SSH замінює протокол Telnet майже у всіх сферах застосування. У деяких випадках, коли для тестування не потрібне шифрування (наприклад, у локальній мережі), протокол Telnet все ще може використовуватися.

1.7 SSH протокол для захисту TCP-з'єднання

SSH-тунелювання – це метод транспортування мережевих даних через зашифроване з'єднання SSH. Його можна використовувати для додавання шифрування до застарілих програм або для впровадження VPN (віртуальних приватних мереж) та доступу до служб Інтернету через брандмауери.
SSH є стандартом для безпечного віддаленого входу та передачі файлів через ненадійні мережі. Він також забезпечує спосіб захисту трафіку даних будь-якої даної програми за допомогою переадресації портів, по суті, тунелювання будь-якого порту TCP/IP через SSH [8]. Це означає, що трафік даних

програми спрямовується на потік всередині зашифрованого з’єднання SSH, щоб його не можна було підслуховувати чи перехопити під час його передачі. Тунелювання SSH дає змогу додавати безпеку мережі до застарілих програм, які не підтримують шифрування [8].

На малюнку (Рис.1.4.) представлено спрощений огляд тунелювання SSH. Захищене з'єднання через ненадійну мережу встановлюється між клієнтом SSH і сервером SSH . Це з’єднання SSH зашифровано, захищає конфіденційність і цілісність, а також автентифікує сторони, що спілкуються. З’єднання SSH використовується програмою для підключення до сервера програм. Якщо тунелювання ввімкнено, програма зв’язується з портом на локальному хості, який слухає клієнт SSH. Потім клієнт SSH пересилає програму через свій зашифрований тунель на сервер. Потім сервер підключається до реального сервера програм - зазвичай на тій же машині або в тому ж центрі обробки даних, що й сервер SSH. Таким чином, зв’язок з додатком захищений, без необхідності змінювати програму або процеси кінцевого користувача [5].

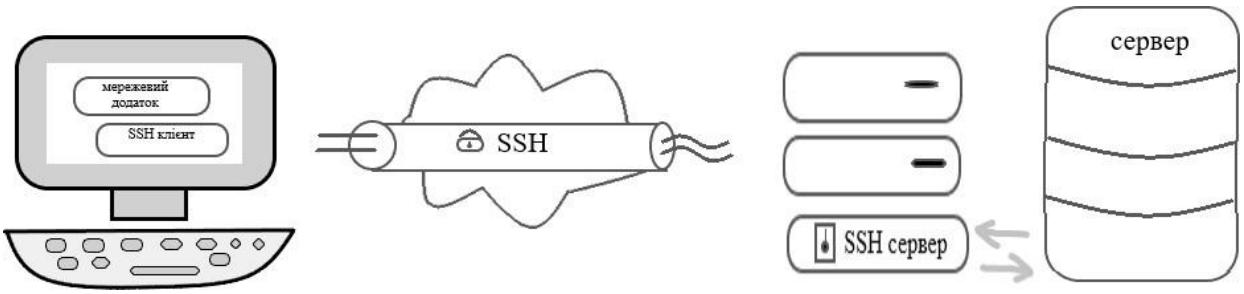


Рис 1.4. Спрощений огляд тунелювання SSH

Недоліком є те, що будь-який користувач, який може увійти на сервер, може увімкнути переадресацію портів. Дана функція широко використовується IT-спеціалістами для входу на свої домашні машини або сервери в хмарі, пересилаючи порт із сервера назад у корпоративну Інтернет мережу на свою робочу машину або відповідний сервер.

Тунелі SSH широко використовуються в багатьох корпоративних середовищах, які використовують мейнфреймові системи як серверні програми. У таких середовищах самі програми можуть мати дуже обмежену вбудовану підтримку безпеки.

1.8 Worker Service та Windows Service в .NET Framework

Програми-служби – є незамінними компонентами у роботі будь-якої операційної системи, вони можуть називатися по різному в залежності від ОС, але головний принцип їх роботи – це те що кінцевий користувач не помічає їх роботи, бо вона ведеться в тіньовому режимі і не є типовою програмою, яку можна запустити та побачити її інтерфейс. Більш того, багато програмних комплексів використовують служби для важливих задач, які необхідно виконувати навіть після закриття програми користувачем. Також існує потреба в розробці поодиноких служб, які будуть постійно виконувати поставлені задачі, такі як наприклад постійне встановлення та зберігання TCP з'єднання між віддаленими комп'ютерами. Операційні системи різного ґатунку, натомість, дають змогу реєструвати служби, вмикати або вимикати їх власноруч, встановлювати для них принцип початку роботи, тощо. Завдяки цьому служби ідеально підходять для використання на сервері, а також у ситуаціях, коли необхідні процеси, що довго виконуються, які не заважають роботі користувачів на тому ж комп'ютері. Служби можуть виконуватися в контексті безпеки певного облікового запису користувача, який відрізняється від облікового запису, який увійшов до системи користувача або облікового запису комп'ютера за промовчанням. .NET Framework дає змогу розробникам створювати такі служби для ОС Windows або кросплатформені програми, які можна скомпілювати та зареєструвати, наприклад, під Linux.

Розглянемо два типи проектів, які можна створити на базі .NET Framework мовою C# :

1) Windows Service

   Служби під ОС Windows можна зручно почати розробляти у середовищі Visual Studio, Rider, тощо. Вони дозволяють обрати тип даний проекту, після чого автоматично створяться всі необхідні файли, але це не обов'язково, бо можна створити і порожній проект а потім додати до нього всі необхідні класи.

   Зазвичай найважливішими для розробника є файл Program.cs і власне вузол служби Service.cs. Служба є звичайною програмою, але вона не запускає сама по собі.

   Усі виклики та звернення до неї проходять через менеджер керування службами (Service Control Manager або SCM). Коли служба запускається автоматично при старті системи або вручну, то SCM звертається до методу Main у класі Program, що знаходиться в однойменному файлі Program.cs.

2) Worker Service

Worker Service були представлені в .NET Core 3.0 і дозволяють запускати фонові служби за допомогою розміщеної служби.
Іншим способом запуску фонових служб є запуск розміщених служб у веб-додатку ASP.NET Core . Однак, якщо розміщена служба має проблеми з продуктивністю, це може вплинути на стабільність веб-програми. Worker Service призначена виключно для завдань, які виконуються у фоновому режимі. Як результат, вони ідеальні для завдань, які інтенсивно завантажують центральний процесор , або для завдань за розкладом на основі часу. Також такий проект служби зручний, якщо треба розробити службу під ОС Linux, тощо.

1.9 Мережні засоби платформи .NET Framework

Транспортний рівень TCP/IP перший рівень до прикладного з яким може взаємодіяти розробник для створення мережевих застосунків. Для роботи з сокетами в .NET були розроблені класи, які значно полегшують написання мережевих програм та служб (Таблиця 1.3.)

Клас .NET	Короткий опис
Socket	Реалізує інтерфейс сокетів Берклі. Надає широкій набір методів та властивостей для мережевої взаємодії. Клас дозволяє виконувати синхронну та асинхронну передачу даних за допомогою певного переліку протоколів зв'язку (TCP, IP, UDP, ICMP).
TcpClient	Клас надає методи для підключення, відправлення та отримання потокових даних через мережу в синхронному блокувальному режимі.
TcpListener	Прослуховує підключення від TCP-клієнтів у мережі.
UdpClient	Клас надає методи для відправлення та отримання датаграм UDP без підключення в блокувальному синхронному режимі. Оскільки UDP є транспортним протоколом без підключення, встановлювати підключення до віддаленого вузла перед надсиланням та отриманням даних не потрібно.
SocketExeption	Виняток, що створюється у разі виникнення помилки на

	сокеті.
SocketType	Вказує тип сокета, що є екземпляром класу Socket .
AddressFamily	Вказує схему адресації, яку може використовувати екземпляр класу Socket .

Таблиця 1.3. Основні класи для роботи з сокетами в .NET [8]

Базовим класом при побудові мережних програм є клас System.Net.Sockets.Socket, деякі властивості та методи якого представлено в таблицях 1.4. та 1.5.

Властивості	Короткий опис
IsBound	Отримує значення, що вказує, чи об'єкт Socket прив'язаний до конкретного локального порту.
Connected	Отримує значення, що вказує, чи об'єкт Socket підключається до віддаленого вузла в результаті останньої операції Send або Receive . Значення trueякщо об'єкт Socket в результаті останньої операції був підключений до віддаленого ресурсу; інакше - значення false.
RemoteEndPoint	Повертає віддалену кінцеву точку. Об'єкт EndPoint , з яким взаємодіє об'єкт Socket . Якщо використовується протокол, орієнтований на з'єднання, RemoteEndPoint властивість отримує об'єкт EndPoint , що містить віддалену IP-адресу та номер порту, до якого підключений об'єкт Socket . Якщо використовується протокол без підключення, RemoteEndPoint містить віддалену IP-адресу за промовчанням та номер порту, з яким Socket буде взаємодіяти.
LocalEndPoint	Повертає локальну кінцеву точку.
SocketType	Повертає тип служби Socket .
SendTimeout	Отримує або встановлює значення, що вказує на проміжок часу, після якого для синхронного виклику Send закінчиться час тайм-ауту.
ProtocolType	Отримує тип протоколу об'єкта Socket .

Таблиця 1.4. Опис властивостей класу System.Net.Sockets.Socket [8]

Властивості	Короткий опис
Accept	Створює новий об'єкт Socket для новоствореного підключення. Accept синхронно витягує з черги запитів на підключення сокета, що прослуховує перший очікуваний запит на підключення, а потім створює і повертає новий об'єкт Socket .
Bind	Зв'язує об'єкт Socket із локальною кінцевою точкою. Перед викликом методу необхідно викликати метод Listen .
Close	Закриває підключення Socket та звільняє всі пов'язані ресурси.
Connect	Встановлює підключення до віддаленого вузла.
Disconnect	Закриває підключення до сокету та дозволяє повторно його використовувати.
Dispose	Звільняє всі ресурси, що використовуються поточним екземпляром класу Socket .
Receive	Отримує дані від приєднаного сокету
Poll	Визначає статус сокету.
Listen	Переводить сокет у режим прослуховування

Таблиця 1.5. Опис методів класу System.Net.Sockets.Socket [8]

2 ПРАКТИЧНА ЧАСТИНА

2.1 Пояснення принципу роботи програмного коду мережевого програмного комплексу

Перед початком розгляду програмного коду розробленого мережевого програмного комплексу мовою C# у вигляді служби Windows за допомогою якого можна здійснювати віддалений доступ до комп’ютера, доцільно ознайомитися з основним алгоритмом його роботи (Рис 2.1.) на прикладі ініціювання віддаленого доступу за допомогою SSH.

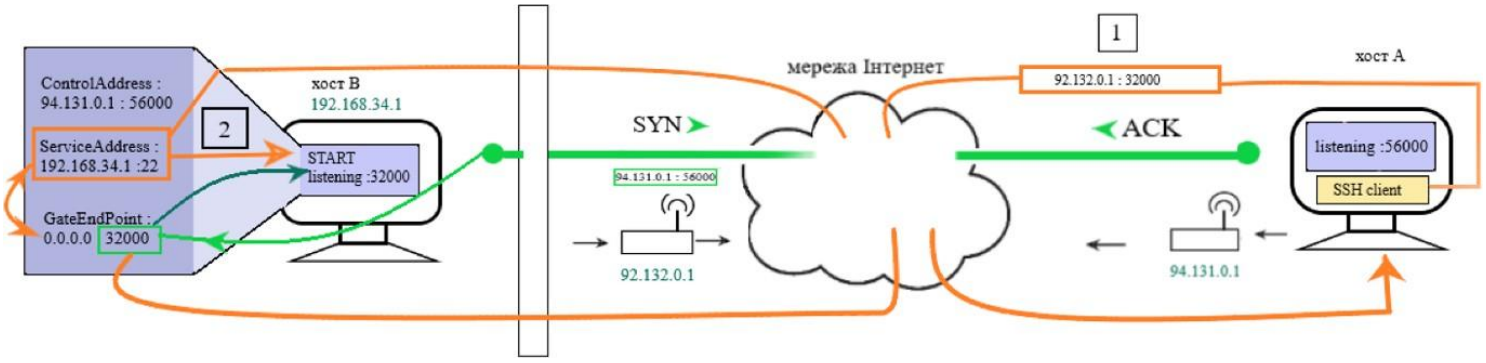


Рис 2.1. Схематичне зображення принципу роботи програми

При спробі під 'єднатися з хосту А до зовнішньої IP-адреси хоста В використовуючи SSH клієнт (наприклад) на порт 32000, хост В ініціює під'єднання до сокету 192.168.34.1:22, та отримані від нього дані будуть передаватися до хоста А також через TCP Proxy. Таким чином за допомогою служби TCP proxy можна буде емулювати реальне з'єднання по протоколам віддаленого доступу між двома хостами, керуючи портами на обох машинах.

Розроблений програмний комплекс має дві реалізації: під ОС Windows та Linux та має дві версії проекту : Windows Service та Worker Service відповідно.

Посилання на репозиторій з повним вихідним кодом проекту :
https://github.com/Hanna-Kushnarenko/TCPProxy-Windows- - реалізація під Windows
https://github.com/Hanna-Kushnarenko/TCPproxy-Linux- - реалізація під Linux

Розглянемо ключові моменти в роботі програмного коду на основі реалізації Worker Service.

1. Головний виконуваний файл Worker.cs, що відповідає саме за логіку роботи служби

Клас Worker наслідує клас BackgroundService, що базовим класом для реалізації тривалого IHostedService, який визначає методи для об’єктів, якими керує хост.

```
namespace TCPlinux;

public class Worker : BackgroundService
{
```

Далі визначається метод StartAsync, що запускається, коли хост програми готовий запустити службу.

```
    public override async Task StartAsync(CancellationToken cancellationToken)
    {
        TcpProxyManager.Instance.Run(false);
        await base.StartAsync(cancellationToken);
    }
```

Та визначається метод StopAsync, що запускається, коли хост програми зупиняє службу.

```
    public override async  Task StopAsync(CancellationToken cancellationToken)
    {
        TcpProxyManager.Instance.Stop();
        await base.StopAsync(cancellationToken);
    }
```

Реалізація методу Dispose в першу чергу призначена для звільнення некерованих ресурсів.

```
    public override void Dispose()
    {
        base.Dispose();
    }

}
```

2. В методах StartAsync та StopAsync класу Worker викликано метод з класу TcpProxyManager, цей клас реалізує загальну логіку роботи програми, тож розглянемо цей клас детальніше.

Приєднуємо всі використані в класі бібліотеки .NET разом зі власною бібліотекою класів Config

```
using System;
using System.Collections.Generic;
using  System.Configuration;
using System.IO;
using System.Text;
using System.Threading;
using System.Xml.Serialization;
using Config;
namespace TCPlinux;

public class TcpProxyManager
{
```

Ініціалізуємо необхідні змінні та екземпляри класів, що будуть використані в коді далі.

- manageThread – це екземпляр класу Thread , що створює і керує потоком, встановлює його пріоритет і отримує його статус.
- stopEvent – представляє подію синхронізації потоку, яку, якщо вона сигналізується, потрібно скинути вручну.
- configChangedEvent – представляє подію синхронізації потоку, яка, коли сигналізується, автоматично скидається після звільнення одного потоку, який є на черзі.
- linkConfig_– екземпляр власного класу LinlkConfig, що реалізує представлення xml файлу конфігурації.
- linkConfigFileName – в даній змінній зберігається назва конфігураційного файлу, для подальшого його зчитування.
- linkConfigWatcher – прослуховує сповіщення про зміни файлової системи та викликає події, коли змінюється каталог або файл у каталозі
- LastConfigUpdated – в цій зміній зберігаємо дату останнього оновлення файлу конфігурації.
- serverControlLink – об’єкт власного класу ServerControlLink, що зберігає інформацію щодо порту на який потрібно очікувати встановлення з’єднання, діапазону дозволений IP адрес, паролю для з’єднання, тощо. (дані з файлу LinkConfig.xml)
- remoteGateLink – об’єкт власного класу RemoteGateLink, що зберігає інформацію щодо порту, IP адреси комп'ютера на який потрібно встановлювати TCP зв’язок, власну локальну IP адресу, тощо. Таких об’єктів може бути безліч, тому змінна типу List (дані з файлу LinkConfig.xml)

```
private Thread manageThread_;
private ManualResetEvent stopEvent_ = new ManualResetEvent(false);
private AutoResetEvent configChangedEvent_ = new AutoResetEvent(false);
```

```
        private LinkConfig linkConfig_;
        private string linkConfigFileName_ = "LinkConfig.xml";
        private FileSystemWatcher linkConfigWatcher_;
        private DateTime LastConfigUpdated = DateTime.MinValue;
        private List<SimpleLink> simpleLinks_ = new List<SimpleLink>();
        private ServerControlLink serverControlLink_;
        private List<RemoteGateLink> remoteGateLink_ = new List<RemoteGateLink>();
        private int operationCount_;
        private static TcpProxyManager instance_;
        public WaitHandle StopHandle => (WaitHandle)this.stopEvent_;
```

Далі представлена реалізація методу Run, що викликається при початку роботи служби, даний метод використовує делегат ParameterizedThreadStart для виконання методу DoWork та слідкує за зміною файла LinkConfig.

```
        public void Run(bool wait)
        {
            manageThread_ = new Thread(new ParameterizedThreadStart(DoWork));
            linkConfigFileName_ = "LinkConfig.xml";
            linkConfigWatcher_ = new FileSystemWatcher (AppDomain.CurrentDomain.
BaseDirectory, linkConfigFileName_);
            linkConfigWatcher_.NotifyFilter = NotifyFilters.LastWrite;
            linkConfigWatcher_.Changed += new FileSystemEventHandler(OnChanged);
            linkConfigWatcher_.EnableRaisingEvents = true;
            LoadConfiguration();
            manageThread_.Start();
            if (!wait)
                return;
            manageThread_.Join();
        }
```

Наступний метод Stop викликається по завершенню служби та зупиняє всі запущені задачі та потоки.

```
        public void Stop()
        {
            this.stopEvent_.Set();
            if (this.manageThread_.ThreadState != ThreadState.Running)
                return;
            this.manageThread_.Join(60000);
            if (this.manageThread_.ThreadState != ThreadState.Running)
                return;
            Logger.Error("Working thread is not finished. Operations active: '{0}'.
Aborting.", (object)this.operationCount_);
            this.manageThread_.Abort();
        }
```

Наступний метод DoWork організує виконання мережевої логіки встановлення з’єднання між віддаленими хостами на основі зчитаних конфігурацій з файлу LinkConfig

```
        public void DoWork(object data)
        {
            try
            {
                StartSockets();
            label_1:
                while (true)
                {
                    switch (WaitHandle.WaitAny(new WaitHandle[2]
                    {
            (WaitHandle) stopEvent_,
            (WaitHandle) configChangedEvent_
                    }, 500, false))
                    {
                        case 0:
                            goto label_19;
                        case 1:
                            LoadConfiguration();
                            StartSockets();
                            continue;
                        default:
                            goto label_3;
                    }
                }
            label_3:
                if (serverControlLink_ != null)
                    serverControlLink_.ProcessPending();
                foreach (SimpleLink simpleLink in this.simpleLinks_)
                    simpleLink.ProcessPending();
                using (List<RemoteGateLink>.Enumerator enumerator =
remoteGateLink_.GetEnumerator())
                {
                    while (enumerator.MoveNext())
                        enumerator.Current.ProcessPending();
                    goto label_1;
                }
            }
            catch (Exception ex)
            {
                Logger.Error("Unhandled exception occured", ex);
            }
            finally
            {
```

```
                Logger.Important("Shutting down...");
                ShutDown();
            }
        label_19:
            while (operationCount_ > 0)
            {
                Thread.Sleep(100);
                Thread.MemoryBarrier();
                Logger.Debug("waiting for " + (object)operationCount_ + " operations
to complete...");
            }
        }

        public void EnterAsyncOperation() => Interlocked.Increment(ref
this.operationCount_);

        public void LeaveAsyncOperation() => Interlocked.Decrement(ref
this.operationCount_);

        private void ShutDown()
        {
            this.linkConfigWatcher_.Dispose();
            this.linkConfigWatcher_ = (FileSystemWatcher)null;
            this.StopSockets();
        }
```

Наступний метод зачиняє усі відкриті раніше сокети для TCP з’єднання.

```
        private void StopSockets()
        {
            foreach (SimpleLink simpleLink in this.simpleLinks_)
            {
                try
                {
                    simpleLink.Dispose();
                }
                catch
                {
                }
            }
            simpleLinks_.Clear();
            if (serverControlLink_ != null)
            {
                try
                {
                    serverControlLink_.Dispose();
                }
                catch
                {
```

```
                }
                serverControlLink_ = (ServerControlLink)null;
            }
            foreach (RemoteGateLink remoteGateLink in this.remoteGateLink_)
            {
                try
                {
                    remoteGateLink.Dispose();
                }
                catch
                {
                }
            }
            remoteGateLink_.Clear();
            for (int index = 0; index < 10; ++index)
            {
                if (operationCount_ > 0)
                {
                    Thread.Sleep(100);
                    Thread.MemoryBarrier();
                    Logger.Debug("waiting for " + (object)operationCount_ + "
operations to complete...");
                }
            }
        }
```

На основі заповненого файлу конфігурації, наступний метод визначає які конфігурації були створені та для кожного варіанту створює екземпляр класу з відповідними даними, який вже далі буде визначати мережеві дії для хоста.

```
        private void StartSockets()
        {
            StopSockets();
            foreach (Config.SimpleLink simpleLink in linkConfig_.SimpleLinks)
                simpleLinks_.Add(new SimpleLink(linkConfig_.BindTimeout,
simpleLink));
            if (linkConfig_.ServerControlLink != null)
                serverControlLink_ = new ServerControlLink(linkConfig_.BindTimeout,
linkConfig_.KeepAliveTimeout, linkConfig_.ServerControlLink);
            foreach (Config.RemoteGateLink remoteGateLink in
linkConfig_.RemoteGateLinks)
                remoteGateLink_.Add(new RemoteGateLink(linkConfig_.BindTimeout,
linkConfig_.KeepAliveTimeout, remoteGateLink));
        }
```

Метод LoadConfiguration зчитує дані з файлу конфігурації в об’єкт класу LinkConfig та використовуэться у методах DoWork та Run.

```
        private void LoadConfiguration()
```

```
        {
            try
            {
                using (TextReader textReader = (TextReader)new
StreamReader(AppDomain.CurrentDomain.BaseDirectory + this.linkConfigFileName_,
Encoding.UTF8))
                {
                    this.linkConfig_ = (LinkConfig)new
XmlSerializer(typeof(LinkConfig)).Deserialize(textReader);
                    Logger.LogLevel = (LogLevel)this.linkConfig_.LogLevel;
                }
            }
            catch (Exception ex)
            {
                Logger.Error("Can't load LinkConfig", ex);
                this.linkConfig_ = new LinkConfig();
            }
        }
```

Наступний метод слідкує за змінами у файлі конфігурації та робить в журнал логів запис, що файл було змінено та за можливості перезапускає службу.

```
        private void OnChanged(object source, FileSystemEventArgs e)
        {
            DateTime lastWriteTimeUtc = File.GetLastWriteTimeUtc(e.FullPath);
            if (!(lastWriteTimeUtc > LastConfigUpdated))
                return;
            LastConfigUpdated = lastWriteTimeUtc;
            Logger.Important("Configuration updated on {0}. Reloading...",
(object)LastConfigUpdated);
            configChangedEvent_.Set();
        }
```

Останній елемент даного класу, це конструктор, що реалізує створення власного екземпляру.

```
        public static TcpProxyManager Instance
        {
            get
            {
                if (TcpProxyManager.instance_ == null)
                    TcpProxyManager.instance_ = new TcpProxyManager();
                return TcpProxyManager.instance_;
            }
        }
    }
```

1. Вище розглянуто структуру класу менеджера, що загалом керує початком та зупиненням роботи служби та запускає процес роботи для різних

варіантів конфігурації. Тепер розглянемо окремо клас, який реалізує алгоритм роботи програми, якщо була встановлено конфігурація RemoteGateLinks. Такі налаштування потрібно зробити для того, щоб хост намагався встановити з'єднання з віддаленим комп'ютером. Для цього встановлюється параметр GateControlAddress до якого входить IP-адреса комп'ютера, з якого ми бажаємо отримати доступ та порт, який прослуховує даний комп'ютер. ServiceAddress – це конфігурація, що вказує на службу призначення. GateEndPoint – це конфігурація, що надсилається кінцевому хосту, для очікування команди на цільовий порт даної конфігурації.

```
<RemoteGateLinks>
    <RemoteGateLink>
        <GateControlAddress>
            <Address>192.168.31.147</Address>
            <PortNumber>52002</PortNumber>
        </GateControlAddress>
        <GateEndPoint>
            <IP>0.0.0.0</IP>
            <PortNumber>32022</PortNumber>
        </GateEndPoint>
        <Key></Key>
        <ServiceAddress>
            <Address>192.168.31.72</Address>
            <PortNumber>22</PortNumber>
        </ServiceAddress>
        <Sources>
            <SourcePoint>
                <IPFrom>1.0.0.1</IPFrom>
                <IPTo>223.255.255.254</IPTo>
            </SourcePoint>
        </Sources>
    </RemoteGateLink>

</RemoteGateLinks>
```

Далі розглянемо основні методи класу RemoteGateLink:

Метод ProcessPending, який викликається з класу TcpProxyManager

```
public void ProcessPending()
        {
```

Перевірка для запуску коду тільки якщо цільовий сокет є нулем та перед останньою спробою з'єднання пройшло достатньо часу, щоб спробувати ще.

```
            if ((this.server_ == null) &&
(this.lastConnectTry_.AddMilliseconds((double)this.connectTimeout_) <
DateTime.UtcNow))
            {
                if (this.serverCandidate_ != null)
                {
                    try
                    {
                        this.serverCandidate_.Shutdown(SocketShutdown.Both);
                        this.serverCandidate_.Close();
```

```
                    }
                    catch
                    {
                    }
                    this.serverCandidate_ = null;
                }
                this.lastConnectTry_ = DateTime.UtcNow;
```

Створюємо та ініціюємо з’єднання з віддаленим комп’ютером, якщо з’єднання було розірване фіксуємо це повідомленням.

```
                this.serverCandidate_ = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                object[] args = new object[] { this.linkConfig_.GateControlAddress };
                Logger.Important("Connecting to remote link gateway control port
'{0}'", args);
                TcpProxyManager.Instance.EnterAsyncOperation();
                this.serverCandidate_.BeginConnect(this.linkConfig_.GateControlAddres
s.Address, this.linkConfig_.GateControlAddress.PortNumber, new
AsyncCallback(this.ConnectCallback), null);
            }
            if (this.server_ != null)
            {
                this.lastReceivedMutex_.WaitOne();
                try
                {
                    if
(this.lastReceived_.AddMilliseconds((double)this.keepAliveTimeout_) <
DateTime.UtcNow)
                    {
                        try
                        {
                            object[] args = new object[] {
this.linkConfig_.GateControlAddress, this.lastReceived_.ToLocalTime() };
                            Logger.Info("Connection to remote link gateway control
port timed out '{0}'. Last response on: '{1}'", args);
                            this.server_.Shutdown(SocketShutdown.Both);
                            this.server_.Close();
                        }
                        catch
                        {
                        }
                        this.server_ = null;
                    }
                }
                finally
                {
                    this.lastReceivedMutex_.ReleaseMutex();
                }
```

```
            }
```

Метод ReceiveCandidateCallback викликається при встановленому стабільному TCP з'єднанні.

```
 private void ReceiveCandidateCallback(IAsyncResult ar)
        {
            if (!this.isDisposed_)
            {
                try
                {
                    if (this.serverCandidate_.EndReceive(ar) != 0x10)
                    {
                        throw new ApplicationException("Invalid answer received");
                    }
                    byte[] destinationArray = new byte[0x10];
                    Array.Copy(this.buffer, destinationArray,
destinationArray.Length);
                    if (!string.IsNullOrEmpty(this.linkConfig_.Key))
                    {
                        RijndaelManaged managed = new RijndaelManaged();
                        managed.Padding = PaddingMode.None;
                        managed.IV = destinationArray;
                        managed.Key = Tools.GenerateKey(this.linkConfig_.Key);
                        base.Decryptor = managed.CreateDecryptor();
                        base.Encryptor = managed.CreateEncryptor();
                    }
```

Далі створюємо запит до віддаленого комп’ютера на прослуховування порта, до якого буде створено запит після спроби ініціювати віддалене з’єднання по протоколу прикладного рівня

```
                    object[] objArray = new object[] { "listen ",
this.linkConfig_.GateEndPoint.IP, " ", this.linkConfig_.GateEndPoint.PortNumber };
                    string msg = string.Concat(objArray);
                    base.EncryptAndSend(msg, this.serverCandidate_);
                    string message = base.ReceiveAndDecrypt(this.serverCandidate_);
                    if (message != "ok")
                    {
                        throw new ApplicationException(message);
                    }
                    this.lastReceivedMutex_.WaitOne();
                    try
                    {
                        this.server_ = this.serverCandidate_;
                        this.serverCandidate_ = null;
                        this.lastReceived_ = DateTime.UtcNow;
                    }
                    finally
```

```
                    {
                        this.lastReceivedMutex_.ReleaseMutex();
                    }
                }
                catch (Exception exception)
                {
                    object[] args = new object[] {
Tools.GetSafeRemoteEndPoint(this.serverCandidate_) };
                    Logger.Error("Can't start listening on the remote gate control
'{0}'", exception, args);
                    try
                    {
                        if (this.serverCandidate_.Connected)
                        {
                            this.serverCandidate_.Shutdown(SocketShutdown.Both);
                        }
                        this.serverCandidate_.Close();
                    }
                    catch
                    {
                    }
                    this.serverCandidate_ = null;
                    this.server_ = null;
                    return;
                }
                finally
                {
                    TcpProxyManager.Instance.LeaveAsyncOperation();
                }
                TcpProxyManager.Instance.EnterAsyncOperation();
                this.server_.BeginReceive(this.buffer, 0, this.buffer.Length,
SocketFlags.None, new AsyncCallback(this.ReceiveCallback), this.server_);
            }
        }
```

Наступний метод в результаті свого виконання повинен зробити пару сокетів зі сокету, який пов’язаний з ініціюванням з’єднання по протоколу прикладного рівня та віддалений комп’ютером

```
        private void EstablishGateLink(string handle, string ipaddress)
        {
            Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
            Socket socket2 = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
            try
            {
                object[] args = new object[] { this.linkConfig_.ServiceAddress };
```

```
                Logger.Info("Connecting to service '{0}'", args);
                try
                {
                    socket2.Connect(this.linkConfig_.ServiceAddress.Address,
this.linkConfig_.ServiceAddress.PortNumber);
                }
                catch (Exception exception)
                {
                    base.EncryptAndSend("Connect to service failed: " +
exception.Message, this.server_);
                    throw;
                }
                base.EncryptAndSend("ok", this.server_);
                object[] objArray2 = new object[] {
this.linkConfig_.GateControlAddress, handle };
                Logger.Info("Connecting to remote link '{0}'. Handle '{1}'",
objArray2);
                s.Connect(this.linkConfig_.GateControlAddress.Address,
this.linkConfig_.GateControlAddress.PortNumber);
                byte[] buffer = new byte[0x10];
                if (s.Receive(buffer) != buffer.Length)
                {
                    throw new ApplicationException("Invalid answer received");
                }
                CryptoLink link = new CryptoLink();
                if (!string.IsNullOrEmpty(this.linkConfig_.Key))
                {
                    RijndaelManaged managed = new RijndaelManaged();
                    managed.Padding = PaddingMode.None;
                    managed.IV = buffer;
                    managed.Key = Tools.GenerateKey(this.linkConfig_.Key);
                    link.Decryptor = managed.CreateDecryptor();
                    link.Encryptor = managed.CreateEncryptor();
                }
                link.EncryptAndSend("establish " + handle, s);
                string str = link.ReceiveAndDecrypt(s);
                if (str != "ok")
                {
                    throw new ApplicationException("Invalid answer received:" + str);
                }
                new SocketPair(s, ipaddress + "<->" + s.RemoteEndPoint.ToString(),
socket2, socket2.RemoteEndPoint.ToString()).StartPairWork();
            }
```

2.2 Налаштування мережевого програмного комплексу на двох комп’ютерах з ОС Windows 10

В даній частині налаштовано роботу мережевого програмного комплексу на двох комп’ютерах з ОС Windows 10 та проілюстровано успішне отримання доступу до одного з хостів іншим використовуючи даний програмний комплекс.

Для зручності, далі комп’ютер з якого буде отримано віддалений доступ – хост А, а комп’ютер до якого буде отримано віддалений доступ – хост В.

На хості В налаштовано клієнт Bitvise SSH, що використовується для ініціації підключень до серверів SSH . Зазвичай він використовується в інтерактивному режимі, тому він запускається лише тоді, коли його запускає користувач, але його також можна запустити без нагляду для виконання команд сценаріїв або передачі файлів або для підтримки з’єднання SSH для переадресації портів.

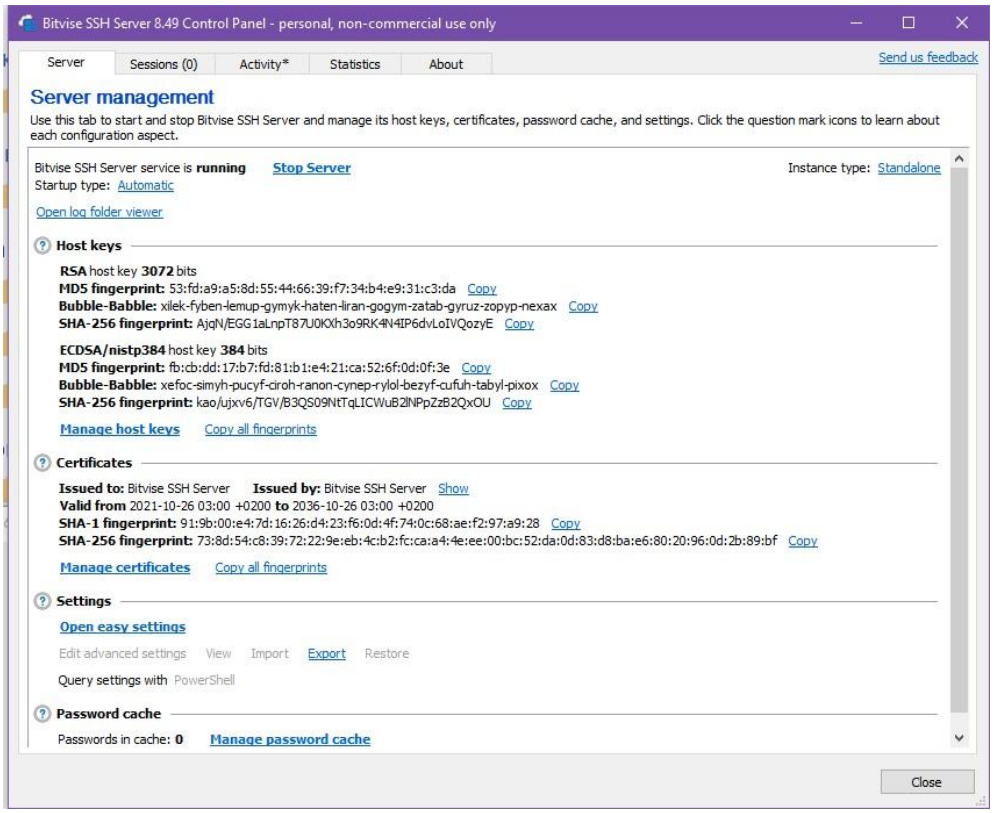


Рис 2.2. Вікно налаштувань Bitvise SSH

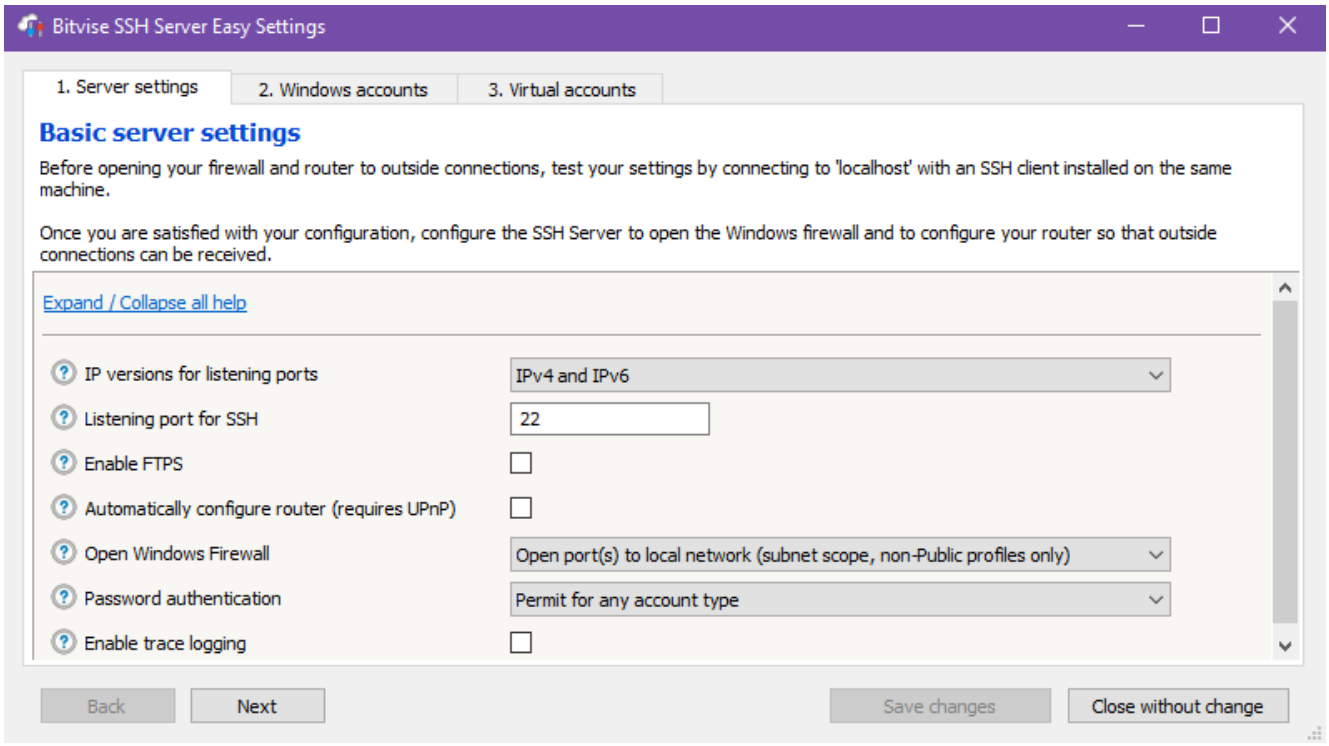


Рис 2.3. Базові налаштування серверу у Bitvise SSH

Надаємо доступ для підключення по SSH для користувача на хості B.

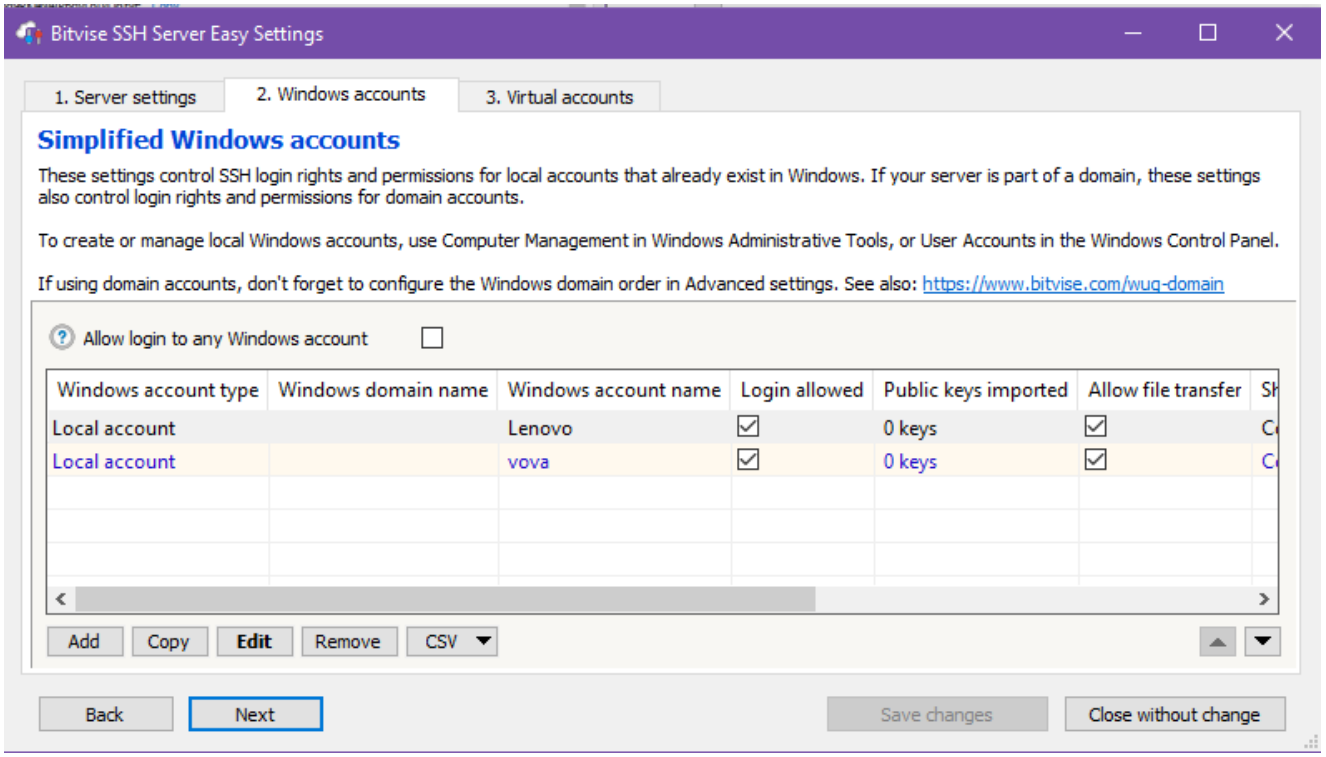


Рис 2.4. Налаштування доступу у Bitvise SSH

Також на хості В створено вхідне правило для брандмауера Windows 10. Для того щоб дозволити підключення до хоста за допомогою TCP та SSH (port = 22)

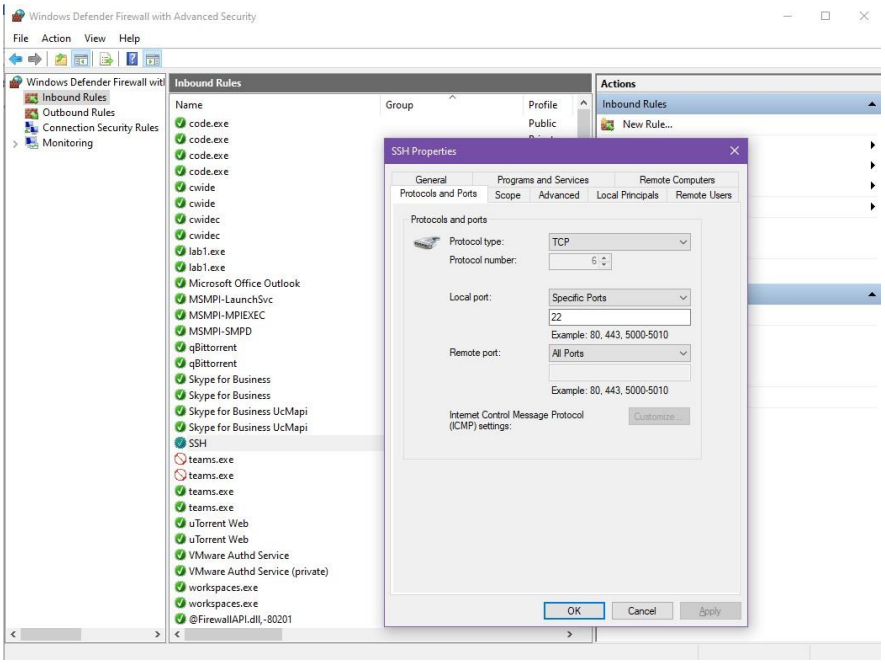


Рис 2.5. Процес створення вхідного правила для брандмауера Windows

Встановлюємо розроблену службу на обидва хости за допомогою додатку InstallUtil.

```
C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug>installUtil TCPV_1.exe
Microsoft (R) .NET Framework Installation utility Version 4.8.4084.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Running a transacted installation.

Beginning the Install phase of the installation.
See the contents of the log file for the C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.exe assembly's progress.
The file is located at C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.InstallLog.
Installing assembly 'C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.exe'.
Affected parameters are:
   logtoconsole =
   assemblypath = C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.exe
   logfile = C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.InstallLog
Installing service Service1...
Service Service1 has been successfully installed.
Creating EventLog source Service1 in log Application...

The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.exe assembly's progress.
The file is located at C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.InstallLog.
Committing assembly 'C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.exe'.
Affected parameters are:
   logtoconsole =
   assemblypath = C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.exe
   logfile = C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug\TCPV_1.InstallLog

The Commit phase completed successfully.

The transacted install has completed.

C:\Program Files (x86)\tcp(v2)\TCPV_1\TCPV_1\bin\Debug>
```

Рис 2.6. Процес реєстрації служби на ОС Windows за допомогою InstallUtil

В директорії проекту скомпільованої служби у відповідному файлі конфігурацій, з якого будуть зчитуватися налаштування для роботи кожного хоста, встановлюємо наступні параметри :

Для хоста А :

```
<ServerControlLink>
    <EndPoint>
        <IP>0.0.0.0</IP>
        <PortNumber>52002</PortNumber>
    </EndPoint>
    <Sources>
        <SourcePoint>
            <IPFrom>1.0.0.1</IPFrom>
            <IPTo>223.255.255.254</IPTo>
        </SourcePoint>
    </Sources>
    <Key></Key>
</ServerControlLink>
```

Рис 2.7. Приклад заповнення файлу конфігурації для машини до якої планується встановити віддалений доступ

Згідно налаштованих параметрів – хост А буде прослуховувати надходження сегменту для встановлення зв’язку по протоколу TCP у діапазоні IP-адрес 1.0.0.0 – 223.255.255.254. Даний діапазон можна змінювати для фільтрації адрес з якими потенційно може бути встановлено з’єднання.

Для хоста В :

```
<RemoteGateLinks>
    <RemoteGateLink>
        <GateControlAddress>
            <Address>192.168.31.147</Address>
            <PortNumber>52002</PortNumber>
        </GateControlAddress>
        <GateEndPoint>
            <IP>0.0.0.0</IP>
            <PortNumber>32022</PortNumber>
        </GateEndPoint>
        <Key></Key>
        <ServiceAddress>
            <Address>192.168.31.72</Address>
            <PortNumber>22</PortNumber>
        </ServiceAddress>
        <Sources>
            <SourcePoint>
                <IPFrom>1.0.0.1</IPFrom>
                <IPTo>223.255.255.254</IPTo>
            </SourcePoint>
        </Sources>
    </RemoteGateLink>

</RemoteGateLinks>
```

Рис 2.8. Приклад заповнення файлу конфігурації для машини з якої планується встановити віддалений доступ

Хост В буде власне ініціювати з’єднання поки не отримає відповідь від хоста А. Для цього встановлюється параметр GateControlAddress до якого входить IP-адреса комп’ютера, з якого ми бажаємо отримати доступ та порт, який прослуховує даний комп’ютер. ServiceAddress – це конфігурація, що вказує на службу призначення, вказується порт 22, бо з’єднання буде затунельвано за допомогою SSH клієнта. GateEndPoint – це конфігурація для прослуховування на хості А, з хоста В буде налаштоване підключення саме на цей порт.

Далі на обох хостах запускаємо встановлену службу з відповідними налаштуваннями збереженими у XML файлі . Служба має назву TCP Service. Її можна побачити у переліку служб на комп’ютері.

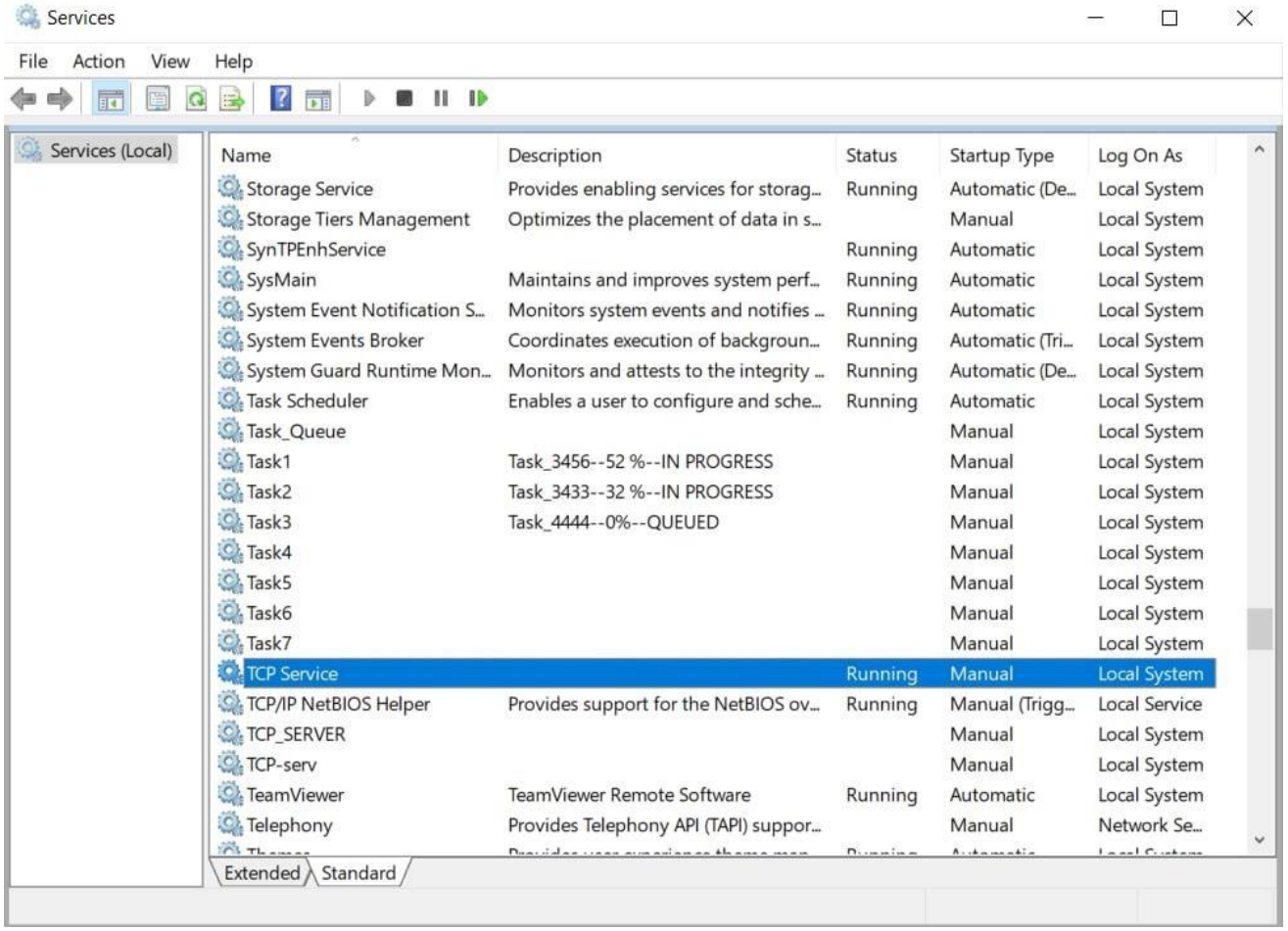
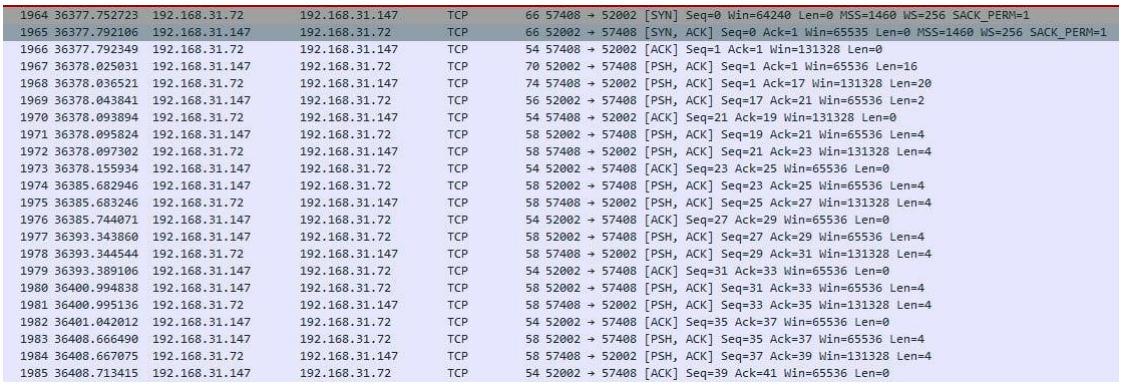


Рис 2.9. Вікно програми для управління службами на ОС Windows

За допомогою аналізу мережевого трафіку додатком WireShark видно, що TCP з’єднання між двома хостами 192.168.31.72 та 192.168.31.147 встановлено та хости продовжують обмінюватися сегментами даних :



1964	36377.752723	192.168.31.72	192.168.31.147	TCP	66 57408 → 52002 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1965	36377.792106	192.168.31.147	192.168.31.72	TCP	66 52002 → 57408 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
1966	36377.792349	192.168.31.72	192.168.31.147	TCP	54 57408 → 52002 [ACK] Seq=1 Ack=1 Win=131328 Len=0
1967	36378.025031	192.168.31.147	192.168.31.72	TCP	70 52002 → 57408 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=16
1968	36378.036521	192.168.31.72	192.168.31.147	TCP	74 57408 → 52002 [PSH, ACK] Seq=1 Ack=17 Win=131328 Len=20
1969	36378.043841	192.168.31.147	192.168.31.72	TCP	56 52002 → 57408 [PSH, ACK] Seq=17 Ack=21 Win=65536 Len=2
1970	36378.093894	192.168.31.72	192.168.31.147	TCP	54 57408 → 52002 [ACK] Seq=21 Ack=19 Win=131328 Len=0
1971	36378.095824	192.168.31.147	192.168.31.72	TCP	58 52002 → 57408 [PSH, ACK] Seq=19 Ack=21 Win=65536 Len=4
1972	36378.097302	192.168.31.72	192.168.31.147	TCP	58 57408 → 52002 [PSH, ACK] Seq=21 Ack=23 Win=131328 Len=4
1973	36378.155934	192.168.31.147	192.168.31.72	TCP	54 52002 → 57408 [ACK] Seq=23 Ack=25 Win=65536 Len=0
1974	36385.682946	192.168.31.147	192.168.31.72	TCP	58 52002 → 57408 [PSH, ACK] Seq=23 Ack=25 Win=65536 Len=4
1975	36385.683246	192.168.31.72	192.168.31.147	TCP	58 57408 → 52002 [PSH, ACK] Seq=25 Ack=27 Win=131328 Len=4
1976	36385.744071	192.168.31.147	192.168.31.72	TCP	54 52002 → 57408 [ACK] Seq=27 Ack=29 Win=65536 Len=0
1977	36393.343860	192.168.31.147	192.168.31.72	TCP	58 52002 → 57408 [PSH, ACK] Seq=27 Ack=29 Win=65536 Len=4
1978	36393.344544	192.168.31.72	192.168.31.147	TCP	58 57408 → 52002 [PSH, ACK] Seq=29 Ack=31 Win=131328 Len=4
1979	36393.389106	192.168.31.147	192.168.31.72	TCP	54 52002 → 57408 [ACK] Seq=31 Ack=33 Win=65536 Len=0
1980	36400.994838	192.168.31.147	192.168.31.72	TCP	58 52002 → 57408 [PSH, ACK] Seq=31 Ack=33 Win=65536 Len=4
1981	36400.995136	192.168.31.72	192.168.31.147	TCP	58 57408 → 52002 [PSH, ACK] Seq=33 Ack=35 Win=131328 Len=4
1982	36401.042012	192.168.31.147	192.168.31.72	TCP	54 52002 → 57408 [ACK] Seq=35 Ack=37 Win=65536 Len=0
1983	36408.666490	192.168.31.147	192.168.31.72	TCP	58 52002 → 57408 [PSH, ACK] Seq=35 Ack=37 Win=65536 Len=4
1984	36408.667075	192.168.31.72	192.168.31.147	TCP	58 57408 → 52002 [PSH, ACK] Seq=37 Ack=39 Win=131328 Len=4
1985	36408.713415	192.168.31.147	192.168.31.72	TCP	54 52002 → 57408 [ACK] Seq=39 Ack=41 Win=65536 Len=0

Рис 2.10. Процес обміну даними між двома хостами за протоколом TCP

Можемо перевірити наявність успішного з’єднання за допомогою команди netstat :

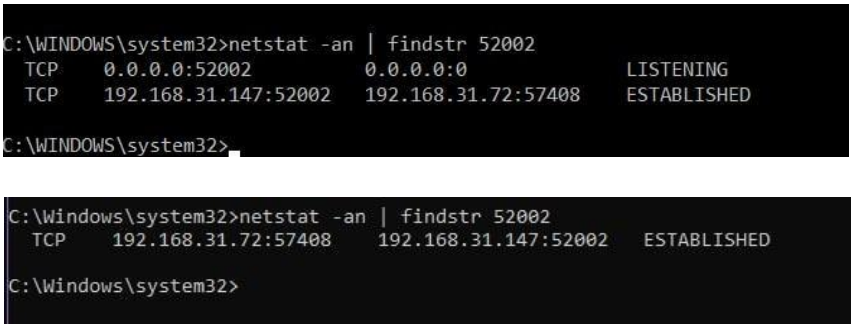


Рис 2.11. Результат виконання команди netstat на двох хостах з ОС Windows після запуску на налаштування розробленої служби

За допомогою Putty з хоста А отримаємо віддалений доступ до хоста В :

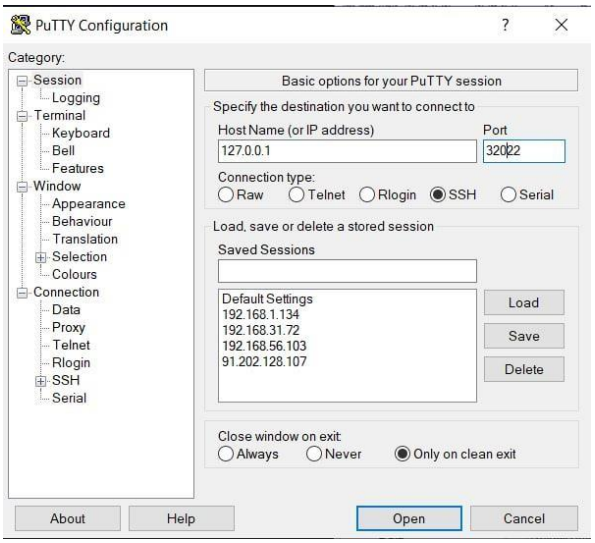


Рис 2.12. Приклад налаштування вікна програми Putty для встановлення віддаленого доступу за участю розробленої служби

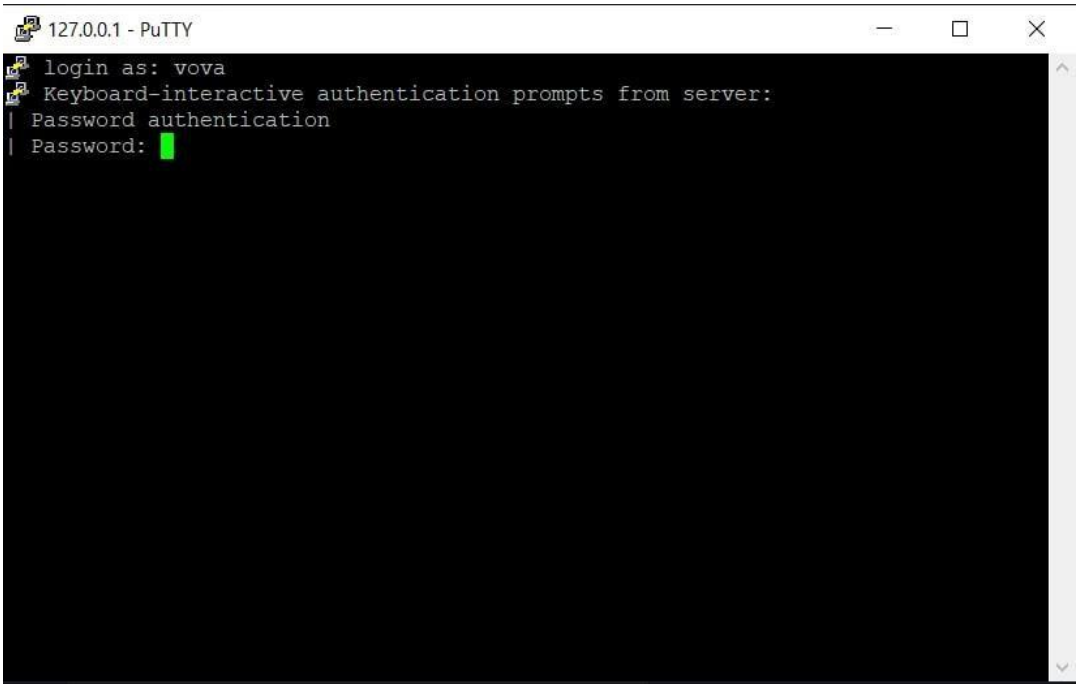


Рис 2.13. Представлений інтерфейс командного рядка для керування віддаленим комп’ютером

У результаті успішного введенні паролю та логіну користувача на віддаленому комп’ютері буде отримано доступ до командного рядку користувача та можна буде виконувати команди наприклад :

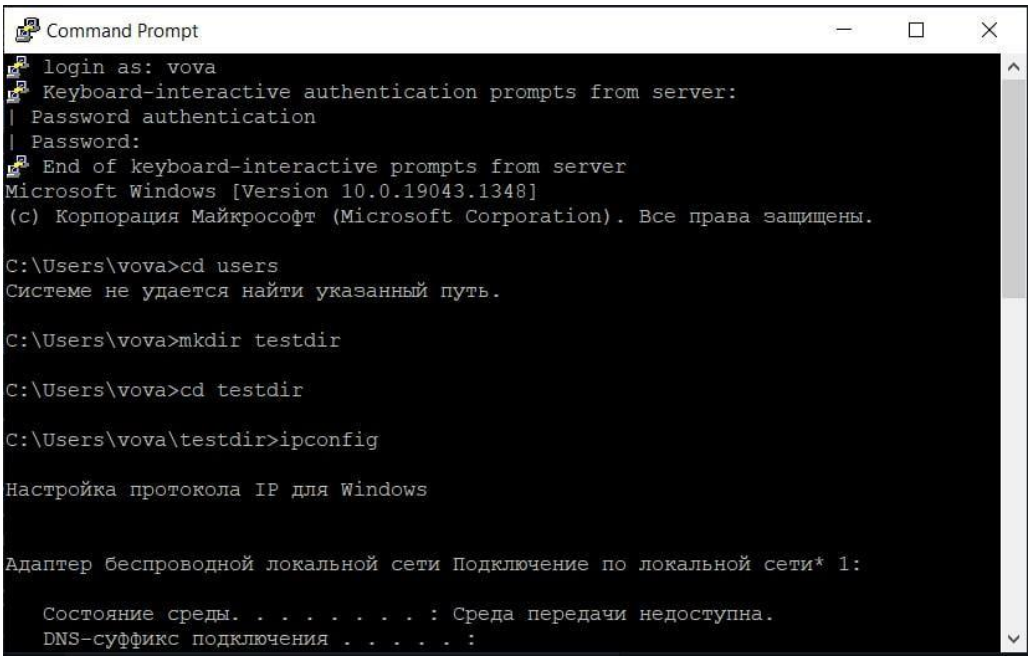


Рис 2.14. Приклад використання можливостей віддаленого доступу через інтерфейс командного рядка

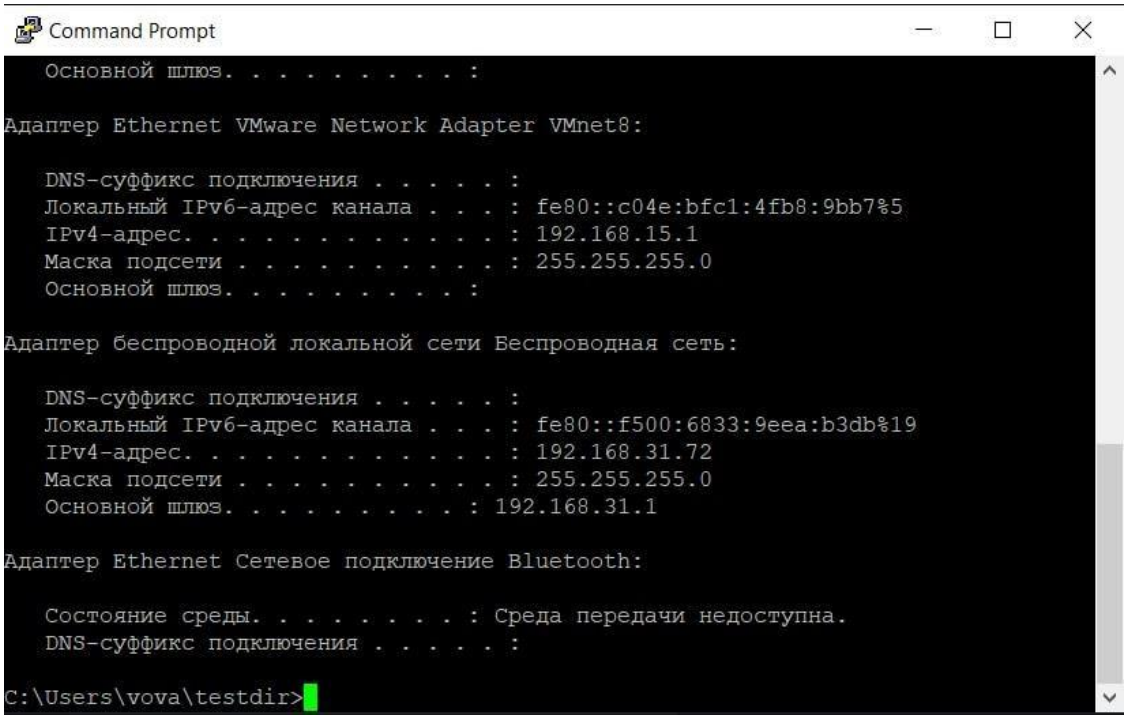


Рис 2.15. Приклад використання можливостей віддаленого доступу через інтерфейс командного рядка

Після вимкнення служби на хості В –з’єднання між хостами розірвалося.

Рис 2.16. Результат аналізу мережевого трафіку за протоколом TCP для комп'ютера А після вимкнення служби на комп’ютері В

2.3 Налаштування мережевого програмного комплексу на двох комп'ютерах з ОС Linux (Ubuntu)

Для зручності розгортання служби для ОС Linux створюємо автономну програму як один файл - середовище виконання .NET і всі необхідні залежності об'єднані в один виконуваний файл. Для цього виконаємо команду у каталозі с проектом служби WorkerService :

```
dotnet publish -c Release -r linux-x64 --self-contained=true -
p:PublishSingleFile=true -p:GenerateRuntimeConfigurationFiles=true -o artifacts
```

Далі створений виконуваний файл разом з конфігураційним xml файлом помістимо в папку /bin/TCP_proxy :

```
root@hanna-VirtualBox:/bin/TCP_proxy# ls
LinkConfig.xml  TCPlinux
root@hanna-VirtualBox:/bin/TCP_proxy#
```

Рис 2.17. Процес створення директиви TCP_proxy

По-друге, треба розмістити створений сервіс для операційної системи Linux на обох хостах з даною ОС (в прикладі, що розглядається далі - дистрибутив Ubuntu 20.04). Системний менеджер Linux (Systemd) використовує файли конфігурації служби, які визначають, що робить служба, чи потрібно її перезапускати, тощо.
Створюємо файл для конфігурації з розширенням .service :

```
root@hanna-VirtualBox:/home/hannaadmin# cd /etc/systemd/system
root@hanna-VirtualBox:/etc/systemd/system# >TCP_proxy.service
root@hanna-VirtualBox:/etc/systemd/system#
```

Рис 2.18. Процес створення файла для конфігурації з розширенням .service

Текст службового файлу виглядає так :

```
[Unit]
Description=TCP proxy service

[Service]
Type=notify
ExecStart=/usr/bin/TCP_proxy

[Install]
WantedBy=multi-user.target
```

Далі виконуємо таку команду, щоб Systemd завантажив цей новий файл конфігурації:
sudo systemctl daemon-reload

```
root@hanna-VirtualBox:/etc/systemd/system# systemctl daemon-reload
```

Далі запускаємо службу за допомогою наступної команди :
sudo systemctl start TCP_proxy.service

```
root@hanna-VirtualBox:/etc/systemd/system# systemctl start TCP_proxy.service
```

Переглянемо статус служби :
sudo systemctl status TCP_proxy.service

```
root@hanna-VirtualBox:/etc/systemd/system# systemctl status  TCP_proxy.service
● TCP_proxy.service - test service
     Loaded: loaded (/etc/systemd/system/TCP_proxy.service; disabled; vendor preset: enabled)
     Active: active (running) since Fri 2022-05-27 02:40:45 +05; 9s ago
   Main PID: 4009 (TCPlinux)
      Tasks: 17 (limit: 2103)
     Memory: 66.3M
     CGroup: /system.slice/TCP_proxy.service
             └─4009 /usr/bin/TCP_proxy/TCPlinux
```

Рис 2.19. Результат виконання команди статус для TCP_proxy.service

Служба виконується, але без заповненого конфігураційного файлу результатів її роботи ми не отримаємо.
Заповнимо файл LinkConfig.xml для двох хостів. Хост А – комп’ютер з якого поступати запит на отримання віддаленого доступу, а комп’ютер до якого буде отримано віддалений доступ – хост В.
IP адреса для хоста А - 192.168.56.101
IP адреса для хоста В - 192.168.56.102
Представлені IP адреси – адреси локальної мережі, але замість них можуть бути використані зовнішні IP адреси.

Конфігураційний файл Для хоста B :

```
<ServerControlLink>
        <EndPoint>
                <IP>0.0.0.0</IP>
                <PortNumber>52002</PortNumber>
        </EndPoint>
        <Sources>
                <SourcePoint>
                        <IPFrom>1.0.0.1</IPFrom>
                        <IPTo>223.255.255.254</IPTo>
                </SourcePoint>
        </Sources>
        <Key></Key>
  </ServerControlLink>
```

Рис 2.20. Приклад заповнення файлу конфігурації для машини до якої планується встановити віддалений доступ

Конфігураційний файл Для хоста A :

```
<RemoteGateLinks>
        <RemoteGateLink>
                <GateControlAddress>
                        <Address>192.168.56.102</Address>
                        <PortNumber>52002</PortNumber>
                </GateControlAddress>
                <GateEndPoint>
                        <IP>0.0.0.0</IP>
                        <PortNumber>32022</PortNumber>
                </GateEndPoint>
                <Key></Key>
                <ServiceAddress>
                        <Address>192.168.56.101</Address>
                        <PortNumber>22</PortNumber>
                </ServiceAddress>
                <Sources>
                        <SourcePoint>
                                <IPFrom>1.0.0.1</IPFrom>
                                <IPTo>223.255.255.254</IPTo>
                        </SourcePoint>
                </Sources>
        </RemoteGateLink>

</RemoteGateLinks>
```

Рис 2.21. Приклад заповнення файлу конфігурації для машини з якої планується встановити віддалений доступ

Після заповнення конфігураційних файлів обов’язково треба перезавантажити служби на обох комп’ютерах за допомогою команди sudo systemctl restart TCP_proxy.service

Після перезавантаження маємо змогу побачити результат виконання служби на обох хостах :
На хості B :

```
root@hanna-VirtualBox:/etc/systemd/system# netstat -ltnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      501/systemd-resolve
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      635/cupsd
tcp        0      0 0.0.0.0:52002           0.0.0.0:*               LISTEN      4009/TCPlinux
root@hanna-VirtualBox:/etc/systemd/system#
root@hanna-VirtualBox:/etc/systemd/system# netstat -an|grep ESTABLISHED
tcp        0      0 192.168.56.102:52002    192.168.56.101:49120    ESTABLISHED
udp        0      0 192.168.56.102:68       192.168.56.100:67       ESTABLISHED
```

Рис 2.22. Результат виконання команди netstat на хості B

На хості A :

```
root@hanna-VirtualBox:/home/hannaadmin# netstat -an|grep ESTABLISHED
tcp        0      0 192.168.56.101:49116    192.168.56.102:52002    ESTABLISHED
udp        0      0 192.168.56.101:68       192.168.56.100:67       ESTABLISHED
root@hanna-VirtualBox:/home/hannaadmin#
```

Рис 2.23. Результат виконання команди netstat на хості A

Між двома хостами маємо встановлений надійний TCP зв’язок, далі можна під’єднуватися з хоста A на хост B за допомогою SSH або іншого протоколу прикладного рівня, порт якого необхідно вказати у конфігураційному файлі.

2.4 Налаштування мережевого програмного комплексу між двома комп’ютерами з ОС Linux (Ubuntu) та ОС Windows 10

Алгоритм реєстрації служб для операційних систем Linux та Windows викладений у попередніх двох пунктах (2.3 та 2.4). Далі розглянемо лише налаштування конфігураційного файлу для хостів з ОС Linux та Windows та можливість встановлення надійного TCP зв’язку між комп’ютерами з ОС Linux та Windows за допомогою розроблених служб.

Хост A - з якого поступати запит на отримання віддаленого доступу (Ubuntu)
IP адреса - 192.168.56.103

```
<RemoteGateLinks>
        <RemoteGateLink>
                <GateControlAddress>
                        <Address>192.168.56.3</Address>
                        <PortNumber>52002</PortNumber>
                </GateControlAddress>
                <GateEndPoint>
                        <IP>0.0.0.0</IP>
                        <PortNumber>32022</PortNumber>
                </GateEndPoint>
                <Key></Key>
                <ServiceAddress>
                        <Address>192.168.56.103</Address>
                        <PortNumber>22</PortNumber>
                </ServiceAddress>
                <Sources>
                        <SourcePoint>
                                <IPFrom>1.0.0.1</IPFrom>
                                <IPTo>223.255.255.254</IPTo>
                        </SourcePoint>
                </Sources>
        </RemoteGateLink>

</RemoteGateLinks>
```

Рис 2.24. Приклад заповнення файлу конфігурації для машини з якої планується встановити віддалений доступ

Хост В - комп’ютер до якого буде отримано віддалений доступ (Windows)
IP адреса - 192.168.56.1

```
<ServerControlLink>
    <EndPoint>
        <IP>0.0.0.0</IP>
        <PortNumber>52002</PortNumber>
    </EndPoint>
    <Sources>
        <SourcePoint>
            <IPFrom>1.0.0.1</IPFrom>
            <IPTo>223.255.255.254</IPTo>
        </SourcePoint>
    </Sources>
    <Key></Key>
</ServerControlLink>
```

Рис 2.25. Приклад заповнення файлу конфігурації для машини до якої планується встановити віддалений доступ

Перезавантажимо служби на обоз хостах та перевіримо встановлене з’єднання :
На хості B :

```
C:\Users\Lenovo>netstat -an|findstr 52002
  TCP    0.0.0.0:52002          0.0.0.0:0              LISTENING
  TCP    192.168.56.3:52002     192.168.56.103:33944   ESTABLISHED
```

Рис 2.26. Результат виконання команди netstat на хості B

На хості A :

```
root@hanna-VirtualBox:/home/hannaadmin# netstat -atn
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.8:60116       192.168.1.4:139         ESTABLISHED
tcp        0      0 192.168.56.103:33944    192.168.56.3:52002      ESTABLISHED
root@hanna-VirtualBox:/home/hannaadmin# putty
```

Рис 2.27. Результат виконання команди netstat на хості A

За допомогою Putty з хоста A отримаємо віддалений доступ до хоста B :

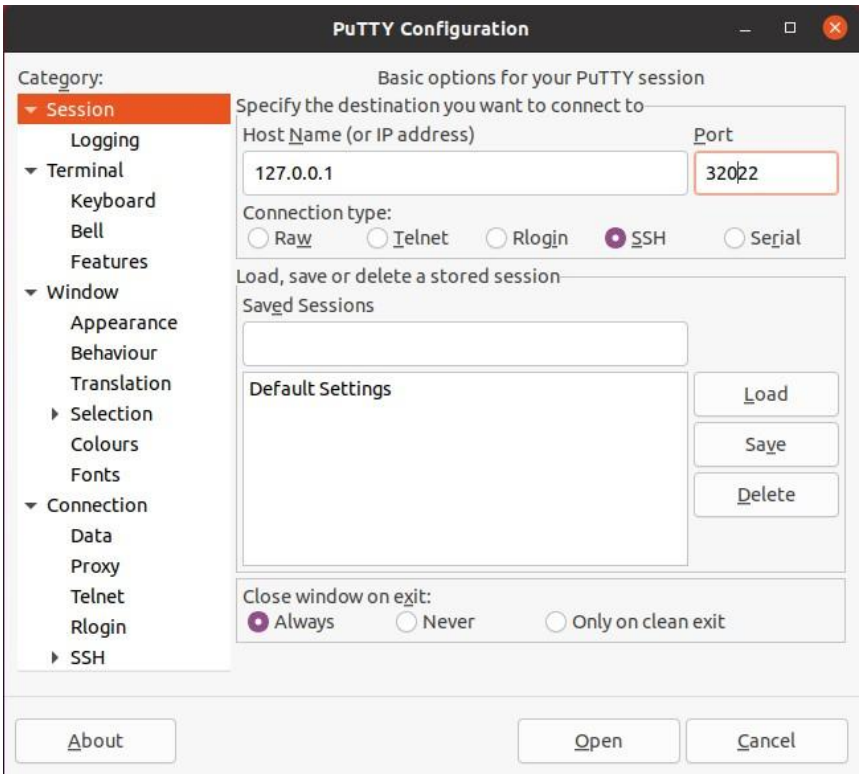


Рис 2.28. Приклад налаштування вікна програми Putty для встановлення віддаленого доступу за участю розробленої служби

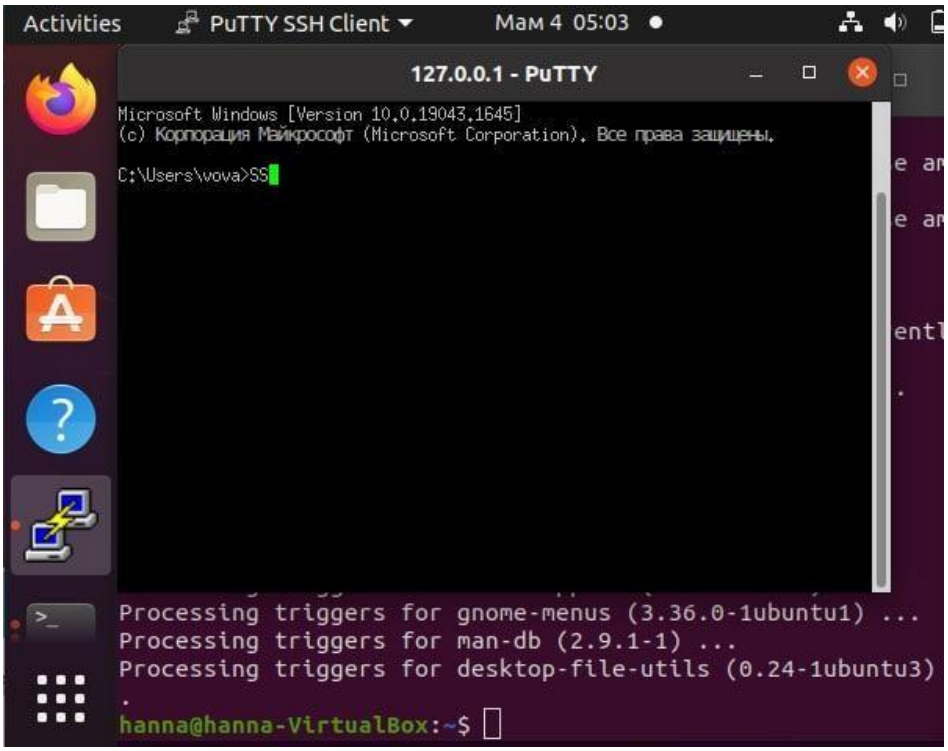


Рис 2.29. Представлений інтерфейс командного рядка для керування віддаленим комп’ютером

ВИСНОВКИ

Усі підприємства і установи, в тому числі КНУ імені Тараса Шевченка, мають локальну мережу, з'єднану з мережею Інтернет NAT-маршрутизатором. Через це підприємство може не мати можливостей для управління ресурсами із зовнішньої мережі за рядом причин: провайдер не виділив підприємству зовнішню IP-адресу, як це роблять мобільні оператори; зовнішня IP-адреса є динамічною і надто часто змінюється; підприємство не має доступу до NAT-маршрутизатора, який налаштовують працівники провайдера; підприємство має бюрократичну або надмірно жорстку політику, що забороняє дистанційні підключення. В роботі продемонстровано, що навіть в таких умовах організувати дистанційних доступ до внутрішніх ресурсів підприємства можна, якщо в його локальній мережі розмістити внутрішню частину служби TCP-проксі, яка буде утримувати постійне вихідне TCP-з'єднання з зовнішньою частиною цієї служби, розташованою далеко за межами підприємства і доступною за реальною статичною IP-адресою.

Створене програмне рішення показало, що платформа Microsoft .NET має вбудовану бібліотеку System.Net.Sockets, яка має повний набір методів, що дозволяють організувати необхідну маніпуляцію TCP-з'єднань між внутрішньою і зовнішньою частинами служби TCP-проксі, кожна з яких може бути запущена або як тимчасова прикладна програма або зареєстрована як перманентна системна служба операційної системи MS Windows або GNU Linux.

Розроблений мережевий програмний комплекс дозволяє авторизованому працівнику установи, підприємства, або власнику приватного житла заздалегідь налаштувати всередині відповідної приватної мережі довільну мережеву службу, зокрема SSH, HTTP, HTTPS, MySQL, SMTP, після чого мати змогу приєднатися до цієї налаштованої авторизованої служби, перебуваючи за межами відповідної установи, підприємства і у такий спосіб обійти бар'єр NAT-маршрутизатора підприємства, провайдера або мобільного оператора.

Можливість авторизованої дистанційної роботи з внутрішніми ресурсами підприємства є надзвичайно актуальною саме зараз, в умовах гострого дефіциту пального, в умовах карантинних обмежень, в умовах відкритої агресії з боку деспотичних тоталітарних імперії, коли необов'язкова поїздка на підприємство для того, щоб скопіювати на флешку «важливий» файл може коштувати цілого життя для працівника, який міг би виконати ту саму операцію не виходячи зі свого будинку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Essentials of TCP/IP : Lab-Based Approach 1st Edition, Shivendra S. Panwar, Shiwen Mao, Jeong-dong Ryoo, Yihan Li – 288 с.
2. Internetworking with TCP/IP, Vol. 1: Principles, Protocols and Architecture , 4st Edition, Douglas E. Comer – 783 с.
3. Remote desktop software The Ultimate Step-By-Step Guide, Gerardus Blokdyk, – 312 с.
4. Complete Book of Remote Access: Connectivity and Security (Best Practices 24) 1st Edition, Kindle Edition – 392 с.
5. C# Network Programming 1st Edition, Richard Blum – 656 с.
6. Transmission Control Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/data_communication_computer_network/transmission_control_protocol.htm
7. Transmission Control Protocol (protocol specification) [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc793
8. SSH Tunnel [Електронний ресурс] – Режим доступу до ресурсу: https://www.ssh.com/academy/ssh/tunneling
9. The Secure Shell (SSH) Connection Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/rfc4254/
10. TCP/IP ports [Електронний ресурс] – Режим доступу до ресурсу: http://www.steves-internet-guide.com/tcpip-ports-sockets/
11. Socket [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/api/system.net.sockets.socket?view=netframework-4.8