МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ НЭТИ | Факультет прикладной математики и информатики

КАФЕДРА ТЕОРЕТИЧЕСКОЙ И ПРИКЛАДНОЙ ИНФОРМАТИКИ

ЛАБОРАТОРНАЯ РАБОТА № 5

ПО ДИСЦИПЛИНЕ «ОПЕРАЦИОННЫЕ СИСТЕМЫ И КОМПЬЮТЕРНЫЕ СЕТИ»

АНАЛИЗ ФУНКЦИОНИРОВАНИЯ И ДИАГНОСТИКА IP-СЕТЕЙ

Бригада	ВЕСЕЛЫЙ ДЕНИС
№1	ВОРОНЧУК ИЛЬЯ
Группа	ПМИ-32
Преподаватели	КОБЫЛЯНСКИЙ В.Г.
	СИВАК М.А.

Новосибирск, 2025

Цель работы

Приобретение практических навыков работы с сетевыми командами операционных систем Windows и Linux, предназначенными для анализа и диагностики сетей TCP/IP, а также разработка собственного приложения для диагностики сети.

Ход выполнения работы

Этап 1: Диагностика IP-сетей с помощью стандартных утилит

1. Получение информации об операционной системе и аппаратной платформе

Задание: Подключиться к серверу fpm2.ami.nstu.ru и с помощью команды `uname` *получить полную информацию об установленной операционной системе и аппаратной платформе, полученный результат включить в отчет.*

Выполнение: С помощью команды `uname -a` на удаленном сервере `students.ami.nstu.ru` была получена полная информация об установленной операционной системе и аппаратной платформе. Результат выполнения команды приведен на рис. 1.

```
[pmi-b3701@students ~]$ uname -a
Linux students.ami.nstu.ru 3.10.0-327.3.1.el7.x86_64 #1 SMP Wed Dec 9 14:09:15 UTC
2015 x86_64 x86_64 x86_64 GNU/Linux
```

Рис. 1: Информация об ОС и платформе сервера

Пояснение к результату: Вывод команды `uname -a` содержит следующие сведения, представленные в строгом порядке:

- `Linux` — имя ядра операционной системы.
- `students.ami.nstu.ru` — сетевое имя узла (hostname).
- `3.10.0-327.3.1.el7.x86_64` — версия релиза ядра. В данном случае, это ядро версии 3.10.0, собранное для дистрибутива Enterprise Linux 7 (el7) под 64-битную архитектуру (x86_64).
- `#1 SMP Wed Dec 9 14:09:15 UTC 2015` — информация о сборке ядра: номер сборки (#1), указание на поддержку многопроцессорности (SMP) и дата/время сборки.
- `x86_64 x86_64 x86_64` — архитектура процессора, тип оборудования и аппаратная платформа.
- `GNU/Linux` — тип операционной системы.

2. Получение статистики по сетевым интерфейсам

Задание: Получить статистику по сетевым интерфейсам РК и сервера fpm2.ami.nstu.ru, пояснить результаты.

Была получена подробная информация о сетевых настройках локального рабочего компьютера (РК) под управлением Windows и удаленного сервера под управлением

Linux. Для этого были использованы команды `ipconfig /all` и `ifconfig` соответственно.

```
PS C:\Users\Admin> ipconfig /all

Настройка протокола IP для Windows

   Имя компьютера  . . . . . . . . . . : DENIS
   Основной DNS-суффикс  . . . . . . . :
   Тип узла. . . . . . . . . . . . . . : Гибридный
   IP-маршрутизация включена . . . . . : Нет
   WINS-прокси включен . . . . . . . . : Нет
   Порядок просмотра суффиксов DNS . . : Home

Адаптер Ethernet Router:

   DNS-суффикс подключения . . . . . : Home
   Описание. . . . . . . . . . . . . . : Intel(R) Ethernet Connection (2) I219-V
   Физический адрес. . . . . . . . . . : 70-85-C2-4F-92-76
   DHCP включен. . . . . . . . . . . . : Да
   Автонастройка включена. . . . . . . : Да
   Локальный IPv6-адрес канала . . . . : fe80::9b89:6cdc:bcdc:4c97%18(Основной)
   IPv4-адрес. . . . . . . . . . . . . : 192.168.0.2(Основной)
   Маска подсети . . . . . . . . . . . : 255.255.255.0
   Аренда получена. . . . . . . . . . : 10 ноября 2025 г. 16:42:01
   Срок аренды истекает. . . . . . . . . . : 11 ноября 2025 г. 18:07:45
   Основной шлюз. . . . . . . . . : 192.168.0.1
   DHCP-сервер. . . . . . . . . . . : 192.168.0.1
   IAID DHCPv6 . . . . . . . . . . . : 108037570
   DUID клиента DHCPv6 . . . . . . . : 00-01-00-01-2B-12-A8-89-70-85-C2-4F-92-76
   DNS-серверы. . . . . . . . . . . : 192.168.0.1
   NetBios через TCP/IP. . . . . . . . : Включен

Адаптер Ethernet vEthernet (Default Switch):

   DNS-суффикс подключения . . . . . :
   Описание. . . . . . . . . . . . . . : Hyper-V Virtual Ethernet Adapter
   Физический адрес. . . . . . . . . . : 00-15-5D-2B-4D-4F
   DHCP включен. . . . . . . . . . . . : Нет
   Автонастройка включена. . . . . . . : Да
   Локальный IPv6-адрес канала . . . . : fe80::e1b8:8818:9aea:c39b%30(Основной)
   IPv4-адрес. . . . . . . . . . . . . : 172.28.96.1(Основной)
   Маска подсети . . . . . . . . . . . : 255.255.240.0
   Основной шлюз. . . . . . . . . :
   IAID DHCPv6 . . . . . . . . . . . : 503321949
   DUID клиента DHCPv6 . . . . . . . : 00-01-00-01-2B-12-A8-89-70-85-C2-4F-92-76
   NetBios через TCP/IP. . . . . . . . : Включен
```

Рис. 2: Сетевые настройки рабочего компьютера (Windows)

Пояснение к результатам на РК (рис. 2): Вывод команды `ipconfig /all` показывает детальную конфигурацию всех сетевых адаптеров. Можно выделить два основных активных интерфейса:

- **Адаптер `Ethernet Router`:** Это основной физический сетевой адаптер (`Intel(R) Ethernet Connection I219-V`), через который компьютер подключен к локальной сети и Интернету.

- **Физический адрес (MAC):** `70-85-C2-4F-92-76` — уникальный идентификатор сетевой карты.
- **IPv4-адрес:** `192.168.0.2` — текущий IP-адрес компьютера в локальной сети. Он был получен автоматически, так как DHCP включен (`Да`).
- **Маска подсети:** `255.255.255.0` — определяет, какая часть IP-адреса относится к сети, а какая — к узлу.
- **Основной шлюз:** `192.168.0.1` — IP-адрес маршрутизатора (роутера), через который компьютер получает доступ к другим сетям, включая Интернет.
- **DHCP-сервер и DNS-серверы:** имеют тот же адрес `192.168.0.1`, что указывает на то, что функции автоматической раздачи адресов и разрешения доменных имен выполняет домашний роутер.

- **Адаптер `vEthernet (Default Switch)`:** Это виртуальный сетевой интерфейс, созданный системой виртуализации Hyper-V. Он функционирует как виртуальный коммутатор для подключения виртуальных машин к сети хоста. Ему присвоена собственная подсеть (`172.28.96.1` с маской `255.255.240.0`), что важно для последующего анализа работы команды `arp`.

```
[pmi-b3701@students ~]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 217.71.130.131  netmask 255.255.255.128  broadcast 217.71.130.255
        inet6 2001:b08:a:1040:400c:7ff:fef2:8a56  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::400c:7ff:fef2:8a56  prefixlen 64  scopeid 0x20<link>
        inet6 2001:b08:a:1040:19ee:5fc5:451f:5cf  prefixlen 128  scopeid 0x0<global
>
        ether 42:0c:07:f2:8a:56  txqueuelen 1000  (Ethernet)
        RX packets 68718095  bytes 15605284949 (14.5 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16186589  bytes 5552244313 (5.1 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 12652006  bytes 12628395059 (11.7 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12652006  bytes 12628395059 (11.7 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:d4:60:b6  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Рис. 3: Сетевые настройки сервера (Linux)

Пояснение к результатам на сервере (рис. 3): Вывод команды `ifconfig` показывает информацию об интерфейсах сервера.

- **Интерфейс `eth0`:** Это основной сетевой интерфейс, через который сервер взаимодействует с внешней сетью.

  - **IPv4-адрес** (`inet`)**:** `217.71.130.131`.
  - **Маска подсети** (`netmask`)**:** `255.255.255.128`.
  - **MAC-адрес** (`ether`)**:** `42:0c:07:f2:8a:56`.
  - **MTU:** `1500` — максимальный размер пакета, который может быть передан через этот интерфейс без фрагментации.
  - **Статистика** `RX/TX packets`**:** Показывает огромное количество принятых (RX, 687 млн) и отправленных (TX, 156 млн) пакетов, что свидетельствует о высокой активности сервера.

- **Интерфейс** `lo`**:** Это "петлевой"интерфейс (loopback) с адресом `127.0.0.1`, используемый для отладки и взаимодействия служб на самом сервере.
- **Интерфейс** `virbr0`**:** Это виртуальный мост, используемый системой виртуализации для связи между виртуальными машинами и хостовой системой.

3. Просмотр содержимого DNS-кэша

Задание: Просмотреть содержимое DNS-кэша, пояснить характеристики записей, очистить кэш.

Был выполнен просмотр и последующая очистка DNS-кэша на локальном рабочем компьютере с помощью утилиты `ipconfig`.

Рис. 4: Фрагмент содержимого DNS-кэша до очистки

Пояснение к рис. 4: На скриншоте показан фрагмент вывода команды `ipconfig /displaydns`. DNS-кэш хранит временные записи о соответствии доменных имен и IP-адресов, чтобы не обращаться к DNS-серверу при каждом запросе к одному и тому же ресурсу. Каждая запись в кэше имеет следующие характеристики:

- **Имя записи (Record Name):** Доменное имя, например, `google.com`.
- **Тип записи (Record Type):** Тип DNS-записи. Например, `1` — это **A-запись**, которая связывает доменное имя с IPv4-адресом.
- **Срок жизни (Time To Live, TTL):** Время в секундах, в течение которого эта запись считается актуальной. На скриншоте видно, что TTL для записей `google.com` равен 254 секундам. По истечении этого времени запись будет удалена из кэша.
- **A-запись (узла):** IP-адрес, соответствующий доменному имени.

```
PS C:\Users\Admin> ipconfig /flushdns

Настройка протокола IP для Windows

Кэш сопоставителя DNS успешно очищен.
PS C:\Users\Admin> ipconfig /displaydns

Настройка протокола IP для Windows

    8.8.8.8.in-addr.arpa
    ----------------------------------------
    Имя записи. . . . . : 8.8.8.8.in-addr.arpa.
    Тип записи. . . . . : 12
    Срок жизни. . . . . : 580716
    Длина данных. . . . . : 8
    Раздел. . . . . : Ответ
    PTR-запись. . . . . : www.dropbox.com


    1.96.28.172.in-addr.arpa
    ----------------------------------------
    Имя записи. . . . . : 1.96.28.172.in-addr.arpa.
    Тип записи. . . . . : 12
    Срок жизни. . . . . : 580716
    Длина данных. . . . . : 8
    Раздел. . . . . : Ответ
    PTR-запись. . . . . : DENIS.mshome.net


    kubernetes.docker.internal
    ----------------------------------------
    Нет записей типа AAAA
```

Рис. 5: Результат очистки DNS-кэша

Пояснение к рис. 5: Команда `ipconfig /flushdns` успешно очистила кэш DNS. Повторный вызов `ipconfig /displaydns` показывает, что старые записи (например, для `google.com`) исчезли. При этом в кэше могут оставаться статические записи, например, из файла `hosts` (как `kubernetes.docker.internal`), или записи, которые были немедленно добавлены системными службами после очистки.

4. Просмотр и изменение содержимого ARP-таблицы

Задание: Просмотреть содержимое ARP-таблицы, пояснить характеристики записей, выполнить добавление и удаление статических записей.

Были просмотрены ARP-таблицы на локальном компьютере (PK) и на удаленном сервере. ARP-протокол (Address Resolution Protocol) используется для преобразования IP-адресов в физические MAC-адреса в пределах одного сегмента сети.

```
PS C:\Users\Admin> arp -a

Интерфейс: 192.168.0.2 --- 0x12
  адрес в Интернете      Физический адрес      Тип
  192.168.0.1           a8-f9-4b-13-4c-4f     динамический
  192.168.0.255         ff-ff-ff-ff-ff-ff     статический
  224.0.0.22            01-00-5e-00-00-16     статический
  224.0.0.251           01-00-5e-00-00-fb     статический
  224.0.0.252           01-00-5e-00-00-fc     статический
  239.255.255.250       01-00-5e-7f-ff-fa     статический
  255.255.255.255       ff-ff-ff-ff-ff-ff     статический

Интерфейс: 172.28.96.1 --- 0x1e
  адрес в Интернете      Физический адрес      Тип
  172.28.111.255        ff-ff-ff-ff-ff-ff     статический
  224.0.0.22            01-00-5e-00-00-16     статический
  224.0.0.251           01-00-5e-00-00-fb     статический
  239.255.255.250       01-00-5e-7f-ff-fa     статический
  255.255.255.255       ff-ff-ff-ff-ff-ff     статический
```

Рис. 6: ARP-таблица на PK (Windows) до изменений

```
[pmi-b3701@students ~]$ arp -a
gw-130-204.ami.nstu.ru (217.71.130.252) at 00:21:1b:ee:dd:c4 [ether] on eth0
gw-130.ami.nstu.ru (217.71.130.254) at 00:00:0c:9f:f0:82 [ether] on eth0
pc-307-03.ami.nstu.ru (217.71.130.223) at 00:15:5d:82:8d:cb [ether] on eth0
pmt-08.ami.nstu.ru (217.71.130.188) at e2:41:f0:ec:ac:a5 [ether] on eth0
screamer.ami.nstu.ru (217.71.130.148) at f8:32:e4:89:1a:8d [ether] on eth0
gw-130-208v.ami.nstu.ru (217.71.130.251) at 00:21:1b:f5:76:46 [ether] on eth0
ksc.ami.nstu.ru (217.71.130.196) at 9e:7f:ba:21:be:c1 [ether] on eth0
fpm.ami.nstu.ru (217.71.130.130) at 00:1e:0b:d9:84:48 [ether] on eth0
armor.ami.nstu.ru (217.71.130.150) at 2c:4d:54:51:91:59 [ether] on eth0
gate.ami.nstu.ru (217.71.130.129) at d4:8c:b5:4d:f6:5b [ether] on eth0
zion.ami.nstu.ru (217.71.130.154) at e0:d5:5e:aa:1e:b6 [ether] on eth0
```

Рис. 7: ARP-таблица на сервере (Linux)

Пояснение к исходным таблицам (рис. 6 и 7): На PK (Windows) вывод сгруппирован по сетевым интерфейсам. Для интерфейса `192.168.0.2` (физический адаптер) видна одна **динамическая** запись для основного шлюза `192.168.0.1`. Динамические записи добавляются автоматически, когда компьютер обращается к другому узлу в локальной сети, и имеют ограниченное время жизни.

Остальные записи являются **статическими** и используются для служебного трафика:

- **Широковещательные (Broadcast):**

  Записи для IP-адресов `192.168.0.255` и `255.255.255.255` сопоставлены с MAC-адресом `ff-ff-ff-ff-ff-ff`. Этот специальный адрес гарантирует, что кадр будет получен всеми устройствами в локальной сети.

- **Групповые (Multicast):**

  Записи для IP-адресов из диапазона `224.0.0.0/4` (например, `224.0.0.251` для mDNS) сопоставлены со специальными MAC-адресами, начинающимися с `01-00-5e`. Они используются для отправки данных определенной группе устройств.

Добавление статической ARP-записи. Была выполнена попытка добавить статическую запись для IP-адреса `192.168.1.250`.

```
PS C:\Users\Admin> arp -s 192.168.1.250 00-11-22-33-44-55
PS C:\Users\Admin> arp -a

Интерфейс: 192.168.0.2 --- 0x12
  адрес в Интернете      Физический адрес      Тип
  192.168.0.1           a8-f9-4b-13-4c-4f     динамический
  192.168.0.255         ff-ff-ff-ff-ff-ff     статический
  224.0.0.22            01-00-5e-00-00-16     статический
  224.0.0.251           01-00-5e-00-00-fb     статический
  224.0.0.252           01-00-5e-00-00-fc     статический
  239.255.255.250       01-00-5e-7f-ff-fa     статический
  255.255.255.255       ff-ff-ff-ff-ff-ff     статический

Интерфейс: 172.28.96.1 --- 0x1e
  адрес в Интернете      Физический адрес      Тип
  172.28.111.255        ff-ff-ff-ff-ff-ff     статический
  192.168.1.250         00-11-22-33-44-55     статический
  224.0.0.22            01-00-5e-00-00-16     статический
  224.0.0.251           01-00-5e-00-00-fb     статический
  239.255.255.250       01-00-5e-7f-ff-fa     статический
  255.255.255.255       ff-ff-ff-ff-ff-ff     статический
```

Рис. 8: Добавление статической записи в ARP-таблицу на РК

Пояснение (рис. 8): Команда `arp -s 192.168.1.250 00-11-22-33-44-55` была выполнена в командной строке с правами администратора. Интересно, что Windows автоматически определила, что IP-адрес `192.168.1.250` не относится к сети физического адаптера (`192.168.0.0/24`). Однако, он также не относится и к сети виртуального адаптера Hyper-V (`172.28.96.0/20`). Из-за этого, система добавила запись в таблицу для интерфейса с наиболее близкой метрикой или настройками, которым оказался виртуальный интерфейс `172.28.96.1`. Новая статическая запись видна в выводе `arp -a`.

Удаление статической ARP-записи. Ранее добавленная статическая запись была удалена.

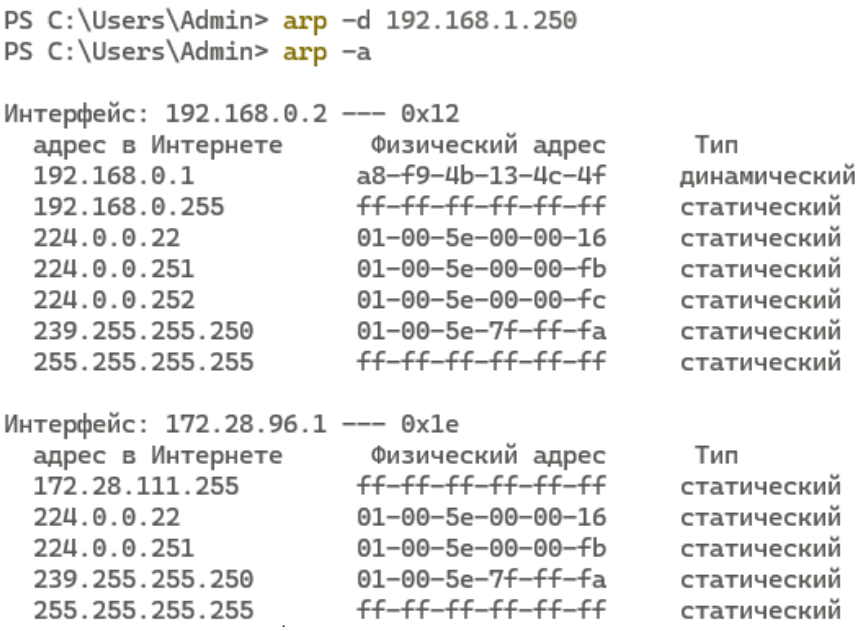

Рис. 9: Удаление статической записи из ARP-таблицы на РК

Пояснение (рис. 9): Команда `arp -d 192.168.1.250` успешно удалила статическое сопоставление. Повторный вызов `arp -a` показывает, что запись `192.168.1.250` исчезла из таблицы для интерфейса `172.28.96.1`.

5. Просмотр содержимого таблицы маршрутизации

Задание: Просмотреть содержимое таблицы маршрутизации, пояснить характеристики записей.

Были просмотрены таблицы маршрутизации на локальном компьютере (РК) и на удаленном сервере. Таблица маршрутизации используется операционной системой для определения, через какой сетевой интерфейс и на какой шлюз отправлять IP-пакеты, предназначенные для разных сетей.

```
PS C:\Users\Admin> route print
===========================================================================
Список интерфейсов
 18...70 85 c2 4f 92 76 ......Intel(R) Ethernet Connection (2) I219-V
  1...........................Software Loopback Interface 1
 30...00 15 5d 2b 4d 4f ......Hyper-V Virtual Ethernet Adapter
===========================================================================

IPv4 таблица маршрута
===========================================================================
Активные маршруты:
Сетевой адрес           Маска сети      Адрес шлюза       Интерфейс  Метрика
          0.0.0.0          0.0.0.0      192.168.0.1      192.168.0.2     25
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
      172.28.96.0    255.255.240.0         On-link       172.28.96.1   5256
      172.28.96.1  255.255.255.255         On-link       172.28.96.1   5256
   172.28.111.255  255.255.255.255         On-link       172.28.96.1   5256
      192.168.0.0    255.255.255.0         On-link       192.168.0.2    281
      192.168.0.2  255.255.255.255         On-link       192.168.0.2    281
    192.168.0.255  255.255.255.255         On-link       192.168.0.2    281
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link       192.168.0.2    281
        224.0.0.0        240.0.0.0         On-link       172.28.96.1   5256
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link       192.168.0.2    281
  255.255.255.255  255.255.255.255         On-link       172.28.96.1   5256
===========================================================================
Постоянные маршруты:
  Сетевой адрес           Маска    Адрес шлюза      Метрика
          0.0.0.0          0.0.0.0         25.0.0.1  По умолчанию
===========================================================================
```

Рис. 10: Таблица маршрутизации на РК (Windows)

Пояснение к таблице на РК (рис. 10): Таблица маршрутизации Windows содержит несколько ключевых записей. Наиболее важной является **маршрут по умолчанию**:

Сетевой адрес	Маска сети	Адрес шлюза	Интерфейс	Метрика
0.0.0.0	0.0.0.0	192.168.0.1	192.168.0.2	25

Эта запись означает, что любой пакет, IP-адрес назначения которого не соответствует ни одной другой, более специфичной записи в таблице, будет отправлен на шлюз `192.168.0.1` (адрес роутера) через локальный интерфейс `192.168.0.2`. **Метрика 25** — это "стоимость"маршрута; система выберет маршрут с наименьшей метрикой, если будет несколько вариантов. Также в таблице присутствуют маршруты для локальных сетей (`192.168.0.0/24`, `172.28.96.0/20`), широковещательных и групповых адресов.

```
[pmi-b3701@students ~]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         217.71.130.254  0.0.0.0         UG    100    0        0 eth0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
217.71.130.128  0.0.0.0         255.255.255.128 U     100    0        0 eth0
```

Рис. 11: Таблица маршрутизации на сервере (Linux)

Пояснение к таблице на сервере (рис. 11): Таблица маршрутизации Linux имеет схожую структуру. Ключевые записи:

- **Маршрут по умолчанию:**

```
Destination  Gateway          Genmask      Flags  Iface
0.0.0.0      217.71.130.254   0.0.0.0      UG     eth0
```

Здесь `Destination 0.0.0.0` также означает "любая сеть". Пакеты для таких сетей будут отправлены на шлюз `217.71.130.254` через интерфейс `eth0`. Флаги `UG` означают, что маршрут активен (`U`) и использует шлюз (`G`).

- **Локальные маршруты:** Записи для сетей `192.168.122.0` и `217.71.130.128` описывают сети, напрямую подключенные к интерфейсам `virbr0` и `eth0` соответственно. Для них шлюз (`Gateway`) не нужен (`0.0.0.0`), и пакеты отправляются напрямую.

6. Определение IP-адресов поисковых систем

*Задание: В командном режиме на РК и на сервере определить IP-адреса поисковых систем в соответствии с вариантом (**Вариант 1**: google.com), пояснить результаты.*

Были определены IP-адреса для доменного имени `google.com` с помощью утилиты `nslookup` как на локальном компьютере (РК), так и на удаленном сервере.

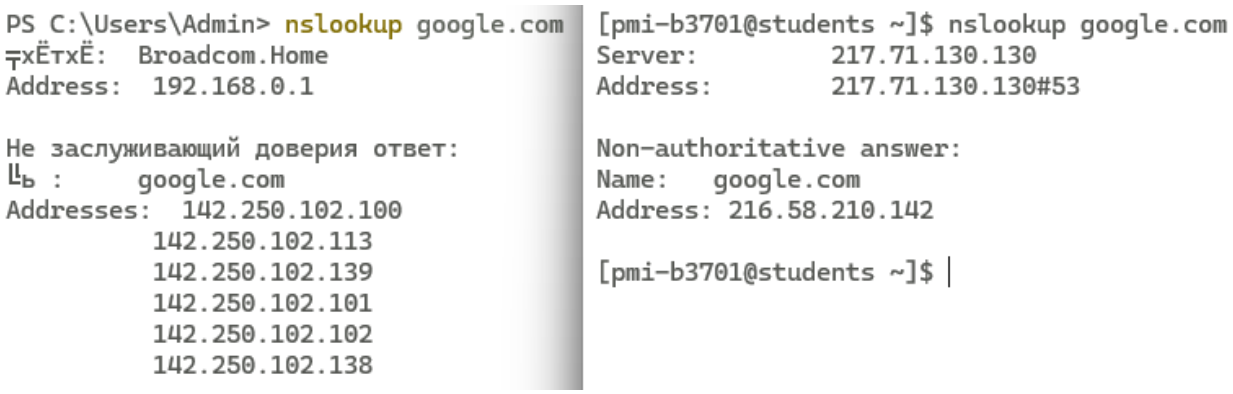

Рис. 12: Определение IP-адресов для google.com на РК и сервере

Пояснение к результатам (рис. 12): Утилита `nslookup` отправляет запрос к настроенному по умолчанию DNS-серверу для получения IP-адреса, связанного с указанным доменным именем.

- **На РК (Windows):** Запрос был отправлен DNS-серверу `192.168.0.1` (домашний роутер). В ответ было получено **шесть** различных IPv4-адресов для домена `google.com`.
- **На сервере (Linux):** Запрос был отправлен DNS-серверу `217.71.130.130`. В ответ был получен **один** IPv4-адрес.

Основной вывод: Крупные сервисы, такие как Google, используют множество серверов, распределенных по всему миру. DNS-серверы настроены так, чтобы возвращать разные IP-адреса в зависимости от местоположения запрашивающего, загруженности серверов и других факторов. Это делается для:

- **Балансировки нагрузки (Load Balancing):** Распределение запросов пользователей между множеством физических серверов.

- **Отказоустойчивости (Fault Tolerance):** Если один сервер выйдет из строя, трафик будет автоматически перенаправлен на другие.
- **Географической оптимизации (GeoDNS):** Возвращается IP-адрес сервера, который географически находится ближе к пользователю, что уменьшает задержки.

Отличие в количестве возвращаемых адресов на РК и сервере объясняется тем, что они используют разные DNS-серверы с разной политикой ответа.

7. Пингование и трассировка узлов сети

*Задание: В командном режиме на РК и на сервере определить IP-адрес узлов сети в соответствии с номером варианта, выполнить его пингование и трассировку. (**Вариант 1**: mit.edu, vk.com).*

Была выполнена проверка доступности и трассировка маршрута до узлов `mit.edu` и `vk.com` с локального компьютера (РК) и с удаленного сервера.

Анализ маршрута до `mit.edu`.

```
PS C:\Users\Admin> ping mit.edu

Обмен пакетами с mit.edu [88.221.97.35] с 32 байтами данных:
Ответ от 88.221.97.35: число байт=32 время=64мс TTL=53
Ответ от 88.221.97.35: число байт=32 время=62мс TTL=53
Ответ от 88.221.97.35: число байт=32 время=63мс TTL=53
Ответ от 88.221.97.35: число байт=32 время=63мс TTL=53

Статистика Ping для 88.221.97.35:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 62мсек, Максимальное = 64 мсек, Среднее = 63 мсек
PS C:\Users\Admin> tracert mit.edu

Трассировка маршрута к mit.edu [88.221.97.35]
с максимальным числом прыжков 30:

  1    <1 мс    <1 мс    <1 мс  Broadcom.Home [192.168.0.1]
  2     2 ms     2 ms     3 ms  10.145.17.254
  3     *        7 ms     4 ms  172.16.100.29
  4     4 ms     3 ms     3 ms  172.16.200.100
  5     *        9 ms     5 ms  host-62-9.nir-telecom.ru [212.15.62.9]
  6     4 ms     2 ms     3 ms  178.178.107.44
  7     *        *        *     Превышен интервал ожидания для запроса.
  8   103 ms   115 ms    98 ms  netnod-ix-ge-a-sth-1500.akamai.com [194.68.123.130]

  9     *        *        *     Превышен интервал ожидания для запроса.
 10     *        *        *     Превышен интервал ожидания для запроса.
 11     *        *        *     Превышен интервал ожидания для запроса.
 12    66 ms    62 ms    63 ms  a88-221-97-35.deploy.static.akamaitechnologies.com
[88.221.97.35]

Трассировка завершена.
```

Рис. 13: Пинг и трассировка до mit.edu с РК (Windows)

Пояснение к рис. 13: Утилита `ping` показывает, что узел `mit.edu` (IP: `88.221.97.35`) доступен, среднее время отклика (RTT) составляет 63 мс. Утилита `tracert` показывает маршрут до цели. Видно, что пакет проходит через 12 маршрутизаторов (хопов). Первые узлы (`192.168.0.1`, `10.145.17.254`) принадлежат локальной сети и сети провайдера. Промежуточные узлы с 7 по 11 не отвечают на ICMP-запросы (отмечены звездочками '*'), что является стандартной практикой безопасности для многих магистральных маршрутизаторов. Конечный узел принадлежит сети `akamai.com` — это крупная CDN-сеть (Content Delivery Network), что говорит о том, что сайт `mit.edu` использует сервисы Akamai для быстрой доставки контента.

```
[pmi-b3701@students ~]$ ping -c 4 mit.edu
PING mit.edu (23.210.114.10) 56(84) bytes of data.
From gw-130.ami.nstu.ru (217.71.130.254): icmp_seq=1 Redirect Network(New nexthop: gate.ami.nstu.ru (217.71.130.1
29))
From gw-130.ami.nstu.ru (217.71.130.254) icmp_seq=1 Redirect Network64 bytes from a23-210-114-10.deploy.static.ak
amaitechnologies.com (23.210.114.10): icmp_seq=1 ttl=49 time=83.1 ms
64 bytes from a23-210-114-10.deploy.static.akamaitechnologies.com (23.210.114.10): icmp_seq=2 ttl=49 time=83.0 ms
64 bytes from a23-210-114-10.deploy.static.akamaitechnologies.com (23.210.114.10): icmp_seq=3 ttl=49 time=83.1 ms

--- mit.edu ping statistics ---
3 packets transmitted, 3 received, +1 errors, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 83.046/83.109/83.146/0.044 ms
[pmi-b3701@students ~]$ traceroute mit.edu
traceroute to mit.edu (23.210.114.10), 30 hops max, 60 byte packets
 1  gw-130-208v.ami.nstu.ru (217.71.130.251)  3.244 ms  3.448 ms  3.698 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

Рис. 14: Пинг и трассировка до mit.edu с сервера (Linux)

Пояснение к рис. 14: На сервере `ping` также показывает доступность узла, но при этом наблюдаются ICMP-сообщения `Redirect Network`. Это означает, что шлюз по умолчанию (`gw-130.ami.nstu.ru`) информирует сервер о наличии более оптимального маршрута в пределах локальной сети НГТУ. Утилита `traceroute` не смогла построить полный маршрут. После первого же узла (`gw-130-208v.ami.nstu.ru`) все последующие маршрутизаторы не отвечают. Это, скорее всего, связано с настройками безопасности сети НГТУ, которые блокируют исходящие UDP или ICMP пакеты с малым TTL, используемые утилитой `traceroute`.

Анализ маршрута до `vk.com`.

```
PS C:\Users\Admin> ping vk.com

Обмен пакетами с vk.com [87.240.129.133] с 32 байтами данных:
Ответ от 87.240.129.133: число байт=32 время=54мс TTL=52
Ответ от 87.240.129.133: число байт=32 время=53мс TTL=52
Ответ от 87.240.129.133: число байт=32 время=53мс TTL=52
Ответ от 87.240.129.133: число байт=32 время=52мс TTL=52

Статистика Ping для 87.240.129.133:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 52мсек, Максимальное = 54 мсек, Среднее = 53 мсек
PS C:\Users\Admin> tracert vk.com

Трассировка маршрута к vk.com [87.240.129.133]
с максимальным числом прыжков 30:

  1    <1 мс    <1 мс    10 ms  Broadcom.Home [192.168.0.1]
  2     2 ms     2 ms     3 ms  10.145.17.254
  3     9 ms     7 ms     6 ms  172.16.100.29
  4     2 ms     2 ms     3 ms  172.16.200.100
  5    10 ms     4 ms     3 ms  host-62-9.nir-telecom.ru [212.15.62.9]
  6     5 ms     6 ms     3 ms  178.178.107.44
  7     *        *        *     Превышен интервал ожидания для запроса.
  8    50 ms    49 ms    49 ms  178.176.137.3
  9     *        *        *     Превышен интервал ожидания для запроса.
 10     *        *        *     Превышен интервал ожидания для запроса.
 11     *        *        *     Превышен интервал ожидания для запроса.
 12     *        *        *     Превышен интервал ожидания для запроса.
 13    53 ms    52 ms    53 ms  srv133-129-240-87.vk.com [87.240.129.133]
```

Рис. 15: Пинг и трассировка до vk.com с РК (Windows)

Пояснение к рис. 15: Узел `vk.com` (IP: `87.240.129.133`) доступен, среднее время отклика — 53 мс. Маршрут до него состоит из 13 хопов. Как и в предыдущем случае, часть промежуточных маршрутизаторов не отвечает на запросы.

```
[pmi-b3701@students ~]$ ping -c 4 vk.com
PING vk.com (87.240.132.67) 56(84) bytes of data.
64 bytes from srv67-132-240-87.vk.com (87.240.132.67): icmp_seq=1 ttl=50 time=55.0 ms
From gw-130.ami.nstu.ru (217.71.130.254): icmp_seq=2 Redirect Network(New nexthop: gate.
ami.nstu.ru (217.71.130.129))
From gw-130.ami.nstu.ru (217.71.130.254) icmp_seq=2 Redirect Network64 bytes from srv67-
132-240-87.vk.com (87.240.132.67): icmp_seq=2 ttl=50 time=54.9 ms
64 bytes from srv67-132-240-87.vk.com (87.240.132.67): icmp_seq=3 ttl=50 time=55.0 ms

--- vk.com ping statistics ---
3 packets transmitted, 3 received, +1 errors, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 54.977/55.016/55.059/0.272 ms
[pmi-b3701@students ~]$ traceroute vk.com
traceroute to vk.com (87.240.129.133), 30 hops max, 60 byte packets
 1  gw-130-208v.ami.nstu.ru (217.71.130.251)  1.462 ms  1.710 ms  1.992 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

Рис. 16: Пинг и трассировка до vk.com с сервера (Linux)

Пояснение к рис. 16: Ситуация аналогична трассировке до `mit.edu` с сервера. `ping` работает (и также получает ICMP Redirect), а `traceroute` не может пройти дальше первого шлюза из-за сетевых политик безопасности. Это подтверждает, что проблема не в конечном узле, а в конфигурации исходящего трафика на сервере или в сети университета.

8. Использование интерактивных сетевых сервисов

Задание: С помощью интерактивных сетевых сервисов (например, ping-admin.ru) выполнить трассировку, определить местонахождение и владельца узла сети в соответствии с номером варианта. Результат трассировки в виде скриншота географической карты представить в отчете и выполнить его анализ. Начальный пункт трассировки — г. Новосибирск.

Была выполнена трассировка маршрутов до узлов `mit.edu` и `vk.com` с помощью онлайн-сервиса `ping-admin.ru`, позволяющего визуализировать путь пакетов на географической карте.

Анализ маршрута до `mit.edu`.

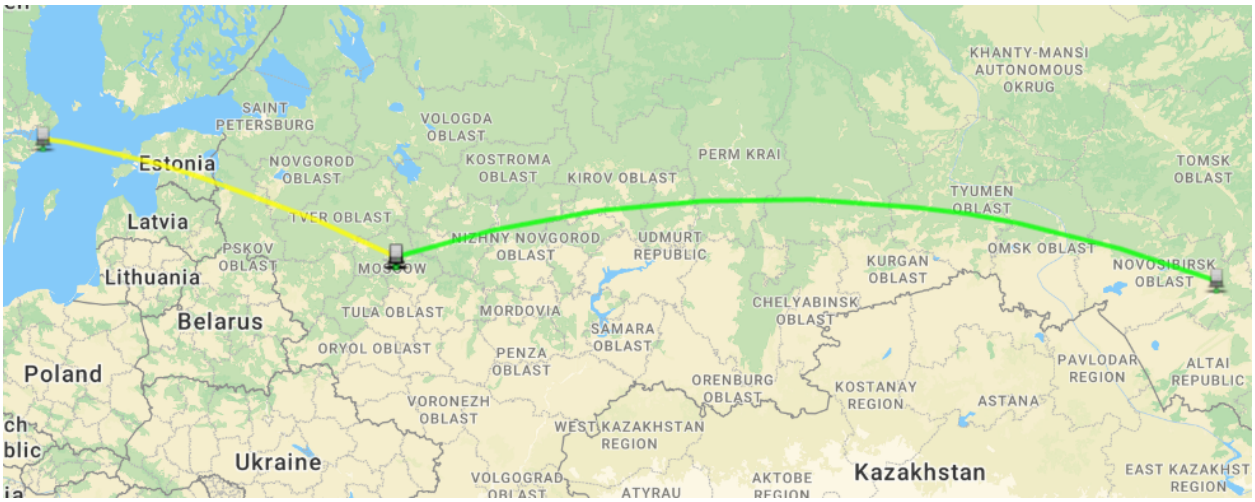

Рис. 17: Визуализация трассировки маршрута до mit.edu

№	Хост	IP	AS	Время, мс
1.	192.168.10.1	192.168.10.1		79,873
2.	217.65.85.193	217.65.85.193	AS12389	94,948
3.	217.107.120.165	217.107.120.165	AS12389	142,196
4.	a88-221-97-35.deploy.static.akamaitechnologies.com	88.221.97.35	AS16625	140,596

Рис. 18: Таблица узлов маршрута до mit.edu

Пояснение к результатам (рис. 17 и 18): Трассировка до `mit.edu` показывает международный маршрут.

- **Маршрут:** Пакет начинает свой путь из Новосибирска, проходит через узлы в Москве, затем уходит в Европу (вероятно, Стокгольм, судя по карте) и достигает конечного узла.
- **Автономные системы (AS):** Первые узлы после локального шлюза (`192.168.10.1`) принадлежат провайдеру Ростелеком (`AS12389`). Конечный узел (`88.221.97.35`) принадлежит автономной системе `AS16625`, которая принадлежит компании **Akamai Technologies**.
- **Вывод:** Это подтверждает анализ из предыдущего пункта: сайт `mit.edu` использует глобальную сеть доставки контента (CDN) Akamai. Запрос из России обслуживается не головным сервером MIT в США, а ближайшим к пользователю сервером Akamai, расположенным в Европе.

Анализ маршрута до `vk.com`.

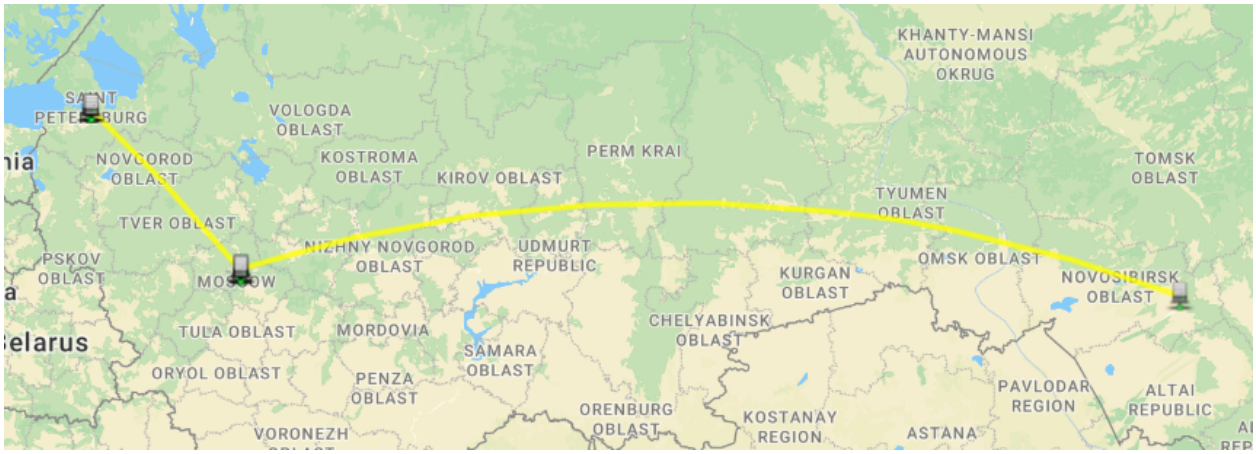

Рис. 19: Визуализация трассировки маршрута до vk.com

№	Хост	IP	AS	Время, мс
1.	192.168.10.1	192.168.10.1		92,772
2.	217.65.85.193	217.65.85.193	AS12389	109,353
3.	188.254.2.149	188.254.2.149	AS12389	145,248
4.	85.175.225.154	85.175.225.154	AS12389	143,245
5.	srv133-129-240-87.vk.com	87.240.129.133	AS47541	145,914

Рис. 20: Таблица узлов маршрута до vk.com

Пояснение к результатам (рис. 19 и 20): Трассировка до `vk.com` показывает полностью внутренний, российский маршрут.

- **Маршрут:** Пакет идет из Новосибирска в Москву, а далее в Санкт-Петербург, где и находится конечный сервер.
- **Автономные системы (AS):** Маршрут также начинается в сети Ростелеком (`AS12389`). Однако конечный узел `87.240.129.133` принадлежит автономной системе `AS47541`, владельцем которой является непосредственно компания **VK**.
- **Вывод:** В отличие от `mit.edu`, сервис `vk.com` для данного запроса обслуживается собственными серверами, расположенными в России. Это демонстрирует два разных подхода к хостингу: использование глобального CDN и размещение на собственных мощностях в целевом регионе.

Этап 2: Разработка приложения для диагностики сети

*Задание: Реализовать Windows-приложение, которое будет выполнять основные функции одной из утилит мониторинга сети в соответствии с таблицей. Результат работы функций должен быть идентичен результату работы программ ping и tracert. (**Вариант 1**: аналог ping, функция "запрос эхо-повтора").*

Было разработано консольное приложение для Windows на языке C++, реализующее базовую функциональность утилиты `ping`. Программа использует "сырые"сокеты

(Raw Sockets) для ручного формирования и отправки ICMP-эхо-запросов, а также для приема и анализа ответных IP-пакетов. Для создания такого сокета и выполнения программы требуются права администратора.

Результаты работы программы

Ниже представлены скриншоты работы разработанного приложения при пинговании узлов `mit.edu` и `vk.com`, указанных в варианте задания.

```
PS C:\Users\Admin\Desktop\LabsS5\lab5> .\ping.exe mit.edu
Pinging mit.edu [23.194.226.2] with 32 bytes of data:
Reply from 23.194.226.2: bytes=32 time=190.702ms TTL=52
Reply from 23.194.226.2: bytes=32 time=188.145ms TTL=52
Reply from 23.194.226.2: bytes=32 time=192.462ms TTL=52
Reply from 23.194.226.2: bytes=32 time=187.62ms TTL=52

Ping statistics for 23.194.226.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 187ms, Maximum = 192ms, Average = 189ms
```

Рис. 21: Результат работы программы для `mit.edu`

```
PS C:\Users\Admin\Desktop\LabsS5\lab5> .\ping.exe vk.com
Pinging vk.com [87.240.132.78] with 32 bytes of data:
Reply from 87.240.132.78: bytes=32 time=53.9803ms TTL=52
Reply from 87.240.132.78: bytes=32 time=56.5492ms TTL=52
Reply from 87.240.132.78: bytes=32 time=54.364ms TTL=52
Reply from 87.240.132.78: bytes=32 time=51.6987ms TTL=52

Ping statistics for 87.240.132.78:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 51ms, Maximum = 56ms, Average = 54ms
```

Рис. 22: Результат работы программы для `vk.com`

Пояснение к результатам: Программа успешно выполняет все поставленные задачи:

- Разрешает доменное имя в IP-адрес.
- Отправляет 4 ICMP-эхо-запроса с интервалом в 1 секунду.
- Принимает эхо-ответы, декодирует их и выводит на экран IP-адрес отправителя, размер полученных данных, время отклика (RTT) и TTL.
- В конце выводит итоговую статистику: количество отправленных, полученных и потерянных пакетов, а также минимальное, максимальное и среднее время отклика.

Вывод

В ходе выполнения лабораторной работы были освоены практические навыки использования стандартных утилит командной строки (`ipconfig`, `arp`, `route`, `nslookup`, `ping`,

`tracert`) для диагностики и анализа функционирования IP-сетей. Была изучена на практике структура и назначение DNS-кэша, ARP-таблицы и таблицы маршрутизации, а также проанализированы реальные маршруты IP-пакетов в глобальной сети.

На втором этапе, в соответствии с вариантом задания, была успешно спроектирована и реализована собственная версия утилиты `ping` с небольшой частью функционала для Windows на языке C++.

Приложение

ping.cpp

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS

#include <winsock2.h>
#include <ws2tcpip.h>

#include <algorithm>
#include <chrono>
#include <iostream>
#include <numeric>
#include <string>
#include <vector>

#pragma comment(lib, "Ws2_32.lib")

// Директива #pragma pack заставляет компилятор располагать поля структуры
// вплотную друг к другу, без выравнивания
#pragma pack(push, 1)

// Структура, описывающая IP-заголовок (первые 20 байт IP-пакета)
// для парсинга ответа от удаленного хоста
struct IpHeader {
  unsigned char iph_ihl : 4;   // Длина заголовка в 32-битных словах (обычно 5)
  unsigned char iph_ver : 4;   // Версия IP (для IPv4 всегда 4)
  unsigned char iph_tos;       // Тип сервиса (редко используется)
  unsigned short iph_len;      // Общая длина пакета в байтах
  unsigned short iph_ident;    // Идентификатор (используется для сборки
фрагментов)
  unsigned short iph_flags;    // Флаги и смещение фрагмента
  unsigned char iph_ttl;       // Время жизни (Time To Live)
  unsigned char iph_protocol;  // Протокол транспортного уровня (для ICMP это 1)
  unsigned short iph_chksum;   // Контрольная сумма заголовка
  unsigned int iph_sourceip;   // IP-адрес отправителя
  unsigned int iph_destip;     // IP-адрес получателя
};

// Структура, описывающая заголовок ICMP-сообщения (8 байт)
// для создания нашего эхо-запроса
struct IcmpHeader {
```

```
  unsigned char icmp_type;     // Тип сообщения (8 для эхо-запроса, 0 для эхо-
ответа)
  unsigned char icmp_code;     // Код сообщения (для эхо-запроса/ответа всегда 0)
  unsigned short icmp_chksum;  // Контрольная сумма всего ICMP-пакета
  unsigned short icmp_id;      // Идентификатор, чтобы отличить ответы для нашей
                               // программы от других
  unsigned short icmp_seq;     // Порядковый номер пакета в сессии
};

#pragma pack(pop)

// Константы для ICMP-протокола
const int ICMP_ECHO_REQUEST = 8;
const int ICMP_ECHO_REPLY = 0;
const int ICMP_PACKET_SIZE = 32;  // Размер поля данных, которое мы будем
добавлять к ICMP-пакету

/**
 * @brief Вычисляет 16-битную контрольную сумму для ICMP-пакета по алгоритму RFC
 * 1071. Контрольная сумма - это 16-битное дополнение до единицы от суммы всех
 * 16-битных слов пакета.
 * @param buffer Указатель на буфер с пакетом (заголовок + данные).
 * @param size Размер буфера в байтах.
 * @return 16-битная контрольная сумма в сетевом порядке байт.
 */
unsigned short calculate_checksum(unsigned short *buffer, int size) {
  unsigned long cksum = 0;
  // Суммируем все 16-битные слова
  while (size > 1) {
    cksum += *buffer++;
    size -= sizeof(unsigned short);
  }
  // Если остался один байт, добавляем его
  if (size) {
    cksum += *(unsigned char *)buffer;
  }
  // Складываем старшую и младшую части суммы, пока старшая не обнулится
  cksum = (cksum >> 16) + (cksum & 0xffff);
  cksum += (cksum >> 16);
  // Инвертируем результат (дополнение до единицы)
  return (unsigned short)(~cksum);
}

/**
```

```
 * @brief Разбирает полученный от сети IP-пакет, извлекает из него ICMP-ответ и
 * выводит информацию на экран.
 * @param buffer Буфер с полученными данными (содержит полный IP-пакет).
 * @param bytes Количество полученных байт.
 * @param rtt Вычисленное время отклика в миллисекундах.
 * @param from_addr Структура с адресом отправителя.
 * @param expected_id Ожидаемый идентификатор ICMP-пакета, чтобы убедиться, что
 * ответ предназначен именно нам.
 * @return true, если получен корректный эхо-ответ, иначе false.
 */
bool decode_reply(char *buffer, int bytes, double rtt, sockaddr_in *from_addr,
                  unsigned short expected_id) {
  // Преобразуем начало буфера в указатель на структуру IP-заголовка
  IpHeader *ip_header = (IpHeader *)buffer;
  // Вычисляем длину IP-заголовка (значение в поле ihl * 4 байта)
  int ip_header_len = ip_header->iph_ihl * 4;

  // Проверяем, что полученный пакет достаточно большой, чтобы вместить IP и
  // ICMP заголовки
  if (bytes < ip_header_len + sizeof(IcmpHeader)) {
    std::cout << "Received packet is too small." << std::endl;
    return false;
  }

  // ICMP-заголовок начинается сразу после IP-заголовка
  IcmpHeader *icmp_header = (IcmpHeader *)(buffer + ip_header_len);

  // Проверяем, является ли это эхо-ответом (тип 0)
  if (icmp_header->icmp_type == ICMP_ECHO_REPLY) {
    // Проверяем, совпадает ли ID пакета с тем, что мы отправляли
    if (icmp_header->icmp_id == expected_id) {
      char ip_str[INET_ADDRSTRLEN];
      // Преобразуем IP-адрес отправителя из двоичного формата в строку
      inet_ntop(AF_INET, &(from_addr->sin_addr), ip_str, INET_ADDRSTRLEN);

      int data_size = bytes - ip_header_len - sizeof(IcmpHeader);

      std::cout << "Reply from " << ip_str << ": bytes=" << data_size << " time="
<< rtt << "ms"
                << " TTL=" << (int)ip_header->iph_ttl << std::endl;
      return true;
    }
  }
```

```
  return false;
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
    std::cerr << "Usage: " << argv[0] << " <hostname>" << std::endl;
    return 1;
  }

  WSADATA wsaData;
  if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
    std::cerr << "WSAStartup failed." << std::endl;
    return 1;
  }

  // Разрешение доменного имени в IP-адрес
  hostent *host = gethostbyname(argv[1]);
  if (host == nullptr) {
    std::cerr << "Could not resolve hostname: " << argv[1] << std::endl;
    WSACleanup();
    return 1;
  }

  // Заполняем структуру адреса назначения
  sockaddr_in dest_addr;
  dest_addr.sin_family = AF_INET;
  dest_addr.sin_addr.s_addr = *(u_long *)host->h_addr_list[0];
  dest_addr.sin_port = 0;  // Для ICMP порт не используется

  char dest_ip_str[INET_ADDRSTRLEN];
  inet_ntop(AF_INET, &dest_addr.sin_addr, dest_ip_str, INET_ADDRSTRLEN);

  std::cout << "Pinging " << argv[1] << " [" << dest_ip_str << "] with " <<
ICMP_PACKET_SIZE
            << " bytes of data:" << std::endl;

  // Создание "сырого" сокета (Raw Socket)
  // AF_INET - семейство адресов IPv4
  // SOCK_RAW - тип сокета, позволяющий работать с IP-пакетами напрямую
  // IPPROTO_ICMP - указываем, что мы будем работать с протоколом ICMP
  SOCKET sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
  if (sock == INVALID_SOCKET) {
    std::cerr << "Failed to create raw socket. Error: " << WSAGetLastError()
              << ". Try running as Administrator." << std::endl;
```

```
    WSACleanup();
    return 1;
  }

  // Устанавливаем таймаут на получение ответа
  int timeout = 1000;  // мс
  setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout));

  // Подготовка буфера для отправляемого ICMP-пакета
  char send_buf[sizeof(IcmpHeader) + ICMP_PACKET_SIZE];
  IcmpHeader *icmp_header = (IcmpHeader *)send_buf;

  icmp_header->icmp_type = ICMP_ECHO_REQUEST;                    // Тип - эхо-
запрос
  icmp_header->icmp_code = 0;                                    // Код - 0
  icmp_header->icmp_id = (unsigned short)GetCurrentProcessId();  // Используем ID
процесса как
                                                                 // уникальный
идентификатор

  // Заполняем поле данных произвольными символами
  memset(send_buf + sizeof(IcmpHeader), 'D', ICMP_PACKET_SIZE);

  // Переменные для сбора статистики
  int packets_sent = 0;
  int packets_received = 0;
  std::vector<double> rtt_times;

  // Отправляем 4 пакета и ждем ответы
  for (int i = 0; i < 4; ++i) {
    icmp_header->icmp_seq = i;     // Порядковый номер
    icmp_header->icmp_chksum = 0;  // Обнуляем контрольную сумму перед
вычислением новой
    icmp_header->icmp_chksum = calculate_checksum((unsigned short *)send_buf,
sizeof(send_buf));

    // Фиксируем время перед отправкой
    auto start_time = std::chrono::high_resolution_clock::now();

    // Отправляем пакет
    if (sendto(sock, send_buf, sizeof(send_buf), 0, (sockaddr *)&dest_addr,
sizeof(dest_addr)) ==
        SOCKET_ERROR) {
```

```
      std::cerr << "sendto failed with error: " << WSAGetLastError() <<
std::endl;
      break;
    }
    packets_sent++;

    // Буфер для приема ответа (размер больше, т.к. придет полный IP-пакет)
    char recv_buf[1024];
    sockaddr_in from_addr;
    int from_addr_len = sizeof(from_addr);

    // Блокирующая операция: ждем ответа (не дольше таймаута)
    int bytes_received =
        recvfrom(sock, recv_buf, sizeof(recv_buf), 0, (sockaddr *)&from_addr,
&from_addr_len);

    // Фиксируем время после получения ответа
    auto end_time = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double, std::milli> rtt = end_time - start_time;

    if (bytes_received == SOCKET_ERROR) {
      // Если recvfrom вернул ошибку, проверяем, не таймаут ли это
      if (WSAGetLastError() == WSAETIMEDOUT) {
        std::cout << "Request timed out." << std::endl;
      } else {
        std::cerr << "recvfrom failed with error: " << WSAGetLastError() <<
std::endl;
      }
    } else {
      // Если данные получены, декодируем ответ
      if (decode_reply(recv_buf, bytes_received, rtt.count(), &from_addr,
icmp_header->icmp_id)) {
        packets_received++;
        rtt_times.push_back(rtt.count());
      }
    }

    Sleep(1000);
  }

  std::cout << "\nPing statistics for " << dest_ip_str << ":" << std::endl;
  int packets_lost = packets_sent - packets_received;
  double loss_percent = (packets_sent > 0) ? ((double)packets_lost / packets_sent
* 100.0) : 0;
```

```
  std::cout << "    Packets: Sent = " << packets_sent << ", Received = " <<
packets_received
            << ", Lost = " << packets_lost << " (" << (int)loss_percent << "%
loss)," << std::endl;

  if (!rtt_times.empty()) {
    double min_rtt = *std::min_element(rtt_times.begin(), rtt_times.end());
    double max_rtt = *std::max_element(rtt_times.begin(), rtt_times.end());
    double avg_rtt = std::accumulate(rtt_times.begin(), rtt_times.end(), 0.0) /
rtt_times.size();

    std::cout << "Approximate round trip times in milli-seconds:" << std::endl;
    std::cout << "    Minimum = " << (int)min_rtt << "ms"
              << ", Maximum = " << (int)max_rtt << "ms"
              << ", Average = " << (int)avg_rtt << "ms" << std::endl;
  }

  closesocket(sock);
  WSACleanup();

  return 0;
}
```