Symulator Komendy Ping w Stylu JunOS

Adrian Cich, Adrian Prędki, Jakub Puch

17 stycznia 2025

Spis treści

1 Wprowadzenie

Celem projektu jest stworzenie wieloplatformowego skryptu symulującego działanie komendy `ping` znanej z systemu JunOS. Projekt zakłada implementację funkcji umożliwiających diagnostykę sieci przy pomocy protokołu ICMP, jednocześnie uwzględniając parametry wejściowe, takie jak liczba pakietów, rozmiar danych czy czas oczekiwania na odpowiedź. Dokumentacja ma charakter techniczny i może być używana do decyzji o wdrożeniu rozwiązania w środowisku klienta.

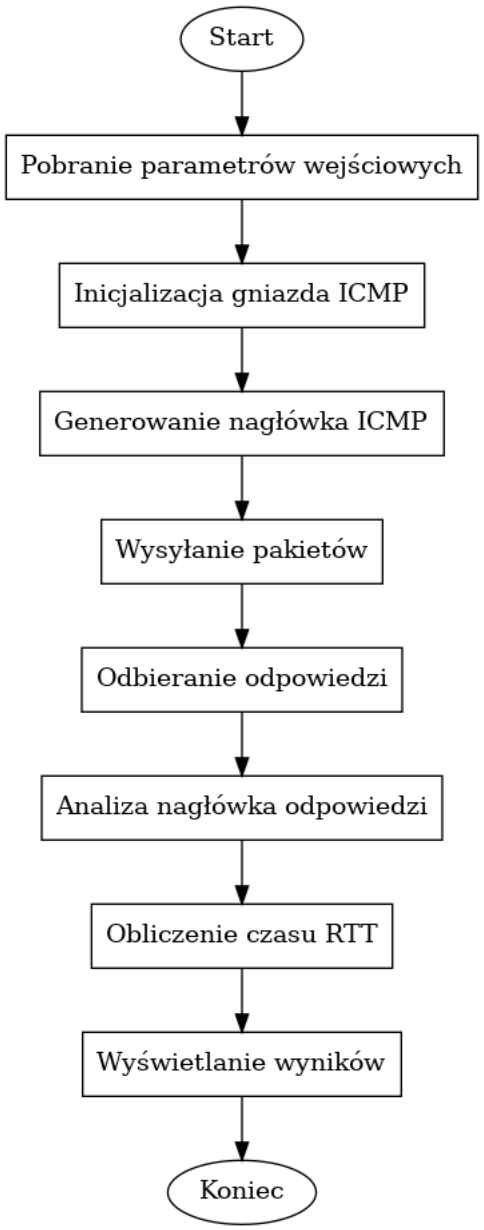
2 Opis projektu

2.1 Zakres funkcjonalności

- Wysyłanie pakietów ICMP z parametrami:
  - Liczba pakietów (domyślnie 5).
  - Rozmiar pakietu (domyślnie 64 B).
  - Interwał między pakietami (domyślnie 1 sekunda).
- Wyświetlanie statystyk:
  - Liczba wysłanych/odebranych pakietów.
  - RTT (min, avg, max).
  - Procent utraconych pakietów.
- Obsługa wyjątków:
  - Przerwanie przez użytkownika (`CTRL+C`).
  - Timeout dla odpowiedzi ICMP.
- Obsługa Linux z wykorzystaniem bibliotek systemowych.

3 Architektura skryptu

3.1 Diagram przepływu:



Rysunek 1: Diagram przepływu.

Diagram przepływu przedstawia proces działania skryptu "Ping"w stylu JunOS. Rozpoczyna się od pobrania parametrów wejściowych, takich jak liczba pakietów czy rozmiar danych. Następnie inicjalizowane jest gniazdo ICMP, po czym generowany jest nagłówek ICMP dla każdego pakietu. Pakiety są wysyłane do docelowego hosta, a skrypt oczekuje na odpowiedzi. Otrzymane dane są analizowane w celu wyliczenia czasu RTT i innych statystyk. Na końcu wyniki są wyświetlane, a skrypt kończy działanie. Diagram ilustruje ten przepływ krok po kroku.

3.2 Szczegóły implementacyjne

- **Srodowisko:** Linux
- **Język:** Python
- **Kompatybilne z:** Junos OS 11.1
- **Biblioteki:**
  - `socket`: Obsługa surowych gniazd.
  - `struct`: Tworzenie nagłówka ICMP.
  - `time`: Pomiar czasu RTT.
  - `select`: Obsługa timeoutów.

4 Porównanie z oficjalną dokumentacją `ping` w JunOS

- **Parametry wejściowe:** JunOS obsługuje dodatkowe flagi (np. DF - Don't Fragment). Implementacja Python umożliwia ustawienie liczby pakietów, rozmiaru i interwału.
- **Statystyki:** JunOS wyświetla bardziej szczegółowe dane, np. procent strat na różnych interfejsach. Implementacja Python ogranicza się do podstawowych statystyk RTT.
- **Rozszerzenia:** JunOS obsługuje tryb sweep. W implementacji Python można go dodać jako opcję.

5 Kluczowe elemenenty programu

Funkcja obliczająca sumę kontrolną

```
def checksum(source_string):
    """Calculate the checksum of a source string."""
    sum = 0
    count_to = (len(source_string) // 2) * 2
    ...
    return answer
```

Funkcja `checksum` oblicza sumę kontrolną ICMP, zapewniając integralność danych przesyłanych w pakietach.

Generowanie pakietu ICMP

```
def create_packet(seq_number, packet_size):
    """Create a new ICMP echo request packet."""
    header = struct.pack('!BBHHH', ICMP_ECHO_REQUEST, ICMP_CODE, 0, os.
        getpid() & 0xFFFF, seq_number)
    data = (packet_size - 8) * b'Q'  # Payload filled with 'Q'
    ...
    return header + data
```

Funkcja `create_packet` generuje pakiet ICMP zawierający nagłówek oraz dane. Nagłówek zawiera informacje takie jak typ pakietu i suma kontrolna.

Wysyłanie pakietów

```
def send_ping(sock, addr, seq_number, packet_size):
    """Send an ICMP packet."""
    packet = create_packet(seq_number, packet_size)
    sock.sendto(packet, (addr, 1))
```

Funkcja `send_ping` odpowiada za wysłanie pakietów ICMP przy użyciu gniazda surowego (`socket`).

Obsługa odpowiedzi ICMP

```
def receive_ping(sock, timeout):
    """Receive the ping response."""
    ready = select.select([sock], [], [], timeout)
    ...
    return time_received, seq, ttl
```

Funkcja `receive_ping` odbiera odpowiedzi ICMP, analizuje nagłówek i wyodrębnia kluczowe informacje (np. TTL, numer sekwencji).

Funkcja główna

Pełny skrypt został załączony w pliku do raportu. Poniżej zaprezentowano jego fragmenty z omówieniem.

```
def ping(host, count=5, packet_size=64, rapid=False, wait=1):
    """Ping a host."""
    addr = socket.gethostbyname(host)
    ...
    while seq_number < count:
        send_ping(sock, addr, seq_number, packet_size)
        ...
        print(f'{packet_size + 8} bytes from {addr}: icmp_seq={seq} ttl
            ={ttl} time={rtt:.3f} ms')
```

Funkcja `ping` zarządza całym procesem: od wyszukiwania adresu hosta, przez wysyłanie i odbieranie pakietów, aż po wyświetlenie wyników.

6 Kod źródłowy

W poniższej sekcji zaprezentowano kod źródłowy programu napisanego w języku Python, implementującego symulator komendy `ping` w stylu JunOS. Kod zawiera pełną funkcjonalność opisaną w poprzednich rozdziałach.

```
1 import socket
2 import struct
3 import time
4 import os
5 import sys
6 import platform
```

```
7  import select
8  import argparse
9
10 # ICMP Header Fields
11 ICMP_ECHO_REQUEST = 8
12 ICMP_CODE = 0
13
14 def checksum(source_string):
15     """Calculate the checksum of a source string."""
16     sum = 0
17     count_to = (len(source_string) // 2) * 2
18     count = 0
19
20     while count < count_to:
21         this_val = source_string[count + 1] * 256 + source_string[count
               ]
22         sum += this_val
23         sum = sum & 0xffffffff
24         count += 2
25
26     if count_to < len(source_string):
27         sum += source_string[-1]
28         sum = sum & 0xffffffff
29
30     sum = (sum >> 16) + (sum & 0xffff)
31     sum += (sum >> 16)
32     answer = ~sum
33     answer = answer & 0xffff
34     answer = answer >> 8 | (answer << 8 & 0xff00)
35     return answer
36
37 def create_packet(seq_number, packet_size):
38     """Create a new ICMP echo request packet."""
39     if packet_size < 64 or packet_size > 1500:
40         print("Rozmiar pakietu musi mie ci  si  w zakresie 64  1500
               bajt w.")
41         sys.exit(1)
42
43     header = struct.pack('!BBHHH', ICMP_ECHO_REQUEST, ICMP_CODE, 0, os.
           getpid() & 0xFFFF, seq_number)
44     data = (packet_size - 8) * b'Q'  # Payload filled with 'Q'
45     my_checksum = checksum(header + data)
46     header = struct.pack('!BBHHH', ICMP_ECHO_REQUEST, ICMP_CODE,
           my_checksum, os.getpid() & 0xFFFF, seq_number)
47     return header + data
48
49 def send_ping(sock, addr, seq_number, packet_size):
50     """Send an ICMP packet."""
51     packet = create_packet(seq_number, packet_size)
52     sock.sendto(packet, (addr, 1))
53
54 def receive_ping(sock, timeout):
55     """Receive the ping response."""
56     start_time = time.time()
57     while True:
58         ready = select.select([sock], [], [], timeout)
59         if ready[0] == []:  # Timeout
60             return None, None, None
```

```
61          time_received = time.time()
62          recv_packet, addr = sock.recvfrom(1024)
63          icmp_header = recv_packet[20:28]
64          icmp_type, icmp_code, _, _, seq = struct.unpack('!BBHHH',
                icmp_header)
65          ttl = recv_packet[8]
66          if icmp_type == 0 and icmp_code == 0:  # Echo reply
67              return time_received, seq, ttl
68
69  def ping(host, count=5, packet_size=64, rapid=False, wait=1):
70      """Ping a host."""
71      try:
72          addr = socket.gethostbyname(host)
73          print(f'PING {host} ({addr}): {packet_size} data bytes')
74      except socket.gaierror as e:
75          print(f'Ping request could not find host {host}. Please check
                the name and try again.')
76          return
77
78      try:
79          sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.
                IPPROTO_ICMP)
80      except PermissionError:
81          print("Root privileges are required to run this script.")
82          sys.exit(1)
83
84      rtts = []
85      sent_packets = 0
86      received_packets = 0
87      seq_number = 0
88
89      try:
90          while seq_number < count:
91              sent_packets += 1
92              send_time = time.time()
93              send_ping(sock, addr, seq_number, packet_size)
94              try:
95                  recv_time, seq, ttl = receive_ping(sock, timeout=1)
96                  if recv_time:
97                      received_packets += 1
98                      rtt = (recv_time - send_time) * 1000
99                      rtts.append(rtt)
100                     print(f'{packet_size + 8} bytes from {addr}:
                            icmp_seq={seq} ttl={ttl} time={rtt:.3f} ms')
101                 else:
102                     print(f'Request timeout for icmp_seq {seq_number}')
103             except Exception as e:
104                 print(f'Error: {e}')
105             seq_number += 1
106             if not rapid:
107                 time.sleep(wait)
108
109     except KeyboardInterrupt:
110         print("\n--- Ping interrupted by user ---")
111     finally:
112         sock.close()
113         # Print statistics
114         print(f'\n--- {host} ping statistics ---')
```

```
115         if sent_packets > 0:  # Avoid division by zero
116             print(f'{sent_packets} packets transmitted, {
                    received_packets} packets received, '
117                   f'{(sent_packets - received_packets) / sent_packets *
                        100:.1f}% packet loss')
118             if rtts:
119                 print(f'round-trip min/avg/max = {min(rtts):.3f}/{sum(
                        rtts)/len(rtts):.3f}/{max(rtts):.3f} ms')
120         else:
121             print("No packets were transmitted.")
122
123 if __name__ == "__main__":
124     if platform.system().lower() != 'windows' and os.geteuid() != 0:
125         print("This script requires root privileges to run on Linux.")
126         sys.exit(1)
127
128     parser = argparse.ArgumentParser(description="Ping a host in JunOS
            style")
129     parser.add_argument("host", help="Target host to ping (e.g.,
            192.168.1.3 or example.com)")
130     parser.add_argument("--size", type=int, default=64, help="Rozmiar
            pakietu w bajtach (domy lnie: 64, zakres: 64  1500  )")
131     parser.add_argument("--count", type=int, default=5, help="Number of
             packets to send (default: 5)")
132     parser.add_argument("--rapid", action="store_true", help="Send
            packets as fast as possible without waiting")
133     parser.add_argument("--wait", type=int, default=1, help="Wait time
            in seconds between packets (default: 1 second)")
134     args = parser.parse_args()
135
136     ping(args.host, count=args.count, packet_size=args.size, rapid=args
            .rapid, wait=args.wait)
```

Listing 1: Kod źródłowy programu Python

7 Adresacja hosta

Przed uruchomieniem skryptu należy odpowiednio skonfigurować adresację hosta. Ważne jest, aby interfejs sieciowy, za pomocą którego będą wysyłane pakiety ICMP, był poprawnie skonfigurowany z odpowiednim adresem IP (IPv4 lub IPv6).

W celu sprawdzenia dostępnych interfejsów sieciowych oraz ich bieżącej konfiguracji można użyć polecenia:

```
1 ifconfig
```

Komenda ta wyświetla listę aktywnych interfejsów oraz przypisane do nich adresy IP. Na podstawie tej informacji należy określić, który interfejs będzie wykorzystywany.

Jeżeli interfejs wymaga przypisania nowego adresu, można to zrobić za pomocą następującego polecenia:

```
1 sudo ip address add $ip_address dev $interface
```

Gdzie:

- `$ip_address` – docelowy adres IP, który chcemy przypisać do interfejsu (np. `10.0.0.2/24`).

- `$interface` – nazwa interfejsu sieciowego, np. `eth0` lub `wlan0`.

Przykład dodania adresu IPv4 do interfejsu `eth0`:

```
sudo ip address add 10.0.0.2/24 dev eth0
```

Adres `10.0.0.2/24` został przypisany do interfejsu `eth0` hosta testowego i wykorzystany w dalszych etapach dokumentacji, w rozdziale *Testy i wyniki działania połączeń do wybranych hostów*. Wszystkie testy zostały wykonane z użyciem tego adresu jako źródła zapytań.

Po wykonaniu tych kroków skrypt będzie mógł wysyłać pakiety ICMP przy użyciu odpowiedniego interfejsu i adresacji. Upewnienie się, że konfiguracja sieci jest poprawna, pozwoli uniknąć błędów związanych z komunikacją w sieci.

8 Implementacja i uruchamianie skryptu

Aby uruchomić skrypt `ping_simulator.py` na systemie Linux, należy wykonać następujące kroki:

8.1 Kopiowanie skryptu na system Linux

Skrypt można dodać do systemu Linux poprzez przesłanie pliku za pomocą dowolnego menedżera plików lub narzędzia, np. `scp` (Secure Copy Protocol). Przykład przesłania pliku z innego komputera:

```
scp ping_simulator.py user@linux_host:/home/user/ping/
```

W powyższym przykładzie:

- `ping_simulator.py` – nazwa pliku skryptu.
- `user` – nazwa użytkownika na zdalnym systemie Linux.
- `linux_host` – adres IP lub nazwa hosta systemu Linux.
- `/home/user/ping/` – lokalizacja, w której plik zostanie zapisany.

8.2 Ustawienie uprawnień do uruchamiania skryptu

Po przesłaniu pliku należy upewnić się, że skrypt posiada odpowiednie uprawnienia do wykonania. Można to zrobić za pomocą polecenia:

```
chmod +x ping_simulator.py
```

8.3 Uruchamianie skryptu

Skrypt wymaga uprawnień administratora (root), ponieważ korzysta z gniazd surowych (`raw sockets`). Aby go uruchomić, należy użyć polecenia `sudo`:

```
sudo ./ping_simulator.py <adres_hosta>
```

Przykład uruchomienia skryptu:

```
sudo ./ping_simulator.py 8.8.8.8
```

W powyższym przykładzie:

- `8.8.8.8` – adres hosta, który będzie pingowany.

8.4 Weryfikacja działania

Podczas uruchamiania skrypt wyświetli szczegółowe informacje o wysyłanych pakietach, odpowiedziach hosta, czasach RTT oraz statystykach końcowych.

9 Testy i wyniki działania polaczen do wybranych hostow

9.1 Nieosiągalny host LAN (10.0.0.5)

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 10.0.0.5 --count 3
PING 10.0.0.5 (10.0.0.5): 64 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 0 packets received, 100.0% packet loss
```

9.2 Osiagalny host LAN (10.0.0.3)

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 10.0.0.3 --count 3
PING 10.0.0.3 (10.0.0.3): 64 data bytes
72 bytes from 10.0.0.3: icmp_seq=0 ttl=64 time=0.239 ms
72 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.212 ms
72 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.111 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max = 0.111/0.187/0.239 ms
```

9.3 Local host (127.0.0.1)

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 127.0.0.1 --count 3
PING 127.0.0.1 (127.0.0.1): 64 data bytes
72 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.136 ms
72 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.285 ms
72 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.187 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max = 0.136/0.203/0.285 ms
```

9.4 Nieosiagalny do hosta poza LAN (10.255.255.1)

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 10.255.255.1 --count 3
PING 10.255.255.1 (10.255.255.1): 64 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

--- 10.255.255.1 ping statistics ---
3 packets transmitted, 0 packets received, 100.0% packet loss
```

9.5 Osiagalny do hosta poza LAN (8.8.8.8)

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 8.8.8.8 --count 3
PING 8.8.8.8 (8.8.8.8): 64 data bytes
72 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=29.095 ms
72 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=47.602 ms
72 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=19.364 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max = 19.364/32.020/47.602 ms
```

9.6 Osiagalny do DNS (www.google.com)

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py www.google.com
PING www.google.com (142.250.186.196): 64 data bytes
72 bytes from 142.250.186.196: icmp_seq=0 ttl=120 time=51.625 ms
72 bytes from 142.250.186.196: icmp_seq=1 ttl=120 time=50.086 ms
72 bytes from 142.250.186.196: icmp_seq=2 ttl=120 time=50.658 ms
72 bytes from 142.250.186.196: icmp_seq=3 ttl=120 time=50.782 ms
72 bytes from 142.250.186.196: icmp_seq=4 ttl=120 time=69.206 ms

--- www.google.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max = 50.086/54.472/69.206 ms
```

10 Dodatkowe parametry i ich uzycie

10.1 Parametr count

Parametr count określa liczbę pakietów ICMP, które mają zostać wysłane w ramach polecenia ping, pozwalając na kontrolowanie czasu trwania testu i ilości zebranych danych. Jest przydatny do ograniczenia liczby prób w celu uniknięcia niepotrzebnego obciążenia sieci lub uzyskania precyzyjnych statystyk w określonym zakresie. Przyklad uzycia ponizej:

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 8.8.8.8 --count 5
PING 8.8.8.8 (8.8.8.8): 64 data bytes
72 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=27.826 ms
72 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=36.065 ms
72 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=26.812 ms
72 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=51.583 ms
72 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=52.420 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max = 26.812/38.941/52.420 ms
```

10.2 Parametr rapid

Parametr `-rapid` umożliwia wysyłanie pakietów ICMP w trybie przyspieszonym, bez standardowego oczekiwania między kolejnymi próbami. Zamiast szczegółowego opisu każdego pakietu, wyniki są wyświetlane w formie horyzontalnej za pomocą znaków:
`!` oznacza sukces (otrzymano odpowiedź ICMP).
`.` oznacza brak odpowiedzi w czasie 500 ms (timeout).
Tryb rapid umożliwia szybką diagnozę połączeń w krótkim czasie, np. wysyłając wiele pakietów w ciągu kilku sekund, co symuluje zachowanie flood ping znane z systemów FreeBSD lub JunOS.

Przykład działania w przypadku sukcesu:

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 8.8.8.8 --rapid --count 10
PING 8.8.8.8 (8.8.8.8): 64 data bytes
!!!!!!!!!!

--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max = 33.912/34.195/35.034 ms
```

Przykład działania w przypadku braku odpowiedzi:

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 192.168.1.3 --rapid --count 10
PING 192.168.1.3 (192.168.1.3): 64 data bytes
..........

--- 192.168.1.3 ping statistics ---
10 packets transmitted, 0 packets received, 100.0% packet loss
```

10.3 Parametr wait

Parametr –wait określa czas w sekundach, jaki należy odczekać między wysyłanymi pakietami ICMP podczas działania skryptu ping. Domyślnie wynosi 1 sekundę, ale użytkownik może go dostosować, aby wydłużyć lub skrócić przerwy, co pozwala na precyzyjne dostosowanie częstotliwości testów do wymagań diagnostycznych sieci. Przyklad uzycia ponizej:

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 8.8.8.8 --count 5 --wait 3
PING 8.8.8.8 (8.8.8.8): 64 data bytes
72 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=26.275 ms
72 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=24.526 ms
72 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=21.071 ms
72 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=24.757 ms
72 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=20.241 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max = 20.241/23.374/26.275 ms
```

10.4 Parametr size

Parametr `-size` umożliwia określenie rozmiaru pakietu ICMP w bajtach, co pozwala na symulację różnych obciążeń sieci. Rozmiar pakietu może być ustawiony w zakresie od 64 do 1500 bajtów. Dzięki temu użytkownik może dostosować test diagnostyczny do konkretnych wymagań sieciowych.

Funkcja generująca pakiety ICMP uwzględnia rozmiar ustawiony za pomocą parametru `-size`. Wynik działania dla pakietu o rozmiarze 128 bajtów:

```
adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 8.8.8.8 --size 128
PING 8.8.8.8 (8.8.8.8): 128 data bytes
136 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=23.838 ms
136 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=26.573 ms
136 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=21.122 ms
136 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=22.102 ms
136 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=23.934 ms
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max = 21.122/23.514/26.573 ms
```

Rozmiar pakietu ICMP wynosi 136 bajtów, ponieważ do określonego w parametrze –size rozmiaru danych (128 bajtów) automatycznie dodawany jest 8-bajtowy nagłówek ICMP.

10.5 Obsługa przerwań i przykład działania

```
1 adrian@ubuntu:~/ping$ sudo ./ping_simulator.py 8.8.8.8
2 PING 8.8.8.8 (8.8.8.8): 64 data bytes
3 72 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=21.998 ms
4 72 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=24.482 ms
5 72 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=19.143 ms
```

```
72 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=23.739 ms
^C
--- Ping interrupted by user ---

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max = 19.143/22.340/24.482 ms
```

Powyższy przykład ilustruje, jak skrypt rejestruje wszystkie wysłane i odebrane pakiety, a po przerwaniu wyświetla statystyki zebrane do momentu zatrzymania działania. Funkcjonalność ta zwiększa użyteczność programu, szczególnie w przypadku długotrwałych testów.

11 Podsumowanie i wnioski

Skrypt spełnia wymagania projektu i symuluje działanie komendy `ping` w stylu JunOS. Projekt uwzględnia kluczowe funkcjonalności, takie jak wysyłanie pakietów ICMP z parametrami, zbieranie statystyk (RTT, liczba odebranych pakietów, procent strat) oraz obsługę wyjątków, w tym przerwanie przez użytkownika (`CTRL+C`). Testy przeprowadzone na różnych hostach wykazały poprawność działania skryptu w scenariuszach z hostami osiągalnymi i nieosiągalnymi w sieci LAN oraz poza nią.

Funkcjonalność skryptu obejmuje:

- Elastyczne parametry wejściowe (`-size`, `-count`, `-rapid`, `-wait`), które pozwalają na dostosowanie działania skryptu do różnych scenariuszy diagnostycznych.
- Obsługę wywołań w środowisku Linux z odpowiednimi uprawnieniami.
- Wyświetlanie szczegółowych wyników, takich jak czasy RTT, procent strat oraz statystyki liczby wysłanych i odebranych pakietów.
- Obsługę przerwań (`CTRL+C`), co zwiększa użyteczność narzędzia w praktycznych zastosowaniach.

Wnioski z projektu:

- Skrypt może być używany jako narzędzie diagnostyczne w sieciach IPv4 i IPv6, pod warunkiem poprawnej konfiguracji adresacji i środowiska testowego.
- Projekt wykazuje zbliżone działanie do funkcjonalności `ping` w JunOS, ale z pewnymi ograniczeniami, np. brakiem zaawansowanych statystyk oraz trybu sweep.
- Przygotowanie środowiska (np. konfiguracja adresacji) jest kluczowe dla poprawnego działania skryptu.
- Użycie surowych gniazd ICMP wymaga uprawnień administratora, co może ograniczać zastosowanie skryptu na systemach z restrykcyjnymi politykami bezpieczeństwa.

Dalsze możliwości rozwoju:

- Implementacja funkcji `Don't Fragment`, co pozwoli na badanie ograniczeń MTU w sieci.

- Dodanie trybu `sweep`, umożliwiającego dynamiczne zwiększanie rozmiaru pakietów w celu testowania stabilności sieci.
- Rozszerzenie kompatybilności o środowiska inne niż Linux (np. Windows, przy użyciu bibliotek takich jak `scapy`).
- Ulepszenie wyświetlania wyników i wprowadzenie bardziej zaawansowanych statystyk, np. rozkładu RTT czy wyników z różnych interfejsów.

Projekt stanowi solidną podstawę do dalszych usprawnień, a jego funkcjonalność i elastyczność czynią go użytecznym narzędziem w diagnostyce sieci.