Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA METROLOGII I ELEKTRONIKI

Technologie telekomunikacyjne
Projekt Ethernet

Autorzy: Krzysztof Cisło i Jakub Działowy
Kierunek studiów: Mikroelektronika w Technice i Medycynie

Kraków, 2021

Spis treści

1. ARP

ARP (Address Resolution Protocol), to protokół sieciowy umożliwiający mapowanie logicznych adresów warstwy sieciowej na fizyczne adresy warstwy łącza danych. Wykorzystywany jest np. w ramach większych systemów TCP/IP do weryfikacji adresów MAC przed wysłaniem dalszych wiadomości lub do identyfikacji potencjalnych konfliktów adresów IP, ponieważ różne węzły o identycznych adresach IP będą miały różne adresy MAC.

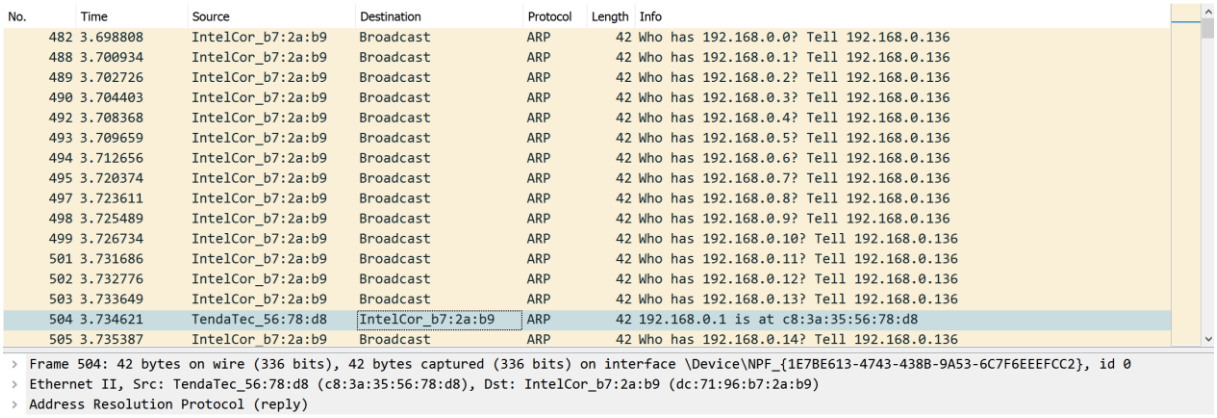
1.1. Ćwiczenie: ARP scanner

Celem ćwiczenia jest stworzenie prostego skanera ARP, który sprawdza dostępne adresy MAC dla zakresu adresów IP od 192.168.0.0 do 192.168.0.255. Program skanuje adresy IP w danej sieci poprzez wysłanie zapytań pod kolejne adresy z podanego zakresu, a gdy uzyska odpowiedź z adresem MAC, to wyświetla w konsoli IP i MAC urządzenia. Kod programu przedstawiono na listingu 1. Przy wywołaniu funkcji skorzystano z maski /24, która pozwala pokryć zadany zakres adresów z polecenia do zadania.

```
1   from scapy.all import srp,Ether,ARP,conf
2
3   def scan_arp(ip):
4       print("Scanning...")
5       conf.verb = 0 # verbose disable
6       arp_r = ARP(pdst=ip)
7       br = Ether(dst='ff:ff:ff:ff:ff:ff')
8       request = br/arp_r
9       answered, _ = srp(request, timeout=1)
10      print(f"IP \t\t\t MAC")
11      for i in answered:
12          ip, mac = i[1].psrc, i[1].hwsrc
13          print(ip, '\t\t', mac)
14
15
16  if __name__ == "__main__":
17      scan_arp("192.168.0.0/24")
```

Listing 1. Kod programu wyświetlającego IP i MAC urządzeń podłączonych do sieci.

Za pomocą programu Wireshark obserwowano wysyłane i odbierane pakiety protokołu ARP, widoczne na rysunkach 1. i 2. Zapytania są wysyłane pod kolejne adresy IP, a gdy dany adres jest dostępny, to zgodnie z oczekiwaniami odsyła adres MAC. Wydruk programu w konsoli przedstawiono na rysunku 3.



No.	Time	Source	Destination	Protocol	Length	Info
482	3.698808	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.0? Tell 192.168.0.136
488	3.700934	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.136
489	3.702726	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.2? Tell 192.168.0.136
490	3.704403	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.3? Tell 192.168.0.136
492	3.708368	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.4? Tell 192.168.0.136
493	3.709659	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.5? Tell 192.168.0.136
494	3.712656	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.6? Tell 192.168.0.136
495	3.720374	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.7? Tell 192.168.0.136
497	3.723611	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.8? Tell 192.168.0.136
498	3.725489	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.9? Tell 192.168.0.136
499	3.726734	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.10? Tell 192.168.0.136
501	3.731686	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.136
502	3.732776	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.12? Tell 192.168.0.136
503	3.733649	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.13? Tell 192.168.0.136
504	3.734621	TendaTec_56:78:d8	IntelCor_b7:2a:b9	ARP	42	192.168.0.1 is at c8:3a:35:56:78:d8
505	3.735387	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.14? Tell 192.168.0.136

```
> Frame 504: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{1E7BE613-4743-438B-9A53-6C7F6EEEFCC2}, id 0
> Ethernet II, Src: TendaTec_56:78:d8 (c8:3a:35:56:78:d8), Dst: IntelCor_b7:2a:b9 (dc:71:96:b7:2a:b9)
> Address Resolution Protocol (reply)
```

Rysunek 1. Początkowy fragment rozwiązania w Wireshark'u.

856	4.492812	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.253? Tell 192.168.0.136
857	4.493953	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.254? Tell 192.168.0.136
858	4.495821	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.255? Tell 192.168.0.136

Rysunek 2. Końcowy fragment rozwiązania w Wireshark'u.

```
Scanning...
IP                    MAC
192.168.0.1           c8:3a:35:56:78:d8
192.168.0.102         9e:de:d0:28:d5:f6
192.168.0.136         dc:71:96:b7:2a:b9
192.168.0.124         84:c5:a6:e7:20:4d
```

Rysunek 3. Rezultat uruchomionego programu ARP.

2. ICMP

Protokół ICMP (Internet Control Message Protocol) jest powszechnie używany do sprawdzania połączeń z innymi komputerami. Potocznie nazywa się „pingowaniem”, ponieważ działa w taki sam sposób, jak „ping” sonaru wysyłany przez okręty podwodne. Jeśli trafi w cel, powróci echo, które można wychwycić i zbadać.

2.1. Ćwiczenie: Ping program

Celem ćwiczenia jest program, który wysyła polecenie ping, czyli żądanie ICMP na adres IP. W przypadku otrzymania odpowiedzi wyświetla adres IP i powiązany adres MAC w konsoli. W programie przedstawionym na listingu 2. wykorzystano funkcję z wcześniejszego ćwiczenia do uzyskania adresu MAC poprzez protokół ARP. Dwa adresy IP są dostępne, natomiast jeden nie jest dostępny w celu sprawdzenia poprawności działania programu w obydwu przypadkach.

```python
import os
import sys

current = os.path.dirname(os.path.realpath(__file__))
parent = os.path.dirname(current)
sys.path.append(parent)

from icmplib import multiping
from ARP.arp_scanner import scan_arp

list_of_ip = ['192.168.0.206', '192.168.0.102', '192.168.0.136']
hosts = multiping(list_of_ip)

for host in hosts:
    if host.is_alive:
        scan_arp(host.address)
    else:
        print(f'No response from: {host.address}')
```

Listing 2. Kod programu wysyłającego ping i wyświetlającego IP oraz MAC po odebraniu odpowiedzi.

Jak widać na rysunku 4. od jednego z adresów IP nie uzyskano odpowiedzi. Natomiast dwa następne przesłały odpowiedź, a ich adresy IP i MAC zostały wypisane w konsoli.

```
No response from: 192.168.0.206
Scanning...
IP                      MAC
192.168.0.102           9e:de:d0:28:d5:f6
Scanning...
IP                      MAC
192.168.0.136           dc:71:96:b7:2a:b9
```

Rysunek 4. Rezultat uruchomionego ping programu.

Wykorzystana w programie funkcja multiping w pierwszej kolejności z wykorzystaniem protokołu ARP sprawdza czy dane adresy IP są dostępne, co widać na rysunku 5. Następnie wysyła ping do tych dostępnych (rysunek 6.).

No.	Time	Source	Destination	Protocol	Length	Info
1152	5.080541	IntelCor_b7:2a:b9	Broadcast	ARP	42	Who has 192.168.0.206? Tell 192.168.0.136
1962	9.580400	IntelCor_b7:2a:b9	9e:de:d0:28:d5:f6	ARP	42	Who has 192.168.0.102? Tell 192.168.0.136
1966	9.595137	9e:de:d0:28:d5:f6	IntelCor_b7:2a:b9	ARP	42	192.168.0.102 is at 9e:de:d0:28:d5:f6
2104	10.195665	TendaTec_56:78:d8	IntelCor_b7:2a:b9	ARP	42	Who has 192.168.0.136? Tell 192.168.0.1
2105	10.195762	IntelCor_b7:2a:b9	TendaTec_56:78:d8	ARP	42	192.168.0.136 is at dc:71:96:b7:2a:b9

Rysunek 5. Wynik w Wireshark'u po sprawdzeniu przez metodę multiping czy dany adres IP odpowiada.

No.	Time	Source	Destination	Protocol	Length	Info
972	5.039448	192.168.0.136	192.168.0.102	ICMP	98	Echo (ping) request id=0x6daa, seq=0/0, ttl=64 (reply in 980)
980	5.052036	192.168.0.102	192.168.0.136	ICMP	98	Echo (ping) reply id=0x6daa, seq=0/0, ttl=64 (request in 972)

Rysunek 6. Wynik wysłania i odebrania ping w Wireshark'u.

3. UDP

UDP (User Datagram Protocol), to metoda wysyłania danych bezpośrednio do określonego socketu. UDP może być używany jako protokół komunikacji bezpośredniej, w którym wysyła się wiadomości do określonej aplikacji w określonym systemie. W UDP wysyła się sygnał do konkretnego portu i jeśli wszystko poszło dobrze, a aplikacja monitorująca port jest skonfigurowana tak, aby odpowiadać na przychodzące wiadomości, to port wygeneruje odpowiedź, która zostanie otrzymana, a w przeciwnym wypadku nic nie zostanie odebrane. Dzięki temu UDP jest przydatny w sytuacji wysyłania niestandardowych komunikatów danych bezpośrednio z jednego systemu do drugiego.

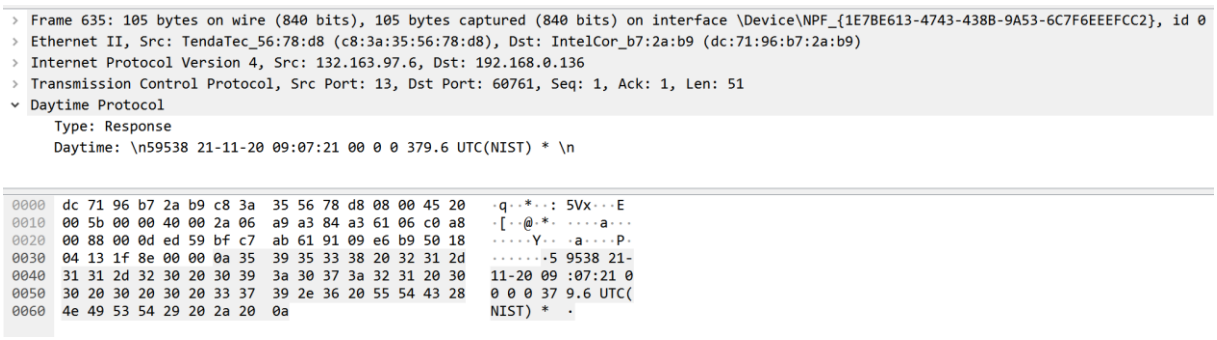
3.1. Ćwiczenie: Odebranie aktualnej daty i czasu

Celem ćwiczenia jest pobranie czasu i daty z socketu 13. na docelowym komputerze. Socket 13 odpowie na komunikat UDP, zwracając godzinę i datę w postaci ciągu ASCII. Treść programu przedstawiono na listingu 3.

```python
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = "time.nist.gov"
port = 13
s.connect((host, port))
while True:
    data = s.recv(1024)
    if data:
        print(data.decode())
    else:
        break

s.close()
```

Listing 3. Kod programu do odbierania daty i godziny z portu 13.

Wyniki działania programu przedstawiono na rysunkach 7. i 8. Odebrana wiadomość z socketu 13. jest ciągiem znaków zawierających datę i czas. Ten ciąg znaków wypisano w konsoli (rysunek 8).



Rysunek 7. Wyniki działania programu w Wireshark'u.

```
59538 21-11-20 10:41:29 00 0 0 608.6 UTC(NIST) *
```

Rysunek 8. Rezultat uruchomionego programu UDP widoczny w terminalu.

4. TCP

TCP (Transmission Control Protocol), to protokół komunikacyjny, stosowany do przesyłania danych między procesami uruchomionymi na różnych maszynach w trybie klient-serwer. Serwer oczekuje na nawiązanie połączenia na określonym porcie, a klient inicjuje połączenie do serwera. TCP gwarantuje dostarczenie wszystkich pakietów w całości, z zachowaniem kolejności i bez duplikatów.

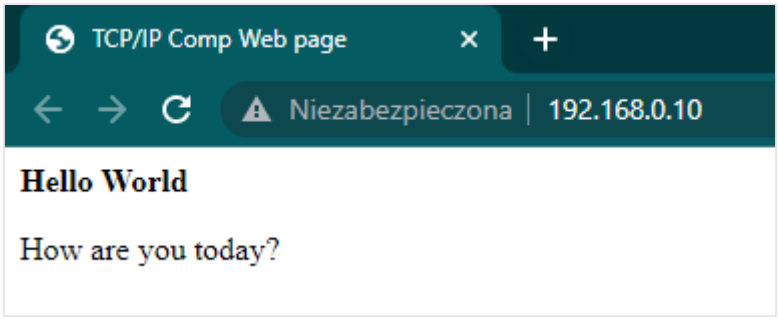
4.1. Ćwiczenie: Wysłanie strony HTML z wykorzystaniem HTTP

Celem ćwiczenia jest stworzenie strony internetowej i wyświetlenie jej w przeglądarce. Kod programu przedstawiono na listingu 4.

```
import socket

HOST = '192.168.0.10'
PORT = 80
print(f'The Web server URL for this would be http://{HOST}:{PORT}/')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.send(b'HTTP/1.0 200 OK\n')
            conn.send(b'Content-Type: text/html\n')
            conn.send(b'\n')
            conn.send(b"""<html>
                      <head>
                      <title>TCP/IP Comp Web page</title>
                      </head>
                      <body>
                      <b>Hello World</b>
                      <p>How are you today?</p>
                      </body>
                      </html>""")
            break
```

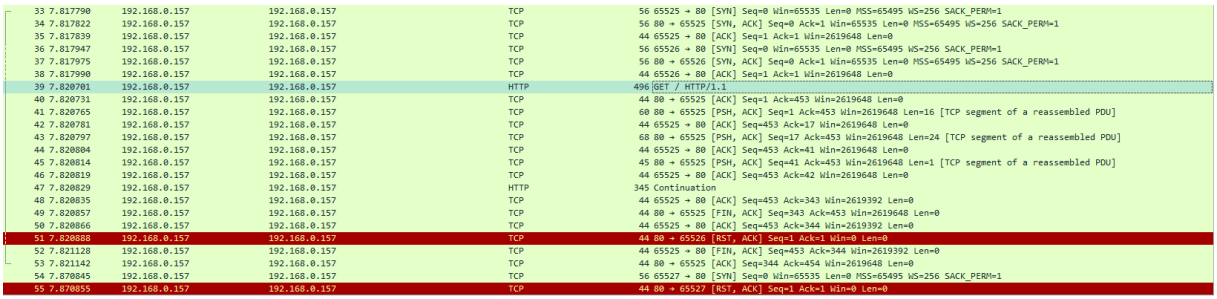
Listing 4. Kod programu wysyłający stronę HTML poprzez HTTP.

Działanie programu sprawdzono przy użyciu przeglądarki internetowej Google Chrome. Wpisano odpowiedni adres lokalny, w tym przypadku: 192.168.0.10, a następnie przeglądarka załadowała stronę, widoczną na rysunku 9.



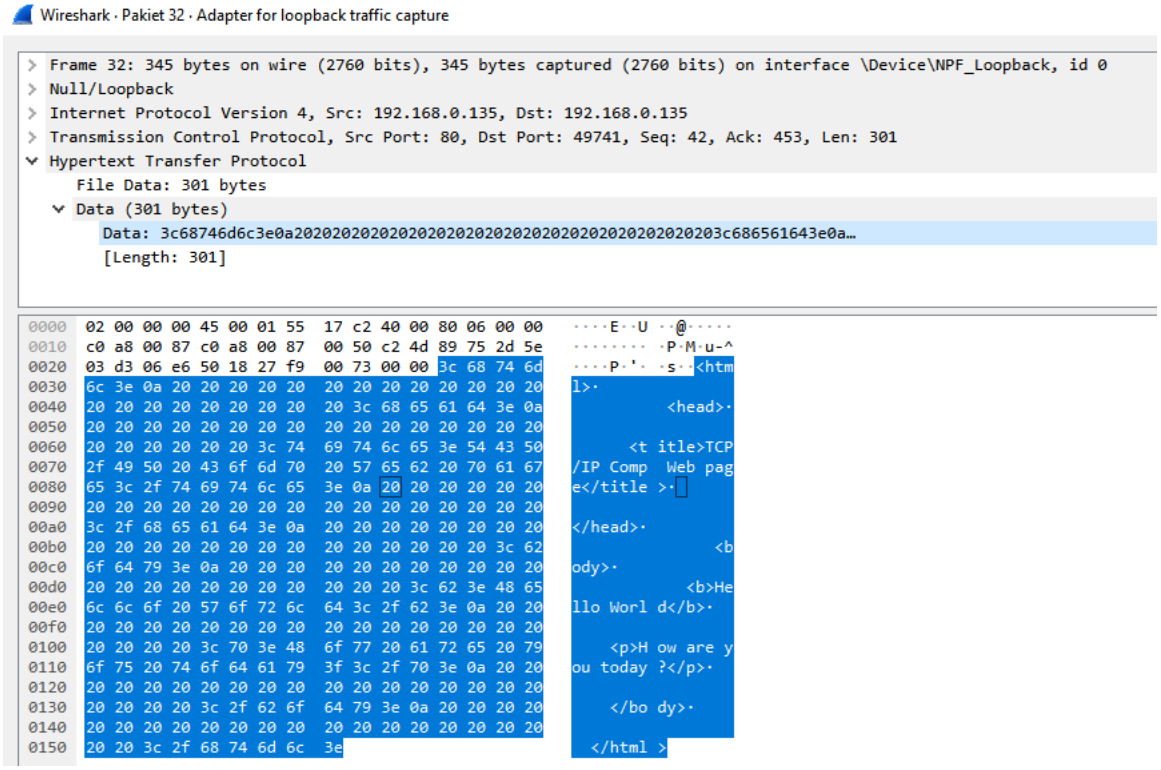
Rysunek 9. Strona załadowana przez przeglądarkę.

Ponadto, przechwycono pakiety przy użyciu programu Wireshark, widoczne na rysunku 10.



Rysunek 10. Przechwycone pakiety protokołu HTTP.

Na rysunku 11. przedstawiono pakiet zawierający dane strony HTML.



Rysunek 11. Pakiet zawierający dane strony HTML.

4.2. Ćwiczenie: Odebranie strony HTML

Celem ćwiczenia jest odebranie strony internetowej i wyświetlenie jej w konsoli. Kod programu przedstawiono na listingu 5.

```python
import socket
from bs4 import BeautifulSoup as bs


HOST = '192.168.0.10'  # The server's hostname or IP address
PORT = 80              # The port used by the server

html_content = []
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'GET /INDEX.HTM HTTP 1.0')
    while True:
        data = s.recv(1024)
        if not data:
            break
        html_content.append(data.decode())

html_string = ' '.join(html_content)

bs_html = bs(html_string, 'html.parser')
print(bs_html.body.get_text())
```

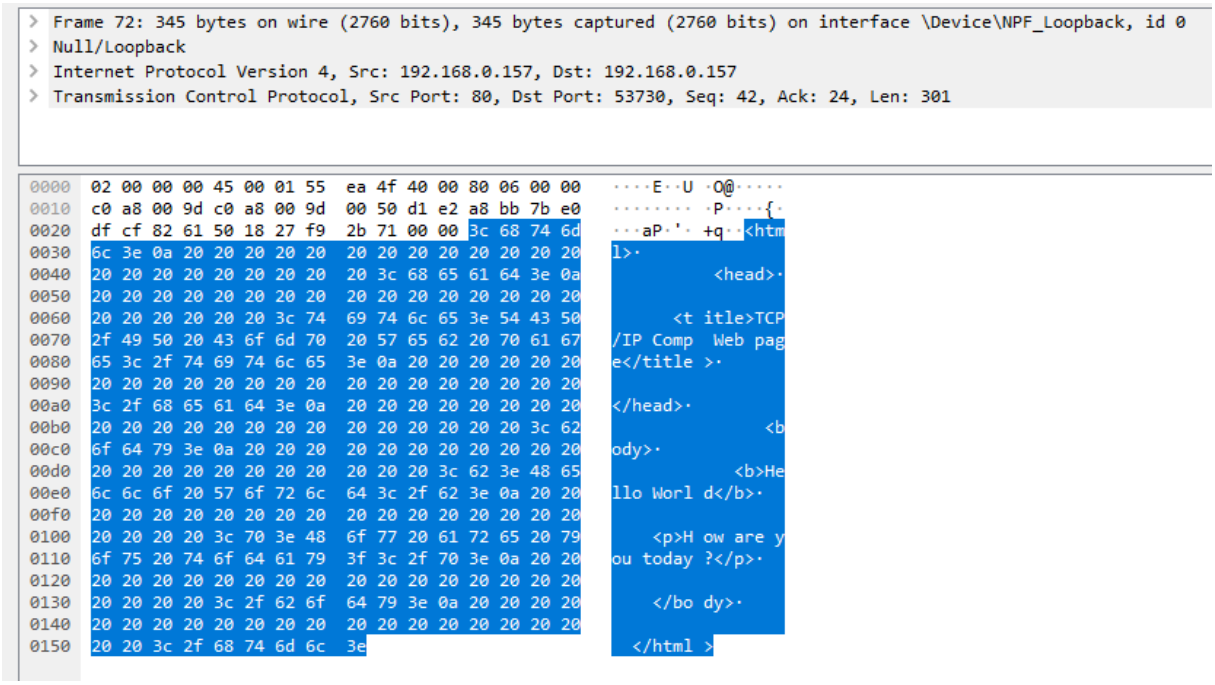
Listing 5. Kod programu odbierający stronę HTML poprzez HTTP.

Działanie programu sprawdzono przy użyciu dwóch terminali. Jeden służył jako serwer i czekał na zapytanie. Natomiast drugi działał jako klient (analogicznie do przeglądarki internetowej). Klient po odebraniu danych wypisał zawartość przesłanej strony, co przedstawiono na rysunku 12.

```
PS C:\Users\krzys\Dropbox\07_term\TT-Technologie-telekomunikacyjne\ProjLabs\TT_ethernet> & C:/Users/krzys/AppData/Local/Programs/Python/Python310/python.exe c:/Users/krzys/Dropbox/07_term/TT-Technologie-telekomunikacyjne/ProjLabs/TT_ethernet/TCP/tcp_client.py

Hello World
How are you today?
```

Rysunek 12. Odebrana strona HTML wyświetlona w konsoli.

Podczas tego testu przechwycono dane przy użyciu programu Wireshark (Rysunek 13).



Rysunek 13. Przechwycone dane strony HTML podczas komunikacji serwera i klienta.

5. SMTP

SMTP (Simple Mail Transfer Protocol), to protokół komunikacyjny opisujący sposób przekazywania poczty elektronicznej w internecie. Najczęściej wykorzystujący port 25. SMTP wymaga sekwencji określonych kroków z odpowiednimi odpowiedziami na różnych etapach.

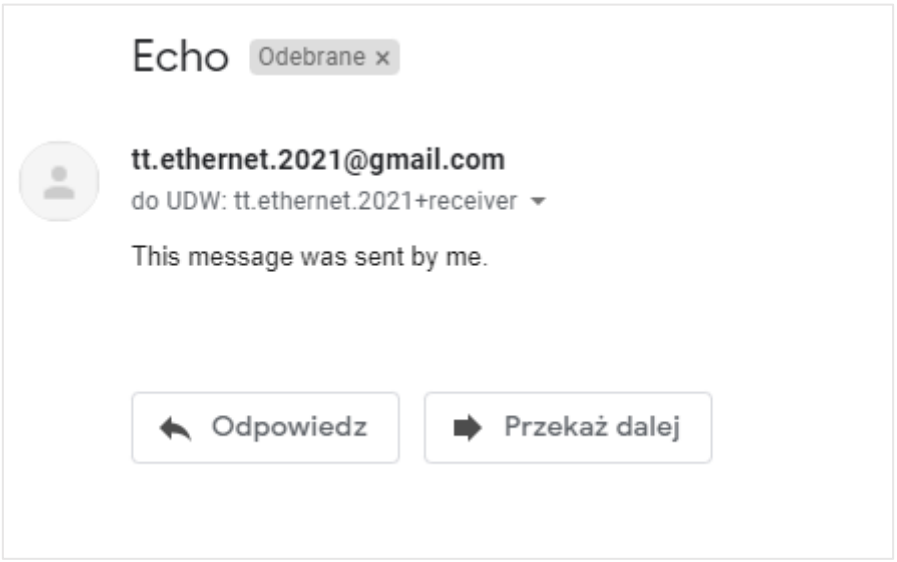
5.1. Ćwiczenie: Wysłanie maila

Celem ćwiczenia było wysłanie prostego maila przy użyciu SMTP. Kod programu przedstawiono na listingu 6. Wykorzystano bibliotekę smtplib oraz szyfrowanie SSL. Na potrzeby tego ćwiczenia utworzono również skrzynkę mailową Gmail.

```
1   import smtplib, ssl
2
3   port = 465  # For SSL
4   smtp_server = "smtp.gmail.com"
5   sender_email = "tt.ethernet.2021@gmail.com"
6   receiver_email = "tt.ethernet.2021+receiver@gmail.com"
7   password = "***"
8   message = """\
9   Subject: Echo
10
11  This message was sent by me."""
12
13  context = ssl.create_default_context()
14  with smtplib.SMTP_SSL(smtp_server, port, context=context) as server:
15      server.set_debuglevel(True)
16      server.login(sender_email, password)
17      server.sendmail(sender_email, receiver_email, message)
```

Listing 6. Kod programu wysyłającego maila poprzez SMTP.

Działanie programu sprawdzono wprost w skrzynce mailowej. Po włączeniu programu otrzymano maila widocznego na rysunku 14.



Rysunek 14. Otrzymany testowy mail.

Przechwycenie danych w Wireshark'u było niemożliwe ze względu na użycie serwera smtp@gmail.com, dlatego wykorzystano tryb „debug" obiektu SMTP_SSL. Zaobserwowane dane przedstawiono na rysunku 15. Wyraźnie widać interaktywny proces i charakterystyczne: „mail FROM", „rcpt TO", „data" oraz „QUIT".

```
PS C:\Users\krzys\Dropbox\07_term\TT-Technologie-telekomunikacyjne\ProjLabs\TT_ethernet> & C:/Users/krzys/AppData/Local/Pro
grams/Python/Python310/python.exe c:/Users/krzys/Dropbox/07_term/TT-Technologie-telekomunikacyjne/ProjLabs/TT_ethernet/SMTP
/send_email.py
send: 'ehlo DESKTOP-VMPK2G3.home\r\n'
reply: b'250-smtp.gmail.com at your service, [2a02:a315:c341:5780:a8b3:439:ebcf:71c6]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [2a02:a315:c341:5780:a8b3:439:ebcf:71c6]\nSIZE 35882577\n8BITM
IME\nAUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH\nENHANCEDSTATUSCODES\nPIPELINING\nCHUNKING\nSMTPUTF8'
send: 'AUTH PLAIN AHR0LmV0aGVybmV0LjIwMjFAZ21haWwuY29tAENpc2xvRHppYWxvd3k=\r\n'
reply: b'235 2.7.0 Accepted\r\n'
reply: retcode (235); Msg: b'2.7.0 Accepted'
send: 'mail FROM:<tt.ethernet.2021@gmail.com> size=45\r\n'
reply: b'250 2.1.0 OK 12sm323842ljq.59 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK 12sm323842ljq.59 - gsmtp'
send: 'rcpt TO:<tt.ethernet.2021+receiver@gmail.com>\r\n'
reply: b'250 2.1.5 OK 12sm323842ljq.59 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK 12sm323842ljq.59 - gsmtp'
send: 'data\r\n'
reply: b'354  Go ahead 12sm323842ljq.59 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead 12sm323842ljq.59 - gsmtp'
data: (354, b'Go ahead 12sm323842ljq.59 - gsmtp')
send: b'Subject: Echo\r\n\r\nThis message was sent by me.\r\n.\r\n'
reply: b'250 2.0.0 OK  1639145858 12sm323842ljq.59 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.0.0 OK  1639145858 12sm323842ljq.59 - gsmtp'
data: (250, b'2.0.0 OK  1639145858 12sm323842ljq.59 - gsmtp')
send: 'QUIT\r\n'
reply: b'221 2.0.0 closing connection 12sm323842ljq.59 - gsmtp\r\n'
reply: retcode (221); Msg: b'2.0.0 closing connection 12sm323842ljq.59 - gsmtp'
PS C:\Users\krzys\Dropbox\07_term\TT-Technologie-telekomunikacyjne\ProjLabs\TT_ethernet>
```

Rysunek 15. Wysyłane i odbierane dane protokołu SMTP.

5.2. Ćwiczenie: Wysłanie maila z zawartością HTML

Celem ćwiczenia było wysłanie maila z zawartością HTML z wykorzystaniem SMTP. Rezultatem miało być odebranie maila z działającym linkiem do strony internetowej. Kod programu przedstawiono na listingach 7 i 8. Program zapewnia odebranie treści maila w formie tekstu, gdyby użytkownik posiadał konto email nieobsługujące HTMLa.

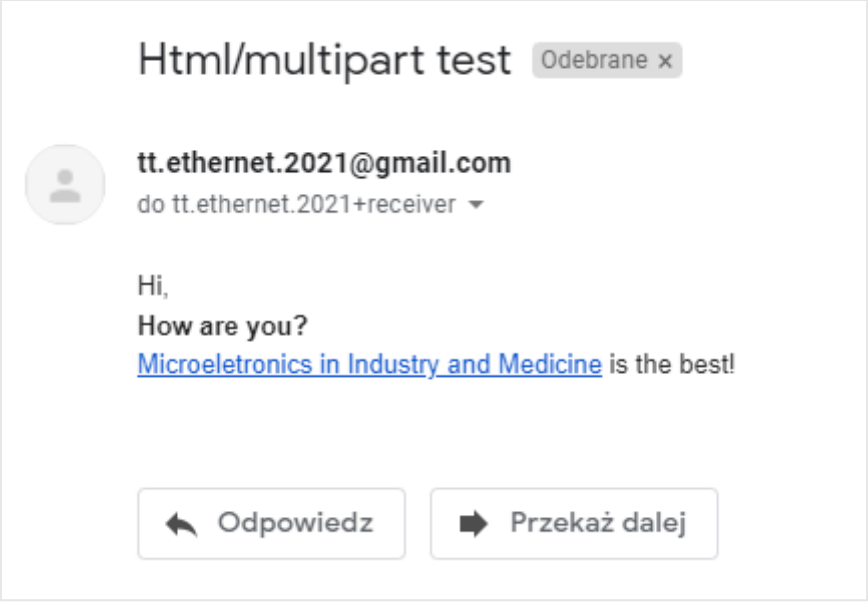
```
import smtplib, ssl
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

sender_email = "tt.ethernet.2021@gmail.com"
receiver_email = "tt.ethernet.2021+receiver@gmail.com"
password = "CisloDzialowy"

message = MIMEMultipart("alternative")
message["Subject"] = "Html/multipart test"
message["From"] = sender_email
message["To"] = receiver_email

# Create the plain-text and HTML version
text = """\
Hi,
How are you?
Microeletronics in Industry and Medicine is the best!
http://www.mtm.agh.edu.pl/"""
html = """\
<html>
  <body>
    <p>Hi,<br>
      <b>How are you?</b><br>
      <a href="http://www.mtm.agh.edu.pl/">Microeletronics in Industry and Medicine</a>
      is the best!
    </p>
  </body>
</html>
"""
```

Listing 7. Część pierwsza kodu programu wysyłającego maila z zawartością HTML.

```
32  # Turn these into plain/html MIMEText objects
33  part1 = MIMEText(text, "plain")
34  part2 = MIMEText(html, "html")
35
36  # Add HTML/plain-text parts to MIMEMultipart message
37  # The email client will try to render the last part first
38  message.attach(part1)
39  message.attach(part2)
40
41  # Create secure connection with server and send email
42  context = ssl.create_default_context()
43  with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
44      server.login(sender_email, password)
45      server.sendmail(
46          sender_email, receiver_email, message.as_string()
47      )
```

Listing 8. Część druga kodu programu wysyłającego maila z zawartością HTML.

Wynik działania programu przedstawiono na rysunku 16. Odbiorca po kliknięciu w link zostanie przekierowany na odpowiednią stronę internetową.



Rysunek 16. Odebrany testowy mail z zawartością HTML.