Notas de Aula

Prof. Luiz Antonio Rodrigues

Vol 1

3.3 Prática 4: UDP

O protocolo UDP [8] oferece um modo simples de enviar mensagens entre dispositivos de uma rede IP. Diferente do TCP, o UDP não estabelece uma conexão antes de enviar dados e não garante que os pacotes cheguem na ordem correta. Ele é não confiável, ou seja, não há garantias de entrega ou confirmação de recebimento, apenas detecção de erros com *checksum*. Se um pacote for perdido ou danificado, o UDP não tentará retransmiti-lo. Pacotes recebidos com erro são simplesmente descartadas.

O código a seguir exemplifica o servidor em Python.

- Cria um *socket* UDP com `socket.AF_INET` (IPv4) e `socket.SOCK_DGRAM` (UDP). Para IPv6, use `socket.AF_INET6`.
- O *socket* é associado a um endereço IP e a uma porta específica usando `bind()`.
- O servidor fica ouvindo por mensagens com `recvfrom()`, que recebe os dados e o endereço do cliente.
- Opcionalmente, o servidor pode enviar uma resposta ao cliente usando `sendto()`.

O código-fonte está disponível aqui.

python - Servidor UDP

```python
import socket

# Criação do socket UDP
udp_server_socket = socket.socket(socket.AF_INET,
                    socket.SOCK_DGRAM)

# Ligação do socket ao endereço e porta
# IPv4='localhost' ou '127.0.0.1', IPv6='::1'
server_address = ('localhost', 12345)

udp_server_socket.bind(server_address)

print("Servidor UDP aguardando mensagens...")

# Recebe dados do cliente
while True:
    # Buffer de 1024 bytes
    data, address = udp_server_socket.recvfrom(1024)
```

```
    print(f"Mensagem recebida: {data.decode()} de {address}")

    # Enviar uma resposta opcional para o cliente
    response = "Mensagem recebida com sucesso"
    udp_server_socket.sendto(response.encode(), address)
    print(f"Resposta enviada para {address}")
```

Para implementar o cliente, o código as seguir pode ser utilizado.

- Cria um socket UDP e envia uma mensagem ao servidor usando `sendto()`.
- Após o envio, o cliente pode receber uma resposta do servidor usando `recvfrom()`.

python - Cliente UDP

```python
import socket

# Criação do socket UDP
udp_client_socket = socket.socket(socket.AF_INET,
                    socket.SOCK_DGRAM)

# Endereço do servidor e porta
# No mesmo host, IPv4='localhost' ou '127.0.0.1', IPv6='::1'
dest_ip = '127.0.0.1' # Altere para o IP de destino
dest_port = 12345 # Porta de destino

server_address = (dest_ip, dest_port)

# Mensagem a ser enviada ao servidor
message = "Ola, servidor UDP!"
udp_client_socket.sendto(message.encode(), server_address)

# Recebe resposta do servidor
data, server = udp_client_socket.recvfrom(1024)
print(f"Resposta do servidor: {data.decode()}")

# Fechando o socket do cliente
udp_client_socket.close()
```

Como executar:

1. Primeiro, execute o código do servidor.

2. Em seguida, execute o código do cliente para enviar a mensagem.

Para verificar o funcionamento da aplicação com o Wireshark:

1. Iniciei a captura de pacotes no Wireshark;
2. Filtre os pacotes por `udp && udp.port==12345` e escolha o primeiro pacote capturado;
3. Analise os campos do pacote: porta de origem, porta de destino, tamanho do pacote e checksum, como na Figura 3.4.

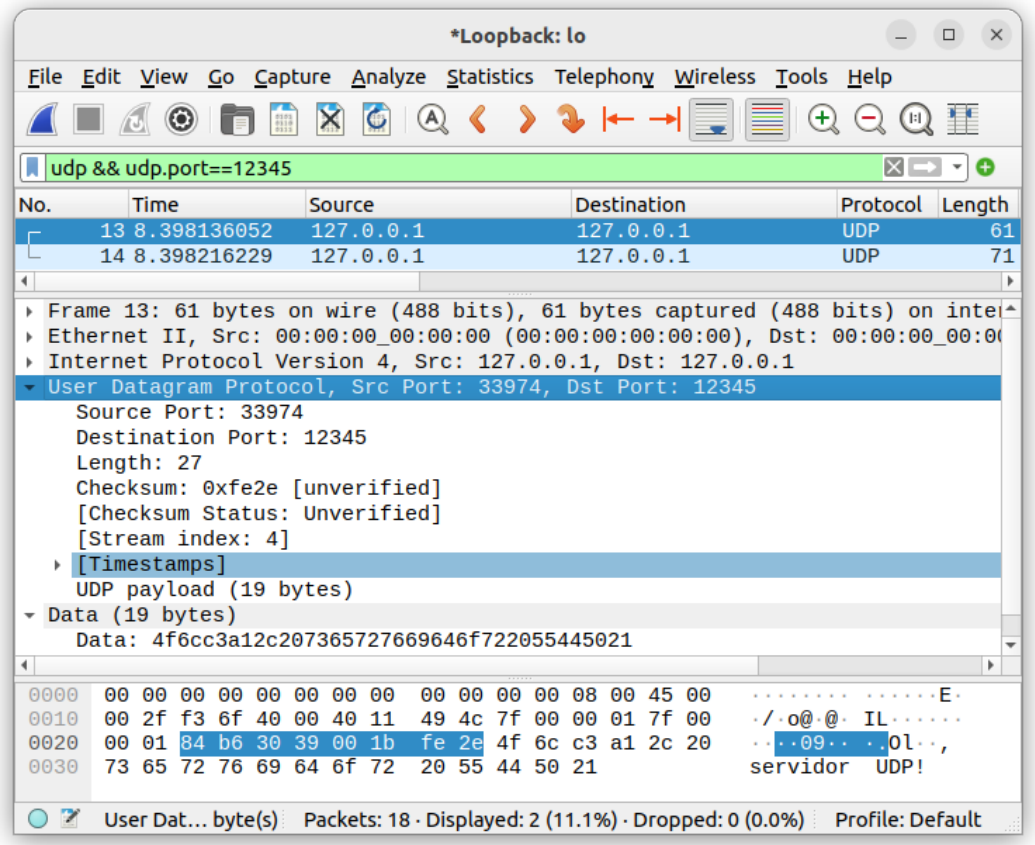

Figura 3.4: Pacotes UDP capturados pelo Wireshark.

3.3.1 UDP com raw sockets

O *raw socket* permite o controle sobre o conteúdo de todo o pacote, incluindo os cabeçalhos IP e UDP. Para enviar um pacote UDP usando r*aw sockets* pode-se usar

a seguinte abordagem. Aqui está o exemplo modificado onde só o cabeçalho UDP é construído manualmente, e o cabeçalho IP é gerado automaticamente pelo sistema operacional.

- O socket é criado com `socket.AF_INET` para IPv4 e `socket.SOCK_RAW`, usando o protocolo `IPPROTO_UDP`. Isso significa que o kernel ainda cuidará da construção do cabeçalho IP, enquanto nós construímos apenas o cabeçalho UDP.
- O cabeçalho UDP é construído manualmente, incluindo as portas de origem e destino, o comprimento total (cabeçalho UDP + dados), e o *checksum* (calculado a partir de um pseudo-cabeçalho).
- O pseudo-cabeçalho é necessário para calcular o *checksum* do UDP e inclui os endereços IP de origem e destino, o protocolo UDP, e o comprimento do datagrama UDP.
- A mensagem "Olá, servidor UDP!" é convertida em bytes e adicionada ao pacote.
- O pacote contendo o cabeçalho UDP e os dados é enviado diretamente para o destino. O kernel se encarrega de gerar o cabeçalho IP automaticamente.

Privilégios: Você precisará executar este *script* como administrador/root.

python - Cliente UDP raw socket

```python

import socket
import struct

def checksum(msg):
    s = 0
    # Somar as palavras de 16 bits
    for i in range(0, len(msg), 2):
        w = (msg[i] << 8) + (msg[i+1])
        s = s + w
    s = (s >> 16) + (s & 0xffff)
    s = s + (s >> 16)
    return ~s & 0xffff

# Endereço IP e porta do destino
dest_ip = '127.0.0.1'    # Altere para o IP de destino
source_ip = '127.0.0.1'  # IP de origem

```

```
# Criar socket UDP usando IPPROTO_UDP (apenas cabeçalho UDP)
raw_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW,
             socket.IPPROTO_UDP)

# Construção do cabeçalho UDP
source_port = 1234  # Porta de origem
dest_port = 12345   # Porta de destino
data = b"Ola, servidor UDP!"     # Dados a serem enviados
udp_len = 8 + len(data)  # Comprimento do cabeçalho UDP + dados
udp_check = 0  # Inicialmente, checksum é 0

# Cabeçalho UDP sem checksum
udp_header = struct.pack('!HHHH', source_port, dest_port,
                          udp_len, udp_check)

# Pseudo-cabeçalho para calcular o checksum UDP
pseudo_header = struct.pack('!4s4sBBH',
                socket.inet_aton(source_ip),
                socket.inet_aton(dest_ip), 0,
                socket.IPPROTO_UDP, udp_len)
pseudo_packet = pseudo_header + udp_header + data

# Calculando o checksum UDP
udp_check = checksum(pseudo_packet)

# Atualizando o cabeçalho UDP com o checksum correto
udp_header = struct.pack('!HHHH', source_port, dest_port,
                         udp_len, udp_check)

# Pacote final: cabeçalho UDP + dados
packet = udp_header + data

# Enviando o pacote
raw_socket.sendto(packet, (dest_ip, dest_port))
print(f"Pacote UDP enviado para {dest_ip}")

# Fechando o socket do cliente
raw_socket.close()
```

UDP Multicast

O UDP multicast é usado para transmitir mensagens em uma rede local para um grupo de dispositivos.

Os endereços multicast são específicos. No IPv4, pode-se usar qualquer endereço no intervalo de 224.0.0.0 a 239.255.255.255. Endereços IPv6 multicast começam com ff0X::/8, onde X indica o escopo (por exemplo, ff02::1 é o escopo link-local).

A seguir está um exemplo de código Python para comunicação via UDP multicast, que permite o envio de dados de um servidor para vários clientes ao mesmo tempo. Esse exemplo é funcional em uma rede local (LAN).

- O servidor cria um socket UDP e define o endereço multicast (neste caso, `224.1.1.1`), que é um endereço dentro do intervalo de endereços de multicast.
- O servidor configura o TTL (*Time to Live*), que define o número máximo de roteadores que o pacote multicast pode atravessar. Neste caso, um único salto.
- Ele então envia periodicamente uma mensagem para o grupo multicast, que pode ser recebido por todos os dispositivos inscritos no grupo.

python - Servidor UDP Multicast

```python
import socket
import struct
import time

# Criação do socket UDP
# IPv4: 224.0.0.0 a 239.255.255.255, IPv6='ff02::1'
multicast_group = '224.1.1.1'
server_address = ('', 12345)

# Configuração do socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Definindo TTL (Time to Live) para o pacote multicast
ttl = struct.pack('b', 1)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)

# Enviando mensagens ao grupo multicast
try:
    while True:
```

```
        message = "Mensagem multicast UDP"
        print(f"Enviando: {message}")
        sent = sock.sendto(message.encode(),
                           (multicast_group, 12345))
        time.sleep(2)  # Enviar a cada 2 segundos
finally:
    sock.close()
```

O código do cliente é apresentado a seguir. O cliente também cria um socket UDP e se inscreve no grupo multicast especificado (`224.1.1.1`). Ele usa o método `recvfrom()` para receber as mensagens enviadas ao grupo multicast pelo servidor.

python - Cliente UDP Multicast

```python
import socket
import struct

# Criação do socket UDP
multicast_group = '224.1.1.1'  # Mesmo grupo do servidor
server_address = ('', 12345)

# Configuração do socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Vincular o socket à porta do servidor
sock.bind(server_address)

# Informar que o cliente quer se juntar ao grupo multicast
group = socket.inet_aton(multicast_group)
mreq = struct.pack('4sL', group, socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

# Receber mensagens do grupo multicast
try:
    while True:
        print("Aguardando mensagem multicast...")
        data, address = sock.recvfrom(1024)
        print(f"Recebido: {data.decode()} de {address}")
finally:
    sock.close()
```