# Pycharm community를 이용한 TFTP 클라이언트 구현

2024. 12. 15  
성명: 배준하  
Github URL: [Github URL]

## I. TFTP 프로토콜 개요

Trivial File Transfer Protocol (TFTP)은 UDP 기반의 파일 전송 프로토콜로, 간단한 파일 업로드와 다운로드를 지원합니다. TFTP는 69번 포트를 사용하며, 주로 간단한 파일 전송 작업에 사용됩니다. 본 과제에서는 TFTP 클라이언트를 구현하여 RRQ (Read Request)와 WRQ (Write Request) 메시지를 이용한 파일 송수신을 수행하였습니다.

## II. 프로그램 설명

### 1. 프로그램 개요

본 프로그램은 파이썬 소켓 프로그래밍을 사용하여 작성되었으며, TFTP 서버와 파일을 송수신하는 기능을 제공합니다. 클라이언트는 get 명령어를 통해 파일을 다운로드하거나, put 명령어를 통해 업로드를 수행할 수 있습니다.

### 2. 구현 기능

- 서버와의 연결 설정 (UDP 기반)

프로그램은 UDP 프로토콜을 사용하여 TFTP 서버와 통신합니다. TFTP는 UDP 기반의 단순 파일 전송 프로토콜로, 클라이언트는 TFTP 서버의 IP 주소와 포트를 통해 데이터그램을 송수신합니다.클라이언트는 초기 설정에서 socket.AF\_INET과 socket.SOCK\_DGRAM을 사용해 UDP 소켓을 생성하고, 이를 통해 데이터 패킷을 송수신합니다.

- RRQ, WRQ 메시지를 통한 파일 다운로드 및 업로드

**RRQ(Read Request):**  
클라이언트가 서버에 파일 다운로드를 요청하는 메시지입니다. 사용자는 명령어에서 get 옵션과 파일명을 입력하여 요청을 보낼 수 있습니다.

**WRQ(Write Request):**  
클라이언트가 서버에 파일 업로드를 요청하는 메시지입니다. 명령어에서 put 옵션과 파일명을 입력하여 요청을 전송합니다.  
메시지 형식은 RRQ와 동일하며 Opcode가 2(WRQ)로 설정됩니다.클라이언트는 요청 메시지를 전송한 후, 서버의 응답에 따라 데이터를 주고받거나 에러 처리를 진행합니다.

- 타임아웃 및 오류 처리

UDP는 비연결형 프로토콜로, 데이터가 손실될 가능성이 있습니다. 따라서 클라이언트는 패킷 송수신 시 타임아웃을 설정하여 서버 응답이 없을 경우 재시도 로직을 수행합니다.

또한, TFTP에서 정의한 에러 코드(0~7)를 처리하여 전송 실패 시 사용자에게 원인을 알립니다. 예를 들어, 파일이 존재하지 않거나 권한 문제가 발생하면 에러 메시지를 출력합니다.

- ACK 메시지를 이용한 전송 확인

클라이언트는 서버로부터 데이터패킷을 받을 때마다 해당 블록 번호를 포함한 ACK 메시지를 전송하여 성공적으로 데이터를 수신했음을 알립니다.

서버가 ACK를 받지 못할 경우, 동일한 데이터 블록을 다시 전송하므로 클라이언트는 중복 데이터를 방지하기 위해 블록 번호를 기반으로 확인합니다.

- 명령어를 통한 포트 설정 지원

기본적으로 TFTP는 69번 포트를 사용하지만, 특정 환경에서 서버의 포트가 다르게 설정될 수 있습니다. 이를 지원하기 위해 사용자가 명령어 옵션을 통해 포트를 지정할 수 있도록 설계했습니다.

클라이언트는 사용자가 지정한 포트를 통해 서버와 통신하며, 요청 메시지 전송 이후 서버가 할당한 전송 포트로 데이터 송수신을 이어갑니다.

### 3. 사용법

1. 프로그램 실행 명령어:  
 $ python3 tftp\_client.py 서버\_IP [get|put] 파일명 [-p 포트번호]

서버\_IP: TFTP 서버의 IP 주소를 입력합니다.

[get|put]: 파일을 다운로드하려면 get을, 파일을 업로드하려면 put을 입력합니다.

파일명: 다운로드하거나 업로드할 파일의 이름을 입력합니다.

-p 포트번호: 서버가 기본 포트(69)가 아닌 다른 포트를 사용할 경우, 이 옵션을 통해 포트를 지정할 수 있습니다. 포트번호를 생략하면 기본 포트(69)가 사용됩니다.  
  
2. 사용 예시:  
 -파일 다운로드:  
서버에서 파일을 다운로드하려면 get 명령어를 사용합니다. 예를 들어, example.txt라는 파일을 다운로드하려면 아래와 같이 실행합니다.

$ python3 tftp\_client.py 203.250.133.88 get example.txt

이 명령은 서버 203.250.133.88에서 example.txt 파일을 다운로드합니다.

- 파일 업로드:

서버에 파일을 업로드하려면 put 명령어를 사용합니다. 예를 들어, example.txt라는 파일을 업로드하려면 아래와 같이 실행합니다.

$ python3 tftp\_client.py 203.250.133.88 put example.txt

이 명령은 클라이언트에서 example.txt 파일을 서버에 업로드합니다.

- 커스텀 포트 사용:

서버가 기본 포트(69)가 아닌 다른 포트를 사용할 경우 -p 옵션을 사용하여 포트를 지정할 수 있습니다. 예를 들어, 서버가 9988번 포트를 사용할 경우 아래와 같이 실행합니다.

$ python3 tftp\_client.py 203.250.133.88 -p 9988 put example.txt

이 명령은 203.250.133.88 서버의 9988번 포트를 통해 example.txt 파일을 업로드합니다.

**추가 설명**:

프로그램 실행 후 파일 다운로드 또는 업로드가 완료되면, 파일이 클라이언트의 현재 작업 디렉터리에 저장되거나, 업로드된 파일은 서버의 지정된 위치에 저장됩니다.

명령어 옵션을 통해 TFTP 서버와의 안정적인 파일 송수신을 보장하며, 타임아웃 및 오류 처리 기능이 포함되어 있어 전송 중 발생할 수 있는 문제를 자동으로 처리합니다.

## III. 소스코드 주석

# 파일 업로드 요청을 위한 WRQ(Write Request) 메시지를 서버로 전송하는 함수

def send\_wrq(filename, mode):

# WRQ 메시지 포맷 설정: Opcode, 파일명, 모드

format = f'>h{len(filename)}sB{len(mode)}sB'

# WRQ 패킷 생성: OPCODE[WRQ], 파일명과 모드(둘 다 null-terminated)

wrq\_message = pack(format, OPCODE['WRQ'], bytes(filename, 'utf-8'), 0, bytes(mode, 'utf-8'), 0)

# 서버에 WRQ 메시지 전송

sock.sendto(wrq\_message, server\_address)

# 파일 다운로드 요청을 위한 RRQ(Read Request) 메시지를 서버로 전송하는 함수

def send\_rrq(filename, mode):

# RRQ 메시지 포맷 설정: Opcode, 파일명, 모드

format = f'>h{len(filename)}sB{len(mode)}sB'

# RRQ 패킷 생성: OPCODE[RRQ], 파일명과 모드(둘 다 null-terminated)

rrq\_message = pack(format, OPCODE['RRQ'], bytes(filename, 'utf-8'), 0, bytes(mode, 'utf-8'), 0)

# 서버에 RRQ 메시지 전송

sock.sendto(rrq\_message, server\_address)

# 데이터 전송에 대한 확인 응답인 ACK(Acknowledgment) 메시지를 서버에 전송하는 함수

def send\_ack(seq\_num, server):

# ACK 메시지 포맷 설정: Opcode와 블록 번호

format = f'>hh'

# ACK 메시지 생성: OPCODE[ACK], 블록 번호

ack\_message = pack(format, OPCODE['ACK'], seq\_num)

# 서버에 ACK 메시지 전송

sock.sendto(ack\_message, server)

# ACK 메시지 내용 출력 (디버깅용)

print(f"ACK 전송: 블록 번호 {seq\_num}")

print(ack\_message)

# 데이터 블록을 서버로 전송하는 함수

def send\_data(block\_num, data\_block, server):

"""Send DATA packet to server."""

# DATA 메시지 포맷 설정: Opcode, 블록 번호, 데이터

format = f'>hh{len(data\_block)}s'

# 데이터 패킷 생성: OPCODE[DATA], 블록 번호와 데이터

data\_message = pack(format, OPCODE['DATA'], block\_num, data\_block)

# 서버에 데이터 패킷 전송

sock.sendto(data\_message, server)

print(f"데이터 블록 {block\_num} 전송 완료")

# 파일 업로드를 처리하는 함수 (파일을 서버에 전송)

def put\_file(filename, mode):

# 파일 업로드를 위한 WRQ 메시지 전송

send\_wrq(filename, mode)

# 파일을 읽기 모드로 열기

try:

file = open(filename, 'rb')

except FileNotFoundError:

print("파일을 찾을 수 없습니다.")

sys.exit(1)

block\_num = 0

while True:

# 파일을 블록 크기(BLOCK\_SIZE)만큼 읽어오기

file\_block = file.read(BLOCK\_SIZE)

block\_num += 1

# 파일 블록을 서버에 전송

send\_data(block\_num, file\_block, server\_address)

# 서버로부터 ACK 응답을 기다리기

try:

data, server\_new\_socket = sock.recvfrom(516) # 최대 516바이트 수신 (2바이트 opcode + 2바이트 블록 번호 + 512바이트 데이터)

opcode = int.from\_bytes(data[:2], 'big')

if opcode == OPCODE['ACK']:

ack\_block\_num = int.from\_bytes(data[2:4], 'big')

# ACK가 예상한 블록 번호와 일치하는지 확인

if ack\_block\_num == block\_num:

print(f"블록 {block\_num}에 대한 ACK를 받았습니다.")

else:

print(f"예상과 다른 ACK 블록 번호: {ack\_block\_num}")

elif opcode == OPCODE['ERROR']:

error\_code = int.from\_bytes(data[2:4], byteorder='big')

# 오류 발생 시 에러 메시지 출력

print(f"서버에서 오류 발생: {ERROR\_CODE[error\_code]}")

file.close()

sys.exit(1)

except socket.timeout:

print(f"블록 {block\_num}에 대한 ACK를 기다리는 중 타임아웃 발생")

file.close()

sys.exit(1)

# 마지막 블록이 블록 크기보다 작은 경우 전송 완료

if len(file\_block) < BLOCK\_SIZE:

print("파일 업로드가 완료되었습니다.")

break

# 파일 닫기

file.close()

# 메인 함수: 명령줄 인자를 파싱하고 동작 모드를 결정하여 적절한 함수를 호출

# 예: python3 tftp\_client.py 203.250.133.88 get example.txt

if \_\_name\_\_ == "\_\_main\_\_":

# 명령줄 인자 파싱

parser = argparse.ArgumentParser(description='TFTP 클라이언트 프로그램')

parser.add\_argument(dest="host", help="서버 IP 주소", type=str)

parser.add\_argument(dest="operation", help="get 또는 put 명령어", type=str)

parser.add\_argument(dest="filename", help="파일명", type=str)

parser.add\_argument("-p", "--port", dest="port", type=int)

args = parser.parse\_args()

# 서버 IP 주소 및 포트 설정

server\_ip = args.host

server\_port = DEFAULT\_PORT

server\_address = (server\_ip, server\_port)

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# 기본 전송 모드 설정 (octet 모드)

mode = DEFAULT\_TRANSFER\_MODE

operation = args.operation

filename = args.filename

# 파일 다운로드 요청 (get) 또는 업로드 요청 (put)에 따라 함수 호출

if operation == "get":

send\_rrq(filename, mode) # RRQ 요청 (파일 다운로드)

file = open(filename, 'wb') # 받은 파일을 로컬에 저장

expected\_block\_number = 1

while True:

data, server\_new\_socket = sock.recvfrom(516) # 최대 516바이트 수신

opcode = int.from\_bytes(data[:2], 'big')

if opcode == OPCODE['DATA']:

block\_number = int.from\_bytes(data[2:4], 'big')

if block\_number == expected\_block\_number:

send\_ack(block\_number, server\_new\_socket)

file\_block = data[4:]

file.write(file\_block)

expected\_block\_number += 1

print(file\_block.decode())

else:

send\_ack(block\_number, server\_new\_socket)

elif opcode == OPCODE['ERROR']:

error\_code = int.from\_bytes(data[2:4], byteorder='big')

print(ERROR\_CODE[error\_code])

file.close()

os.remove(filename)

break

# 마지막 블록인 경우 파일 전송 종료

if len(file\_block) < BLOCK\_SIZE:

file.close()

print("파일 전송이 완료되었습니다.")

break

elif operation == "put":

put\_file(filename, mode) # 파일 업로드 처리

else:

print("잘못된 동작입니다. 'get' 또는 'put'을 사용하세요.")

sys.exit(1).

## IV. 코멘트

이번 과제를 통해 UDP 기반 네트워크 프로그래밍과 TFTP 프로토콜의 동작 원리를 깊이 이해할 수 있었습니다. UDP의 비연결형 특성에서 발생하는 문제들, 예를 들어 데이터 손실이나 순서 보장 문제를 해결하기 위해 타임아웃 처리와 ACK 확인을 직접 설계하면서, 네트워크 통신이 실제로 어떻게 이루어지는지에 대해 새롭게 알 수 있었습니다. 특히 TFTP 프로토콜에서 사용되는 RRQ와 WRQ메시지, 그리고 데이터/ACK 의구조를 구현하면서 프로토콜 설계가 프로그램의 안정성에 얼마나 중요한지 실감했습니다.

이 과제는 단순히 코드를 작성하는 것이 아니라, 네트워크 프로그래밍에 대한 실질적인 이해를 높일 수 있는 좋은 기회였습니다. 프로그램을 정확하고 안정적으로 만드는 것이 얼마나 중요한지 깨닫게 되었고, 효율적이고 신뢰할 수 있는 네트워크 설계의 중요성도 다시 한번 인식하게 되었습니다. 이번 경험을 바탕으로, 앞으로 더 복잡한 네트워크 응용 프로그램을 설계하고 구현하는 데 자신감을 가질 수 있을 것 같습니다. 또한, 복잡한 시스템을 다룰 때 효율성과 신뢰성을 고려하는 설계의 중요성을 잘 알게 되었습니다.