Slovenská Technická Univerzita Fakulta Informatiky a Informačných technológií

Komunikácia s využitím UDP protokolu

Počítačové a Komunikačné Siete

Norbert Matuška
11-19-2023

Contents

Štvrtok 18:00 AIS_ID: 110849
xmatuskan@stuba.sk

Zadanie

Navrhnite a implementujte program s použitím vlastného protokolu nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP. Program umožní komunikáciu dvoch účastníkov v lokálnej sieti Ethernet, teda prenos textových správ a ľubovoľného súboru medzi počítačmi (uzlami).

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošle súbor inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Ak je posielaný súbor väčší, ako používateľom definovaná max. veľkosť fragmentu, vysielajúca strana rozloží súbor na menšie časti - fragmenty, ktoré pošle samostatne. Maximálnu veľkosť fragmentu musí mať používateľ možnosť nastaviť takú, aby neboli znova fragmentované na linkovej vrstve.

Ak je súbor poslaný ako postupnosť fragmentov, cieľový uzol vypíše správu o prijatí fragmentu s jeho poradím a či bol prenesený bez chýb. Po prijatí celého súboru na cieľovom uzle tento zobrazí správu o jeho prijatí a absolútnu cestu, kam bol prijatý súbor uložený.

Program musí obsahovať kontrolu chýb pri komunikácii a znovuvyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po zapnutí programu, komunikátor automaticky odosiela paket pre udržanie spojenia každých 5s pokiaľ používateľ neukončí spojenie ručne. Odporúčame riešiť cez vlastne definované signalizačné správy a samostatné vlákno.

Úloha 1 - Návrh programu a komunikačného protokolu

Hodnotí sa prehľadnosť a zrozumiteľnosť odovzdanej dokumentácie ako aj kvalita navrhovaného riešenia. 5 bodov získa študent, ktorý má v dokumentácií uvedené všetky podstatné informácie o fungovaní jeho programu. Korektne navrhnutú štruktúru hlavičky vlastného protokolu, opis použitej metódy kontrolnej sumy a fungovania ARQ, metódy pre udržanie spojenia, diagram spracovávania komunikácie na oboch uzloch (sekvenčný), popis jednotlivých častí zdrojového kódu (knižnice, triedy, metódy, ...). Podčiarknuté požiadavky sú minimálne.

Návrh

Úvod

Cieľom je vytvoriť program na prenos textových správ a súborov medzi dvoma počítačmi v lokálnej sieti pomocou vlastného protokolu nad UDP. Program bude implementovaný v jazyku Python, kvôli uľahčeniu práce vďaka veľkému množstvu built-in funkcií a externých knižníc.

Komponenty programu

Program bude pozostávať z dvoch hlavných častí – vysielač (sender) a prijímač (receiver).

Vysielač (sender)

Vysielač bude mať na zodpovednosť nasledovné:

- Rozdelenie veľkého súboru na menšie fragmenty.
- Pridávanie hlavičky k jednotlivým fragmentom.
- Posielanie fragmentov pomocou UDP.
- Čakanie na potvrdenia prijatia fragmentov od prijímača.
- Znovuodoslanie fragmentov v prípade neúspešného doručenia.

Pseudo kód:

```
function handleCommunication():
    sendConnectionSetupMessage()
    if receiveAcknowledgementForSetup():
        sendFile(file)
    sendConnectionTeardownMessage()

function sendFile(file):
    fragments = splitFileIntoFragments(file)
    for fragment in fragments:
        header = createHeader(fragment)
        sendData(header + fragment)
        if not receiveAcknowledgement():
            resendData(header + fragment)

function createHeader(fragment):
    header = new Header()
    header.sequenceNumber = fragment.sequenceNumber
    header.checksum = calculateChecksum(fragment.data)
    return header

function receiveAcknowledgement():
    # čakanie na potvrdenie od prijímača
    # vráti True, ak bolo prijaté, inak False
```

Prijímač (receiver)

Zodpovednosti:

- Prijímanie a kontrola fragmentov.
- Vedenie záznamov o prijatých fragmentoch.
- Odosielanie potvrdení vysielaču.
- Zloženie pôvodného súboru z prijatých fragmentov.

Pseudo kód:

```
# implementácia s typom header (nadviazanie spojenia, fragment, atď.)
function receiveData():
    while True:
        message = receiveMessage()
        switch (message.type):
            case ConnectionSetup:
                sendAcknowledgementForSetup()
            case DataFragment:
                processFragment(message)
                sendAcknowledgementForFragment(message)
            case ConnectionTeardown:
                sendAcknowledgementForTeardown()
                break

# implementácia bez prvotného nadviazania spojenia
function receiveData():
    while True:
        data, header = receiveFragment()
        if isValid(data, header.checksum):
            sendAcknowledgement(header.sequenceNumber)
            storeFragment(data)
            if isLastFragment(header):
                break

function isValid(data, checksum):
    return calculateChecksum(data) == checksum

function sendAcknowledgement(sequenceNumber):
    # odoslanie potvrdenia späť k vysielaču
```

Návrh protokolu

Hlavička (Header)

Type	ID	Frag_ID	Frag_size	
Checksum		SeqN	Príznaky	DATA

Typ	ID	Frag_ID	Frag_size	Checksum	SeqN	Príznaky	Data
1 byte	2 bytes	2 byte	2 bytes	4 bytes	2 bytes	1 byte	

Typ – Typ správy, ktorý identifikuje účel správy (napr. Nadviazanie Spoje, Dátový Fragment, Potvrdenie, Ukončenie Spoje).

ID – Identifikátor správy alebo teda jednotlivej komunikácie

Frag_ID - Poskytuje jedinečný identifikátor pre každý fragment súboru, čo umožňuje prijímaču správne zoradiť fragmenty.

Frag_size – informuje prijímača o veľkosti obsahu fragmentu

Checksum - Slúži na overenie integrity dát počas prenosu. Ak sa vypočítaný kontrolný súčet prijatých dát nezhoduje s hodnotou v hlavičke, dáta boli poškodené.

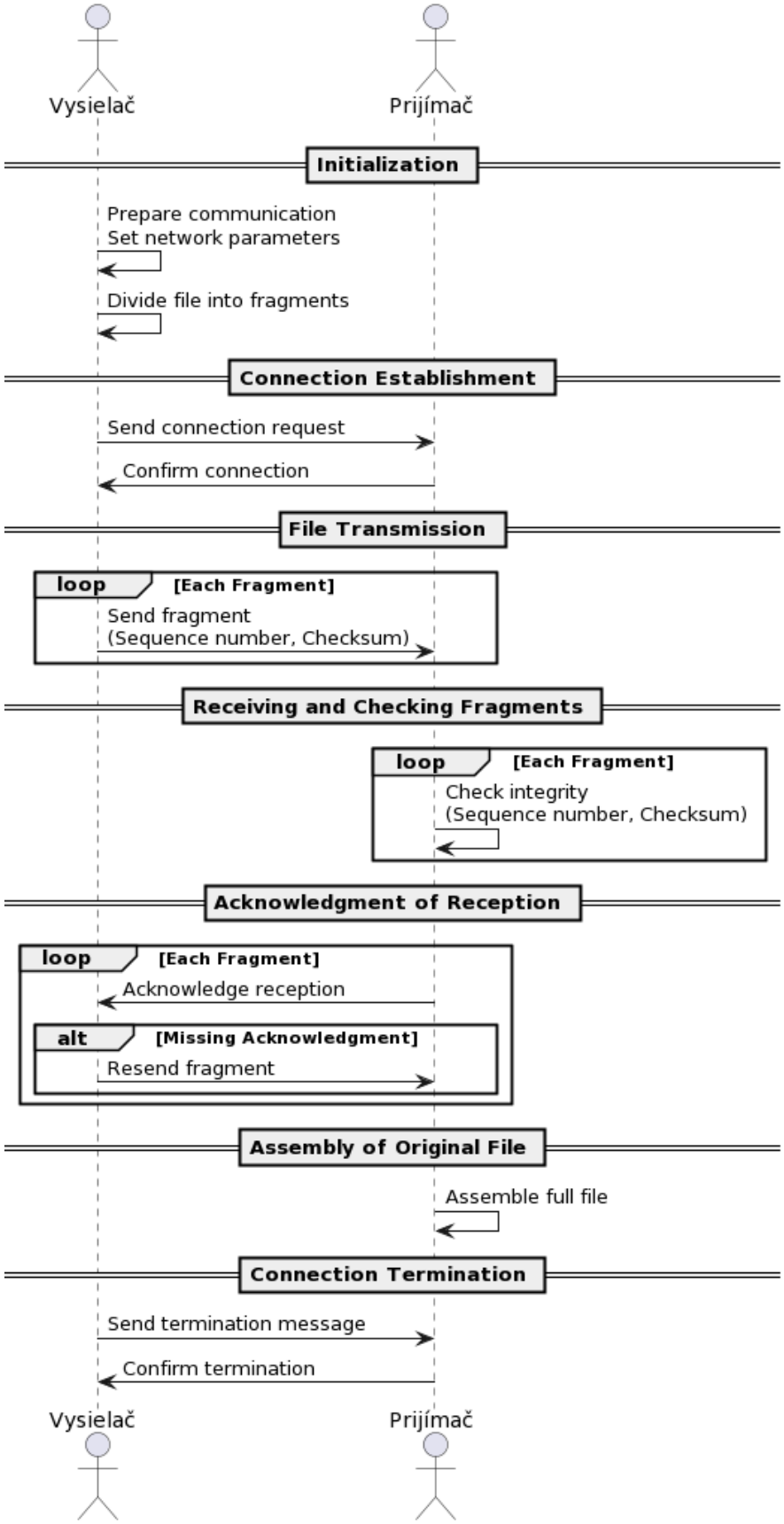
SeqN - Sledovať a riadiť poradie správ alebo fragmentov v dlhodobejšom komunikačnom toku. Toto môže byť užitočné pre detekciu straty dát a správne poradie spracovania viacerých súčasných správ alebo transakcií.

Príznaky - Slúži na signalizáciu špeciálnych stavov alebo podmienok, ako je napríklad priorita správy, vyžiadanie znovu odoslania, indikácia chyby alebo môže aj indikovať, o aký typ dát ide

Komunikačný proces

Komunikačný proces medzi vysielačom a prijímačom je možné rozdeliť do niekoľkých kľúčových fáz.

1. Inicializácia
    a. Vysielač a prijímač sa pripravia na komunikáciu, nastavia sieťové parametre (IP adresy, porty).
    b. Vysielač rozdelí veľký súbor na menšie fragmenty podľa definovanej maximálnej veľkosti fragmentu.
2. Nadviazanie spojenia
    a. Vysielač začne komunikáciu odoslaním správy na nadviazanie spojenia.
    b. Prijímač potvrdí nadviazanie spojenia.
3. Odoslanie súboru
    a. Vysielač postupne odosiela fragmenty súboru, každý s vlastnou hlavičkou obsahujúcou dôležité informácie (sekvenčné číslo, kontrolný súčet)
4. Prijímanie a kontrola fragmentov
    a. Prijímač prijíma fragmenty, kontroluje ich integritu (pomocou kontrolného súčtu) a poradie (sekvenčné číslo).
5. Potvrdzovanie prijatia
    a. Po úspešnom prijatí fragmentu prijímač odošle späť potvrdenie o prijatí.
    b. V prípade, že vysielač neprijme potvrdenie o prijatí, znovu odošle príslušný fragment.
6. Zloženie pôvodného súboru
    a. Prijímač po prijatí všetkých fragmentov zloží späť celý súbor.
7. Ukončenie spojenia
    a. Po dokončení prenosu dát vysielač odošle správu na ukončenie spojenia.
    b. Prijímač potvrdí ukončenie spojenia.


Vysielač
Prijímač
Initialization
Prepare communication
Set network parameters
Divide file into fragments
Connection Establishment
Send connection request
Confirm connection
File Transmission
loop
[Each Fragment]
Send fragment
(Sequence number, Checksum)
Receiving and Checking Fragments
loop
[Each Fragment]
Check integrity
(Sequence number, Checksum)
Acknowledgment of Reception
loop
[Each Fragment]
Acknowledge reception
alt
[Missing Acknowledgment]
Resend fragment
Assembly of Original File
Assemble full file
Connection Termination
Send termination message
Confirm termination
Vysielač
Prijímač

Udržiavanie spojenia

Periodické odosielanie „keep-alive“ správ pre udržovanie aktívneho spojenia. Príklad pre možnú implementáciu tohto pomocou threadov:

```
def keep_alive(sender_socket, destination):
    while True:
        sender_socket.sendto(b'keep_alive', destination)
        time.sleep(5)

# Inicializácia socketu
sender_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
destination = ('<IP_ADRESA_PRIJÍMAČA>', <PORT_PRIJÍMAČA>)

# Spustenie keep-alive vlákna
thread = threading.Thread(target=keep_alive, args=(sender_socket,
destination))
thread.start()
```

Kontrola chýb

Automatic Repeat reQuest (ARQ)

ARQ je metóda pre zabezpečenie spoľahlivého prenosu dát nad nespoľahlivými protokolmi. Základným princípom ARQ je potvrdenie prijatia každého dátového bloku. Ak vysielač nedostane potvrdenie (ACK) v stanovenom čase, predpokladá sa, že dátový blok bol stratený alebo poškodený, a vysielač ho znova odosiela.

Implementovať sa dajú rôzne metódy, ale nechcem sľubovať veľa takže zatiaľ si vyberám metódu Stop-and-Wait: Po odoslaní každého bloku čaká vysielač na potvrdenie, než odosiela ďalší blok.

Cyclic Redundancy Check (CRC)

CRC je metóda používaná na detekciu chýb v digitálnych dátach. Pri vysielači sa pre každý odosielaný dátový blok (napríklad fragment súboru) vypočíta CRC hodnota, ktorá sa potom pridá do hlavičky dátového bloku. Prijímač potom vypočíta CRC hodnotu pre prijaté dáta a porovná ju s hodnotou v hlavičke. Ak sa tieto hodnoty nezhodujú, znamená to, že v dátach došlo počas prenosu k chybe.

```
def calculate_crc(data):
    # Vypočíta CRC hodnotu pre dané dáta
    return crc_value

def send_data(data):
    crc = calculate_crc(data)
    # Pridá CRC hodnotu do hlavičky a odosiela dáta
    send(header + crc + data)

def receive_data(data):
    received_crc = extract_crc_from_header(data)
    if calculate_crc(data) != received_crc:
        # Chyba v dátach, požiadať o znovuodoslanie
```

Implementácia

Program bude implementovaný v jazyku python a knižnice, ktoré budem s najväčšou pravdepodobnosťou používať sú: time(keep alive signál), threading((pseudo)paralelné vykonávanie procesov pre prijímača a vysielača), socket(vytvaránie socketu spojenia), struct(binárne dáta), os/sys(práca so súbormi na počítači).

Testovanie

Základný prenos dát

Cieľ: Overiť, že základný prenos dát (textové správy a súbory) funguje správne.

Scenár: Vysielač odošle rôzne typy súborov (malé, stredné, veľké) a textové správy prijímačovi. Prijímač overí integritu a poradie prijatých dát.

```
def test_basic_data_transfer():
    sender = Sender()
    receiver = Receiver()

    sender.sendFile("test_file_small.txt")
    sender.sendMessage("Hello World")

    assert receiver.hasReceivedFile("test_file_small.txt")
    assert receiver.hasReceivedMessage("Hello World")
```

Test fragmentácie a znovu-zloženia súborov

Cieľ: Zabezpečiť, že fragmentácia a znovu-zloženie veľkých súborov funguje správne.

Scenár: Vysielač pošle veľký súbor, ktorý prekračuje maximálnu veľkosť fragmentu, a prijímač ho rozdelí na fragmenty, prijme, a potom zloží späť.

```
def test_file_fragmentation_and_reassembly():
    sender = Sender(max_fragment_size=1024)
    receiver = Receiver()

    sender.sendFile("large_test_file.zip")

    assert receiver.hasReceivedFile("large_test_file.zip")
    assert receiver.isFileIntact("large_test_file.zip")
```

Simulácia straty paketov

Cieľ: Otestovať odolnosť protokolu voči strate paketov.

Scenár: Umele simulovať stratu paketov počas prenosu. Vysielač by mal znovu odoslať chýbajúce fragmenty.

```
def test_packet_loss_resilience():
    network = NetworkSimulator(loss_rate=0.1)  # 10% strata paketov
    sender = Sender(network=network)
    receiver = Receiver(network=network)

    sender.sendFile("test_file_loss.txt")

    assert receiver.hasReceivedFile("test_file_loss.txt")
    assert receiver.isFileIntact("test_file_loss.txt")
```

Test kontroly chýb

Cieľ: Overiť efektívnosť mechanizmov kontroly chýb.

Scenár: Umele poškodiť niektoré dáta v odosielaných paketoch. Prijímač by mal detekovať chybu a požiadať o znovu-odoslanie.

```
def test_error_detection_and_correction():
    network = NetworkSimulator(corrupt_rate=0.05)  # 5% chybovosti
    sender = Sender(network=network)
    receiver = Receiver(network=network)

    sender.sendFile("test_file_error.txt")

    assert receiver.hasReceivedFile("test_file_error.txt")
    assert receiver.isFileIntact("test_file_error.txt")
```

Všetky testy majú pri sebe len pseudo-kódy a slúžia len na lepšiu predstavu spôsobu testovania.

Zmeny oproti návrhu

Nová Hlavička

SeqN, Príznaky a frag_size bolo zbytočné posielať v každom pakete, preto som upravil hlavičku nasledovne:

TYPE_ID	MSG_ID	FRAG_ID	CHECKSUM
1 byte	2 bytes	2 bytes	4 bytes

TYPE_ID možné typy:

```
TYPE_SYN = 1
TYPE_SYN_ACK = 2
TYPE_ACK = 3
TYPE_TXT_MSG = 4
TYPE_FILE = 5
TYPE_FILE_END = 6
TYPE_KEEP_ALIVE = 7
TYPE_CHANGE_FRAG_SIZE = 8
TYPE_MESSAGE_END = 9
TYPE_SWITCH = 10
TYPE_FRAGS_AND_SIZE = 11
```

Keep Alive

Keep alive nakoniec neimplementujem, bolo to nad moje sily.

Základny opis kódu

Moj skript implementuje základný komunikačný systém pomocou soketov UDP, čo umožňuje výmenu textových správ a súborov medzi odosielateľom a prijímateľom. Zahŕňa funkcie ako iniciačný handshake, potvrdenie prijatých paketov, prenos súborov a simuláciu chýb.

Konštanty

- TYPE_SYN, TYPE_SYN_ACK, TYPE_ACK atď.: Celé čísla reprezentujúce rôzne typy správ v protokole.

- msg_id: Globálna premenná na sledovanie ID správy.

- MAX_FRAGMENT_SIZE: Maximálna veľkosť fragmentu dát.

- HEADER_FORMAT: String predurčeného formátu hlavičky na pack funkciu a unpack funkciu pre hlavičku.

Trieda Header

- Účel: Predstavuje hlavičku správy.

- Metódy: - __init__(type_id, msg_id, frag_id, checksum): Konštruktor na inicializáciu hlavičky. - pack(): Vracia bajtovú reprezentáciu hlavičky. - unpack(data): Statická metóda na rozbalenie bajtovej reprezentácie do objektu Header.

Trieda Sender

- Účel: Zabezpečuje odosielanie správ a súborov.

- Inicializácia: Vyžaduje IP servera a port.

- Metódy: - initiate_handshake(): Iniciuje handshake s prijímateľom.

- wait_for_ack(expected_msg_id, expected_frag_id): Čaká na potvrdenie pre konkrétne ID správy a fragmentu.

- send_message(): Zakóduje a odošle textovú správu vo fragmentoch.

- send_file(): Odošle súbor vo fragmentoch.

- change_fragment_size(): Zmení maximálnu veľkosť fragmentu pre odosielanie správ.

- switch(): Prepne úlohu z odosielateľa na prijímateľa.

Trieda Receiver

- Účel: Zabezpečuje prijímanie správ a súborov.

- Inicializácia: Vyžaduje IP servera a port.

- Metódy: - handle_handshake(): Zaoberá sa procesom handshake pri štarte.

- send_ack(msg_id, frag_id, addr): Odošle potvrdenie odosielateľovi.

- receive_message(): Prijíma správy a zaoberá sa rôznymi typmi prijatých dát.

Funkcie

- compute_crc(data): Vypočíta kontrolný súčet CRC32 pre dané dáta pomocou knižnice zlib

crc32(data) vypočíta kontrolný súčet CRC-32 pre dané dáta. CRC-32 je populárny algoritmus kontrolného súčtu používaný na detekciu chýb pri ukladaní alebo prenose dát. Nie je kryptograficky bezpečný, ale ponúka dobrú úroveň detekcie chýb pre praktické použitie. Výsledkom crc32(data) je celé číslo, ktoré predstavuje kontrolný súčet vstupných dát. Tento kontrolný súčet je 32-bitové číslo, odtiaľ názov CRC-32.

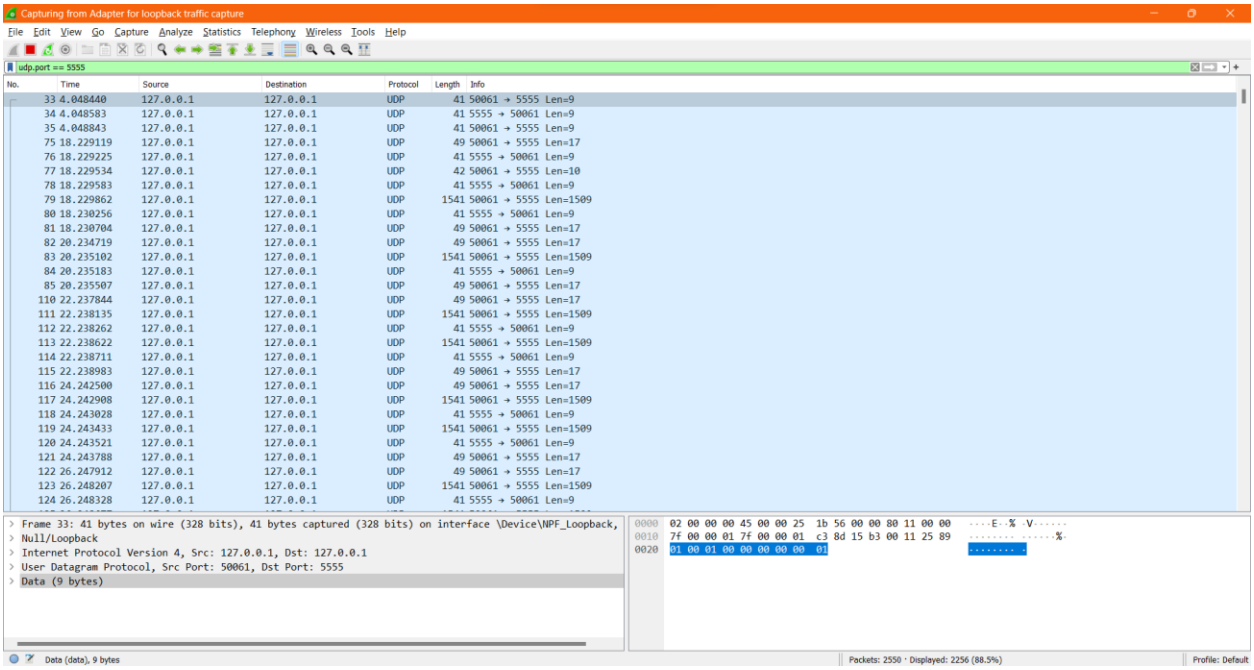
Bitová operácia AND s 0xffffffff efektívne oreže výsledok zlib.crc32(data) na 32 bitov. Toto je potrebné, pretože vnútorné reprezentácie kontrolného súčtu môžu byť závislé od platformy (buď 32-bitové alebo 64-bitové) a táto operácia zabezpečuje konzistentný 32-bitový výstup naprieč všetkými platformami.

- start_sender(server_ip, port): Spustí odosielateľa s danou IP a portom.

- start_receiver(server_ip, port): Spustí prijímateľa s danou IP a portom.

- main_menu(): Zobrazí hlavné menu pre používateľa, aby si vybral úlohu (Odosielateľ/Prijímateľ).

Ukážka komunikácie vo wireshark-u



Chybu simulujem nasledovne:

```
if str(error).lower() == "yes" and count < num_of_errors:
    if random.random() < 0.5:
        checksum = compute_crc(fragment) + 1
        count += 1
        header = Header(TYPE_TXT_MSG, msg_id, frag_id, checksum)
        self.sock.sendto(header.pack() + fragment, (self.server_ip,
self.port))
        if not self.wait_for_ack(msg_id, frag_id):
            self.sock.sendto(header.pack() + fragment, (self.server_ip,
self.port))
```