Počítačové a komunikačné siete

Komunikácia s využitím UDP protokolu

Vereski Tijana

2019/2020

Cvičiaci: Marek Galinski

1. 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 binárneho 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.

Komunikátor musí obsahovať kontrolu chýb pri komunikácii a znovuvyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 20-60s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

Program musí mať nasledovné vlastnosti (minimálne):

1. Program musí byť implementovaný v jazykoch C/C++ alebo Python s využitím knižníc na prácu s UDP socket, skompilovateľný a spustiteľný v učebniach. Odporúčame použiť python modul socket, C/C++ knižnice sys/socket.h pre linux/BSD a winsock2.h pre Windows. Použité knižnice a funkcie musia byť schválené cvičiacim. V programe môžu byť použité aj knižnice na prácu s IP adresami a portami:

arpa/inet.h

netinet/in.h

2. Program musí pracovať s dátami optimálne (napr. neukladať IP adresy do 4x int).

3. Pri posielaní súboru musí používateľovi umožniť určiť cieľovú IP a port.

4. Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu.

5. Obe komunikujúce strany musia byť schopné zobrazovať:

a. názov a absolútnu cestu k súboru na danom uzle,

b. veľkosť a počet fragmentov.

6. Možnosť odoslať minimálne 1 chybný fragment (do fragmentu je cielene vnesená chyba, to znamená, že prijímajúca strana deteguje chybu pri prenose).

7. Prijímajúca strana musí byť schopná oznámiť odosielateľovi správne aj nesprávne doručenie fragmentov.

8. Možnosť odoslať súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor. Akceptuje sa iba ak program prenesie 2MB súbor do 60s bez chýb.

2. Analýza:

2.1 Enkapsulácia a hlavičky

Program bude využívať funkcionalitu UDP socketu na posielanie a prijímanie dát medzi dvomi uzlami a implementovať vlastný návrh enkapsulácie dát prostredníctvom umelej hlavičke, taktiež bude používať CRC na kontrolovanie fragmentov.

Enkapsulácia je pridanie ďalších hlavičiek k dátam ako prechádzajú jednotlivo cez vrstvy.Koná sa na štyroch úrovniach a pri každej úrovni sa pridá nová hlavička k dátam. Najprv sa pridá hlavička protokolu UDP, potom IP protokolu a nakoniec sa pridá Ethernet 2. Dáta ktoré sa budú posielať (správa alebo text) budú zapísane do bytového poľa a na základe indexu sa budú vyberať z poľa a postupne posielať v paketoch. Paket bude pozostávať z hlavičky, dát a CRC kontrolného kódu.

CRC:
CRC je spôsob detekcie chýb ktorý kontroluje náhodné zmeny nespracúvaných dát. Používa sa pri digitálnych sieťach a úložných zariadení. Na základe niektorých bytov sa vypočítava zabezpečovací údaj (použitím xor-u) ktorý sa porovná s kontrolnou hodnotou na konci data rámcu. Ak sa tieto dva údaje zhodujú, dá sa prenesený blok s vysokou pravdepodobnosťou predpokladať za správny.

3. Návrh programu:

3.1 Hlavné vlastnosti protokolu UDP z pohľadu zadania:

UDP je “nespoľahlivý” protokol. Prenáša datagramy medzi počítačmi v sieti ale na rozdiel od TCP nezaručuje, že sa prenášaní paket nestratí, že sa nezmení poradie paketov, ani že sa niektorý paket nedoručí viackrát.

V našom zadaní má byť schopný UDP komunikátor odosielať súbory a textové správy. Je teda možné, že príde k chybe v spojení alebo podobnej chybe, a preto nasledovná správa bude poslaná len keď prechádzajúca bude potvrdená.

3.2 Návrh vlastného protokolu:

Program bude napísaný v jazyku Python vo PyCharm 2019 vývojovom prostredí. Bude sa používať knižnica ktorá umožní prácu so socketom. Bude obsahovať triedy hlavicka, sender, reciever.

Pred každou správou používateľ bude mať možnosť zadať cieľovú adresu a port. Bude sa odosielať text\súbor, pričom každý uzol bude mať možnosť zobraziť správu alebo adresu súboru. Pri posielaní sa dáta rozloží do fragmentov podľa zadanej veľkosti fragmentu.

Program bude spĺňať základné požiadavky, a pri tom bude udržiavať konekciu s vysielaním resp. primaním keepalive packetov.

Pri spusteni programu si najprv vyberieme ci sme sender alebo reciver (1 alebo 2).

- Sender:

Používateľ si vyberie či chce poslať správu alebo súbor. Následne si vyberie veľkosť fragmentu a zadá cieľovú IP adresu. Po zadaní IP adresy trida vytvorí pripojenie na cieľový EndPoint (IP adresa a port). Celá správa bude zapísaná do poľa bajtov a s toho poľa sa bude vyberať časť správy po indexoch na základe zadanej dĺžky fragmentu. Keď pošle poslednú správu, počká na odpoveď od prijímacej strany a ukončí spojenie.

- Receiver:

Bude primať dáta postupne po fragmentoch. Akonáhle prímy fragment, skontroluje či je rámec správny, či dostál správne číslo fragmentu a či je CRC odpoveď správna. Potom data časť rámcu uloží do poľa v ktorom postupne skladá celú správu alebo súbor. Na konci pošle správu že je data prijaté.

- Fragment:

Hlavička					
Typ	Veľkosť fragmentu	Počet fragmentov	Čislo fragmentu	Checksum (CRC)	Data
1B	4B	4B	4B	4B	>1580B

Data rámec

Typ	Keep alive
1B	4B

Keepalive rámec

- Typ:

0	Data text
1	Meno súboru
2	Data súbor
3	KeepAlive

0 = Sender zasiela textovú správu

1 = Sender zasiela meno súboru recevierovi

2 = Sender zasiela súbor receiverovi

3 = KeepAlive

- KeepAlive:

Mechanizmus na udržovanie spojenia (keep-alive) bude znamenať, že vysielacia strana bude v intervale každých 5 sekúnd posielať tzv. keep-alive správu od poslednej doručenej správy. Keď 3x pošle keep-alive správu a nedostane do 20 sekúnd odpoveď, vysielacia strana to bude považovať, že nastala chyba v sieti, zruší spojenie a vypíše chybovú hlášku.

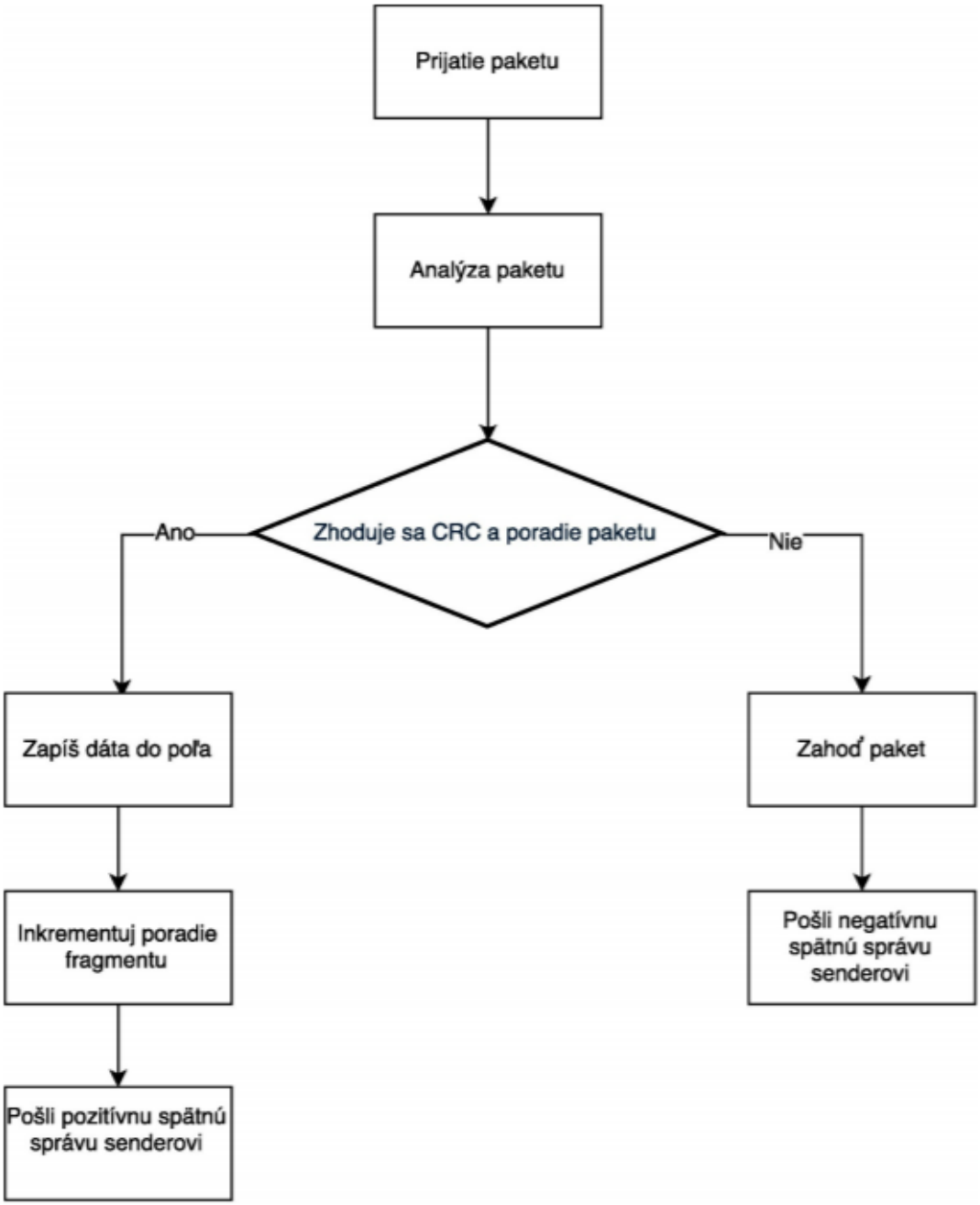
- Chybný fragment:

Na simulovanie chybných rámcov bude slúžiť špeciálna vetva programu, ktorá bude pri odosielaní zo strany klienta upravovať rámce po vypočítaní CRC, respektíve bude CRC v hlavičke generovať náhodne alebo upraví správny údaj na nesprávny (napríklad inkrementovaním CRC políčka v hlavičke a podobne). Týmto spôsobom sa zaručí, že odoslaný rámec nie je v poriadku a bude sa očakávať reakcia servera/prijímateľa v podobne odpovede typu „Nesprávne doručený rámec“.

- Používateľské rozhranie:

Program bude používať rozhranie príkazového riadku, pričom na počiatku sa zobrazí legenda všetkých príkazov, a po každej funkcii sa vypíše správa o tom čo sa práve vykonalo, alebo v prípade chybe, sa vypíše chybová hláška.

- Vývojový diagram:



4. Implementácia a zmeny oproti návrhu

Sender a Receiver fungujú rovnako ako to bolo napísané v návrhu.

Oproti návrhu, pridali sme Typ DATARESPONSE:

Receiver odpovedá Senderovi s indexom správi na ktorú čaká.

DATATEXT = 0

FILENAME = 1

DATAFILE = 2

DATARESPONSE = 3

KEEPALIVE = 4

Pridali sme timer TMT_WAIT_FOR_MSG na 2 sekundy.

Ak Receiver neodpovie senderovi s DATARESPONSE za tie 2 sekundy Sender znovu zašle fragment.

- Sender – na posielanie správ a súborov

Najprv zadáme ip adresu a port. Program skúsi vytvoriť socket, nastaví endpoint na IP adresu Receivera. Dalej si používateľ vyberie čo chce zaslať (1= správa, 2=súbor, alebo chceme skončiť 3= koniec).

Po výbere čo chce zaslať, program sa opýta na maximálnu veľkosť fragmentu. Ak zvolí číslo väčšie ako celková veľkosť správy alebo súboru tak sa veľkosť fragmentu nastaví na rovnakú hodnotu ako veľkosť správy alebo súboru. Taktiež zistí či sa posledný packet bude líšiť od ostatných. Ak pri delení celkovej veľkosti správy s veľkosťou fragmentu existuje zvyšok, to znamená že posledný packet bude trochu menší.

```
    # V ramci pola byteData zluci elemnty podla maxFrag size a zapise novy list do toSend
    # cize ak byteData=["Hello_world"] a maxFragSize = 4
    # toSend bude ["Hell", "o_w", "orl", "d"]
    toSend = list()  # list dat ktore sa zapisu do fragmentov
    for fragIdx in range(0, len(byteData), self.maxFragSize):
        fragIdx2 = fragIdx + self.maxFragSize
        if fragIdx2 < len(byteData):
            toSend.append(bytes(byteData[fragIdx:fragIdx2]))
        else:
            toSend.append(bytes(byteData[fragIdx:]))
            break
```

Nakoniec sa program opýta či používateľ chce poslať chybný rámec. Po zaslaní chybného fragmentu receiver si vypíta správny fragment, ktorý mu sender prepošle.

```
while fragIdx < fragCount:
    fragData = toSend[fragIdx]
    fragHeader.setFragIdx(fragIdx)
    fragHeader.setFragSize(len(fragData))
    fragment.setData(fragData)
    fragment.checksum = getCRC(fragment.data)
    if self.fragErr != 0 and self.fragErr == fragIdx:
        print("Posielam chybny fragment")
        fragment.checksum = 0
        self.fragErr = 0
    print("Posielam fragment:" + str(fragment))
    self.senderSocket.sendto(packFragment(fragment), (self.ipAddress, self.port))
    recMsg = None
```

- Receiver

Načíta IP adresu a port, a otvorí socket.

Ak bol zadaný Typ DATATEXT Receiver vie ze čaká správu, inak vypíše chybu.

Ak Receiver dostane Typ FILENAME, vypíše cestu kde sa má súbor s manom uložiť, a čaká na súbor.

```
while True:
    recMsg = ""
    recMsg, addr = recSocket.recvfrom(4096)
    if recMsg:
        fragType, fragment = unpackFragment(recMsg)
        if fragType == DATATEXT:
            if fragment.header.fragIdx == 0:
                expectedType = DATATEXT
            elif expectedType != DATATEXT:
                print("chyba, cakal som text")
                exit()
        if fragType == DATAFILE:
            if fragment.header.fragIdx == 0:
                expectedType = DATAFILE
            elif expectedType != DATAFILE:
                print("chyba, cakal som data file")
                exit()
        if fragType == FILENAME:
            if fragment.header.fragIdx == 0:
                expectedType = FILENAME
            elif expectedType != FILENAME:
                print("chyba, cakal som file name")
                exit()
```

Ak boli prijaté všetky fragmenty program vypíše správu.

```
# Ak boli prijate vsetky fragmenty
if fragment.header.fragIdx == fragment.header.fragCount - 1:
    if fragType == DATATEXT:
        # ak je sprava, vypisat
        expectedType = None # Dalej sa caka hocico
        print("Sprava bola prijata uspesne:")
        print(receiverBuffer.decode('utf-8'))
    if fragType == FILENAME:
        # Ak bolo prijate meno suboru, caka sa na data suboru
        expectedType = DATAFILE # Dalej sa caka data suboru
        filePath = os.path.join(defaultFilePath, receiverBuffer.decode('utf-8'))
        print("Cakam na subor, cesta:", filePath)
    if fragType == DATAFILE:
        # Prijate boli vsetky fragmenty suboru
        expectedType = None # Dalej sa caka hocico
        with open(filePath, "wb") as newFile:
            newFile.write(receiverBuffer)  # Zapis subor na cestu
    print("====================")
```

- CRC

```
def getCRC(byteData):
    return zlib.crc32(byteData) & 0xffffffff
```

5. Literatúra:

1. Appendix E Python - Sockets Examples

https://sandilands.info/sgordon/teaching/netlab/its332ap5.html

2. Charles R. Severance - *Python for Everybody*

http://do1.dr-chuck.com/pythonlearn/EN_us/pythonlearn.pdf