Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Oliver Leontiev

Zadanie 2: Komunikácia s využitím UDP protokolu

Predmet: Počítačové a komunikačné siete
Cvičiaci: Ing. Kristián Košťál, PhD.

ZS 2020/2021

Zadanie úlohy

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 prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 10-60s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

Automatic Repeat Request (ARQ) metóda

ARQ metóda umožňuje spoľahlivé spojenie medzi uzlami, tým že chybné a stratené fragmenty sa posielajú znova. Implementujeme ARQ metódu **stop-and-wait**.

Každý odoslaný dátový fragment musí byť **potvrdení prijímateľom** predtým, než sa pošle ďalší. Ak potvrdenie nepríde (**timeout**) alebo príde **negatívne** (fragment bol prijatý s chybou), tak sa pošle dátový fragment ešte raz.

Prijímateľ potvrdzuje každý dátový fragment. Pokiaľ mu ešte neprišiel posledný fragment komunikácie a neprišiel mu ďalší (**timeout**), tak pošle znova potvrdenie na predošlý fragment.

Timeout nastane po **1 sekunde**. Tento čas sa resetne vždy keď uzol príjme fragment / potvrdenie a beží len keď uzol čaká na fragment / potvrdenie.

Rozlišovanie duplikátov

Na rozlišovanie duplicitných fragmentov a potvrdení, sa využíva **sequence number** v hlavičke protokolu. Toto číslo označuje počet doteraz prijatých bajtov. Pri potvrdzovaní prijímateľ posiela sequence number, ktoré očakáva ďalšie. Vďaka tomu vie uzol, či mu prišiel nový fragment, alebo duplikát, ktorý bol odoslaný z dôvodu problému v komunikácii.

Prijímateľ posiela potvrdenia (aj negatívne aj pozitívne) vždy s hodnotou, ktorú očakáva ďalšiu. Ak je potvrdenie pozitívne, tak je to nasledovná hodnota, ak je potvrdenie negatívne, tak stále tá istá, keďže fragment prišiel chybný. Pri naviazaní komunikácie mu odosielateľ pošle namiesto sequence number veľkosť fragmentu. Odosielateľ nastavuje sequence number vždy podľa toho, koľko bajtov dát sa zatiaľ prenieslo.

Hlavička protokolu

Protokol bude mať **7 bajtovú hlavičku**, ktorá sa skladá zo **signalizačného bajtu (flag), sequence number** (4 bajty) a **CRC-16** (2 bajty).

7 bajtov:

Flag	Sequence number	CRC

Signalizačný bajt (FLAG):

Signalizuje druhému uzlu stav komunikácie. Ako signály posielame jednobajtové ASCII znaky.

Signály:

"**M**" - MESSAGE - začiatok komunikácie / nadviazanie spojenia – uzol pošle **správu**

"**L**" - FI**L**E - začiatok komunikácie / nadviazanie spojenia – uzol pošle **súbor** (najprv názov)

"I" - F**I**LE - Fragment obsahuje názov súboru

"P" - PUSH – fragment obsahuje dáta

"F" - FINISH - koniec komunikácie - posledný fragment **správy/súboru**

"a" - ack -potvrdenie prijatia fragmentu **názvu súboru** bez chyby (úspešný CRC check)

"**A**" - ACK - potvrdenie prijatia fragmentu bez chyby (úspešný CRC check)

"**N**" - NACK - prijatý fragment bol chybný (neúspešný CRC check)

"K" - keepalive - udržiava spojenie v čase medzi komunikáciami

"B" – BYE – oznamuje koniec spojenia

Sequence number:

Obsahuje informáciu o prenesenom počte bytov v rámci komunikácie. Umožňuje rozpoznávať duplicitné fragmenty a potvrdenia. Prijímateľ si pomocou sequence number vyžiadava fragmenty. Na začiatku pošle odosielateľ v tomto poli veľkosť jedného fragmentu.

CRC:

CRC hodnota vytvorená metódou **CRC-16**, ktorá slúži na overenie správnosti prenosu fragmentu bez poškodenia.

Cyclic redundancy check (CRC) metóda

Na kontrolu správnosti doručených dát použijeme metódu **CRC-16**.

polynóm: $x^{16} + x^{15} + x^2 + 1$ – 0x8005 - **11000000000000101**

výplň: 0000000000000000

Xorujeme od najvyššieho bitu a na konci neinvertujeme zvyšok.

Postup:

výpočet zvyšku:

1. K dátam (budú tvoriť ľavú časť) v binárnej podobe pripojíme výplň (16 núl - pravá časť)

2. Xorujeme s polynómom od najvyššieho bitu (na ktorom je 1).

3. Ak je ľavá časť vynulovaná, končíme a v pravej časti máme zvyšok (CRC hodnotu). Inak ideme na krok 2.

kontrola správnosti:

1. K dátam (budú tvoriť ľavú časť) v binárnej podobe pripojíme výplň (CRC hodnota - pravá časť)

2. Xorujeme s polynómom od najvyššieho bitu (na ktorom je 1).

3. Ak je ľavá časť vynulovaná, končíme a v pravej časti máme zvyšok. Inak ideme na krok 2.

Ak je zvyšok po kontrole 0, tak dáta neobsahujú chybu.

Ukážka:

Ako dáta použijeme binárny kód znaku **'a' - 01100001**

výpočet zvyšku:

```
01100001 0000000000000000
 1100000 0000000101
00000001 0000000101000000
       1 1000000000000101
00000000 1000000101000101
```

pošleme dáta + zvyšok - 01100001 1000000101000101

kontrola správnosti:

```
01100001 1000000101000101
 1100000 0000000101
00000001 1000000000000101
       1 1000000000000101
00000000 0000000000000000
```

Dostali sme zvyšok 0 - dáta sú **správne**.

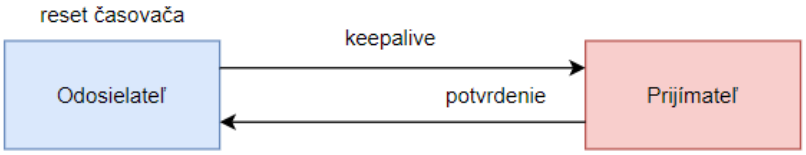
Metóda pre udržanie spojenia (keepalive)

Na udržiavanie spojenia sa budú používať správy bez dát a s **flagom "K"** v hlavičke protokolu a potvrdenia týchto správ ("A" / "N"). Sequence number bude vždy 0xff-ff-ff-ff, aby sa nemiešalo s dátami.

Pri nečinnosti (neprebieha komunikácia) beží na každom uzle **10 sekundový** keepalive časovač, ktorý sa spustí po dokončení komunikácie - prijatí posledných dát na prijímacom uzle a prijatí posledného potvrdenia na odosielacom uzle.

Odosielací uzol pošle **prvú keepalive** správu **hneď po prijatí posledného potvrdenia** na konci komunikácie. Vtedy spustí aj svoj keepalive časovač. Následne očakáva potvrdenie od prijímacieho uzla.

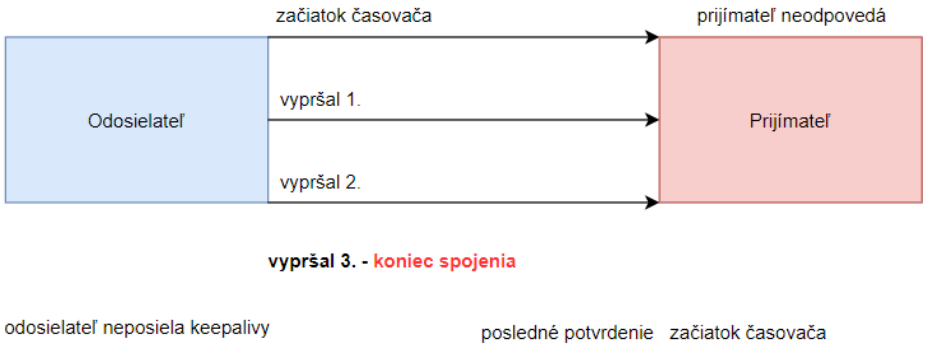
Prijímací uzol resetne svoj časovač a pošle potvrdenie odosielateľovi.

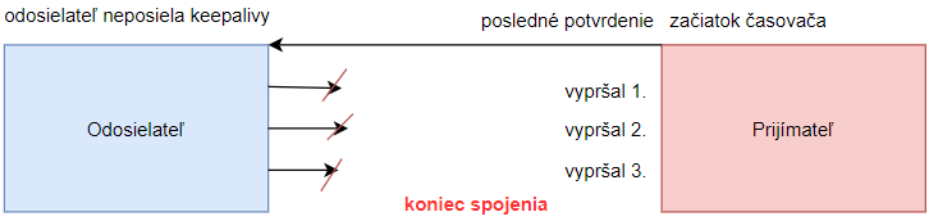


Odosielateľ prijme potvrdenie a resetne svoj časovač. Následne čaká na vypršanie časovača, kým pošle nový keepalive.

Pokiaľ prijímací uzol príjme chybný fragment, tak vyžaduje jeho znovuvyslanie (pomocou negatívneho potvrdenia), ale svoj keepalive časovač resetne aj tak. Pokiaľ odosielateľ príjme negatívne potvrdenie, tak hneď zopakuje predošlý keepalive - časovač resetne iba pri legitímnom potvrdení.

Spojenie je **ukončené** pokiaľ **odosielateľ 3-krát nedostane odpoveď** (3-krát mu vyprší časovač) alebo **prijímateľovi 3-krát vyprší časovač.**





Pokiaľ jedna strana vie o tom, že preruší spojenie (je vypnutá v rámci programu a nie vypnutím programu či prístroja), tak pošle paket s flagom **"B"** a druhá strana tiež spojenie preruší. Tento paket nie je nijako opätovaný a strana, ktorá ho posiela nečaká na potvrdenie. Pokiaľ sa pokazí alebo stratí, tak ukončenie nastane až po prekonaní vyššie uvedených podmienok.

Diagramy spracovávania komunikácie

Modrý obdĺžnik predstavuje hlavičku poslanú **odosielateľom** (v resp. prijatú prijímateľom)

Červený obdĺžnik predstavuje hlavičku poslanú **prijímateľom** (v resp. prijatú odosielateľom)

SQ – Sequence number, SQ+ – Nasledujúce sequence number

Diagram odosielateľa:

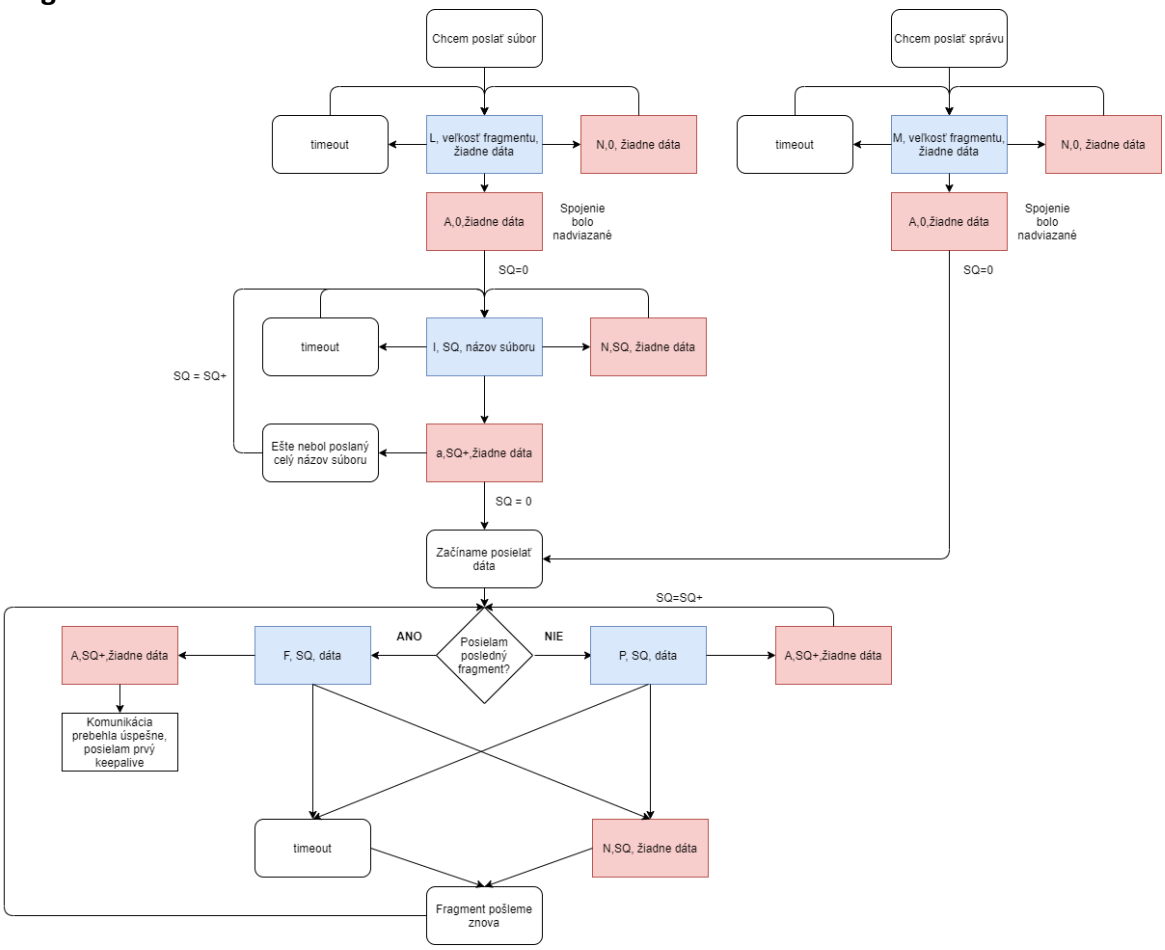
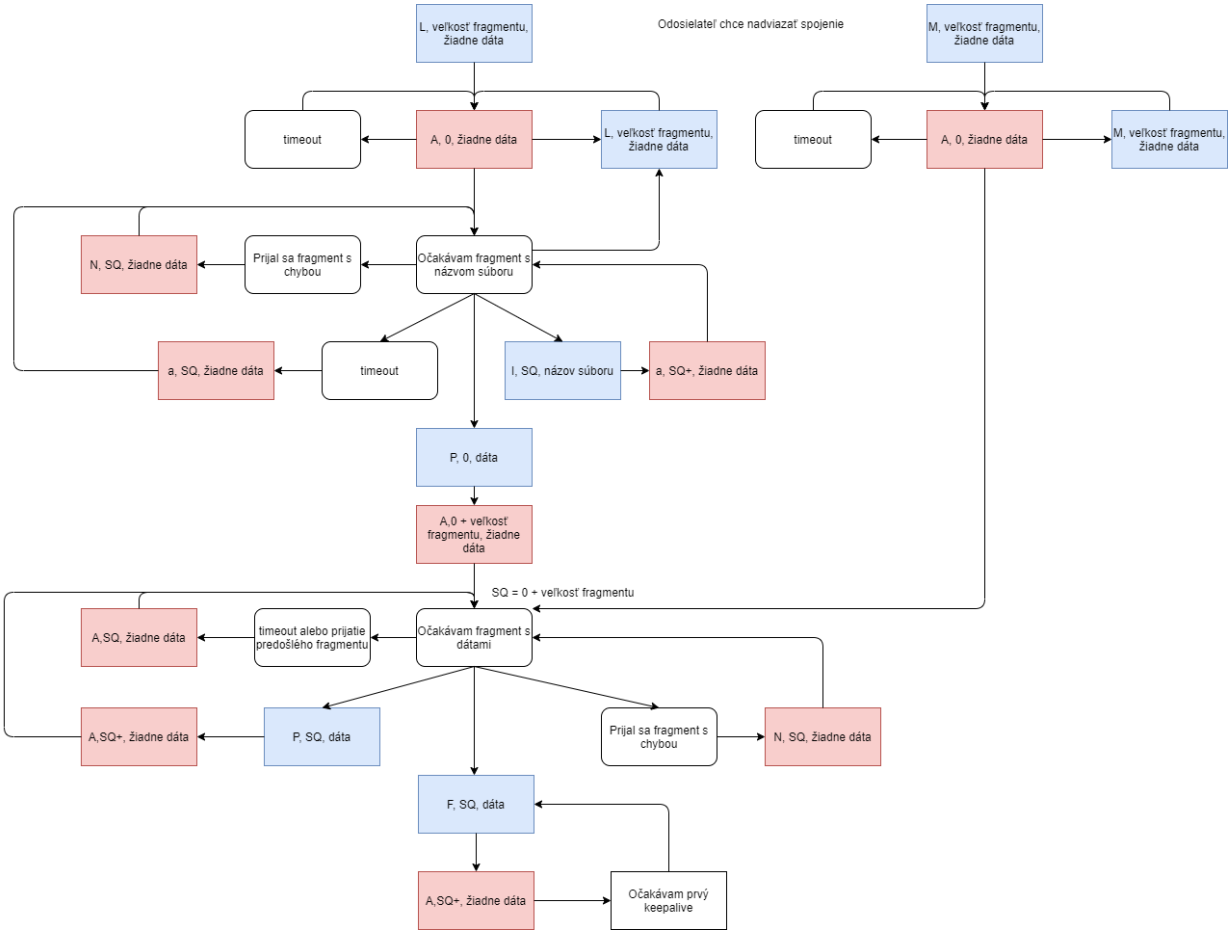


Diagram prijímateľa

Očakávané SQ sa inkrementuje po úspešnom prijatí fragmentu.



Veľkosť fragmentov

Minimálna veľkosť dátovej časti Ethernet II rámca je 46 bajtov. Pokiaľ je dát menej, tak sú automaticky vyplnené na linkovej vrstve. Tým pádom **minimálnu veľkosť nemáme** / neriešime.

Chceme zabrániť fragmentácii na linkovej vrstve. Maximálna veľkosť dátovej časti Ethernet II rámca je 1500 bajtov. Naše fragmenty obsahujú IP hlavičku (20B), UDP hlavičku (8B) a našu hlavičku (7B). **Maximálna veľkosť fragmentu**, ktorú môže používateľ zadať je 1500 – 20 – 8 – 7 = **1465 bajtov**.

Informáciu o požadovanej veľkosti fragmentu zadáva používateľ na odosielacom uzle. Ten následne dáta rozdelí na fragmenty a postupne posiela prijímateľovi. Pokiaľ je dát menej ako veľkosť fragmentu (pri poslednom alebo prvom fragmente), tak pošle menší fragment - nevypĺňa ho.

Úmyselné zavedenie chyby

Chyba v dátach

Na vstupe zadáme, ktorý fragment má obsahovať chybu. Na konci komunikácie odosielací uzol vypíše, ktorý fragment musel poslať 2-krát a prijímací uzol vypíše v ktorom fragmente detegoval chybu.

Chyba predstavuje **vynulovanie 3 náhodných bajtov** fragmentu. Ak je niektorý s vybraných bajtov 0x00, tak je zmenený na 0xff.

Zmeny oproti návrhu

1. Namiesto alternujúceho bajtu sme implementovali sequence number, ktorý nám umožňuje jednoduchšie sledovať správnu postupnosť fragmentov a taktiež vytvára miesto na poslanie maximálnej veľkosti fragmentu na začiatku komunikácie. Tým sa zväčšila veľkosť hlavičky na 7 bajtov.

2. Pribudli nové flagy. “**P”** a “**I”** umožňujú prakticky rozlíšiť, kedy posiela uzol dáta a kedy názov súboru. Na potvrdenie názvu súboru používame špeciálny flag **“a”**. Keďže prvý fragment názvu súboru a prvý fragment dát majú to isté sequence number (0), tak ak by sa stratil prvý fragment dát a prijímateľ si vypýtal druhý fragment názvu súboru znova (lebo nevie, že už mu boli posielané dáta), tak by poslal potvrdenie, ktoré vyzerá ako potvrdenie prijatia prvých dát (sequence number = veľkosť fragmentu). Ak sa stratí prvý fragment dát a odosielateľ prijme potvrdenie s “a” flagom, tak vie že prijímateľ znova potvrdzuje názov súboru a nie prvé dáta, a teda ich neprijal.

3. Keepalive časovač sme nakoniec znížili na 10 sekúnd, a počet neprijatých keepalivov pred ukončením spojenia je rovnaký na oboch stranách (3). Odosielateľ posiela keepalivy vždy, keď príjme nejaký paket, ale svoj časovač neresetne, kým nedostane korektné potvrdenie. Pridali sme odoslanie paketu s flagom **“B”** v prípadoch, že jedna strana vie o tom, že sa odpája. Nemusíme čakať na vypršanie keepalivov.

4. Pri simulovaní chyby sme nakoniec strácanie fragmentov vypustili.

Implementácia

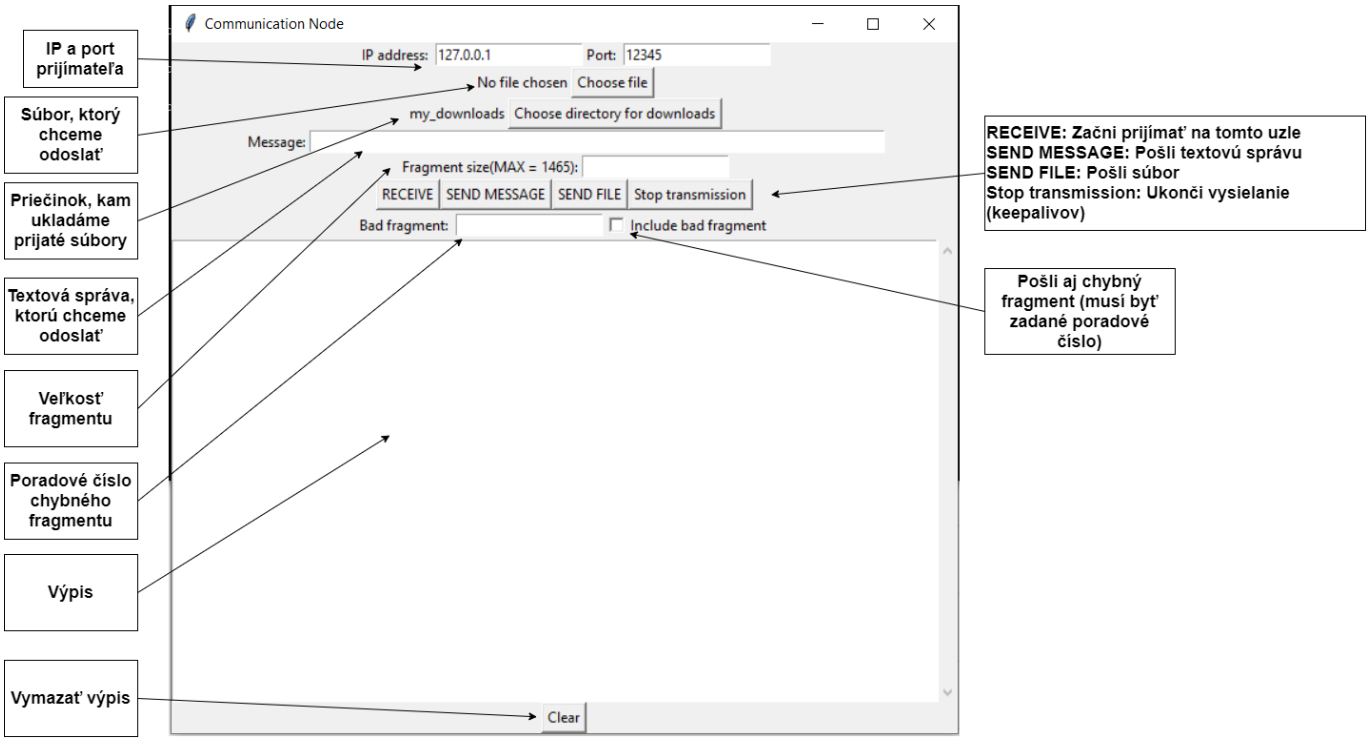
Program je v jazyku Python 3 a spúšťa sa v konzole cez súbor *commnode.py* a to štandardným spôsobom:

```
$ py commnode.py
```

Sockety sú v niektorých prípadoch neblokujúce (timeout = 0): prijímateľ pred začiatkom komunikácie alebo obidva počas keepalive fázy. Je to preto aby sme mohli program bez problému ukončiť a znova zapnúť bez toho aby sme čakali na ukončenie threadov.

Ovládanie GUI

IP adresa a port musia byť zadané na obidvoch stranách a musia byť rovnaké (prijímacieho uzlu)



Dôležité časti programu

Status komunikátorov

Oba komunikátory majú **vlastnú premennú status (1-5)**, podľa ktorej vedia, v ktorej fáze komunikácie sa nachádzajú:

Status 0 - Komunikátor je neaktívny

Status 1 - Čakám na začatie komunikácie

Status 2 – Prebieha prenos textovej správy

Status 3 – Prebieha prenos názvu súboru

Status 4 – Prebieha prenos súboru

Status 5 – Prebieha posielanie keepalive správ

Triedy:

Trieda ***CommNode***

Súbor: CommNode.py

Čo robí: Vykresľuje GUI pre komunikátor.

Trieda ***Receiver***

Súbor: Receiver.py

Argumenty: IP adresa, port, výstup a adresár na ukladanie súborov

Čo robí: Obsahuje funkcionalitu prijímania a spracúvania prijatých dát. Otvorí socket pre prijímanie dát.

Trieda ***Sender***

Súbor: Sender.py

Argumenty: IP adresa, port, výstup, dáta, veľkosť fragmentu a poradové číslo chybného fragmentu

Čo robí: Obsahuje funkcionalitu posielania dát a spracúvania prijatých potvrdení.

Funkcie:

receive() - Receiver.py

```
def receive(self):
    while self.status > 0:
        try:
            data, addr = self.sock.recvfrom(4096)
            self.respond(data, addr)
            if self.status == 5:
                self.keepalive_phase()
        except socket.error:
            if 2 <= self.status <= 4:   # Resend ACK if timeout
                self.resend_ack()
            continue
```

Prijme dáta a rozhodne sa, ako na ne odpovie – z funkcie respond() volá handler funkcie. Pokiaľ vypršal timeout pošle znova potvrdenie na predošlý fragment (ak už začala komunikácia).

keepalive_phase() - Receiver.py

```python
# Await keepalives from sender, if 3 timers expire then close the connection
def keepalive_phase(self):
    ka_limit = 10
    self.sock.setblocking(False)
    waiting = True
    elapsed_timers = 0
    start = time.time()
    while self.status == 5 and elapsed_timers < 3:
        try:
            if waiting:
                if time.time() - start > ka_limit:
                    elapsed_timers += 1
                    start = time.time()
            data, addr = self.sock.recvfrom(4096)
            self.respond(data, addr)
            elapsed_timers = 0
            start = time.time()
        except socket.error:
            continue
    if self.status == 5:
        self.display_log("Connection ended: no keepalive")
        self.status = 1
    elif self.status == 0:
        self.sock.sendto(prt.create_header('B', MAX_SEQ_NUM, None), self.sender_addr)
```

Očakáva keepalive správy od odosielaceho uzla v 10 sekundových intervaloch. Pokiaľ vypršia 3 časovače, tak preruší spojenie a čaká na začatie komunikácie (status 1). Pokiaľ je uzol vypnutý z GUI (statu 0) tak vyšle ukončovací paket (“B”).

send_data() - Sender.py

```python
# Send data fragments in order and wait for response (socket blocks)
def send_data(self, data):
    fragment = next(data)
    frag_num = 1
    while True:
        try:
            self.seq_num = int.from_bytes(fragment[1:5], byteorder="big")
            if self.status != 3:
                self.display_log(f'{frag_num}. Sending data fragment ({len(fragment) - 7} bytes)')
            if frag_num == self.bad_frag and (self.status == 2 or self.status == 4):    # Send corrupted fragment
                self.sock.sendto(prt.make_mistake(fragment), self.receiver)
                self.bad_frag = 0
            else:
                self.sock.sendto(fragment, self.receiver)
            response, addr = self.sock.recvfrom(4096)
            if self.check_response(response):
                fragment = next(data)
                frag_num += 1
            else:
                continue
        except StopIteration:
            break
        except socket.timeout:
            continue
        except ConnectionResetError:
            self.display_log('Receiver is offline!')
            self.status = 0
            break
```

Postupne posiela dátové fragmenty, ktoré už sú pripravené v liste a majú pridané hlavičky. Pokiaľ je číslo fragmentu rovnaké ako zadané číslo chybného fragmentu, tak zavolá zanesenie chyby do fragmentu predtým, ako ho pošle. Čaká na odpoveď a pokiaľ je nevhodná alebo vyprší timeout, tak pošle fragment ešte raz.

keepalive_phase() - Sender.py

```
# Await keepalives from sender, if 3 timers expire then close the connection
def keepalive_phase(self):
    ka_limit = 10
    self.sock.setblocking(False)
    waiting = True
    elapsed_timers = 0
    start = time.time()
    while self.status == 5 and elapsed_timers < 3:
        try:
            if waiting:
                if time.time() - start > ka_limit:
                    elapsed_timers += 1
                    start = time.time()
            data, addr = self.sock.recvfrom(4096)
            self.respond(data, addr)
            elapsed_timers = 0
            start = time.time()
        except socket.error:
            continue
    if self.status == 5:
        self.display_log("Connection ended: no keepalive")
        self.status = 1
    elif self.status == 0:
        self.sock.sendto(prt.create_header('B', MAX_SEQ_NUM, None), self.sender_addr)
```

Vysiela keepalive správy v 10 sekundových intervaloch. Pokiaľ 3-krát nedostane korektnú odpoveď, tak preruší spojenie a čaká na začatie komunikácie (status 1). Pokiaľ príjme neočakávaný paket, zopakuje keepalive okamžite. Pokiaľ je uzol vypnutý z GUI (status 0) tak vyšle ukončovací paket (“B”).