POČÍTAČOVÉ A KOMUNIKAČNÉ SIETE

Dokumentácia k prvému zadaniu

Komunikácia s využitím UDP protokolu

Autor: Adam Šípka
Dátum: 24.10. 2019
Semester: ZS 2019/2020
Čas cvičenia: Piatok 8:00 – 9:50
Cvičiaci: Ing. Peter Kaňuch

Zadanie:

- Navrhnúť a implementovať program s použitím vlastného protokolu nad protokolom UDP transportnej vrstvy sieťového modelu TCP/IP.
- Program umožní prenos textových správ a ľubovoľného binárneho súboru medzi dvoma počítačmi.
- Program bude pozostávať z vysielacej a prijímacej časti. Vysielací uzol pošle súbor inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát.
- Používateľ musí mať možnosť zadať maximálnu veľkosť fragmentu. Ak je posielaný súbor väčší, rozloží sa na jednotlivé fragmenty, ktoré sa pošlú samostatne.
- Po prijatí fragmentu prijímateľ vypíše správu s poradím a či bol prenesený bez chýb.
- Ak dôjde k chybe, tak musí byť možné znovuvyžiadanie chybných fragmentov.
- Pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 20-60s pokiaľ používateľ neukončí spojenie.

UDP/TCP analýza:

UDP (user datagram protocol) je protokol, ktorý operuje v sieťovej vrstve a prenáša datagramy po sieti. Na rozdiel od TCP protokolu nie je potrebné pri nadviazaní spojenia predviesť 3-way handshake. UDP je vhodný pre aplikácie, ktoré si nemôžu dovoliť oneskorenie pri opakovanom prenose chybných paketov a ktoré nevyžadujú overenie alebo opravu údajov (online hry, streamovanie, a podobne). Jednou z výhod UDP je jeho vysoká rýchlosť, keďže neberie veľký ohľad na to, aby sa prepravované datagramy dostali na miesto určenia, čím šetrí čas, takže je aj vhodný pre jednoduché a časovo citlivé účely. Vďaka tomu, že UDP nevyužíva end-to-end spojenie, je vhodný pre vysielanie, pri ktorom sú prenášané datagramy adresované ako prijaté pre všetky zariadenia, ktoré sú v danej sieti pripojené. Zároveň malé množstvo požiadavok na pripojenie a overovania dát môže pri prenose spôsobiť množstvo problémov. Nie je garantované, že odoslané pakety prídu v správnom poradí a taktiež nemáme overené, či je počítač, ktorému dáta posielame pripravený na ich prijatie. Okrem toho môže dôjsť k chybe a nie je isté, či príjemcovi budú všetky odoslané pakety aj doručené. Avšak UDP header poskytuje checksum, aby mohol overiť, či prijatý paket neobsahuje nejaké chyby.

Okrem poľa pre checksum je header tvorený z 3 ďalších polí, každé z nich má veľkosť 2 byty (alebo 16 bitov). Pole zdrojového portu, z ktorého sa dáta odosielajú, môže byť nastavené na nulu, ak od príjemcu nepotrebujeme žiadnu odpoveď. Nasledujúce pole je určené pre číslo cieľového portu. V ďalšom je uložená dĺžka headeru a dát daného paketu. Minimálna dĺžka je 8 bytov (minimálna dĺžka headeru), maximálna je 65 535 bytov, avšak ak prenášame dáta cez Ethernet, tak maximálna veľkosť rámca je 1526 bytov (1500 bytov určených pre dáta, 26 pre Ethernet header). Posledné pole je určené pre už vyššie spomenutý checksum (pri IPv4 je voliteľné, pri IPv6 povinné).

TCP (transmission control protocol) je spoľahlivý komunikačný protokol, ktorý tiež operuje v transportnej vrstve. Za spoľahlivý sa považuje preto, lebo garantuje, že odoslané dáta sa dostanú k príjemcovi a dorazia k nemu v takom poradí, v akom boli odoslané. Pri využití TCP nedochádza k strateniu paketov, ak náhodou paket nedorazí do cieľa, tak odosielateľ ho pošle znova. Avšak TCP protokol je pomalší ako UDP. Je to spojovo orientovaný protokol, to znamená, že pred tým ako dôjde k presunu dát, zariadenia by mali medzi sebou nadviazať spojenie (tzv. three-way handshake). V prvom kroku chce klient vytvoriť spojenie so serverom a pošle mu SYN (Synchronize Sequence Number), čo je náhodne zvolené poradové číslo, ktoré informuje server o tom, že by klient chcel začať komunikáciu. Server odošle naspäť SYN a ACK (Acknowledgement). SYN je náhodne poradové číslo pre server, ACK je prijaté SYN od klienta a zväčšené o jedna.
V poslednom, treťom kroku klient odošle serveru ACK zväčšené o jedna (SYN servera). Ak 3-way handshake prebehne úspešne, až potom je spojenie nadviazané a zostane tak až pokiaľ nedôjde k ukončeniu spojenia.

Pôvodný návrh môjho protokolu:

Na začiatku programu si užívateľ vyberie, či chce mať funkciu servera, alebo klienta. Po úspešnom poslaní dát bude možné odhlásiť sa zo svojej funkcie a prehodiť ju na druhú. Samozrejme ak si prvý užívateľ vyberie funkciu odosielateľa, tak druhému ostáva už len funkcia príjemcu. Pre nadviazanie spojenia medzi klientom a serverom, musí klient zadať IP adresu servera a číslo portu na ktorý sa chce pripojiť. Potom server odošle klientovi správu v ktorej hovorí, že pripojenie bolo úspešné a presun sa môže začať. Týmto sa zaistí, že obe strany sú aktívne, klient nebude odosielať dáta do prázdna a server nebude donekonečna čakať na začatie prenosu. Serveru sa taktiež zobrazí správa, ktorá hovorí z akej adresy sa klient pripojil. Keď bude toto spojenie nadviazané, tak klient môže začať odosielať správy a súbory. Bude možné vybrať si z rôznych typov súborov, ktoré chceme na server poslať.

Pred tým, ako sa začne samotný presun dát, tak klient najprv odošle úvodný paket, ktorý hovorí o tom, aký typ súboru posielame, aká je jeho veľkosť a koľko paketov má prísť. Týmto spôsobom ušetríme niekoľko bytov pri posielaní súborov na ktoré bude potrebných veľké množstvo paketov, lebo ich header bude môcť mať menšiu veľkosť, keďže informácie, ktoré by boli vo všetkých paketoch rovnaké sa pošlú len raz, v úvodnom pakete. Až potom sa začne transport samotných dát. Každý paket obsahuje header v ktorom je uložené jeho poradie, veľkosť a checksum. Ten získame tak, že celkovú veľkosť dát v posielanom pakete vydelíme nejakým číslom a zvyšok po delení zapíšeme do checksumu. Potom, keď bude paket doručený, tak tiež vydelíme jeho veľkosť rovnakým číslom a zvyšok po delení porovnáme s hodnotou, ktorá je zapísaná na mieste checksumu. Toto zvyšuje šancu, že doručené dáta dorazili nepoškodené.

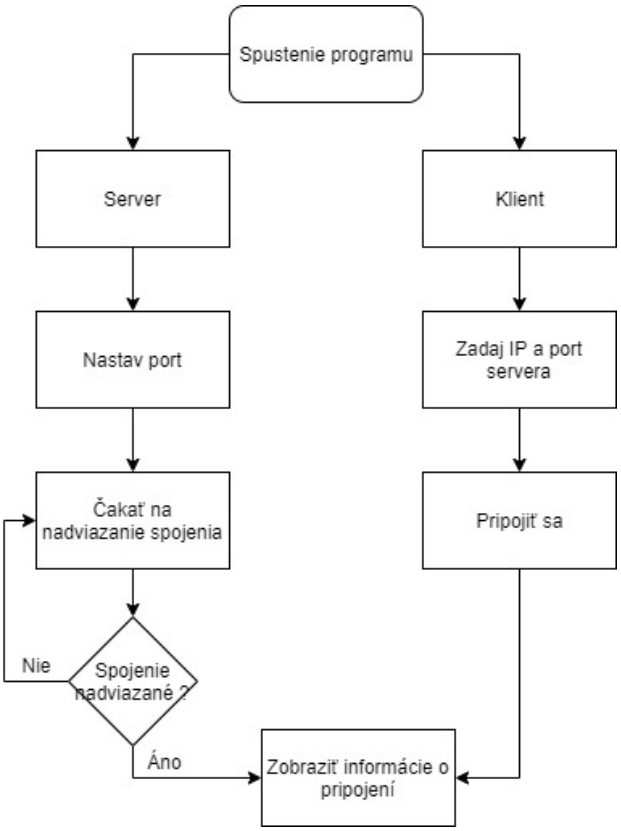
Treba kontrolovať, či každý paket dorazil do cieľa. Výhodnejšie je posielať pakety v skupinách a keď bude celá skupina odoslaná, tak až potom prebehne kontrola, či boli všetky doručené. Server o tom odošle informáciu klientovi, ktorý bude podľa nej ďalej postupovať. Ak sa všetky dáta dostanú bezproblémovo do cieľa, bude sa môcť odoslať ďalšia skupina paketov.

Ak nejaký paket nedorazí do cieľa, alebo dorazí poškodený tak server o tom odošle správu klientovi a môže si znova vyžiadať zaslať chýbajúci paket, ktorý bude následne odoslaný.
V poslednom pakete, ktorý sa odošle, až keď všetky dáta boli poslané sa nachádza meno súboru, aby sme ho mohli uložiť. Ak neposielame súbor, ale iba textovú správu, tak posledný paket bude prázdny.

Ovládanie programu bude jednoduché na používanie a realizované pomocou jednopísmenových príkazov v konzole (t – poslať textovú správu, s – poslať súbor, k - koniec a pod.), ktoré budú zobrazené priamo počas chodu programu. Na implementáciu programu som sa rozhodol použiť Python, keďže má väčšie množstvo funkcii, vďaka ktorým nebudem musieť robiť niektoré veci manuálne (napr. prevod medzi dátovými typmi).

Vysvetlenie chodu programu

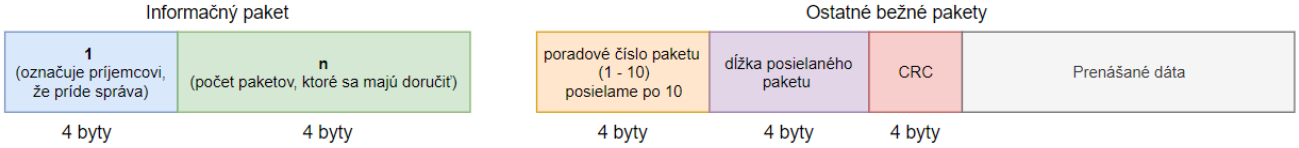
Po zapnutí programu sa spustí funkcia, v ktorej si používateľ vyberie, či sa chce prihlásiť ako odosielateľ, alebo príjemca. Program je schopný fungovať oboma spôsobmi a meniť typy užívateľov bez prerušenia spojenia, avšak nedokáže odosielať a prijímať v rovnakom čase. Povedzme, že na PC1 si vyberieme funkciu príjemcu. V tom prípade nastavíme port, na ktorý chceme prijímať dáta. Potom čakáme, kým sa nepripojí aj odosielateľ z PC2. Ten musí zadať IP adresu príjemcu a aj port, ktorý nastavil. Až vtedy dôjde k úspešnému nadviazaniu spojenia. Obom používateľom sa zobrazí správa o úspešnom pripojení. Na PC1 sa zobrazí správa, ktorá hovorí z akej IP adresy a portu sa odosielateľ pripojil. Jemu sa zas zobrazí informácia, na akú IP a port je napojený.



Odosielanie dát

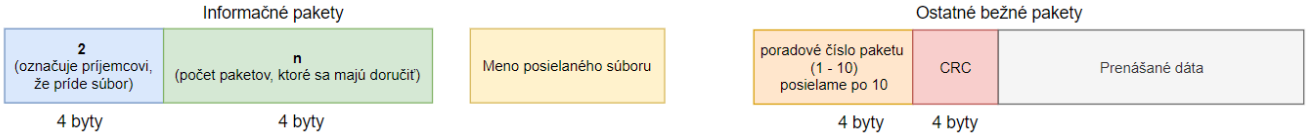
Ak sme nastavený ako odosielateľ, tak máme k dispozícii viacero možností. Môžeme sa odhlásiť, poslať správu, alebo súbor a zapnúť/vypnúť keep alive. Pri odhlásení sa nám zobrazí nový výber možností, znova si tam môžeme vybrať užívateľa, alebo vypnúť program. Keby sme sa chceli prihlásiť ako príjemca, tak už nemusíme zadávať port, keďže spojenie bolo už nadviazané. Rovnako, keď sa chceme prihlásiť ako odosielateľ, tiež už nie je potrebné zadávať IP a port.

Pri posielaní správy najprv zadáme jej samotný text a veľkosť fragmentu. Potom máme možnosť pridať chybu, ktorá úmyselne poškodí jeden z prvej skupiny posielaných paketov. Paket je z nej vybraný náhodne. Poškodenie simulujem tak, že zmením checksum daného paketu. Potom odošleme informačný paket, ktorý obsahuje informáciu o tom, že príde správa zložená z ***n*** paketov. Prvý znak informuje o tom, aký typ dát príde, a zvyšok je počet paketov, ktoré sa majú doručiť. Tieto informácie je lepšie posielať samostatne, pred začatím presunu dát, lebo by bolo zbytočné posielať ich v každom pakete. Vďaka tomuto je aj hlavička pri odosielaní paketov menšia.

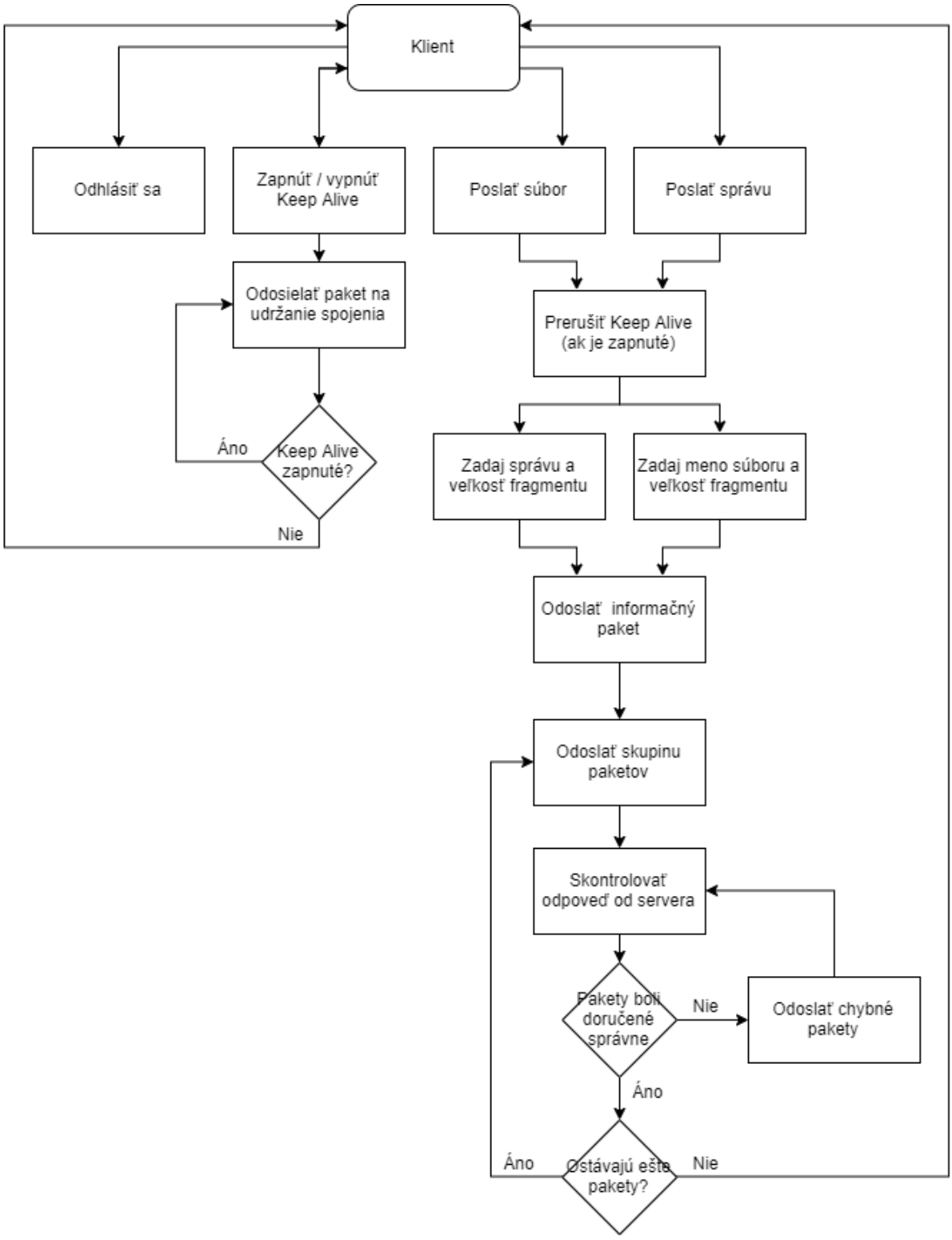


Potom sa môžeme presunúť na samotný transport dát. Oddelíme z napísanej správy fragment veľkosti nami vyššie zadanej a vytvoríme k nemu hlavičku. Vtedy je už paket pripravený na odoslanie. K dátam pripojíme vytvorenú hlavičku, a okrem toho, že paket odošleme, tak ho aj pridáme do poľa, ktoré nám bude neskôr slúžiť na kontrolu spätnej väzby. Toto všetko vykonáme pre každý z desiatich paketov, lebo posielame po desiatich. Ak by bolo náhodou paketov v skupine menej ako desať, tak odošleme prázdny paket, vďaka ktorému príjemca vie, že všetky dáta boli už odoslané. Keď už bude celá skupina paketov odoslaná, tak čakáme na spätnú väzbu od príjemcu. Tá môže buď obsahovať informáciu o tom, že všetky pakety boli doručené, alebo obsahuje poradové číslo poškodených, či nedoručených paketov. Ak boli všetky doručené v poriadku, tak sa zobrazí správa, že presun prebehol úspešne. Ak nie, tak ich odosielateľ odošle znova a čaká na ďalšiu odpoveď od príjemcu. Pred tým, ako sa začne transport ďalšej skupiny, tak sa vynuluje pole, ktoré obsahovalo pakety z predchádzajúcej skupiny. Tento cyklus opakujeme pre všetky skupiny paketov, až pokiaľ máme ešte dáta na odoslanie, vtedy sa pošle prázdny paket, ktorý ukončuje presun.

Odosielanie súborov je takmer totožné s odosielaním správ. Najprv zadáme meno súboru, potom veľkosť fragmentu a tiež máme možnosť pridať chybu do jednej skupiny balíkov. Jediný rozdiel je v posielaní prvého informačného paketu, za ktorým nasleduje ďalší, ktorý obsahuje meno posielaného súboru. Až potom sa začne presun samotných dát. Tam hlavička obsahuje len poradové číslo a CRC. Pakety sa taktiež posielajú po desiatich a transport sa tiež ukončuje prázdnym paketom.



Po odoslaní súboru, alebo správy sa vrátime do menu pre odosielateľa, kde máme znova prístup k vyššie spomenutým možnostiam. Okrem posielania mám v programe ešte implementované keep alive, ktoré zabraňuje tomu, aby sa spojenie so serverom prerušilo.



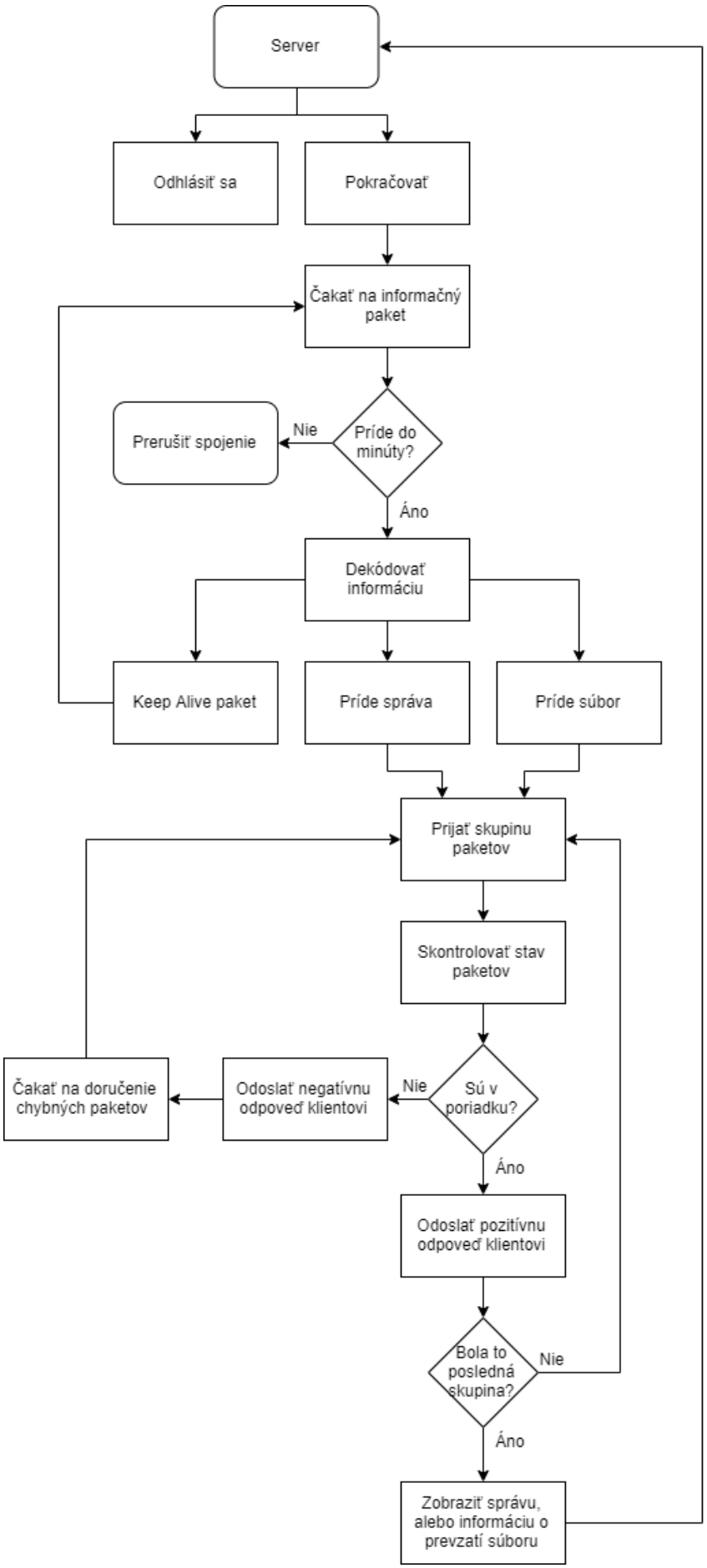
Keep alive máme možnosť spustiť kedykoľvek, keď sa nachádzame v menu odosielateľa. Ak chceme poslať ďalšiu správu, alebo súbor, tak keep alive sa automaticky preruší, pokiaľ tak neurobí užívateľ ešte predtým. Je realizovaný pomocou threadu, ktorý v päťsekundových intervaloch odosiela príjemcovi keep alive paket. Tento stav sa prepína pomocou globálnej premennej ***thread_status***.

Prijímanie dát

Keď máme nastavený program na prijímanie, tak po nadviazaní spojenia máme len dve možnosti. Odhlásiť sa, alebo pokračovať, čo nastaví príjemcu na čakanie príchodu paketov. Vtedy môže dôjsť k ďalším štyrom možnostiam. Ak do minúty nedorazí žiaden paket, tak spojenie s klientom sa preruší a server sa vypne, keďže v programe má byť nastavená doba, po ktorej dôjde k prerušeniu spojenia pri neaktivite odosielateľa. Program sa nevypne, len nás vráti do prvej funkcie, kde si vyberáme užívateľa. Ak dorazí nejaký paket, tak ho dekódujeme a postupujeme ďalej podľa doručenej informácie. Môže sa jednať buď o keep alive paket, alebo informačný paket, ktorý oznamuje, či príde správa, alebo súbor. Keep alive slúži len na resetovanie doby, po ktorej by ináč došlo k prerušeniu spojenia. Ak nám príde ten, tak sa nám na obrazovke vypíše správa, v ktorej sa hovorí, že spojenie sa udržuje.

Keď vieme, že nám bude doručená správa a vieme aj z koľko paketov sa má skladať. Táto informácia sa vypíše na našej obrazovke. Potom sa spustí funkcia ***recieve_message()***, ktorá ako parametre, okrem server socketu, dostane aj počet paketov, ktoré nám majú byť doručené. Po dorazení paketu oddelíme z doručených dát prvých dvanásť bytov, v ktorých je uložená hlavička, a zvyšok uložíme do premennej v ktorej sa nachádza fragment aktuálne doručenej správy. Doručené dáta ešte predtým skontrolujeme, či dorazili v poriadku. Funkcia určená na kontrolu nám vráti poradové číslo paketu a true, alebo false, podľa toho, či bol daný paket doručený správne. Ak nebol, tak jeho poradové číslo sa pridá do premennej, ktorá obsahuje čísla poškodených paketov. Ak bol doručený správne, tak zvýšime premennú, ktorá ráta počet doručených balíkov, o jedna. Okrem toho pridáme dekódovaný fragment do poľa, v ktorom sa postupne buduje celá správa. Tento proces vykonávame pre celú skupinu doručených paketov. Potom porovnáme počet všetkých doručených paketov s celkovým počtom, ktorý má prísť. Ak ešte nie sú všetky, tak sa spustí funkcia, ktorá kontroluje, či nám nejaké pakety nechýbajú. Dostáva ako parameter správne aj nesprávne doručené pakety. Ak je dĺžka správne doručených desať, tak žiadne nechýbajú a funkcia môže skončiť. Ak nie je, tak zisťujeme, či sa v správne aj nesprávne doručených nachádza dokopy desať paketov. Keď nájdeme paket, ktorý sa nenachádza ani v jednom, tak vieme, že nebol doručený. Pridáme ho do zoznamu poškodených paketov, ktoré budú následne odoslané klientovi. Keď je prvá skupina doručená, tak sa vypíše počet nepoškodených a správne prijatých paketov. Potom sa odosielateľovi pošle správa, ktorá hovorí, buď to, že všetky pakety boli doručené v poriadku, alebo sa pošle reťazec, v ktorom sú uložené poradové čísla chybných, alebo nedoručených paketov. Potom čakáme na klienta, kým doručí požadované pakety, ktoré sa následne znova skontrolujú. Keď chýbajúci paket dorazí, tak chýbajúci fragment správy doplníme do poľa. Opakujeme, pokiaľ nebudú všetky pakety doručené. Vtedy sa cyklus prijímania preruší a na obrazovku sa vypíše doručená správa a aj informácia o tom, koľko paketov nám prišlo.

Ak nám má byť doručený súbor, tak postup je veľmi podobný. Na začiatku sa nám vypíše, aký súbor nám príde a z koľkých paketov sa bude skladať. Doručené fragmenty neskladáme do súboru hneď ako sú doručené, ale až keď sú všetky pakety prijaté. Počas prijímania ich ukladáme do poľa, ktoré bude neskôr postupne poskladané do súboru. Keď už je súbor kompletný, tak sa zobrazí informácia, že súbor bol doručený a koľko paketov nám prišlo. Súbor sa uloží do priečinku, kde sa nachádza aj samotný program. Po úspešnom doručení sa nám znova zobrazí to isté menu s dvoma možnosťami. Odhlásenie, alebo pokračovať v prijímaní dát. Teda odhlásiť sa a prepnúť na iný typ používateľa je možné len po doručení správy, alebo súboru.


Server
Odhlásiť sa
Pokračovať
Čakať na informačný paket
Prerušiť spojenie
Nie
Príde do minúty?
Áno
Dekódovať informáciu
Keep Alive paket
Príde správa
Príde súbor
Prijať skupinu paketov
Skontrolovať stav paketov
Čakať na doručenie chybných paketov
Odoslať negatívnu odpoveď klientovi
Nie
Sú v poriadku?
Áno
Odoslať pozitívnu odpoveď klientovi
Bola to posledná skupina?
Nie
Áno
Zobraziť správu, alebo informáciu o prevzatí súboru

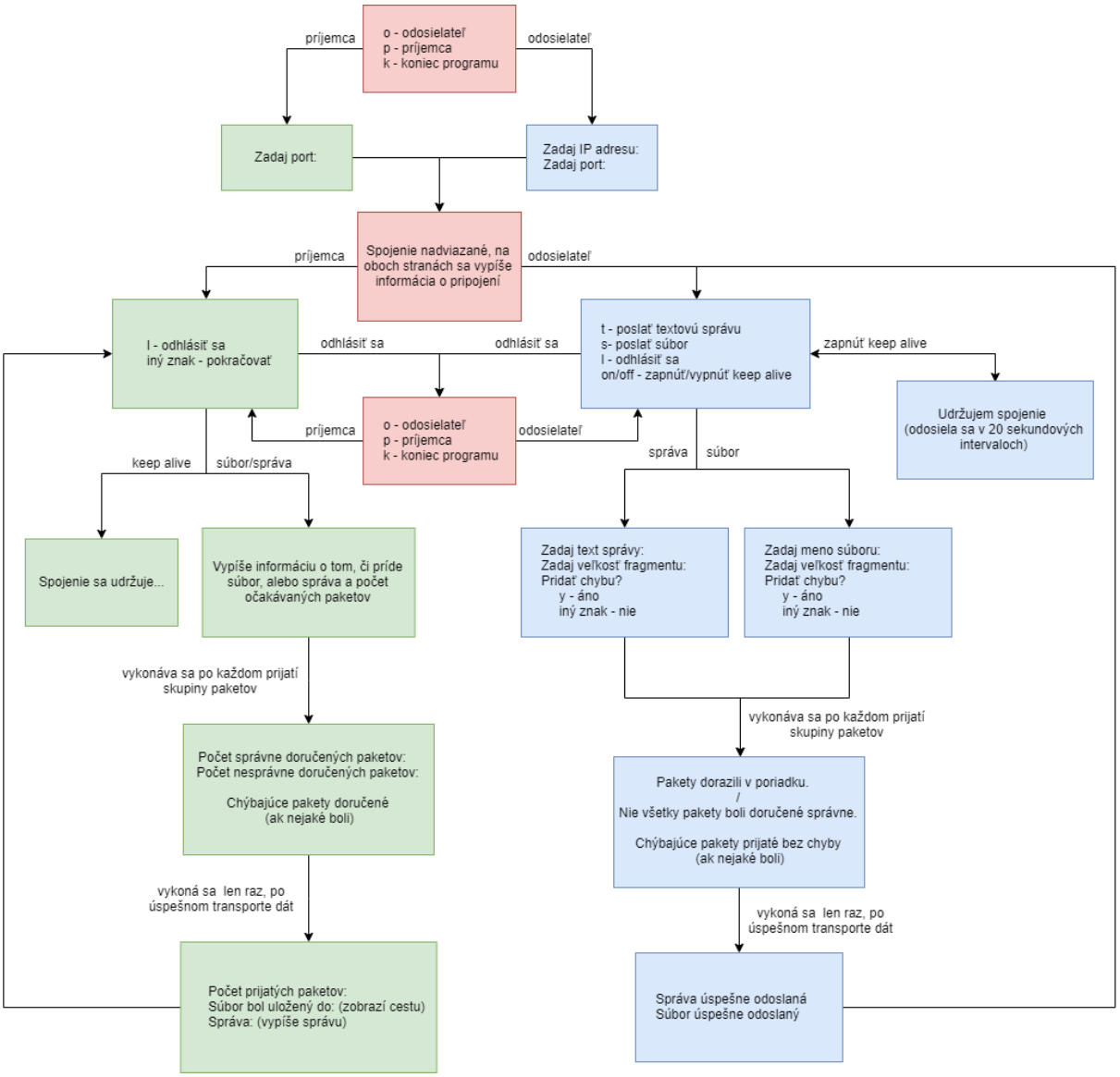
Zmeny oproti návrhu

Zmien oproti návrhu nie je veľa, len namiesto vlastne vytvorenej CRC funkcie som použil funkciu z knižnice binascii, keďže je efektívnejšia ako môj pôvodný nápad. V návrhu som taktiež chcel kontrolovať prijaté pakety, až keď bude doručená celá skupina desiatich. Ale výhodnejšie je kontrolovať ich, hneď keď sú doručené, informáciu o kontrole si uložiť, toto vykonať pre všetky pakety v skupine a až potom odosielateľovi poslať spätnú väzbu. Ak by teda bol nejaký paket chybný, tak bude odosielateľom znova zaslaný až po prijatí spätnej väzby a nie hneď počas prvotného posielania paketov, ako som pôvodne chcel. Posledná zmena oproti návrhu je to, že pri posielaní súboru posledný paket mal obsahovať jeho meno. Meno súboru bude namiesto toho doručené ešte pred samotným presunom jednotlivých fragmentov.

Implementácia, zoznam použitých knižníc a ich využitie

Program som sa rozhodol implementovať v Pythone, keďže má väčšie množstvo funkcii, vďaka ktorým nebudem musieť robiť niektoré veci manuálne a taktiež má k dispozícii veľké množstvo knižníc, vďaka ktorým sa mi čiastočne uľahčila tvorba programu. Ovládanie je realizované pomocou príkazov v konzole. Popis ovládania je tam vypísaný tiež, takže ovládanie programu je jednoduché.

Ovládanie programu, UI



Použité knižnice:

socket – spojenie dvoch užívateľov a prenos dát medzi nimi

struct – slúži na vytvorenie hlavičiek

math – použité pri vyrátaní počtu paketov, slúži na zaokrúhľovanie hore

threading – použité pri keep alive, ktorý je realizovaný cez thread

time – taktiež použité pri keep alive, zariaďuje, aby sa keep alive pakety posielali v daných intervaloch

binascii – použité pri vyrátaní CRC

random – použité pri náhodnom výbere chybného paketu pri simulácii chýb

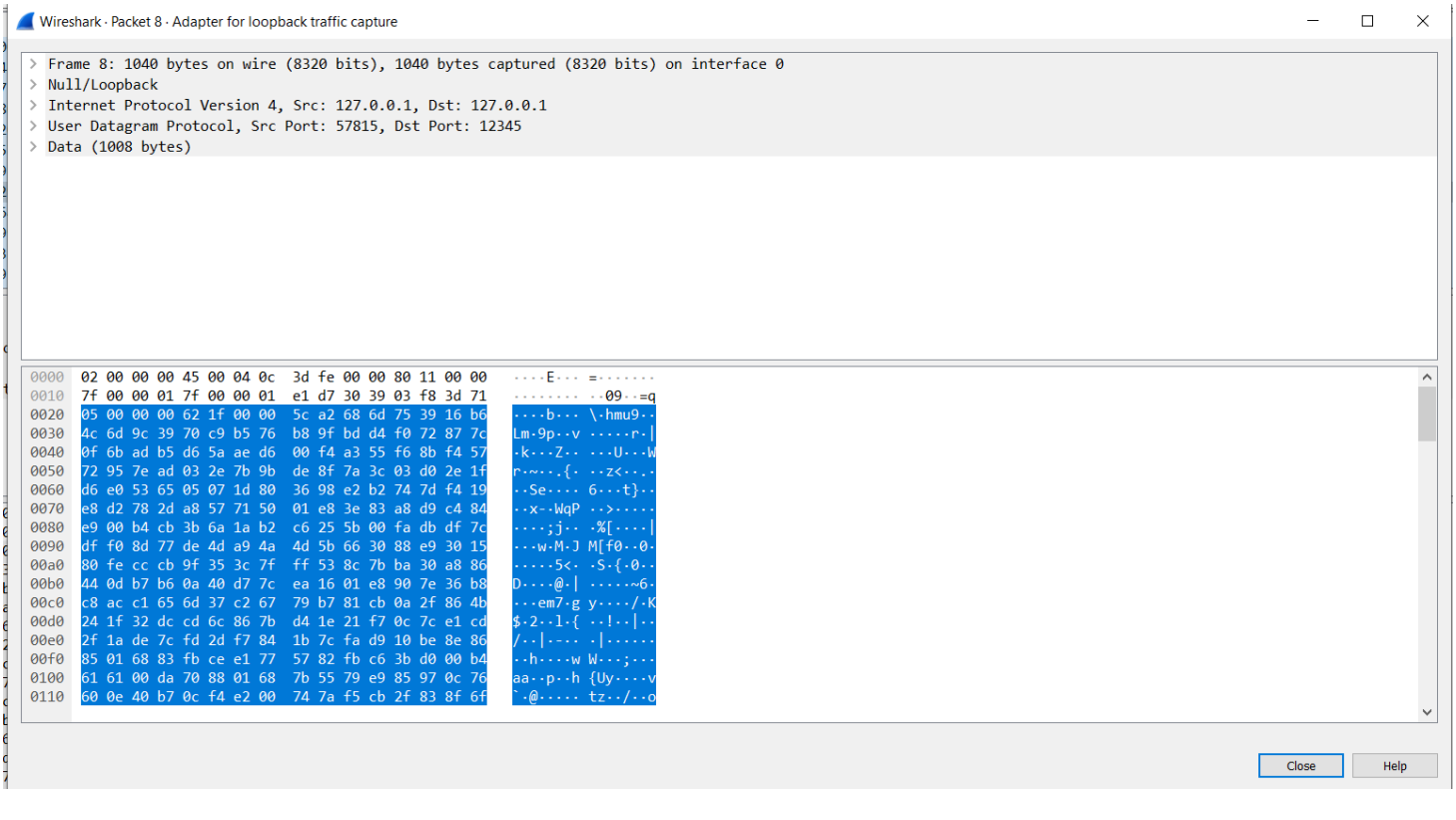
os – použité pri zisťovaní veľkosti súboru, aby som vďaka nej mohol vyrátať počet paketov

Splnené požiadavky a scenáre

Program spĺňa všetky minimálne vlastnosti, pracuje s dátami optimálne, používateľ je schopný zadať IP adresu a port servera, taktiež aj veľkosť fragmentu. Tá môže byť pri posielaní správ najviac 1488, lebo nemá dôjsť k fragmentácii na linkovej vrstve, kde je maximálna možná veľkosť 1500, a hlavička pri posielaní správ má veľkosť 12 bytov. Pri posielaní súborov je zas maximálna možná veľkosť fragmentu 1492, lebo tam je hlavička 8 bytová. Taktiež máme možnosť odoslať jeden úmyselne chybný fragment, ktorý je potom detegovaný na strane príjemcu. Ten potom vie odosielateľovi oznámiť, správne aj chybné doručenie paketov. Taktiež sa dá úspešne preniesť 2 MB veľký súbor do minúty, aj pri menšej veľkosti fragmentov.

Wireshark

V programe Wireshark môžeme vidieť veľkosť dát prenášaného paketu. Veľkosť fragmentu pri tomto testovacom prenose bola nastavená na 1000 bytov, ale celková veľkosť dát je 1008, keďže pri transporte súborov mám 8 bytovú hlavičku. Okrem veľkosti prenášaných dát môžeme taktiež vidieť IP adresu a port oboch používateľov a aj samotné prepravované dáta.



Záver

Môj program, ktorý prenáša dáta po sieti pomocou UDP protokolu, zvyšuje jeho spoľahlivosť, keďže prenášané pakety, sú na prijímacej strane kontrolované a ak je v nich nejaká vada, tak ich odosielateľ zašle znova. Program sa delí na dve časti, prijímaciu a odosielaciu, a po úspešnom pripojení sa môžu užívatelia medzi sebou vymeniť, bez toho, aby sa spojenie prerušilo. Po viacnásobnom testovaní som došiel k tomu, že projekt spĺňa všetky podmienky, funguje spoľahlivo a aj dostatočne rýchlo na to, aby sa v ňom skombinovala rýchlosť UDP a bezpečnosť TCP protokolov, aj keď, bezpečnosť samozrejme nedosahuje úroveň TCP protokolu a rýchlosť je vďaka kontrolovaniu doručených paketov mierne spomalená. Program môže slúžiť ako provizórna náhrada TCP, ak spoľahlivosť nie je príliš požadovaná a stále si chceme uchovať rýchlosť prenosu.