SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
BAKALARSKE STUDIUM. ODBOR: IB

Navrch vlastného protokolu nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP

Student: Valentyn Ipolitov
AIS Id : 92033
Cvicenia : pondelok 15:00
Cvičiaci :Ing. Kristián Košťál, PhD.

Table of contents

1. Zadanie ulohy.

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

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošlesúbor 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 rozložísúbor 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 cieľovomuzle tento zobrazí správu o jeho prijatí a absolútnu cestu, kam bol prijatý súbor uložený.

Komunikátor musí obsahovať kontrolu chýb pri komunikácii a znovuvyžiadaniechybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých20-60s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definovanésignalizačné správy.

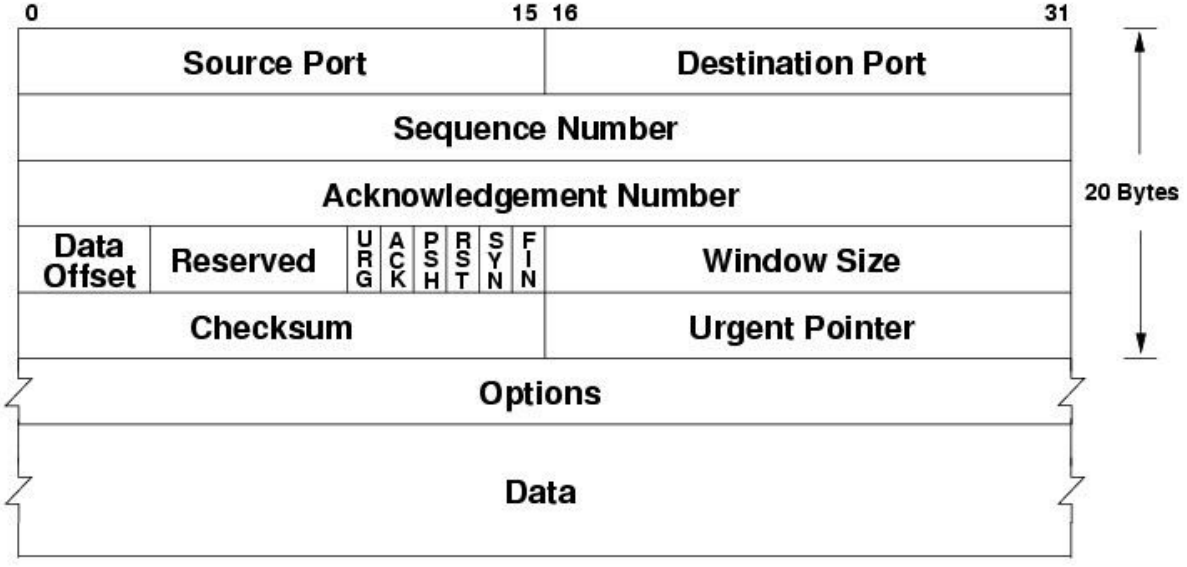
2. Analyza UDP a TCP

TCP a UDP sú protokoly, ktoré sa používajú na prenášania informácie ako postupnosť bitov, alebo paketov cez internet. TCP a UDP súčasťou transportnej vrstvy, ktorá je postavená nad IP protokolom sietovej vrstvy.
V podstate keď posielate dáta pomocou TCP alebo UDP, tak posielate pakety na konkrétnu IP adresy počítača. Protokoly sa odlišujú najmä preto že sa používajú na odosielanie rôznych druhov dáta.

2.1 TCP

TCP je najpoužívanejším protokolom na Internete. Napríklad, každý krát keď stláčame nejaký link na internete, z veľkou pravdepodobnosťou používame TCP na prenos informácie.
Charakteristiky TCP:

- Zaručuje ze prijímač dostane všetky pakety. Aby sa presvedčiť ze príjemca dostal dáta, posiela naspäť odpoveď o prijatí. Keď príjemca nie dostane správne dáta, tak posiela spravu s požiadavkou odoslať paket ešte raz. Kým nedostane potrebný paket, tak začne prijatia nasledujúceho v poradí paketu.
  Pakety taktiež sa overujú na chyby, napríklad, pomocou check summ, ktorý sa nachádza v hlavičke. Cely vyššie spomenutý proces spôsobí **spoľahlivého prenosu dat**. Kvoli spolahlivemu prenosu TCP je pomalší ako UDP.
- Keď **dáta** sú rozdelene na viac časti, tak TCP zaručuje nám že **budú prichádzať postupne v správnom poradí**. Pre tento účel sa používa Sequence Number, ktorý sa nachádza v hlavičke každého paketu.
- Ma väčšiu hlavičku niez UDP.
  Veľkosť hlavičky je 20-80 bytov. Hlavička je zobrazená na obrázku 1.

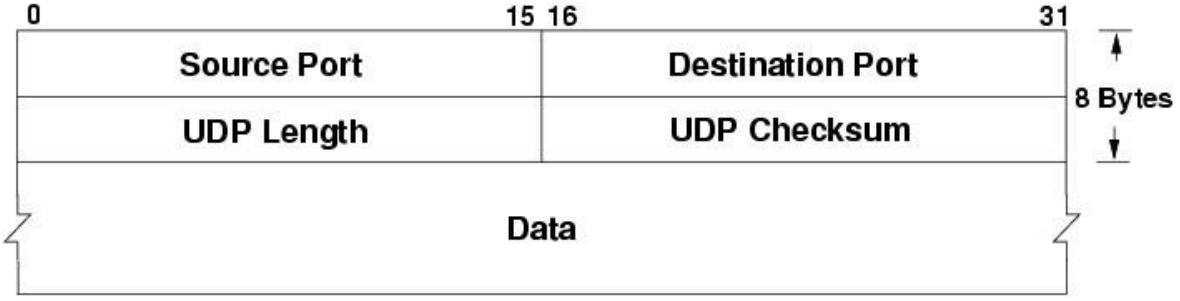


Obrázok 1. TCP hlavička

2.2 UDP

UDP je menej používanejší protokol na internete. Značným rozdielom je, že UDP nevyžaduje, aby príjemca odosiela spravu, o prijatí paketu.
Charakteristiky UDP:

- **Nespoľahlivý prenos**. Keď používate UDP protokol, pakety sa iba posielajú príjemcovi. Odosielateľ nebude čakať kým recipient dostane pakety ale bude ďalej pokračovať v odosielaní nasledujúcich dáta.
- Rýchlejší prenos ako TCP
  používa keď pre nás je viac dôležitá rýchlosť prenášania dat nez ich správnosť a integrita
- Typické sa stretávame s ním v aplikáciách pre stream video, voice and video calling a online video hry.
- UDP ma 8mi bytovu dlzku hlavicky.



Obrázok 2. UDP hlavička.

2.3 Sumarizácia pre protokoly UDP a TCP

TCP :

- Najpoužívanejší protokol na internete
- Zaručuje doručenie všetkých paketov
- Posiela pakety v poradí
- Pomaly a vyžaduje viac resursov

UDP :

- Používa sa pre stream videa, volania cez internet, live broadcasting
- Nezaručuje doručenie všetkých paketov
- Pakety nemusia prísť v poradí
- Rýchlejší a vyžaduje menej resursov

3. Navrh riesenia

3.1 Implementacne prostredie a zakladne technicko-implementacne informacie

Program budem implementovat v programovacom jazyku Python 3 s použitím nasledujúcich kniznic: socket, sys, threading,time, os, math.

Naco sa vyuzivaju jednotlive kniznice:

Nazov kniznice	**Kratky popis ako vyuzivam**	**pouzite funkcii**
`socket`	Na komunikiciu medzi klientom a serverom, pripojenie a vymen sprav.	`sendto(), recvfrom(), socket(), bind()`
`sys`	Na vychod z programu.	`exit()`
`threading`	Na rozdiel programu medzi niekolkimi threadami. Dalej bude presnejsi opis.	`Thread(), start()`
`time`	Na pocet casu, stopovac.	`time()`
`os`	Na zistenie info o subore.	`path.isFile(), path.getSize()`

Pre riešenie danej úlohy budem používať **konzolovú aplikáciu**.

Na implementaciu tejto ulohe bude vytvoreny jeden program. Program bude mat moznost sa spravat bud ako **klient** alebo ako **server**. Pouzivatel pri spusteni proramu si zvoli jednu z moznosti. Pre uspesnu komunikaciu clienta a servera *najprv sa spusta server*.

Taktiez sa da bez zastavenia programu vymenit roly clienta a servera.

Opis pouzivaniea programu:

Na komunikaciu medzi dvoma stranmi bude vyuzite 2 porty (jeden na ktorom pocuva strana, a druhy - cez ktory strana odosiela)

Pre komunikaciu obe strany maju vediet porty cez ktore budu komunikovat, teda pri spusteni ako server napriklad, `PORT_LISTEN` sa nastavi na `30000` a `PORT_SENT_TO` na `40000`.

Opacne na strane klienta `PORT_LISTEN` sa nastavi na `40000` a `PORT_SENT_TO` na `30000` a dodatocne sa nastavi IP adressa servera, aby client vediel kam ma pripojit sa.

Po pripojeni klient vie posielat na server textove spravy a subory, s tym ze program zarucuje ze poslany object dosjde na server celostny (vsetke fragmenty) a zabezpeci integritu prenasanych dat.

Pri posielani suboru, server bude informovany o rozmere suboru a o pocte fragmentov.

Pocas prijatia fragmentu bude strana informovana o zakladnych info: poradove cislo fragmentu, spravny/nespravny fragment. Po prijati celeho suboru zobrazi sa success notification a absolutna cesta k prijatemu suboru.

Kontrola prijatych fragmentov uskutoční pomocou Sequence Number a CRC.

Chybny fragment deteguje sa pomocou CRC. CRC zabezpeci to, ze ked aspon jeden bit bude odlisny od odoslaneho, tak CRC (16b alebo 32b) nesedi. Uvazoval som aj nad Checsum, ale overuje to iba spravny pocet prenesenych bitov a teda je uprimne vatsia sansa False Positive resulta(nespravny fragment vidime ako spravny).

V pripade, ze niake fragmenty nebudu spravne alebo nedostanu sa do ciele, tak pridame ich do queue a vyziadame este raz na konci komunikacii. Nerobim overenie, napr. pre kazde 10 fragmentov, lebo na priklade IMG(obrazku) mne viacej paci myslenka ze pouzivatel aj tak dostane cely subor, mozno bez niektorych casti ale keby niaky fragment stale bol zle prenasany, kvoli napriklad niakej attake MIM, tak nebude v slucke ziadat ten fragment kym neprerusi mozno spojenia, ale nazacatku dostane vsetke spravne fragmenty a potom uz bude ziadat o chybajuce. Ale na ibej strane, ked hovorime o velke subory, nevieme zarucit ze fragmenty budu poslane z cache pamate ako by pri overeni pri kazdych 10ich fragmentoch, a moze to spomalit proces.

Pre overenie funkcionality bude moznost zobrazenia chybneho prenosu fragmenta.

Server vie zmenit **max dlzku fragmentu** ktoru vie prijat. Ohladom na to ze necheme aby sa fragmentoval na linkovej vrstve, mame si vypocitat max moznu dlzku, ktoru vie si zvolit server: `1500B - 8B - 6BMB = 1486B`. Kde `8B` je dlzka UDP headera a `6MB` je dlazka mojho headera.

Pri spojeni klient sa dozvie max dlzku fragmentu od servera. Ked server pocas komunikacii vymeni dlzku, tak spravanie podobne observeru, posle vsetkym klientom spravu o zmene svojej dlzky. Je to spravene za ucelom jednoduchsej komunikacie, lebo ked chceme preniest subor tak vzdy mame si opytat prijemcu o mac. dlzku fragmentu...pri dlhsej komunikacie stale si to pytat je zbytocne a narocne na implementaciu. Keby ze sme si rozsirili nas program

v buducnosti tak tato implementacie bola by lepsi, lebo by bol min. zasah do roznych funkcie, co je spravne.

Program bude umoznovat posielanie 2MB suboru v relativne rychly cas.

Taktiež bude implementovany **KeepAlive** pre spojenie medzi stranami. Jeho funkcia bude v ukonceni medzi stranmi pri strate spojenia. Teda **kazdych 10s.** budem odosielat signal s servera na klienta a po prijati odpovede viem ze klient je stale pripojeny.

Vlastna hlavicka nad UDP:

Robim zadanie zo signalami, lebo signaly podla mna je prave ta vec ktora potrebna, je sucastou pravidiel protokolu. Kazdy signal definuje dalsie spravanie protokolu, je to dost pochopitelne a dobre sa s tym pracuje, pripadne rozsiruje.

Teraz opisem hlavicku s ktorou budem pracovat a dalej v dokumente este sa vratim ku ostatnym bodom.

Kedze poziadavka na max. subor ktory program mal by preniest 2MB - 2097152B - 2^21 a minimalny fragment si moze prijmac zvolit 1B(realne 1B dat a hlavicka), tak by sme musieli vydelit aspon 21bit pre Sequency Number.

Este potrebujeme priestor na CRC, ktrory typicky 16 alebo 32 bity, ja si zvolim 16 bitov.

Este 1B je pre signaly - max 256 roznych signalov co je ovela viac ako potrebujem.

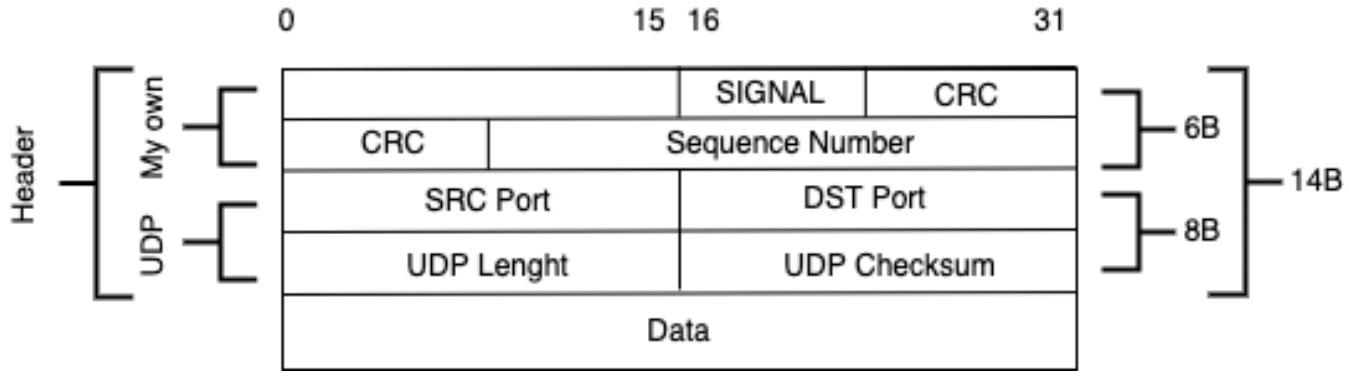
Signal code	Description
0	`SYN` - pri zaciatok spojenia
1	`ACK` - ako potvrdzdenia, odozva
2	`ACKF` - ako finalne potvrdzenie, nie zarucene ze sa nestreti
3	`MSG` - na zoslanie jednoduchej spravy ako 1 fragmentu
4	`MSGP` - poslanie spravy viacermi fragmentmi
5	`FILE` - poslanie filu 1 fragmentom
6	`FILEP` - poslanie filu viacermi fragmentmi
7	`REQP` - vyzaduje konkretny fragment
8	`SUCCP` - uspesne prijatie vsetkych fragmentov

9	`KA` - Keep Alive sprava
10	`CHUS` - vystranie funcionality servera a klienta
11	`GETCONF` - Vymena zakladnych udajov potrebnych pre komunikaciu
12	`HALT` - Ukoncenie spojenia

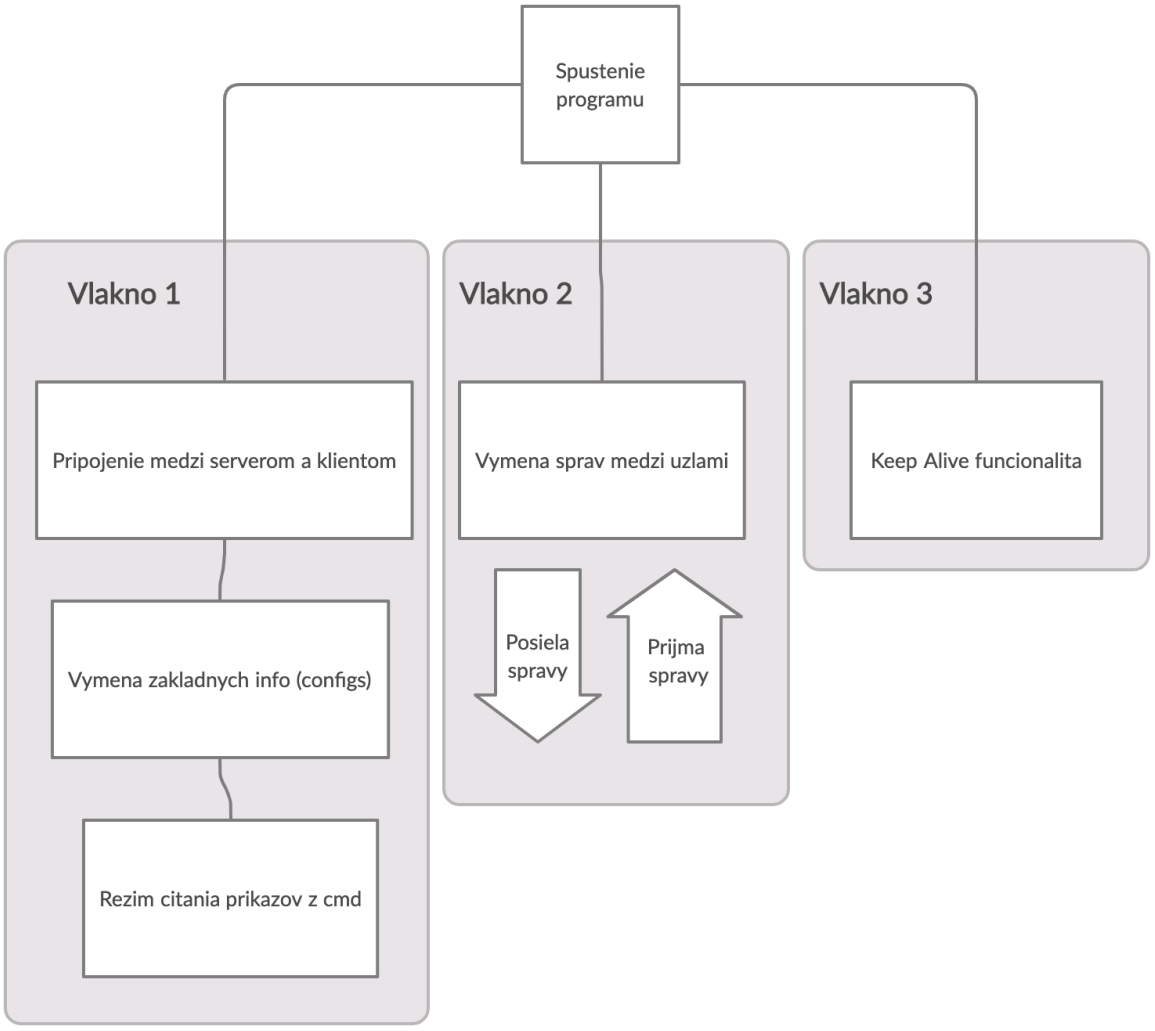
Takze celkovo potrebujem v hlavicke mat 21+16 +8 = 45 bitov navyse a zaokruhli sa to na 5 bytov(48bitov). 24 bity pojdu pre Sequence Number a 16 pre CRC a 8 pre signaly. Dokopy moja hlavicka ma 6B

Spolu s hlavickou UDP mam 14B.

Na obrazku je vidiet navrch hlavicky nad UDP.



Opis vlakien programu:



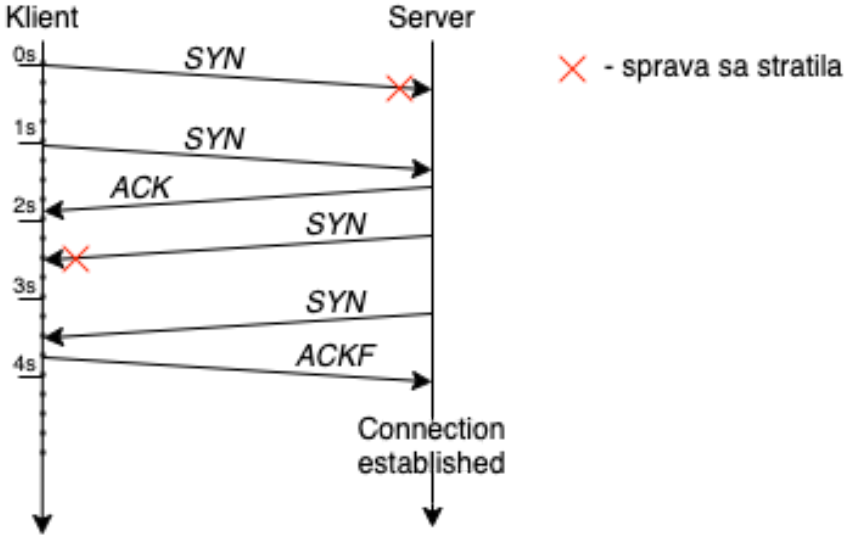
3.2 Podrobny navrh pre jednotlive caste zadania

1. Establish secure connections

   Pre spojenie **klient posiela SYN** kazdu sekundu, kym nedostane ACK alebo SYN od servera.
   Server na zaciatku **caka** na **SYN** od klienta a ihned **ako dostane** SYN tak **odosle ACK**, **a** dalej **odosiela SYN klientovi**. Preto **keby** ze **ACK** odpoved zo servera sa **strati** a serve zacne posielat **SYN** rovnako ako klient, tak preto je mozne aby klient **zobral** aj SYN **namiesto ACK**.

Dalej kedze je to cca rovnaka situacia pri **server-klient SYN-ACK** a ACK z klienta moze stratit, tak server snazi sa iba do 10 sekund posielat SYN na klienta. Po 10 sekundach spojenie sa uzatvori a klientovi treba skusit pripojit sa este raz.

Na obrazku dole, je vidiet jedno z moznych uspesnych pripojeni: klient posiela SYN a nedostava odpoved, tak o sekundu posle este jeden SYN a dostane ACK.
Server posiela SYN a nedostane odpoved do sekundy, tak posle este jeden SYN na ktory uz dostane ACK od Klienta a preto spojenie bude uspesne.



Kebyze predstavime tu situaciu medzi dvomi ludmi, co nemozu seba pocut a vidiet, ale vedia iba pisat listy, tak Klient v slucke kazdu sekundu posiela Serverovi list a caka na odpoved. Ked Klient dostane odpoved, tak uz je spokojny ze sa tam nachadza Server na inej strane, ale Klient je slusny a chce aby aj Server vediel ze stale je tam a neodisel kym cakal na odpoved a preto caka na este jeden list od Servera, ktory rovnako posiela ich kazdu sekundu. Ked dostane tento list tak posle svoj, na ktory sa tesi Server ale ked nestihne poslat do 10 sekund, tak Server odide, lebo co ked Klient poslal odpoved serverovi a ta sa stratila postou, tak potom Server ju nikde nedostane a zbytocne bude overovat na postu.

2. Change max. size of file

Max. mozme preniest 1500B cez linkovu vrstvu bez fragmentacii, 8B je hlavicka UDP protokolu a 5B je moja hlavička => 1500 - 8 - 5 = 1487.
1487B je max mozny rozmer fragmentu, ktory si vie nastavit prijemca.

V programi vie nastavit to prikazom `chfsize`.
Hranice fragmentu od 1B az do 1487И.

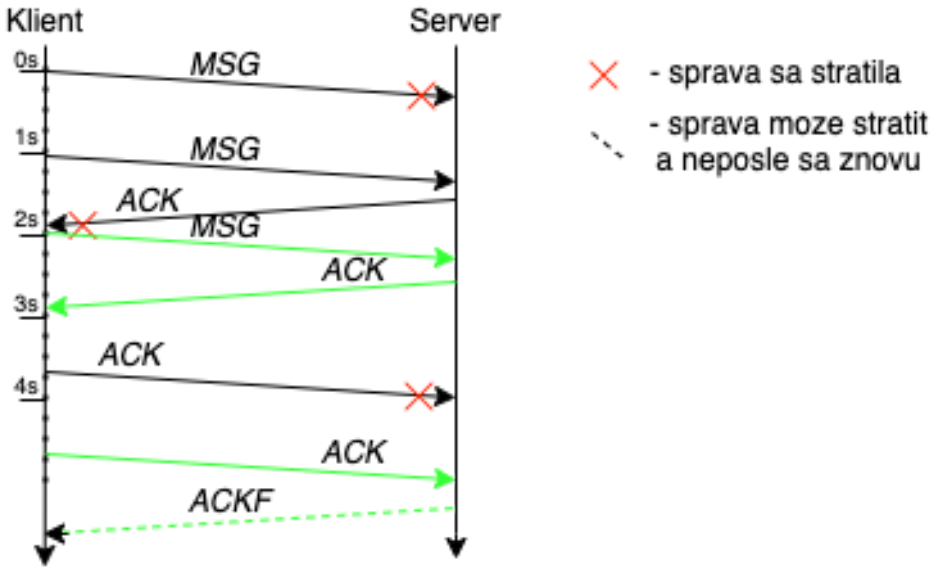
3. Poslanie spravy

a. Sprava sa posle ako 1 fragment

Pre posielanie sprav medzi uzlami budem vyuzivat wrapper funkciu, ktora kazde poslanu spravu bude zarucovat ze sa dostala. *Tuto funkciu budem pouzivat aj dalej* pri posielani suboru, lebo budem potrebovat, napriklad poslat nazov a rozmer suboru prijemcovi.
Funkcia funguje podobne tej, ako sme vytvorali spojenie.
Teraz budu graficke priklady, kde zelenymi sipkami su zobrazene ockavana spravanie na najmensi pocet krokov pri konkretny pripad.

Priklad 1:

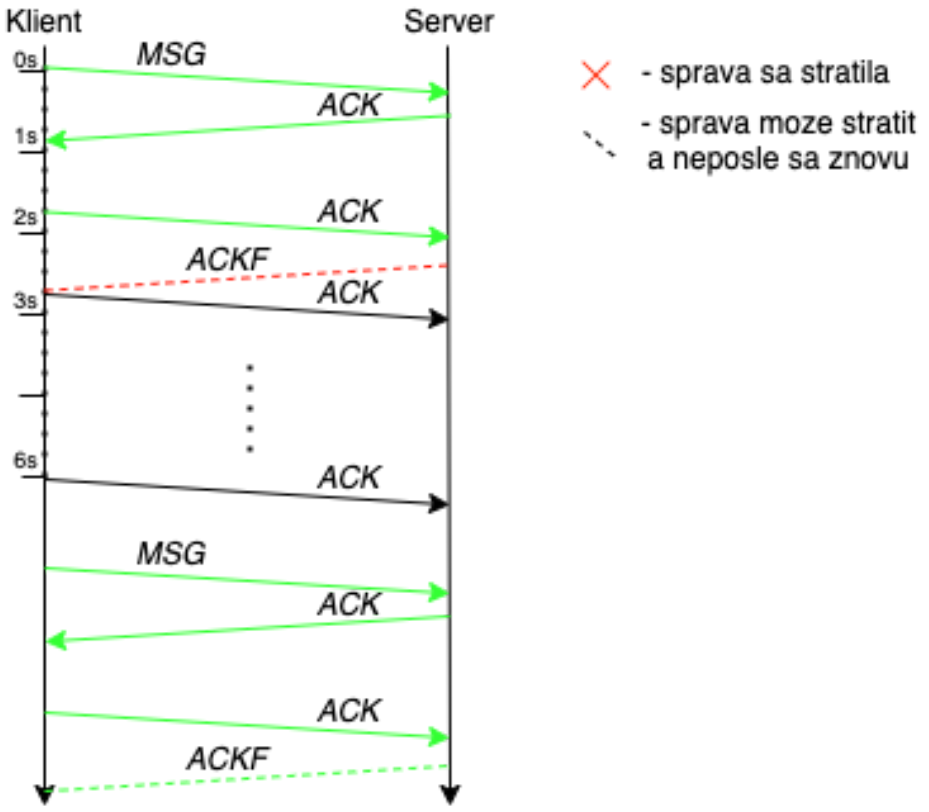
1. **Klient** posle msg a ten nedojde
2. **Kleint** posle message a ten dojde
3. **Server** potvrdi prijatia ale potvrdenie sa strati
4. **Klient** posle message a ten dojde
5. **Server** potvrdi prijatia a to dojde
6. **Klient** potvrdi ze vie ze server prijal spravu a ta nedojde
7. **Klient** potvrdi ze vie ze server prijal spravu a ta dojde
8. **Server** posle Final potvrdenie a to dojde



ACKF moze nedojst klientovi a preto Klient pocas 5s sa snazi poslat ACK ale keby ze nedostane ACKF, bud preto ze stratila alebo pretozr server nedostal ACK tak zacne proces poslania od znova. Je to ukazane na obrazku Priklad 2.

Priklad 2:

1. **Klient** posle message a ten dojde
2. **Server** potvrdi prijatia a to dojde
3. **Klient** potvrdi ze vie ze server prijal spravu a ta dojde
4. **Server** posle Final potvrdenie a to nedojde
5. **Klient** posle message a ten dojde
6. **Server** potvrdi prijatia a to dojde
7. **Klient** potvrdi ze vie ze server prijal spravu a ta dojde
8. **Server** posle Final potvrdenie a to dojde



b. Sprava sa posle ako viacero fragmentov

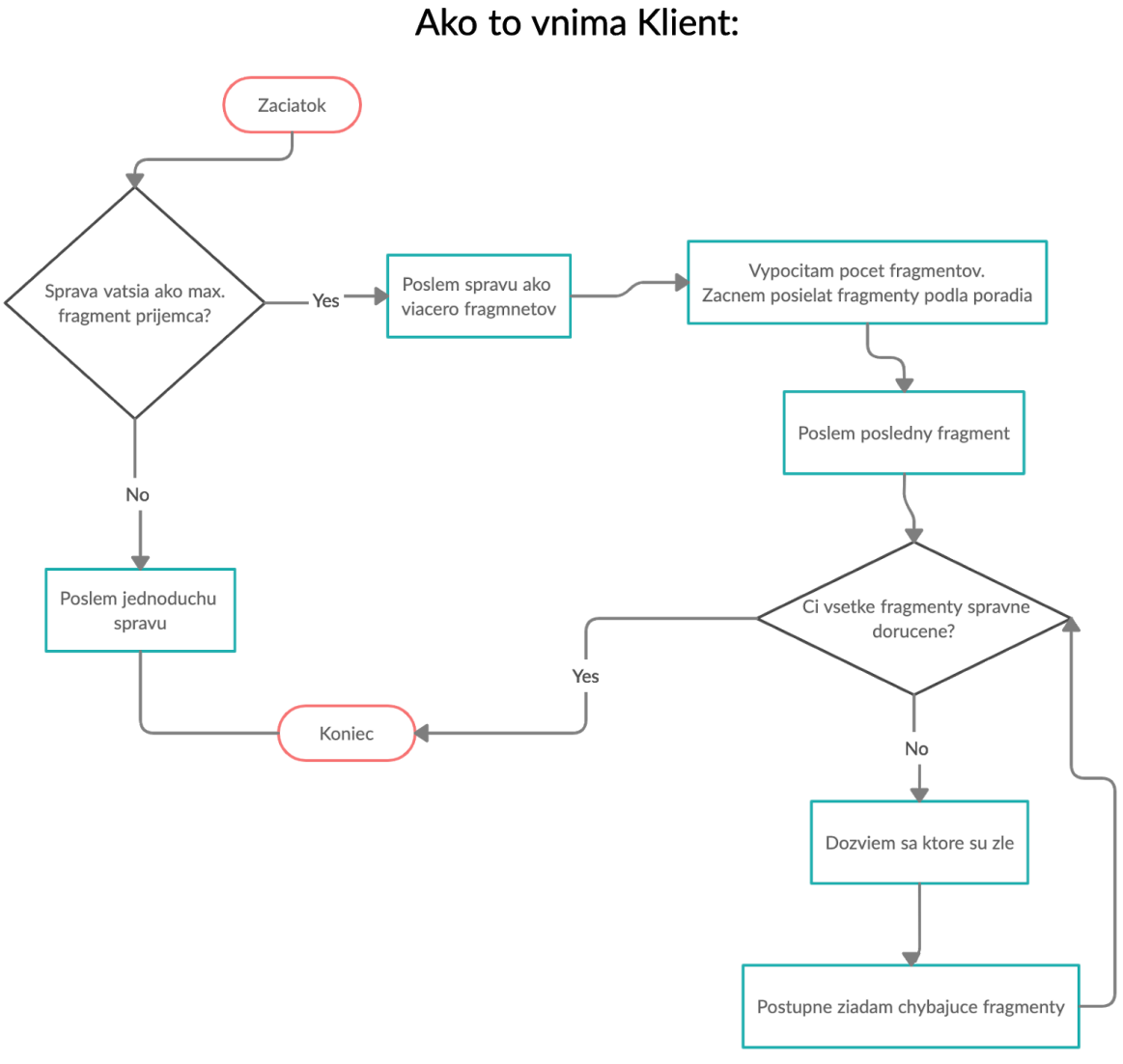
V pripade, ze moja sprava bude vatsia ako max. dlzka fragmentu, co dovoluje prijat druha strana, tak budem musiet fragmetovat spravu a posielat ju po cisasi. Na strane prijemca budem spravu skladat a po prijati vsetkych fragmentov ju zobrazim prijemcovi. Vlastne ta situacia opisuje situaciu pri posielani suboru, nie rozdel pre protokol ale skor pre to ako spracovuju to klient a server(vypis nazvu suboru/vypis spravy atd.)

c. Overovanie spravnosti prenesenej spravy

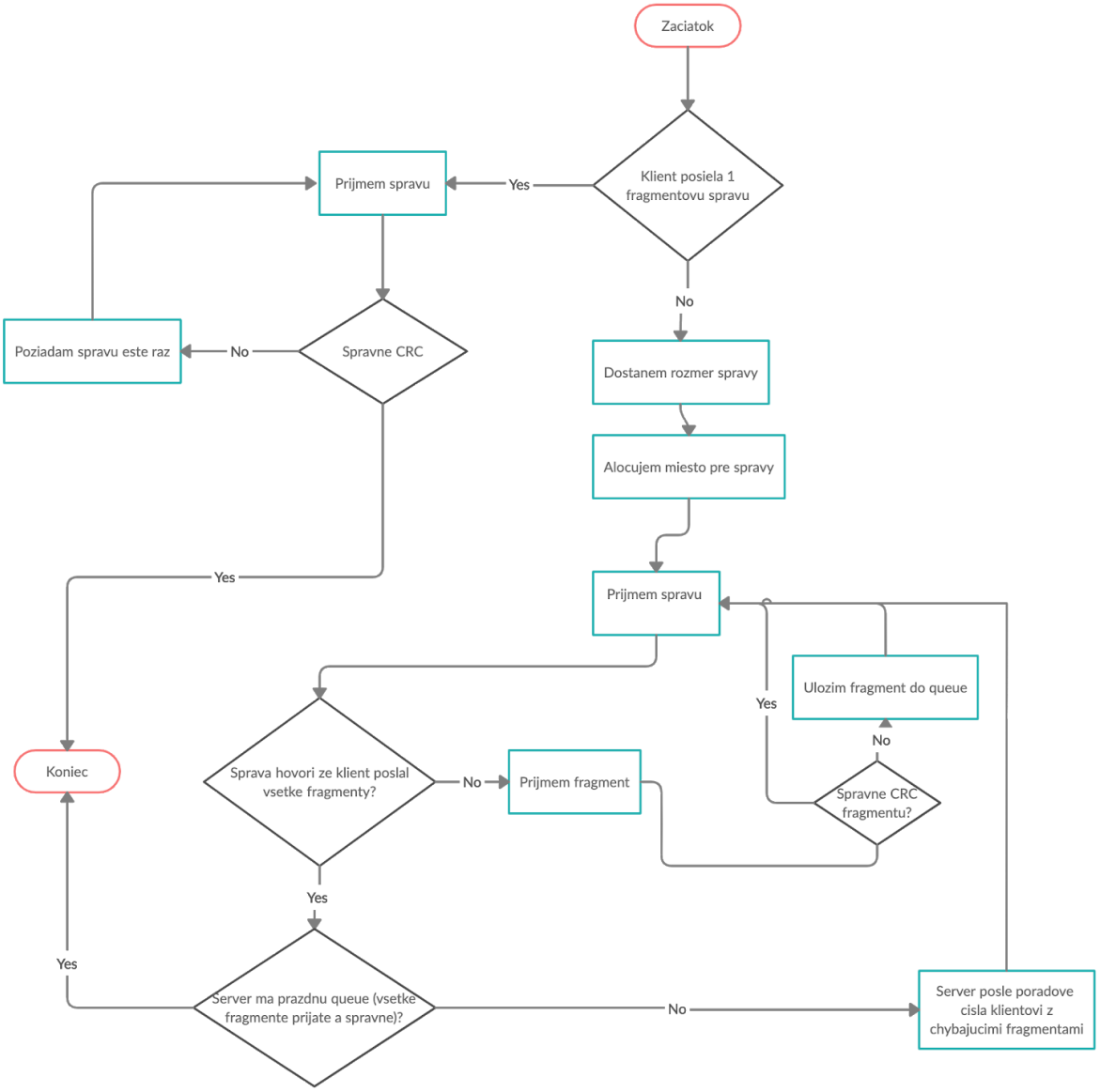
V pripade, ze sprava je odoslana cela na raz, tak overim ju integritu pomocou CRC. V pripade ze odosielam spravu po ciasti, tak overim integritu kazdeho fragmentu pomocou CRC a poradie fragmentu overim pomocou Sequence Number.

d. Zhrnutie

V zhrnuti chcem pre jasnost pridat diagram, v ktorom bude vidiet celkove spravanie dannej funkcionality na vysshej urovni ako zo strany seklientarvera tak aj zo strany servera.



Ako to vnima Server:



4. Posielanie suboru

Subor sa posiela tak, ze klien napise cestu k suboru serverovi sa posle request na poslanie suboru. Pomocou requesta server vidi nazov suboru a rozmer. Dalej server ma na vyber - bud prijme subor alebo nie.

a. Subor sa posle 1 fragmentom

Jedno fragmentovy subor sa posle pomocou signala FILE.

b. Subor sa posle viacermi fragmentmi

Posiela sa pomocou signalu FILEP (file part)

Pri posielani fragmentu server dostane konzolovy vypis s poradovym cislom fragmentu a ci je spravny alebo nie. Po poslani vsetkych fragmentov sever si overi queue, kde uklada nespravne fragmenty a ked su nespravne tak poziada ich znovu.

c. Overenie ci subor je prijaty cely a prijaty spravne

Pomocou CRC sa zistuje spravnost fragmenta ako som uz spominal.

d. Zhrnutie

Aby zbytocne nekomplikovat vysvetlenie diagramom, staci pozriet na diagram pre poslanie spravy a na rozdel, ze posielam subor. Koncept je rovnaky, lebo v posielani spravy pripustam, ze moze sa fragmentovat a poslanie suboru je rovnako prenos bitov cez siet ako aj prenos spravy.

Ale chcem este uviest, ze ked server dostane rozmer suboru, tak alokuje pole(1 policko ma rozme max. fragmenta) a pri tom ako server dostane spravny fragment tak konstantne vyhlada jeho miesto a zapise do pola. Na konci pole sa vrati ako subor.

5. Simulacia chyby prenosa

Pre overenie, ze nam funguje CRC a celkovo spravny prenos dat bude implementovana funkcia, ktora posiela spravu z nespravnym CRC. V konzole bude vidiet interakciu medzi klientom a serverom pomocou vypisov.

6. Keep Alive

Keep alive pomocou noveho vlakna aj na strane clienta, aj na strane servera. Serverom bol vytvoreny novy socket, ktory stale pocuval na porte `55055` a client zosielal tam spravu `KA` opakovane kazde 5s.. Ked ze server nedostane `KA` za 60 sekund tak vie ze spojenie neni alebo mozno ze aj 12 KA signalov sa stratilo.

7. Prepinanie medzi S/C bez reštartu

Pre vymenu bude pouzity signal `CHUS` (change us). Zacat vymenu moze hociaky uzel, ale s tym ze uzel na druhej strane ma potvrdit zmenu.

3.3 Dokunentacia k pouzivaniu rozhraniu

Spojenie

Klient

1. napise port na ktorom pocuva
2. napise port na ktory posiela spravy
3. napise IP addressu prijemcy

Server

1. Napise port na ktorom pocuva
2. napise port na ktory posiela spravy

Poslanie spravy

prikazom sm konzola poziada aby klient napisal spravu a stlacil enter pre ju odoslanie, alebo escape pre zrusenie odoslania

Poslanie subou

prikazom sf konzola poziada o absolutnu cestu suboru. Stlacenim enter pouzivatel potvrdi poslanie suboru (ked existuje), alebo escape zrusi posielanie suboru

Dalsie prikazy:

Prikaz	Description
`chsize`	Zmena max. fragmentu ktory prijmem
`chus`	Vymena roli servera a klienta
`sm`	Poslanie spravy
`sf`	Poslanie suboru
`sem`	Poslanie spravy s chybou v interaktivnom rezime
`kaoff`	Vypnutie Keep Alive
`kaon`	Zapnutie Keep Alive
`exit`	Ukonci programm

4. Zaver

Môj návrh pre túto úlohu bude riešiť jednosmernu komunikáciu dvoch počítačov v siete na základe IP adresy. Na implementáciu bude použitý programovací jazyk Python 3. Aplikáciu sa bude dat spustiť v príkazovom riadku a pomocou vyššie definovaných prepínačov. Aplikacia umozni posielať správy a súboru medzi prepojenými uzlami zo zochovanim integrity dat.

Keďže chceme aby sme mali spoľahlivý prenos dát, tak použijem CRC pre fragmenty.. Fragmenty budu mat v hlavičke sequence number pre detekciu správneho poradia odoslaného fragmentu.

Uz v stadii navrhu lepsie rozumem prenosu dat medzi sieti a protokolam.