

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Adam Benovič
Dokumentácia k 2. zadaniu

Študijný program: INFO4

Ročník: 3.

Predmet: Počítačové a komunikačné siete

Cvičiaci: Dr. Ing. Michal Ries

Akademický rok: 2018/2019

Obsah

1. Zadanie 2 : Komunikácia s využitím UDP protokolu	3
2. Analýza zadania	4
2.1 Hlavičky a enkapsulácia	4
2.2 Dôležité vlastnosti UDP pre potreby zadania	5
2.3 Chyby v prenose a ich oprava	5
2.3 Veľkosť fragmentov	5
2.3 Zhrnutie analýzy a ďalšie špecifikácie	5
3. Návrh riešenia	6
3.1 Špecifikácia technických detailov a užívateľského rozhrania	6
3.2 Návrh vlastnej hlavičky	6
3.3 Dáta a fragmenty	7
3.4 Detekcia chýb	7
3.5 Strata spojenia	7
3.6 Implementačné prostredie, knižnice, funkcie	8
3.7 Vysielanie	9
3.8 Prijímanie	10
3.9 Detekcia chyby	11
4. Implementácia	12
5. Zhodnotenie	12
6. Zdroje	12

1. Zadanie 2 : Komunikácia s využitím UDP protokolu

Nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP navrhnete a implementujete program, ktorý umožní komunikáciu dvoch účastníkov v sieti Ethernet, teda prenos správ ľubovoľnej dĺžky medzi počítačmi (uzlami).

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošle správu inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Vysielajúca strana rozloží správu na menšie časti - fragmenty, ktoré samostatne pošle. Správa sa fragmentuje iba v prípade, ak je dlhšia ako max. veľkosť fragmentu. Veľkosť fragmentu musí mať používateľ možnosť nastaviť takú, aby neboli znova fragmentované na linkovej vrstve.

Po prijatí správy na cieľovom uzle tento správu zobrazí. Ak je správa poslaná ako postupnosť fragmentov, najprv tieto fragmenty spojí a zobrazí pôvodnú správu.

Komunikátor musí vedieť usporiadať správy do správneho poradia, musí obsahovať kontrolu proti chybám pri komunikácii a znovuvyžiadanie chybných rámcov, vrátane pozitívneho aj negatívneho potvrdenia. Pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 60-120s. Odporúčame riešiť cez vlastne definované signalizačné správy.

Program musí mať nasledovné vlastnosti (minimálne):

1. Program musí byť implementovaný v jazyku C/C++ s využitím knižníc na prácu s UDP socket, skompilovateľný a spustiteľný v učebniach. Odporúčame použiť knižnicu *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 správy 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. poslanú resp. prijatú správu,

b. veľkosť fragmentov správy.

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

7. Možnosť odoslať súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor. Akceptuje sa iba ak program preniesie 1MB súbor do 30s bez chýb.

2. Analýza zadania

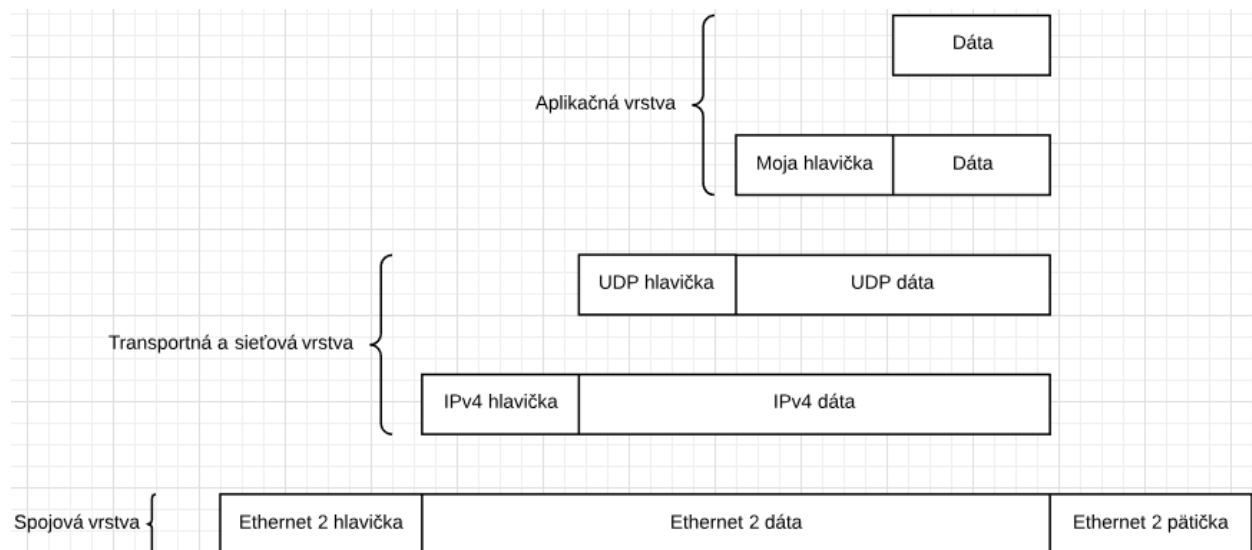
2.1 Hlavičky a enkapsulácia

Program bude zo zadania pracovať s Ethernet 2 rámcami , IP protokolom a UDP protokolom.

Ethernet 2 nám poskytuje základnú štruktúru hlavičiek s cieľovou aj zdrojovou MAC adresou a od 46 až po 1500 Bajtov na dátovú časť. V dátovej časti Ethernet 2 rámca bude vsadený IPv4 protokol, ktorého hlavička nám umožňuje umiestniť informácie o zdrojovej aj cieľovej IP adrese a informácia o použítom protokole na vyššej vrstve – v našom prípade UDP. UDP nám umožňuje posielat' informáciu o zdrojovom a cieľovom porte, na ktorom bude odosielateľ vysielat' a príjmateľ prijímať správy/súbory.

V zostatkovej časti pre dáta budem implementovať ešte vlastný „protokol“. Štruktúre jeho hlavičky a dátovej časti sa budem venovať v dokumente neskôr.

Pri odosielaní dát je nutné ich enkapsulovať. Pri prijímaní dát je nutné ich deenkapsulovať. V mojom programe sa k štandardnej enkapsulácii a deenkapsulácii pridá ešte môj vlastný „protokol“ vo forme hlavičky a dát, pričom dáta môžu byť presne také isté ako užívateľ zadal alebo sa môžu rozdeliť na viac častí – budem riešiť v dokumente ďalej.



Obr.1 Enkapsulácia

2.2 Dôležité vlastnosti UDP pre potreby zadania

1. Nevyžaduje nadviazanie spojenia pred odosielaním dát.
2. Nekontroluje prijatie dát – umožňuje vlastnú implementáciu na aplikačnej vrstve.
3. Je orientovaný na jednoduché prenosy, ideálny pre naše potreby.

2.3 Chyby v prenose a ich oprava

Chyby v prenose nastávajú pomerne často a z rôznych dôvodov, najčastejšie je to nestabilné pripojenie. Keďže UDP ako taký neumožňuje dôkladnú kontrolu prijatých dát a už vôbec nie ich automatickú retransmisiu, je nutné implementovať svoju vlastnú kontrolu aj retransmisiu.

Tomuto sa budem venovať v dokumente neskôr.

2.3 Veľkosť fragmentov

Fragmentácia dát sa štandardne robí pri enkapsulácii a takisto to budem robiť aj ja. Zadanie nám špecifikuje, že užívateľ musí mať povolené zadať veľkosť fragmentu takú, aby už mimo mojej vlastnej fragmentácie dát nedochádzalo k fragmentácii štandardnej – teda na nižších vrstvách – transportnej a sieťovej. Veľkosť fragmentu budem teda musieť počítať zo známych veľkostí rámca a hlavičiek – IPv4, UDP a mojej vlastnej. Viac o tomto probléme v dokumente neskôr.

2.3 Zhrnutie analýzy a ďalšie špecifikácie

Už teda viem, aké protokoly budem používať, že musím deliť správy na fragmenty v aplikačnej vrstve, musím riešiť chyby pri prenose a retransmisiu chýbajúcich dát. Ďalej zo zadania vyplýva, že pri nečinnosti sa program neukončí ale zostane počúvať – odosielateľ musí teda posielat' signalizačnú správu, aby prijímateľ neprestal počúvať. Program musí byť univerzálny – musí vedieť aj prijímať aj odosielať správy.

3. Návrh riešenia

3.1 Špecifikácia technických detailov a užívateľského rozhrania

Program pracuje iba medzi 2 Windows PC v jednej sieti LAN.

Program bude odosielať alebo prijímať (buď len odosiela alebo len prijíma) užívateľom zadané textové správy a textové súbory (.txt) do veľkosti 1MB – limit platí aj pre súbory aj pre správy.

Užívateľské rozhranie bude veľmi jednoduché – v príkazovom riadku. Po spustení programu bude užívateľ vyzvaný pre zvolenie, či chce aby program vysielal (stlačenie 1ky) alebo prijímal (stlačenie 2ky). Pri zvolení prijímania už užívateľ nemusí nič robiť, správy sa mu budú zobrazovať a súbory sa budú ukladať v priečinku, kde sa nachádza program. Pri zvolení odosielať bude užívateľ vyzvaný na zadanie IP adresy prijímajúceho stroja. Po zadaní bude vyzvaný k zvoleniu veľkosti fragmentu. Ďalej bude vyzvaný, či chce odosielať správu (1ka) alebo súbor (2ka). Následne bude vyzvaný k zadaniu správy na odoslanie alebo zadaniu cesty k súboru (ak sa súbor nachádza v rovnakom priečinku ako program, stačí zadať názov súboru aj s príponou, inak treba zadať absolútnu cestu). Po ukončení odosielať bude užívateľ výpisom informovaný, či sa správa poslala úspešne alebo nie.

Veľkosť fragmentu si užívateľ môže vybrať minimálne 64B a maximálne 1452B.

Naraz sa môže posilať maximálne jeden súbor alebo správa. Ďalšie odosielať je možné až po úspešnom odoslaní predchozej správy alebo súboru.

3.2 Návrh vlastnej hlavičky

Index	Checksum	Počet Fragmentov	Typ dát	Veľkosť dát
-------	----------	------------------	---------	-------------

Obr.2 Vlastná hlavička

1. Index – 4B – poradové číslo aktuálneho rámca
2. Checksum – 4B – kontrolný checksum aktuálneho rámca
3. Počet Fragmentov – 4B – celkový počet fragmentov dát
4. Typ dát – 4B – označuje typ prenášaných dát, teda súbor alebo text
5. Veľkosť dát – 4B – veľkosť prenášaných dát v aktuálnom rámci

Veľkosť 4B pre jednotlivé políčka som si zvolil kvôli jednoduchosti – integer má 4B.

3.3 Dáta a fragmenty

Dáta môžu mať maximálnu veľkosť takúto: $1500 - 20 - 8 - 20 = 1452$ B. Musím odrátavať hlavičky protokolov. Teda maximálna veľkosť, ktorú môže užívateľ zadať je 1452 B. Minimálna veľkosť bude nastavená na 64B pre zjednodušenie.

3.4 Detekcia chýb

Rámce budú odosielané v poradí tak ako sa rozdelili – 0, 1, 2, 3... n rámcov. Po prijatí všetkých rámcov prijímateľ overuje checksum. Ak sedí, rámec si uloží. Ak nesedí, rámec zahodí a vyžiada si chybné rámce a tie, čo nedostal vôbec – toto vie podľa poradového a celkového čísla rámcov. Ak už má všetky rámce, odošle signál OK aby odosielateľ vedel, že všetko bolo poslané v poriadku.

Detekcia chyby alebo checksum sa bude robiť na bitových dátach.

Checksum bude číslo vytvorené tak, že sa bude cykliť po jednotlivých bitoch a do checksum sa pripočíta exkluzívne alebo i-teho bitu dát.

Kontrola bude prebiehať reverzne – od čísla sa odpočíta exkluzívny or i-teho bitu dát. Ak víde 0, dáta sú v poriadku, ak nie tak sa zahodia a vyžadujú nanovo.

3.5 Strata spojenia

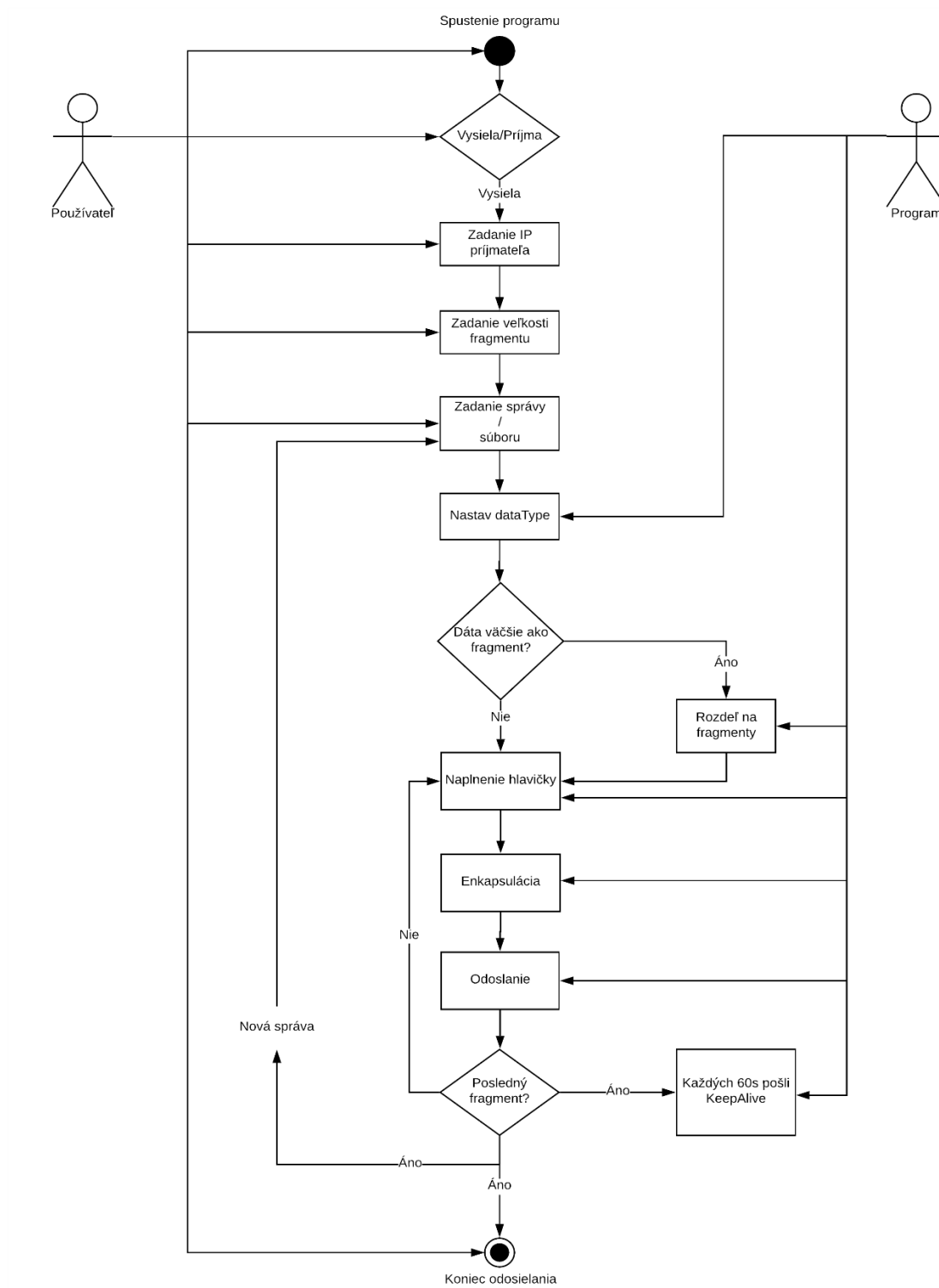
Pri strate spojenia sa používaný socket dostane do nežiadúceho stavu a pri snahe odoslať rámec sa vráti chyba. Preto je nutné daný socket uvoľniť a otvoriť nanovo. V praxi to vyzerá tak, že odosielateľ sa bude snažiť odoslať aj keď nie je pripojený, dostane chybu, socket zatvorí a otvorí nanovo a bude sa znova snažiť odoslať. Toto opakuje až pokiaľ je odoslanie úspešné – spojenie je znovu úspešne nadviazané.

3.6 Implementačné prostredie, knižnice, funkcie

Zadanie som sa rozhodol implementovať pre operačný systém Windows za použitia knižnice winsock2.h spolu s knižnicami arpa/inet.h a netinet/in.h. Zatiaľ som na použitie identifikoval nasledovné funkcie, ktorú sú súčasťou zmienených knižníc alebo sú moje vlastné:

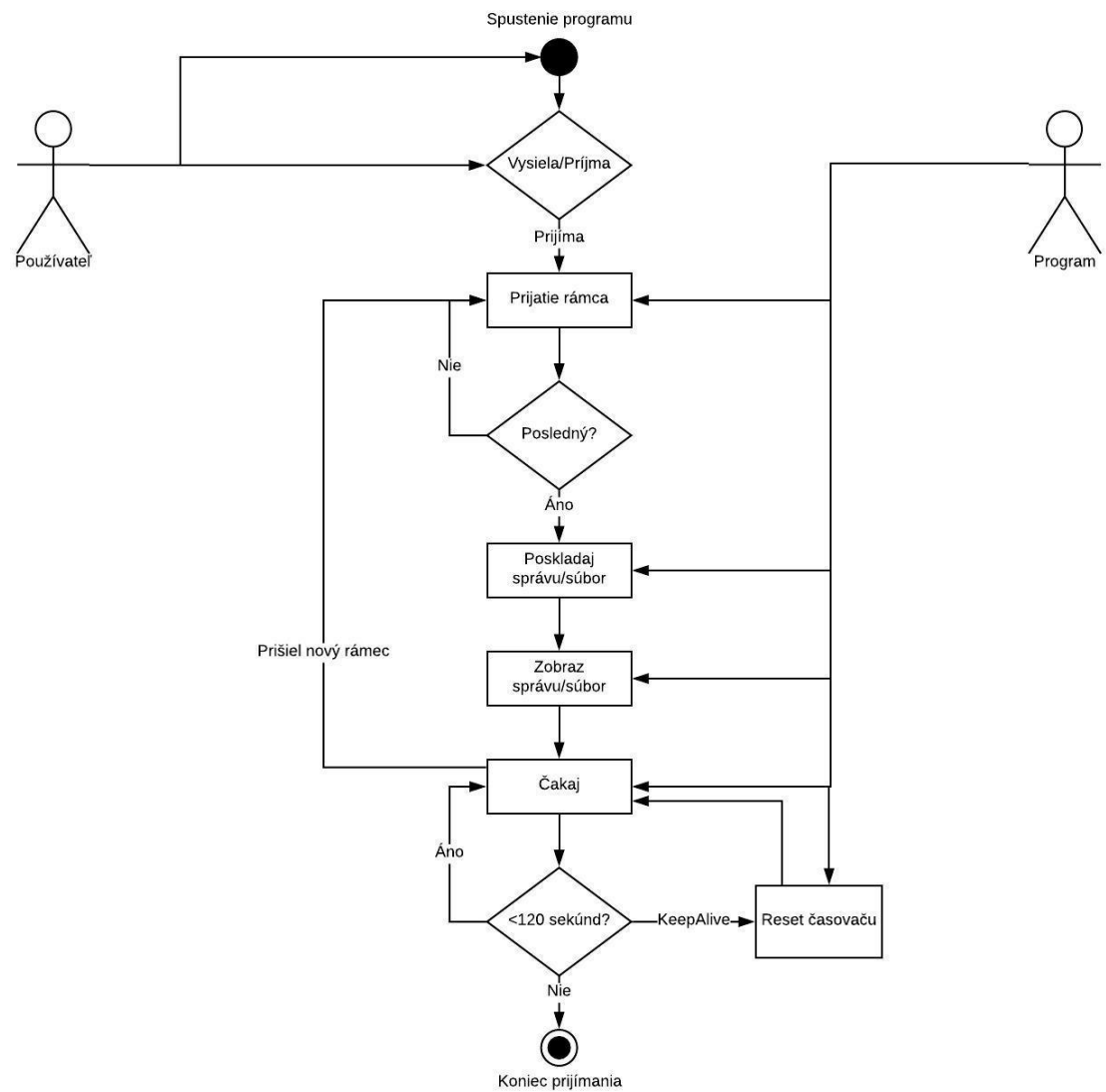
1. socket(int af, int type, int protocol) – vytvorenie socketu pre prijímanie/odosielanie
2. bind(SOCKET s, const struct sockaddr *name, int namelen) – asociácia lokálneho pripojenia so socketom
3. sendto(SOCKET s, const char *buf, int len, int flags, const struct sockaddr *to, int tolen) – ak sa rozhodnem na začiatku urobiť handshake, budem posielat' na konkrétny socket
4. recvfrom(SOCKET s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen) – prijímanie dát na danom sockete
5. closesocket(SOCKET s) – zatvorenie socketu
6. makeChecksum(char *data) – vytvorenie checksumu pre odoslanie
7. checkChecksum(char *data) – skontrolovanie checksumu pri prijímaní
8. toBinaryData(FILE *f) – konvertovanie súboru na binárne dáta
9. toFile(char *data) – konvertovanie prijatých dát späť na súbor
10. fragmentData(char *data) – rozfragmentuje dáta na menšie časti, zadané užívateľom
11. deFragmentData(char *data) – poskladá dáta do stavu pred fragmentáciou

3.7 Vysielanie



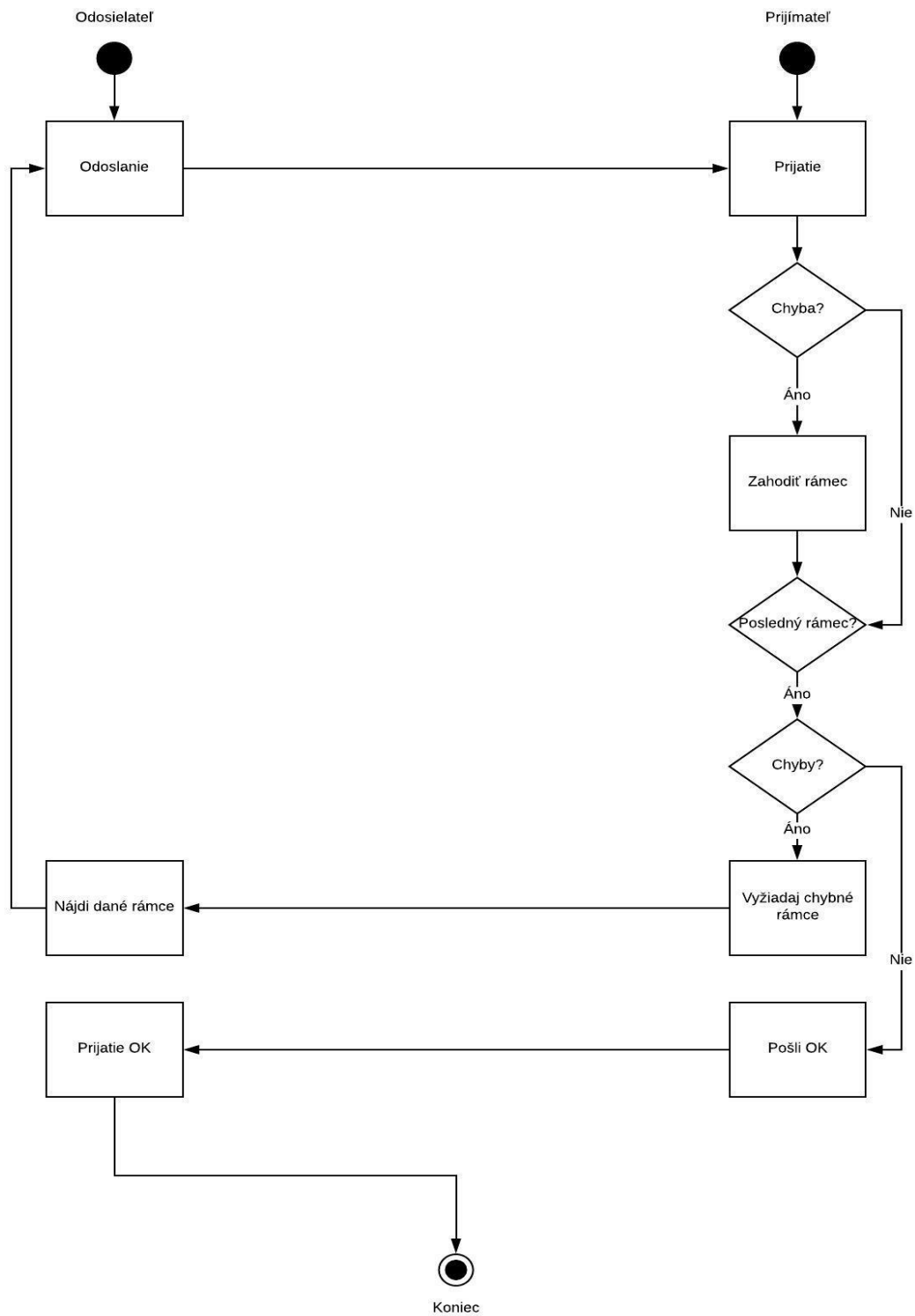
Obr.3 Vysielanie

3.8 Prijímanie



Obr. 4 Prijímanie

3.9 Detekcia chyby



Obr.5 Detekcia chyby

4. Implementácia

5. Zhodnotenie

6. Zdroje

1. Zadanie
2. Vedomosti
3. Prednášky
4. <https://tools.ietf.org/html/rfc768>
5. <https://tools.ietf.org/html/rfc791>
6. <https://docs.microsoft.com/en-us/windows/desktop/api/winsock2/>
7. Konzultovanie so spolužiakmi