Slovenská Technická Univerzita v Bratislave Fakulta informatiky a informačných technológií

Počítačové a komunikačné siete

Zadanie 2 (Návrh) – Komunikácia s využitím UDP protokolu

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 znovu vyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 5-20s 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. Iné knižnice a funkcie na prácu so socketmi 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 simulovať chybu prenosu odoslaním minimálne 1 chybného fragmentu pri prenose súboru (do dátovej časti 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. Pri nesprávnom doručení fragmentu vyžiada znovu poslať poškodené dáta.
8. Možnosť odoslať 2MB súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor, pričom používateľ zadáva iba cestu k adresáru kde má byť uložený.

Odovzdáva sa:

1. Návrh riešenia
2. Predvedenie riešenia v súlade s prezentovaným návrhom

Program musí byť organizovaný tak, aby oba komunikujúce uzly mohli prepínať medzi funkciou vysielača a prijímača bez reštartu programu - program nemusí (ale môže) byť vysielač a prijímač súčasne. Pri predvedení riešenia je podmienkou hodnotenia schopnosť doimplementovať jednoduchú funkcionalitu na cvičení.

Analýza

TCP (transmission control protocol)

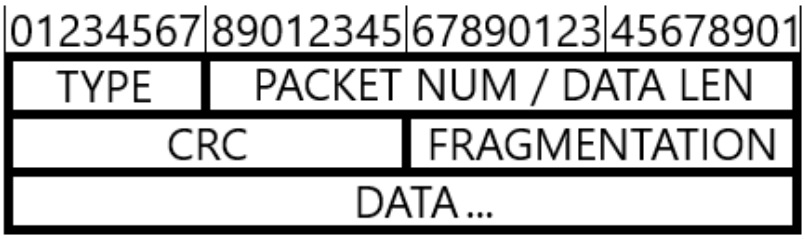
TCP je spojovo orientovaný protokol ktorý operuje v transportnej vrstve a ktorý sa v porovnaní s **UDP** vyznačuje spoľahlivosťou. To znamená že sa používa pri prenose dát kde je potrebná spoľahlivosť keďže sa uisťuje ci transfer dát prebehol úspešne ak neprebehol úspešne pošle sa paket znovu. Na začiatku spojenia sa využíva tzv. “three-way-handshake“ na zabezpečenie spoľahlivej komunikácie. Kvôli týmto vlastnostiam sa ale tento protokol stáva pomalším.

UDP (user datagram protocol)

Vyznačuje sa svojou rýchlosťou ktorú nadobúda pretože nekontroluje straty a nežiada opätovne poslanie paketu. Používa sa keď si nemôžeme dovoliť oneskorenie pri opakovanom prenose chybných paketov *(online hry streamovanie)*. Nerozdeľ od **TCP** nie je potrebne nadviazať “three-way-handshake“. Taktiež podporuje broadcast a multicast. Jedna sprava v **UDP** sa nazýva datagram a jeden datagram môže mat maximálnu veľkosť ***65 535B*** - *hlavička IP* ***(20B)*** - *veľkosť hlavičky UDP* ***(8B)*** - *vlastná hlavička* ***(8B)***. Nato aby sa správa nedelila na transportnej vrstve môže mať maximálnu veľkosť **65 499B**. Pokiaľ na linkovej vrstve nechceme dáta fragmentovať, dáta paketu musia mať max. veľkosť ***1500B*** - ***20B*** *(IP hlavička)* - ***8B*** *(UDP hlavička)* – ***8B*** *(Moja hlavička)* = ***1464B***

Návrh

Vlastná hlavička



Pole **TYPE** o veľkosti **1B** bude vyjadrovať, o aký typ paketu *(správy)* ide

Klientova strana bude používať označenia správ:

0 - Začatie komunikácie

1 - keep alive paket

2 - informačný paket – textová správa

3 - informačný paket – súbor

4 - informácie o súbore (názov, typ, ...)

5 - dáta

Serverová strana bude používať označenia správ:

6 - Potvrdenie začatia komunikácie

7 - Potvrdenie prijatia informačného paketu

8 - Potvrdenie úspešného prijatia paketu

9 - Potvrdenie neúspešného prijatia paketu

10 - Potvrdenie úspešného prijatia všetkých informačných paketov o súbore

11 - Potvrdenie úspešného prijatia všetkých paketov

Pole **PACKET NUM / DATA LEN** bude vyjadrovať, buď poradie paketu, ktorý je posielaný alebo očakávaný. Alebo celkový počet paketov, ktoré budú posielané. Keďže v zadní je uvedené, že máme vedieť odoslať 2MB súbor, čo je cca 2 milióny bajtov. Pri fragmentácií nastavenej na 1B by sa teda muselo posielať približne 2 milióny paketov. Na to, aby som vedel takéto číslo reprezentovať potrebujem min **3B**.

Pole **CRC** o veľkosti **2B** bude obsahovať zvyšok *(výsledok)* po CRC metóde.

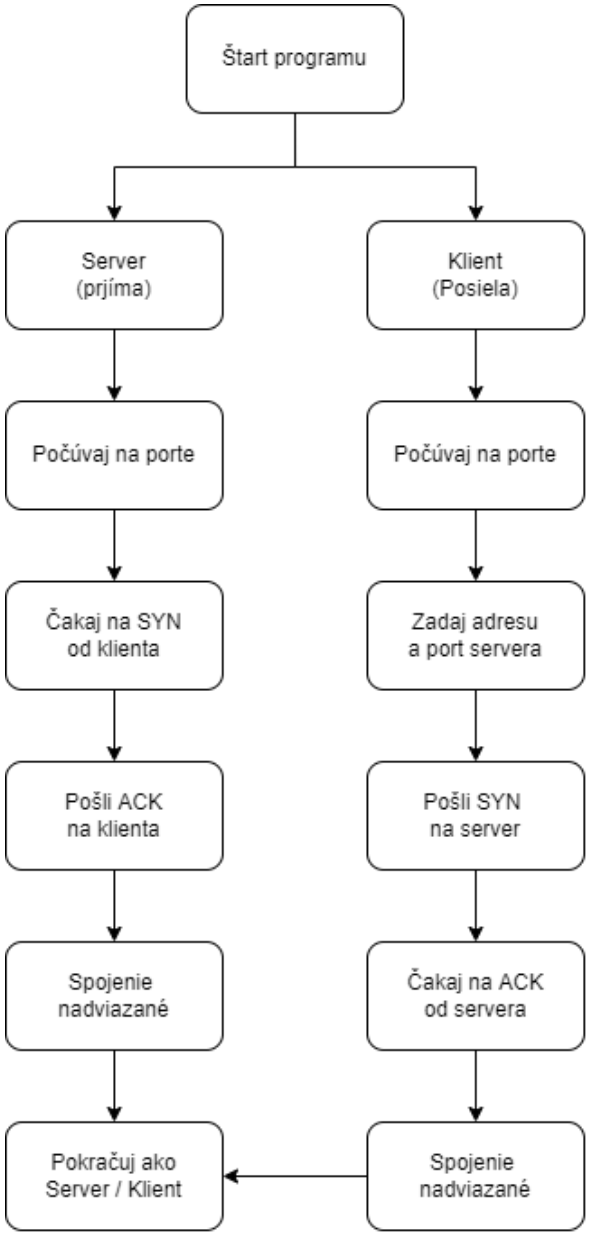
Pole **FRAGMENTATION** o veľkosti **2B** bude obsahovať informáciu o veľkosti nastavenej fragmentácií paketov

Pole **DATA** bude obsahovať samotné odosielané dáta. Pre dáta po odčítaní všedných hlavičiek ostáva maximálna veľkosť **1464B** *(1500B - 20B (IP hlavička) - 8B (UDP hlavička) – 8B (Moja hlavička)).*

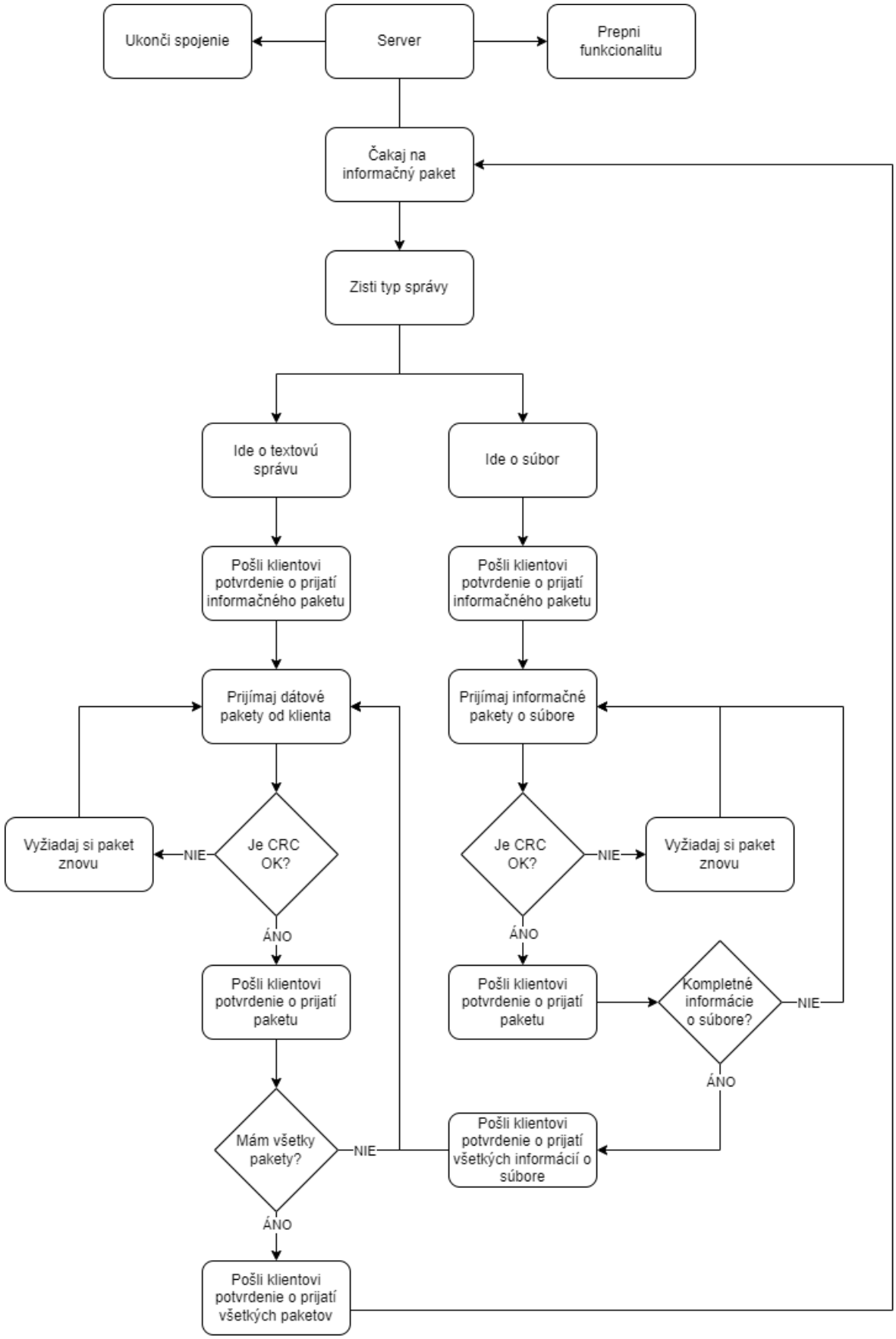
Diagramy spracovávania komunikácie

V nasledujúcich diagramoch je znázornené, ako by mal môj program fungovať. Celkové fungovanie programu je rozdelené do 3 diagramov (nadviazanie spojenia medzi klientom a serverom, spracovanie komunikácie na serveri, spracovanie komunikácie na klientovi). Diagramy nemusia byť úplne vo finálnej podobe a pri záverečnom odovzdaní sa môžu trochu líšiť.

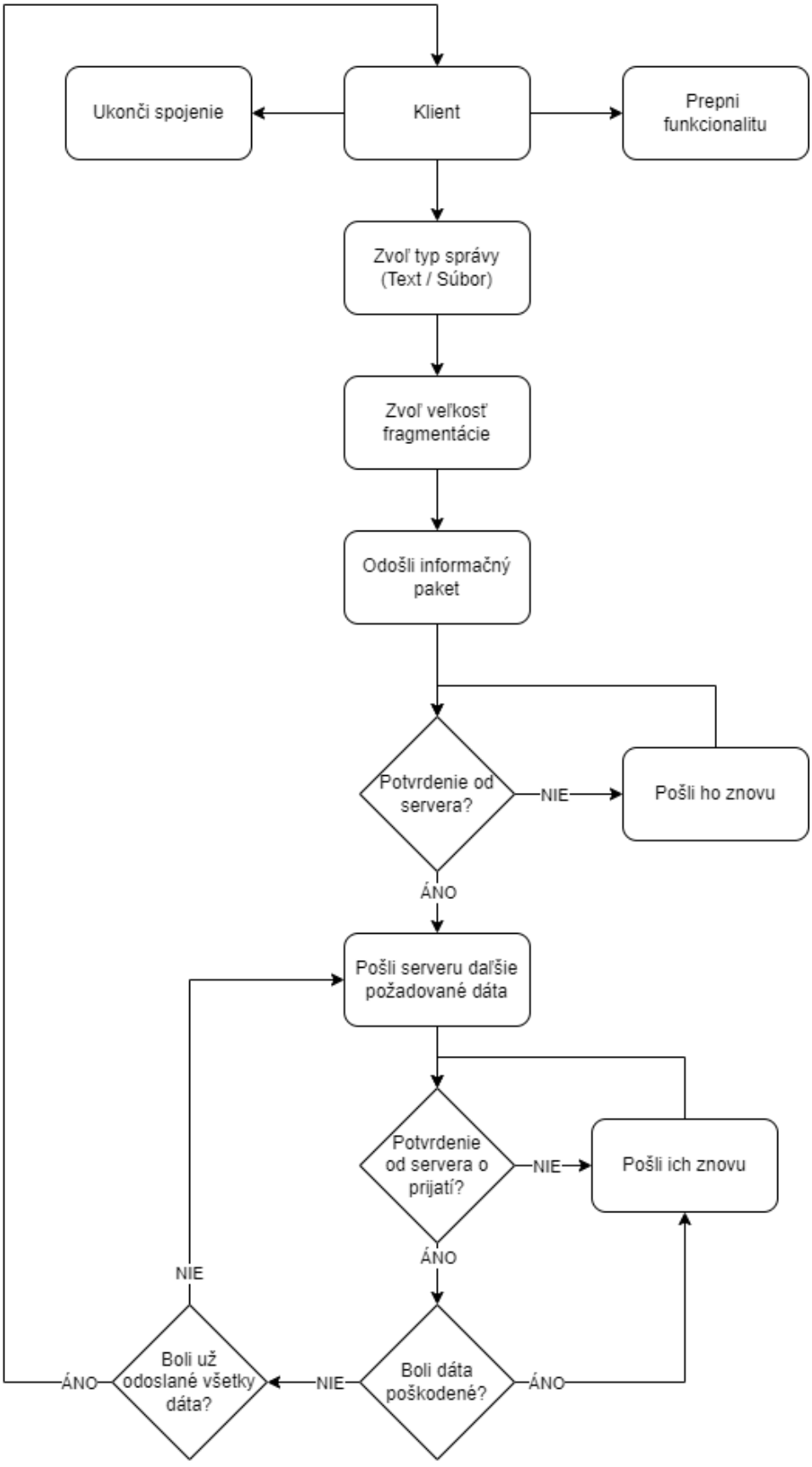
Nadviazanie spojenia medzi klientom a serverom:



Spracovanie komunikácie na serveri:



Spracovanie komunikácie na klientovi:



ARQ metóda

Pre môj návrh som si vybral Stop and Wait ARQ metódu. Celá táto metóda spočíva v tom, že klient zakaždým pošle iba jeden dátový paket. Následne čaká na potvrdenie prijatia paketu zo strany servera.

- Pokiaľ od servera obdrží správu o **neúspešnom** prijatí paketu (poškodený paket), tak mu znovu pošle ten istý.
- Pokiaľ od servera obdrží správu o **úspešnom** prijatí paketu, tak pošle v poradí ďalší dátový paket.
- Pokiaľ od servera nedostane žiadnu správu, tak sa daný paket pokúsi poslať ešte N-krát. Ak aj napriek tomu nedostane žiadnu odpoveď, tak spojenie ukončí.

Kontrola chýb

Na kontrolu chybných dát ako checksum algoritmus asi použijem CRC *(Cyclic Redundancy Check Code)* metódu s pevne zadaným polynómom CRC-16 (x^16 + x^12 + x^5 + 1).

U klienta sa zakaždým pred odoslaním paketu vyráta z odosielaných dát zvyšok po delení týmto polynómom, ktorého veľkosť budú **2B**.

Server si potom overí chybnosť tým, že znovu vydelí dáta týmto polynómom, ale pri delení sa za dáta nepridajú nuly, ale namiesto núl sa pridá vypočítaný zvyšok po delení *(CRC)*, ktorý sa nachádza v hlavičke.

Ak je po delení výsledok rovný nule, tak boli dáta prenesené bezchybne. Pri inom výsledku boli dáta pri prenose nejako poškodené, preto server odošle paket o chybnom prijatí dát a klient mu zase odošle ten istý paket znovu.

Udržiavanie spojenia

Pre udržanie spojenia bude potrebné, aby po nadviazaní spojenia medzi klientom serverom klient v pravidelných intervaloch posielal serveru tzv. keep alive paket. Na serveri bude spustený časovač, po ktorého uplynutí sa spojenie preruší. Pokiaľ však server pred uplynutím časovača obdrží keep alive paket, časovač sa zresetuje a spojenie sa nepreruší.