VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SÍŤOVÁ VRSTVA TCP/IP PRO FPGA

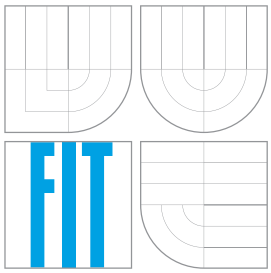
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL KEKELY

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SÍŤOVÁ VRSTVA TCP/IP PRO FPGA

TCP/IP LAYER FOR FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL KEKELY
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL KAJAN
SUPERVISOR

BRNO 2014

Abstrakt

Bakalářská práce se zabývá návrhem a hardvérovou implementací síťové komunikace s využitím síťových protokolú TCP a IP. Cílem práce je navrhnout a implementovat jednotku schopnou takové komunikace s využitím výše uvedených protokolů v FPGA a taktéž tuhle jednotku testovat a verifikovat. Výsledek práce má poskytnout možnost komunikace po síti hardverovým zařízením, které nemají přístup k téhle funkcionalitě ve svém softvérovém vybavení.

Abstract

This bachelor thesis deals with the design and implementation of network communication using network protocols TCP and IP in hardware. Goal of this thesis is to design and implement unit capable of this sort of network communication using aforementioned protocols in FPGA and also to test and verify it. Outcome should give hardware devices, which don't have access to suitable software, the opportunity to communicate using computer networks.

Klíčová slova

TCP, IP, ISO/OSI, FPGA, sítě, hardvér, síťové modely, simulace, verifikace

Keywords

TCP, IP, ISO/OSI, FPGA, networks, hardware, network models, simulation, verification

Citace

Michal Kekely: Síťová vrstva TCP/IP pro FPGA, bakalářská práce, Brno, FIT VUT v Brně, 2014

Síťová vrstva TCP/IP pro FPGA

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Michala Kajana.

.....
Michal Kekely
20. mája 2014

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

V dnešnej dobe sa čoraz viac rozširujú vstavané systémy (anglicky embedded systems). Ani odvetvie sietí sa tomuto rozširovaniu vstavaných systémov nevyhýba. Toto odvetvie, ako každé iné, sa vyvíja závratnou rýchlosťou. Zvyšuje sa počet používateľov počítačových sietí, zvyšuje sa čas strávený jednotlivými používateľmi na sieti a v neposlednom rade sa zvyšuje rýchlosť prenosu dát.

S týmto rozmachom prichádzajú aj nebezpečenstvá. Je preto potrebné vedieť čo najrýchlejšie odhaliť potenciálne nebezpečenstvá alebo nezákonnu činnosť páchanú po sieti. Toho sme schopní dosiahnuť monitorovaním a analýzou sieťovej komunikácie.

Množstvo dát, ktoré je potrebné analyzovať, je však enormné. Klasický prístup riešenia problému softvérovo nám poskytuje vysokú flexibilitu a jednoduchosť, avšak na úkor výkonu a efektivity. Alternatívou je využitie hardvérovej akcelerácie, kedy niektoré kritické časti výpočtu prenesieme do špeciálne navrhnutých hardvérových jednotiek. Tieto hardvérové jednotky môžeme potom navrhnúť s ohľadom na funkcionalitu, ktorú majú zabezpečovať, a teda ich môžeme výrazne optimalizovať a dosiahnuť tak až niekoľkonásobného zrýchlenia. Sme teda schopní analyzovať väčšie množstvo dát v kratšom čase.

Samotná sieťová komunikácia je jednou z funkcií, ktorú je v niektorých prípadoch vhodné riešiť hardvérovo. Je neefektívne exportovať dáta, ktoré sú spracovávané hardvérovo naspäť do softvéru, ktorý by sa postaral o ich odoslanie inému zariadeniu. Pri hardvérovom riešení sieťovej komunikácie odpadáva potreba mať výkonný procesor, prípadne sa výkon procesoru môže využiť na inú činnosť. Zároveň nie je riešenie limitované rýchlosťou prenosu na zbernici medzi softvérom a hardvérom.

Cieľom tejto bakalárskej práce je navrhnúť, implementovať a otestovať hardvérovú jednotku schopnú sieťovej komunikácie pomocou sieťových protokolov TCP a IP. Toto riešenie má v sebe zahŕňať všetky povinné súčasti týchto protokolov a zároveň byť schopné vysporiadať sa s akýmikoľvek situáciami, ktoré podľa špecifikácie týchto protokolov môžu nastať. Dôraz je kladený aj na rýchlosť tohto riešenia, keďže sa predpokladá nasadenie na živej sieti s linkami s rýchlosťou až jedného gigabitu.

Práca je rozdelená do 6 kapitol. Kapitola 2 popisuje princípy fungovania dnešných počítačových sietí s ohľadom práve na sieťovú komunikáciu, rozbor protokolov TCP a IP a ich súčastí. Kapitola 3 obsahuje návrh samotnej hardvérovej jednotky, ktorý vychádza práve z rozboru sieťových protokolov TCP a IP. Táto kapitola navyše podrobnejšie rozoberá niektoré implementačné detaily jednotlivých problémových častí riešenia. Kapitola 4 sa zameriava na testovanie a verifikáciu implementácie. Kapitola 5 analyzuje dosiahnuté výsledky a využiteľnosť riešenia. Nakoniec v kapitole 6 je zhrnutý obsah práce a dosiahnuté výsledky.

Kapitola 2

Teoretický rozbor

Táto kapitola zahrňuje základné teoretické znalosti, ktoré tvoria základ pre následný návrh a implementáciu. Rozbor začína popisom princípu počítačových sietí. Nasleduje popis sieťových modelov ISO/OSI a TCP/IP. Dôraz je kladený hlavne na spôsob komunikácie, dáta a metadáta tejto komunikácie. Nakoniec sú podrobnejšie rozobrané sieťové protokoly IP a TCP, ktoré sú pre túto prácu nosné.

2.1 Počítačové siete

Informácie tejto sekcie boli čerpané hlavne zo zdroja [2]. Počítačové siete sú telekomunikačné siete, ktoré umožňujú zariadeniam výmenu dát. Prvky, ktoré sa v týchto sieťach vyskytujú, môžeme rozdeliť na dva druhy. Prvým sú koncové zariadenia, ide o zariadenia, ktoré sú buď odosielateľom správy alebo príjemcom tejto správy. Medzi koncové zariadenia patria osobné počítače, tablety, IP telefóny a mnohé iné.

Správa sa však od odosielateľa k príjemcovi musí nejakým spôsobom dostať. Pokiaľ sú priamo prepojení nenastáva problém. Akonáhle však potrebujeme prepojiť väčšie množstvo zariadení, ktoré môžu byť na rôznych miestach, nebolo by veľmi výhodné prepojiť priamo každé zariadenie s každým. Využívame preto druhú skupinu zariadení, a to zariadenia sprostredkovateľské. Ide o zariadenia, ktoré zapezpečujú smerovanie a doručenie dát do správneho koncového zariadenia. Do tejto kategórie spadajú rozbočovače a smerovače. Tieto zariadenia sú často pre bežného užívateľa skryté a o ich existencii ani nemusí vedieť.

Všetky tieto zariadenia musia byť nejakým spôsobom prepojené. Na to slúži prenosové médium. V počítačových sieťach je týmto médiom medený alebo optický kábel, prípadne bezdrôtová technológia.

Pre úspešnú komunikáciu ďalej potrebujeme, aby si zariadenia rozumeli. Potrebujeme teda dátam a metadátam, ktoré si vymieňajú, priradiť význam. Toto je úlohou sieťových protokolov. Tieto protokoly určujú štruktúru a význam dát. Zároveň môžu poskytovať mechanizmy pre riadenie prenosu dát. Základné protokoly sú definované dvoma modelmi: modelom ISO/OSI a modelom TCP/IP.

2.1.1 Modely ISO/OSI a TCP/IP

Ide o vrstvovú abstraktnú reprezentáciu vytvorenú ako pômocka pre návrh sieťových protokolov. Vrstvy je možné vidieť na obrázku 2.1. Model OSI predstavuje formálnejší prístup. Rozdeľuje proces sieťovej komunikácie na 7 logických vrstiev, pričom každej vrstve priraďuje unikátnu funkcionalitu a rozhranie na komunikáciu s ostatnými vrstvami. Informácie

sú postupne propagované medzi vrstvami počínajúc aplikačnou na strane odosielateľa, pokračujúc naprieč všetkými vrstvami až po fyzickú, ďalej cez médium do prijímacej strany, kde je spracovaná vrstvami v opačnom poradí.

Tento spôsob oddelenia funkcionality výrazne zjednodušuje tvorbu nových protokolov či zariadení a ich začlenenie do existujúcej siete. Pokiaľ napríklad zmeníme prenosové médium, postačí zmeniť protokol najnižšej vrstvy.

V praxi prechod informácie jednotlivými vrstvami znamená pridanie riadiacich informácií danej vrstvy vo forme hlavičky, prípadne päty. Hovoríme o zabaľovaní dát.

V dnešnej dobe sa využíva hlavne TCP/IP model. Tento model má len 4 vrstvy:

1. Linková vrstva tvorí rozhranie hardvérových zariadení a prenosových médií. Ide o najnižšiu vrstvu modelu, ktorá zabezpečuje hardvérovú nezávislosť celého modelu. Ďalej zabezpečuje prípravu dát na prenos a samotný prenos dát po prenosovom médiu. Protokoly tejto vrstvy sú napríklad Ethernet alebo Token Ring. Dátové entity na tejto vrstve sa nazývajú rámce (anglicky frame).

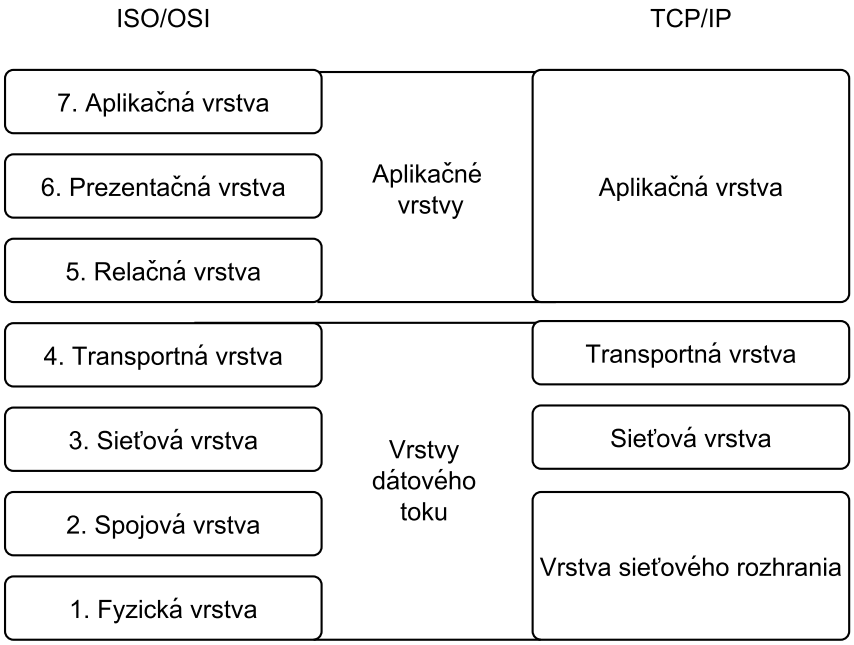
2. Sieťová vrstva sa stará o zasielanie dát naprieč potenciálne viacerými sieťami. Tento proces sa nazýva smerovanie. Táto vrstva má dve základné funkcie. Adresuje a identifikuje zariadenia v sieti pomocou hierarchického systému IP adries a smeruje dáta zo zdroja k cieľu tým, že ich preposiela vždy na ďalší smerovač, ktorý je bližšie pri cieli. Táto vrstva však poskytuje len nespoľahlivý dátový prenos. Protokolom tejto vrstvy je protokol IP a ďalšie protokoly, ktoré funkčnosť protokolu IP podporujú, ako ICMP, IGMP a smerovacie protokoly EIGRP, OSPF a RIP (tieto protokoly sú často priraďované do akejsi medzivrstvy medzi vrstvou sieťovou a transportnou). Dátové entity tejto vrstvy nazývame pakety (anglicky packet).

3. Transportná vrstva slúži na základnú komunikáciu medzi aplikáciami na koncových zariadeniach. Obsahuje teda mechanizmus pre adresovanie jednotlivých aplikácií bežiacich na jednom zariadení pomocou čísel portov. Ďalej definuje mechanizmy pre kontrolu zahltenia a toku, kontrolu správnosti prijatých dát a segmentáciu. Úlohou tejto vrstvy je prenos správ nezávisle na prenosovom médiu či type siete. Protokoly transportnej vrstvy delíme na spojovo-orientované (TCP, SCTP), ktoré vytvárajú a udržujú spojenie a zabezpečujú spoľahlivosť príjmu dát a bezspojové (UDP), ktoré spojenie nenadväzujú. Dátové entity transportnej vrstvy sú segmenty (anglicky segment).

4. Aplikačná vrstva TCP/IP zodpovedá 3 najvyšším vrstvám OSI modelu (aplikačná, prezenčná, relačná). Protokoly tejto vrstvy často využívajú protokoly nižších vrstiev ako čierne skrinky. Často sú spájané s aplikáciami typu klient-server, pričom pre určité aplikácie na strane serveru sú vyhradené príslušné čísla portov (napríklad HTTP má port 80, Telnet má port 23). Porty slúžia na oddelenie jednotlivých komunikácií a typov týchto komunikácií prebiehajúcich na jednom zariadení. Procesy tejto vrstvy sú zavislé na konkrétnej aplikácii.

2.2 Protokol IP

Tento protokol je definovaný v [8]. Ide o protokol sieťovej vrstvy, ktorý zabezpečuje len základnú funkčnosť definovanú touto vrstvou, preto neposkytuje žiadne mechanizmy pre



Obrázok 2.1: Modely ISO/OSI a TCP/IP.

riadenie toku či zaručenie spoľahlivého prenosu. Momentálne je to najrozšírenejší protokol na výmenu dát v sieťach s prepínaním paketov.

Protokol nenadväzuje spojenia a príjemca, ktorý ani nemusí existovať, sa dozvie o komunikácii až v momente príchodu paketu. Protokol sa vyznačuje doručovaním s najväčším úsilím (anglicky best-effort delivery), kedy sa príslušné smerovače na ceste od zdroja snažia nájsť najlepšiu cestu k príjemcovi (táto cesta je daná smerovacími protokolmi a protokol IP tomuto procesu podlieha). Momentálne súbežne existujú dve verzie protokolu. Konkrétne verzie IPv4 a IPv6. Postupne sa prechádza na protokol IPv6 z dôvodu vyčerpania adresového priestoru IPv4, ktorý je obmedzený a výrazne menší ako adresový priestor IPv6. Adresa IPv4 je 32-bitové číslo, ktoré sa z dôvodu čitateľnosti rozdelí na 4 8-bitové čísla, ktoré sa následne zapíšu v desiatkovej sústave oddelené bodkami (napríklad 192.168.1.1). Podobne IPv6 adresa je 128-bitové číslo rozdelené na 8 16-bitových čísel, ktoré sa zapisujú v šestnástkovej sústave oddelené dvojbodkami.

2.2.1 Formát protokolu

Základný blok dát, s ktorým sa na úrovni IP pracuje, sa nazýva paket. Paket v sebe zahŕňa hlavičku a samotné dáta (anglicky payload).

IPv4 hlavička

Obrázok 2.2 zobrazuje štruktúru hlavičky IPv4. Hlavička IPv4 obsahuje nasledujúce položky:

Verzia
4-bitová hodnota určujúca verziu IP, pre IPv4 je rovná 4.

Dĺžka hlavičky (anglicky internet header length, skrátene IHL)
Ide o 4-bitové číslo. Udáva veľkosť hlavičky v 4 bajtových blokoch. Základná veľkosť hlavičky je 20 bajtov, avšak hlavička môže obsahovať nepovinné pole *OPTIONS*.

Typ služby
Toto pole je využité na dve položky a to DSCP a ECN. DSCP obsahuje číslo využívané diferencovanými službami. ECN je nepovinné pole, ktoré slúži na výmenu informácie o zahltení medzi dvoma koncovými bodmi.

Celková dĺžka
Toto 16 bitové pole definuje celkovú dĺžku paketu vrátane hlavičky v bajtoch. Minimálne je táto hodnota 20 (20 bajtová hlavička a žiadne dáta).

Identifikácia
Táto položka sa využíva na identifikáciu fragmentov, ktoré patria do rovnakého paketu (pokiaľ došlo k fragmentácii paketu).

Príznaky

- rezervovaný - musí byť 0
- DF - príznak, ktorý zakazuje fragmentáciu paketu, pokiaľ je nutné paket fragmentovať a tento príznak je nastavený, paket sa zahodí
- MF - tento príznak je nastavený pri všetkých fragmentoch paketu okrem posledného

Offset fragmentu
Toto 13-bitové pole určuje posunutie fragmentu vzhľadom na začiatok pôvodného nefragmentovaného paketu. Merané v 8-bajtových blokoch.

Životnosť paketu (anglicky time to live, skrátene TTL)
Obsah tejto položky určuje životnosť paketu. V praxi udáva počet skokov medzi smerovačmi, ktoré paket prežije (každý smerovač znižuje TTL o 1, v prípade že je TTL nulové, je paket zahodený).

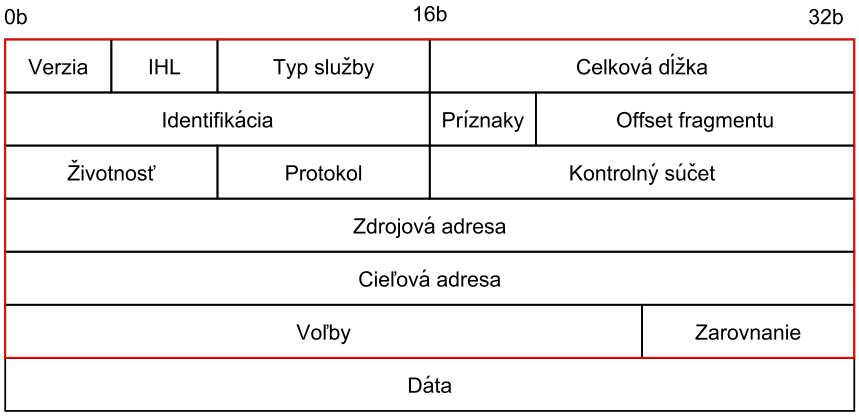
Protokol
Určenie protokolu použitého v dátovej časti paketu.

Kontrolný súčet
Kontrolný súčet, ktorý sa počíta ako jednotkový doplnok súčtu v jednotkovej aritmetike 16-bitových blokov IP hlavičky. Pokiaľ sa kontrolný súčet v tomto poli nezhoduje s prepočítaným kontrolným súčtom, paket sa zahodí.

Zdrojová adresa
Zdrojová IPv4 adresa identifikujúca odosielateľa paketu. Táto adresa sa môže meniť pri preklade adries (anglicky network address translation, skrátene NAT).

Cieľová adresa
Cieľová IPv4 adresa identifikujúca príjemcu paketu. Táto adresa sa môže meniť pri preklade adries.

Voľby
Nepovinná položka, ktorá sa môže použiť na rozširujúce voľby.



Obrázok 2.2: IPv4 hlavička.

IPv6 hlavička

Obrázok 2.3 zobrazuje štruktúru hlavičky IPv6. Hlavička IPv6 obsahuje nasledujúce položky:

Verzia
: 4-bitová hodnota určujúca verziu IP, pre IPv6 je rovná 6.

Traffic Class
: Podľa novej špecifikácie je toto pole využité na dve položky a to DSCP a ECN. DSCP obsahuje číslo využívané diferencovanými službami. ECN je nepovinné pole, ktoré slúži na výmenu informácie o zahltení medzi dvoma koncovými bodmi.

Označenie toku
: Pôvodne vytvorené pre procesy reálneho času. Momentálne nenulová hodnota znamená, že pakety by mali byť smerované rovnakou cestou, aby dorazili do cieľa v rovnakom poradí. V budúcnosti by sa mohlo využívať na detekciu podvrhnutých paketov.

Dĺžka dát
: Toto 16-bitové pole definuje dĺžku dátovej časti paketu v bajtoch.

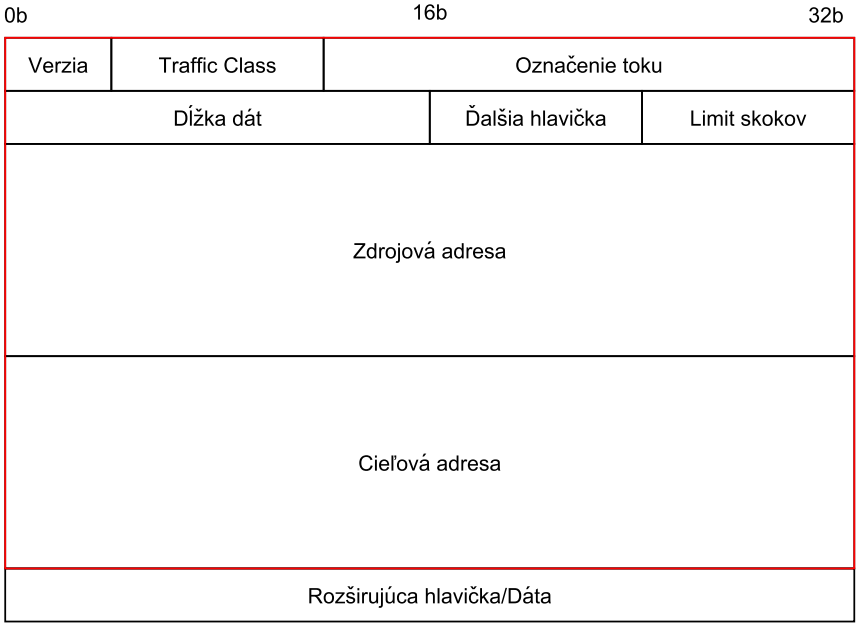
Ďalšia hlavička
: Určuje typ nasledujúcej hlavičky. IPv6 podporuje rozširujúce hlavičky, ktoré rozširujú základnú IPv6 hlavičku. V prípade, že nie je použitá žiadna rozširujúca hlavička, toto pole odkazuje na hlavičku protokolu transportnej vrstvy.

Limit skokov
: Obsah tejto položky určuje životnosť paketu. V praxi udáva počet skokov medzi smerovačmi, ktoré paket prežije (každý smerovač znižuje hodnotu o 1, v prípade že je hodnota nulová, je paket zahodený).

Zdrojová adresa
: Zdrojová IPv6 adresa identifikujúca odosielateľa paketu. Táto adresa sa môže meniť pri preklade adries.

Cieľová adresa
Cieľová IPv6 adresa identifikujúca príjemcu paketu. Táto adresa sa môže meniť pri preklade adries.

Voľby
Nepovinná položka, ktorá sa môže použiť na rozširujúce voľby. Vo verzii IPv6 sú tieto voľby riešené pomocou rozširujúcich hlavičiek.



Obrázok 2.3: IPv6 hlavička.

2.3 Protokol TCP

Protokol TCP je protokolom transportnej vrstvy sieťového modelu ISO/OSI. Poskytuje spoľahlivý spojovo-orientovaný prenos medzi dvoma koncovými bodmi v sieti (vytvára logické spojenie týchto dvoch bodov). Nutné požiadavky, ktoré musí implementácia daného sieťového protokolu obsahovať, je možné nájsť v dokumente [1].

2.3.1 Funkcia protokolu TCP

Keďže ide o protokol transportnej vrstvy, tvorí prechodnú úroveň medzi aplikáciou a protokolom sieťovej vrstvy (napríklad IP).

Protokol IP neposkytuje žiadne prostriedky na detekciu ani opravu viacnásobného prijatia paketu, straty paketu alebo prijatia paketov v nesprávnom poradí. Protokol TCP dokáže tieto problémy detekovať a prípadne dáta znovu poslať alebo usporiadať. Zároveň dokáže riadiť tok dát a tým niektorým týmto problémom predchádzať.

TCP sa v dnešnej dobe vďaka svojej spoľahlivosti využíva hlavne v aplikáciách, pri ktorých vyžadujeme správnosť prijatia dát a sme schopní tolerovať zvýšené oneskorenie

prijatých dát. Nie je teda príliš vhodné využívať tento protokol pri aplikáciách, ktoré pracujú v reálnom čase, ako napríklad IP telefónia alebo streaming.

2.3.2 Formát protokolu

Základný blok dát, s ktorým sa na úrovni TCP pracuje, sa nazýva segment. Segment zahŕňa hlavičku a samotné dáta (anglicky payload). Obrázok 2.4 zobrazuje štruktúru hlavičky. Hlavička obsahuje nasledujúce položky:

Zdrojový port
Obsahuje 16-bitové číslo adresujúce zdrojový port odosielajúcej aplikácie.

Cieľový port
Obsahuje 16-bitové číslo adresujúce cieľový port prijímacej aplikácie.

Sekvenčné číslo
Identifikuje prvý bajt v danom segmente. Čísluje sa každý odoslaný bajt dát. Toto číslo je 32-bitové a pri ustanovení spojenia sa náhodne vygeneruje počiatočné sekvenčné číslo.

Potvrdzovacie číslo
Slúži na potvrdenie príjmu dát. Prijímacia strana nastavuje túto hodnotu na sekvenčné číslo nasledujúceho očakávaného bajtu (teda sekvenčné číslo posledného prijatého bajtu zvýšené o 1). Táto hodnota sa ignoruje v prípade, že nie je nastavený príznak ACK.

Dĺžka hlavičky
Ide o 4-bitové číslo. Udáva veľkosť hlavičky v 4-bajtových blokoch. Základná veľkosť hlavičky je 20 bajtov, avšak hlavička môže obsahovať nepovinné pole *OPTIONS*.

Rezervované bity
Nepoužité bity.

Príznaky TCP (flagy)

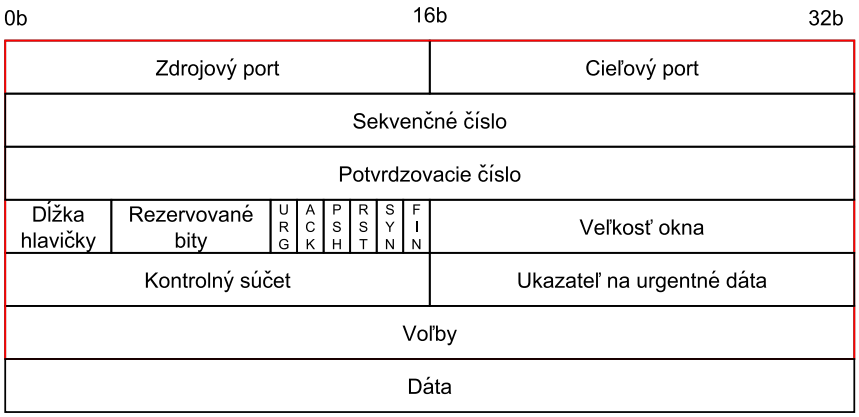
- URG - signalizuje platný ukazovateľ na urgentné dáta
- ACK - signalizuje platné potvrdzovacie číslo
- PSH - tzv. push flag, prijaté dáta by mali byť doručené čo najskôr a nemali by sa zdržovať v prijímacom fronte, dnes už bezvýznamné
- RST - reset spojenia
- SYN - používa sa počas vytvorenia spojenia na synchronizáciu sekvenčných čísel
- FIN - slúži na ukončenie spojenia, odosielateľ signalizuje, že už nebude odosielať dáta

Veľkosť okna
Predstavuje množstvo dát, ktoré je prijímacia strana schopná prijať. V prípade potreby zvýšiť veľkosť okna nad hodnotu danú 16 bitmi je možné použiť položku *OPTIONS*.

Kontrolný súčet
: Kontrolný súčet, ktorý sa počíta ako jednotkový doplnok z TCP segmentu (hlavička a dáta doplnené nulami na párny počet bajtov) a pseudohlavičky IP datagramu, ktorú tvorí zdrojová a cieľová IP adresa, rezervované pole 8 nulových bitov, typ protokolu a dĺžka TCP segmentu.

Ukazovateľ na urgentné dáta
: Udáva offset, ktorý sa pričíta k hodnote sekvenčného čísla a ukazuje na koniec urgentných dát. Platný len v prípade nastavenia URG príznaku. V súčasnosti sa nepoužíva.

Voľby
: Nepovinná položka. Pôvodne, podľa špecifikácie [4], boli len 3 rôzne voľby. Neskoršia špecifikácia [5] pridáva ďalšie. Každá voľba začína jednobajtovým políčkom, ktoré udáva typ voľby.



Obrázok 2.4: Hlavička TCP segmentu.

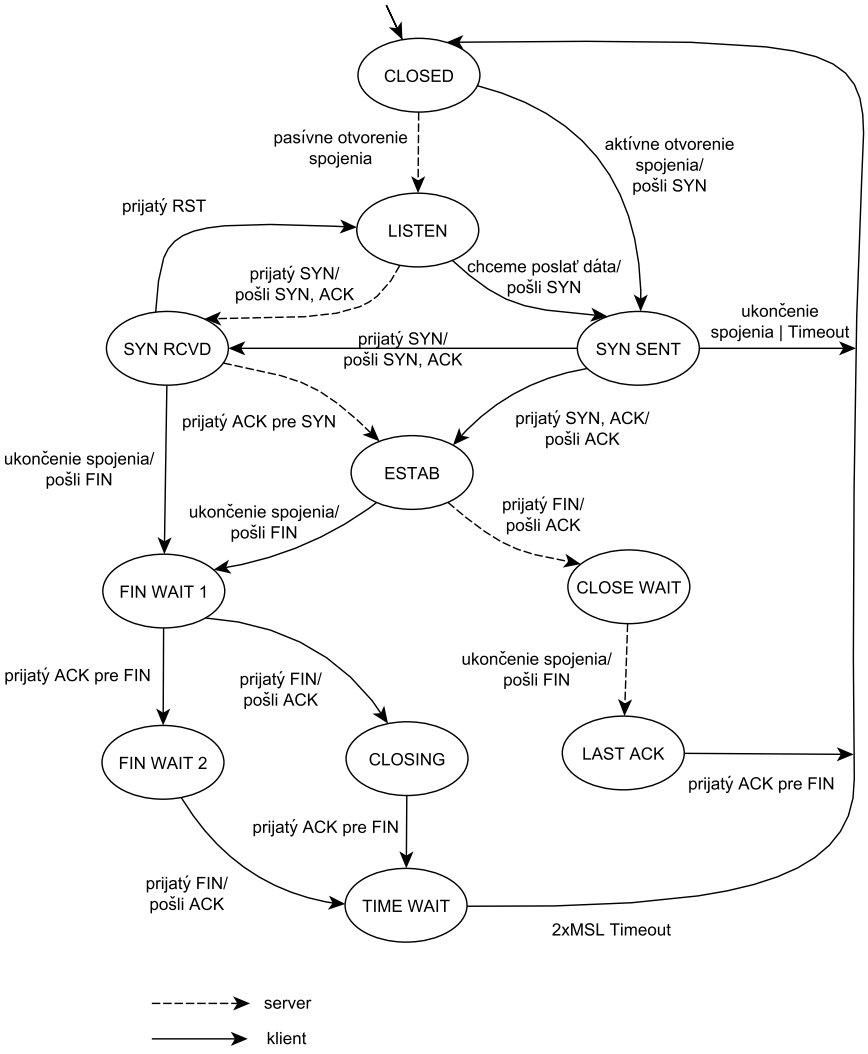
2.3.3 TCP stavový diagram

Stavový diagram komunikácie pomocou protokolu TCP je možné vidieť na obrázku 2.5. Diagram je rovnaký pre obe komunikujúce strany a možno ho rozdeliť na dve samostatné časti pre klienta a pre server.

Stav *ESTAB* reprezentuje stav, v ktorom dochádza k prenosu samotných dát. Stav *CLOSED* reprezentuje situáciu pred samotným nadviazaním spojenia. Stavy *LISTEN*, *SYN SENT* a *SYN RCVD* tvoria trojfázové nadviazanie spojenia. Ukončenie spojenia sa realizuje v stavoch *FIN WAIT 1*, *FIN WAIT 2*, *CLOSING* a *TIME WAIT* pre aktívne ukončenie spojenia, respektíve v stavoch *CLOSE WAIT* a *LAST ACK* pre pasívne ukončenie spojenia.

Hodnota *MSL* reprezentuje dobu, počas ktorej môže segment existovať v sieti. Jeho hodnota je špecifikovaná na 2 minúty. Po čase rovnom dvojnásobku tejto hodnoty taktiež dôjde k ukončeniu spojenia a dané porty je možno znovu použiť. Tento mechanizmus taktiež zabraňuje tomu, aby po vytvorení nového spojenia, ktoré je identifikované rovnako ako nejaké staršie spojenie, nedošlo k akceptovaniu segmentov starého spojenia novým

spojením. Po štarte stanice sa teda čaká aspoň po dobu *MSL*, kým je možné vytvoriť nové spojenia.



Obrázok 2.5: Stavový diagram TCP.

2.3.4 Vytvorenie TCP spojenia

TCP je spojovo-orientovaný protokol. V praxi to znamená, že pred samotným prenosom dát je potrebné vytvoriť spojenie medzi komunikujúcimi bodmi. K tomuto slúži mechanizmus trojfázového nadviazania spojenia (tzv. 3-way hadshake). Počas tohto procesu si komunikujúce strany vymenia 3 TCP segmenty:

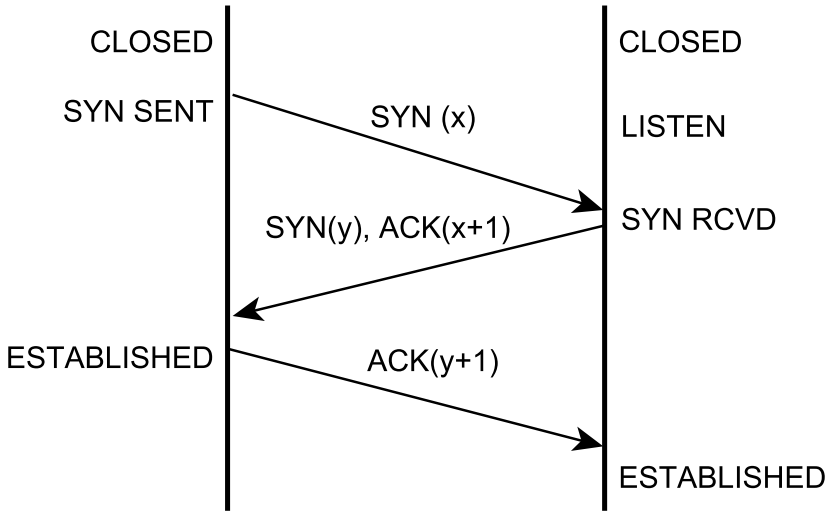
1. Strana iniciujúca spojenie zašle segment s nastaveným SYN príznakom a s náhodným sekvenčným číslom. Sekvenčné číslo sa volí náhodne z bezpečnostných dôvodov.

2. Druhá strana odpovie zaslaním segmentu s nastavenými SYN a ACK príznakmi, vlastným počiatočným sekvenčným číslom a potvrdzovacím číslom rovným prijatému sekvenčnému číslu zvýšenému o 1.

3. Iniciujúca strana odpovie segmentom s nastaveným ACK príznakom a potvrdzovacím číslom zvýšením o 1.

Priebeh nadviazania spojenia je možné vidieť na obrázku 2.6.

V časti TCP hlavičky *OPTIONS* si môžu komunikujúce strany vymeniť hodnotu položky *Maximum Segment Size (MSS)*. Hodnota tejto položky udáva maximálny počet bajtov, ktorý sa môže danej strane zaslať v jednom segmente. Implicitná hodnota je 536 bajtov, obvykle sa však používajú násobky 512 bajtov. Pre komunikáciu na lokálnej sieti typu Ethernet sa používa hodnota 1460 bajtov.



Obrázok 2.6: Trojfázové nadviazanie spojenia (3-way hadshake).

2.3.5 Ukončenie TCP spojenia

Po odoslaní všetkých dát by malo byť spojenie správne ukončené. Ukončenie spojenia prebieha samostatne pre každý smer prenosu. Po ukončení jedného smeru je teda ešte možný prenos dát v opačnom smere. Ukončenie spojenia môže začať ľubovolná strana zaslaním segmentu s nastaveným FIN príznakom. Na strane, ktorá začala ukončenie spojenia, prebieha aktívne ukončenie spojenia, na strane druhej pasívne ukončenie spojenia. Výmena segmentov je podobná ako pri vytvorení spojenia. Počas ukončovania spojenia si komunikujúce strany vymenia 4 TCP segmenty:

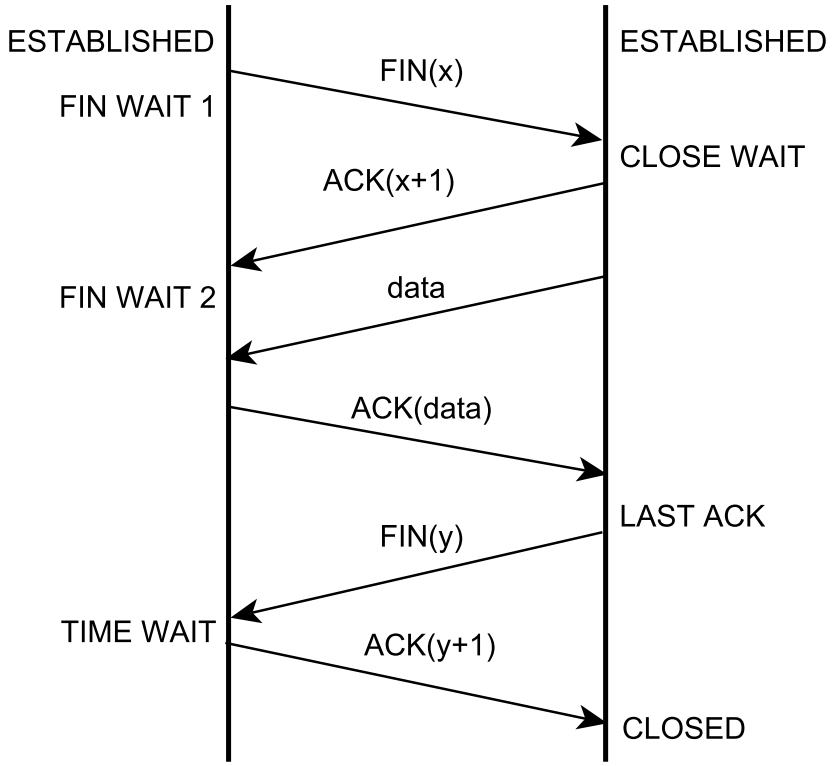
1. Strana iniciujúca ukončenie zašle segment s nastaveným FIN príznakom a sekvenčným číslom, ktoré má nasledovať.

2. Druhá strana odpovie zaslaním segmentu s nastavenými ACK príznakom pre dané sekvenčné číslo. V tomto momente končí prenos v smere z iniciujúcej strany.

3. Druhá strana už tiež nemá čo odoslať, preto zašle segment s nastaveným FIN príznakom.

4. Iniciujúca strana odpovie zaslaním segmentu s nastaveným ACK príznakom. Spojenie je ukončené.

Priebeh ukončovania spojenia je zobrazený na obrázku 2.7.



Obrázok 2.7: Priebeh ukončenia TCP spojenia.

2.3.6 Prerušenie TCP spojenia

K prerušeniu TCP spojenia môže dôjsť vo viacerých prípadoch:

- Snaha o spojenie s neexistujúcim portom - segment s nastaveným RST príznakom so sekvenčným číslom 0 sa vracia hneď po zaslaní úvodného segmentu s nastaveným SYN príznakom.
- Vynútené prerušenie komunikácie - klient zašle serveru segment s nastaveným RST príznakom, server tento segment len príjme a upozorní aplikáciu.
- Prijatie segmentu mimo vytvorené spojenie - v prípade havárie alebo reštartu serveru síce aplikácia na serveri beží, ale nemá znalosť o predchádzajúcich vytvorených spojeniach. V prípade, že sa klient snaží zaslať ďalšie dáta, odpovedá server okamžite segmentom s nastaveným RST príznakom.

2.3.7 Prenos dát

Prenos dát je riadený posuvným oknom. Veľkosť okna ohlasuje príjemca a udáva množstvo dát, ktoré je príjemca schopný naraz prijať. Odosielateľ teda nesmie odoslať viac dát, ako je veľkosť okna. Okno samotné je tvorené dvoma hranicami. Ľavá hranica okna určuje

odoslané a potvrdené dáta, pravá hranica dáta, ktoré ešte nie je možné odoslať, pokým nebudú odoslané a potvrdené ďalšie dáta. Medzi hranicami sa nachádzajú dáta, ktoré môžu byť odoslané alebo už odoslané boli, ale neboli ešte potvrdené. Okno sa môže posúvať len vpravo (nemá zmysel posúvať ho vľavo a znovu posielať už potvrdené dáta).

Veľkosť okna sa počas prenosu môže meniť. Vhodná veľkosť sa dá vypočítať na základe priepustnosti linky a času RTT (Round Trip Time, čas, ktorý je potrebný k zaslaniu dát a následnému prijatiu potvrdenia pre tieto dáta). Špecifikujeme dve techniky pre obsluhu prenosu:

1. Oneskorené zasielanie potvrdení - po prijatí segmentu neposielame potvrdenie okamžite, ale čakáme určitý čas, aby bolo možné spolu s potvrdením zaslať naspäť aj nejaké dáta. Pokiaľ počas čakania príjmeme ďalšie segmenty, zašleme potvrdenie okamžite a spustíme čakanie pre ďalšie segmenty. Príjemca teda nemusí čakať na zaplnenie celého okna.

2. Naglov algoritmus - rozširuje predchádzajúci mechanizmus, pokiaľ sú v okne dáta, ktoré ešte príjemca nepotvrdil, nebudú malé segmenty zaslané, až pokým nedôjde k ich potvrdeniu. Ide o bloky do veľkosti MSS. Snahou TCP je odosielať dáta v čo najväčších blokoch.

Syndróm hlúpeho okna

Ide o problém, ktorý vzniká pri výmene segmentov rôznych veľkostí. Veľkosť okna príjemcu sa zmenší na minimum, na čo odosielateľ reaguje zasielaním segmentov malej veľkosti. Toto ma za následok veľmi neefektívne využitie prenosového pásma, pretože jeho veľká časť sa spotrebuje na réžiu samotného protokolu.

Časovače prenosu dát

Pri prenose dát môžu nastať rôzne, potenciálne brzdiace efekty a stavy. Aby sme takýmto nechceným stavom predchádzali, používame časovače, vypršaním ktorých dochádza k nejakej akcii, ktorá zväčša predchádza spomaleniu komunikácie.

Prvým z týchto časovačov je takzvaný perzistentný časovač (anglicky persist timer). Hodnota tohto časovača býva nastavená v rozmedzí 5 až 60 sekúnd. Pri vypršaní časovača sa odosielateľ pýta na veľkosť okna. Predchádza sa tak problému, kedy je odosielateľ brzdený malou veľkosťou okna na strane príjemcu a z dôvodu napríklad straty potvrdzovacích segmentov, ktoré v sebe môžu niesť aj zmenu veľkosti okna sa nemá odosielateľ ako dozvedieť túto zmenu veľkosti okna. Pokiaľ je veľkosť okna príjemcu nenulová, ale zároveň nižšia ako MSS, odošle sa segment aj takto malých dát a čaká sa aspoň 5 sekúnd. V prípade, že odosielateľ čaká zvýšenie veľkosti okna z veľkosti nula, zasiela sa segment o veľkosti 1 bajtu po vypršaní časovača RTO. Príjemca tento segment môže, ale nemusí prijať a na základe toho vygeneruje potvrdzovací segmet. Doba medzi dvoma pokusmi odoslať tento 1-bajtový paket sa exponenciálne zvyšuje.

Ďalším časovačom je časovač RTO (anglicky retransmission timeout). Vypršaním tohoto časovača dochádza k znovuposlaniu už odoslaných, ale zatiaľ nepotvrdených dát. Hodnota RTO je premenlivá, pretože závisí od nestálych podmienok na sieti. Nastavuje sa dynamicky počas celého spojenia na základe premenlivej hodnoty RTT.

Pri znovuposielaní dát sa uplatňuje mechanizmus rýchleho znovuposielania (anglicky fast retransmit). Príjemca signalizuje prijatie segmentov mimo poradia a okamžite po ich

prijatí. Odosielateľ teda môže dostať potvrdenie rovnakého segmentu viac-krát, čo signalizuje, že príjemca neprijal dáta. V prípade, že počet rovnakých potvrdzovacích paketov presiahne určenú hodnotu (anglicky duptresh, obvykle nastavená na 3), odosielateľ nečaká na vypršanie RTO, ale zašle požadovaný segment okamžite.

Pri znovuposielaní dát sa navyše používajú 2 hranice (anglicky treshold). Hranica R1 udáva počet pokusov na znovuodoslanie. Pokiaľ sa segment stále po R1 pokusoch nepodarí odoslať, musí IP vrstva zvoliť inú trasu pre smerovanie. Hodnota tejto hranice by mala byť aspoň 3. Hranica R2 udáva čas, po vypršaní ktorého sa ukončí práve otvorené TCP spojenie. Pri výmene dátových segmentov by hodnota hranice R2 mala byť vyššia ako 100 sekúnd.

Výpočet RTO

Poznáme viac spôsobov na výpočet RTO z hodnôt RTT. Prvým z nich je výpočet pomocou takzvaného vyhladeného RTT (anglicky smoothed RTT). Pri získaní hodnoty RTT pre najnovší segment upravíme hodnotu SRTT podľa vzorca:

$$SRTT = \alpha * SRTT + (1 - \alpha) * RTT \tag{2.1}$$

Kde α je vyhladzovacia konštanta, ktorej hodnota je väčšinou z intervalu <0.8, 0.9>.

Následne zo SRTT vypočítame RTO ako:

$$RTO = min(ubound, max(lbound, \beta * SRTT)) \tag{2.2}$$

Kde β je oneskorovací násobok s hodnotou z intervalu <1.3, 2.0> a *lbound* a *ubound* sú dolná, resp. horná hranica hodnoty RTO.

V prípade, že je RTT nestabilné (teda môže nadobudnúť hodnotu oveľa vyššiu ako sa očakáva), je predchádzajúci spôsob neefektívny. V dokumente [7] je popísaný vylepšený spôsob, ktorý počíta s históriou zmien hodnôt RTT. V tomto prípade sa počíta odhad priemernej hodnoty SRTT a priemernej odchýlky RTTVAR. Výpočet sa riadi nasledujúcimi vzorcami:

$$SRTT = (1 - g) * SRTT + g * RTT \tag{2.3}$$

$$RTTVAR = (1 - h) * RTTVAR + h * (|RTT - SRTT|) \tag{2.4}$$

$$RTO = SRTT + 4 * RTTVAR \tag{2.5}$$

Kde g je váha priradená vzorke RTT, obyčajne má hodnotu $1/8$ a h je váha priradená vzorke priemernej odchylky, obyčajne s hodnotou $1/4$.

Počiatočné hodnoty sa nastavia nasledovne:

$$SRTT = RTT \tag{2.6}$$

$$RTTVAR = RTT/2 \tag{2.7}$$

Hodnota RTO sa nastaví na 1 sekundu pre dátové segmenty alebo 3 sekundy pre SYN segmenty.

Pri meraní RTT však vzniká problém s nejednoznačnosťou. Pokiaľ prijmeme potvrdenie segmentu, ktorý bol medzičasom znovuposlaný, nevieme s určitosťou povedať, ku ktorému odoslaniu segmentu toto potvrdenie patrí a teda nevieme určiť RTT. Problém rieši Karnov algoritmus. Odhady SRTT a SRTTVAR neaktualizujeme pre znovuposlané pakety. Tým sme teda odstránili nejednoznačnosť merania RTT, avšak v prípade, že dochádza

k znovuposlaniu všetkých segmentov, nikdy nedôjde k aktualizácii hodnoty SRTT a teda ani RTO. Karnov algoritmus preto navyše zavádza exponenciálne zvyšovanie RTO pri znovuposlaní dát, čo v praxi znamená, že pri každom znovuposlaní dát prebehne nasledujúca aktualizácia RTO:

$$RTO = \gamma * RTO \tag{2.8}$$

$$\gamma(n+1) = 2 * \gamma(n) \tag{2.9}$$

$$\gamma(1) = 1 \tag{2.10}$$

Tento algoritmus je veľmi efektívny hlavne v sieťach s vysokou stratovosťou dát.

Riadenie zahltenia

Na riadení zahltenia a teda aj prenosu dát sa podieľajú 4 rôzne algoritmy, pričom v každom momente je aktívny iba jeden z týchto algoritmov. Algoritmy a ich prepínanie sa riadi na základe 3 hodnôt, *cwnd* udáva veľkosť okna odosielateľa (počiatočná hodnota môže byť ľubovolná), *rwnd* dostupnú veľkosť okna príjemcu a *sstresh* určuje hranicu, pri prekročení ktorej dochádza k zmene použitého algoritmu.

Algoritmus pomalého štartu (anglicky slow start) sa aktivuje na začiatku prenosu po vypršaní časovača doby nečinnosti (anglicky idle period), ktorá býva stanovená napríklad na 1 sekundu. Každým prijatým potvrdením segmentu sa postupne zvyšuje hodnota *cwnd*. Hodnota *cwnd* sa môže zvyšovať o jeden segment, avšak je doporučené zvyšovať podľa vzorca:

$$cwnd += \ min(N, MSS) \tag{2.11}$$

Kde N je počet bajtov potvrdených posledným potvrdzovacím segmentom.

Po tom, čo hodnota *cwnd* prekročí hodnotu *sstresh* dôjde k prepnutiu na algoritmus predchádzania zahlteniu (anglicky congestion avoidance). V tejto fáze sa pri potvrdení celého okna odosielateľa zvyšuje hodnota *cwnd* o jeden MSS segment. Pri vypršaní časovača RTO sa nastaví hodnota *sstresh* nasledovne:

$$sstresh = max(FlightSize/2, 2 * MSS) \tag{2.12}$$

Kde *FlightSize* udáva počet bajtov, ktoré boli odoslané, ale nepotvrdené.

Algoritmus pre rýchle znovuposielanie (anglicky fast retransmit) sa aktivuje v prípade, že odosielateľ dostal tri duplicitné potvrdzovacie segmenty. Duplicitné potvrdzovacie segmenty posiela príjemca v prípade, že obdržal dátové segmenty mimo poradia a následne pre každý ďalší prijatý segment až do momentu, kedy dostal správny (očakávaný) segment. Potvrdzovacie segmenty, ktoré potvrdzujú rovnaké dáta, avšak obsahujú zmenu veľkosti okna sa za duplicitné nepovažujú. Pri prijatí tretieho duplicitného potvrdzovacieho paketu dôjde k zaslaniu chýbajúceho paketu a k zmene *sstresh* podľa rovnice 2.12.

Posledným algoritmom je algoritmus rýchleho zotavenia (anglicky fast recovery). Táto fáza nastane po vykonaní rýchleho znovuposlania. Pokiaľ je následne potvrdené celé okno, prechádza sa do fázy predchádzania zahlteniu, v opačnom prípade sa prechádza na fázu pomalého štartu.

Algoritmy rýchleho znovuposlania a zotavenia existujú vo viacerých modifikáciach. Pôvodné verzie *TCP Tahoe* a *TCP Reno* (ktorých algoritmy sú popísané vyššie) boli vylepšené a vznikla tým nová modifikácia *TCP New Reno*, ktorá je definovaná v [3]. Počas fázy rýchleho zotavenia sa pri prijatí duplicitného potvrdzovacieho segmentu zašle nový, ešte neposlaný, segment z konca okna, čím sa okno udržuje plné. Zároveň sa pre každý potvrdzovací

segment, ktorý predstavuje čiastočný postup v potvrdzovaní (teda nejde o duplicitný potvrdzovací segment) predpokladá, že v tomto mieste vzniká ďalšia diera v prijatých segmentoch a preto sa tento segment rovno pošle. Tým je možné efektívnejšie vypĺňať veľké a časté diery v prijatých segmentoch. Je teda možné udržovať vysokú priepustnosť aj pri strate segmentov.

Pokiaľ nedochádza k strate segmentov, ale segmenty sú prijaté mimo poradia a to tak, že medzi dvoma po sebe prijatými segmentami vznikajú diery o veľkosti troch segmentov, dochádza pri *TCP New Reno* k problému. *TCP New Reno* chybne vstupuje do fázy rýchleho znovuposlania a keď sú chýbajúce segmenty prijaté, dochádza k zbytočnému zasielaniu duplicitných potvrdzovacích paketov a teda k zbytočnému znovuposielaniu týchto segmentov. *TCP New Reno* dosahuje pri nízkej stratovosti segmentov porovnateľné výsledky ako staršie verzie a pri vysokej stratovosti prekonáva *TCP Reno*.

Kapitola 3

Návrh a implementácia

Táto kapitola sa zaoberá hlavne návrhom samotnej hardvérovej jednotky. Popis návrhu zahrňuje celkový návrh komponentu, ako aj návrh jednotlivých súčasti tohto komponentu. Návrh vychádza hlavne z teoretických poznatkov. Okrem toho je v tejto kapitole načrtnutý aj spôsob riešenia niektorých problémov súvisiacich so samotnou implementáciou.

3.1 Popis jednotky

Úlohou jednotky je zabezpečiť spoľahlivé odoslanie dát po sieti pomocou protokolov TCP a IP. Táto funkcionalita je plne implementovaná hardvérovo. Jednotka je softvérovo konfigurovateľná. Je možné konfigurovať hlavne cieľovú IP adresu a cieľový port, ale aj iné položky TCP a IP hlavičiek. Zároveň je tieto hodnoty možné z jednotky vyčítať.

Na obrázku 3.1 je ukázané rozhranie jednotky. Do jednotky prichádzajú celkovo dva vstupné dátové toky. Jeden tok tvoria samotné dáta na odoslanie. Druhým tokom sú odpovede od servera. Ide hlavne o potvrdzovacie segmenty k jednotlivým odoslaným dátam. Okrem potvrdzovacích segmentov v tomto toku prichádzajú taktiež segmenty s nastavenými príznakmi SYN, FIN alebo RST. Oba tieto toky sú privedené do jednotky pomocou zbernice AXI4-Stream. Ide o vysokorýchlostnú zbernicu pre dáta vo forme tokov, ktorá je súčasťou rodiny mikroprocesorových zberníc ARM AMBA. Viac informácií o týchto zberniciach je možné získať zo zdroja [10].

Na konfiguráciu jednotky slúži zbernica AXI4-Lite. AXI4-Lite predstavuje odľahčenú verziu zbernice z rodiny ARM AMBA, ktorá okrem signálov pre prenos dát obsahuje aj signály potrebné pre adresovanie a určenie smeru prenosu. Pomocou nej je teda možné zapisovať a vyčítať dáta z jednotlivých registrov. Adresový priestor je popísaný v tabuľke 3.1. Pre zjednodušenie tejto konfigurácie vznikol aj softvérový nástroj schopný zapisovať a čítať jednotlivé hodnoty položiek hlavičiek. Tento nástroj v podstate zaobaľuje adresový priestor jednotky a prístup k AXI4-Lite zbernici.

Dáta na odoslanie prijaté jednotkou sú, po rozšírení o správne hlavičky, posielané pomocou AXI4-Stream zbernice na výstup. Tento výstup je pripojený na jednotku TEMAC (Tri-Mode Ethernet MAC). Táto jednotka je vytvorená spoločnosťou Xilinx a slúži na prístup k ethernetovej linke. Jednotka navyše vyžaduje ďalší riadiaci tok typu AXI4-Stream. Pomocou tohto toku je jednotka TEMAC konfigurovaná. Ide najmä o správne nastavenie typu použitých protokolov a zapnutie niektorej dodatočnej funkcionality jednotky, ako napríklad počítanie kontrolných súčtov a podobne. Dokumentácia tejto jednotky je obsiahnutá v dokumente [9].

Adresa (hexa)	Platné bity	Čítanie/ Zápis	Význam pre IPv4	Význam pre IPv6
x00	16	R/W	Zdrojový TCP port	
x04	16	R/W	Cieľový TCP port	
x08	6	R/W	Hodnota DSCP	
x0C	2	R/W	Hodnota ECN	
x10	16(v4)/20(v6)	R/W	Identifikácia	Označenie toku
x14	4(v4)/8(v6)	R/W	IP Príznaky	Ďalšia hlavička
x18	8	R/W	Životnosť	Limit skokov
x1C	32	R/W	Zdrojová IP adresa	
x20	32	R/W	-	
x24	32	R/W	-	
x28	32	R/W	-	
x2C	32	R/W	Cieľová IP adresa	
x30	32	R/W	-	
x34	32	R/W	-	
x38	32	R/W	-	
x3C	32	R/W	Cieľová MAC adresa	
x40	16	R/W	Cieľová MAC adresa	
x44	32	R/W	Zdrojová MAC adresa	
x48	16	R/W	Zdrojová MAC adresa	
x4C	4	R	IP verzia	

Tabuľka 3.1: Adresový priestor jednotky implementujúcej komunikáciu pomocou protokolov TCP a IP.

3.2 Schéma jednotky

Na obrázku 3.2 je vyobrazené rozdelenie celej jednotky na menšie bloky. Každý blok zodpovedá za určitú časť celkovej funkcionality.

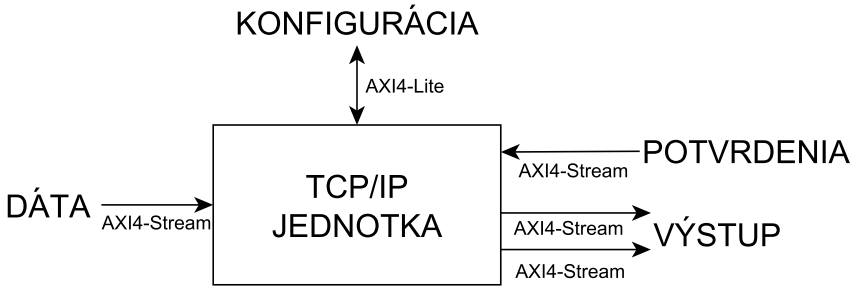
3.2.1 Riadiaci konečný automat

Jadro celej jednotky tvorí riadiaca jednotka. Táto jednotka je na obrázku 3.3. Pozostáva z dvoch konečných automatov a podpornej logiky.

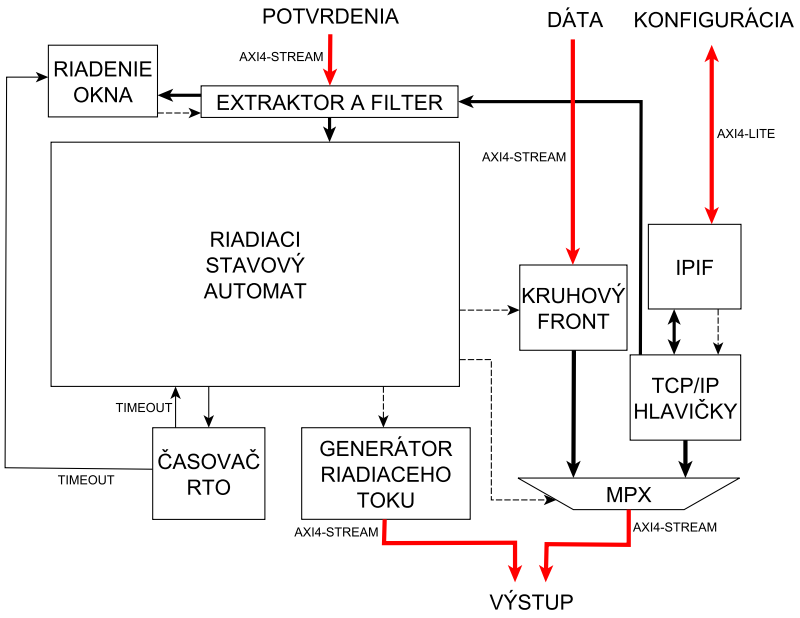
Hlavný riadiaci konečný automat zabezpečuje prechody medzi jednotlivými fázami komunikácie. Keďže jednotka slúži hlavne na odosielanie dát, je možné všeobecný konečný automat pre riadenie TCP komunikácie zjednodušiť len na jeho klientskú časť (obrázok 3.4). Po odoslaní okna dát je potrebné počkať na ich potvrdenie. Stav *ESTAB* je preto rozdelený na dva stavy, stav *SND WIN*, v ktorom je odoslané okno paketov a stav *WAIT*, počas ktorého jednotka čaká na prijatie potvrdenia pre odoslané segmenty.

Výstupný automat zabezpečuje odoslanie dát na výstup v správnom poradí. Ide o správnu následnosť odoslania jednotlivých položiek hlavičiek a následne odoslanie dát. To je zabezpečené nastavením správnej hodnoty výberového signálu multiplexoru.

Výstupný automat je navyše rozšírený o podpornú logiku, ktorá sa stará o správne nastavenie ostatných riadiacich signálov pre okolité jednotky. Ide najmä o riadiace signály kruhového frontu.



Obrázok 3.1: Rozhranie jednotky implementujúcej TCP/IP komunikáciu.

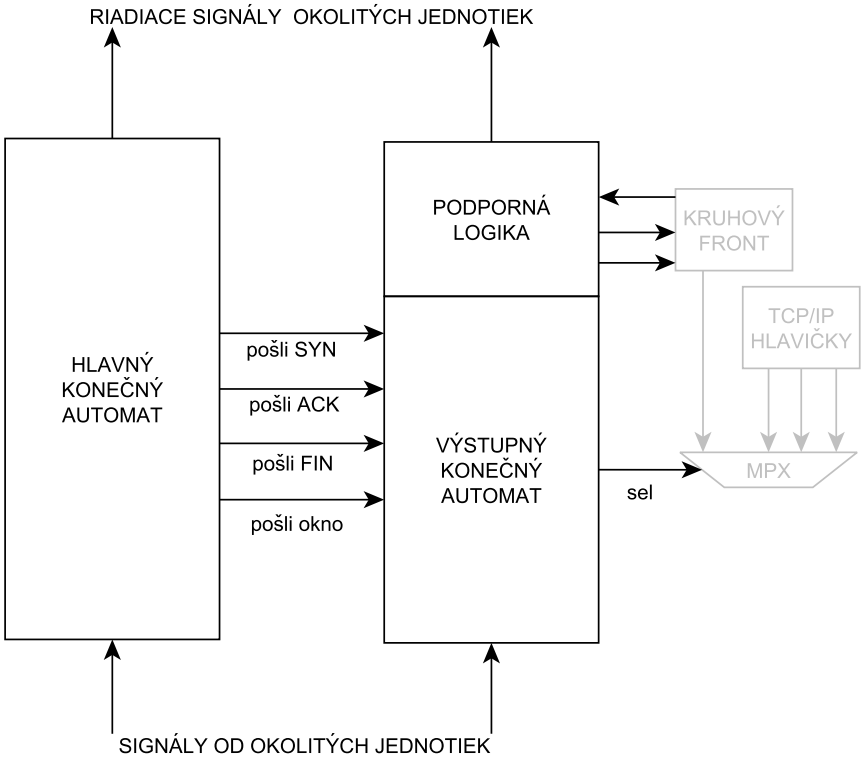


Obrázok 3.2: Celková schéma jednotky implementujúcej TCP/IP komunikáciu.

3.2.2 Extraktor a filter

Táto jednotka slúži na spracovanie odpovedí zo strany servera. Podrobnejšiu schému tejto jednotky je možné vidieť na obrázku 3.5. Z prichádzajúcich segmentov sú najprv vyextrahované hodnoty jednotlivých položiek. Ide hlavne o čísla protokolov, na základe ktorých sa určí, či vôbec ide o TCP/IP komunikáciu. Ďalej sa extrahujú adresy IP, čísla portov, potvrdzovacie číslo a nastavené TCP príznaky.

Tieto hodnoty následne putujú do filtra, ktorý odfiltruje všetku komunikáciu, ktorá sa netýka spojenia vytvoreného medzi serverom a celým komponentom. Extrahované hodnoty cieľovej adresy IP a cieľového TCP portu sa porovnajú s očakávanými hodnotami, tým sa zistí, či segment patrí do správneho spojenia. Očakávané hodnoty cieľovej adresy IP a cieľového TCP portu sú získané z konfigurácie zdrojovej adresy IP a zdrojového TCP portu pre odosielanie (keďže zdrojová adresa resp. zdrojový port segmentu, ktorý jednotka odosiela by mala byť zhodná s cieľovou adresou resp. cieľovým portom prijatých segmentov). Navyše je skontrolované potvrdzovacie číslo. Toto číslo musí, v prípade potvrdzovacieho



Obrázok 3.3: Schéma hlavnej riadiacej jednotky.

segmentu, patriť do momentálneho rozsahu okna segmentov. Pokiaľ segment nebol odfiltrovaný, sú ďalším jednotkám zaslané informácie o nastavených TCP príznakoch a o hodnote potvrdzovacieho a sekvenčného čísla.

Sekvenčné a potrdzovacie čísla tvoria kruhovú postupnosť. Môže teda nastať situácia, že pravú hranicu okna tvorí číslo, ktoré je nižšie ako ľavá hranica okna. Túto situáciu ilustruje obrázok 3.6. V prípade, že je porovnávaná hodnota sekvenčného čísla s hranicami okna segmentov, je nutné rozlišovať dva prípady. Oba prípady sú zapísane vo výrazoch 3.1 a 3.2. Výraz 3.3 spája tieto dva prípady do jedného výrazu. Premenné *min* a *max* zastupujú ľavú a pravú hranicu okna, premenná x určuje hodnotu sekvenčného čísla, ktoré je porovnávané s hranicami okna a premenná *IN* vyjadruje príslušnosť sekvenčného čísla do rozsahu okna.

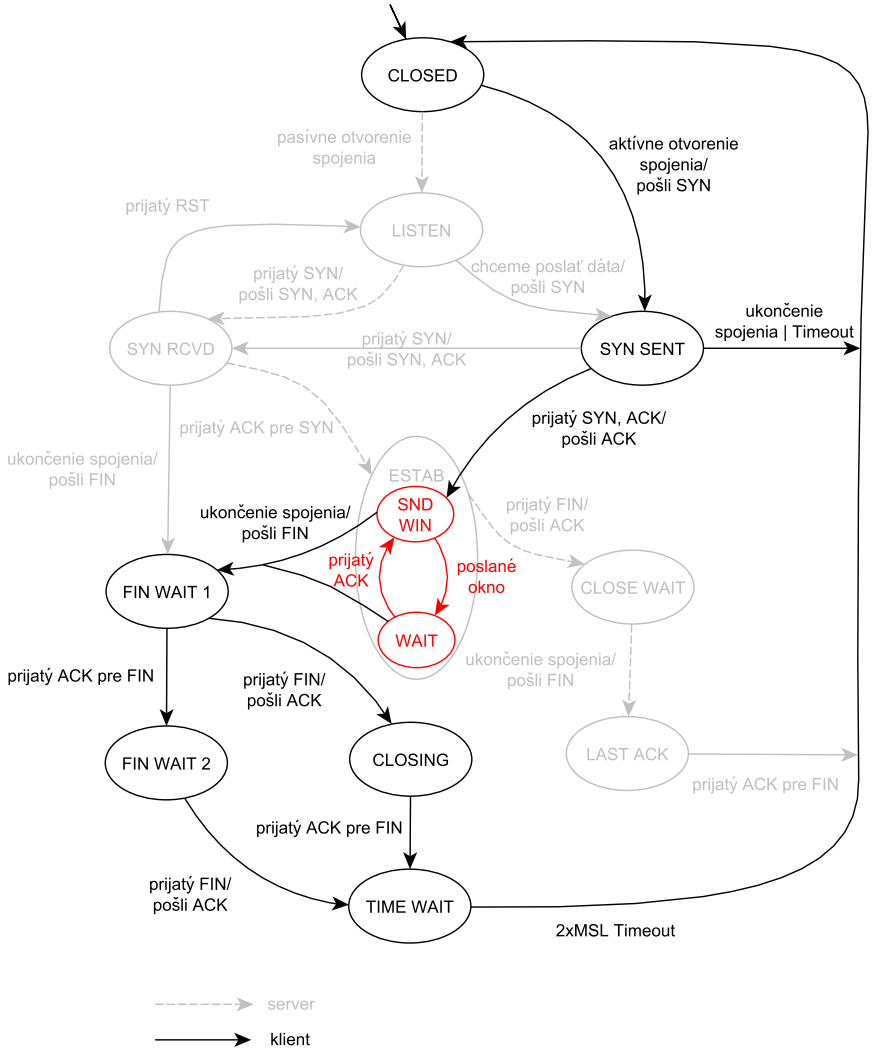
$$max > min \implies IN = (max \geq x) \wedge (x \geq min) \tag{3.1}$$

$$max \leq min \implies IN = (max \geq)x \vee (x \geq min) \tag{3.2}$$

$$IN = \{(max > min) \wedge (max \geq x) \wedge (x \geq min)\} \vee \{[max \leq min] \wedge [(max \geq x) \vee (x \geq min)]\} \tag{3.3}$$

3.2.3 Kruhový front

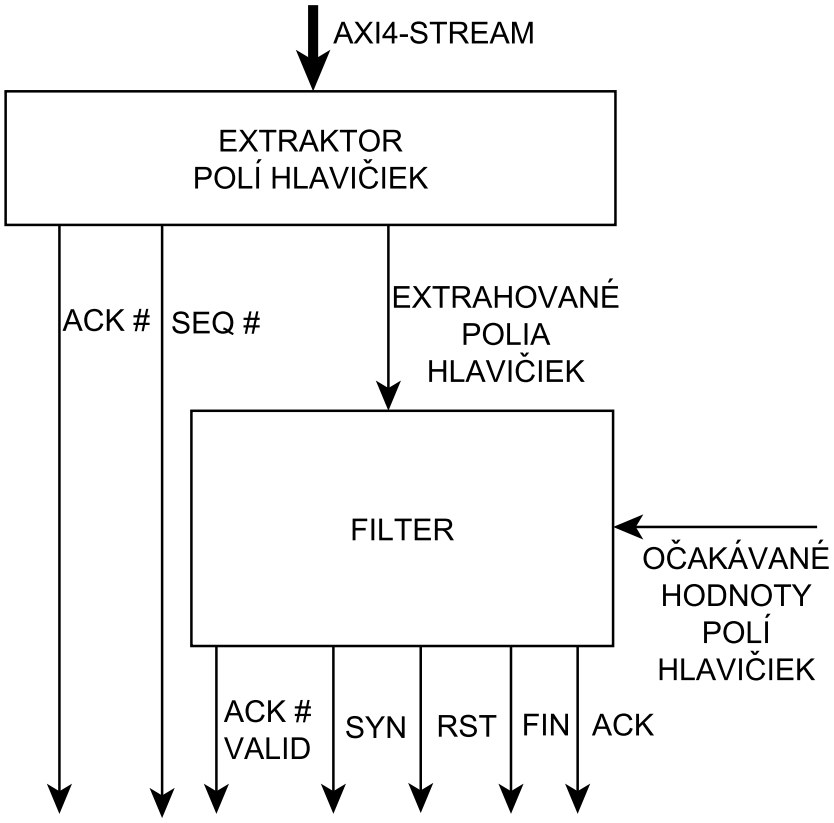
Jednotlivé dáta, ktoré prichádzajú zo vstupu je potrebné dočasne uložiť, aby sme boli schopní ich v prípade potreby preposlať. Po úspešnom poslaní a potvrdení týchto dát je



Obrázok 3.4: Stavový diagram TCP komunikácie pre stranu klienta.

potrebné ich zároveň vymazať a uvoľniť tak miesto pre ďalšie dáta. Na tento účel slúži blokovú pamäť s náhodným prístupom (BRAM). Na obrázku 3.7 je ukázaná schéma zapojenia tejto pamäte.

Nad touto pamäťou pracuje riadiaci konečný automat (obrázok 3.8), ktorý ovláda zápis do pamäte, čítanie z pamäte a zároveň realizuje kruhový front nad touto pamäťou. Na realizáciu kruhového front používa pomocné pole indexov, ktoré obsahuje počiatočné adresy jednotlivých blokov dát, ktoré po zabalení do hlavičiek patria do rovnakého segmentu. Navyše tento automat sprístupňuje pamäť ostatným jednotkám formou 4 jednoduchých signálov. Signál *PRIPRAVENÝ* signalizuje ostatným jednotkám, že pamäť obsahuje nejaké platné dáta a zároveň je automat v stave, kedy je schopný tieto dáta vydať. Signál *OFFSET* určuje poradie bloku dát, s ktorým sa bude pracovať a pomocou signálov *POSLAŤ* a *MAZAŤ* sa ovláda, či má byť blok dát zvolený signálom *OFFSET* poslaný na výstup alebo majú byť všetky bloky až po tento blok zmazané.



Obrázok 3.5: Schéma bloku na extrakciu hlavičiek a filtrovanie.

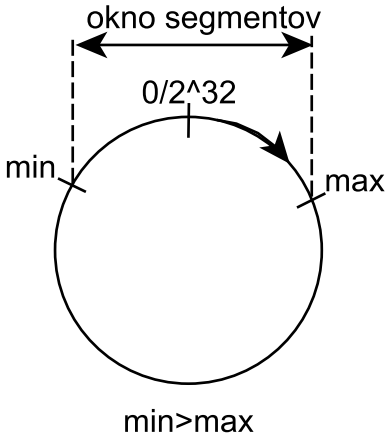
3.2.4 TCP/IP hlavičky

Položky TCP a IP hlavičky sú uložené v jednoduchých registroch (obrázok 3.9). Do týchto registrov je možné softvérovo zapisovať a vyčítať z nich hodnoty pomocou zbernice AXI4-Lite. Zbernicu zo strany TCP/IP jednotky ovláda komponent AXI4-Lite IP interface (IPIF, dokumentácia [13]) od spoločnosti Xilinx. Komponent zároveň prevádza adresu zbernice AXI4-Lite na vektor bitových signálov typu 1 z n, kedy je na základe adresy aktívny vždy len jeden bit vektoru. Tento vektor je potom možné použiť ako povoľovacie signály pre jednotlivé registre obsahujúce položky TCP a IP hlavičky.

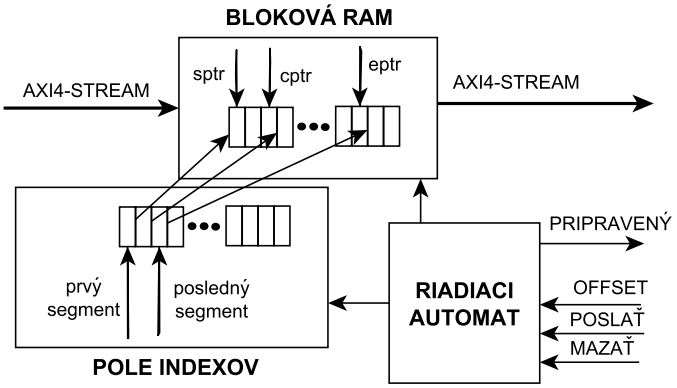
Výstupy registrov spoločne s výstupom kruhového frontu sú privedené na multiplexor, ktorý je ovládaný hlavným riadiacim konečným automatom. Riadiaci automat teda môže jednoduchým nastavením výberového signálu multiplexoru ovládať, ktorá časť segmentu je v daný čas posielaná. Výstupy registrov obsahujúcich zdrojovú adresu IP, zdrojový TCP port sú navyše privedené do komponentu *EXTRAKTOR A FILTER*, kde sú použité pri filtrácii prichádzajúcich odpovedí.

3.2.5 Časovač RTO

Tento komponent realizuje počítanie času RTO z hodnôt RTT a zároveň samotný časovač. Schéma je na obrázku 3.10. Funkcionalita komponentu je zabezpečená dvoma voľne bežiacimi čítačmi, komparátorom a logikou, ktorá prepočítáva hodnoty RTT. Jeden čítač



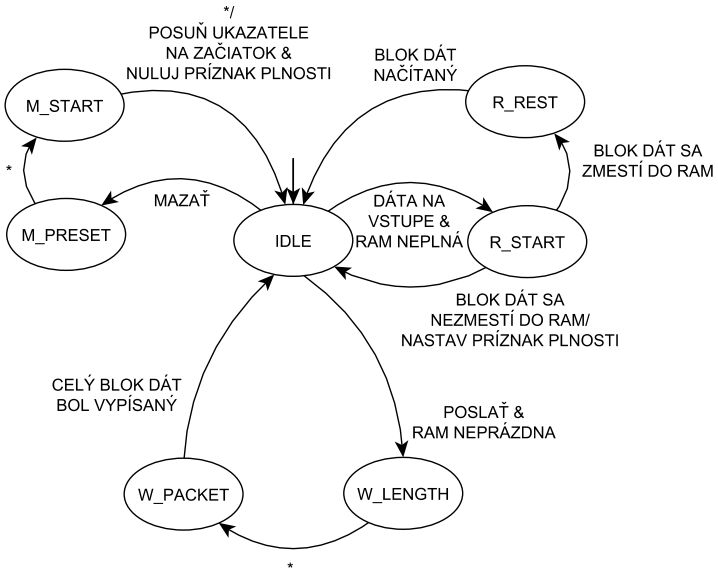
Obrázok 3.6: Problém s kruhovou následnosťou sekvenčných čísel.



Obrázok 3.7: Schéma bloku realizujúceho kruhový front nad pamäťou.

a komparátor slúžia na generovanie vypršania časovača. V prípade, že čítač dosiahne vypočítanú hodnotu RTO, dochádza k signalizácii vypršania a zároveň k vynulovaniu tohto čítača.

Druhý čítač slúži na meranie jednotlivých časov potvrdení. V prípade príchodu potvrdenia dochádza k uloženiu momentálnej hodnoty tohoto časovača a následnému vynulovaniu oboch čítačov. Uložená hodnota udáva nové RTT, ktoré sa použije pre výpočet novej hodnoty RTO. Výpočet RTO bol zvelený ako výpočet aritmetického priemeru z posledných dvoch hodnôt RTT. Zložitejšie spôsoby výpočtu vyžadujú násobenie desatinnými číslami, resp. delenie celými číslami. Takéto výpočty nie je možné efektívne implementovať v hardvéri. Delenie dvomi je naopak veľmi jednoduché implementovať, kedže ide len o bitový posun.



Obrázok 3.8: Konečný automat ovládajúci prácu nad pamäťou.

3.2.6 Riadenie veľkosti okna

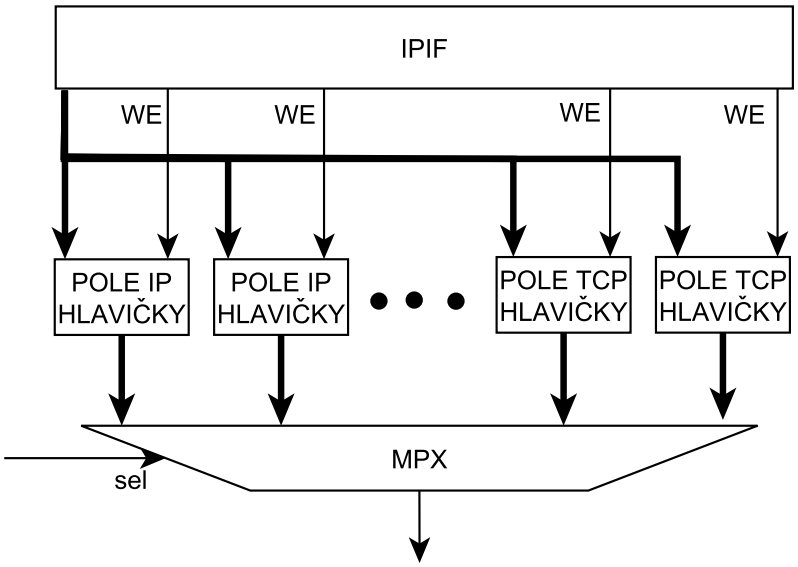
Hranice okna sa uchovávajú v dvoch registroch. Tieto registre sú na obrázku 3.11. V jednom z registrov sa uchováva dolná hranica okna (teda minimálne sekvenčné číslo). Táto hodnota sa prepíše pri prijatí potvrdenia práve na potvrdzovacie číslo obsiahnuté v tomto potvrdení. Druhý register obsahuje veľkosť okna. Tento register je rozšírený o schopnosť inkrementovať svoju hodnotu. K inkrementácii dochádza v prípade prijatia validného potvrdenia a to o hodnotu MSS.

V prípade vypršania časovača RTO dochádza k zmene veľkosti okna na polovicu. Minimálna veľkosť okna je rovná hodnote MSS, teda nemôže nastať situácia, že by okno bolo menšie ako jeden segment. Výstup registru obsahujúceho hodnotu veľkosti okna posunutý o bit doprava je privedený na vstup tohoto registru. Ako vstupný signál pre povolenie zápisu do registru funguje signál vypršania časovača RTO. Tým je zabezpečené, že k zápisu polovičnej hodnoty do registru dôjde len v prípade vypršania časovača RTO.

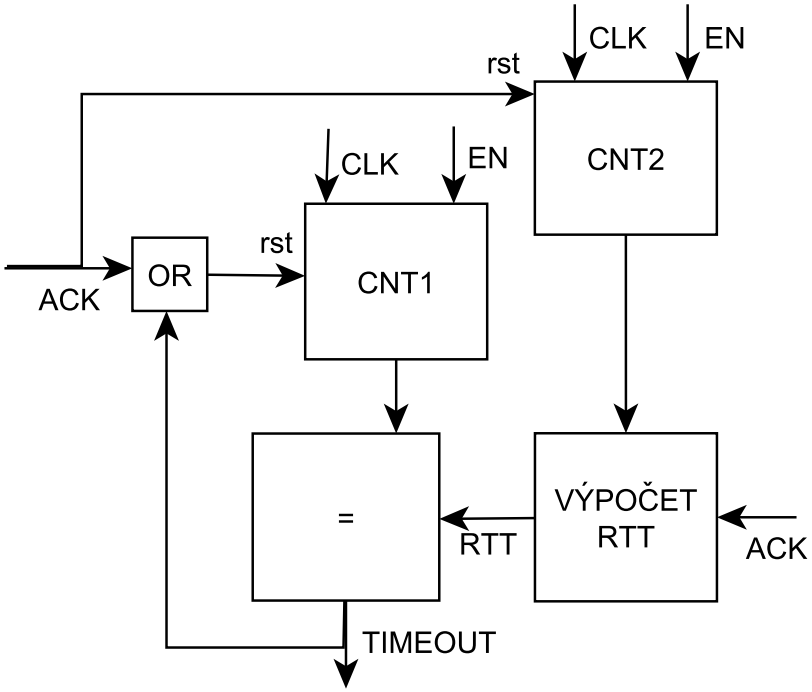
Z dôvodu zjednodušenia a zrýchlenia implementácie boli algoritmy riadenia zahltenia zjednodušené do podoby, keď je počas celej komunikácie aktívny algoritmus predchádzania zahlteniu.

3.2.7 Generátor riadiaceho toku

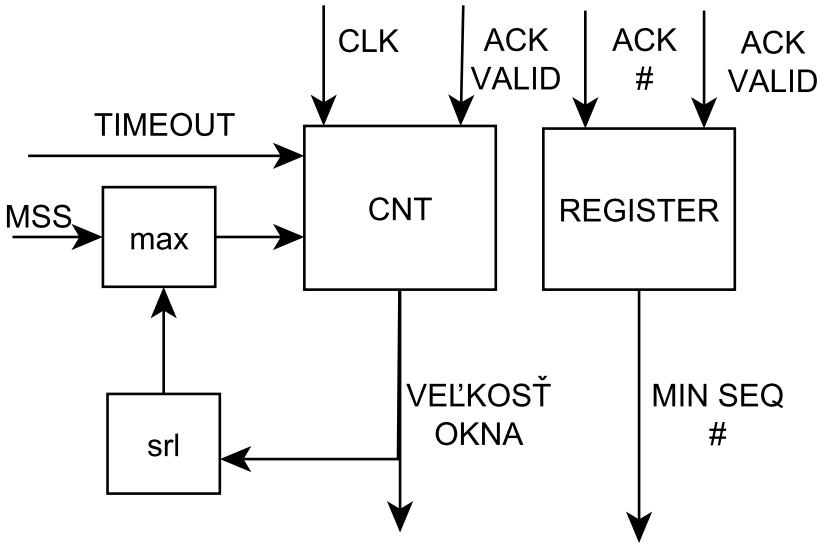
Pre každý odoslaný segment je nutné generovať riadiace informácie. Tieto informácie zahŕňajú typ protokolov, povolenie počítania kontrolných súčtov, ukazovateľ na začiatok TCP hlavičky a ukazovateľ na začiatok položky pre kontrolný súčet. Konkrétne nastavenie týchto položiek a ich poradie je zobrazené na obrázku 3.12 a je obsiahnuté v dokumente [11].



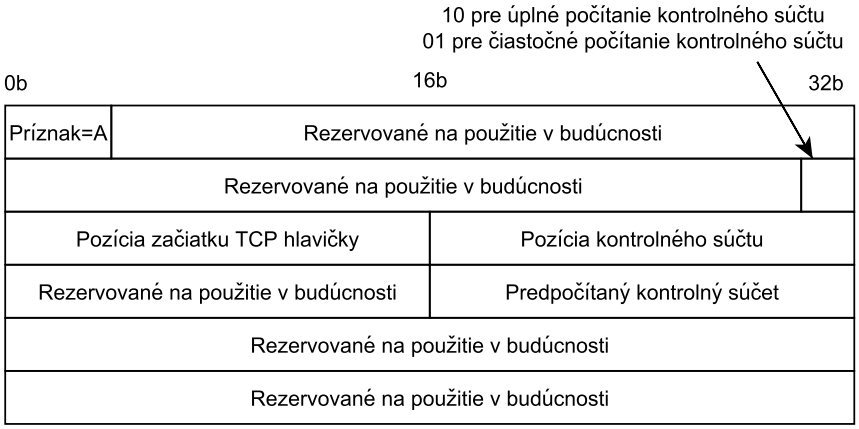
Obrázok 3.9: Schéma bloku uchovávajúceho hodnoty položiek TCP a IP hlavičky.



Obrázok 3.10: Schéma bloku časovača RTO.



Obrázok 3.11: Schéma riadenia veľkosti okna.



Obrázok 3.12: Usporiadanie riadiacich informácií pre vysielané TCP/IP dáta.

Kapitola 4

Testovanie a verifikácie

Táto kapitola popisuje jednotlivé postupy použité pri testovaní a verifikácii funkčnosti implementácie. Testovanie prebiehalo simulovaním s využitím programov Xilinx ISE Design Suite 14.1 a Xilinx ISE Simulator (ISim). Na verifikácie bol využitý jazyk System Verilog.

4.1 Testovanie jednotlivých súčastí

Prvým krokom pri overovaní správnosti implementácie bolo testovanie jednotlivých súčastí implementácie. Ku každému funkčnému bloku bolo vytvorené testovacie prostredie, ktorého úlohou je nahradiť okolie tohto testovaného bloku. Testovacie prostredie ovláda vstupné signály bloku a zachytáva hodnoty výstupných signálov.

Vďaka tomuto testovaciemu prostrediu je možné jednotky zmysluplne simulovať bez potreby okolitých jednotiek. Na základe analýzy hodnôt zachytených výstupných signálov bloku je potom možné určiť, či daný blok funguje, ako by mal. Obrázok 4.1 ukazuje príklad zobrazenia zachytených signálov v programe Xilinx ISE Simulator.

Týmto prístupom je možné otestovať najmä, či sa implementácia bloku zhoduje s návrhom tohto bloku.

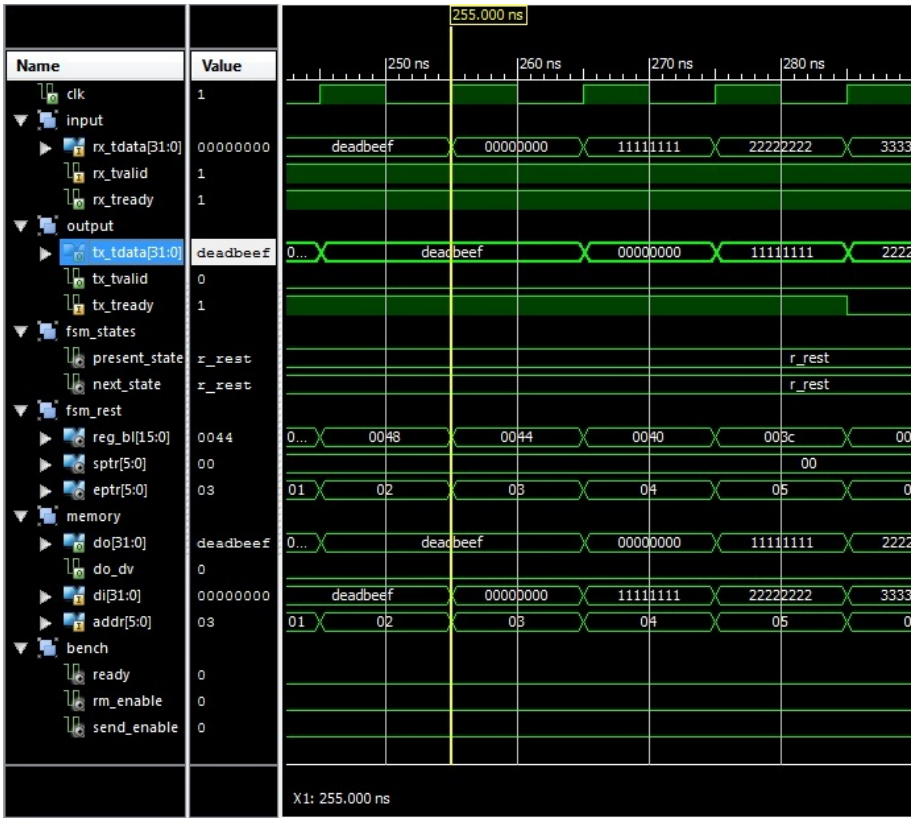
4.2 Testovanie celku

Ďalším krokom bolo otestovať implementáciu ako celok. Opäť je vytvorené testovacie prostredie, ktoré nahrádza okolie jednotky a ovláda rozhranie jednotky.

Analýzou zachytených hodnôt signálov je v tomto prípade možné odhaliť väčšinu synchronizačných chýb a chýb v komunikácii medzi jednotlivými funkčnými blokmi. Ide najmä o chyby, kedy jeden blok očakáva nastavenie signálu v inom takte, ako v ktorom je nastavený blokom, ktorý za toto nastavenie zodpovedá.

Rozhranie celej jednotky je tvorené zbernicami typu AXI4-Stream. Pre účely ovládania, resp. zachytávania dát z týchto zberníc boli vytvorené pomocné komponenty *AXI_GEN* a *AXI_MON*. Komponent *AXI_GEN* dokáže zo vstupného súboru načítať dáta v šesťnástkovej sústave, na základe ktorých riadi zbernicu AXI4-Stream. Komponent *AXI_MON* naopak dokáže dátový tok zo zbernice AXI4-Stream uložiť do výstupného súboru.

Celkový tok dát v simulácii je načrtnutý na obrázku 4.2. Tester naplní vstupné súbory vstupnými dátami. Tieto súbory sú načítané inštanciami komponentu *AXI_GEN*, ktoré ich prevedú na dátové toky AXI4-Stream zberníc, ktoré vstupujú do testovanej jednotky. Výstup vo forme dvoch tokov AXI4-Stream zbernice je privedený do dvoch inštancií



Obrázok 4.1: Zachytené signály pri simulovaní komponentu kruhový front.

komponentu *AXI_MON*, ktoré ho zapíšu do výstupných súborov.

4.3 Verifikácie

Testovaním v simuláciách bolo potvrdené, že jednotlivé súčasti a zároveň jednotka ako celok vykonávajú požadovanú funkcionalitu. Nie je však možné potvrdiť, že túto funkcionalitu vykonávajú pre každý prípustný segment, resp. pre každú prípustnú postupnosť segmentov.

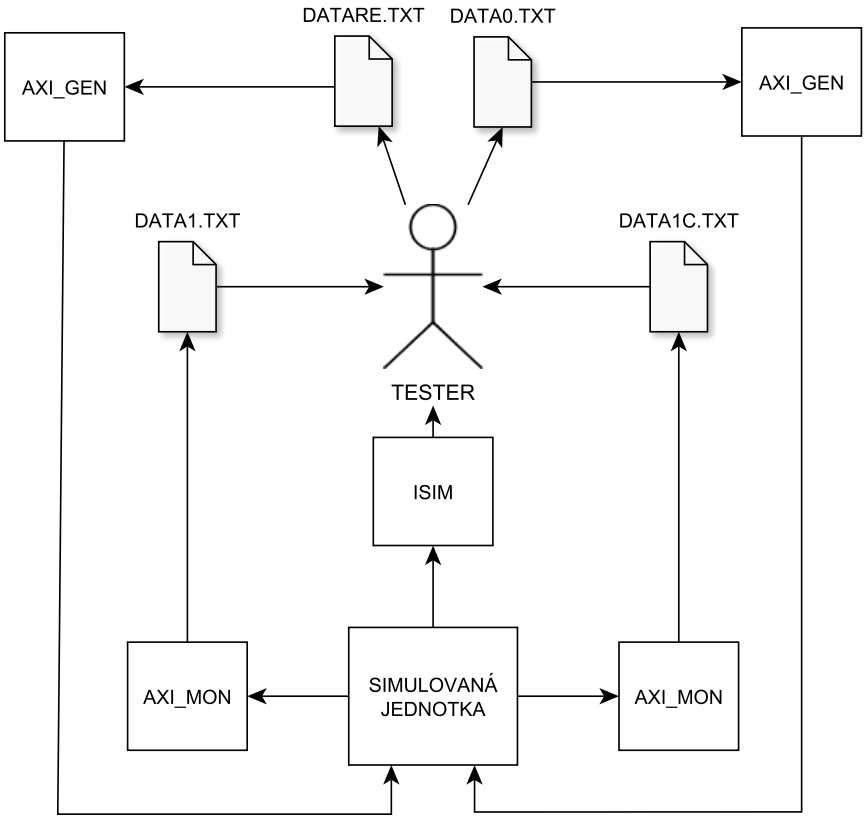
Nasledujúcim krokom pri overovaní správnosti implementácie je preto verifikovanie riešenia. Najlepší a najvyužívanejší spôsob verifikovania hardverových implementácií je pomocou takzvanej funkčnej verifikácie.

Ide o typ verifikácie, ktorá má za úlohu skontrolovať, čí výsledná hardvérová implementácia vyhovuje zadaniu.

Verifikáciami boli testované nie len náhodné následnosti náhodných segmentov, ale aj niektoré špecifické prípady, ako sú napríklad vytváranie a ukončovanie spojenia, zasielanie kamikaze segmentov (segment s nastavenými všetkými TCP príznakmi) a podobne. Samotné verifikácie by sa dali rozdeliť do viacerých etáp.

1. etapa

V tejto etape verifikácii bolo testované vytvorenie spojenia a prenos dát. Nepočítalo sa so stratami segmentov ani s posielaním nezmyselným segmentov zo strany serveru. Účelom bolo otestovať, či je implementácia schopná korektného prenosu dát.



Obrázok 4.2: Tok dát v simulácii s využitím komponentov *AXI_GEN* a *AXI_MON*.

2. etapa

Táto etapa mala za úlohu otestovať schopnosť znovuposielať nepotvrdené segmenty. Zo strany servera teda chodili odpovede zo značným oneskorením alebo až pri prijatí duplicitných segmentov. Sledovalo sa najmä, či je implementácia korektne schopná prispôsobiť časovač RTO a veľkosť okna.

3. etapa

Ďalšia funkcionalita, ktorá bola otestovaná, bola schopnosť správne filtrovať segmenty zo strany servera. Server teda okrem potvrdzovacích segmentov zasielal aj nezmyselné segmenty a sledovalo sa, ako sa s týmto stavom dokáže implementácia vysporiadať.

4. etapa

Poslednú, ale zároveň najdôležitejšiu etapu tvorí spojenie všetkých predchádzajúcich etáp do jednej. Vstupné pakety sú generované náhodne a odpovede zo strany servera majú náhodné oneskorenie a je šanca straty každého segmentu. Zo strany servera navyše chodia aj náhodne generované segmenty. Týmto je možné sa čo najviac priblížiť reálnej sieti. Zároveň sa v tejto etape verifikuje správna súčinnosť všetkých jednotiek implementácie.

Kapitola 5

Výsledky

V tejto kapitole sú analyzované výsledky dosiahnuté implementáciou. Ide o výsledky získané testovaním a verifikáciou. Zároveň táto kapitola obsahuje náročnosť riešenia na spotrebu zdrojov na čipe.

5.1 Spotreba zdrojov

Veľmi dôležitou informáciou o výslednej implementácii je spotreba zdrojov na čipe. Tabuľka 5.1 ukazuje spotrebu jednotlivých jednotiek ako aj celej implementácie. Tieto údaje boli získané pomocou programu Xilinx ISE Design Suite 14.1. Zvolený čip je z rodiny Spartan 6, konkrétne model xc6slx45-3fgg484. Pre porovnanie sú navyše pridané hodnoty UDP exportéra, ktorý táto implementácia v budúcnosti nahradí. Tento exportér len exportuje prijaté dáta s využitím protokolov UDP a IPv6 a vznikol vrámci projektu Sec6Net[1].

5.1.1 Spotreba pamäte

Implementácia navyše využíva blokovú pamäť typu RAM. Tento zdroj je na čipe taktiež obmedzený. Podľa dokumentu [12] sú na čipe z rodiny Spartan 6 dostupné 18-kilobitové pamäte, ktoré však môžu byť použité ako dve 9-kilobitové pamäte. Pre každých 8 bitov sa používa paritný bit, preto reálna kapacita každej pamäte je len 8 kilobitov, teda jeden kilobajt.

Konkrétne na čipe xc6slx45-3fgg484 je dostupných 116 blokových pamätí. Po rozdelení každej na dve nezávisle je ich teda 232. Celková využiteľná kapacita je teda 232 kilobajtov, čo pri implicitnej maximálnej veľkosti segmentu 536 bajtov predstavuje niečo vyše 430 segmentov. Týmto je limitovaná veľkosť okna, kedže nemá zmysel pracovať s oknom, ktoré je väčšie ako množstvo dát, ktoré je jednotka schopná uložiť.

5.2 Maximálna frekvencia

Ďalším ukazovateľom výslednej implementácie je maximálna frekvencia, pri ktorej dokáže bezchybne fungovať. Tento údaj bol získaný rovnako ako spotreba zdrojov na čipe a konkrétne hodnoty zachytáva tabuľka 5.2. Opäť je pre porovnanie pridaný UDP exportér.

Z tabuľky 5.2 je jasne vidno, že najpomalšie sú jednotky, v ktorých dochádza k najväčším výpočtom, ako napríklad extraktor s filtrom. V tejto jednotke je potrebné porovnávať

[1] Projekt Sec6Net, dostupné na: http://www.fit.vutbr.cz/research/view_project.php.cs?id=517.

Jednotka	Cieľ optimalizácie	Slice registre	Slice LUT	LUT FF páry
Riadenie veľkosti okna (*win_control*)	plocha	3 (0%)	13 (0%)	14
	rýchlosť	3 (0%)	13 (0%)	14
Časovač RTO (*rto_timer*)	plocha	95 (0%)	143 (0%)	144
	rýchlosť	95 (0%)	143 (0%)	144
Generátor riadiaceho toku (*cstream_gen*)	plocha	7 (0%)	41 (0%)	42
	rýchlosť	7 (0%)	44 (0%)	46
Kruhový front (*rbuffer_ent*)	plocha	25 (0%)	176 (0%)	177
	rýchlosť	28 (0%)	179 (0%)	181
Extraktor a filter (*hfe_filter*)	plocha	69 (0%)	360 (1%)	360
	rýchlosť	72 (0%)	363 (1%)	364
Hlavný automat (*tcp_fsm*)	plocha	3 (0%)	13 (0%)	13
	rýchlosť	3 (0%)	14 (0%)	14
Výstupný automat (*fsm*)	plocha	88 (0%)	433 (1%)	434
	rýchlosť	123 (0%)	421 (1%)	424
UDP exportér	plocha	642 (1%)	824 (3%)	901
Celkovo (IPv4)	plocha	428 (1%)	785 (2%)	880
	rýchlosť	460 (1%)	821 (3%)	1035
Celkovo (IPv6)	plocha	694 (1%)	1412 (5%)	1605
	rýchlosť	726 (1%)	1442 (5%)	1781

Tabuľka 5.1: Zdroje na čipe Spartan 6 (xc6slx45-3fgg484) zabrané jednotlivými jednotkami.

a sčítavať bitové vektory, čo v konečnom dôsledku znižuje maximálnu dosiahnuteľnú frekvenciu. Tieto jednotky sú preto vhodnými kandidátmi na optimalizáciu.

5.3 Odhadovaná priepustnosť

Na základe hodnôt nameraných hodnôt je možné odhadnúť výslednú priepustnosť riešenia.

5.3.1 Ideálna priepustnosť

Najprv spočítame maximálnu možnú priepustnosť odvodenú z maximálnej frekvencie. Táto priepustnosť je dosiahnuteľná len v ideálnom prípade, keď je doba potvrdzovania segmentov veľmi nízka, ideálne neexistujúca. V takomto prípade sa okno nikdy nestihne naplniť, pretože skôr ako stihne byť celé okno poslané, dôjde k prijatiu potvrdenia prvého zaslaného segmentu a teda k posunutie okna doprava, čím sa uvoľní miesto v okne.

Implementácia je schopná poslať jedno 32-bitové slovo v každom hodinovom takte. Z dôvodu uvedeného vyššie sa zároveň implementácia nikdy nedostane do iného stavu ako stavu, v ktorom sú zasielané segmenty. Výsledná priepustnosť preto bude zodpovedať rovnici:

$$p = 32 * f \tag{5.1}$$

Kde p je hodnota priepustnosti a f je frekvencia hodinového signálu. Dosadením hodnoty maximálnej frekvencie (tabuľka 5.2), pri ktorej je schopná implementácia bechybného fungovania, do rovnice 5.1 získavame maximálnu priepustnosť. Na čipe z rodiny Spartan 6 to konkrétne je okolo 320 megabajtov za sekundu alebo 2560 megabitov za sekundu.

Jednotka	**Cieľ optimalizácie**	**Min. perióda (ns)**	**Max. frek. (MHz)**
Riadenie veľkosti okna (*win_control*)	plocha	3.650	273.991
	rýchlosť	3.716	269.103
Časovač RTO (*rto_timer*)	plocha	3.607	277.266
	rýchlosť	3.607	277.266
Generátor riadiaceho toku (*cstream_gen*)	plocha	4.180	239.217
	rýchlosť	1.884	530.856
Kruhový front (*rbuffer_ent*)	plocha	4.742	210.868
	rýchlosť	5.388	185.608
Extraktor a filter (*hfe_filter*)	plocha	8.963	111.565
	rýchlosť	6.146	162.706
Hlavný automat (*tcp_fsm*)	plocha	2.527	395.757
	rýchlosť	2.500	400.064
Výstupný automat (*fsm*)	plocha	6.446	155.130
	rýchlosť	5.465	182.983
UDP exportér	plocha	7.540	132.617
Celkovo (IPv4)	plocha	12.713	78.661
	rýchlosť	11.464	87.230
Celkovo (IPv6)	plocha	12.713	78.661
	rýchlosť	11.464	87.230

Tabuľka 5.2: Časovanie, ktoré sú schopné jednotlivé jednotky dodržať na čipe Spartan 6 (xc6slx45-3fgg484).

5.3.2 Priepustnosť pri oneskorení

V reálnej sieti však nejaké oneskorenie určite bude existovať. V takomto prípade môže nastať situácia, že celé okno už bolo odoslané, avšak ešte nebol žiaden zo segmentov potvrdený. Jednotka preto čaká a neodosiela dáta až do momentu, kedy prijme potvrdenie niektorého z odoslaných segmentov. Vtedy až dochádza k posunu a uvolneniu okna.

Čas strávený samotným posielaním segmentov tentoraz môžeme zanedbať, pretože odosielanie v tomto prípade funguje štýlom zreťazenej linky. Určitý čas po odoslaní prvého segmentu príde jeho potvrdenie, čím sa uvoľní miesto v okne a teda začne odosielanie ďalšieho segmentu. Pri ukončení odosielania tohto segmentu však dôjde k prijatiu potvrdenia druhého segmentu, ktorý bol odoslaný v okne s prvým a teda sa opäť uvoľní okno. Takto to pokračuje až do momentu, kým nebol potvrdený posledný segment z prvého okna.

Priepustnosť zodpovedá rovnici:

$$p = w/RTT \tag{5.2}$$

Kde p je hodnota priepustnosti, w je veľkosť okna a RTT predstavuje čas od poslania segmentu po prijatie potvrdenia daného segmentu.

Ako už bolo spomenuté v sekcii 5.1.1, veľkosť okna je limitovaná dostupnou pamäťou. Na čipe z rodiny Spartan 6 je 232 kilobajtov pamäte. Preto napríklad pri oneskorení potvrdení segmentov o 4 milisekundy je priepustnosť implementácie na tomto čipe podľa rovnice 5.2 rovná 58 megabajtov za sekundu alebo 464 megabitov za sekundu.

Skutočná priepustnosť je navyše ovplyvnená stratovosťou segmentov a premenlivosťou oneskorení potvrdení. Vypočítané hodnoty predstavujú hornú hranicu priepustností.

Kapitola 6

Záver

Táto bakalárska práca je zameraná na návrh a implementáciu hadvérovej jednotky schopnej sieťovej komunikácie s využitím protokolov TCP a IP. Hlavný dôraz bol kladený hlavne na správnosť a rýchlosť riešenia.

Riešenie som začal naštudovaním a rozborom potrebných teoretických poznatkov. Ide najmä o základné znalosti o dnešných počítačových sieťach a znalosti o protokoloch TCP a IP, ktoré tvoria základ riešenia.

Následne som, opierajúc sa o získané znalosti, vytvoril návrh výsledného riešenia. Návrh vychádzal z môjho už predtým dokončeného komponentu UDP exportéra, ktorý vznikol v rámci projektu Sec6Net[1]. Najprv som vytvoril celkový návrh komponentu, v ktorom som rozdelil funkcionalitu do jednotlivých blokov. Následne som podrobnejšie navrhol jednotlivé bloky a tým som aj špecifikoval ich rozhranie a spôsob, akým budú komunikovať s ostatnými komponentmi. V tejto etape som taktiež navrhol spôsob, akým budú niektoré problémové časti riešené. Navyše som niektoré časti, ako napríklad počítanie RTO, zjednodušil tak, aby ich implementácia v hadvéri bola jednoduchšia, efektívnejšia a rýchlejšia. Výsledný návrh bol potom prezentovaný na konferencii Student EEICT 2014 [6].

Po vytvorení samotného návrhu som prešiel k implementácii. Snažil som sa čo najviac držať návrhu, tak aby sa zdrojové kódy a samotná implementácia čo najviac zhodovali s návrhom.

Popri samotnej implementácii zároveň vznikali testovacie prostredia pre jednotlivé komponenty. Tieto prostredia mali za úlohu overiť, či implementácia zodpovedá návrhu.

Po ukončení implementácie som vytvoril aj testovacie prostredie pre celý výsledný komponent. S využitím tohto prostredia som v simuláciach ladil a opravoval implementáciu až pokiaľ nefungovala správne voči podmienkam vytvoreným týmito prostrediami.

Ďalším krokom bolo verifikovať riešenie. Na tento účel vznikli verifikačné prostredia. Tieto prostredia sú veľmi podobné tým testovacím, avšak pokrývajú rozsiahlejšie stavové priestory.

Nakoniec som vytvoril ešte softvérový nástroj, ktorý slúži na zápis a čítanie jednotlivých položiek hlavičiek pomocou AXI4-Lite zbernice.

Výsledná implementácia je schopná fungovať na frekvencii až 87.230 MHz a zaberá len 5% zdrojov na čipe xc6slx45-3fgg484 z rodiny Spartan 6. Na tomto čipe dokáže teoreticky dosiahnuť priepustnosti až 2560 megabitov za sekundu.

Výsledné riešenie bude použité ako náhrada UDP exportéra. Počíta sa teda z reálnym využitím na reálnej sieti. Plánované pokračovanie práce v budúcnosti je práve otestovať

[1] Projekt Sec6Net, dostupné na: http://www.fit.vutbr.cz/research/view_project.php.cs?id=517.

riešenie nasadením na skutočnej sieti a prípadne optimalizovať riešenie. Následne sa predpokladá prechod na novú platformu. Implementáciu preto bude potrebné upraviť pre túto platformu.

Literatúra

[1] Braden, R.: Requirements for Internet Hosts - Communication Layers. RFC 1122, October 1989.
URL http://www.ietf.org/rfc/rfc1122.txt

[2] Cisco Networking Academy: *CCNA Exploration Course Booklet*. Course Booklets, Cisco Press, September 2009, ISBN 978-1-58713-234-8.

[3] Henderson, T.; Floyd, S.; Gurtov, A.; aj.: The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582, 2012.
URL http://www.ietf.org/rfc/rfc6582.txt

[4] Information Sciences Institute of University of Southern California: TRANSMISSION CONTROL PROTOCOL: PROTOCOL SPECIFICATION. RFC 793, September 1989.
URL http://www.ietf.org/rfc/rfc793.txt

[5] Jacobson, V.; Braden, R.; Borman, D.: TCP Extensions for High Performance. RFC 1323, 1992.
URL http://www.ietf.org/rfc/rfc793.txt

[6] Kekely, M.: TCP/IP communication in FPGA. In *Proceedings of the 20th Conference STUDENT EEICT 2014*, Brno, CZ: Brno University of Technology, 2014.

[7] Paxson, V.; Allman, M.; Chu, J.; aj.: Computing TCP's Retransmission Timer. RFC 6298, 2011.
URL http://www.ietf.org/rfc/rfc6298.txt

[8] Postel, J.: INTERNET PROTOCOL: PROTOCOL SPECIFICATION. RFC 791, September 1981.
URL http://www.ietf.org/rfc/rfc791.txt

[9] Xilinx: Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC. UG 074, 2010.
URL http://www.xilinx.com/support/documentation/user_guides/ug074.pdf

[10] Xilinx: AXI Reference Guide. UG 761, 2011.

[11] Xilinx: LogiCORE IP AXI Ethernet (v2.01a). DS 759, 2011.
URL http://www.xilinx.com/support/documentation/ip_documentation/axi_ethernet/v2_01_a/ds759_axi_ethernet.pdf

[12] Xilinx: Spartan-6 FPGA Block RAM Resources. UG 383, 2011.
URL http://www.xilinx.com/support/documentation/user_guides/ug383.pdf

[13] Xilinx: LogiCORE IP AXI4-Lite IPIF (v1.01.a). DS 765, 2012.
URL http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif_ds765.pdf

Dodatok A

Obsah CD

Priložené CD obsahuje:

- doc/ - zdrojové súbory pre preklad pomocou LaTeX.
- HW/ - zdrojové súbory hadvérovej časti riešenie a súbory pre simulácie.
- res/ - výpisy zo syntézy hadvérových častí, ktoré obsahujú okrem iného aj spotrebu zdrojov a maximálnu dosiahnuteľnú frekvenciu.
- SW/ - zdrojové súbory softvérovej časti riešenia a knižnice potrebné pre preklad.
- bp_xkekel01.pdf - text bakalárskej práce.
- README - súbory obsahujúce podrobnejší popis obsahu jednotlivých priečinkov a návod k prekladu a spusteniu riešenia.