VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ŘÍZENÍ TOKU DAT NA ÚROVNI TRANSPORTNÍ VRSTVY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

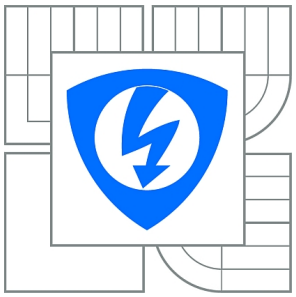
AUTOR PRÁCE
AUTHOR

Bc. MICHAL PÁNEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ŘÍZENÍ TOKU DAT NA ÚROVNI TRANSPORTNÍ VRSTVY

DATA-FLOW CONTROL ON TRANSPORT LAYER

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

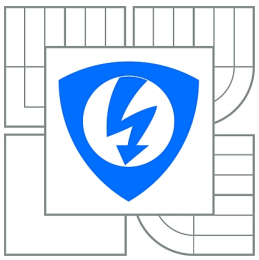
AUTOR PRÁCE — Bc. MICHAL PÁNEK

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. JAROSLAV KOTON, Ph.D.

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Michal Pánek	***ID:***	136570
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Řízení toku dat na úrovni transportní vrstvy

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte mechanismy řízení toku dat pracující na úrovni transportní vrstvy v součinnosti s protokolem TCP. Zaměřte se především na aktuální algoritmy určené pro správu velikosti okna vysílací entity TCP. Tyto algoritmy kriticky porovnejte, definujte místo jejich nasazení a popište konfiguraci volitelných metod implementace protokolu na straně vysílací a přijímací entity. Pozornost pak věnujte popisu mechanismu TCP Vegas, kdy na základě změny jeho parametrů proveďte jeho hlubší analýzu.

DOPORUČENÁ LITERATURA:

[1] ALLAN, M., PAXSON, V., BLANTON, E.: TCP Congestion Control, Networ Working Group, Standard Track RFC 5681, 2009.
[2] PUŽMANOVÁ, R.: TCP/IP v kostce, 2. vydání KOPP, ISBN 978-80-7232-388-3, 2010.

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: doc. Ing. Jaroslav Koton, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Aby bolo možné ľahko odosielať data medzi dvoma koncovými prvkami bez preťaženia, sú potrebné metódy, ktoré vhodne riadia tok dát a sú schopné vhodne vyhodnotiť možné stavy preťaženia. Takéto metódy na kontrolu dátového toku sú priamo riešené na transportnej vrstve. Táto vrstva ponúka radu mechanizmov zameranú na riešenie tejto problematiky. Cieľ tejto práce je rozdelený do troch častí. Prvá časť opisuje integráciu transportnej vrstvy TCP/IP modelu, a v schopnosti spracovať TCP dátového toku. Druhá časť popisuje metódy pre riadenie preťaženia, ich začlenenie užívaním prostredia. Zameriava sa najmä na metódy TCP Reno a TCP Vegas. Ich simulácie a analýza prenosu dátového toku. Tretia časť sa zaoberá detailnou analýzou TCP Vegas. Analyzuje možné parametre α a β rámci TCP Vegas, a kombináciou TCP Vegas a TCP Reno.

KĽÚČOVÉ SLOVÁ

TCP Vegas, TCP Reno, TCP Thaoe, transportná vrstva, kontrola zahltenia, TCP, okno zahltenia, CWND

ABSTRACT

In order to easily send data between two end elements without congestion, methods that suitably control flow of date and evaluate possible overload state are necessary. One such method is to control the data flow directly on the transport layer. This layer offers a range of mechanisms dedicated to deal with this issue. The aim of this paper is divided into three parts. The first part describes the integration of transport layer TCP/IP model, and the ability to process TCP data stream. The second part describes methods to manage congestion, their integration by usage environment. It mainly focuses on methods of TCP Reno and TCP Vegas. Their simulation and analysis on transmission the data stream stream. The third part deals with the analysis in detail of TCP Vegas. Analyzes possible parameters for α a β within the TCP Vegas, and a combination of TCP Vegas and TCP Reno.

KEYWORDS

TCP Vegas, TCP Reno, TCP Thaoe, transport layer, congestion control, TCP, congestion window, CWND

PÁNEK, Michal *Řízení toku dat na úrovni transportní vrstvy*: diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 77 s. Vedúci práce bol doc. Ing. Jaroslav Koton, Phd.

PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „Řízení toku dat na úrovni transportní vrstvy“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád bych poděkoval vedoucímu diplomové práce panu Doc. Ing. Jaroslavovi Kotonovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v tejto diplomovej práci bol realizovaný v laboratóriách podporených projektom SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno ....
(podpis autora)

OBSAH

ZOZNAM OBRÁZKOV

ÚVOD

V októbri roku 1986 došlo počas prenosu dát k viacerým zahlteniam siete čo viedlo kolapsom. Dôsledkom týchto kolapsov bola niekoľko násobne znížená priepustnosť dát[6].

Aby bolo možné bez problémov posielať dáta medzi dvomi koncovými prvkami bez vzniku zhltenia, je potrebné tento tok dát určitým spôsobom usmerniť a kontrolovať možné stavy preťaženia. O usmernenie toku dát a jeho kontrolou sa stará transportná vrstva TCP/IP modelu. Na transportnej vrstve operujú dva protokoly, UDP, (User Datagram Protocol) a TCP (Transmission Control Protocol). Z týchto dvoch protokolov, je TCP schopné riadiť dátový tok, jeho priepustnosť a riešiť možné prípady preťaženia v sieti. Tento protokol je za pomoci navrhnutých metód, schopný riešiť kritické stavy priepustnosti dátového toku. Primárnou úlohou je snaha zabrániť vzniku zahltenia pri prenose dát, jednotlivé metódy umožňujú pristupovať k problému vzniknutého preťažením a správajú sa rôznorodo.

Metódy sa časom vyvíjali a prispôsobovali rastúcim požiadavkom prenosových rýchlosti dát doby. Úlohou práce je preskúmať jednotlivé metódy používané na transportnej vrstve, ich možnosti a určiť ich individuálny prístup k dátovému toku.

1 ÚDEL TRANSPORTNEJ VRSTVY V MODELU TCP/IP

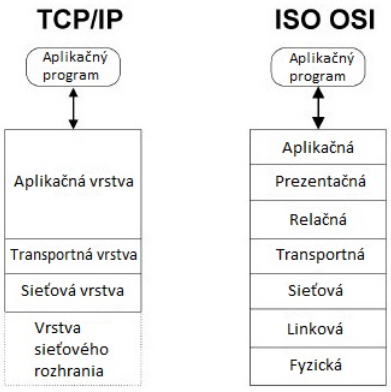
Transportná vrstva je jedna z vrstviev tvoriacích TCP/IP[1] model používaný v novodobých sieťach a sieťových prvkov. Je nástupcu modelu ISO/OSI ktorý sa používal v ranných dobách.

1.1 Referenčný model TCP/IP

Sú známe dva referenčné modely TCP/IP a ISO/OSI model. Zatiaľ čo sa ISO/OSI model riadil myšlienku zložitých sieti a jednoduchých koncových prvkov, TCP/IP sa zameriaval na jednoduché siete a inteligentné koncové body. Vzrastom požiadavkov na siete, sa stal prijateľnejší model TCP/IP a tým úplne nahradil[1].

1.1.1 TCP/IP

Skladá sa zo štyroch vrstiev a je využívaný dnešným internetom. Štyri vrstvy sú: Vrstva sieťového rozhrania, Sieťová vrstva, Transportná vrstva a Aplikačná vrstva. Vrstva sieťového rozhrania do seba spája Fyzickú a Linkovú vrstvu, Aplikačná vrstva do seba zhrňuje aj vrstvu Relačnú a Prezentačnú v prípade ich potreby je schopná Aplikačná vrstva tieto vrstvy poskytnúť. Na Obr.1.1 je možne vidieť štruktúru modelov TCP/IP a ISO/OSI.



Obr. 1.1: Modely TCP/IP a ISO/OSI.[1]

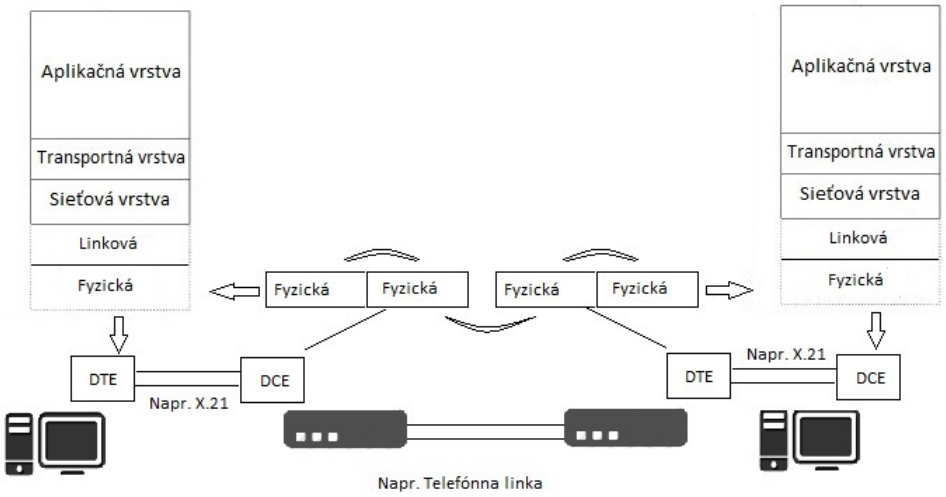
[1] 3. vrstva TCP/IP modelu a 4. vsrtva ISO/OSI modelu.

1.1.2 Fyzická vrstva

Vrstva sieťového rozhrania - Fyzická vrstva je najnižšia vrstva modelu a rieši prenos elektrických, elektromagnetických alebo optických signálov pri komunikácií. Zároveň špecifikuje aj tvar konektorov používaných prepojovacími káblami. Protokoly používané vrstvou špecifikujú:

- Elektrické signály
- Tvary konektorov
- Typ média
- Prenosovú rýchlosť
- Moduláciu
- Kódovanie
- Typ synchronizácie

Vrstva sa stará a je zodpovedná za aktiváciu komunikácie a deaktiváciu fyzického spojenia, toto spojenie môže byť vytvorené vo forme dátového okruhu medzi DTE a DCE[2]. Dátový okruh predstavuje komunikačnú cestu po fyzických médiach. Fyzicky prenos môže prebiehať v plnom alebo polovičnom duplexu a môže isť o spojenie dvojbodové alebo mnohobodové. Na Obr.1.2 je možné vidieť komunikáciu na Fyzickej vrstve.



Obr. 1.2: Komunikácia na úrovni Fyzickej vrstvy.[1]

V tejto kapitole sa použili poznatky z [1].

[2]DTE si môžeme predstaviť ako počítač a DCE ako modem

1.1.3 Linková vrstva

Zaisťuje výmenu dát medzi susednými počítačmi pri sériových linkách a v prípade lokálnej siete sa stará o výmenu dát v tejto sieti. Poskytuje jedno alebo viacero spojení medzi dvoma sieťovými entitami v susedných systémoch, spojenie sa vytvára a ruší dynamicky. Vrstva musí umožňovať:

- zahájenie, udržanie a ukončenie spojenia
- rozvetvenie dátových spojení
- formátovanie rámcov, detekciu a opravu chýb, riadenie toku
- identifikáciu koncových bodov spojenia
- zoraďovanie prenášaných rámcov
- oznamovanie neopraviteľných chýb sieťovej vrstve
- **fyzické adresovanie**

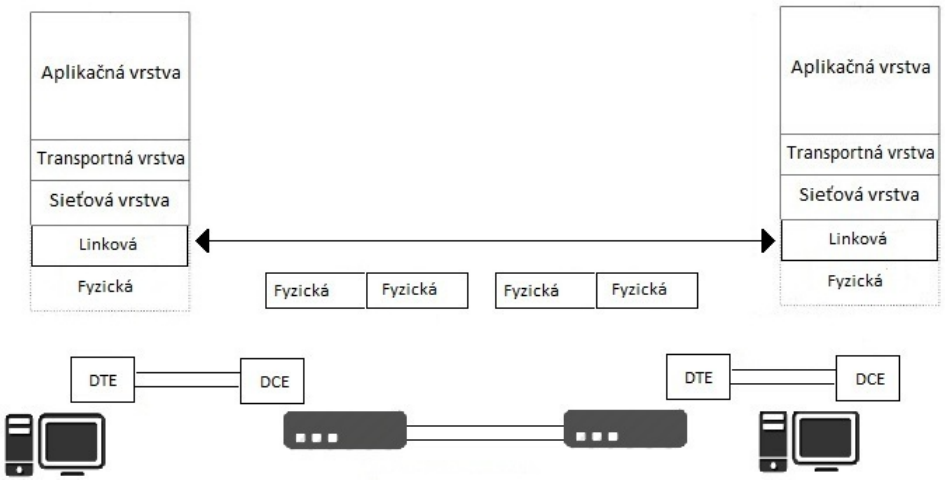
Linková vrstva[3] je jednou z vrstiev ktorá prenáša dáta po blokoch, používa *linkový rámec*[4]. Na Obr.1.3 je zobrazená stavba používaného rámca.

Obr. 1.3: Linkový rámec.[1]

Rámec sa zkladá zo **záhlavia(Header)**, **Data(Payload)** a **zápätia(Trailer)**. Pri komunikácií je rámec uvedený synchronizačnou sekvenciou, za ním nasleduje záhlavie ktoré v sebe obsahuje údaje pre prenos rámca(adresu príjemca, adresu odosielateľa...), v dátovej časti sa nachádzajú zapuzdrené[5] prenášané dáta, celý rámec je zakončený zapätim, ktoré obsahuje kontrolný súčet[6]. Obr.1.4 zobrazuje komunikáciu na linkovej vrstve, kde vrstva „nevidí“ fyzickú vrstvu. Pre jednoduchšiu realizáciu sa linková vrstva delí do dvoch pod vrstiev[2]:

- **podvrstva riadenia logického spoja** (LLC) - poskytuje rozhranie medzi konkrétnym prenosovým prostriedkom a vyššími vrstvami(susedí so sieťovou vrstvou).
- **podvrstva riadenia prístupu k prenosovému prostriedku** (MAC) - poskytuje funkcie pre daný prenosový prostriedok (susedí s fyzickou vrstvou). Adresa MAC[7] označuje stanicu - každé fyzické pripojenie k sieti. Adresa má

[3]často tiež označovaná ako spojová vrstva
[4]obecne sa jednotky v referenčnom modelu označujú ako dátové pakety
[5]dátové pakety sa v rámci referenčného modelu zapuzdrujú pre prenos do seba z hora nadol.
[6]umožňuje rozpoznať či došlo k poškodeniu rámca počas prenosu, napr. CRC, MD5, parita
[7]jedinečná pre každé zariadenie

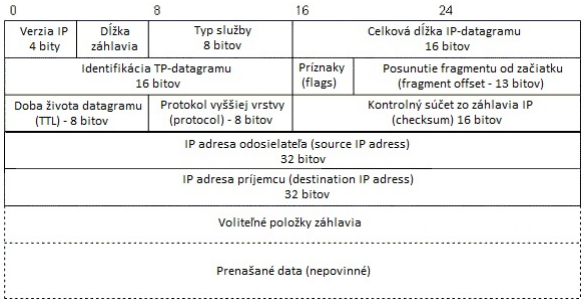


Obr. 1.4: Komunikácia na úrovni linkovej vrstvy.[1]

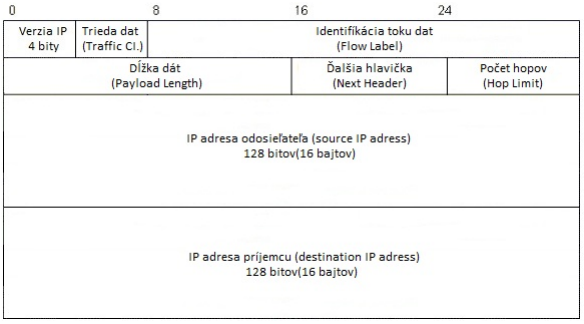
dĺžku 48 bitov, skladá sa z dvoch častí, z kódu výrobcu OUI(24 bitov) a označenia fyzického rozhrania (24 bitov).

1.1.4 Sieťová vrstva

Často nazývaná aj IP vrstva sa stará o prenos dát medzi vzdialenými počítačmi v internetu. Na základe sieťovej adresácie (logické adresovanie) sa vrstva zo základu stará o **komunikácie v komplexnej sieti**, **smerovanie**, **prenos paketov**. Základnou jednotkou je *IP datagram(paket)* zložený zo **Záhlavia** a **Dátového poľa**. V rámci Internetu sa používajú 2 varianty datagramu a to IP datagram IPv4 a IPv6. Na Obr.1.5 a 1.6 je možné vidieť ich štruktúru. Z obrázkov je možné vidieť rozdielnu hlavičku. IPv6 sa snaží poskytnúť väčší adresovací priestor, ktorý už IPv4 so svojou 32 bitovou adresáciou poskytnúť nedokáže (2^{32} adries) z toho dôvodu je adresa u IPv6 128 bitová (2^{128} adries). Na Sieťovej vrstve je medzi susednými smerovačmy

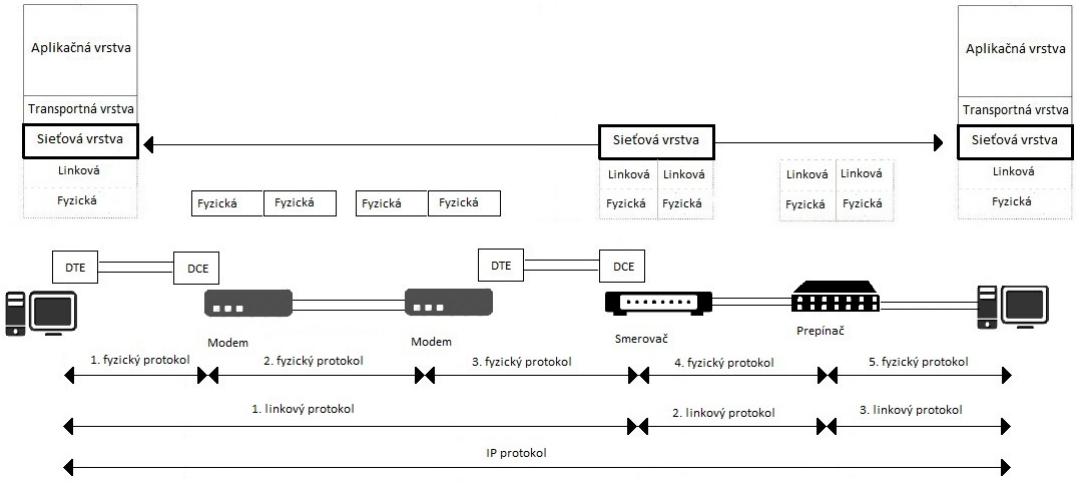


Obr. 1.5: Datagram IPv4.[1]



Obr. 1.6: Datagram IPv6.[1]

vždy priame spojenie. Prichádzajúci sieťový paket, smerovač najprv rozbalí z rámca a pri odosielaní ho opäť zabalí. Na Obr.1.7 je vidieť komunikáciu na úrovni Sieťovej vrstvy, pričom sama vrstva nevidí zariadenia pracujúce na vrstvách pod ňou a zároveň ju ani nezaujíma aké protokoly tieto vrstvy použili. Každý prenesený IP datagram má vo svojej hlavičke adresu príjemcu čo je základná informácia pre prenos týchto dát. Tým že obsahuje každý datagram tieto informácie, môže byť prenášaný osobitne[1][2].



Obr. 1.7: Komunikácia na úrovni Sieťovej vrstve.[1]

1.1.5 Transportná vrstva

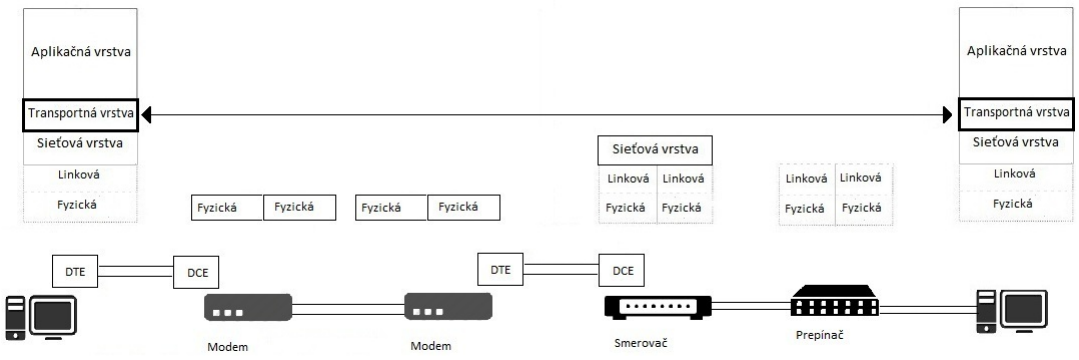
Stará sa o predávanie dát medzi aplikáciami na vzdialených počítačoch, poskytuje transparentný prenos s možnosťou dosiahnúť požadovanú kvalitu a je schopná optimalizácie pre rôzne sieťové služby. Vrstva sa spolieha na služby nižších vrstiev, a nevidí žiadne zariadenia ani sieťové infraštruktúry. Kvalitu služieb, ktorú je schopná vrstva poskytnúť závisí na triede služieb. Triedy sú charakterizované kombináciou viacerých parametrov, ako sú priepustnosť, doba prenosu, zbytková chybovosť. Zvolená kvalita je následovne dodržiavaná po dobu celého spojenia.

Disponuje funkciami:

- adresovanie (zobrazenie transportných adries na sieťové)
- multiplexovanie
- rozvetvenie transportných spojení
- koncové riadenie poradia
- koncová detekcia chýb a ich oprava
- formatovanie(fragmentácia)
- riadenie toku

Adresovanie

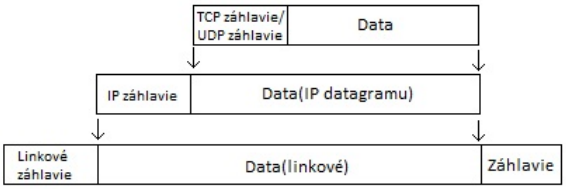
Priradí sieťovú adresu transportnej vrstve, ktorá slúži odpovedajúcej entite požadujúcej transportné spojenie. Transportná entita je schopná obsluhovať viacero požadujúcich entit. Jedna sieťová adresa môže združovať viacero transportných adries, v rámci transportnej entity. Na Obr.1.8 je možné vidieť komunikáciu na úrovni Transportnej vrstvy s ohľadom na ostatné vrstvy.



Obr. 1.8: Komunikácia na úrovni Transportnej vrstvy.[1]

Transportná vrstva poskytuje **transportnú službu so spojením**(TCP) a **transportnú službu so bez spojenia**(UDP). Na transportnej vrstve operujú protokoly UDP a TCP. Protokoly sa následovne zapuzdrujú do IP datagramu Obr.1.9[1][2]:

- **UDP** - umožňuje prenos blokov bez kontroly
- **TCP** - umožňuje nadviazanie spojenia jeho udržanie a ukončenie



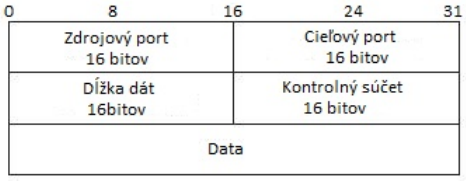
Obr. 1.9: Zapuzdrenie protokolov transportnej vrstvy.[1]

K svojej činnosti používajú porty ako adresy pre jednotlivé aplikácie, porty sa delia do troch kategórií:

- **Známe porty** - porty v rozsahu 0 až 1023, vyhradené pre najbežnejšie služby
- **Registrované porty** - v rozsahu 1024 až 49151, potrebná registrácia u organizácie ICANN
- **Dynamické a súkromé porty** - v rozsahu 49152 až 65535, možno ľubovoľne používať

UDP(User Datagram Protocol)

Dá sa označiť ako nespojová nespoľahlivá služba. Základnou jednotkou je *UDP datagram*. Funguje na princípe, kedy sa odosielateľ nestará o to či datagrami dorazili alebo či sa po ceste datagram stratil. Posiela svoje datagrami do doby kým všetky neodošle a spolieha sa, že vzniklé problémy vyrieši aplikačný protokol. Obr.1.10 popisuje jeho stavbu, ktorá je oproti TCP oveľa jednoduchšia.



Obr. 1.10: Stavba UDP datagramu.[1]

1.1.6 Aplikačná vrstva

Poskytuje komunikačný systém aplikačným procesom a snaží sa o umožnenie vzájomnej spolupráce. Aplikačné protokoly a spoliehajú na protokoly UDP a TCP.

Aplikačná vrstva požiada jeden z protokolov Transportnej vrstvy o vytvorenie dátového toku a následovne do neho vloží dáta aplikácií. Aplikačná vrstva poskytuje služby:

- Prenos správ
- Identifikácia komunikujúcich parametrov
- Zistenie stupňa okamžitej pripravenosti komunikujúceho partnera
- Stanovenie poverenia pre komunikáciu
- Mechanizmy ochrany správ
- Synchronizácia spolupracujúcich aplikácií
- Výber spôsobu dialógu, jeho spôsob nadviazania a ukončenia
- Dohoda o tarifov
- Dohoda o obmedzení syntaxe správ
- Dohoda o zodpovednosti za opravu chýb

Aplikačných protokolov je obrovské množstvo, dajú sa rozdeliť na dve skupiny:

- **Užívateľské protokoly** - používané koncovými uživateľmi(HTTP,FTP...)
- **Služobné protokoly** - používané pre správu Internetu a jeho správnu funkciu(smerovacie tabuľky, RADIUS...)

1.2 Transmission Control Protocol-TCP

Protokol Transportnej vrstvy, označovaný tiež ako spojový a spoľahlivý. Vytvára spojenie tzv. virtuálny okruh, medzi koncovými aplikáciami a tak zabezpečí medzi nimi spoľahlivý prenos. Všetky prenášané bajty sú číslované, takže v prípade straty alebo poškodenia je príjemca schopný tieto data znovu vyžiadať.

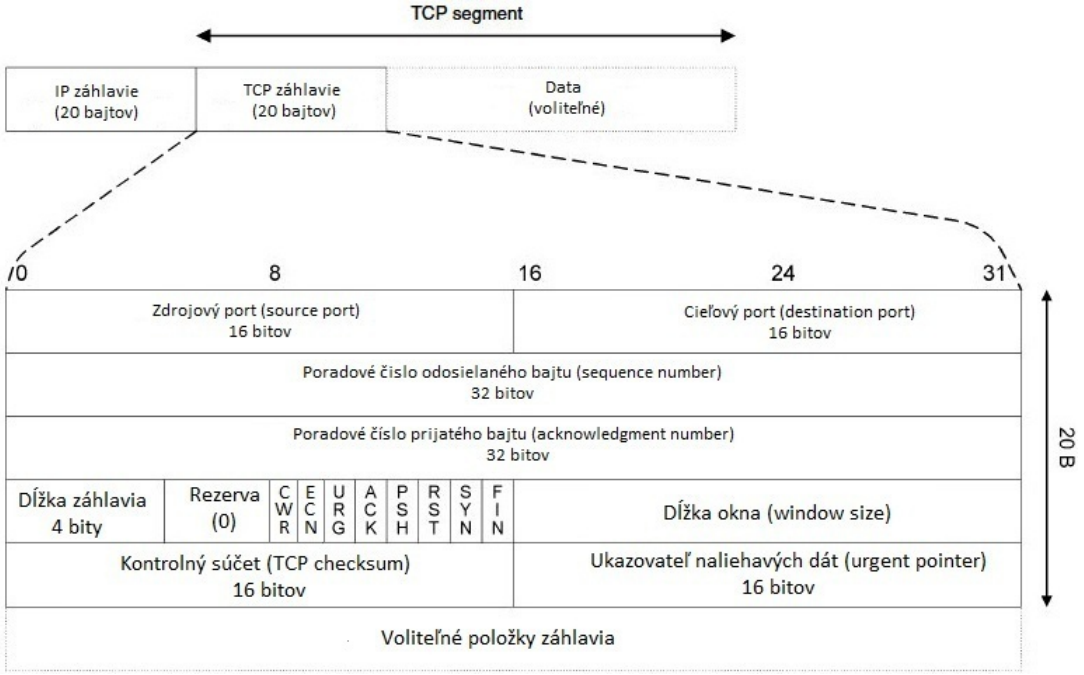
TCP disponuje týmito základnými vlastnosťami:

- **spoľahlivá transportná služba**, doručí danému cieľu všetky data, ktoré boli odoslané od zdroja. Pre potvrdenie správneho prijatia, sa používa pozitívne potvrdzovanie a pri strate alebo poškodení schopnosť použiť opätovné poslanie dát.
- **vytvorenie služby so spojením**, fáza zahájenia spojenia a jeho ukončenie (s využitím 3-way handshake pre vytvorenie spojenia).
- **efektívne využitie prenosových kanálov**, vysielanie prebieha s myšlienkou použitia vyrovnávacej pamäte, odosielateľ pred vlastným odoslaním, nazhromaždil dostatok dát alebo mu vyprší časový limit.
- **transparentný prenos**, môže prenášať ľubovoľný počet dát vyššieho protokolu.

- **duplexné spojenie**, komunikácia v oboch smeroch, kedy TCP posiela spätné potvrdenia.
- **rozlišovanie viacerých adresátov**, využíva porty aby sa dali rozlíšiť komunikácie pre rôzne aplikácie bežiace na jednom koncovom zariadení

1.2.1 TCP segment

Základnou prenosovou jednotkou je, ktorý sa následovne zapúzdri (podobne ako UDP) do IP datagramu Obr.1.9. Hlavička TCP segmentu ma 20 bajtov, na Obr.1.11 je vidieť štruktúru TCP segmentu, ktorá je už z prvého pohľadu oveľa rozsiahlejšia než štruktúra, ktorou disponuje UDP.



Obr. 1.11: Stavba TCP segmentu.[1]

- **zdrojový port**, označuje port odosielateľa ide o identifikáciu zdrojového aplikačného procesu.
- **cieľový port**, označuje port príjemcu ide o identifikáciu cieľového aplikačného procesu[8].
- **poradové číslo odosielaného bajtu**, je to číslo prvého bajtu v TCP segmentu toku dat smerom od odosielateľa k príjemcovi. Pri nadviazaní spojenia sa začína náhodným poradovým číslom označovaným ako ISN.

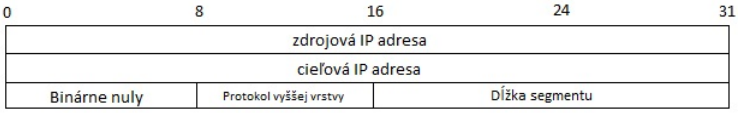
[8]zdrojový a cieľový port spolu s IP adresáta a príjemcu jednoznačne identifikujú spojenie v Internete v danom okamžiku.

- **poradové číslo prijatého bajtu**, vyjadruje číslo bajtu, ktorý je následovne očakávaný. Týmto príjemca potvrdzuje, Že boli správne prijaté všetky bajty do poradového čísla prijatého bajtu-1.
- **dĺžka záhlavia**, vyjadruje sa v násobkoch 32 bitov.
- **dĺžka okna**, vyjadruje počet bajtov, ktoré je možné vysielať bez priebežného potvrdzovania, bude príjemcom ešte akceptovaný.
- **príznaky**, jedná sa o 8 bitové pole, kde každý príznak plní určitú funkciu:
  1. **CWR** - Potvrdí prijatý segment s nastaveným príznakom ECN-Echo. Po prijatý už príjemca nenastavuje u odosielaných segmentov príznak ECN-Echo.
  2. **ECE**(ECN-Echo) - Potvrdí prijatie IP datagramu s príznakom ECN. Príznak je nastavený u všetkých segmentov do doby prijatia CWR. Umožňuje prenášať oznámenia o zahltení siete.
  3. **URG** - označuje segment s urgentnými datami.
  4. **ACK** - potvrdzuje platnosť Poradového čísla prijatého bajtu.
  5. **PSH** - signalizuje alebo požaduje okamžité doručenie segmentu protokolu vyššej vrstvy.
  6. **RST** - signalizuje odmietnutie TCP spojenia
  7. **SYN** - signalizuje, je žiadosťou o nadviazanie nového spojenia.
  8. **FIN** - signalizuje, je žiadosťou o ukončenie spojenia.
- **ukazovateľ naliehavých dát**, používa sa v prípade nastavenia príznaku URG, označuje posledný bajt urgentných dát.
- **kontrolný súčet**, zabezpečenie celého segmentu. Je potrebný párny počet bajtov, ak tomu tak nieje, doplní sa o jeden bajt do párneho počtu. Počíta sa zo pseudo záhlavia.
- **voliteľné položky záhlavia**, obsahujú doplnkové informácie o prenášanom segmente s maximálnou veľkosťou 40 bajtov (maximálna veľkosť segmentu, časové razítko...).

Pseudo záhlavie

Používa sa pre zabezpečenie (kontrolný súčet) segmentu, obsahuje IP adresu zdroja a cieľa ako aj číslo protokolu a dĺžku pôvodného segmentu (dĺžka segmentu bez pseudo záhlavia). Ide o fiktívne záhlavie ktoré sa pre výpočet dáva pred samotné záhlavie TCP[9] je možné ho vidieť na Obr.1.12

[9]pseudo záhlavie sa používa pri výpočte kontrolného súčtu aj u UDP.



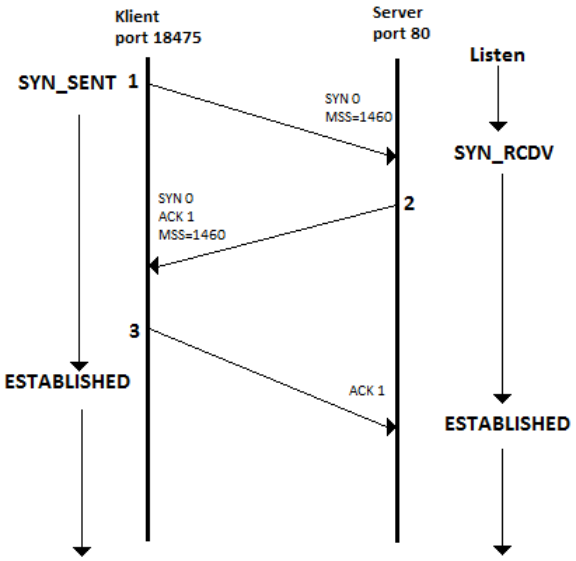
Obr. 1.12: Pseudozáhlavie.[2]

1.2.2 Fáze komunikácie TCP

TCP rozlišuje tri fáze spojenia, tými sú **nadviazanie spojenia, samotný prenos dát a ukončenie existujúceho spojenia**.

Nadviazanie spojenia

Každá dátová komunikácia ktorá pre svoj prenos použije protokol TCP, musí pred vlastným prenosom dát preniesť trojicu segmentov. Tento proces sa volá **three-way handshaking** a je to proces nadväzovania spojenia. Obr.1.13 znázorňuje nadväzovanie spojenia, ktoré sa skladá z troch správ **SYN, SYN-ACK a ACK**.



Obr. 1.13: 3-way handshake.[1]

Klient komunikuje so serverom a vysiela segment **1** chce , kde port klienta je 18475 a port serveru 80. Klient vygeneruje náhodne číslo v intervalu 0 až $2^{32} - 1$ a toto číslo použije ako štartovacie poradové číslo. Následovne nastavy príznak SYN, aby server vedel že sa jedná o nadväzovanie spojenia. Klient odošle segment s príznakom SYN a MSS na stranu serveru. MSS pošle ako oznámenie akú veľkú dátovú časť si praje prijať. Pomocou segmentu **2**, odpovedá server na prijatý segment podobne

ako klient vygeneruje náhodné číslo a posiela na stranu klienta príznak SYN a MSS, spolu s potvrdením ACK už prijatého bajtu. Veľkosť MSS sa dohaduje v prvých dvoch segmentoch. Posledný segment **3** je potvrdenie zo strany klienta na prijatý bajt. Po prevzatí potvrdenia serverom, došlo k úspešnému nadviazaniu spojenia a môže byť zahájený vlastný prenos dát. Pri nadväzovaní spojenia sa môže klient spolu so serverom nachádzať v rôznych stavoch:

- Server:
  1. LISTEN (Poslúcha) - server poslúcha a je pripravený na spojenie.
  2. SYN_RCVD - server obdržal prvý TCP segment s príznakom SYN.
  3. ESTABLISHED - server nadviazal spojenie a je pripravený na prenos.
- Klient:
  1. SYN_SEND - klient odoslal prvý segment s príznakom SYN.
  2. ESTABLISHED - klient nadviazal spojenie a je pripravený na prenos.

Prenos dát

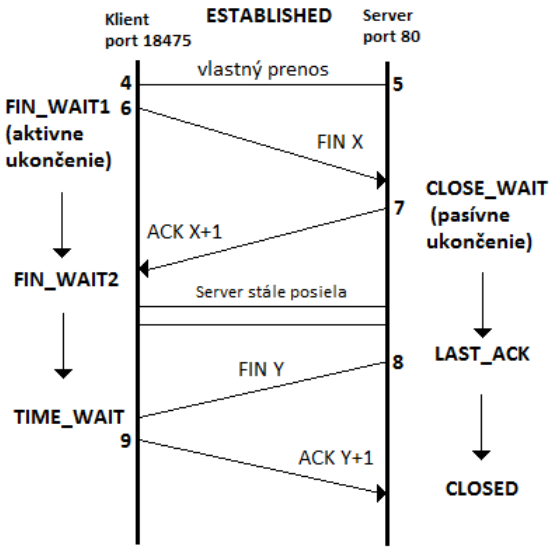
Nastáva okamžite po splnení nadviazania spojenia, dochádza k prenosu segmentov podľa poradových čísiel[10], pozitivného potvrdzovania a opätovného vysielania. Porty vybrané pri nadviazaní sú používané po dobu celého spojenia, vytvorený spoj trvá do doby kým nieje odoslaný segment s príznakom FIN.

Ukončenie spojenia

Ukončovanie spojenia môže zahájiť klient aj server, ukončenie sa zahajuje odoslaním segmentu s príznakom FIN. Strana, ktorá ako prvá odošle príznak FIN vytvára aktívne ukončenie, druhá strana ukončenie pasívne. Strana, ktorá vytvorí aktívne ukončenie už nieje schopná ďalej odosielať data. A však strana druhá môže v odosielaní pokračovať do doby, kým sama neuzavrie spojenie. Tento stav od aktívneho ukončenia po celkové ukončenie spojenia sa označuje ako polo uzavreté spojenie. Narozdiel od nadväzovania spojenia, kde postačili tri segmenty, je pre riadne ukončenie spojenia potreba štyri segmenty. Obr.1.14 znázorňuje ukončenie spojenia spolu so stavmi, v ktorých sa nachádzajú klient a server po dobu ukončovania. Segmenty **4 a 5** značia že prenos stále prebieha. Segmentom **6** odošle klient príznak FIN a žiada o ukončenie spojenia. Server pomocou segmentu **7** odpovedá a potvrdzuje prijatú správu. V prípade že by tento segment obsahoval tiež príznak FIN došlo by k celkovému ukončeniu spojenia, server však FIN príznak neposlal čim značí, že chce ešte v komunikácií pokračovať. Nastáva polo uzavreté spojenie. Na koniec server požiada tiež o ukončenie a vyšle segmentom **8** tiež príznak FIN. Segment **9** značí potvrdenie

[10]Poradové číslo segmentu pri prenose dát pokračuje v poradí, teda nasledujúcim číslom ktoré bolo vygenerované pre príznak SYN.

prijatia správy čím sa uzatvára spojenie. Klient a server sa počas ukončenia spojenia



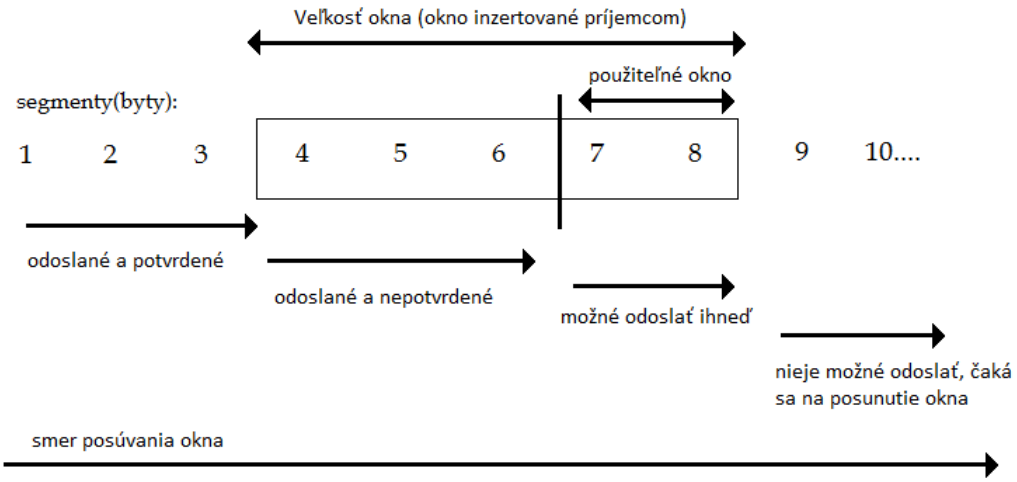
Obr. 1.14: Ukončenie spojenia.[1]

nachádzajú v rôznych stavoch:

- FIN_WAIT1 - strana prvá odosielajúca príznak FIN, už nemá dáta na odosielanie a tak žiada o ukončenie spojenia.
- CLOSE_WAIT - označuje stranu v pasívnom ukončení, strana prijala FIN ale sama segment s príznakom FIN neposlala.
- FIN_WAIT2 - strana dostala odpoveď na svoj segment s príznakom FIN, nastaví sa do stavu FIN_WAIT2 a čaká kým druhá strana nepošle segment s príznakom FIN. Pokiaľ je spojenie nečinné 11,25 minúty, spojenie sa automaticky zmení na stav CLOSED.
- LAST_ACK - strana v pasívnom ukončení odošle všetky dáta, a posiela posledný segment s príznakom FIN.
- TIME_WAIT - strana v aktívnom ukončení prijala príznak FIN, v tomto stave už sú odoslané všetky dáta a samostatný prenos už neprebieha. Potvrdí prijatý, toto potvrdenie už nieje spätne potvrdzované, a ostáva v stave TIME_WAIT 2 minúty a ukončí sa.
- CLOSED - druhá strana prijala potvrdenie svojho segmentu s príznakom FIN a prechádza do stavu CLOSED.

1.2.3 Prenos segmentov - Kĺzajúce okno

Kĺzajúce okno je technika, ktorá umožňuje odosielať skupiny bytov podľa veľkosti okna. Maximálny počet nepotvrdených bytov je daný veľkosťou okna. Vysielacia strana využíva vyrovnávacie pamäte, kde si uchováva všetky nepotvrdené segmenty, ktoré vysiela do doby kým neobdrží potvrdenie o ich doručení. Ak by potvrdenie neobdržala, odosiela znovu a opäť čaká na potvrdenie. Poradové čísla označujú poradové číslo prvého bytu dátovej časti, s ohľadom na pôvodné poradie dat. Mechanizmus okna je schopný premenlivosti, každé potvrdenie, ktoré príjme odosielateľ zároveň obsahuje informáciu o tom koľko ďalších bytov je schopný príjemca akceptovať. Podľa obdržanej informácie, je tak schopný odosielateľ upraviť veľkosť vysielacieho okna. Kĺzajúce okno sa snaží o efektívne využívanie šírky pásma, tým že obmedzuje réžiu jednotlivých potvrdení. Výkon okna určuje veľkosť okna a rýchlosť, ktorou sieť dokáže prijímať dáta. Nečinnosť siete je možné regulovať zväčšovaním okna, pretože odosielanie môže byť také rýchle ako ich len sieť dokáže prenášať[2].



Obr. 1.15: Kĺzajúce okno.[2]

Syndrom Hlúpeho okna

V prípade že by bol príjemca až moc pomalý, mohol by požadovať veľkosť okna o hodnote 1 (jeden byte). Vysielacia strana by bola tak priam nútená posielať malé segmenty rovnakej veľkosti, čo by spôsobovalo zvyšovanie réžie prenosu. Pre každý malý segment, by bolo potrebné spraviť radu operácií na strane odosielateľa aj príjemcu. Syndrom hlúpeho okna môže vzniknúť aj na strane odosielateľa, v prípade že dáta k odosielaní sa pripravujú pomalu, môže si TCP umieniť odosielanie po jednom byte. K predchádzaniu syndrómu hlúpeho okna sa využíva Nagleoveho algoritmu,

vyznačuje sa minimálnou réžiou[2]. Vyžaduje od oboch strán heuristický prístup k zamedzovaniu tohoto syndrómu. Príjemca sa musí snažiť vyhnúť požiadavke na príliš malé okno. Príjemca z pravidla počká s odosielaním potvrdení prijatých segmentov, kým sa neuvoľní viac pamäti. Podržanie potvrdení však nesmie byť príliš dlhé aby vysielacia strana neodoslala segmenty znovu. Preto oneskorenie potvrdení nesmie prekročiť 500 ms. Vysielač sa chráni proti syndrómu tak, že počká kým sa naakumuluje dostatočný objem dat a potom začne vysielať. Vyšle prvý byte spojenia, a začne vyčkávať kým nepríde potvrdenie alebo kým nieje objem akumulovaných dát aspoň vo veľkosti MSS.

1.2.4 Voliteľné implementácie

TCP umožňuje k niektorým svojím častiam individuálny prístup a nastavenie. Povoľuje voliteľné implementácie ktoré môžu byť nastavené a tak ovplyvňujú efektivitu komunikácie, pričom dvojica implementácií s rôznymi voliteľnými časťami budu stále kompatibilné[3]. Oblasťami voliteľných implementácií je:

- **Politika odosielania**
- **Politika doručenia**
- **Politika prevzatia správ**
- **Politika opakovaného vysielania**
- **Politika potvrdzovania**

Politika odosielania

Dáta prijaté TCP entitou sú pred samostatný odosielaním uložené vo vyrovnávacej pamäti. Pokiaľ tieto dáta neobsahujú dáta na okamžité odosielanie (príznak PUSH), je si schopná vysielacia entita TCP riadiť vysielanie sama. TCP je schopné zostaviť segment pre každý blok dát ktoré obdrží. Alebo vyčkáva a zbiera dáta po určitý časový interval, z ktorého vytvorí jeden segment na odoslanie. Spôsoby odosielania závisia na efektivite prenosu. Odosielanie malých segmentov zabezpečuje rýchlu odozvu systému, ale môže spôsobiť syndróm hlúpeho okna. Pričom odosielanie veľkých segmentov ma menšiu réžiu na zostavenie a spracovanie segmentu[3].

Politika doručenia

Pokiaľ prijaté dáta nemajú byť predané vyššej vrstve okamžite (príznak PUSH), entita TCP rozhodne sama kedy tieto dáta predá. Môže užívateľské dáta predať okamžite v poradí v ktorom prišli, alebo počká a ukladá segmenty do vyrovnávacej pamäte. Spôsoby doručenia závisia na efektivite prenosu. Doručovanie veľkých blokov zamedzuje zbytočným volaní funkcií a zvyšovaním nárokov na spracovanie

ako je tomu u malých blokov, zato užívateľ nedostane svoje dáta tak rýchlo akoby chcel[3].

Politika prevzatia správ

Keď prídu všetky segmenty v správnom poradí cez TCP spojenie, entita ich umiestni do vyrovnávacej pamäte pre následovne doručenie užívateľovi. Môže však nastať situácia kedy dáta neprichádzajú v správnom poradí a prijímacia entita ma na výber z dvoch možností[3]:

- **prevzatie podľa poradí (in-order)** - sú prijaté iba tie segmenty, ktoré dorazili v správnom poradí, všetky ostatné sú zahodené. Výhodami sú že nevyžaduje žiadnu zložitú implementáciu, má ale väčšie nároky na sieť lebo po vypršaní časovača sa odosielajú zahodené segmenty znovu. AK sa stratí jeden z odoslaných segmentov je nutné odoslať znovu každý už vyslaný segment.
- **prevzatie podľa okna (in-window)** - sú prijaté všetky segmenty v rámci okna. Výhodou je menší počet prenosov, ale má komplikovanejší spôsob ukladania dát do vyrovnávacej pamäte a evidovanie segmentov, ktoré prišli mimo poradie.

Politika opakovaného vysielania

V prípade, ktorom TCP neobdrží potvrdenie odoslaného segmentu v určitom časovom limite, odosiela TCP entita tieto segmenty znovu. TCP má v tomto prípade na výber z troch možnosti opätovného odosielania[3]:

- **Iba prvý (First-only)** - spravuje sa iba jeden časovač pre celú frontu, pokiaľ je prijaté potvrdenie, segmenty sú odstránené z fronty a časovač je nastavený na pôvodnú hodnotu. Pokiaľ nieje obdržané žiadne potvrdenie, je odoslaný prvý segment z fronty a časovač sa nastaví na pôvodnú hodnotu.
- **Blokovo (Batch)** - spravuje sa iba jeden časovač pre celú frontu, pokiaľ je prijaté potvrdenie, segmenty sú odstranené z fronty a časovač je nastavený na pôvodnú hodnotu. Pokiaľ nieje obdržané žiadne potvrdenie, sú odoslané všetky segmenty vo fronte a časovač sa nastaví na pôvodnú hodnotu.
- **Jednotlivo (Individual)** - každý segment vo fronte ma svoj vlastný časovač, pokiaľ je prijaté potvrdenie, segmenty sú odstránené z fronty a časovače týchto segmentov sa zrušia. V prípade že nieje obdržané potvrdenie, je vysielaný segment, ktorému vyprší časovač a tento časovač a nastaví na pôvodnú hodnotu.

Možnosť **Iba prvý (First-only)** je primerane efektívny, pretože opätovne odosiela len segment, ktorý sa stratil alebo ktorému sa stratilo potvrdenie. Následujúci segment môže byť odoslaný iba v prípade že segment ktorý je pred ním obdržal

potvrdenie. Čo značne môže zvyšovať prenos v prípade vzniklej chyby. Možnosť opätovného prenosu **Blokovo (Batch)**, je časom prenosu na tom lepšie lebo prenesie skupinu segmentov, nevýhodou je že v tejto skupine môžu byť aj segmenty ktoré opätovný prenos nepotrebujú, tým dochádza ku zbytočnému prenosu segmentov. Možnosť **Jednotlivo (Individual)** posielať segmenty rieši problémy vzniknuté u prvej možnosti a však vyžaduje vyššiu výpočetnú kapacitu[3].

Politika potvrdzovania

Pokiaľ prichádzajú segmenty v správnom poradí, TCP entita ma dve možnosti odosielania potvrdenia[3]:

- **Okamžite (Immediate)** - dáta sú prijaté, je okamžite odoslaný prázdny segment s potvrdzovacím číslom.
- **Kumulatívne (Cumulative)** - dáta sú prijaté, uloží si do pamäte, aby tieto segmenty potvrdila a vyčkáva na ďalší segment. Potom odosiela potvrdenie s potvrdzovacím číslom práve prijatého segmentu. Aby nedochádzalo k oneskoreniam a opätovným odosielaniam zo strany odosielateľa, pri čakaní na segment, používa sa časovač. Pokiaľ tento časovač vyprší je segment ktorý bol zapamätaný potvrdený prázdnym segmentom.

Okamžité odosielanie potvrdení je jednoduché a odosielateľ ma nepretržité informácie o odoslaných segmentov, to však vytvára zbytočnú záťaž. Kumulatívne potvrdzovanie tento problém rieši ale je oveľa náročnejšie vzhľadom na používané časovače.

1.2.5 Zahltenie siete a spôsoby riešenia zahltenia

Pomocou okna príjemca definuje, koľko dát je schopný v daný moment prijať. Nastáva však problém kedy je odosielateľ a príjemca na rôznej rýchlosti siete. Pokiaľ by bol odosielateľ na rýchlej sieti a príjemca na pomalej, tak sa mohol snažiť posielať dáta až po veľkosť okna. Toto by bolo príjemcov sieťou neprijateľné a ta by v tomto dôsledku zahlcovania začala dáta zahadzovať.

Okno zahltenia

TCP entita sa snaží predísť tomuto zahlcovaniu alebo v prípade vzniku zahltenia siete sa sním čo najefektívnejšie vysporiadať. Aby mohla TCP entita vôbec vedieť koľko si môže dovoliť poslať nepotvrdených dat, používa na strane odosielateľa okno. Toto okno sa volá *Okno zahltenia* (CWND) a postupne sa navyšuje,

zároveň pri vzniku zahltenia, CWND spomaľuje dátový tok. Aby samotné okno nespôsobovalo zahltenie, môže isť po maximálny nastavený limit SSTHRESH, ktorý signalizuje pravdepodobnosť zahltenia. SSTHRESH je udržovaný v násobkoch veľkosti odosielaného segmentu. Odosielateľ môže odosielať také množstvo dat, ktoré neprevyšuje okno inzerované príjemcom (WINDOW) a neprevýši ani okno CWND, snaží sa v každom okamžiku spojenia splniť podmienku[3]

$$awnd = \min[kredit, cwnd] \tag{1.1}$$

awnd, je okno povolenia, udávajúce koľko segmentov bez prijatia potvrdenia smú byť odoslané, *cwnd* je okno zahltenia a *kredit* je počet neužitých kreditov pridelených pri poslednom prijatom potvrdení vyjadrujúci segmenty vypočítaný ako *WINDOW/ dĺžka segmentu*.

Riadenie stavu zahltenia

O zahltenie sa stará TCP pomocou mechanizmu riadenia zahltenia (congestion control). Zátaž siete narastá exponenciálne s každým segmentom, ktorý bol znovu odoslaný v dôsledku nedoručenia potvrdenia. S rastúcou zátažou rastie taktiež možnosť, že dôjde ku kolapsu siete. TCP sa snaží kolapsu vyhnúť a používa zo základu mechanizmy:

- **Dynamické znižovanie veľkosti okna**
- **Pomalý štart**
- **Rýchly opakovaný prenos**
- **Rýchle zotavenie**

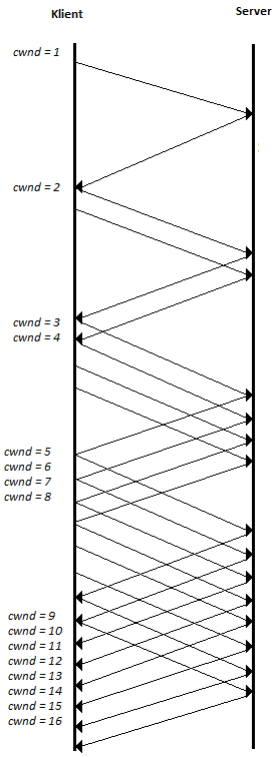
Samostatný TCP prenos sa dá rozdeliť na dve fáze[6]:

- **1. fáza Pomalého štartu** - nesú znalosti o parametroch siete, TCP nevie koľko dát si môže dovoliť preniesť.
- **2. fáza vyhýbania sa zahlteniu** - TCP si uvedomuje že prenáša dáta rýchlosťou u ktorej by mohlo dôjsť k zahlteniu.

Pomalý štart

Slúži ako základný mechanizmus pre nadviazanie spojenia a vlastného prenosu dát. V počiatočnej fázy nemá TCP žiadne znalosti o parametroch siete, čiže nevie aká môže byť veľkosť CWND. Teoreticky by mohol TCP posielať všetko čo ma hneď na maxime, ale to by viedlo k zahlteniu siete. Aby sa predišlo podobným problémom, používa sa mechanizmus pomalého štartu. Ten funguje na princípe postupného zvyšovania CWND okna. Kedy najprv nastaví $cwnd = 1$ a odošle jeden segment a počká na jeho potvrdenie. Pokiaľ potvrdenie obdrží, nastaví $cwnd = 2$ odošle dva segmenty a opäť čaká, pokiaľ obdrží potvrdenia, nastaví $cwnd = 4$ vyšle štyri

segmenty. CWND sa inkrementuje postupne za každé prijaté potvrdenie a exponenciálne navyšuje až po maximum a tým zaplní dostupnú kapacitu nepretržitým tokom segmentov[1][2][4]. Princíp pomalého štartu je možné vidieť na Obr.1.16.



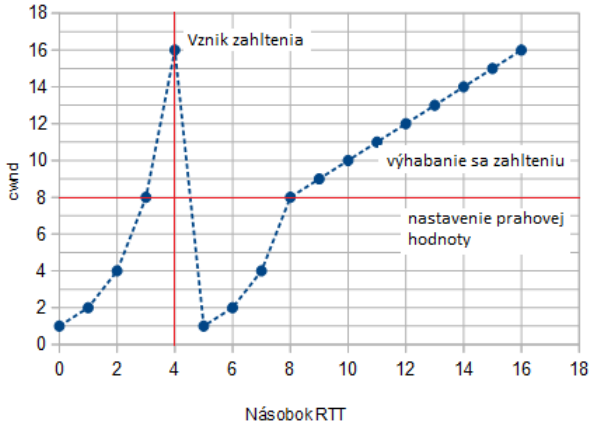
Obr. 1.16: Princíp pomalého štartu.[2]

Ovládanie okna pri zahltení

Mechanizmus pomalého štartu by mohol navyšovať okno zahltenia teoreticky do nekonečna. To však nieje možné lebo samotný mechanizmus by týmto navyšovaním spôsobil zahltenie. Preto pri indikácií zahltenia, spôsobené stratou segmentu (indikované neprijatím potvrdenia segmentu), sa veľkosť okna CWND nastaví na 1 čiže $cwnd = 1$. A spustí sa opäť mechanizmus pomalého štartu. Z uvedeného vyplíva že ide až o moc agresívny prístup, ktorý keby sa dial pri každom stratenom segmente, by prenosu dát moc neprospelo. Preto sa používa variácia pomocou prahovej hodnoty SSTHRESH. Tá sa delí na tri časti:

- **Nastavenie prahovej hodnoty** na polovicu okna CWND pri indikácií zahltenia ktorá bude $ssthresh = cwnd/2$
- **Nastavenie okna zahltenia** na počiatočnú hodnotu $cwnd = 1$, a spustenie pomalého štartu do bodu kedy okno zahltenia bude rovné prahovej hodnote $cwnd = ssthresh$ (okno zahltenia rastie exponenciálne).

- **Prednastavením rastu okna zahltenia** kde okno CWND je rovné prahu $cwnd = ssthresh$, už nedochádza k exponenciálnemu rastu, ale za každé uplynutie doby odozvy CWND rastie o 1.

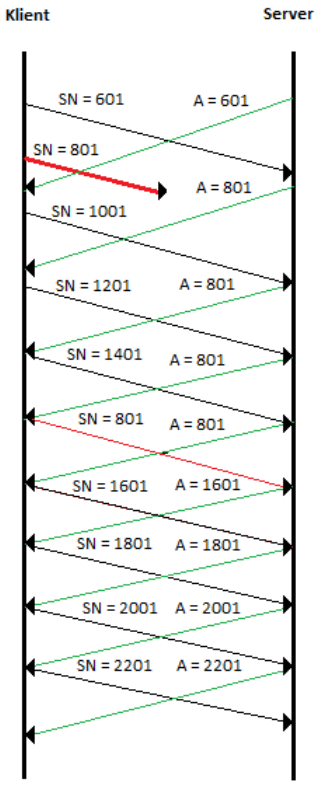


Obr. 1.17: Nastavanie okna CWND a prahovej hodnoty SSTHRESH.[1]

Obr.1.17 popisuje variáciu, je možné vidieť dosiahnutie maxima okna zahltenia 16, po vzniku zahltenia je $cwnd = 1$ a hodnota SSTHRESH sa nastaví na polovicu. Po dosiahnutí prahu oknom zahltenia je stúpanie okna CWND miernejšie[1][2][4].

Rýchly opakovaný prenos

Pri odosielaní segmentov, a vzniku zahltenia počas plnej prevádzky kapacity siete, musí byť schopná TCP entita rozoznať kedy dôjde k zahlteniu. A následovne segmenty ktoré neboli doručené opätovne odoslať a dokázať sa čo najskôr dostať na pôvodnú rýchlosť prenosu dat. TCP entita zo základu využíva dva mechanizmy a to mechanizmus Rýchleho zotavenia a Rýchleho opätovného vysielania.
Indikáciu zahltenia TCP entita identifikuje prijatím duplicitného potvrdenia posledného úspešne poslaného segmentu. Samotný jeden duplikát by však nebol dostačujúci a považuje sa za pomerne bežnú vec. V prípade jediného duplikátu môže TCP entita usúdiť že odoslaný segment sa jednoducho oneskoril a neberie to ako stav zahltenia. Podobná úvaha TCP spočíva aj v prijatí druhého duplikátu daného segmentu. Zmena nastáva až po prijatí tretieho duplikátu, TCP berie tretí duplikát ako indikáciu stavu zahltenia a začne k problému pristupovať. Okamžite odošle stratený segment a čaká na potvrdenie nie tento krát práve odoslaného segmentu (strateného), ale očakáva potvrdenie už posledného nasledujúceho segmentu. Princíp mechanizmu znázorňuje Obr.1.18. Na obrázku je vidieť stratu segmentu SN = 801, odosielateľ zatiaľ odosiela segmenty ďalej pretože nevie že došlo k strate segmentu. Príjemca nieje schopný spracovať zatiaľ prijaté segmenty, preto ich uchováva



Obr. 1.18: Technika rýchleho opakovaného prenosu.[1][6]

vo vyrovnávacej pamäti, a posiela stále odpoveď s označením A = 801, ktorý mu zatiaľ neprišiel. Príjemca až po troch duplicitných potvrdeniach zisťuje že došlo ku strate segmentu, a vysiela požadovaný segment. Príjemca po obdržaní strateného segmentu, potvrdí všetky prijaté segmenty vo vyrovnávacej pamäti posledným očakávaným segmentom čiže SN = 1601.

Rýchle zotavenie

V mechanizmus pomalého štartu dokáže pri zahltení kompenzovať strátavosť obmedzením okna a tým znížiť dátový tok. Aj keď je prístup pomalého štartu v celku učinný, dal by sa brať už sa zastaralý dokonca aj agresívny. Exponenciálny nárast počiatočného okna vedieť rýchlo k zahlteniu siete a následovné zníženie okna CWND na minimum pri strate je obmedzujúce. Tieto chyby alebo nedostatky sa snaží napraviť mechanizmu Rýchleho zotavenia. Ten následovne vynecháva počiatočný exponenciálny nábeh. Pracuje následovne[1][2][4][6]:

- **1. V prípade** prijatí troch duplicitných potvrdení sa prahová hodnota SSTHRES nastaví na $ssthres = cwnd/2$. Odošle okamžite stratený segment a nastaví

CWND na $cwnd = ssthres + 3$. Nastavenie CWND je z dôvodu doručenia ďalších troch segmentov po prijatí troch duplicitných potvrdení.

- **2. V prípade** prijatia ďalších duplicitných potvrdení pre stratený segment sa zvýši hodnota CWND o 1 a je snaha odoslať ďalší segment.
- **3. V prípade** prijatia potvrdenia, ktoré potvrdzuje už iný segment, sa nastaví hodota CWND na $cwnd = ssthres$. Potvrdenie, ktoré príjemca obdržal bolo kumulatívne potvrdenie všetkých odoslaných segmentov a segmentu strateného.

2 METÓDY PRE SPRÁVU OKNA TCP

Mechanizmy používané protokolom TCP slúžia k zabráneniu vzniku kolapsu preťaženia, existuje celá škála týchto mechanizmov v závislosti na použitej technológií a potrebe (drátové spojenie, bez drátové spojenie alebo satelitné vysielanie, vysoko rýchlostné siete...). Jednotlivé mechanizmy slúžia pre správu a nastavovanie okna TCP, pri indikácií zahltenia, alebo sa mu snažia predom predísť.

2.1 Rozdelenie mechanizmov správy okna

V počiatkoch počítačových sieti a internetu prvými mechanizmami, ktoré spravovali okno TCP, boli Thaoe a Reno. Jednotlivé mechanizmy, ktoré vznikli následovne sa dajú podľa použitého média a prostredia rozdeliť na tri skupiny:

- **Mechanizmy pre drátové siete**
- **Mechanizmy pre bez drátové siete**
- **Mechanizmy pre satelitné spojenia**

2.1.1 Mechanizmus TCP Thaoe

Prvými mechanizmami, ktoré mali riešit otázku zahltenia boli Thaoe a Reno. Zakladali sa na jednoduchých princípoch, ktoré dokázali rozpoznať indikáciu zahltenia a pristupiť k nemu. Thaoe a Reno spolu s každým ďalším mechanizmom ktorý bol vyvinutý sa snaží splniť tri podmienky[7]:

- Určiť koľko šírky pásma je k dispozícií
- Snaha udržovať rovnováhu medzi prenosom a zahltením
- Reagovať na vznik zahltenia

Úplne prvý mechanizmus pre správu okna TCP Thaoe. Slúžil ako predloha pre budúce metódy. Obsahoval v sebe mechanizmy Pomalého štartu a Rýchly opakovaný prenos. Pre vyhýbanie sa zahltenie používal Dynamické znižovanie veľkosti okna, kde vypršanie časovaču bolo indikáciou zahltenia. Následovne využíval mechanizmy pomalého štartu a Rýchleho opakovaného prenosu aby preposlal stratené segmenty, a upravil podľa špecifikácií metód prahovú hodnotu SSTHRESH a okno zahltenia CWND. Podstatnou nevýhodou metódy Tahoe bola identifikácia stratených segmentov. Pretože musel čakať celý časový interval, kým bol schopný detekovať že došlo ku stráte. Musel zároveň pre každý stratený segment vyčkávať, kým nevyprší časovač a nevyprázdni sa linka (nastavením $cwnd = 1$)[6][7].

2.1.2 Mechanizmus TCP Reno

Reno v sebe zahrňuje základné princípy Thaoe a zároveň ich rozširuje. Je doplnený o inteligenciu detekovať stratený segment skôr než Thaoe (skôr než vyprší časovač), a to tak že vyžaduje za každý odoslaný segment potvrdenie. Samozrejme počíta s tým že obdržanie jedného duplicitného potvrdenia môže byť len oneskorením. Preto indikuje zahltenie až po troch duplicitných segmentoch. A pomocou Rýchleho opakovaného prenosu sa snaží stratené segmenty preposielať bez toho aby musel čakať na časovač. Zároveň implementuje techniku Rýchleho zotavenia aby zredukoval dátový tok. Týmto spôsobom ostane v linke polovica predchádzajúceho dátového toku (podľa špecifikácie Rýchleho zotavenia), nastavením CWND a SSTHRESH. Vďaka lepšiemu prístupu a menším čakacím dobám je schopný Reno rýchlejšie dosiahnuť pôvodnej prenosovej kapacity. Reno má však jeden veľký nedostatok, ten spočíva pri veľkej strátavosti. Pretože môže detekovať iba jeden stratený segment v rámci okna. V prípade, že sa ich stratí viac, musí Reno najskôr úspešne preposlať predchodzí pôvodný segment a obdržať potvrdenie, aby potom posielať ďalšie stratené segmenty postupne jeden za druhým. Taktiež môže nastať zníženie CWND dva krát pokiaľ by boli stratené segmenty v jednom okne. Ďalšou nevýhodou je v prípade ktorom by bolo okno veľmi malé, nemuselo by sa preniesť dosť duplicitných potvrdení na indikáciu zahltenia a Reno by musel podobne ako Thaoe čakať na časovač[6][7].

NewReno

NewReno je jemnou modifikáciou pôvodného Rena, snaží sa vyladiť jeho nedostatky vyššej strátavosti a detekovať viacero stratených segmentov naraz[7][8]. Mechanizmy Pomalého štartu a Rýchly opakovaný prenos sú obdobné ako u Rena. Zásadným vylepšením je zabránenie aby sa CWND neredukovala viac než jeden krát pri viacerých strátach v rámci jedného okna. Toto bolo dosiahnuté tým, že NewReno zostáva v dobe Rýchleho zotavenia do doby kým neobdrží všetky potvrdenia segmentov ktoré boli pred tým odoslané.

Prístup TCP Reno ku zhlteniu

Pri zahltení Reno postupuje následovne:

1. Po prijatí troch duplicitných potvrdení vstupuje do Rýchleho zotavenia, nastavuje hodnoty CWND, SSTHRES a odošle stratený segment(viz. kapitola1.2.5)
2. Posiela nové segmenty ak mu to okno CWND a okno inzerované príjemcom umožňuje.
3. Čaká na príchodzie potvrdenia, potvrdenia môžu byť úplne alebo parciálne

- Ak prijaté ACK potvrdzuje všetky segmenty(úplné ACK) odoslané počas rýchleho zotavenia, opustí rýchle zotavenie, nastaví $cwnd = ssthresh$, a prejde do stavu vyhýbaniu sa zahltenia.
- Ak prijaté ACK je potvrdzuje len časť segmentov (neúplné ACK), predpokladá že následovný segment ktorý nieje zahrnutý v potvrdení sa stratil. Zníži CWND o počet segmentov ktoré boli potvrdené, inkrementuje o 1 a pošle nový segment ak to hodnota CWND povoľuje a okno. Rýchle zotavenie opustí jedine v prípade ak boli potvrdené všetky segmenty v okne. Toto čiastočné znižovanie okna v dôsledku neúplných ACK je snaha, aby po uplynutí Rýchleho zotavenia ostalo v sieti približne SSTHRES počet dat.

Resetovanie časovača pre opätovné odosielanie

Časovač pre opätovné odosielanie stratených segmentov sa nastavuje do pôvodnej hodnoty v prípade že je prijaté neúplné potvrdenie. Sú dve varianty:

- **Nedočkavá varianta NewReno** - časovač sa vynuluje iba pre prvý parciálny ACK. V tomto prípade ak je stratený veľký počet segmentov z okna, vyprší časovač odosielateľa pre opätovné odosielanie a je nútení začať Pomalým štartom.
- **Pomalá ale Stabilná varianta NewReno** - časovač sa vynuluje pre každé parciálne ACK. V prípade ak je stratený veľký počet segmentov, TCP entita nemusí začať od Pomalého štartu ale začne stratené segmenty odosielať a to jeden segment na RTT.

Ani jedna z variant však nieje optimálna, u Nedočkavej varianty môže pri RTO = RTT nastať vypršanie času opätovného odosielania aj v prípade malej strátavosti. U varianty Pomalá ale Stabilná môže pri RTO > RTT NewReno zostať v Rýchlom zotavení až moc pridlho, v prípade veľkej strátavosti[6][7][8].

2.1.3 Mechanizmus TCP Vegas

Mechanizmus TCP Vegas je rozsiahlou modifikáciou mechanizmus TCP Reno. Je založený na princípe aktívneho merania, ktoré je efektívnejšie pri identifikácií nadchádzajúceho sa zahltenia. Pomocou tohoto princípu TCP-V nepotrebuje vyčkávať na duplicitné potvrdenia, aby zahájilo opatrenia pri vzniknutom zahltení. Kde Reno muselo počkať na duplikáty, sieť sa už nachádzala v stave zahltenia, TCP-V je schopné predísť zahlteniu skôr než k tomu dôjde. Podobne ako jeho predchodca aj TCP-V implementuje v sebe predchodzie mechanizmy a však so značnou úpravou. TCP-V v sebe zahrňuje tri veľké zmeny[7]:

- **Modifikovaný pomalý štart**

- **Nový mechanizmus pre opätovné odosielanie segmentov**
- **Nový spôsob vyhýbania sa zahlteniu**

Spôsob výpočtu RTT

Výpočet RTT je u TCP Vegas neoddeliteľnou súčasťou jeho spôsobu funkčnosti. Aby mohol RTT vypočítať čo najpresnejšie, zaznamenáva systémové hodiny (čas) pre každý segment ktorý odoslal. Pri prijatí ACK, TCP-V prečíta uloženú hodnotu hodín a spraví výpočet RTT na základe času príchodu ACK a času zaznamenaného pre daný segment. Presný výpočet RTT slúži k dvom účelom. Prvým účelom je oveľa presnejšie určovanie timeoutu a druhý, slúži k dôkladnejšiemu opätovnému odosielaniu[14].

Modifikovaný pomalý štart

V prípade pomalého štartu u TCP Reno sa okno CWND zvyšovalo každým prijatím potvrdením ACK, resp. zdvojnásobilo okno CWND na jeden RTT. To viedlo k exponenciálnemu rastu kým sa neprekročí *awnd* alebo počiatočný prah, do doby kým sa nezačali strácať dáta. Pokiaľ by bola prahová hodnota okna moc malá exponenciálny rast by sa zastavil príliš skoro čo by viedlo k malej priepustnosti. V prípade veľkej hodnoty, by okno CWND rástlo po dobu dostupnej šírky pásma kým by nespôsobilo zahltenie. TCP-V preto svoje okno CWND zvyšuje exponenciálne iba na každý ďalší RTT počas pomalého štartu, pričom pomalý štart opustí len v prípade kedy CWND prešlo k prekročeniu jeho prahu. Počas dvoch po sebe idúcich RTT sa okno CWND nemení, ostáva fixné, vďaka čomu je možné zmerať rozdiel medzi *Očakávanou* hodnotou priepustnosti a *Aktuálnou* hodnotou priepustnosti[14][21].

Mechanizmus pre opätovné odosielanie

TCP Reno opätovne odosiela segmenty v prípade vypršaní časovača alebo po prijatí troch duplicitných potvrdení. TCP-V tuto myšlienku prijal a rozšíril, vďaka tomu že je schopný zaznamenávať jednotlivé časy pre každý segment individuálne pristupuje k potvrdeniam následovne[14][21]:

- V prípade **duplicitného ACK**, TCP-V overí či je rozdiel, medzi aktuálnym časom a časom zaznamenaným pri odoslaní, väčší než je hodnota časového limitu. Pokiaľ je, TCP-V opätovne odošle daný segment bez toho aby musel čakať na tri duplicitné potvrdenia.
- V prípade **neduplicitného ACK**, pokiaľ sa jedná o prvý alebo druhý prijatý ACK po opätovnom odoslaní, TCP-V znovu kontroluje či hodnota zaznamenaná pri odoslaní neprevyšuje hodnotu časového limitu. Pokiaľ je, TCP-V

znovu odošle príslušný segment. Tento spôsob poslúži v prípade ak by došlo ku strate aj počas samotného opätovného odosielania.

TCP-V je schopný rýchlejšie odhaliť stratu segmentu, miesto toho aby vyčkával až na tri duplikáty premeriava každé prijate potvrdenie. Pokiaľ by rozpoznávanie straty podľa času zlyhalo, TCP-V použije identifikáciu podľa troch duplicitných potvrdení, a okno CWND zníži iba o 1/4 v prípade ktorom by bol čas poslednej redukcie CWND väčší než čas aktuálneho RTT.

Prístup TCP Vegas ku zahlteniu

Prístup mechanizmu TCP-V k zahlteniu závisí na na zmenách určenej a meranej priepustnosti v určitom časovom úseku. Pracuje následovným spôsobom:

1. Určí sa *BaseRTT* ako minimálna nameraná hodnota RTT, ta sa zvolí na základe šírenia oneskorenia(ideo RTT segmentu ktorý sa nenachádzal v preplnenom spojení). Následovne vypočíta *Očakávanú* priepustnosť:

$$Ocakavana = \frac{cwnd}{BaseRTT}. \tag{2.1}$$

   kde, *cwnd* je momentálna hodnota okna zahltenia.

2. Vypočíta sa *Aktuálna* priepustnosť, ktorá sa počíta pre každý RTT:

$$Aktualna = \frac{cwnd}{RTT}. \tag{2.2}$$

   kde, RTT je najnovšia odmeraná RTT.

3. Začne sa porovnávať*Aktuálna* a *Očakávana* označená tiež ako *diff*:

$$diff = Aktualna - Ocakavana$$

$$= \left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) BaseRTT$$

$$= cwnd\left(1 - \frac{BaseRTT}{RTT}\right). \tag{2.3}$$

   diff ako výsledná hodnota je nezáporná, a slúži k nastaveniu okna CWND.

4. TCP-V ma implementované dve konštanty α a β[11], podľa ktorých riadi CWND:

$$cwnd = \begin{cases} cwnd + 1 & diff < \alpha, \quad (2.4a) \\ cwnd & \alpha \leq diff \leq \beta\ , \quad (2.4b) \\ cwdn - 1 & diff > \beta. \quad (2.4c) \end{cases}$$

   Pokiaľ $diff < \alpha$, TCP-V zväčší CWND lineárne počas ďalšieho RTT.
   Pokiaľ $diff > \beta$, TCP-V zníži CWND lineárne počas ďalšieho RTT.
   Pokiaľ $\alpha \leq diff \leq \beta$, TCP-V ponecháva CWND nezmenené.

[11] α a β boli určené experimentálne, TCP-V si ich počas záhajenia spojenia nastaví sám.

Z výsledného postupu, ktorým TCP-V pristupuje k prenosu dátového toku, je zrejme že riadenie samotného toku a zahltenia prebieha medzi parametrami α a β. Následovne je stanovené v prípade, ktorom bude *Aktuálna* priepustnosť oveľa menšia než *Očakávaná* priepustnosť, je veľká pravdepodobnosť že linka je preplnená. A mala by byť prenosová rýchlosť dátového toku znížená. Ak bude situácia, ktorej bude *Aktuálna* priepustnosť blízko *Očakávanej* priepustnosti, spojenie nemusí využívať plný potenciál prenosovej kapacity a dátový tok by sa mal zvýšiť.

2.2 Mechanizmy pre pre drátové siete

Drátové siete tvoria najväčšiu časť všetkých sieti. Od metalických po optické vlákna si prešli rozsiahlym vývojom prenosovej kapacity, preto bola vyvinutá rada mechanizmov aby splnila prenosové rozmanitosti rýchlosti drátových sieti. Medzi mechanizmy pre drátové siete patria:

- **BIC TCP**
- **CUBIC TCP**
- **HighSpeed TCP**
- **Hamilton TCP**
- **Scalable TCP**
- **TCP Vegas**
- **TCP Westwood+**
- **TCP Low-Priority**
- **TCP Illinois**
- **Compound TCP**

BIC TCP

Mechanizmus BIC bol vytvorený pre veľké tučné siete (long fat networks), ktoré sú charakteristické veľmi vysokou latenciou. Jeho algoritmus pracuje na princípe, kedy sa snaží nájsť maximum, na ktorom by udržal okno po veľmi dlhú dobu. Používa binárny vyhľadávací algoritmus spolu s aditívnym zvýšením. TCP BIC svoje využitie môže poskytnúť v sieťach na veľmi veľkú vzdialenosť, vysokým RTT a problémom dosiahnuť plnú šírku pásma. Jeho algoritmus sa snaží nájsť taký kompromis, aby dokázal posielať veľmi rýchlo svoje data po dlhú dobu so zaručením lineárnej RTT spravodlivosti a škálovateľnosti[9].

CUBIC TCP

Mechanizmus CUBIC je derivátom BIC TCP, jedná sa o optimalizovaný mechanizmus pre veľké tučné siete s veľmi vysokou latenciou. Podstatou zmenu priniesol

zmenšenou agresivitou ovládania okna, kde sa na okno pozerá ako na kubickú funkciu od doby vzniknutej posledným zahltením. Podstatná zmena TCP CUBIC spočíva v nepotrebe prijatí ACK aby mohol navýšiť svoje okno. Jeho veľkosť okna je závislá iba na poslednom vzniknutom zahltení. Pri vysokom RTT by chodili potvrdenia moc pomaly, a tým by sa okno zvyšovalo neefektívne, preto TCP CUBIC zvýši svoje okno v závislosti na poslednom zahltení než čakať na potvrdenia. Týmto spôsobom je schopný udržať dostatočné veľkú rýchlosť aj v sieťach s vysokou latenciou[10].

HighSpeed TCP

HighSpeed TCP je novodobým mechanizmom, navrhnutým pre siete s veľmi veľkou prenosovou rýchlosťou kde je potrebné aby entita TCP bola schopná veľký objem dát čo najrýchlejšie a v prípade zahltení dokázať sa okamžite zotaviť. Vychádza z viacerých mechanizmov určených pre podobné podmienky. K svojej činnosti používa funkciu, ktorá sa stará o správu okna. Zároveň implementuje tri parametre podľa ktorých riadi SSTHRESH, a tým sa dokáže prispôsobovať rôznym stavom v sieti. HS-TCP sa snaží udržiavať dlhodobé vysoko rýchlostné spojenie po dlhú dobu a redukovať pri tom straty na úkor "spravodlivosti"[11].

Hamilton TCP

Hamilton TCP je zástancom mechanizmov používaných pre veľké tučné siete s veľmi vysokou latenciou. Jeho algoritmus a spôsob správy je založený na predchádzajúcich stratách. Dá sa pokladať za veľmi agresívny mechanizmus, ktorý stráži aká veľká strata bola pri poslednom zahltení a podľa nej následovne zväčšuje okno. Zároveň sa snaží riešiť otázku "RTT nespravedlnosti", ktorá sa vyskytuje u väčšiny mechanizmov[12].

Scalable TCP

Scalable TCP je hrubou modifikáciou TCP NewReno, bol vyvinutý v snahe dosiahnuť vyššiu priepustnosť a škálovateľnosť. Podstatnou zmenou je spôsob znižovania CWND okna, kde pôvodný mechanizmus znižoval pri zahltený o 1/2, S-TCP znížuje iba o 1/8 do doby kým sa zahltenie nevyrieši. Podstatnou zmenou si prešiel aj samotné zotavenie, ktoré je upravený tak, že po zahltení sa okno zvyšuje o jeden segment každých sto prijatých potvrdení[13].

TCP Vegas

TCP Vegas je mechanizmus založený na oneskorení jednotlivých paketov. Pomocou meraní RTT určuje, predpovedá kedy sa môže vyskytnúť zahltenie siete a následovne

adaptívne nastavuje svoje okno. Mechanizmus prepočítava neustále hodnoty RTT z čoho si určuje parameter zahltenia. Tento parameter slúži ako kontrolný bod, ak sa okno CWND s parametrom nezhoduje, prenastaví okno tak aby vyhovovalo danému parametru. TCP-V ma snahu úplne minimalizovať strátavosť paketov a však za cenu neuplného využitia prenosovej kapacity[14].

TCP Low-Priority

TCP Low-Priority bol navrhnutý ako mechanizmus správy okna pre služby s nízkou prioritou. Snaží sa využívať prebytočnú šírku pásma, ktorú nepoužili ostatné dátové toky. Mechanizmus je navrhnutý aby pracoval len zo strany odosielateľa, ide o jednosmerný spôsob správy okna, kde na druhej strane je bratský mechanizmus, s ktorým TCP-LP úzko spolupracuje. TCP-LP sa používa pre služby ktoré pracujú s pomalou prenosovou rýchlosťou a potrebujú dostatočne veľkú odozvu pri komunikácií[15].

TCP Illinois

TCP Illinois je mechanizmus pre veľké tučné siete. Operuje na strane odosielateľa, jeho snahou je dosiahnuť vyššiu priemernú hodnotu priepustnosti. Zároveň sa snaží dodržať "spravodlivosť" a efektívne deliť zdroje siete. TCP-I je založený na strátavosti a oneskorení, podľa veľkosti strátavosti segmentov svoje okno zväčší alebo zmenší, podľa oneskorenia určí o koľko toto okno zväčšiť alebo znížiť[16].

Compound TCP

Compound TCP je mechanizmom slúžiaci pre prenos v sietiach s veľkou šírkou pásma a veľkým oneskorením. Primárnou úlohou je dosiahnuť rýchly prenos a udržať si pri tom "spravodlivosť" v sieti. K svojej činnosti používa meranie oneskorení front paketov, vychádza z myšlienky kde malé oneskorenie znamená rapídne zvyšovanie okna. C-TCP je podobný TCP Vegas, a však narozdiel od neho nepoužíva parameter na ovládanie okna, ale udržuje dva okna CWND. Z týchto dvoch okien následovne určí stav siete a prispôsobuje sa mu. Podstatným rozdielom oproti TCP Vegas je schopnosť predísť pretrvávanému zahlteniu, C-TCP dokáže svoje okno redukovať s okamžitou odozvou na oneskorenie[17]

2.3 Mechanizmy pre pre bez drátové siete

Bez drátové siete slúžia ako jednoduchý spoj medzi uživateľom (jeho prístupovou stanicou) a samostatnou sieťou. Majú pomerné veľké zastúpenie, a ich využívanie sa zvyšuje. Medzi mechanizmy pre bez drátové siete patria:

- **TCP Veno**
- **TCP Westwood+**

TCP Veno

Patrí medzi novodobo používané mechanizmy pre správu okna TCP skrz bez drátové siete. TCP Veno vychádza z TCP NewReno a TCP Vegas, kde je snaha z výšiť priepustnosť dát a znížiť stratavosť paketov skrz bez drátové technológie. Kombinuje techniky oboch mechanizmov, je schopný určovať stav pripojenia, dedukovať o aký typ straty paketov nastal - strata zahltením alebo náhodná strata. Podľa meraní očakávaného a aktuálneho RTT určuje parametre ktoré predvídajú nastávajúce zahltenie[18].

TCP Westwood+

Mechanizmus Westwood+ je novodobou modifikáciou pôvodného TCP Westwood a ten z TCP Reno. Je používaný pri drátových aj bez drátových sieťach. Jeho zaujímavosťou je že sa jedna len o mechanizmus z jednej strany. Slúži len na strane odosielateľa (na strane príjemcu býva často TCP Reno), je založený na odhade šírky pásma medzi koncovými bodmi, podľa ktorého sa snaží správne alebo vhodne nastaviť CWND a SSHTRESH po tom čo nastane aspoň jedna príhoda zahltenia. Šírku pásma určuje pomocou low-pass filtrovania rýchlosti ktorou sa vracajú potvrdenia. Narozdiel od Rena ktoré nastaví CWND na polovicu, Westwood nastavuje svoje okno a prah adaptívne s ohľadom na šírku pásma ktorá bola používana v momente zahltenia. Westwood+ priniesol vylepšený algoritmus pre lepší odhad dostupnej šírky pásma[19].

2.4 Mechanizmy pre satelitné spojenia

Slúžia ku komunikácií medzi zariadeniami kde bez drátové spojenie nemôže byť dosiahnuté, sú schopné komunikovať na veľmi veľké vzdialenosti. Majú najmenšie zastúpenie. Medzi mechanizmy pre satelitné spojenia patria:

- **TCP Hybla**

TCP Hybla

Mechanizmus TCP Hybla bol vyvinutý za účelom vylepšiť efektívnosť prenosu skrz satelitné a rádiové spoje, kde dôsledku chýb dochádzalo k stráte paketov a TCP entita to brala ako zahltenie, následovným dôsledkom veľmi dlhých RTT a neskorých príchodov ACK, bolo okno zahltenia veľmi limitované. TCP Hybla sa toho snaží dosiahnuť odstranením prenosového burstu a paketovými rozostupovacimi technikami, čím zabraní aby dochádzalo k predčasnému zahlteniu vyrovnávacích pamätí smerovačov[20].

3 POROVNÁVANIE TCP RENO A TCP VEGAS

3.0.1 Simulačné prostredie NS2

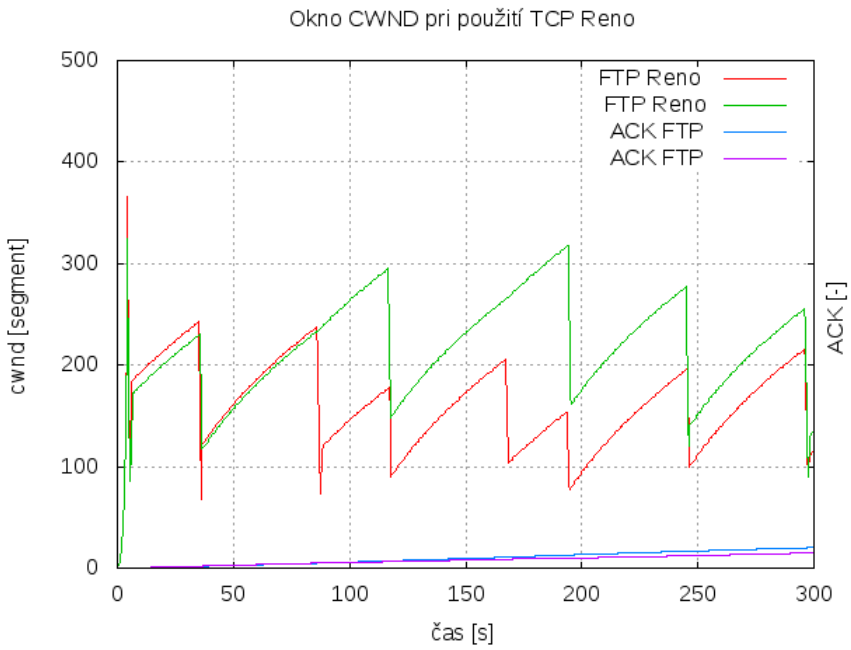
NS2 alebo tiež Network Simulator, je prostredie pre simulácii rôznych stavov siete. Poskytuje možnosť simulácií TCP, smerovania a multikastových protokolov skrz drátovú alebo bez drátovú sieť. NS2 bol navrhnutý pre OS Linux, FreeBSD, Solaris, Mac OS X a Windows (Cygwin). Pre lepšiu funkčnosť a presnejšiu simuláciu jednotlivých metód bolo do NS2 vložené rozšírenie Linux TCP[12], ktoré slúži ako rozširujúci dodatok pre NS2.

3.1 Simulácia dvoch dátových tokov pomocou TCP Reno na vysielacej strane

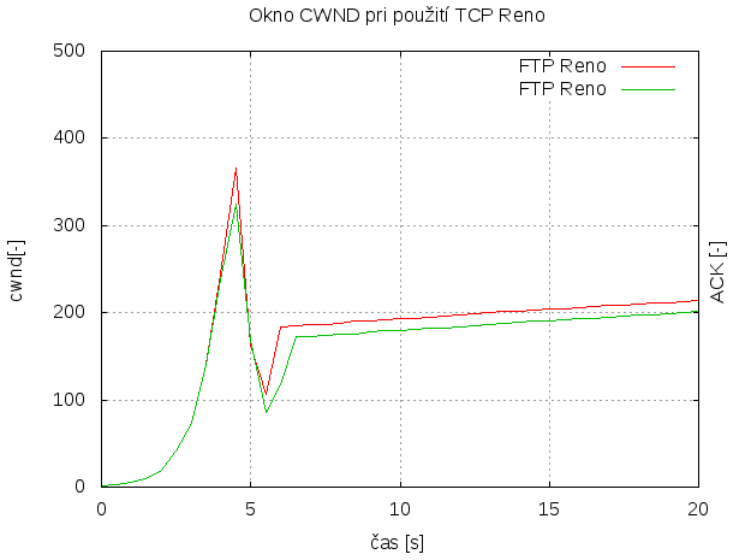
Pomocou metody TCP NewReno, boli simulované dva dátové toky na šírke pásma o veľkosti 10 Mbit/s. Nastavenia metódy boli zvolené tak aby odpovedali zvolenému scenáriu. Podrobný prehľad nastavení pre danú metódu je možné vidieť v prílohe A.1. V grafe 3.1 je možné vidieť ovládanie okna CWND pomocou metódy TCP Reno pri prenose dvoch dátových tokov so šírkou 10 Mbit/s. Správanie jednotlivých priebehov vychádza z teoretickej časti práce. Pri počiatočnom prenose, ktorý je možný vidieť v čase 0 s až 6 s, nastáva fáza pomalého štartu, pri každom prijatom potvrdení je pre dátové toky okno CWND zvýšené o jedna, do doby kým nedôjde k indikácií zahltenia, po pomalom štarte, nastavení prahových hodnôt a následovným ich dosiahnutí sa exponenciálny rast mení na lineárny. Rast okna CWND je miernejší tento jav je možný vidieť v čase 7 s, v grafe 3.2 a nastáva fáza vyhýbania sa zahlteniu. V grafe 3.3, je možné vidieť jednotlive príchodzie potvrdenia segmentov. V čase 4 s došlo k zahlteniu, odosielateľ je nútený znovu vyslať stratený segment a jeho potvrdenie dostal spolu za ostatné nepotvrdené segmenty až v čase 5 s, z tohoto dôvodu graf 3.3 ukazuje výkyv pri prijatí tohoto potvrdenia.

Z grafu 3.1 je možné taktiež vidieť správanie sa okna počas prenosu. Je vidieť, že jednotlivé okná su pre oba dátové toky rozdielne. TCP Reno nerieši "spravodlivosť" a tak nechá aby jednotlivé dátové toky súperili o prenosovú šírku. Rozdielnosť okien sa dá vysvetliť na meniace sa oneskorenie, v dôsledku súperenia tokov o šírku pásma vzniká premenlivé RTT, to spôsobuje rozdelený rast okna CWND. Prvý FTP Reno (červený priebeh) má menšie hodnoty RTT, tým je mu umožnené rýchlejšie

[12]Jedná sa o patch, ktorý je potrebný rozbaliť v adresári programu NS2.

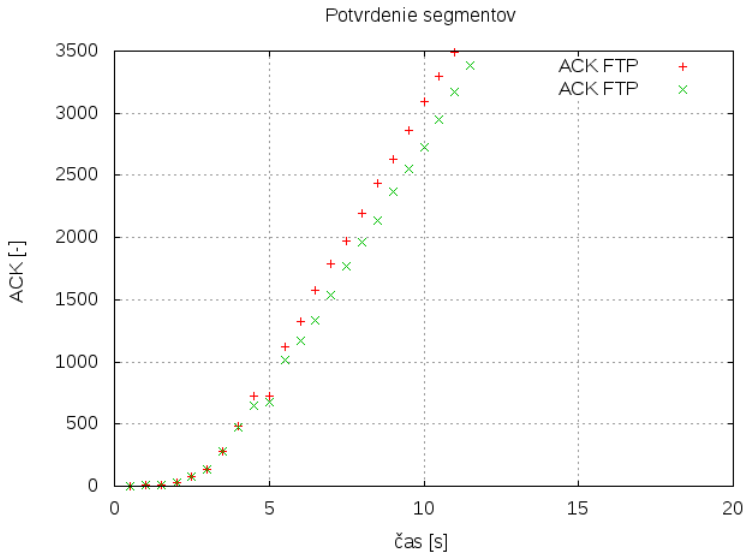


Obr. 3.1: Graf okna zahltenia CWND a potvrdení ACK za použitia TCP Reno.



Obr. 3.2: Graf pomalého štartu metody TCP Reno.

zvyšovať svoje okno a preniesť viac dát za kratšiu dobu, a však je taktiež vidieť, čím rýchlejšie sa okno CWND zvyšuje tým rýchlejšie sa dostava do stavu zahltenia a musí svoje okno CWND znížiť a preposielať stratené segmenty. Druhý FTP Reno (zelený priebeh), má vyššie hodnoty RTT, z toho dôvodu čaká dlhšie kým mu prídu

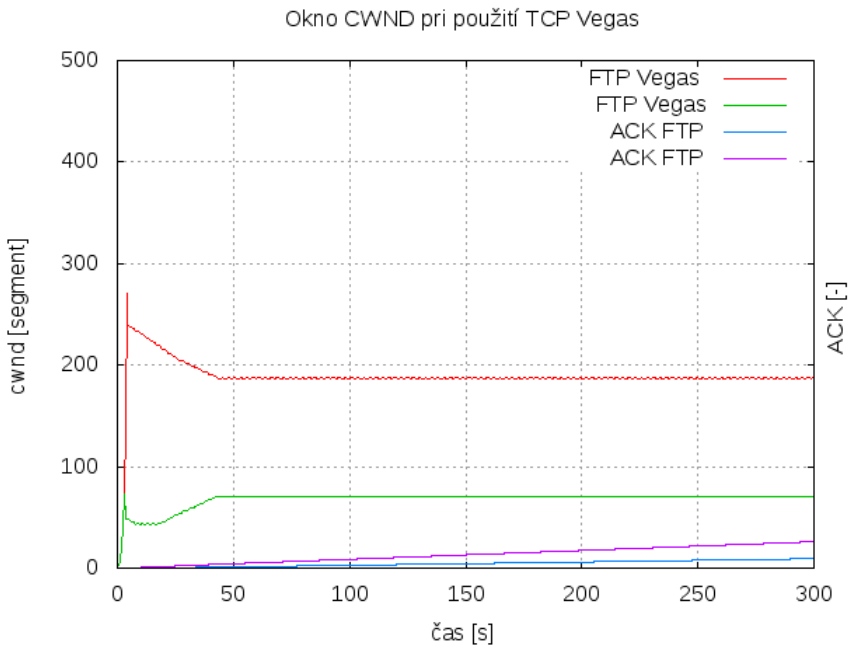


Obr. 3.3: Graf ACK pomalého štartu metody TCP Reno.

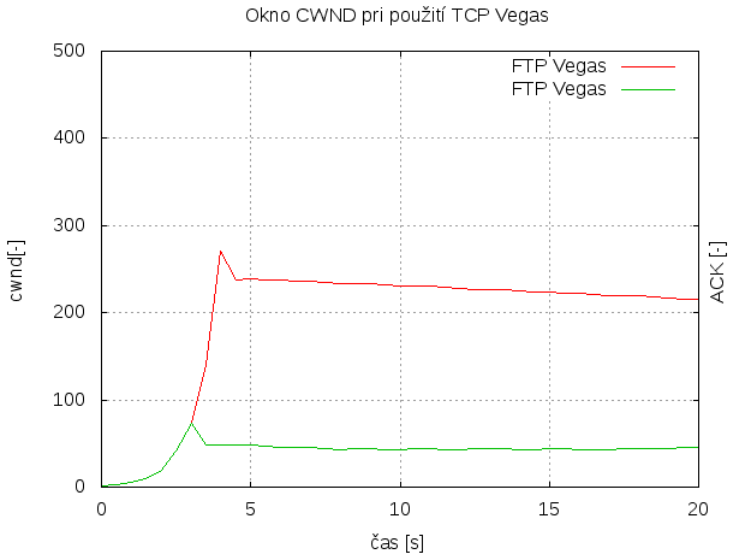
potvrdenia a svoje okno CWND zvyšuje pomalšie. Síce sa pomalejšie dostane ku kritickej hranici a zhlteniu zato prenesie menej dát za rovnaký čas ako prvý FTP Reno.

3.2 Simulácia dvoch dátových tokov pomocou TCP Vegas na vysielacej strane

Pomocou metody TCP Vegas, boli simulované dva dátové toky na šírke pásma o veľkosti 10 Mbit/s. Nastavenia metódy boli zvolené tak aby odpovedali zvolenému scenáriu. Podrobný prehľad nastavení pre danú metódu je možné vidieť v prílohe A.2. V grafe 3.4 je možné vidieť ovládanie okna CWND pomocou metódy TCP Vegas pri prenose dvoch dátových tokov so šírkou 10 Mbit/s. Pri počiatočnom prenose ktorý je možný vidieť v čase 0 s až 4 s, nastáva fáza pomalého štartu, pri každom následujúcom RTT sa hodnota okna CWND zvyšuje exponenciálne do doby kým sa neprekročí prah. V grafe 3.5 je možné vidieť pomalý štart, v čase 3 s a 4 s narazil okno CWND na prah, čím sa končí exponenciálny rast pomalého štartu a TCP Vegas sa dostáva do fázy vyhýbania sa zahlteniu. Graf 3.6 ukazuje príchodzie potvrdenia odoslaných segmentov, je možné vidieť, že narozdiel od TCP Reno, nedošlo k stráte segmentu počas pomalého štartu. Je to spôsobené schopnosťou metody predikovať zahltenie.

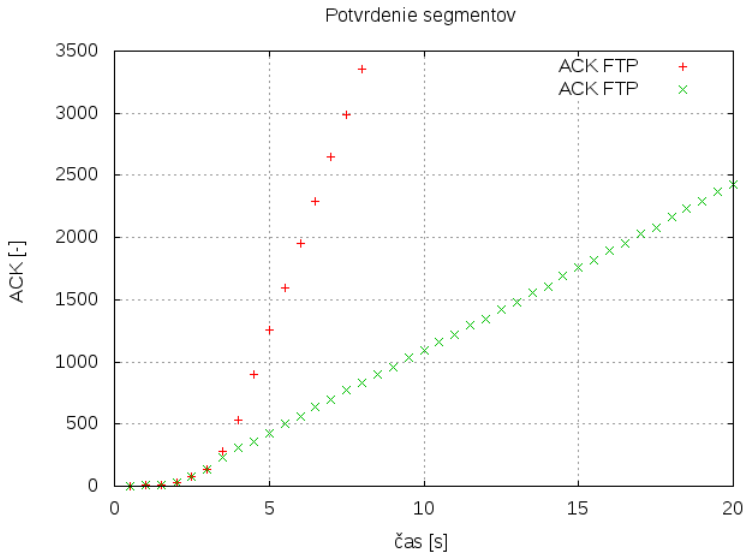


Obr. 3.4: Graf okna zahltenia CWND a potvrdení ACK za použitia TCP Vegas.



Obr. 3.5: Graf pomalého štartu metody TCP Vegas.

Graf 3.4 zároveň ukazuje správanie sa okna CWND počas prenosu. Dvojica okien sa správa rozdielne, miesto chaotického správania sú rozdelené a ich hodnota okna CWND sa dá považovať za konštantnú. Tento spôsob úpravy okien vychádza z algoritmu metódy, kedy sa nepoužíva pre indikáciu zahltenia duplicitné potvrdenia,



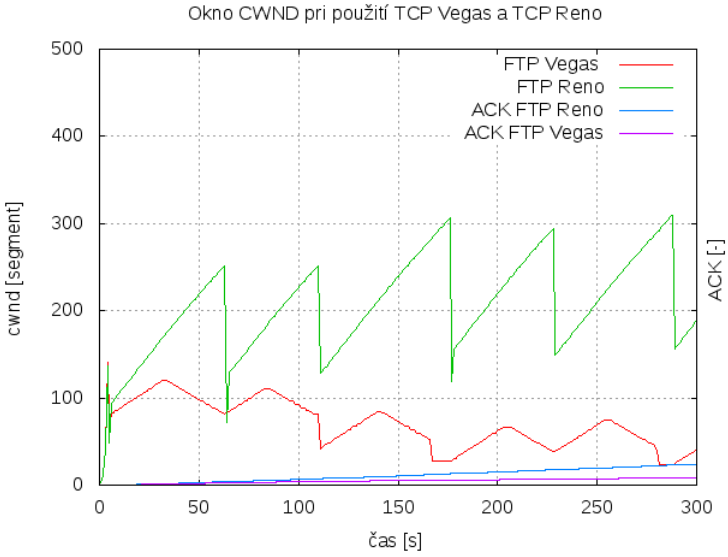
Obr. 3.6: Graf ACK pomalého štartu metody TCP Vegas.

ale používa sa predikcia zahltenia pomocou vypočítanej hodnoty *diff* vychádzajúcej z nameranej BaseRTT a RTT. Pretože je RTT rozdielny pre oba dátové toky výsledná hodnota *diff* bude rozdielna a tak okno CWND rôzne veľké. V čase krátko po pomalom štarte prvý FTP Vegas (červený priebeh) je nasledovaný znížením hodnoty okna CWND, vzniknutý v dôsledku $diff > \beta$, kedy jeho hodnota *diff* bola väčšia než parameter β. Naopak druhý FTP Vegas (zelený priebeh) svoju hodnotu okna CWND navýšil dôsledku $diff < \alpha$, kedy jeho hodnota *diff* bola menšia než parameter α. Obe zmeny okna majú lineárny charakter. TCP Vegas ďalej rieši správu okna pri viacerých dátových tokov tak aby nedochádzalo ku chaotickým zmenám ako u TCP Reno. Okna sú rozdielne veľké a po dobu prenosu sa nemenia. Aj keď sa hodnota RTT mení pre oba dátové toky, nieje dostatočné kritická aby dochádzalo ku zmenám veľkosti okna a od času 47 s je splnená podmienka $\alpha \leq diff \leq \beta$ 2.4b pre oba dátove toky, hodnota okna CWND nemení a nedochádza ku stratám.

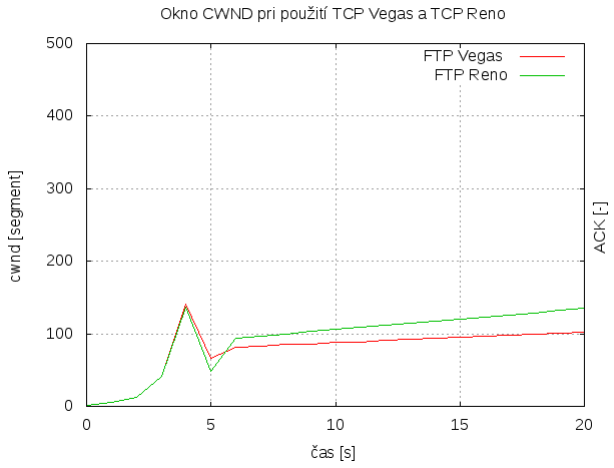
3.3 Simulácia dvoch dátových tokov pomocou TCP Vegas a TCP Reno na vysielacej strane

Simulácia zahrňuje použitie oboch metód pri vysielaní a zdielaní rovnakých poskytnutých zdrojov o šírke pásma 10 Mbit/s. Pre jeden FTP tok bola použitá metóda TCP Reno a pre druhý FTP tok metóda TCP Vegas. Podrobný prehľad nastavení pre danú metódu je možné vidieť v prílohe A.4. Pri počiatočnom prenose, ktorý je

možný vidieť v čase 0 s až 4 s, nastáva fáza pomalého štartu graf 3.8.



Obr. 3.7: Graf okna zahltenia CWND a potvrdení ACK za použitia TCP Vegas a TCP Reno na vysielacej strane.



Obr. 3.8: Graf pomalého štartu metody TCP Vegas a TCP Reno na vysielacej strane.

Po prekročení hranice prahu sa obe metódy dostávajú do fázy vyhýbania sa zahlteniu a začínajú lineárne navyšovať svoje CWND okná. V grafe 3.7 je možné vidieť ovládanie okna CWND pomocou metódy TCP Vegas a TCP Reno pri prenose dvoch dátových tokov so šírkou 10 Mbit/s. V predchádzajúcich simulaciach boli metódy TCP Vegas a TCP Reno použite oddelene. V tejto simulácií boli použité naraz na dva rozdielne FTP dátové toky. V grafe 3.7 je možné vidieť priebeh ich ovládania okna CWND. Obe metódy ovládajú svoje okná rozdielne v závyslosti na

ich algoritme. Graf ukazuje priebeh spojenia, FTP Reno (metóda TCP Reno) prebieha z cela bez vplyvu FTP Vegas (TCP Vegas), pretože jeho podstata leží v prijatí ACK segmentov bez akejkoľvek predikcie zahltenia, bude zvyšovať svoje okno kým nedôjde k samotnému zahlteniu. FTP Vegas je z dôvodu vplyvu TCP Reno utlačovaný. Pretože samotné TCP Reno ide až po hranicu, kým nenastane zahltenie spôsobuje väčšie RTT časy, na ktorých je TCP Vegas závislý. Jeho spôsob predikcie a zväčšovania okna závisí práve na hodnotách nameraných RTT. Vzniknuté zahltenie spôsobené TCP Reno zvýši RTT a aby sa TCP Vegas vyhol zahlteniu znižuje svoje okno CWND na minimálnu hodnotu.

3.4 Výsledné zhodnotenie metód TCP Reno a TCP Vegas

Simulácie ukázali, že TCP Vegas je vďaka svojej vlastnosti predikcie viac spoľahlivejší a snaží sa takmer o nulovú strátavosť. Taktiež sa snaží udržať konštantnú úroveň prenosu po celu jeho dobu trvania. A však narozdiel od TCP Reno, ktoré sa snaží isť za hranicu maximálnej prenosovej šírky, TCP Vegas nevyužíva všetku prenosovú kapacitu, ktorá mu je poskytnutá a to v dôsledku agresívnej úpravy CWND okna striktným dodržiavaním presne stanovených parametrov. V prípad použitia oboch metód naraz, sa ako vhodnejším ukazuje TCP Reno. TCP Vegas, ktorý je riadený predikciou zahltenia a parametrami α a β pre úpravu okna CWND má minimálnu priepustnosť v snahe nestratiť segmenty.

4 PARAMETRE ALFA A BETA PRE SPRÁVU OKNA CWND MECHANIZMU TCP VEGAS

Táto časť sa zameriava na samotný mechanizmus TCP Vegas, na jeho prístup k dátovému toku, simuláciou a možnosťami spŕavy okna CWND za rôznych hodnôt parametrov α a β.

4.1 Parametre alfa a beta

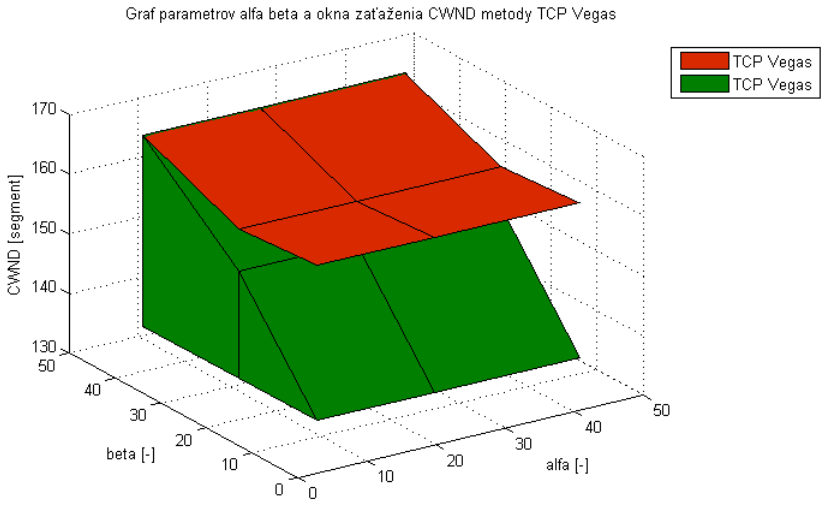
Ako základné hodnoty parametrov pre metódu TCP Vegas sú hodnoty kde $\alpha = 4$ a beta $\beta = 8$[24]. Všetky dostupné mechanizmy a ich parametre sa dajú nastaviť v konfiguračnom súbore Linuxu sysctl.conf. V prípade len jedného dátového toku je mechanizmus schopný bezproblémovo posielať svoje data bez straty a nutnosti opätovného preposielania. Problém však nastáva pri prenose väčšieho počtu dátových tokov súčasne. Ako je možno vidieť v grafe 3.4, už pri prenose dvoch tokov len za použitia metódy TCP Vegas, nieje rozdelenie poskytnutých zdrojov spravodlivé. O to k väčšiemu problému nastáva v prípade kombinácií TCP Vegas s iným mechanizmom. Graf 3.7 zobrazuje presne tento stav kedy je TCP Vegas na najnižšej možnej úrovni. V týchto prípadoch sú už základné hodnoty α a β nedostatočné. Táto kapitola sa preto venuje možnosti zmeny parametrov α a β, tak aby ich výstupe hodnoty pri prenose dátových tokov nevykazovali predošlé nedostatky.

4.1.1 Matica parametrov alfa a beta použitím metódy TCP Vegas

Ako už bolo spomenuté, pri použití metódy TCP Vegas na dva dátové toky so základnými parametrami $\alpha = 4$ a $\beta = 8$, je rozdelenie prenosovej kapacity nerovnomerne a tak jeden z dátových tokov je zvýhodnený voči druhému. Toto rozdelenie so základnými parametrami zobrazuje graf 3.4. Aby došlo k spravodlivému rozdeleniu medzi oba prenosy, boli parametre α a β vybalancované a postupne zvyšované. Výsledné parametre α a β a ich dosiahnuté okno CWND je možné vidieť v modelu 4.1. Pomer parametrov bol nastavený podľa A.3 a pre simulácie bol následovný:

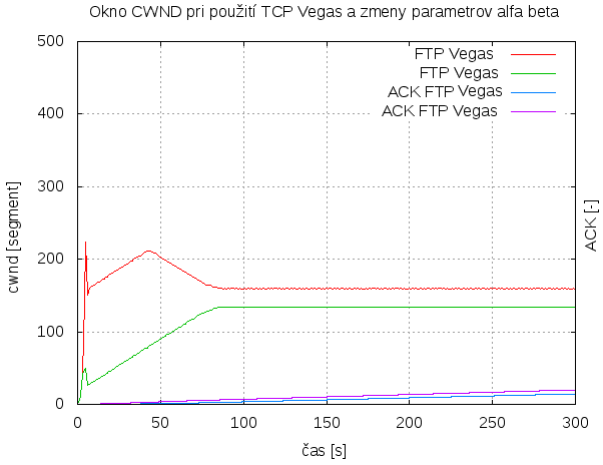
- $\alpha = 8$ a $\beta = 8$
- $\alpha = 25$ a $\beta = 25$
- $\alpha = 46$ a $\beta = 46$

Kde posledná kombinácia bola hraničnou kombináciou a zároveň rozdelenie prenosovej kapacity bolo spravodlivo delené medzi oba FTP prenosy.

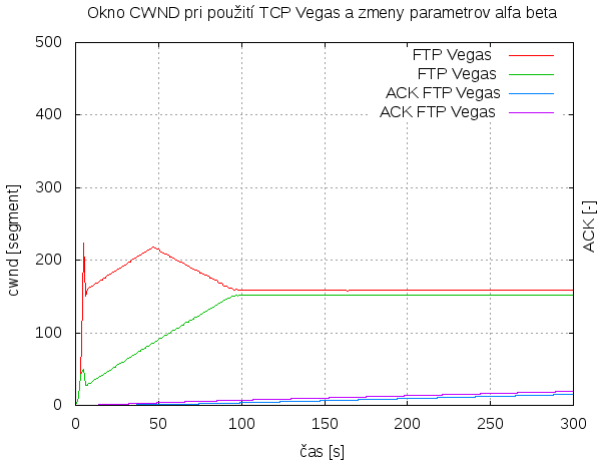


Obr. 4.1: Model parametrov α, β a okna CWND metody TCP Vegas.

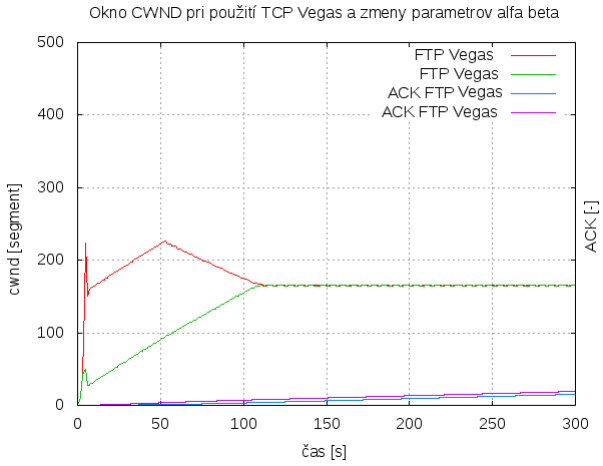
Simuláciu prenosu pre parametre $\alpha = 8$ a $\beta = 8$ zobrazuje graf 4.2. Je možné vidieť pomerne veľké zvýšenie okna CWND pre druhý FTP prenos (zelený priebeh). Kde hodnota okna viac násobne vzrástla oproti oknu CWND so základnými parametrami graf 3.4. Následovne boli parametre zvýšené na hodnotu $\alpha = 25$ a $\beta = 25$ zobrazené na grafe 4.3. Podobne ako v minulom prípade, opäť sa okno pre potlačený FTP prenos (zelený) zvýšilo, graf zobrazuje skoro presné delenie prenosovej šírky na polovicu medzi oba dátové toky. Pomer parametrov vzhľadom na okno CWND taktiež zobrazuje model 4.1. Kde je vidieť vzrast okna CWND pre zelený priebeh a pokles okna CWND pre červený priebeh. Aby sa prenosová kapacita správne delila medzi oba dátové toky FTP Vegas bola hodnota parametrov nastavená na $\alpha = 46$ a $\beta = 46$. Výsledné delenie zobrazuje graf 4.4. Je možné vidieť takmer dokonalé rozdelenie prenosovej kapacity medzi oba dátové toky. Zároveň nedochádza k žiadnej strate a oba dátové toky si držia konštantnú úroveň po celu dobu prenosu. Tento stav zobrazuje aj model 4.1, kde došlo k rastu oboch priebehoch. Parametre $\alpha = 46$ a $\beta = 46$, sa dajú taktiež označiť za hraničné. Pri zvýšení parametrov nad hodnotu 46, sa začne TCP Vegas správať chaoticky a nedrží si konštantnú úroveň, môžeme ho pozorovať na grafe 4.5. Ďalšou zaujímavosťou je, kde v dôsledku nastavenia parametrov $\alpha = 46$ a $\beta = 46$, došlo k celkovému zvýšeniu okna CWND. Model 4.1 zobrazuje zvýšenie oboch okien, lepšie zobrazenie celkového zvýšenia CWND je vidieť na grafoch 4.6(a) a 4.6(b). Kde v 150 s došlo k úplnému zastaveniu druhého dátového toku a tým sa mohol prvý dátový tok dostať na svoje maximum. V prípade týchto dvoch grafoch je možné sledovať že pri nastavených parametroch $\alpha = 46$ a $\beta = 46$, je maximum okna CWND vyššie než v jeho základných hodnotách.



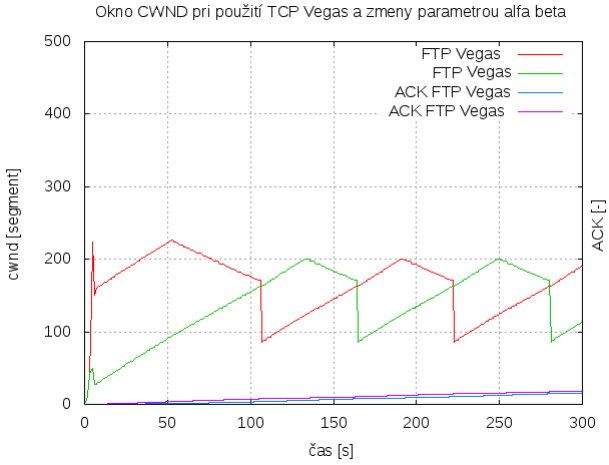
Obr. 4.2: Graf okna zahltenia CWND pri použití parametrov $\alpha = 8$ a $\beta = 8$ TCP Vegas.



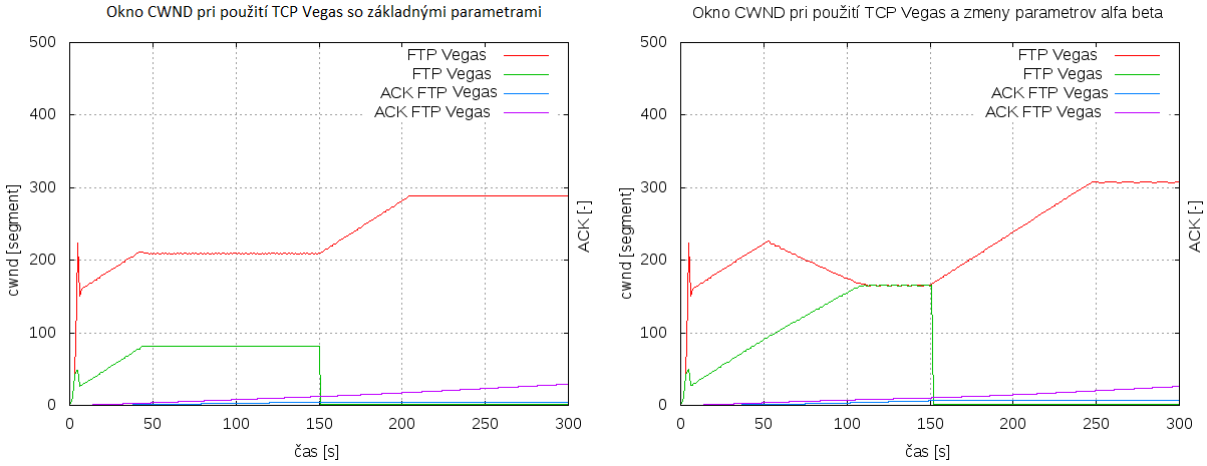
Obr. 4.3: Graf okna zahltenia CWND pri použití parametrov $\alpha = 25$ a $\beta = 25$ TCP Vegas.



Obr. 4.4: Graf okna zahltenia CWND pri použití parametrov $\alpha = 46$ a $\beta = 46$ TCP Vegas.



Obr. 4.5: Graf okna zahltenia CWND pri použití parametrov $\alpha = 47$ a $\beta = 47$ TCP Vegas.



(a) (b)

Obr. 4.6: (a) Graf okna CWND TCP Vegas s pôvodnými parametrami, (b) graf okna CWND TCP Vegas s $\alpha = 46$ a $\beta = 46$

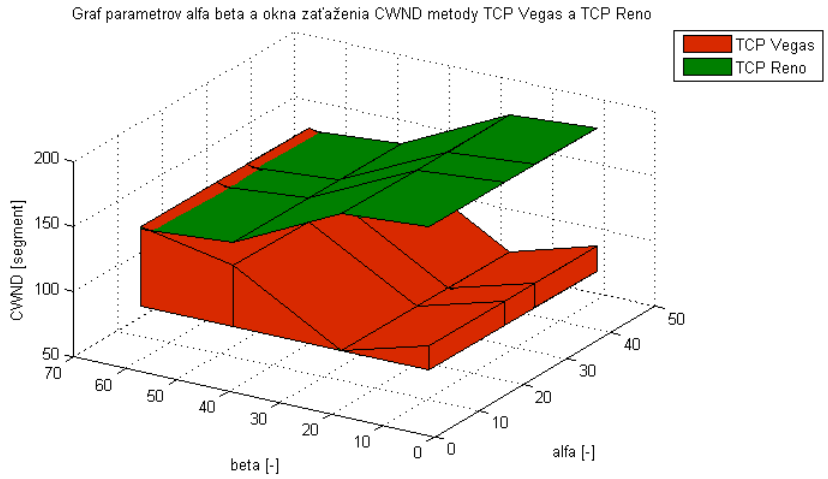
4.1.2 Matica parametrov alfa a beta použitím metody TCP Vegas a TCP Reno

Pri použití metódy TCP Vegas a TCP Reno na dva dátové toky so základnými parametrami $\alpha = 4$ a $\beta = 8$, je rozdelenie prenosovej kapacity nerovnomerne a TCP Vegas je voči TCP Reno znevyhodnený, jeho okno CWND je na najnižšej možnej úrovni a správa sa chaoticky. Toto rozdelenie so základnými parametrami zobrazuje graf 3.7. Aby došlo k lepšiemu rozdeleniu medzi oba prenosy, prípadnému zdvihnutia okna CWND, boli parametre α a β vybalancované a postupne zvyšované. Výsledné parametre α a β a ich dosiahnuté okno CWND je možné vidieť v modelu 4.7.

Pomer parametrov zvolených pre simulácie bol následovný:Pomer parametrov bol nastavený podľa A.4 a pre simulácie bol následovný:

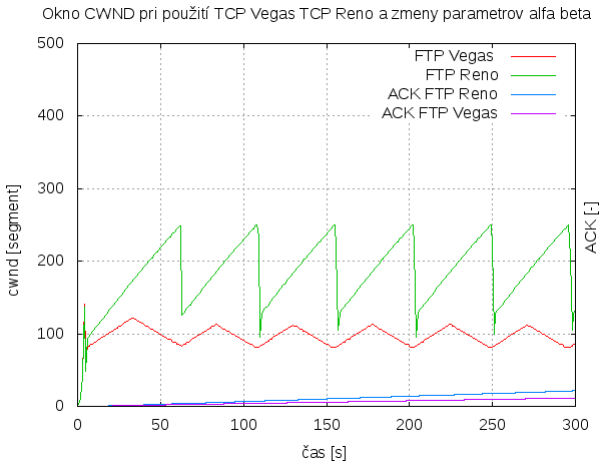
- $\alpha = 8$ a $\beta = 8$
- $\alpha = 25$ a $\beta = 25$
- $\alpha = 46$ a $\beta = 46$
- $\alpha = 32$ a $\beta = 64$

Kde sa kombinácia parametrov $\alpha = 8$ a $\beta = 8$ dá pokladať za najlepšiu v rámci konzistencie a pomer $\alpha = 32$ a $\beta = 64$ za najvyššie možné okno CWND .

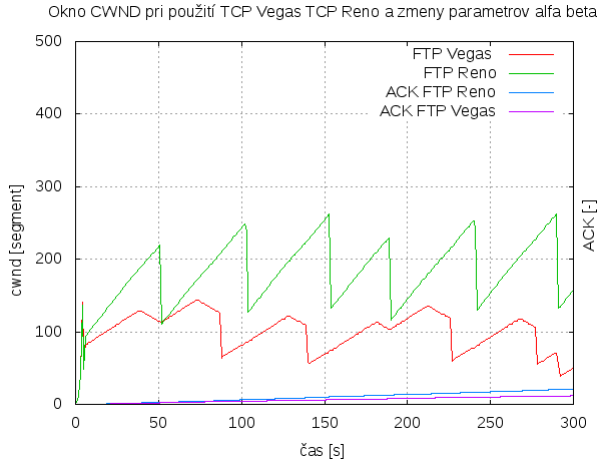


Obr. 4.7: Model parametrov α, β a okna CWND metody TCP Vegas a TCP Reno.

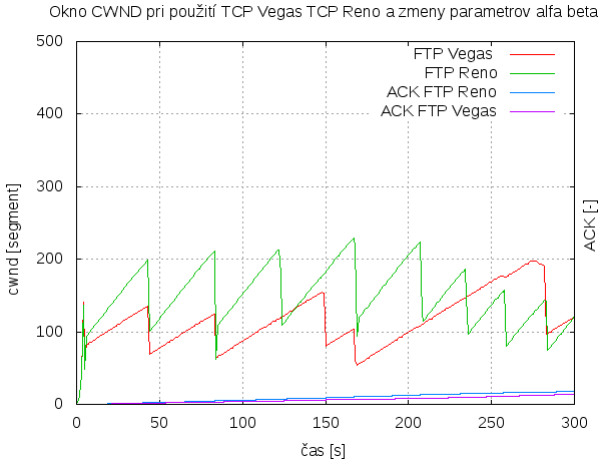
Podobne ako v prípade samostatného prenosu TCP Vegas, bol aj pre prenos kombinácie metódy TCP Vegas a TCP Reno vytvorený model parametrov vzhľadom k oknu CWND. Simuláciu pre parametre $\alpha = 8$ a $\beta = 8$ zobrazuje graf 4.8. Je možné vidieť zásadnú zmenu oproti základným parametrom graf 3.7. TCP Vegas sa nespráva chaoticky ale mení svoje okno CWND v pravidelných intervaloch s konštantnými hodnotami. Zároveň si udržuje tieto hodnoty po celú dobu prenosu. Graf 4.9 zobrazuje nastavenie $\alpha = 25$ a $\beta = 25$. Nastavenie v rámci týchto metód nebolo vyhovujúce, prinieslo nepriaznivý výsledok, došlo k zníženiu okna CWND a TCP Vegas sa správal podobne ako prenos so základnými parametrami. Chovanie sa parametrov v rámci okna CWND popisuje model 4.7, kde je vidieť značne zníženie okna. Parametre boli následovne zvýšené, $\alpha = 46$ a $\beta = 46$ graf 4.10, u týchto hodnôt došlo ku značnému vzrastu okna CWND čo je aj vidieť v modelu 4.7. Parametre a však nepriniesli len zvýšenie tohoto okna, TCP Vegas už nedokázalo presne predpovedať vznik zahltenia, preto muselo využívať techniku TCP Rena, kde po duplicitných potvrdeniach znižovalo svoje okno na polovicu. Úmernosť medzi oknom CWND a parametrami α a β sa zdala byť nedostačujúca.



Obr. 4.8: Graf okna zahltenia CWND pri použití parametrov $\alpha = 8$ a $\beta = 8$ TCP Vegas a TCP Reno.

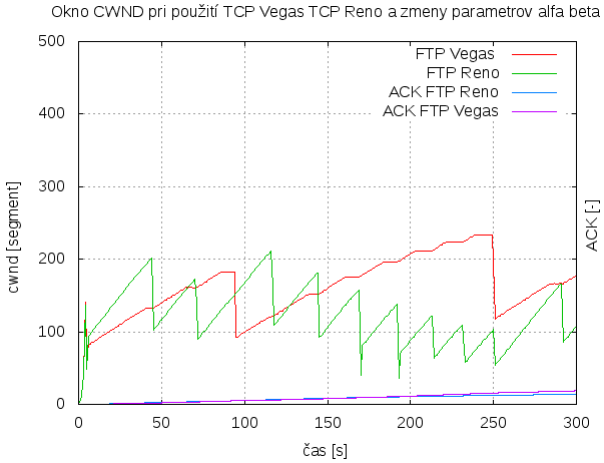


Obr. 4.9: Graf okna zahltenia CWND pri použití parametrov $\alpha = 25$ a $\beta = 25$ TCP Vegas a TCP Reno.

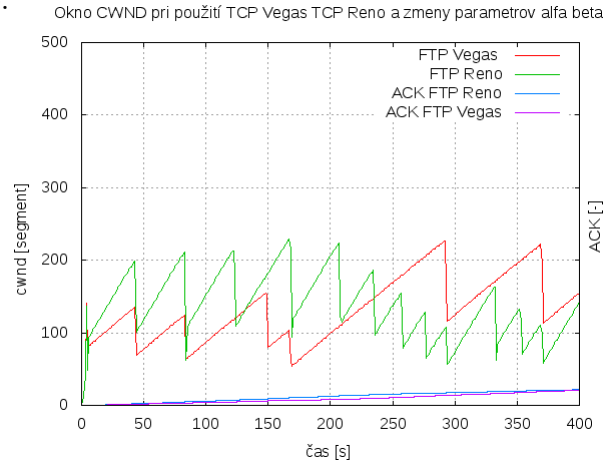


Obr. 4.10: Graf okna zahltenia CWND pri použití parametrov $\alpha = 46$ a $\beta = 46$ TCP Vegas a TCP Reno.

Preto bola zvolená posledná hodnota parametrov. Graf 4.11 zobrazuje nastavenie $\alpha = 32$ a $\beta = 64$, u týchto hodnôt došlo k opätovnému rastu okna CWND, model 4.7 nám zobrazuje, že v tomto prípade okno TCP Vegas bolo schopné prenášať väčšie okno. Veľkosť okna však bola na úkor vzniknutého zahltenia a takmer úplnej straty predikcie. Podstatným rozdielom je rast okna TCP, ktoré je stále dané samotnou predikciou, preto sa okno TCP Vegas nezvyšuje tak kriticky rýchlo ako je tomu u TCP Rena. Vďaka tomuto postupu vzniká niekoľko násobne menej zahltení v rámci TCP Vegas. Následovné je schopné preniesť viac dát s menším počtom vzniknutých zahltení a strát. Pri zvýšení parametrov nad hodnotu $\alpha = 32$ a $\beta = 64$, sa začne TCP Vegas správať chaoticky a jeho okno CWND sa dá pokladať za neefektívne, môžeme ho pozorovať na grafe 4.12.

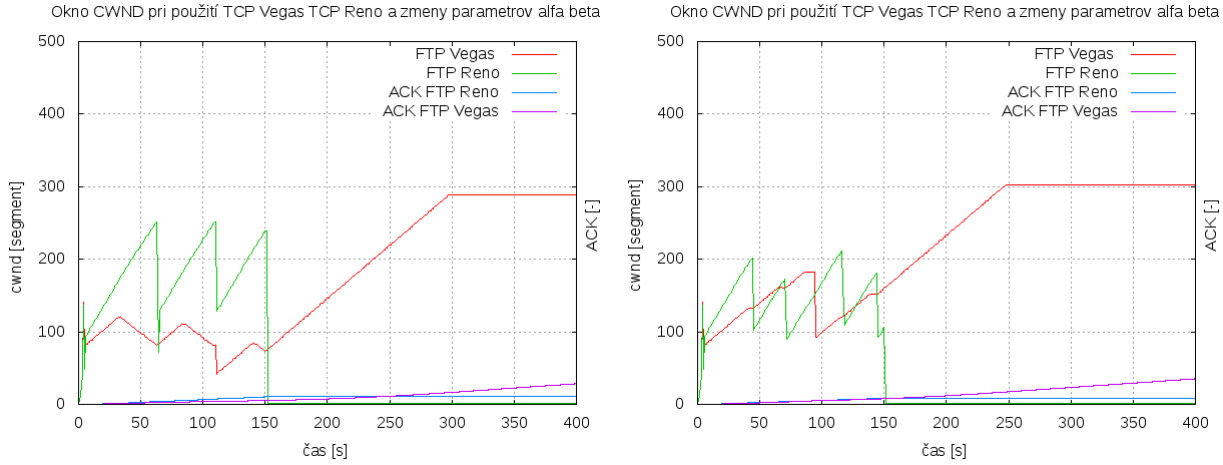


Obr. 4.11: Graf okna zahltenia CWND pri použití parametrov $\alpha = 32$ a $\beta = 64$ TCP Vegas a TCP Reno.



Obr. 4.12: Graf okna zahltenia CWND pri použití parametrov $\alpha = 64$ a $\beta = 128$ TCP Vegas a TCP Reno.

Chaotické správanie sa dá pochopiť tak, že TCP Vegas na neschopnosť predvídať zahltenie, musí využiť nižšie mechanizmy k zabráneniu vzniku zahltenia. Model 4.7 zobrazuje zvýšenie okna CWND, čo bolo cieľom, lepšie zobrazenie celkového zvýšenia CWND je vidieť na grafoch 4.13(a)) a 4.13(b). Kde v 150 s došlo k úplnému zastaveniu druhého dátového toku a tým sa mohol prvý dátový tok dostať na svoje maximum. V prípade týchto dvoch grafoch je možné sledovať že pri nastavených parametroch $\alpha = 32$ a $\beta = 64$, je maximum okna CWND vyššie než v jeho základných hodnotách. Taktiež si bol mechanizmus TCP Vegas schopný zachovať svoju schopnosť predikcie, a aj s takto nastavenými parametrami pri samostatnom dátovom toku nedochádza k zahltení a prenos dátového toku sa dá považovať za konštantný.



(a) (b)

Obr. 4.13: (a) Graf okna CWND TCP Vegas a TCP Reno s pôvodnými parametrami, (b) graf okna CWND TCP Vegas a TCP Reno s $\alpha = 32$ a $\beta = 64$

5 ZÁVER

Cieľom práce bolo oboznámiť sa s mechanizmami riadenia dátového toku na úrovni transportnej vrstvy používané protokolom TCP, ich preskúmaním a overením efektívnosti pri prenose dát. Podrobným preskúmaním a analýzou parametrov TCP Vegas

Práca sa delí na štyri časti. Prvá časť rozoberá umiestnenie Transportnej vrstvy v rámci TCP/IP modelu a jeho zrovnanie s ostatnými vrstvami. Neoddeliteľnou časťou tejto vrstvy je TCP protokol ktorému bol venovaný značný priestor, práca popisuje jeho vlastnosti, funkciu, a spôsob fungovania v rámci komunikácie. V rámci teoretického popisu protokolu TCP boli rozobraté voliteľné implementácie, ako možnosti prístupu k dátovým jednotkám na úrovni transportnej vrstvy. Podstatnou činnosťou protokolu je dohlaď nad dátovým tokom počas prenosu, kedy sa snaží predísť alebo riešiť možné vzniknuté stavy zahltenia. Boli definované dva stavy prenosu, fáza Pomalého štartu a fáza vyhýbania sa zahlteniu. Funkčnosť týchto fáz sa môže líši použitými metódami na ovládanie okna zahltenia. Druhá časť sa venuje práve týmto metódam. Popisuje metódu Thaoe, ktorá vznikla ako prvá metóda pre spravu stavu zahltenia, implementuje v sebe jednoduché mechanizmy pomalého štartu a rýchleho opakovaného prenosu. Následovne sa zo základným myšlienok tejto metódy odvodili ďalšie, ktoré by splňovali požiadavky dnešnej doby. Práca tieto metódy delí ma tri úrovne, mechanizmy pre drátové siete, mechanizmy pre bez drátové siete a mechanizmy pre satelitné spojenia. Každá úroveň v sebe zahrňuje zoznam metód pre ňu dostupných s popisom ich základnej funkčnosti.

Tretia časť podrobne rozoberá metódy TCP Reno a TCP Vegas. Tieto dve metódy zahrňujú rozdielne princípy, ktorými sa dá ku správe okna pristupovať. Druhá časť taktiež popisuje simuláciu týchto dvoch metód, ukazuje ich výhody a nevýhody pri prenose dát. Zobrazuje spôsob spracovania dátového toku len za použitia TCP Vegas, a prenos dát za kombinácie TCP Vegas a TCP Reno. Z dosiahnutých výsledkov je vidieť, že v prípade použitia TCP Vegas sa delí prenosová kapacita nerovnomerne. Čo spôsobuje nadradenosť jedného z dátových tokov. V prípade kombinácie metód je TCP Vegas na najnižšej možnej úrovni, kde vplyvom TCP Reno nieje schopná udržiavať dostatočne veľké okno zahltenia.

Záver práce v sebe zahrňuje práve podrobnú analýzu TCP Vegas. Táto metóda je riadená za pomoci parametrov α a β, ktoré slúžia ako hranice pre navyšovanie a znižovanie okna zahltenia. Boli vytvorené dva modely na základe zmeny parametrov a zmeny okna zahltenia. Prvý model popisuje vplyv parametrov na zmenu

okna pri použití TCP Vegas na paralelný prenos dát. Boli nájdené vhodne pomery parametrov, podľa ktorých sa dokázala prenosová kapacita správne rozdeliť medzi oba dátové toky. Druhý model popisuje vplyv parametrov na zmenu okna zahltenia v kombinácií TCP Vegas a TCP Reno. Pretože TCP Reno pracuje na iných princípoch než TCP Vegas nieje možné spravodlivo rozdeliť prenosovú kapacitu. Snaha tohoto modelu bola pozdvihnúť TCP Vegas z jeho minima, a presiahnuť TCP Reno vo veľkosti okna zahltenia. Pre túto kombináciu odpovedali dve kombinácie parametrov. Tým prvým bol pomer $\alpha = 8$ a $\beta = 8$, kde sa TCP Vegas prestal správať chaoticky a menil svoje okno v pravidelných intervaloch s konštantnými hodnotami. V s týmito parametrami nedochádzalo k zahlteniu v rámci TCP Vegas a prenos dátového toku bol stabilný. Druhou kombináciou bol pomer $\alpha = 32$ a $\beta = 64$, kde TCP Vegas presiahol okno TCP Reno. Nevýhodou bol vznik zahltenia v samotnom TCP Vegas, aj stýmto nedostatkom stále vykazoval lepšie prenosové vlastnosti voči TCP Reno, Zahltenia spôsobené touto kombináciou sa vyskytovali v menších počtoch, a rast okna zahltenia TCP Vegas sa nezvyšoval kriticky rýchlo vďaka funkčnosti predikcie zahltenia.

LITERATÚRA

[1] KABELOVÁ, Alena a Libor DOSTÁLEK. *Velký průvodce protokoly TCP/IP a systémem DNS.* 5., aktualiz. vyd. Brno: Computer Press, 2008, 488 s. ISBN 978-80-251-2236-5.

[2] PUŽMANOVÁ, Rita. *TCP/IP v kostce.* 2. upr. a rozš. vyd. České Budějovice: Kopp, 2009, 619 s. ISBN 978-80-7232-388-3.

[3] SINGH, Brijendra. *Data communications and computer networks.* 2nd ed. New Delhi: Prentice_Hall of India, 2006. ISBN 978-812-0329-690.

[4] Doporučenie RFC 2001: *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms* [online]. Január 1997 [cit. 2014-11-14]. dostupné online: `http://tools.ietf.org/html/rfc2001`

[5] Ghassan A. Abed, Mahamod Ismail and Kasmiran Jumari, *Behavior of cwnd for TCP Source Variants over Parameters of LTE Networks. Information Technology Journal, 10: 663-668* [online]. 2011 [cit. 2014-11-14]. dostupné online: `http://scialert.net/abstract/?doi=itj.2011.663.668`

[6] YAN CHEUNG, Shun. *Mathematics & Computer Science. Congestion Avoidance in TCP* [online]. 2008 [cit. 2014-11-14]. dostupné online: `http://www.mathcs.emory.edu/~cheung/Courses/558a/Syllabus/6-transport/TCP.html`

[7] A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas. *EECS Instructional Support Group Home Page* [online]. 2005 [cit. 2014-11-14]. dostupné online: `http://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf`

[8] Doporučenie RFC 2582: *The NewReno Modification to TCP's Fast Recovery Algorithm* [online]. Január 1991 [cit. 2014-11-14]. dostupné online: `https://tools.ietf.org/html/rfc2582`

[9] XU, Lisong, Khaled HARFOUSH a Injong RHEE. *Binary Increase Congestion Control for Fast, Long Distance Networks* [online]. 2004 [cit. 2014-11-14]. dostupné online: `http://an.kaist.ac.kr/courses/2006/cs540/reading/bic-tcp.pdf`

[10] XU, Lisong a Injong RHEE. *CUBIC: A New TCP-Friendly High-Speed TCP Variant* [online]. 2006 [cit. 2014-11-14]. dostupné online: `http://www4.ncsu.edu/~rhee/export/bitcp/cubic-paper.pdf`

[11] Doporučenie RFC 3649: *HighSpeed TCP for Large Congestion Windows* [online]. December 2003 [cit. 2014-11-14]. dostupné online: `http://www.ietf.org/rfc/rfc3649.txt`

[12] SHORTEN, Robert, Douglas LEITH. *H-TCP: TCP for high-speed and long-distance networks* [online]. 2006 [cit. 2014-11-14]. dostupné online: `http://www.hamilton.ie/net/htcp3.pdf`

[13] MORRIS, Robert. *Scalable TCP Congestion Control* [online]. 2000 [cit. 2014-11-14]. dostupné online: `http://pdos.csail.mit.edu/~rtm/papers/tp.pdf`

[14] S. BRAKMO, Lawrence, Sean O'MALLEY a Larry L. PETERSON. *TCP Vegas: New Techniques for Congestion Detection and Avoidance* [online]. 1994 [cit. 2014-11-14]. dostupné online: `http://pages.cs.wisc.edu/~akella/CS740/F08/740-Papers/BOP94.pdf`

[15] KUZMANOVIC, Aleksandar a Edward W. KNIGHTLY. *TCP-LP: Low-Priority Service via End-Point Congestion Control* [online]. 2014 [cit. 2014-11-14]. dostupné online: `http://networks.blogs.rice.edu/files/2014/08/tcp-lp.pdf`

[16] LIU, Shao, Tamer BAS¸AR a R. SRIKANT. *TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks* [online]. 2006 [cit. 2014-11-14]. dostupné online: `http://www.ifp.illinois.edu/~srikant/Papers/liubassri06perf.pdf`

[17] TAN, Kun, Jingmin SONG, Qian ZHANG a Murari SRIDHARAN. *TA Compound TCP Approach for High-speed and Long Distance Networks* [online]. 2005 [cit. 2014-11-14]. dostupné online: `http://research.microsoft.com/pubs/70189/tr-2005-86.pdf`

[18] PENG FU, Cheng *TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks* [online]. 2003 [cit. 2014-11-14]. dostupné online: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1177186`

[19] KLIAZOVICH, Dzmitry, Fabrizio GRANELLI a Daniele MIORANDI. *TCP Westwood+ Enhancement in High-Speed Long-Distance Networks* [online]. 2006 [cit. 2014-11-14]. dostupné online: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4024212`

[20] CAININ, Carlo, FIRRINCIELI Rosario. *TCP Hybla: a TCP enhancement for heterogeneous networks* [online]. 2004 [cit. 2014-11-14]. dostupné online:

http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/Papers/TCP-Hybla.pdf

[21] SHIHADA, Basem, Qiong ZHANG, Pin-Han HO a Jason P. JUE. *A Novel Implementation of TCP Vegas for Optical Burst Switched Networks* [online]. 2010 [cit. 2014-11-14]. dostupné online: http://hph16.uwaterloo.ca/~bshihada/publications/OSN10-Vegas.pdf

[22] The Network Simulator - ns-2. *Information Sciences Institute* [online]. 2010 [cit. 2014-12-11]. dostupné online: http://www.isi.edu/nsnam/ns/

[23] A Linux TCP implementation for NS2. *The California Institute of Technology* [online]. 2006 [cit. 2014-12-11]. dostupné online: http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/

[24] INTERNATIONAL, Sponsored by ACM SIGCOMM with support given by the Information Sciences and Technology Center of SRI. *SIGCOMM '88 Symposium, communications, architectures protocols: Stanford, California*. August 16-19, 1988. New York, N.Y: Association for Computing Machinery, 1988. ISBN 0897912799.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

DTE	Data terminal equipment
DCE	Data Communications Equipment
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
LLC	Logical Link Control
MAC	Media Access Control
OUI	Organization Unique Identifier
LAN	Local area network
WAN	Wide Area Network
MSS	Maximum segment size
ISN	Initial Sequence NUmber
CWND	Congestion Window Size
SSTHRESH	Slow Start Threshold
BIC	Binary Increase Congestion control
CUBIC	Cubic Binary Increase Congestion control
HS-TCP	Highspeed TCP
H-TCP	Hamilton TCP
S-TCP	Scalable TCP
W-TCP	TCP Westwood+
TCP-LP	TCP Low-Priority
TCP-I	TCP Illinois
C-TCP	Compound TCP
TCP-H	TCP Hybla
TCP-V	TCP Vegas
RTO	Retransmission Timeout Value

ZOZNAM PRÍLOH

A NASTAVENIE SIMULÁTORU NS2

A.1 Konfiguračný súbor TCP Reno

```
#Make a NS simulator
  set ns [new Simulator]

  # Define a 'finish' procedure
  proc finish {} {
     exit 0
  }

  # Create the nodes:
  set n0 [$ns node] # odosielací uzol
  set n1 [$ns node] # prímaci uzol



  # Create the links:
  $ns duplex-link $n0 $n1 10Mb 170ms DropTail #vytvorenie trasy

  #vytvorenie odosielateľa pre prvý prenos
  set tcp1 [new Agent/TCP/Linux]
  $tcp1 set packetSize_ 1448 #veľkosť paketu
$tcp1 set  window_ 3000    #veľkosť okna
  $tcp1 set timestamps_ true #časová značka
$tcp1 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp1 select_ca reno" # zvolenie metody zahltenia "TCP Reno"
  $ns attach-agent $n0 $tcp1

  #vytvorenie odosielateľa pre druhý prenos
  set tcp2 [new Agent/TCP/Linux]
  $tcp2 set packetSize_ 1448 #veľkosť paketu
$tcp2 set window_ 3000   #veľkosť okna
  $tcp2 set timestamps_ true #Časová značka
$tcp2 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp2 select_ca reno" # zvolenie metódy zahltenia "TCP Reno"
  $ns attach-agent $n0 $tcp2
```

```
#vytvorenie príjemcu pre prvý prenos
set sink1 [new Agent/TCPSink/Sack1/DelAck]
$sink1 set ts_echo_rfc1323_ true  #dodatočný parameter pre lepšie odosielanie
$sink1 set generateDSacks_ false  #voľba potvrdení, kumulatívne ACK
$sink1 set interval_ 200ms #voľba odosielacieho času ACK
$ns attach-agent $n1 $sink1

#vytvorenie príjemcu pre druhý prenos
set sink2 [new Agent/TCPSink/Sack1/DelAck]
$sink2 set ts_echo_rfc1323_ true #dodatočný parameter pre lepšie odosielanie
$sink2 set generateDSacks_ false #voľba potvrdení, kumulatívne ACK
$sink2 set interval_ 200ms #voľba odosielacieho času ACK
$ns attach-agent $n1 $sink2

#nastavenie správnej trasy pre oba prenosy
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2

#vytvorenie FTP prenosu pre prvý prenos
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

#vytvorenie FTP prenosu pre druhý prenos
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

#nastavenie časovačov pre zahájenie simulácie, a oboch FTP prenosov
$ns at 0.1   "$ftp1 start"
$ns at 700.0 "$ftp1 stop"
$ns at 0.1   "$ftp2 start"
$ns at 700.0 "$ftp2 stop"
# Set simulation end time
$ns at 1000.0 "finish"

set outfile1  [open  "WinFile1"  w] #ukladanie nameraných hodnôt
set outfile2  [open  "WinFile2"  w] #ukladanie nameraných hodnôt
```

```
  ##################################################
  ## Obtain CWND from TCP agent
  ##################################################

  proc plotWindow {tcpSource outfile} {
     global ns

     set now [$ns now]
     set cwnd [$tcpSource set cwnd_] #zachytávanie cwnd
    set ack [$tcpSource set ack_]   #zachytávanie ack

  ###Print TIME CWND   for  gnuplot to plot progressing on CWND and ack
     puts  $outfile  "$now $cwnd $ack" #ukladanie výstupu

     $ns at [expr $now+1] "plotWindow $tcpSource  $outfile"
  }

  $ns  at  0.0  "plotWindow $tcp1  $outfile1"
$ns  at  0.0  "plotWindow $tcp2  $outfile2"

  # Run simulation !!!!
  $ns run #začiatok simulácie
```

A.2 Konfiguračný súbor TCP Vegas

```
#Make a NS simulator
  set ns [new Simulator]

  # Define a 'finish' procedure
  proc finish {} {
     exit 0
  }

  # Create the nodes:
  set n0 [$ns node] # odosielací uzol
  set n1 [$ns node] # prímaci uzol
```

```
  # Create the links:
  $ns duplex-link $n0 $n1 10Mb 170ms DropTail #vytvorenie trasy

  #vytvorenie odosielateľa pre prvý prenos
  set tcp1 [new Agent/TCP/Linux]
  $tcp1 set packetSize_ 1448 #veľkosť paketu
$tcp1 set  window_ 3000    #veľkosť okna
  $tcp1 set timestamps_ true #časová značka
$tcp1 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp1 select_ca vegas" # zvolenie metody zahltenia "TCP Vegas"
  $ns attach-agent $n0 $tcp1

  #vytvorenie odosielateľa pre druhý prenos
  set tcp2 [new Agent/TCP/Linux]
  $tcp2 set packetSize_ 1448 #veľkosť paketu
$tcp2 set window_ 3000   #veľkosť okna
  $tcp2 set timestamps_ true #Časová značka
$tcp2 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp2 select_ca vegas" # zvolenie metódy zahltenia "TCP Vegas"
  $ns attach-agent $n0 $tcp2

  #vytvorenie príjemcu pre prvý prenos
  set sink1 [new Agent/TCPSink/Sack1/DelAck]
  $sink1 set ts_echo_rfc1323_ true  #dodatočný parameter pre lepšie odosielanie
  $sink1 set generateDSacks_ false  #voľba potvrdení, kumulatívne ACK
  $sink1 set interval_ 200ms #voľba odosielacieho času ACK
  $ns attach-agent $n1 $sink1

  #vytvorenie príjemcu pre druhý prenos
  set sink2 [new Agent/TCPSink/Sack1/DelAck]
  $sink2 set ts_echo_rfc1323_ true #dodatočný parameter pre lepšie odosielanie
  $sink2 set generateDSacks_ false #voľba potvrdení, kumulatívne ACK
  $sink2 set interval_ 200ms #voľba odosielacieho času ACK
  $ns attach-agent $n1 $sink2

  #nastavenie správnej trasy pre oba prenosy
  $ns connect $tcp1 $sink1
  $ns connect $tcp2 $sink2
```

```
#vytvorenie FTP prenosu pre prvý prenos
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

#vytvorenie FTP prenosu pre druhý prenos
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

#nastavenie časovačov pre zahájenie simulácie, a oboch FTP prenosov
$ns at 0.1   "$ftp1 start"
$ns at 700.0 "$ftp1 stop"
$ns at 0.1   "$ftp2 start"
$ns at 700.0 "$ftp2 stop"
# Set simulation end time
$ns at 1000.0 "finish"

set outfile1  [open  "WinFile1"  w] #ukladanie nameraných hodnôt
set outfile2  [open  "WinFile2"  w] #ukladanie nameraných hodnôt

#####################################################
## Obtain CWND from TCP agent
#####################################################

proc plotWindow {tcpSource outfile} {
   global ns

   set now [$ns now]
   set cwnd [$tcpSource set cwnd_] #zachytávanie cwnd
 set ack [$tcpSource set ack_]   #zachytávanie ack

###Print TIME CWND   for  gnuplot to plot progressing on CWND and ack
   puts  $outfile  "$now $cwnd $ack" #ukladanie výstupu

   $ns at [expr $now+1] "plotWindow $tcpSource  $outfile"
}
```

```
  $ns  at  0.0  "plotWindow $tcp1  $outfile1"
$ns  at  0.0  "plotWindow $tcp2  $outfile2"

  # Run simulation !!!!
  $ns run #začiatok simulácie
```

A.3 Konfiguračný súbor TCP Vegas so zmenou paramterov

```
#Make a NS simulator
  set ns [new Simulator]

  # Define a 'finish' procedure
  proc finish
     exit 0



  # Create the nodes:
  set n0 [$ns node] # odosielací uzol
  set n1 [$ns node] # prímaci uzol

  # Create the links:
  $ns duplex-link $n0 $n1 10Mb 170ms DropTail #vytvorenie trasy

  #vytvorenie odosielateľa pre prvý prenos
  set tcp1 [new Agent/TCP/Linux]
  $tcp1 set packetSize_ 1448 #veľkosť paketu
$tcp1 set  window_ 3000    #veľkosť okna
  $tcp1 set timestamps_ true #časová značka
$tcp1 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp1 select_ca vegas" # zvolenie metody zahltenia "TCP Vegas"
  $ns attach-agent $n0 $tcp1

  #vytvorenie odosielateľa pre druhý prenos
  set tcp2 [new Agent/TCP/Linux]
  $tcp2 set packetSize_ 1448 #veľkosť paketu
$tcp2 set window_ 3000   #veľkosť okna
  $tcp2 set timestamps_ true #časová značka
$tcp2 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp2 select_ca vegas" # zvolenie metódy zahltenia "TCP Vegas"
  $ns attach-agent $n0 $tcp2

  #vytvorenie príjemcu pre prvý prenos
  set sink1 [new Agent/TCPSink/Sack1/DelAck]
```

```
$sink1 set ts_echo_rfc1323_ true  #dodatočný parameter pre lepšie odosielanie
$sink1 set generateDSacks_ false  #voľba potvrdení, kumulatívne ACK
$sink1 set interval_ 200ms #voľba odosielacieho času ACK
$ns attach-agent $n1 $sink1

#vytvorenie príjemcu pre druhý prenos
set sink2 [new Agent/TCPSink/Sack1/DelAck]
$sink2 set ts_echo_rfc1323_ true #dodatočný parameter pre lepšie odosielanie
$sink2 set generateDSacks_ false #voľba potvrdení, kumulatívne ACK
$sink2 set interval_ 200ms #voľba odosielacieho času ACK
$ns attach-agent $n1 $sink2

#nastavenie správnej trasy pre oba prenosy
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2

#vytvorenie FTP prenosu pre prvý prenos
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

#vytvorenie FTP prenosu pre druhý prenos
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

#nastavenie časovačov pre zahájenie simulácie, a oboch FTP prenosov
$ns at 0.1   "$ftp1 start"
$ns at 700.0 "$ftp1 stop"
$ns at 0.1   "$ftp2 start"
$ns at 700.0 "$ftp2 stop"
# Set simulation end time
$ns at 1000.0 "finish"

set outfile1  [open  "WinFile1"  w] #ukladanie nameraných hodnôt
set outfile2  [open  "WinFile2"  w] #ukladanie nameraných hodnôt
```

```
  ##################################################
  ## Obtain CWND from TCP agent
  ##################################################

  proc plotWindow tcpSource outfile
     global ns

     set now [$ns now]
     set cwnd [$tcpSource set cwnd_] #zachytávanie cwnd
   set ack [$tcpSource set ack_]   #zachytávanie ack

  ###Print TIME CWND   for  gnuplot to plot progressing on CWND and ack
     puts  $outfile  "$now $cwnd $ack" #ukladanie výstupu

     $ns at [expr $now+1] "plotWindow $tcpSource  $outfile"


  $ns  at  0.0  "plotWindow $tcp1  $outfile1"
$ns  at  0.0  "plotWindow $tcp2  $outfile2"

#nastavenie parametrou alfa beta, kde x = (8, 25, 46) je hodnota parametru
$ns at 0.0  "$tcp(1) set_ca_default_param vegas alpha x"
$ns at 0.0  "$tcp(1) set_ca_default_param vegas beta x"
$ns at 0.0  "$tcp(2) set_ca_default_param vegas alpha x"
$ns at 0.0  "$tcp(2) set_ca_default_param vegas beta x"

  # Run simulation !!!!
  $ns run #začiatok simulácie
```

A.4 Konfiguračný script pre TCP Vegas a TCP Reno so zmenou parametrov

```
#Make a NS simulator
  set ns [new Simulator]

  # Define a 'finish' procedure
  proc finish
     exit 0



  # Create the nodes:
  set n0 [$ns node] # odosielací uzol
  set n1 [$ns node] # prímaci uzol

  # Create the links:
  $ns duplex-link $n0 $n1 10Mb 170ms DropTail #vytvorenie trasy

  #vytvorenie odosielateľa pre prvý prenos
  set tcp1 [new Agent/TCP/Linux]
  $tcp1 set packetSize_ 1448 #veľkosť paketu
$tcp1 set  window_ 3000    #veľkosť okna
  $tcp1 set timestamps_ true #časová značka
$tcp1 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp1 select_ca vegas" # zvolenie metody zahltenia "TCP Vegas"
  $ns attach-agent $n0 $tcp1

  #vytvorenie odosielateľa pre druhý prenos
  set tcp2 [new Agent/TCP/Linux]
  $tcp2 set packetSize_ 1448 #veľkosť paketu
$tcp2 set window_ 3000   #veľkosť okna
  $tcp2 set timestamps_ true #Časová značka
$tcp2 set partial_ack_ true #zapnutie parciálneho odosielania
  $ns at 0 "$tcp2 select_ca reno" # zvolenie metódy zahltenia "TCP Reno"
  $ns attach-agent $n0 $tcp2
```

```
#vytvorenie príjemcu pre prvý prenos
set sink1 [new Agent/TCPSink/Sack1/DelAck]
$sink1 set ts_echo_rfc1323_ true  #dodatočný parameter pre lepšie odosielanie
$sink1 set generateDSacks_ false  #voľba potvrdení, kumulatívne ACK
$sink1 set interval_ 200ms #voľba odosielacieho času ACK
$ns attach-agent $n1 $sink1

#vytvorenie príjemcu pre druhý prenos
set sink2 [new Agent/TCPSink/Sack1/DelAck]
$sink2 set ts_echo_rfc1323_ true #dodatočný parameter pre lepšie odosielanie
$sink2 set generateDSacks_ false #voľba potvrdení, kumulatívne ACK
$sink2 set interval_ 200ms #voľba odosielacieho času ACK
$ns attach-agent $n1 $sink2

#nastavenie správnej trasy pre oba prenosy
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2

#vytvorenie FTP prenosu pre prvý prenos
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

#vytvorenie FTP prenosu pre druhý prenos
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

#nastavenie časovačov pre zahájenie simulácie, a oboch FTP prenosov
$ns at 0.1   "$ftp1 start"
$ns at 700.0 "$ftp1 stop"
$ns at 0.1   "$ftp2 start"
$ns at 700.0 "$ftp2 stop"
# Set simulation end time
$ns at 1000.0 "finish"

set outfile1  [open  "WinFile1"  w] #ukladanie nameraných hodnôt
set outfile2  [open  "WinFile2"  w] #ukladanie nameraných hodnôt
```

```
  ##################################################
  ## Obtain CWND from TCP agent
  ##################################################

  proc plotWindow tcpSource outfile
     global ns

     set now [$ns now]
     set cwnd [$tcpSource set cwnd_] #zachytávanie cwnd
   set ack [$tcpSource set ack_]   #zachytávanie ack

  ###Print TIME CWND   for  gnuplot to plot progressing on CWND and ack
     puts  $outfile  "$now $cwnd $ack" #ukladanie výstupu

     $ns at [expr $now+1] "plotWindow $tcpSource  $outfile"


  $ns  at  0.0  "plotWindow $tcp1  $outfile1"
$ns  at  0.0  "plotWindow $tcp2  $outfile2"

#nastavenie parametrou alfa beta, kde x = (8, 25, 46, 128) je hodnota parametru
$ns at 0.0  "$tcp(1) set_ca_default_param vegas alpha x"
$ns at 0.0  "$tcp(1) set_ca_default_param vegas beta x"

  # Run simulation !!!!
  $ns run #začiatok simulácie

  # Run simulation !!!!
  $ns run
```