FAKUTLA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITY KOMENSKÉHO
BRATISLAVA

OVPLYVŇOVANIE PRENOSOVÝCH CHARAKTERISTÍK TCP/IP KOMUNIKÁCIE

DIPLOMOVÁ PRÁCA

AUTOR: MARIAN MORÁVEK
VEDÚCI: RNDR. RICHARD OSTERTÁG

BRATISLAVA
APRÍL 2005

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry.

V Bratislave
27. apríla 2005 _____

Ďakujem svojmu vedúcemu diplomovej práce, RNDr. Richardovi Ostertágovi za cenné rady a pripomienky pri písaní tejto práce.

Obsah

Úvod

Do globálnej siete Internet sa pripája čoraz väčší počet používateľov. Správcovia siete a poskytovatelia pripojenia preto v súčasnosti čelia problémom s riadením sieťovej premávky a menežovaním prístupu do tejto siete. Ide hlavne o obmedzovanie jednotlivých používateľov čo do rýchlosti prenosu dát, ako aj o diferenciáciu premávky, zabezpečenie rozličných požiadaviek pre zákazníkov a poskytnutie určitej kvality služby (QoS quality of service).

Reálne problémy, ktoré sa vyskytujú, sú napríklad, aby užívatelia sťahujúci veľké objemy dát nepriaznivo neovplyvňovali iných užívateľov, ktorí pracujú s terminálovými službami typu telnet alebo surfujú po internetových stránkach. Je potrebné zabezpečiť rovnomerné rozdelenie rýchlosti medzi používateľov zdieľajúcich spoločnú linku, alebo zaručiť dohodnuté rýchlosti. Pre každého používateľa jednotlivo treba taktiež zabezpečiť, aby popri sťahovaní veľkého objemu dát niekoľkými tokmi, mohol surfovať po webových stránkach a otvárať ďalšie toky. Ide o diferenciáciu premávky, ktorá je dnes stále na nízkej úrovni a zväčša pri otvorení zopár tokov sťahujúcich dáta dochádza k tomu, že pripojenie poskytnuté poskytovateľom nereaguje dobre na ďalšie podnety zo strany užívateľa (surfovanie po webe, pridanie ďalšieho toku sťahujúceho dáta a podobne) a tieto činnosti bývajú spomalené alebo úplne nefunkčné. Poskytnutie pripojenia pre užívateľa nesmie viesť k preťaženiu alebo zahlteniu siete. Sieť musí mať mechanizmy ako sa proti preťaženiu brániť a ako zabezpečiť stálu dostupnosť sieťových prostriedkov. Toto sa dosahuje obmedzovaním prenášaných dát a tvarovaním sieťovej premávky tzv. ***shaping***. V súčastnosti vzniká aj problém so zabezpečením rýchlosti prenosu dát pre jednotlivých používateľov. Kým v minulosti táto rýchlosť značne závisela od technológie, ktorá tvorila hornú hranicu rýchlosti (modemy, ISDN) dnes sa k poskytovateľom pripájajú užívatelia rýchlejšími technológiami (ADSL, mikrovlnné pripojenie, ethernet) a je na strane poskytovateľa, aby im reguloval horný limit rýchlosti. Aj pre toto obmedzovanie rýchlosti sa zaužíval pojem ***shaping*** a existuje niekoľko algoritmov, spomenutých v tejto práci, ktoré toto zabezpečujú.

V dnešnom Internete je ešte stále väčšina dát po sieti prenášaná protokolom TCP. Tento v sebe implementuje vlastné mechanizmy kontrolujúce tok vysielaných dát. Pre sieťového administrátora je výhodné poznať tieto mechanizmy a taktiež poznať ako spolupracujú s metódami na riadenie a obmedzovanie premávky.

Táto práca sa zaoberá prenosovými charakteristikami protokolu TCP, metódami ich ovplyvňovania a podáva prehľad systému kontroly premávky, ktorý je dostupný v operačnom systéme Linux. Cieľom prvej kapitoly je podať prehľad algoritmov, ktoré sa stali súčasťou protokolu TCP počas jeho vývoja, a ktoré vplývajú na prenosové charakteristiky tohoto protokolu. Kapitola sa nesnaží detailne popísať všetky možnosti a algoritmy protokolu TCP, sústredí sa iba na tie črty, ktoré sú podstatné pri ovplyvňovaní prenosu. V kapitole je taktiež poukázané na niektoré limity a nedostatky protokolu.

Cieľom druhej kapitoly bolo podať ucelený prehľad metód, ktoré sa používajú pri kontrole sieťovej premávky a poukázať na ich silné a slabé stránky. Súčasťou kapitoly je aj popis hierarchického zdieľania linky, ktoré sa používa ako nástroj na tvarovanie premávky v operačnom sytéme Linux.

Náplňou tretej kapitoly bolo obohatiť prácu o simulácie, ktorých cieľom je empirické meranie chovania sa protokolu TCP za rôznych podmienok ako aj to, ako viacero tokov TCP zdieľa spoločnú linku.

Posledná, štvrtá kapitola, je venovaná téme kontroly premávky, ktorá je dostupná v operačnom systéme Linux. Cieľom bolo preniknúť do problematiky implementácie tejto kontroly, ktorá je veľmi slabo zdokumentovaná a zistiť možnosti a úroveň tejto implementácie.

1. Základné charakteristiky protokolu TCP

1.1 Úvod

Termínom TCP/IP sa zvykne označovať rodina protokolov IP, TCP a UDP, ako aj referenčný model týchto protokolov. Pri diskusii o sieťových protokoloch sa zvykne porovnávať s refenčným modelom ISO/OSI, čo je sedem vrstvový štandardný model určný pre sieťovú komunikáciu (nebudeme ho bližšie rozoberať, vymenujeme len vrstvy: fyzická, linková, sieťová, transportná, spojovacia, prezentačná, aplikačná). Prvým dvom vrstvám v referenčnom modeli OSI/ISO prislúcha v modeli TCP/IP vrstva, ktorá sa zvykne označovať ako host-to-network vrstva a nie je bližšie špecifikovaná. Vyžaduje sa od nej iba to, aby vedela prenášať pakety vrstvy IP. Tretej vrstve ISO/OSI prislúcha vrstva *IP* (internet protokol). Tento protokol je *nespoľahlivý* (negarantuje doručenie paketu v sieti) protokol, ktorý nevytvára *spojenia* (pred samotnou komunikáciu oboch strán nie je fáza, ktorá by vytvorila spojenie medzi koncovými stanicami, na konci sa spojenie neruší) a posiela po sieti entity nazvané *pakety* (prúd dát rozdelený na menšie skupiny s doplňujúcimi informáciami, určenými na prenos). Štvrtej vrstve v modeli ISO/OSI prislúcha protokol *TCP* (transmission control protokol) ako aj protokol *UDP* (user datagram protokol). V súčastnosti je TCP protokol dominantným protokolom používaným v IP sieťach. Uvádza sa, že 80-95 % premávky vo veľkých IP sieťach je prenášaných pomocou protokolu TCP. V práci sa venujeme najmä protokolu TCP a jeho prenosovým charakteristikám, protokol UDP je z tohto pohľadu pre nás nezaujímavý. Vrstve 5 a 6 v modeli ISO/OSI nepríslúcha žiaden protokol v modeli TCP/IP. Siedmej vrstve ISO/OSI prislúcha rovnako nazvaná vrstva v modeli TCP/IP - *aplikačná vrstva.* Sem sa zvyknú zaradiť všetky protokoly, ktoré na svoj prenos používajú protokoly TCP alebo UDP. Patria sem napríklad protokoly telnet, ssh, ftp, http, nfs a mnoho ďalších.

1.2 Základný popis protokolu TCP

Cieľom protokolu TCP je prenos dát, ktorý by sa dal charakterizovať nasledovnými vlastnosťami:

- *spojovaný* (pred samotným vysielaním dát sa vytvorí spojenie)
- *potvrdzovaný* (pre každý poslaný segment (jednotka posielaná po sieti na úrovni protokolu TCP) sa generuje potvrdzovacia správa ACK)
- *spoľahlivý* (protokol garantuje, že sa dáta prenesú, alebo ak to nie je možné, tak to oznámi vyššej vrstve)
- *zachovávajúci poradie prenášaných údajov* (dáta prichádzajú vyššej vrstve v tom poradí v akom boli vyslané)

TCP spája vždy dvoch účastníkov komunikácie. Vytvorené spojenie je *full duplexné*, čo znamená, že obe strany môžu dáta vysielať aj prijímať súčastne. Pre jednoduchosť budeme v ďalšom texte uvažovať, že sa jedná o *half duplexné*, čo bude pre nás znamenať, že dáta vysiela iba jedna strana (*vysielateľ*) a druhá ich prijíma (*prijímateľ*) a vysiela naspäť potvrdzovaciu správu ACK.

V ďalšom texte sa predpokladajú základné znalosti protokolu TCP. Patria k nim nasledovné mechanizmy:

- sliding window system
- mechanizmus nadväzovania a rušenia spojenia
- stavy spojenia TCP
- mechanizmus potvrdzovania vyslaných segmentov

Čitateľ ich môže nájsť napríklad v [1], [2] alebo v [4]. V tejto kapitole sa budeme sústrediť najmä na mechanizmy, ktoré považujeme za kľúčové v otázkach prenosových charakteristík protokolu TCP.

1.3 Zabezpečenie spoľahlivosti TCP

Na prenosové vlastnosti protokolu TCP má výrazný vplyv spôsob, akým je implementovaná spoľahlivosť prenosu. Spoľahlivosť je zabezpečená:

- potvrdzovaním poslatých segmentov potvrdzovacími správami ACK
- jednoduchým preposielaním stratených segmentov

1.3.1 Potvrdzovanie poslatých segmentov

Potvrdzovanie je jednoúrovňové v tom zmysle, že iba vysielajúca strana dostáva pre vyslané dáta potvrdzovaciu správu ACK. Na príjem ACK už neodpovedá ďalším potvrdením. To značí, že prijímajúca strana sa nemá ako presvedčiť, že potvrdenie ACK ktoré poslala, skutočne došlo. V dôsledku tejto jednoduchej metódy dochádza k preposielaniu dát v sieti aj vtedy, keď dôjde iba k strate potvrdzovacej správy ACK a nie reálne prenášaných dát.

Retransmisia paketu závisí aj od potvrdzovacej politiky ktorá sa dohodne na začiatku spojenia. TCP štandartne podporuje kumulatívnu potvrdzovaciu politiku (cumulative acknowledgement system), novšie varianty umožňujú aj vylepšenú selektívnu potvrdzovaciu politiku tzv. SACK (bližšie napr. v RFC 2018). Pri použití kumulatívnej potvrdzovacej politiky sa používa okrem vypršania časovača pre príjem potvrdzovacej správy ACK aj iný signál oznamujúci možnú stratu paketov. Je to príjem viacerých duplicitných potvrdzovacích správ ACK (v literatúre sa zvyknú označovať ako dupACK alebo out-of-orderACK), ktorými príjemca potvrdzuje viackrát za sebou tie isté dáta. Táto situácia nastáva vo chvíli, keď príjemca obdrží dáta v inom poradí v akom boli vyslané. Špecifikácia TCP vyžaduje, aby o tomto fakte príjemca bezodkladne oboznámil vysielajúcu stranu a vygeneroval dupACK. Príjemca uvedenú správu môže interpretovať dvomi možnými spôsobmi:

1) v sieti došlo k preusporiadaniu dát a dáta eventuálne neskôr prídu
2) v sieti došlo k strate zopár segmentov, avšak ďalšie segmenty stále prichádzajú k príjemcovi

1.3.2 Preposielanie stratených segmentov

Z pohľadu vysielateľa sa pre každý vyslaný segment spustí časovač. Ak pre daný segment príde do uplynutia časovača potvrdzovacia správa ACK, tak sa daný segment považuje za úspešne prenesený. Ak potvrdzovacia správa ACK nepríde do vypršania časovača, vysielateľ predpokladá stratu posielaného segmentu a pošle ho znova. Interval časovača pre takto preposlatý segment sa zdvojnásobí čo sa zvykne označovať ako *exponencial backoff*. Existuje horná hranica intervalu časovača, kedy sa segment považuje za nedoručiteľný. Nastavenie tejto hodnoty závisí od implementácie.

1.3.3 Interval časovača

Z hľadiska využitia prenosovej kapacity linky je dôležité správne nastavenie intervalu časovača pre každý segment. Tento interval hovorí o tom, koľko sa má čakať na prijatie potvrdzujúcej správy ACK pre daný segment. Ak by sme nastavili hodnotu príliš nízko, nemuseli by nám pri výkyvoch v sieti dostatočne rýchlo prísť potvrdzujúce správy ACK a zaplnili by sme zbytočne sieť duplicitnými dátami. Pre veľké hodnoty by musela aplikácia na druhej strane zbytočne dlho čakať na dáta, ktoré sa stratili prenosom v sieti.

Ako ***RTT (round trip time)*** sa zvykne označovať čas za ktorý vyšle vysielateľ segment a vráti sa mu naspäť potvrdzovacia správa ACK. Keďže podmienky v sieti sa neustále menia, je potrebné udržiavať a znovu merať hodnotu RTT. Klasická metóda výpočtu intervalu časovača špecifikovaná v [RFC 739] je nasledovná. Označme si ***M*** ako aktuálne nameranú hodnotu RTT pre nejaký segment ***S***. Označme ***R*** ako odhad RTT a budeme ho počítať nasledovne:

$$R = a * R + (1 - a)M$$

a je vyhladzovací faktor a odporúča sa použiť hodnotu $a = 0{,}9$. Možno si všimnúť, že odhad ***R*** závisí z 90% na predchádzajúcej hodnote ***R*** a iba z 10% na novo nameranej hodnote ***M***. Z toho plynie, že malé výkyvy veľmi hodnotu R nezmenia. Samotný interval časovača, ktorý budeme označovať aj ***RTO (retransmit timeout)*** sa potom vypočíta nasledovne

$$RTO = b * R$$

b faktor sa označuje ako ***delay variance*** alebo ***RTT variance*** a odporúča sa nastaviť hodnotu $b = 2$. Van Jacobson & Karels [3] neskôr zmenili tento algoritmus a uvádzajú že je potrebné okrem sledovania vyhladeného odhadu ***R*** sledovať aj ***zmenu RTT***. Takto sa interval viacej prispôsobuje zmenám v podmienkach prenosu. ***RTO*** sa potom počíta na základe nasledovných ukazovateľov:

- meranie RTT jednotlivých segmentov (***M***)
- udržiavanie vyhladeného odhadu RTT (***A***)
- udržiavanie vyhladenej strednej odchýlky RTT (***D***)

Stredná odchýlka je síce len aproximácia štandardnej odchýlky, je však jednoduchšia na výpočet, keďže sa nemusí používať operácia odmocnenia. Autori zostavili nasledovné rovnice pre výpočet ***intervalu časovača RTO***.

$$d = \mathrm{M} - \mathrm{A}$$
$$\mathrm{A} = \mathrm{A} + 1/8 * d$$
$$\mathrm{D} = \mathrm{D} + ¼ (|d| - D)$$
$$RTO = A + 4 * D$$

1.4 Kontrola toku protokolu TCP (flow control)

V protokole TCP sa ako *kontrola toku* zvykne označovať „kontrola toku medzi vysielateľom a prijímateľom“. Poznáme ešte druhý typ kontroly toku, ktorý sa nazýva *predchádzanie zahlteniu (congestion control)*.

Kontrola toku dát je realizovaná systémom sliding window a bola súčasťou protokolu TCP skôr ako metóda predchádzania zahlteniu. Pri každom prenose segmentu si obe strany posielajú veľkosti svojich okien. Veľkosťami okna sa v každom segmente oznamuje, ako je daná strana pripravená prijímať dáta. Môže dôjsť aj k uzavretiu okna, kedy strana vyšle segment oznamujúci nulovú veľkosť okna. Ide o signál že daná strana nie je schopná spracovávať žiadne ďalšie dáta a teda sa musí s posielaním ďalších dát čakať. Veľkosť okna rieši problém „rýchly vysielateľ, pomalý prijímateľ“. Pomalý prijímateľ zmenšením veľkosti okna oznamuje vysielateľovi, že nestíha prijímať dáta a žiada aby vysielateľ znížil rýchlosť vysielania.

1.5 Predchádzanie zahlteniu protokolu TCP (congestion control)

Zahltením siete sa rozumie stav, kedy do siete vstupuje viac dát ako je sieť v danom momente schopná preniesť.

1.5.1 Zahltenie z pohľadu smerovačov

Z pohľadu smerovača môže zahltenie nastať z dvoch príčin:

1) rýchla vstupná linka sa snaží dostať priveľa paketov na pomalú výstupnú linku, alebo viacej vstupných liniek je smerovaných cez jednu výstupnú, pričom suma vstupných rýchlostí jednotlivých liniek je väčšia ako kapacita výstupnej linky
2) vnútorná priepustnosť smerovača je obmedzená. Smerovač nemusí zvládať záťaž dát prechádzajúcich cez neho, aj keď sú výstupné porty nezahltené

Zahltenie siete je z pohľadu smerovačov riešené nasledovne: smerovače sú v podstate iba jednoduché mechanizmy, ktoých úlohou je paket prijať, vypočítať preňho výstupnú linku, uložiť ho do fronty a neskôr preposlať na výstupnú linku (tzv. store-forward mechanizmus). Pre každú výstupnú linku si smerovače držia frontu, kde sa ukladajú pakety určené na poslatie (tzv. *output queueing*). Výstupná linka má konečnú šírku prenosového pásma a ak

dáta prichádzajú prirýchlo, tak sa fronta začína plniť. Pri veľkej záťaži dochádza k zahadzovaniu paketov, keďže fronty majú taktiež obmedzenú kapacitu a prichádzajúce pakety nie je kam ukladať. Existuje viacero metód zahadzovania paketov, spomenuté budú v kapitole 2. Metódy zahadzovania paketov majú značný dopad na prenosové charkteristiky protokolu TCP. Ovplyvňujú napríklad prenosovú rýchlosť TCP a spôsob zdieľania kapacity výstupnej linky (tzv. fairness).

1.5.2 Význam a história metódy predchádzanie zahlteniu

Originálna špecifikácia protokolu TCP (RFC 793) v sebe zahrňovala iba jeden typ kontroly toku spomínaný v kapitole 1.4. Táto kontrola slúžila na ochranu príjemcu pred preplnením jeho vyrovnávajúcej pamäte, ktorá bola určená pre dané spojenie. Pôvodná špecifikácia síce uvádzala, že k strate segmentov môže dochádzať v dôsledku chýb na sieti alebo zahltenia siete, ale nešpecifikovala mechanizmy na dynamické prispôsobenie rýchlosti toku ako reakciu na zahltenie.

V RFC 896 sa uvádza, že v komplexných sieťach, kde sa používal protokol IP spolu s protokolom TCP podľa špecifikácie RFC 793, boli zistené nezvyčajné problémy so zahltením siete. IP smerovače boli náchylné k fenoménu nazvanom *congestion collapse - kolaps v dôsledku zahltenia*. Uvádza sa, že v októbri 1986 zažil Internet prvý kolaps v dôsledku zahltenia. Linka ktorá mala priepustnosť 32kbps prenášala efektívne dáta iba rýchlosťou 40bps, ostatná kapacita linky bola využitá na viacnásobné prenášanie stratených segmentov.

Pri nadviazaní spojenia si obe strany oznámia veľkosti svojich prijímacích okien. Na veľkosti nie je kladený žiadny limit. V súčasnosti sú počítače veľmi rýchle (rozumej rádovo rýchlejšie ako sieť) a tak by si teoreticky mohli oznámiť čo najväčie okná. TCP protokol potom začne posielať dáta rýchlosťou výstupnej linky. Otvorenie takéhoto spojenia môže znamenať úplné preťaženie a smerovače by začali zahadzovať veľké množstvo paketov. TCP by strácalo pakety, ktoré by stále preposielalo v nezmenenej rýchlosti. Neformálne by sa dalo povedať, že kolaps siete v dôsledku zahltenia vznikne, keď zvýšenie zaťaženia siete vedie k malému využitiu skutočnej kapacity sieťe. V TCP sieťach sa dialo vtedy, keď TCP preposielalo segmenty ktoré boli **ešte v sieti** (prenášali sa v čase keď vysielateľ vyslal znova taký istý segment, lebo z nejakých príčin predpokladal jeho stratu) alebo segmentov, ktoré už **prijímateľ prijal** (ale vysielateľ sa z nejakého dôvodu toto nedozvedel a predpokladal stratu segmentu).

Kolaps siete v dôsledku zahltenia sa dá rozdeliť na dva typy (podľa [5]):

- *klasický kolaps* – je to situácia, ktorá vedie k tomu, že momentálna efektívna kapacita siete je iba zlomok reálnej, aj keď je sieť zahltená (tento problém rieši metóda predchádzanie zahlteniu protokolu TCP, ktorá bude popísaná nižšie)
- *nový kolaps* – vzniká v dôsledku nedoručených paketov. Táto situácia vzniká, ak určitá časť siete je využitá na prenos paketu, ktorý je v nejakom smerovači zahodený kvôli zahlteniu ešte pred dosiahnutím cieľa. Táto časť siete potom zbytočne mrhá svojimi prenosovými kapacitami.

Popularita Internetu spôsobila, že vznikli a stále vznikajú propiertálne implementácie protokolu TCP. Niektoré z nich nesprávne implementujú kontrolu zahltenia, iné implementujú agresívnejšie politiky pri prenose a bývajú označované ako rýchlejšie TCP. Logickým dôsledkom zavedenia čoraz agresívnejších protokolov je, že sa však vrátime späť k bodu, kedy už v sieti znova nebude žiaden mechanizmus na prevenciu zahltenia a Internet sa znovu stane zahlteným.

Jedna zo známych ciest, ako môže používateľ dostať zo siete väčšiu šírku prenosového pásma bez zmeny protokolu je zmeniť granularitu: otvorením viacerých spojení na koncovú stanicu. Dôvod tohto konania vychádza z predpokladu, že jednotlivé TCP toky súperiace o linku si medzi sebou túto linku rovnomerne rozdelia. Potom užívateľ vlastniaci viacero tokov, dostane vyšší zlomok kapacity linky. Takýmto spôsobom sa črtá iná cesta k zahlteniu – namiesto implementovania čoraz agresívnejších politík pri prenose, otvoriť čoraz viacej kanálov pri prenose.

Z hľadiska implementácie a podpory kontroly zahltenia sa protokoly zvyknú klasifikovať do nasledovných skupín:

1) *TCP kompatibilný* – sú protokoly, ktoré majú implementovanú podporu prevencie proti zahlteniu podobne ako TCP
   Definícia: protokol je *TCP compatible* ak jeho tok v stabilnom stave (keď sa ustáli rýchlosť) nepoužije viacej kapacity zdieľanej linky ako ostatné toky s podobnými charakteristikami (RTT, stratovosť paketov – tieto charakteristiky vplývajú na rýchlosť prenosu protokolu TCP ako bude ukázané na konci kapitoly)
2) *Nezodpovedné protokoly* – nereagujú na stratu paketov a ani na zahltenie siete. Ide o stále narastajúce množstvo aplikácii založených na UDP protokole, ktorých kontrola zahltenia je neadekvátna alebo neexistujúca (to značí, že sa rýchlosť vysielania nezníži, ak aplikácia dostane signál o zahltení). Ak sa pri zahltení nespraví zo strany toku žiadna akcia, toto môže viesť k ďalšiemu kolapsu siete Internet. Z tohto pohľadu sa vyžaduje, aby aplikácie mali v sebe mechanizmy na predchádzanie zahltenia. Z pohľadu siete sa zase vyžaduje, aby bola schopná sa brániť voči takýmto tokom.
3) *Zodpovedné ale nekompatibilné s TCP* – majú implementovanú určitú kontrolu zahltenia, ale je buďto veľmi agresívna alebo úplne iná ako tá čo je implementovaná v TCP.

V diplomovej práci predpokladáme iba implementácie protokolu TCP, ktoré korektne implementujú kontrolu zahltenia t.j. *TCP compatibile*. Zároveň predpokladáme, že prenos po sieti sa bude realizovať jedným TCP spojením.

1.5.3 Implementácia kontroly zahltenia v protokole TCP

Algoritmy na kontrolu zahltenia boli navrhnuté pánom van Jacobsonom v roku 1986 a sú v súčasnosti vyžadované pri implementáciách TCP (viď RFC 2581). Tieto mechanizmy pracujú na koncových staniciach spojenia a spôsobujú, že TCP spomalí vysielanie v časoch zahltenia siete. V niektorých článkoch sa zvyknú označovať ako *Internet meltdown fix*.

Prevencia zahltenia bola teda vyriešená na úrovni protokolu TCP, tzn. že samotný protokol TCP je zodpovedný za predchádzanie zahlteniu.
Jedná sa o štyri algoritmy špecifikované v RFC 2001 a RFC 2581:

- Pomalý štart (Slow start)
- Predchádzania zahlteniu (Congestion avoidance)
- Rýchle preposielanie (Fast retransmit)
- Rýchle zotavenie (Fast recovery)

Ciele, ktoré by mali byť takto dosahované sú:

- vysoké využitie kapacity linky
- predchádzanie zahlteniu
- férovosť v zdieľaní linky s ostatnými tokmi

Pri návrhu algoritmov sa predpokladá, že k strate paketov dochádza z 99% pri preťažení siete a iba z veľmi malej časti – oveľa menšej ako 1% - v dôsledku chýb na linkách alebo nesprávneho smerovania. Preto strata paketu s najväčšou pravdepodobnosťou signalizuje preťaženie niekde na sieti. Pripomeňme, že signál o strate segmentu dostáva TCP protokol dvoma možnými spôsobmi:

1) pre daný segment vyprší časovač skôr ako príde potvrdzovacia správa ACK
2) príde viackrát za sebou rovnaká potvrdzovacia správa ACK, ktorá potvrdzuje príjem dát bezprostredne prechádzajúcich daný segment

Viacnásobné prijatie potvdzujúcej správy ACK, ktorá potvrdzuje tie isté údaje, signalizuje buďto stratu segmentu alebo zmenu poradia prijímaných segmentov. Pre úplnosť treba dodať, že ak príjemca dostane segment, ktorý prišiel v inom poradí ako očakáva, musí túto skutočnosť ihneď oznámiť vysielajúcej strane. Toto sa realizuje tak, že príjemca (znovu) pošle potvrdzovaciu správu ACK za dáta, ktoré doteraz kontinuálne prijal, čím signalizuje, že čaká na príchod dát ktoré nasledujú bezprostredne za nimi.

1.5.4 Pomalý štart a kontrola zahltenia

Pomalý štart (slow start)

TCP protokol nezačne po nadviazaní spojenia posielať pakety plnou rýchlosťou výstupného rozhrania až po naplnenie veľkosti okna na strane príjemcu. Miesto toho posiela pakety pomaly tak, že pošle jeden paket, počká až kým mu príde potvrdzovacia správa ACK, na korú zareaguje vyslaním dvoch paketov do siete atď. Na každú prišlú potvrdzovaciu správu ACK pošle do siete dva ďalšie pakety. Rýchlosť vysielania sa postupne zvyšuje až po hodnotu, kedy sa začnú pakety strácať. Táto fáza sa nazýva fáza *pomalého štartu* (pomalý preto, lebo vysielanie paketov sa začína pomaly). Zvyšovanie rýchlosti vo fáze pomalého štartu sa deje exponenciálne.

Predchádzanie zahlteniu (congestion avoidance)

Pomalý štart je implementovaný v spolupráci s algoritmom *predchádzanie zahlteniu*, aj keď ide o dva nezávislé algoritmy. Tento algoritmus sa snaží predchádzať zahlteniu siete tým spôsobom, že tempo zvyšovania rýchlosti zmení z agresívneho exponenciálneho na pomalé lineárne. Algoritmus reaguje na stratu paketov znížením aktuálnej rýchlosti na polovicu a zmenou tempa zvyšovania rýchlosti z exponenciálneho na lineárne, aby sa pomaly blížil k bodu zahltenia.

Implemetácia

Pre každé spojenie sa pre stranu posielateľa dát popri veľkosti vysielacieho okna *snd.wnd* (čomu na strane prijímateľa odpovedá prijímacie okno) udržuje aj veľkosť tzv okna zahltenia (congestion window) *cwnd*. Zavádza sa pojem efektívne okno *wind,* ktoré je definované ako

wind = min (snd.wnd, cwnd)

Ďalej sa zavádza premenná *sstreshd* (slow start threshold – medzná hranica pomalého štartu), ktorá odďeľuje od seba fázu pomalého štartu a fázu predchádzania zahlteniu.

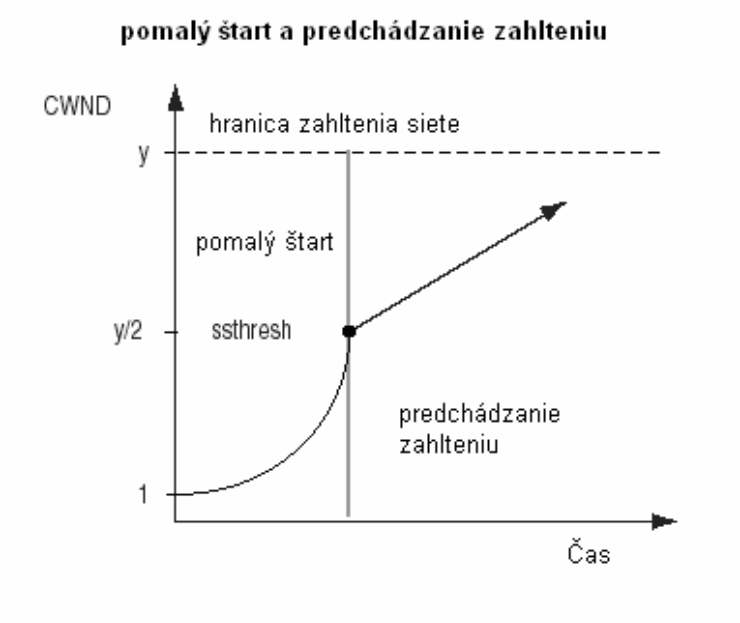
Význam premennej *cwnd* je udržiavať veľkosť okna, ktoré by sa malo zväčšovať postupne a tým by mala rásť aj prenosová rýchlosť spojenia. Má však rásť len do rozmerov, pri ktorých sa zistí zahltenie. Zahltenie sa zisťuje stratou paketov. Vtedy veľkosť okna *cwnd* klesne na jedna a premenná *sstreshd* sa nastaví na polovicu veľkosti okna, pri ktorom došlo k zahlteniu. Význam efektívneho okna je v tom že obmedzuje rýchlosť na úroveň ktorá by mala vyhovovať aj obmedzeniam v sieti aj obmedzeniam na strane príjemcu.

Na dvojicu okien *snd.wnd, cwnd* sa dá pozerať aj nasledovne: *cwnd* je kontrola toku na strane posielateľa – posielateľ (sender) nikdy nevypustí do siete viac nepotvrdených dát ako si myslí, že sieť stíha a *snd.wnd* je kontrola toku na strane prijímateľa – nikdy neviem naraz spracovať viac dát ako je *snd.wnd* (na strane prijímateľa je to *rcv.wnd*)

Formálnejšie je algoritmus navrhnutý takto[1]:

1) pri inicializácii spojenia sa nastaví *cwnd = 1* (segment), *sshtresh = 65535* (bajtov), resp. odpovedajúci počet segmentov
2) vysielateľ sa zaväzuje, že do siete nikdy nepošle naraz viac nepotvrdených dát ako je veľkosť *wind = min (snd.wnd, cwnd)*
3) ak detekujem zahltenie oboma metódami (t.j. vypršanie timera alebo duplikátny príchod takej istej potvrdzovacej správy ACK) tak nastavím *sshtresh = wind / 2* (segmenty, resp odpovedajúci počet bytov) (minimálne však 2 segmenty) a *cwnd = 1*
4) spojenie sa z hľadiska týchto dvoch algoritmov môže nachádzať v dvoch stavoch:
   - v stave pomalého štartu – to je za podmienky ak *cwnd* < *sshtresh*
   - v stave congestion avoidance – to je za opačnej podmienky *cwnd*>= *sshtresh*

5) podľa toho v akom stave je spojenie, tak sa zvyšuje hodnota okna cwnd nasledovne
   - pre pomalý štart cwnd = cwnd + 1 pre každú prijatú správu ACK
   - pre predchádzanie zahlteniu cwnd = cwnd + 1/cwnd pre každú prijatú správu ACK



Poznámky k algoritmu

- zaujímavý je vzťah medzi prijatím potvrdzovacej správy ACK a časom RTT (round trip time). Ak je veľkosť okna *cwnd* = *A*, tak v čase RTT by som mal prijať zhruba *A* potvrdzovacích správ ACK. To značí, že za čas RTT sa vo fáze prechádzanie zahlteniu zvýši veľkosť okna o 1 segment, kdežto pri metóde pomalého štartu o *A* (t.j. o toľko, koľko segmentov som poslal do siete).
- hodnoty *cwnd, sshtresh* sú v reálnych implemetáciách uvádzané v bajtoch, v schématickom popise sa pre prehľadnosť zvyknú uvádzať v segmentoch
- jedinou hodnotou ktorá sa zvyšuje postupne je hodnota *cwnd*. Hodnota *sshtresh* sa zvyšuje absolútne a to pri detekcii zahltenia. Jej úlohou je oddeliť od seba stav kedy sa zrýchľuje exponenciálne a kedy lineárne
- pomalý štart a predchádzanie zahlteniu nútia TCP protokol redukovať hodnotu *cwnd* na jedna, vždy keď sa detekuje strata paketu. Ak zahltenie siete trvá dlhší čas, množstvo premávky poslatej do siete klesá exponenciálne. Toto prispieva k vyprázdneniu front na smerovačoch a k odstráneniu zahltenia

1.5.5 Rýchle preposlatie a rýchle zotavenie

Tieto dva algoritmy sú modifikáciou a vylepšením predošlých dvoch metód. Jacobson ich navrhol v roku 1990. Riešia otázku, aký názor by mal mať vysielateľ na duplicitný príjem rovnakých potvrdzujúcich správ ACK. Tieto môžu znamenať preusporiadanie dát v sieti alebo naozajstnú stratu segmentov. Tieto algoritmy zároveň zvyšujú priepustnosť protokolu TCP pri strate malého počtu paketov.

Rýchle preposlatie (fast retransmit)

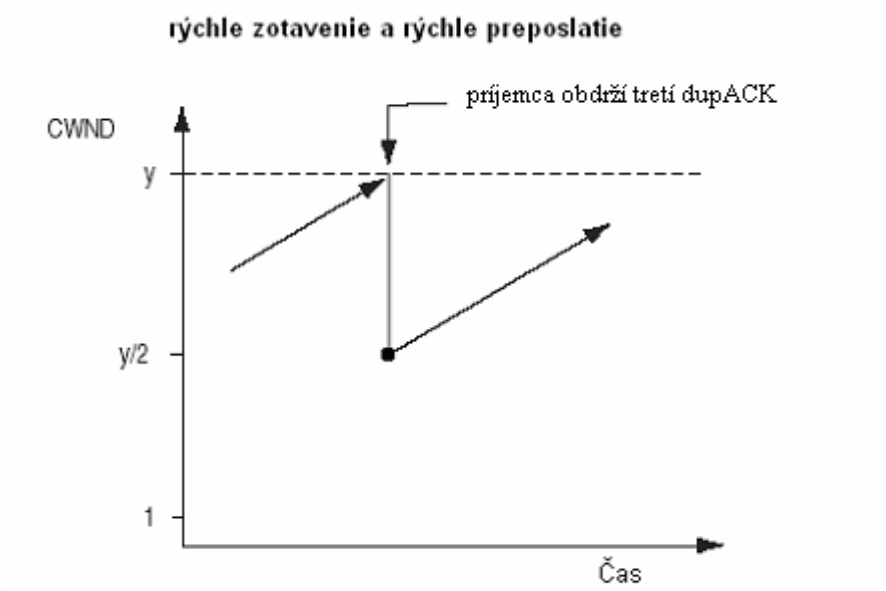
Pôvodná špecifikácia TCP predpokladá pri príjme správy dupACK, že došlo k strate segmentu a prepošle ho znova. Myšlienka rýchleho preposlatia je taká, že správa dupACK

skutočne znamená preusporiadanie paketov v sieti. Miesto toho, aby TCP okamžite reagovalo preposlaním chýbajúceho segmnetu, počká si až dostane tri rovnaké správy dupACK. Rýchle preposlatie zvyšuje výkonnosť TCP nasledovnými spôsobmi

- eliminuje nepotrebné preposielania paketov a plytvanie sieťovými prostriedkami, keď dôjde k preusporiadaniu paketov v sieti a nie k strate
- zvyšuje využiteľnosť skutočnej prenosovej kapacity

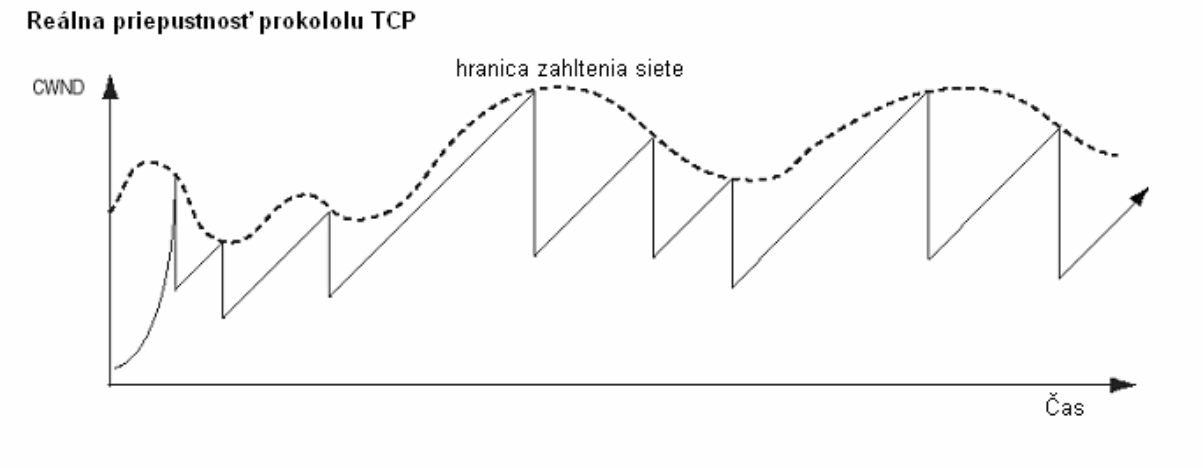
Rýchle zotavenie (fast recovery)

Keď vysielateľ obdrží správu dupACK, znamená to, že dáta stále tečú sieťou, pretože TCP vygeneruje túto správu iba ak prichádzajú ďalšie segmenty(v inom poradí akom by mali). V tomto prípade by pôvodná špecifikácia TCP išla do módu pomalého štartu a začala by veľkosťou okna *wind=1* (segment) (minimálna rýchlosť vysielania). Rýchle zotavenie zabezpečí až po obdržaní troch správ dupACK zníženie rýchlosti vysielania. Táto sa zníži iba na polovicu (nie na minimálnu rýchlosť) a spustí sa mód lineáneho zrýchľovania s fázou predchádzanie zahlteniu. Idea je taká aby sme ochránili linku pred úplným vyprázdnením po fáze rýchleho preposlatia. Každá správa dupACK reprezentuje paket, ktorý opustil linku a dostal sa až k prijímateľovi. To značí, že dáta stále prechádzajú a nie je dôvod začínať od nulovej rýchlosti.



Rýchle zotavenie zabraňuje TCP spojeniu, aby znížilo rýchlosť vysielania na nulu po vykonaní rýchleho preposlatia (znižovanie *cwnd* na hodnotu 1 sa deje už len pri vypršaní časovača). Fast retransmit pomáha iba vtedy, ak dôjde k strate jedného paketu. Nie je veľmi nápomocné pri stratách väčšieho množstva paketov.

Na obrázku nižšie je ilustrácia ako sa mení hodnota cwnd (plná tenká čiara) v závislosti od zahltenia siete (prerušovaná hrubá čiara) pri použití spomínaných algoritmov.

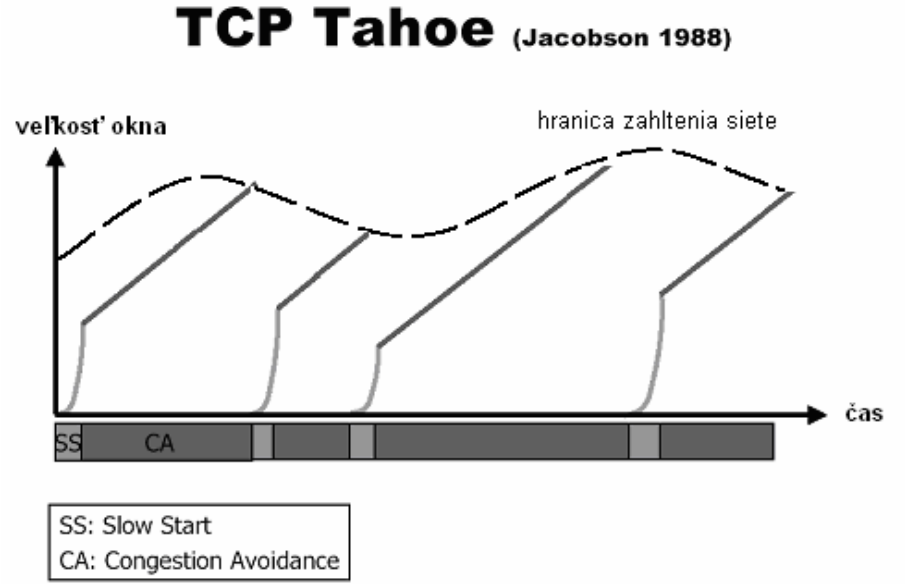


Pozn: Dá sa pozorovať, že okno rastie lineárne (vo fáze predchádzanie zahlteniu) a klesá exponenciálne (so stratou každého paketu) – toto sa zvykne označovať ako AIMD (aditive increase, multiplicative decrease) aditívny nárast, exponenciálny zostup.

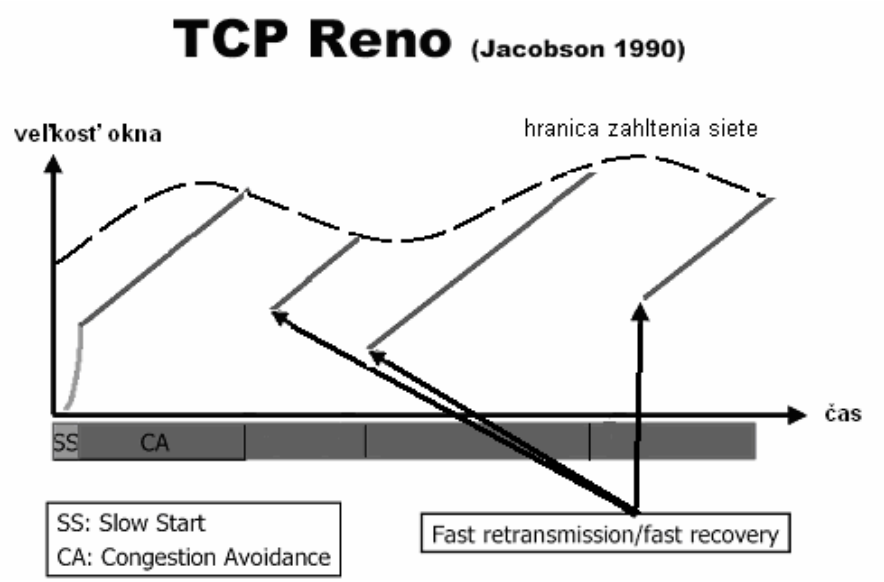
1.5.6 História nasadenia kontroly zahltenia

Systémy kontroly zahltenia navrhnuté van Jacobsonom boli postupne nasadzované do rôznych implementácií protokolu TCP. Neskôr boli modifikované a postupne optimalizované, hlavná myšlienka však stále ostala rovnaká. Nasleduje krátky zoznam rôznych implememntácií protokolu TCP a rozdielov ktoré v nich sú:

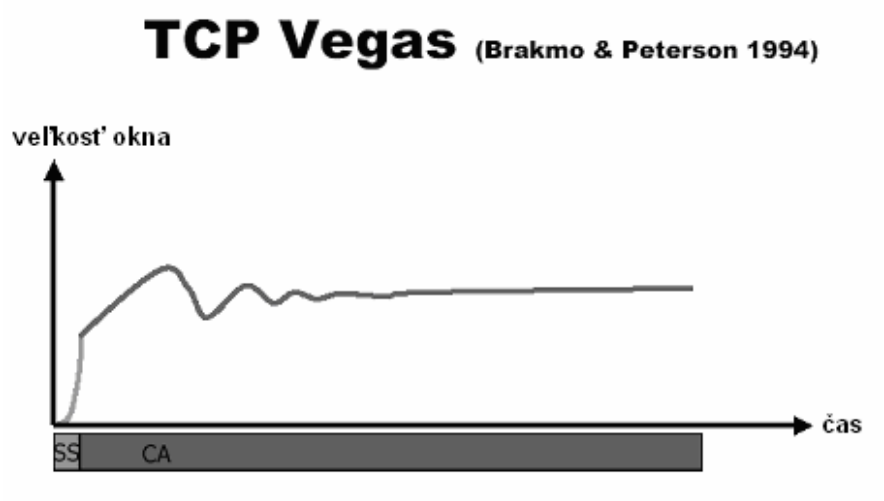
- **TC Tahoe**, implementovaný v 4.3BSD v roku 1988 obsahoval v sebe prvé tri mechanizmy z kontroly zahltenia – pomalý štart, predchádzanie zahlteniu, rýchle preposlatie



- **TCP Reno** implementovaný v 4.3BSD v r.1990 už v sebe obsahoval všetky štyri van Jacobsonove algoritmy. Reno zlepšilo nedostatky Tahoe v oblasti priepustnosti, keď nastáva strata jediného paketu.



- **TCP Vegas**, diskutovaný v rôznych dokumentoch z r. 1994 zmenilo celú myšlienku van Jacobsonových algoritmov na predchádzanie zahlteniu a zaviedlo nový predpoklad o výskyte zahltenia. Vegas sleduje RTT svojich paketov a pri zvyšovaní tejto hodnoty zmenšuje hodnotu *cwnd* a tým aj rýchlosť vysielania a naopak. Predpokladá, že RTT sa zvyšuje v dôsledku toho, že sa začínajú plniť výstupné fronty na smerovačoch a teda nastáva zahltenie. Zmena nastala aj v slow start algoritme, kde sa zvyšuje rýchlosť o polovicu pomalšie. V oblasti fairness, t.j. pri súperení o linku so staršími implementáciami protokolu TCP, je Vegas v nevýhode.



- **TCP SACK** špecifikovaný v RFC 2180 z r.1996 vylepšuje priepustnosť Reno v prípadoch, keď dôjde k strate viacerých paketov. Ak príjemca dostane pakety nekontinuálne, pošle vysielajúcej strane ACK s rozšírením o SACK. Tu mu oznamuje, ktoré pakety prijal a ktoré mu chýbajú. Vysielateľ potom môže preposlať iba chýbajúce segmenty.
- **TCP New Reno** špecifikovaný v RFC 2582 z apríla r.1999 vylepšuje priepustnosť, keď dôjde k strate viacerých paketov, ktoré patria do údajov v jednom okne. Toto platí pre implementáciu Reno, ktorá nepoužíva rozšírenie SACK.

1.5.7 Kritika kontroly zahltenia

Mechanizmy kontroly zahltenia zavedené do protokolu TCP vyriešili síce problém zahltenia a kolapsu siete, priniesli však so sebou ďalšie problémy, ktoré možno nájsť napr. v [6]. Sú to problémy v súvislosti s výkonom (priepustnosťou) na vysokorýchlostných sieťach (rádovo gigabity/s) a stále populárnejších bezdrátových sieťach.

- Na vysokorýchlostných sieťach je linérne zrýchľovanie veľmi pomalé. Výsledkom predchádzaniu zahlteniu je nevyužitie plnej kapacity linky a výrazné poddimenzovanie možností prenosového pásma linky. Pre ilustráciu možno uviesť tabuľku podľa [7]. V tejto tabuľke sú uvedené časy tzv. zotavenia sa protokolu zo straty paketu pri rôznych prenosových rýchlostiach. Tabuľka ilustruje modifikáciu algoritmov na predchádzanie zahlteniu v TCP pre vysokorýchlostné siete. Táto implementácia protokolu TCP bola nazvaná ScalableTCP. Autor predpokladá konštantný RTT=200ms. Na začiatku simulácie sa predpokladá, že TCP vysiela na plnej rýchlosti linky. Potom dôjde k strate paketu, na čo TCP zareaguje znížením rýchlosti na polovicu. V tabuľke sú uvedené časy, za ako dlho sa protokol TCP dostane späť na plnú rýchlosť linky.

Rate	**Štandartné TCP čas zotavenia**	**Scalable TCP čas zotavenia**
1Mbps	1.7s	2.7s
10Mbps	17s	2.7s
100Mbps	2min	2.7s
1Gbps	28min	2.7s
10Gbps	4hod 43min	2.7s

- Na bezdrátových sieťach zase neplatí predpoklad, na ktorom je postavená kontrola zahltenia protokolu TCP. V týchto sieťach dochádza k stratám paketov skutočne v dôsledku chybovosti a nie zahltenia siete. Ak TCP zistí stratu paketov, tak zníži rýchlosť vysielania na polovicu. Takto sa pri náhodných a krátkotrvajúcich interferenciách v dôsledku ktorých dochádza k stratám paketov znižuje prenosová rýchlosť protokolu a dochádza k slabému využitiu skutočnej prenosovej kapacity linky.

Z uvedeného vyplýva, že TCP potrebuje ďalšie mechanizmy na nasadenie pre siete tzv. novej generácie medzi ktoré patria gigabitové siete a bezdrátové siete. Pre vysokorýchlostné siete a linky LFN (viď 1.6) sa objavujú rôzne modifikácie protokolu TCP ako sú napríklad Fast-TCP, HS-TCP, Wetwood TCP alebo projekt Web100 [38]. Pre bezdrátové siete možno uviesť napríklad [39].

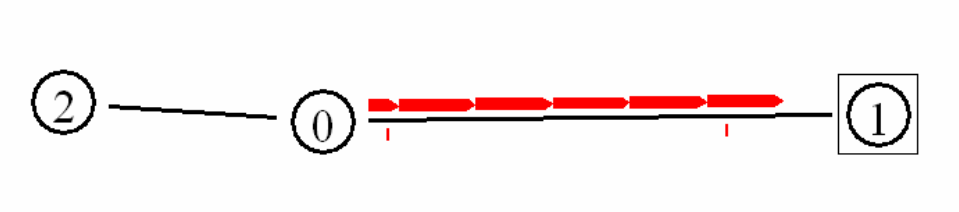
1.6 Bandwidth delay product linky a linky typu LFN

V RFC 1323 sa uvádza, že protokol TCP je navrhnutý tak, aby sa adaptoval na prenosy keď charakteristiky linky sú nasledovné: propagačné oneskorenie linky je v rozmedzí 1ms – 100sekúnd a prenosová kapacita od 100bit/s – 10^7 bit/s.

S týmito dvoma veličinami sa zvykne pre danú linku definovať charakteristika, ktorá sa nazýva bandwidth-delay product (nebudeme prekladať) BDP a je daná nasledovným vzťahom:

$$BDP = BW*delay$$

kde *BW* je prenosová kapacita linky (bandwidth) a *delay* je propagačné oneskorenie (t.j. oneskorenie v dôsledku šírenia sa signálu v linke). Veličina určuje kapacitu linky, t.j. koľko dát môže byť maximálne v danom časovom okamihu vo fáze prenosu. Veličina je dôležitým faktorom pri prenose TCP a určuje veľkosť okna potrebného na plné využitie kapacity linky.



Uvažujme linky s vysokou hodnotou BDP, ktoré sa zvyknú označovať ako linky typu LFN (elephant – long fat networks – dlhé hrubé linky). Zväčša sa jedná o transkontinentálne linky s vysokou priepustnosťou rádovo gigabity/s, ktoré však majú vysoké propagačné oneskorenie. Rýchlosť svetla v kábloch sa pohybuje na úrovni 200 000 km/s a napríklad signál z Európy do USA sa rozšíri za cca 20 ms. Ak zoberieme čas RTT, ako čas na vyslanie paketu a prijatie potvrdzovacej správy ACK, hodnota sa zdvojnásobuje a dostávame čas 40 ms (to značí, že 40 ms od odvysielania paketu sme schopní preňho prijať potvrdzovaciu správu ACK). Pre veľkosť okna protokolu TCP sú v hlavičke venované 2 byty a maximálna veľkosť poslatých a nepotvrdených bytov v sieti tak dosahuje hodnotu 65535. Toto je limitujúcim faktorom pôvodnej špecifikácie a takto obmedzená veľkosť okna zapríčiňuje, že jeden TCP tok nie je schopný využiť plnú prenosovú kapacitu linky.

Príklad: Uvažujme rýchlosť prenosu 1 Gbit/s t.j. 1 000 000 000 bit/s. Uvažujme RTT 40 ms (to značí že *delay = 20ms*). Dostávame hodnotu veľkosti okna *VO*, ktoré je potrebné na využitie plnej prenosovej kapacity linky. Je to množstvo dát, ktoré sú z pohľadu odosielateľa vo fáze prenosu a pre ktoré ešte neprišla potvrdzujúca správa ACK.

$$VO = 1\,000\,000\,000 * 0{,}04 = 40\,000\,000 \text{ bitov} = 5\,000\,000 \text{ bytov} = 5 \text{ Mbyte}$$

Hodnota 65535 bytov sa v tomto javí ako veľmi malá a TCP by pri danom RTT dosiahlo rýchlosť iba okolo

$$65\,535 / 0{,}04 = 1\,638\,375 \text{ byte/s} = 13\,107\,000 \text{ Mbit/s}$$

Preto RFC 1323 špecifikuje rozšírenie, ktoré je spätne kompatibilné a umožňuje interpretovať 16bitové číslo ako 32 bitové. Na použití tohto rozšírenia sa musia dohodnúť obe strany

spojenia vo fáze nadväzovania spojenia. Ak aspoň jedna strana toto nepodporuje, tak sa komunikuje podľa staršej špecifikácie.

1.7 Rovnica priepustnosti protokolu TCP

Táto kapitola hovorí o jednom zo spôsobov odvodenia známej rovnice maximálnej priepustnosti protokolu TCP. Rovnica je významná v tom, že nám dovoľuje zistiť parametre ktoré ovplyvňujú rýchlosť prenosu protokolu TCP. Pri odvodení rovnice sme postupovali podľa [9].

1.7.1 Rýchlosť vysielania TCP

Aktuálna rýchlosť vysielania protokolu TCP závisí od veľkosti efektívneho okna vysielateľa. Označme *W* ako veľkosť tohto okna v paketoch. Toto číslo označuje počet paketov, ktoré boli vyslané do siete a pre ktoré ešte neprišla potvrdzovacia správa ACK. Je to teda množstvo dát, ktoré sú práve prenášané sieťou na ceste k prijímateľovi alebo sú už prenesené u prijímateľa a je pre ne prenášaná potvrdzovacia správa ACK. Vysielateľ sa zaväzuje, že do siete nepošle naraz nikdy viacej dát ako je hodnota W paketov. Po odoslatí W paketov naraz môže byť do siete vyslaný jeden nový paket až vtedy, ak vysielateľ príjme aspoň jednu potvrdzovaciu správu ACK (toto sa zvykne označovať aj ako *self clocking* mechanizmus protokolu TCP).

Označme ďalej *T* ako hodnotu času RTT t.j. množstvo času ktoré uplynie po vyslaní paketu a príjmu správy ACK. Potom aktuálnu rýchlosť vysielania *BW* v segmentoch/sekundu dostaneme zo vzťahu

$$BW = W / T$$

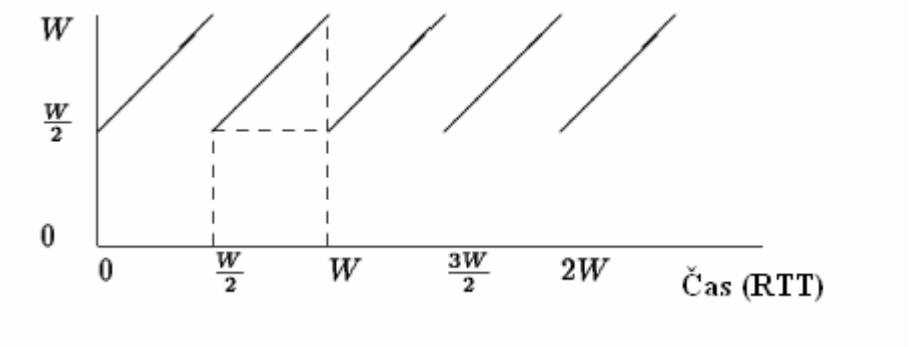
(resp v bytoch za sekundu zo vzťahu $BW = W*MSS / T$, kde *MSS* je maximum segment size – dohodnutá najväčšia veľkosť segmentu).

1.7.2 Maximálna priepustnosť TCP

Algoritmus na predchádzanie zahlteniu ovplyvňuje prenosové charakteristiky protokolu TCP v podmienkach slabej až miernej stratovosti paketov (pri veľkej strátovosti to koriguje interval časovača, ktorý môže vypršať skôr ako príde tretí dupACK). Zvyšovanie okna sa deje lineárne, keď sa zvýši aditívne o konštantu za každú hodnotu času RTT a pri každom signáli zahltenia sa zase multiplikatívne zmenší (AIMD). Signál zahltenia je daný stratou paketu, avšak nie každá strata paketu je označená ako signál zahltenia. Toto platí hlavne pre novšie implementácie TCP, napr. SACK viacero strát paketov počas jedného RTT označí ako jeden signál zahltenia.

Pred odvodením vzťahu prijmime zopár zjednodušujúcich predpokladov. Nech TCP beží nad stratovou linkou s konštatným RTT. Linka má dostatok priepustnosti nato, aby sa v smerovačoch neplnili výstupné fronty. Stratovosť linky namodelujeme jednoduchou

konštantnou pravdepodobnosťou *p* zahodenia paketu – predpokladá sa, že linka prenesie za sebou 1/p paketov a potom jeden zahodí. Predpokladajme, že prenos je riadený výlučne veľkosťou okna *cwnd*, a teda iba metódou predchádzaniu zahltenia. Za týchto predpokladov má veľkosť okna *cwnd* dokonalý periodický zubatý priebeh a jeden zub nazveme *cyklus*. Nech *W* je maximálna veľkosť okna, ktorú môžeme za daných okolností dosiahnuť (t.j. aditívnym zväčšovaním z hodnoty W/2 až po stratu paketu) meraná v paketoch. Potom zo správania sa protokolu po strate paketu vieme, že minimálna veľkosť okna musí byť *W/2* paketov. Nech prijímateľ posiela potvrdzovaciu správu ACK za každý prijatý segment - potom sa okno otvorí o jeden segment za čas RTT. Aby okno narástlo do veľkosti W z veľkosti W/2, uplynie čas W/2*RTT, čo je čas trvania jedného cyklu.



Dĺžka cyklu je určená parametrom *p*. Počet dát prenesených počas jedného cyklu je

$$\left(\frac{W}{2}\right)^2 + \frac{1}{2}\left(\frac{W}{2}\right)^2 = \frac{3}{8}W^2[paketov]$$

Podľa predpokladu každý cyklus prenesie *1/p* paketov, potom sa jeden stratí. Dostávame teda

$$\frac{1}{p} = \frac{3}{8}W^2$$

$$W = \sqrt{\frac{8}{3p}}$$

Pre priepustnosť platí symbolický vzťah

$$BW = \frac{(pocetDat\,\mathrm{Pr}\,enesenychZaCyklus)}{časTrvaniaCyklu}[bytov/s]$$

Pakety sa na množstvo dát konvertujú prenásobením konštantou MSS. Pre maximálnu priepustnosť dostávame teda vzťah

$$BW = \frac{MSS * \frac{3}{8}W^2}{RTT * \frac{W}{2}} = \frac{\frac{MSS}{p}}{RTT * \sqrt{\frac{2}{3p}}}$$

Po extrakcii konštánt do jedného výrazu dostávame

$$BW = \frac{MSS}{RTT}\frac{C}{\sqrt{p}}$$

Autori udávajú, že tento model a rovnica sú obmedzené a platia iba za určitých podmienkach (keď vysielanie protokolu riadi algoritmus predchádzanie zahlteniu). V praxi je však tento vzorec veľmi známy a ako autori ukazujú na simuláciách, má dostatočne širokú platnosť. Z uvedeného vzorca mimo iné vyplýva, že prenosovú rýchlosť TCP toku môžeme riadiť tromi parametrami:

- veľkosťou paketu
- zmenou RTT t.j.. zdržovaním paketov
- strátovosťou

2. Kontrola premávky

Smerovač je zdieľaným prostriedkom v sieti. Veľa problémov, s ktorými sa v sieťach stretávame je spojených s alokáciu a využívaním limitovaného množstva zdieľaných prostriedkov o ktoré súperia užívatelia siete, aplikácie a triedy služieb. Prostriedky, o ktoré sa delia sú:

- pridelenie miesta vo výstupnej fronte
- výstupná šírka pásma linky.

Predpokladajme model routerov ako *store-forward* (príjmi a pošli ďalej) mechanizmy s metódou *output queueing* (radenie do výstupných front). Podporované sú dve operácie: zaradenie príchodzieho paketu do fronty a vybratie paketu z fronty a poslatie na výstupnú linku. Dôvod prečo majú routre výstupné fronty (output queues) je ten, že takto sa absorbujú periodické a krátke zhlukovité výkyvy v premávke, kedy sú pakety dočasne uložené do front miesto toho, aby boli zahodené. Neskôr v čase menšieho zaťaženia siete sa výstupná fronta vyprázdni a pakety sú prenesené. Výhodou tohto prístupu je zníženie stratovosti paketov a možnosť existencie krátkych zhlukov. Nevýhody sú zvýšenie zdržania pri prenose a variácia zdržania (jitter).

Zahltenie smerovača nastáva v okamihu, keď paket príde na výstupný port rýchlejšie ako môže byť odvysielaný. Existujú v podstate dva druhy algoritmov v smerovačoch, ktoré riešia problém zahltenia

- QSD (queue scheduling disciplines) riadia veľkosť šírky prenosového pásma pridelené každej servisnej triede na výstupnom porte. QSD teda umožňuje kontrolu nad prístupom jednotlivých tried k obmedzenému sieťovému zdroju „**výstupná šírka pásma linky“.** Tento proces sa deje pri výbere paketu z fronty a jeho poslatí na výstupnú linku.

- QMM (queue memory management) kontroluje počet paketov vo výstupnej fronte. Kontrola sa deje pri operácii zaradenia príchodzieho paketu do fronty. Podľa určitých kritérií (napr. plnosť fronty) sa určí, či skutočne dôjde k zaradeniu paketu do fronty, alebo sa predpokladá zahltenie a dôjde k zahodeniu paketu. Týmto spôsobom QMM kontroluje prístup k ďalšiemu obmedzenému zdieľanému sieťovému prostriedku **„miesto vo výstupnej fronte“.**

2.1 Algoritmy QSD

Algoritmy QSD umožňujú riadenie prístupu ku konečnej výstupnej kapacite linky tým, že určí, ktorý paket vybrať z fronty pre ďalšie poslatie. Algoritmy sa líšia v komlexnosti, menežovateľnosti, presnosti a stupňom férovosti zdieľania linky. Implementácie algoritmov QSD v smerovačoch sú značne špecifické a neexistujú skoro žiadne štandardy – väčšinou sa zoberie nejaký štadartný algoritmus QSD, ktorý sa modifikuje pre konkrétne potreby.

Stanovíme zopár cieľov, ktoré by mal algoritmus QSD spĺňať. Podľa [10] sú to napríklad tieto

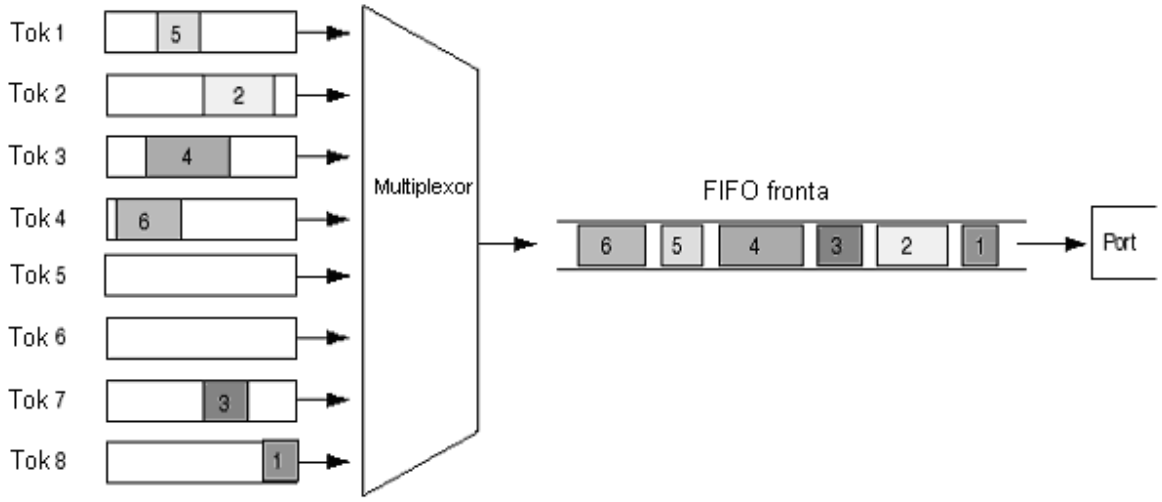
- podpora férovej distribúcie výstupnej kapacity linky medzi jednotlivé triedy poskytovaných služieb

- podpora poskytnutia časti výstupnej kapacity linky pre rôzne triedy
- ochrana a izolácia jednotlivých tried poskytovaných služieb. Ak sa vstupný tok dát v triede nechová tak, ako by mal (t.j. podľa dohodnutej triedy poskytovaných služieb), aby to nijako neovplyvňovalo ostatné triedy
- distibúcia nadbytočnej kapacity linky, t.j. ak trieda zrovna nevyužíva plnú kapacitu, ktorá je jej pridelená, aby si to ostatné triedy mohli medzi sebou nejako rozdeliť
- možnosť jednoduchej implementácie, aby pri vysokorýchlostných sieťach nedochádzalo pri posielaní paketov k veľkým oneskoreniam

Nasleduje zoznam tradičných algoritmov QSD ktoré bývajú súčasťou mnohých smerovačov a veľká časť z týchto algoritmov je dostupná aj v Linuxe. Kapitola zahŕňa základný popis a zoznam výhod a nevýhod, ktoré konkrétny algoritmus ponúka. Kapitola vychádza zo značnej časti z článkov [10], [11] a [12].

2.1.1 FIFO

FIFO je najzákladnejší agoritmus QSD. V radení FIFO sa ku každému príchodziemu paketu dostane rovnakej služby. Pakety sú posielané na výstupnú linku v takom poradí v akom prišli.



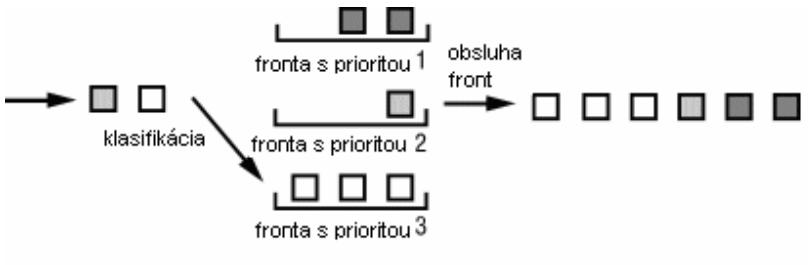
Výhody:
- nenáročné na výpočtový výkon
- predikovateľné chovanie sa FIFO fronty, nedochádza k preusporiadavaniu paketov a zdržanie závisí iba od veľkosti fronty
- zachováva zhlukovitý ráz vstupnej premávky, na rozdiel od ďalších disciplín, kde dochádza k určitému priemernému zdržaniu

Nevýhody:
- keďže sa so všetkými paketmi zachádza rovnako, neumožňje FIFO queueing definovať triedy služieb
- ak dochádza k zahlteniu, tak to ovplyvní všetky pakety rovnako (nemožno niektoré zvýhodniť), preto FIFO nie je vhodné pre real time alebo garantované služby

- pri zahltení víťazí pri zdieľaní linky UDP protokol nad TCP. Toto je spôsobené algoritmom na predchádzanie zahltenia protokolu TCP, ktorý pri zahltení zníži rýchlosť vysielania. UDP podobný mechanizmus nemá a tak ovládne linku
- FIFO zavádza predsudky voči zhlukovej premávke (napr. www), kedy náhly zhluk paketov z jedného zdroja môže zahltiť celú frontu a odstaviť tak ostatné toky od linky

2.1.2 Priority queueing (PQ)

PQ poskytuje to jeden z najjednoduchších mechanizmov ako zaviesť určitú úroveň tried poskytovaných služieb – teda ako diferencovať premávku. Je definovaných niekoľko výstupných FIFO front. Každá z nich má určitú prioritu, každá z priorít je rôzna. Platí, že z fronty s určitou prioritou môže byť vybraný paket na odoslanie iba, ak sú všetky fronty s vyššími prioritami prázdne. Fronty s vyššou prioritou majú teda absolútnu prednosť pred frontami s nižšou prioritou. Toto sa môže využiť napríklad v čase zahltenia siete. Do tried s najvyššou prioritou sa dá premávka, ktorá obhospodaruje a spravuje sieť a sieťové protokoly (napríklad routing protokoly) a dá sa tým zvýšiť stabilita siete. Trieda s najvyššou prioritou dostáva najvyšší stupeň kvality služieb, ktoré výstupná linka môže poskytnúť, t.j. najvyššiu prenosovú rýchlosť, najmenšiu strátovosť paketov, najnižšie zdržanie a variácie zdržania.



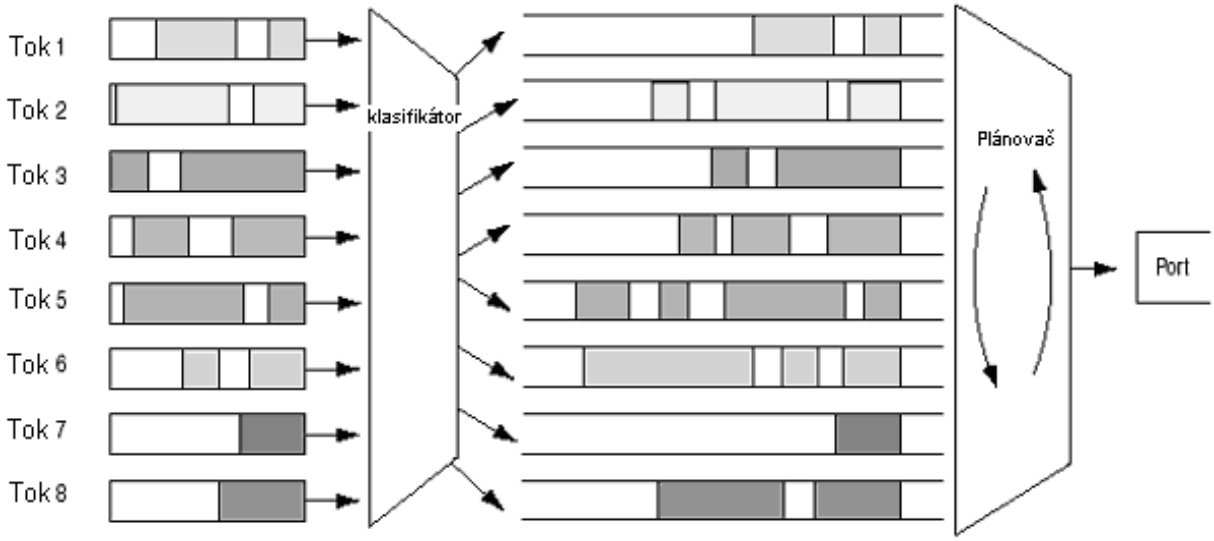
Výhody:
- nízka výpočtová náročnosť
- umožňuje určitý stupeň diferenciácie premávky. *Real time* sieťová premávka (typ premávky citlivej na oneskorenia) môže mať úplnú prednosť pred *bulk* sieťovou premávkou (typ premávky ktorá sa snaží preniesť dáta čo najväčšou rýchlosťou a nie je citlivá na oneskorenia)

Nevýhody:
- ak sa vstupný tok v triede s vysokou prioritou približuje ku kapacite výstupnej linky, alebo ju prekračuje, dochádza k výraznému oneskoreniu, neskôr až k úplnému odmietnutiu služby pre triedy s nižšou prioritou (DoS: denial of service). Tento problém sa zvykne riešiť napríklad policerom (viď kapitola 4), ktorý obmedzí rýchlosť prichádzajúcich paketov do triedy s vysokou prioritou (napr. zahadzovaním, alebo po prekročení určitej rýchlosti pakety pôvodne určené pre triedu s vysokou prioritou budú smerované do tried s nižšou prioritou). Navyše, ak je oneskorené posielanie dát z front s nižšou prioritou priveľké, vysielajúca strana toto môže chápať ako stratenie paketu a daný paket vyšle znova – týmto prispeje k zahlteniu siete a znižuje efektívnosť využitia svojej triedy poskytnutej služby.
- viac tokov klasifikovaných do triedy s rovnakou prioritou má rovnaké nevýhody akoby sa jednalo o jednoduché FIFO

- PQ nerieši problém férovosti medzi UDP a TCP. Ak by sme TCP dali do triedy s vyššou prioritou a neobmedzovali ho, tak TCP sa snaží využiť kapacitu linky na maximum a tak UDP premávka v triede s nižšou prioritou by *hladovala* (starvation)

2.1.3 Fair Queueing (FQ)

Mechanizmus FQ bol navrhnutý Johnom Naglom v roku 1987. Cieľom FQ bolo dosiahnuť nasledovnú *triedu poskytnutej služby*: „každý tok má rovnaký podiel na výstupnej šírke prenosového pásma linky“. Pri vstupe paketu do FQ je tento klasifikovaný do toku (napríklad ako tok môžeme v TCP označiť pakety s rovnakými štvoricami: zdrojová adresa, zdrojový port, cieľová adresa, cieľový port). Každý tok má pridelenú FIFO frontu. Na výstupe sú FIFO fronty obsluhované cyklicky round robin algoritmom (RR). RR je postavený na paketoch, t.j. každá FIFO fronta v každom kole môže poslať jeden paket. FQ sa zvykne označovať aj ako radenia paketov založené na tokoch.



Výhody:
- extrémne zhlukové toky, alebo toky ktoré sa chovajú zle, nemajú žiaden vplyv na kvalitu služby poskytovanú ostatným tokom, pretože toky sú od seba izolované.

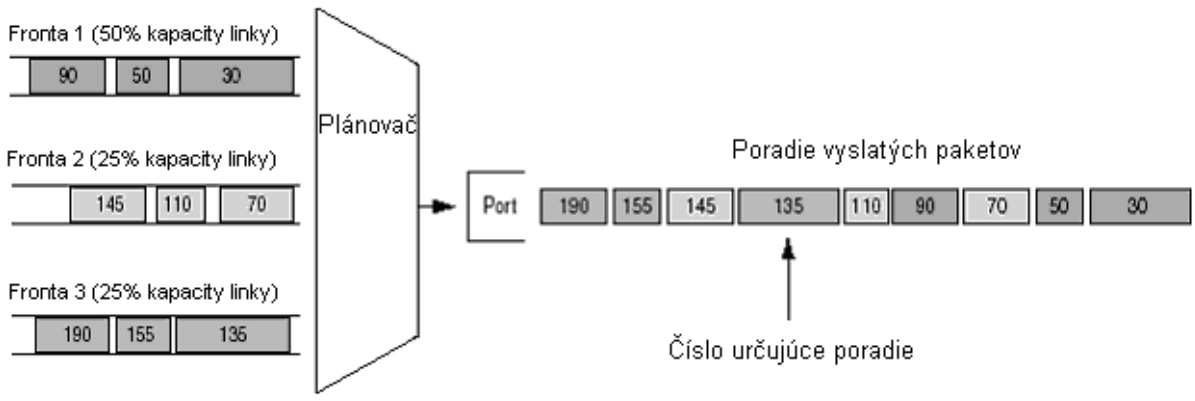
Nevýhody:
- toky sú obsluhované rovnako, to neumožňuje zvýhodniť žiaden tok pred iným
- problémy sú s tým, že z každého FIFO sa neposielajú byty ale pakety, tie majú rôznu veľkosť a tak je problém s tým, že toky s veľkými paketmi zoberú väčšiu časť kapacity výstupnej linky a sú týmto zvýhodnené
- veľkosť zdržania sa paketu v smerovači s FQ závisí od toho, koľko tokov prechádza cez výstupnú linku. Ak paket príde do prázdneho FIFO queue, ktoré zrovna zmeškalo svoje kolo, tak musí čakať dosť dlho, až sa na neho dostane rada. Takto je nemožné nijako ohraničiť zdržanie sa paketu, čo znamená, že tento algoritmus nie je vhodný na garantovanú premávku a real time premávku
- FQ predpokladá, že vieme efektívne klasifikovať premávku do tokov. Toky sa dajú aj zneužiť, keď si aplikácia alebo užívateľ otvorí viacej tokov, tak dostane dokopy väčšiu časť výstupnej kapacity linky

- cez smerovače na Internete prechádzajú tisícky, až desaťtisícky spojení – pre každé spojenie manažovať FIFO je neúnosné a spomaľujúce a ťažké na efektívnu implementáciu

2.1.4 Weighted fair queueing (WFQ)

WFQ bol navrhnutý v roku 1989. WFQ je základom pre skupinu algoritmov QSD, ktoré majú odstrániť nedostatky FQ modelu. Miesto toho, aby každý tok dostal svoju vlastnú frontu, definujú sa tzv. triedy. Je definovaný pevný počet tried. Paket sa do triedy dostane po klasifikácii, podľa určitých kritérií. Každá trieda v sebe obsahuje FIFO frontu. Triedy dostávajú rôzny podiel výstupnej kapacity linky tým, že sa im priďeľujú váhy. WFQ rieši už aj otázku rôznej dĺžky paketov v rozdielnych FIFO frontách. Z tried vyberá pakety plánovač a dáva ich na výstupnú linku.
Implementácia plánovača, ktorý sa používa, je aproximáciu teoretického modelu GPS (general processor sharing). Pokiaľ GPS je teoretický model, ktorý sa v praxi nedá implementovať, jeho správanie je podobné technike bit-by-bit WRR (weighted round robin – viď 2.1.5).
Táto technika by sa dala popísať nasledovne: Každá FIFO fronta(trieda) má priradenú váhu, alebo časť z výstupnej kapacity linky. V jednom kole WRR každý queue pošle toľko bitov ako je jeho váha. Keďže v sieti IP sa neposielajú bity ale pakety, tak existuje ešte entita Packet Reassemebler, ktorá z prúdu bitov spraví pakety. Ak packet reassembler narazí na posledný bit z paketu, tak daný paket sa pošle.

V reálnych implementáciách WFQ sa pakety nerozbíjajú na bity. Každý došlý paket sa klasifikuje a uloží do fronty (triedy). Pre danú výstupnú rýchlosť linky, dané veľkosti všetkých paketov vo všetkých frontách, počet aktívnych front a relatívnych váh priradených každej fronte sa pre došlý paket vypočíta čas, do ktorého by mal byť poslaný. Paket s najnižším časom sa potom pošle. Vypočítaný čas nie je čas poslatia paketu, je to len číslo, ktoré hovorí o poradí vysielania paketov vo frontách.



Výhody:
- izolácia jednotlivých tokov od ostatných,
- grancia časti výstupnej prenosovej rýchlosti

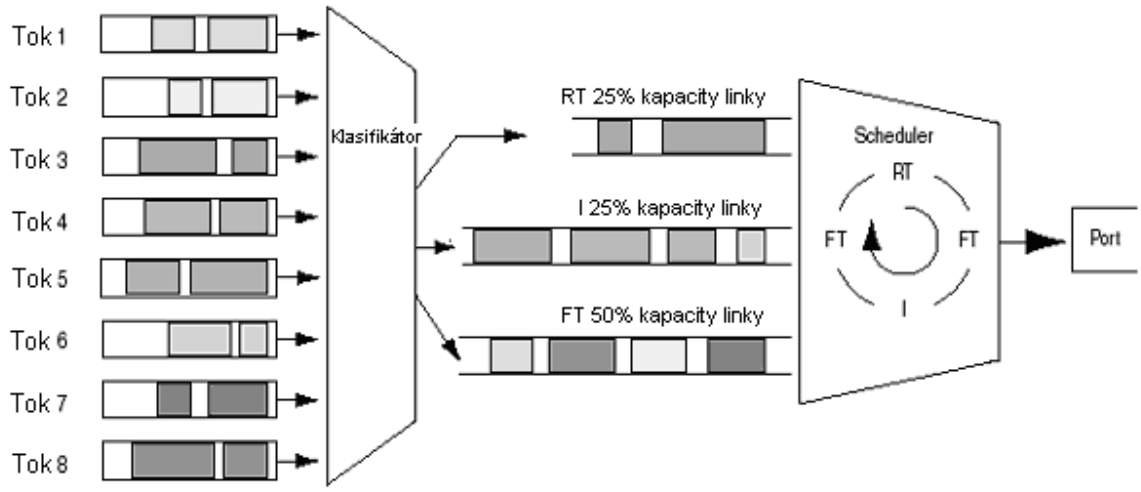
Nevýhody:
- výpočet a zoradenie paketov podľa časov je výpočtovo náročné

- viacero tokov patriacich rovnakej triede má nevýhody ako FIFO
- aj keď oproti FQ sa zdržanie paketu dá garantovať (je definovaných iba konečný počet tried), stále môže byť veľký a nevhodný pre real time služby

2.1.5 Weighted round robin (WRR)

WRR je základ pre skupinu algoritmov QSD, ktoré majú za úlohu odstrániť nevýhody FQ a PQ. Konkrétne odstraňuje nasledovné obmedzenia:
- tokom môže byť pridelená rôzna kapacita linky (na rozdiel od FQ)
- každý tok v jednom kole WRR odošle aspoň jeden paket čím sa prechádza starvation v PQ.

Princíp je nasledovný: Príchodzie pakety sa klasifikujú do konečného počtu tried, pričom každá z nich obsahuje v sebe FIFO frontu. Trieda má poskytnúť určitú úroveň služby. WRR definuje tzv. servisné kolo v ktorom obslúži každú frontu niekoľkokrát. Pri obsluhe fronty sa z fronty odoberie práve jeden paket. Ak je fronta prázdna, tak sa neobsluhuje. Hrubé rozdelenie výstupnej prenosovej kapacity linky je dosahované tým, že pre každú triedu je určené koľko krát sa z nej odošle paket v jednom servisnom kole. Zaujímavý je aj fakt, že pri reálnych implementáciách sa pri zaraďovaní paketov do FIFO front zväčša používa RED technika (viď sekcia. 2.2.2).



Výhody:
- jednoduchá implementácia predurčuje WRR na vysokorýchlostné siete
- v každom servisnom kole sa z každej FIFO queue oddoberie aspoň jeden paket, čím sa prechádza vyhladoveniu (starvation) jednotlivých tried

Nevýhody:
- iba hrubé rozdelenie výstupnej prenososvej kapacity linky, závislé na velkosti paketov. Táto technika je viacej vhodná pre siete s fixnou dĺžkou paketov ako je napr ATM.

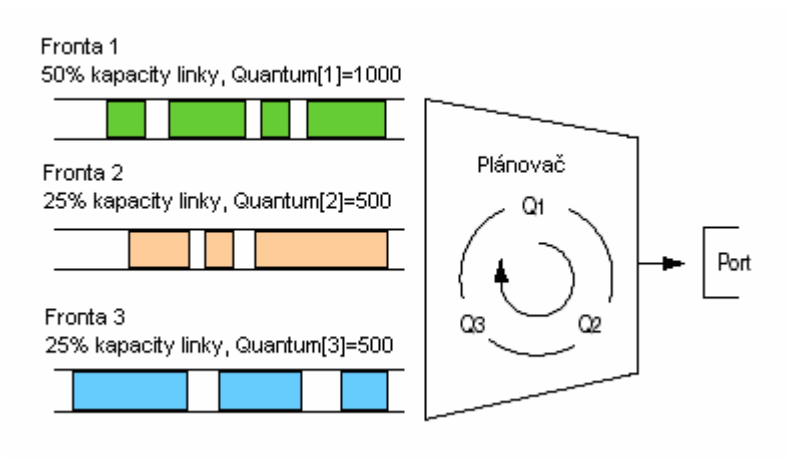
2.1.6 Deficit Weighted round robin (DWRR)

DWRR má odstrániť nedostatky WRR aj WFQ. Bol navrhnutý v roku 1995. Rieši nedostatok WRR, ktorý nerozdeľuje linku tak ako by mal pri rôznej veľkosti paketov. Ďalej rieši problém WFQ, ktorý používa príliš komplexný algoritmus na výpočet ktorý paket má byť poslatý ako ďalší. Je definovaných niekolko tried služieb, tie majú vlastné FIFO fronty. Pre každú triedu sú definované nasledovné parametre:

Váha[F] – pre každú triedu (frontu F) definované percento z výstupnej kapacity linky
DeficitCounter[F] – premenná v ktorej sa pre každú triedu zapamätá, koľko bytov z nej v jednom kole DWRR bolo poslatých
Quantum[F] – je max počet bytov, ktoré môže daná FIFO fronta vyslať v jenom kole DWRR. Kvantum je odvodené od váhy pridelenej triede.

Pri príchode paketu je tento klasifikovaný a zaradený do príslušnej fronty. Na výstupe beží DWRR, ktorý v jednom kole obslúži postupne všetky fronty. Každá fronta F je obslúžená nasledovne:

1) „nabije“ sa DeficitCounter nasledovne
   DeficitCounter[F] = DeficitCounter[F] + Quantum[F]
2) z danej triedy F pošli toľko paketov, aby suma ich dĺžok bola nanajvýš *DeficitCounter[F]*. Nech je to *B* bytov.
3) *DeficitCounter[F] = DeficitCounter[F] – B*
4) Obslúž ďalšiu frontu F

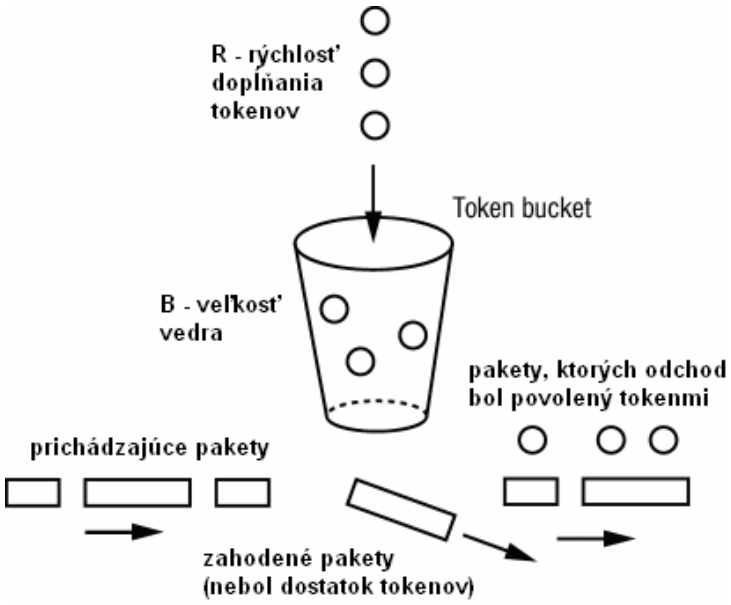


Výhody:
- jednotlivé triedy sa medzi sebou neovplyvňujú (izolácia tried)
- presná kontrola zdieľania linky
- algoritmus je jednoduchý a výpočtovo nenáročný, je možnoť nasadiť ho aj na vysokorýchlostné linky

Nevýhody:
- viacero tokov v jednej triede má nevýhody FIFO
- nemožnosť garancie oneskorenia (aj keď je možné garantovať hornú hranicu závisiacu od počtu tried)

2.1.7 Token bucket TB (vedro)

TB je ďalšou metódou na obsluhu fronty a zároveň spôsob ako tvarovať dátový tok (traffic shaping). Idea metódy je jednoduchá. Máme vedro obsahujúce v každom časovom okamihu určitý počet tokenov, pričom každý z nich predstavuje povolenie k odoslatiu alebo inému spracovaniu určitého objemu dát. Na začiatku je nádoba plná. Pri príchode každého paketu sa overí, či je počet tokenov v nádobe dostatočný veľkosti paketu. Ak áno, paket sa zaradí do fronty na odoslatie na výstupné zariadenie a z vedra sa odoberie odpovedajúci počet tokenov. Ak nie, tak je paket zahodený alebo pozdržaný, kým v nádobe nebude dostatok tokenov. TB je charakterizovaný dvoma parametrami *R* a *B*. Parameter *R* predstavuje rýchlosť dopĺňania tokenov a *B* maximálnu kapacitu vedra. Situácia je zobrazená aj na nasledovnom obrázku.



2.2 Active queue memory management

Uvažujme problém zaradenia paketu do fronty, ktorá patrí výstupnej linke. Pri zahltení linky sa fronta začína plniť. Fronty majú obmedzenú kapacitu a preto sa pri vyššej rýchlosti príchodu paketov, ako je rýchlosť výstupnej linky musí naplniť. Otázkou zostáva, či zahadzovať pakety až keď je fronta plná a príchodzí paket nie je kam uložiť, alebo robiť proaktívnu politiku a zahadzovať pakety ešte pred naplnením fronty a či to má nejaký význam. Výber, ktorý paket bude zahodený a ako, má vplyv na prenosové charakteristiky TCP spojenia a na riešenie zahltenia. Prepokladá sa, že cez router prechádza na výstupnú linku veľa TCP spojení. Existuje viacero techník, ktoré sa rôznym spôsobom stavajú k uvedenému problému. Tieto techniky sa zvyknú označovať ako AQMM (active queue memory management) aktívny prístup k problematike manažovania zdieľanej pamäte. V kapitole bude uvedených niekoľko techník, ktoré sú dostupné v Linuxe, simulátoroch (napr. ns2) a v rôznych smerovačoch.

2.2.1 Tail drop

Jedná sa o najjednoduchšie zahadzovanie paketov, ktoré sa vlastne nedá ani označiť ako AQMM. Fronta sa plní FIFO princípom, a ak je výstupná fronta pre danú výstupnú linku plná a príde ďalší paket, tak tento je zahodený. Koncovému užívateľovi sa nezvykne o tom poslať žiadna správa a je na ňom, aby stratu paketu detekoval. Paket nie je sieťovým prvkom neskôr ani preposlaný, predpokladá sa, že tieto problémy si vyriešia koncové stanice.

Výhody
- jednoduché na implementáciu

Nevýhody
- tail drop zahadzuje pakety až keď je fronta úplne plná, t.j. sú vyčerpané všetky sieťové prostriedky (fronta aj prenosová kapacita). Toto okrem iného aj znamená, že sieťový prvok v danom momente nie je schopný akceptovať už žiadne zhlukové výkyvy, až pokým sa fronta neuvoľní – takto sa stratí hlavný účel fronty. Plná fronta blokuje ďalšie príchodzie pakety v prenose, ktoré sú následne zahadzované. Takto môže malý počet spojení úplne ovládnuť frontu a zamedziť ďalším spojeniam v prenose paketov
- fronta môže byť skoro plná aj dlhý čas (sieťové prostriedky sú značne vyčerpané – prenosová kapacita úplne, miesto vo fronte čiastočne). Takýto stav však koncové stanice neinterpretujú ako zahltenie až do chvíľe, keď dôjde k preplneniu fronty a zahadzovaniu paketov t.j. sieťové prostriedky sú už úplne vyčerpané.
- tail drop je veľmi nevýhodná technika pre premávku založenú na protokole TCP. TCP stratu paketov berie ako signál, že na sieti je zahltenie a zníži rýchlosť vysielania. Lenže tail drop spôsobí, že všetky TCP spojenia prechádzajúce daným routerom znížia svoju rýchlosť, keďže je fronta plná a pakety sa im zahodia. Toto vedie k fenoménu nazvanému *globálna TCP synchronizácia*, kedy rýchlosť premávky osciluje medzi preplnenými FIFO frontami a využitím plnej kapacity výstupných liniek a medzi prázdnymi frontami a využitím iba zlomku kapacity výstupných liniek. Priemerné využitie skutočnej kapacity výstupných liniek sa týmto drasticky znižuje.

- TCP sa zo straty jedného paketu spamätá rýchlejšie, ako zo straty niekoľkých paketov. Preto ak zahodíme pre dané TCP spojenie viacero paketov, vedie to k výraznej degradácii prenosovej rýchlosti daného toku.
- dlhé fronty vedú k zvýšeniu zdržania paketov v sieti
- zahadzovanie paketov plytvá sieťovými prostriedkami, keďže daný paket sa musí preposlať znova

2.2.2 Random early detection (RED)

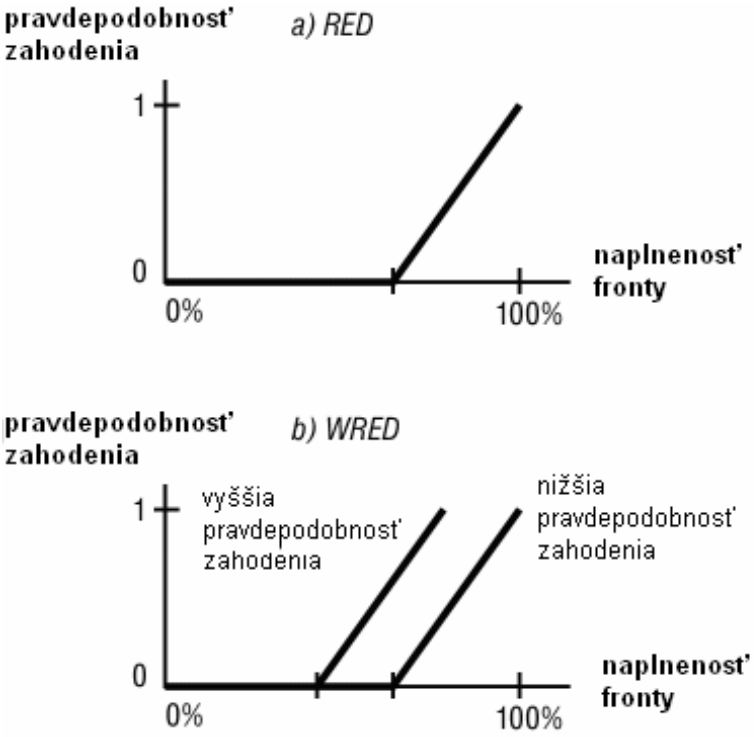
Tail drop bola najjednoduchšia technika ako pristúpiť k manažmentu naplnenia výstupnej fronty. Tejto technike však chýbajú akékoľvek náznaky AQMM. AQMM umožňuje smerovačom proaktívne reagovať na možnosť zahltenia pozorovaním napĺňania sa výstupnej fronty. Namiesto čakania na úplné naplnenie a potom zahadzovanie všetkých nasledovných došlých paketov, AQMM reaguje na zahltenie buďto zahadzovaním alebo označkovaním paketov ešte pred preplnením fronty. Týmto sa sledujú nasledovné ciele:

- eliminácia fenoménu globálnej synchronizácie TCP spojení
- zachovanie podpory krátkych zhlukových výkyvov premávky, kedy fronta tieto výkyvy absorbuje, avšak proaktívne sa koncovým hostom naznačí aby znížili rýchlosť vysielania
- kontrola priemernej dĺžky fronty umožňuje znížiť zdržanie paketov prechádzajúcich daným smerovačom

RED je technika z AQMM, ktorá je v súčasnosti nasadená vo veľkých IP sieťach a na backbone smerovačoch. Zahodením jediného paketu pošle router pre TCP spojenie signál, že nastalo zahltenie. Táto technika by mala pracovať v spolupráci s algoritmami kontroly toku technikou predchádzanie zahlteniu ktorá je súčasťou protokolu TCP. TCP reaguje na stratu paketu znížením rýchlosti vysielania. Týmto sa predíde k preplneniu výstupnej fronty.

RED používa tzv. *packet drop profil*, čo je vlastne krivka, ktorá kontroluje agresivitu zahadzovania jednotlivých paketov. Krivka vyjadruje závislosť medzi pravdepodobnosťou zahodenia príchodzieho paketu a naplnením fronty. Ak dosiahne naplnenie fronty určitú medz, začne smerovač zahadzovať pakety náhodne vybraných TCP spojení ešte pred tým, ako dôjde k zahlteniu. Pravdepodobnosť zahodenia paketu sa zvyšuje naplnením fronty (viď obrázok). Tým dôjde k zníženiu objemu vysielaných dát od niektorých odosielateľov a k plynulému vyrovnaniu celkového objemu prichádzajúcich dát s kapacitou výstupnej linky.

WRED (weighted RED) je rozšírenie RED o viacero tried služieb. Pakety sa určitým spôsobom pri príchode klasifikujú do tried. Každá trieda má nastavený nejaký profil (agresívnejšie alebo menej agresívne zahadzovanie) podľa kvality služby ktorú chceme zabezpečiť.



Výhody:

- zavedenie RED nevyžaduje modifikáciu TCP protokolu
- prístup k riešeniu zahltenia je proaktívny a nemalo by nikdy dôjsť k úplnému zaplneniu fronty a následnému tail drop zahadzovaniu
- zachováva sa podpora zhlukovej premávky, keďže ide o FIFO frontu. Pakety sú vysielané v tom poradí v akom prišli. Nevýhodou môže byť, že niektoré pakety zo zhlukovej premávky budú zahodené.
- RED podporuje TCP, lebo nezahadzuje zhluky paketov z jediného TCP toku v dôsledku preplnenia
- RED umožňuje držať naplnenosť fronty pod určitou úrovňou a pomáha k lepšiemu využitou prenosovej kapacity výstupnej linky. Množstvo paketov držaných vo fronte nie je ani príliš malé, čo by mohlo viesť k poddimenzovaniu výstupnej kapacity, a ani sa neblíži k plnej kapacite, keď by sme museli zahodiť všetky príchodzie pakety z množstva TCP spojení, ktoré by následne znížili rýchlosť.
- RED podporuje teoreticky férové zahadzovanie paketov medzi jednotlivými TCP spojeniami - a nemusí si držať stavovú informáciu pre každé spojenia.

Nevýhody:

- RED môže byť dosť ťažké nastaviť, tak aby dával očakávané výsledky
- RED pracuje dobre iba s TCP protokolom a s jeho algoritmami na predchádzanie zahlteniu. Iné protokoly ako napr. UDP zahadzovanie paketov neberie ako signál pre zníženie rýchlosti
- zahadzovanie paketov nie je veľmi efektívny signál oznamujúci zahltenie. Plytvajú sa pri ňom sieťové prostriedky, keďže paket musí byť preposlaný znova. Uvedený problém rieši napríklad ECN, ktorý však vyžaduje modifikáciu protokolu. Bližšie je špecifikovaný napr v RFC 2481
- jeden z najväčších problémov TCP algoritmu „predchádzanie zahlteniu“ keď sa používa pri tail drop routeroch je ten, že TCP zníži rýchlosť vysielania až vtedy keď detekuje stratu paketov – to značí až keď sa queues v routeroch začínajú prepĺňať a router začína

strácať pakety. Medzi zahodením paketu na routeri a zistením tejto skutočnosti u vysielacieho konca spojenia môže uplynúť značné množstvo času. Takto smerovač zahodí veľké množstvo paketov ešte pred tým ako si vysielajúca strana uvedomí zahltenie siete a zníži rýchlosť vysielania. Hoci RED rieši tento problém proaktívne, t.j. pri vznikajúcom zárodku zahltenia, znevýhodní niektoré toky a predchádza tým preplneniu fronty. Na to aby bola metóda RED efektívna však treba mať dostatočne dlhú frontu a treba mať dobre nastavené pravdepodobnosti spúštajúce skoré zahadzovanie paketov – aby nedochádzalo k poddimenzovaniu linky. Nanešťastie, ak cez daný router prechádza veľké množstvo spojení, tak výsledný tvar toku má veľmi zhlukovitý a výkyvový charakter. Toto môže poraziť aj proaktívnu politiku RED a môže taktiež dôjsť k striedaniu vĺn predimenzovania a poddimenzovania prenosovej kapacity výstupnej linky.

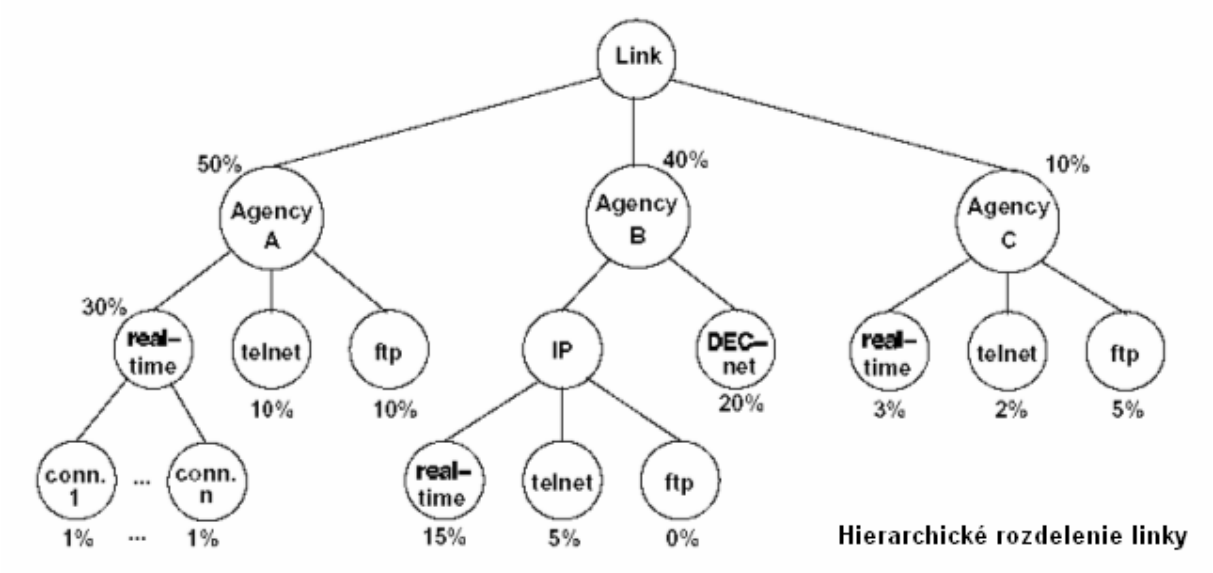
Existujú aj ďalšie algoritmy AQMM. Sú to napríklad BLUE [13], REM [14], AVQ [15]. Bližšie sa im venovať nebudeme.

2.3 Hierarchické zdieľanie linky

Hierarchický model zdieľania linky je mechanizmus, ktorý umožňuje zdieľať linku pre rôzne organizácie, rodiny protokolov a typy premávky. Tento mechanizmus je vhodný na riadenie premávky v čase zahltenia výstupnej linky na nejakom smerovači ako aj na splnenie prirodzených požiadaviek na zdieľanie prenosovej kapacity výstupnej linky. Najmä zaujímavý a prirodzený spôsob zdieľania linky z neho robí dobrý nástroj pre administrátorov siete a poskytovateľov pripojenia.

2.4 Class based queueing (CBQ)

CBQ je hierarchický model zdieľania linky, kde sa definuje hierarchia tried, pričom každej triede je možné prideliť časť výstupnej kapacity linky a prioritu. Hierarchický model zdieľania linky CBQ je popísaný v [17]. Musí platiť, že suma kapacity všetkých detí sa rovná kapacite pridelenej rodičovi. Koreňová trieda by mala dostať 100% kapacity linky. Táto hierarchia sa pridelí výstupnej linke. Pri vstupe paketu na výstupnú linku je tento klasifikovaný do **listovej** triedy kde čaká na vyslanie. Nelistové triedy slúžia na rozdelenie nadbytočnej/nevyužitej kapacity. Táto hierarchia sa zvykne označovať ako link sharing hierarchy.
Prirodzene sa interpretujú kapacity pridelené jednotlivým triedam. Celú linku rozdelíme medzi firmy A,B,C a každej firme garantujeme danú šírku prenosového pásma linky. V rámci firiem tiež garantujeme určitú časť siete pre real time premávku, telnet a ftp a v rámci týchto tried zase kapacitu pre jednotlivé spojenia ako je uvedené na obrázku.



Hierarchické rozdelenie linky

2.4.1 Ciele hierarchického zdieľania linky CBQ

Autori si pri návrhu hierarchického zdieľania linky stanovili nasledovné dva ciele:

Cieľ 1
Každá trieda, ktorá má dostatočný záujem (dáta do nej prichádzajú dostatočnou rýchlosťou) musí dostať zhruba svoju *garantovanú časť výstupnej kapacity linky* za jednotku času. Či trieda dostala dostatočný zlomok kapacity sa meria za časovú konštatnu K, ktorá je jedným z parametrov CBQ. Ako autori uvádzajú, nie je nutné aby toto platilo pre všetky možné veľkosti časovej konštanty. Keď je však konštanta dostatočne malá, dosahujú sa lepšie výsledky. Ak by bola konštanta rádovo v minutách, môže sa stať, že trieda aj na pár minút neposiela pakety a potom ich pošle naraz – čo je nežiadúce. V priemere za konštantu K by však svoju garantovanú kapacitu dostala.
Priorita danej triedy pri získaní garantovanej časti výstupnej kapacity linky nezohráva až takú významnú rolu. Triedy s vyššou prioritou sú síce obslúžené prednostne pred ostatnými triedami, ale len do úrovne dosiahnutia garantovanej prenosovej rýchlosti. Potom môžu byť na istý čas odmlčané. Triedy s vyššou prioritou majú však všeobecne lepší reakčný čas, lebo sú obsluhované prednostne.

Cieľ 2
Druhým cieľom je *distribúcia nadbytočnej kapacity linky*. Ak nejaká trieda nevyužíva dostatočne ani svoju garantovanú kapacitu, potom ostatné triedy, ak majú záujem, tak si tento zvyšok nejako rozumne rozdelia. Čo to znamená rozumne autori nešpecifikujú a hovoria že to záleží na implementácií.
Obvykle sa nadbytočná kapacita rozdelí podľa myšlienky požičiavania si kapacity. Trieda s dostatočným záujmom si požičia kapacitu od svojich predkov v hierarchii. Požičiavanie je rovnomerne rozdelené podľa váhy pridelenej jednotlivým synom danej triedy. Pri rozdeľovaní nadbytočnej kapacity zohráva významnú rolu priorita, keďže triedy s vyššu prioritou dostanú nadbytočnú kapacitu ako prvé. V rámci skupiny tried s rovnakou prioritou dôjde k proporcionálnemu rozdeleniu kapacity podľa váh. Ak napríklad trieda s vysokou prioritou má veľký dopyt, tak môže zobrať celú nadbytočnú kapacitu pre seba a triede s menšou prioritou sa ujde len garantovaná rýchlosť.

2.4.2 Základné pojmy

Autori definovali na konceptuálnej úrovni pojmy, ktoré sú dôležité na pochopenie implementácie ako aj spôsobu fungovania CBQ. Niektoré pojmy sa nebudeme snažiť prekladať a budeme používať ich anglické ekvivalenty.

strom zdieľania linky – každej výstupnej linke je pridelená stromová štruktúra tried, ktorá popisuje hierarchické rozdelenie linky

trieda – je uzlom stromu zdieľania linky

listová trieda – je to trieda (uzol) bez potomkov

plná listová trieda – je listová trieda, ktorá obsahuje v sebe nenulový počet paketov určených na poslanie

vnútorná trieda – je trieda ktorá nie je listová

Link sharing guidelines (LSG) – súbor pravidiel podľa ktorých by mali ísť pakety čakajúce v listových triedach na výstupnú linku.

Link sharing behaviour (LSB) – spôsob akým boli v nedávnej dobe posielané pakety z listových tried.

regulovaná, neregulovaná trieda – ak pakety do triedy prúdia vyššou rýchlosťou ako je rýchlosť, ktorú povoľujú v danom okamihu (závisí od iných tried) pravidlá LSG, tak daná trieda musí mať regulovanú výstupnú rýchlosť (t.j. bude shapovaná), aby naďalej spĺňala LSG.

klasifikátor, estimátor – klasifikátor je entita, ktorá príchodzie pakety klasifikuje do tried hierarchie. Estimátor je entita, ktorá meria/odhaduje akú časť kapacity linky použila za časovú konštantu K (spomínaná vyššie).

nadlimitná, podlimitná, limitná trieda – každá trieda v každom časovom okamihu môže byť označená práve jednou z týchto troch značiek. Trieda je podlimitná, ak nedávno použila menej prenosovej rýchlosti ako je jej garantovaná hodnota. Trieda je limitná, ak nedávno pužila práve toľko prenosovej rýchlosti ako je jej garantovaná hodnota, ináč je nadlimitná. O priradení tejto značky určitej triede rozhoduje estimátor.

spokojná, nespokojná trieda – každá trieda v každom časovom okamihu môže byť navyše označená aj touto značkou. Listová trieda je nespokojná (s LSB), ak v nej čakajú pakety na odoslanie (angl. has persistent backlog) a je podlimitná. Takáto trieda zatiaľ nedostala ani svoj garantovaný tok a keďže v nej čakajú pakety na odoslanie, má právo byť obslúžená – preto je nespokojná s nedávnym správaním sa LSB. Ináč listovú triedu označíme ako spokojnú.
Nelistová trieda je nespokojná ak je ona sama podlimitná a má aspoň jedného potomka, ktorý má plnú frontu. Ináč je spokojná.
POZN: Pre úplnosť možno ešte dodať, že trieda, ktorá je nadlimitná a plná, nemá právo byť nespokojná, lebo nedávno dostala aspoň svoj garantovaný tok. To že obsahuje v sebe pakety vyplýva z toho, že dáta do nej tečú vyššou rýchlosťou ako je jej garantovaná rýchlosť.

level triedy – Pre každú triedu induktívne definujeme premennú level takto:
- Listy majú level 1.
- Nelistové triedy majú level = max{level všetkých detí}+1

exemp, bounded, isolated triedy
Do hierarchie sa zavádzajú ešte ďalšie pojmy. Ak je trieda *exemp*, značí to, že nikdy nebude regulovaná. Teoreticky, ak by sme cez ňu posielali pakety rýchlosťou výstupnej linky, tak vyblokujeme iné triedy. Predpokladá sa, že takáto trieda je na podporu real time služieb a že množstvo údajov, ktoré do nej tečie je inak regulované, aby nezabrala celú linku (napríklad policer-om na hranici siete).
Ak je trieda *bounded*, tak má zakázané si požičiavať nevyužitú kapacitu od svojich predkov. To značí, že maximálnu rýchlosť, ktorú môže dosiahnuť, je iba jej garantovaná hodnota.

Ak je trieda *izolovaná*, tak neumožňuje potomokom, aby si z nej požičali jej nevyužitú kapacitu. Na druhú stranu si však ani ona nemôže požičať od svojho predka. Táto trieda môže byť vyňatá zo stromu a jednoducho jej bude pridelený iba zlomok kapacity linky.

2.4.3 Pravidlá posielania paketov (LSG)

LSG je súbor pravidiel, ktoré autori navrhli na dosiahnutie dvoch vytýčených cieľov v CBQ. V podstate sa pravidlá LSG skladajú z dvoch častí. Prvá časť hovorí o tom, že trieda má dostať svoj garantovaný tok (Cieľ 1) a druhé pravidlo hovorí o tom, ako sa rozdelí nadbytočná nevyužitá kapacita linky (Cieľ 2).

Situácia je o to komplikovanejšia, že autori navrhujú tri sady pravidiel LSG:
- formálne LSG
- ancestor only LSG
- top level LSG

Formálne LSG su pravidlá, ktoré presne dosahujú oba ciele. Na praktickú implementáciu sú však výpočtovo náročné a preto autori navrhli aj dve aproximácie. Tie majú nižšiu výpočtovú náročnosť, pri plnení horeuvedených cieľov sú zaťažené určitou chybou. V texte spomíname iba *formálne LSG* a okrajovo *top level LSG* a *ancestor-only LSG*, pre zvedavého čitateľa odporúčame preštudovať publikáciu [17]. V Linuxe je implementovaná top level LSG aproximácia.

Formálne LSG
Trieda C môže pokračovať v posielaní paketov neregulovane, pokiaľ spĺňa aspoň jedno z nasledovných:
1) je podlimitná alebo limitná (t.j. nie je nadlimitná)
2) je nadlimitná. Potom však musí platiť nasledovné: Trieda má podlimitného alebo limitného predka P, (v ľubovolnej výške smerom ku koreňu v hierarchii) ktorý je na leveli i. Nesmie však v hierarchii existovať žiadna trieda, ktorá je nespokojná a má level ostro menší ako i. (pozn: táto trieda nemusí byť potomok P) Ak nájdeme takéhoto predka P, tak trieda C si od neho požičia ďalší zlomok kapacity. Takýto predok však požičiava, iba ak sú v nižších leveloch všetci spokojní. Ináč oni majú právo byť obslúžení.

Ináč bude trieda regulovaná.

Pozn1: Pravidlo 2 sa dá formulovať aj nasledovným spôsobom
Nech X je level najvyššieho nespokojného uzlu v hierarchii (najvyšší znamená najbližšie ku koreňu). Potom si nikto nemôže požičať z levelu > X (teda trieda C si môže požičať od rodiča ktorý je maximálne na leveli <= X)

Pozn2: Druhé pravidlo zabezpečuje, že nedôjde k požičiavaniu kapacity linky, ak existuje taký list, ktorý môže poslať bez požičiavania (t.j. je podlimitný a obsahuje v sebe pakety na odoslanie – a je tým pádom nespokojný)

Top-level LSG
Pri tejto aproximácii sa zavádza globálna premenná *top-level*, ktorá hovorí o tom z akého maximálneho levelu je triede povolené si požičiavať. Ak *top-level* nastavíme na nekonečno, tak dostaneme ancestor only LSG. Ak sa *top-level* vždy nastaví na najnižší level, ktorý má

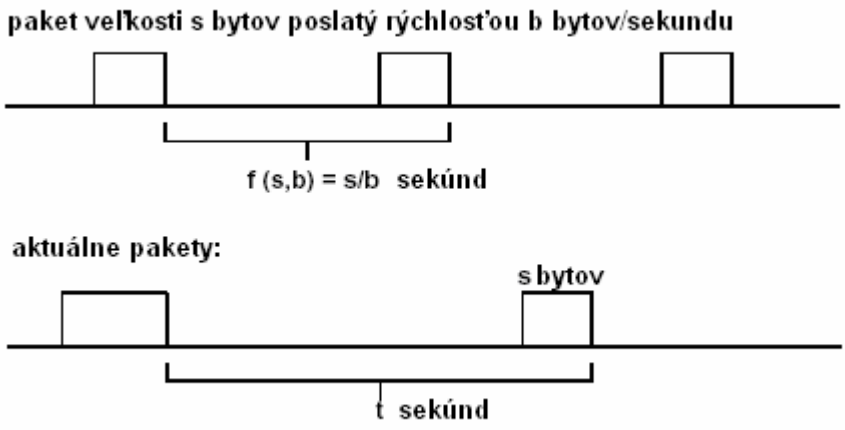
najnižšia nespokojná trieda, tak dostávame fromálne LSG. Ak nastavíme *top-level* na konštantnú hodnotu rovnú jednej, tak iba podlimitné triedy budú môcť posielať pakety a nikto si nebude môcť požičiavať.

Trieda C môže pokračovať bez regulácie ak (platí aspoň jedno):
- trieda nie je nadlimitná
- trieda je nadlimitná a má podlimitného predchodcu, ktorého level je najviac *top-level*

Ináč je trieda regulovaná.

2.4.4 Navrh implementácie estimátora

Úlohou estimátora je určiť limitný status každej triedy. Jeho parametrami sú časová konštanta K a frekvencia F prepočtu limitného stavu triedy. Limitný status triedy je určený meraním medzipaketovej medzery t.j. času *t* ktorý uplynie medzi dvomi za sebou odoslatými paketmi z danej triedy. Táto hodnota sa potom porovnáva s požadovanými hodnotami *f(s,b)=s/b* vypočítanými z garantovanej rýchlosti triedy. V skutočnosti sa namerané hodnoty ešte priemerujú filtrom EWMA (exponencial weighted moving averages), ale to nie je až také podstatné. Situácia je znázornená na nasledovnom obrázku.



2.5 Hierarchical Token bucket (HTB)

Metóda Hierarchical Token Bucket vychádza z modelu hierarchického zdieľania linky. V roku 2003 ju implementoval a zaslúžil sa o jej zaradenie do štandartného Linuxového jadra (od ver 2.4.20) Martin Devera. Metóda je popísaná v [18], [19] a [20].

2.5.1 Rozdiely medzi CBQ a HTB

HTB implementuje hierarchické zdieľanie linky odlišným spôsobom ako CBQ. Nasleduje zoznam odlišností:

- Iná je voľba estimátora. Namiesto merania medzipaketovej medzery sa používa vedro TB, čím sa dosahuje vyššia presnosť a jednoduchosť implementácie.
- Miesto bounded tried sa zavádza parameter *ceil*. Pre každú triedu definujeme dve rýchlosti – garantovanú G a maximálnu M. Garantovanú rýchlosť G dostane trieda vždy, ak má dostatočný záujem. Požičiavanie nadbytočnej kapacity linky je robené rovnako ako v CBQ. Obmedzená je však rýchlosť, ktorú požičiavaním možno dosiahnuť. Ak trieda C má možnosť požičať si viacej prenosovej kapacity od svojich predkov podľa LSG a dosahuje rýchlosť prenosu M, tak si ďalej požičiavať nemôže. Nadbytočná kapacita linky je potom rozdelená medzi ostatné triedy, ktoré ešte nedosiahli svoju rýchlosť M. Z praktického hľadiska je zavedenie druhej rýchlosti M pre každú triedu veľmi zaujímavé a na dosiahnutie podobného scenára v CBQ by sme museli definovať zložitú hierarchiu bounded rodičov pre dané triedy.
- Nie je podpora isolated tried ani exemp tried
- Autor definuje nový algoritmus pre výber paketu z tried. Tento algoritmus je presný, nie je to aproximácia *formálneho LSG*.

Výsledná implementácia HTB dostupná od Linuxového jadra 2.4.20 sa od CBQ líši hlavne jednoduchosťou nastavenia a presnosťou dodržiavania stanovených rýchlostí.

2.5.2 Kritika CBQ

Implementácii CBQ sa vyčítajú nasledovné nedostatky:

- *Prepočítavanie top level premennej*. Prepočítavanie sa deje pri zaradení paketu do triedy a pri poslatí paketu z triedy. Dôvod je zrejmý, obe tieto akcie totiž ovplyvnia, či je list v strome plný alebo nie. Taktiež ovplyvnia aj spokojnosť uzlu, ktorá závisí od plnosti fronty. Spokojnosť uzlu podľa definícia závisí nielen od toho, či je fronta v triede plná alebo nie, ale aj od aktuálneho toku, ktorý nedávno prešiel cez danú triedu. Túto hodnotu meria a odhaduje estimátor a deje sa to pri poslatí paketu z danej triedy. Odhad nedávno použitej rýchlosti sa však mení aj s časom a tak trieda, ktorá bola v čase vyslania paketu klasifikovaná ako nadlimitná, sa môže po uplynutí dostatočnej časovej doby, kedy nebola používaná, zmeniť na podlimitnú. Táto asynchrónna udalosť nie je v *top-level* algoritme zohľadnená a preto celý algoritmus trpí určitou nespresnosťou.
- *Meranie medzipaketovej medzery*. V realite je to veľmi ťažká záležitosť, najmä pri vysokých rýchlostiach, kde ide o mili až nanosekundy. Merania môžu byť dosť nepresné

a závisia od presnej fyzickej rýchlosti rozhrania, ktorá môže kolísať a od rozlíšenia časovača.
- *Potreba znalosti skutočnej rýchlosti rozhrania.* Pri CBQ treba ako parameter zadávať aj skutočnú fyzickú rýchlosť rozhrania, ktorá sa používa pri výpočtoch. Nie je problém určiť rýchlosť napr. ethernetového rozhrania, ale je to nemožné pri zariadeniach ako sú wireless siete a softverové zariadenia (tunely), kde rýchlosť kolíše a záleží na mnohých okolnostiach.
- *Nastavenie estimátora CBQ.* Nastavenie CBQ aby robilo to, čo sa od neho očakáva je značne náročné. Treba uviesť množstvo parametrov a v mnohých prípadoch treba vychádzať z praktických meraní a skúseností. Ďalej aj nastavenie a implementácia CBQ estimátora tiež nie je jednoduchá záležitosť. Autori CBQ spísali o tomto samostatný dokument [21].

Devera uvádza, že vyššie uvedené problémy boli impulzom na vývin novej disciplíny, ktorú nazval HTB a ktorá rieši všetky horeuvedené nedostatky.

2.5.3 Charakteristiky HTB etimátora

Estimátor odhaduje pre každú triedu, koľko prenosovej kapacity použila nedávno a podľa toho jej pridelí značku. Každá trieda v HTB má dva základné parametre
- garantovaný tok r
- maximálny tok c (tzv ceil-strop)

Estimátor v čase *t* pre každú triedu *C* s aktuálnym tokom $A_C(t)$ označí triedu *C* nasledovnou farbou
- zelená ak $A_C(t) < r$ (v CBQ podlimitná)
- žltá ak $r \leq A_C(t) \leq c$ (v CBQ nadlimitná)
- červená ak $A_C(t) > c$ (v CBQ nadlimitná – CBQ nerozlišuje od predchádzajúceho)

Namiesto merania medzipaketovej medzery a výpočtu času odoslania paketu z triedy, sa ako estimátor používa TB (token bucket). Na rozdiel od CBQ estimátora TB je jednoznačne charakterizovaný iba dvoma hodnotami TB(r,b)
r – je rýchloť doplňovania tokenov (v bytoch/sek)
b – je burst, t.j. hĺbka vedra v bytoch

Veľkými výhodami TB oproti estimátoru použitom v CBQ sú

- *nezávislosť na rozlíšní časovača*
  Pri štandardných nastaveniach kernelu je rozlíšenie časovača iba 10 ms. S nízkym rozlíšením časovača sa autor vysporiadal pomocou parametra TB *burst*. Ak daný parameter užívateľ pri definícii triedy neuvedie, tak si ho program sám vypočíta podľa vzťahu

  $$\text{burst} \geq d * \text{r}$$

  kde *d* je rozlíšnie časovača v sekundách (implicitne 10ms) a *r* je parameter TB. Ak je burst nastavený takto, tak sa vedro naplní každých 10ms tokenmi doplna. Za týchto 10 ms sa vedro bez dopĺňania tokenov môže úplne vyprázdniť a bude dodržaný

grantovaný tok *b* aj bez dopĺňania tokenov. Ak by bol burst menší, tak by sme počas 10 ms mohli vyčerpať tokeny skôr.

- *nezávislost od fyzickej rýchlosti rozhrania*
  Pri špecifikovaní hierarchického stromu pre výstupnú linku sa nikde neudáva parameter hovoriaci o fyzickej rýchlosti výstupného zariadenia. V HTB jednoducho pracuje vedro TB a svojou definovanou rýchlosťou dopĺňa tokeny nezávisle od konkrétnej rýchlosti vysielania rozhrania.

V reálnej implementácii HTB každej triede prislúchajú dva TB. Jeden je určený na počítanie garantovaného toku a druhý zase na počítanie maximálneho toku.

2.5.4 Algoritmus výberu ďalšieho paketu (LSG)

V kapitole o rozdieloch HTB a CBQ sme spomínali, že HTB implementuje nový algoritmus, ktorý rozhodne, z ktorej triedy zobrať paket a poslať ho na výstupné rozhranie. Algoritmus je definovaný nasledovne:

1) Zo všetkých plných listov zvoľme taký, ktorý by si pri odoslatí paketu mohol požičať od rodiča *R*, ktorý je na najnižšej úrovni zo všetkých takýchto rodičov. (T.j. plný list si môže úpožičať od predkov $P_1,..P_k$, ktorí sú na ceste ku koreňu. Zoberme predka P_i ktorý nie je červený a má zo všetkých predkov najnižší level. Takýchto predkov nájdime pre všetky plné listy.)
2) Ak je takých listov viac, vyberme ten s najvyššou prioritou
3) Ak máme stále viac listov striedajme ich pravidelne, pričom sa použije DRR(deficit round robin) a z každého listu sa pošle toľko paketov, aby sa zachoval pomer rýchlostí jednotlivých tried

Pozn 1: Predpokladá sa že každý list je rodičom sám sebe a môže si „požičať“ tok od seba – takto sú riešené podlimitné zelené triedy Týmto je garantované, že pri výbere listu vždy najskôr uspokojíme (nehľadiac na prioritu) vždy aspoň garantované toky všetkých listov (ak má list dostatočný záujem).

Pozn2: Pripomeňme pravido, ktoré je nutné splniť pri požičiavaní (formal LSG): Označme X ako level najvyššieho nespokojného uzla C (môže to byť list alebo vnútorná trieda). Potom nie je možné si požičať časť prenosovej rýchlosti od uzlu ktorý je na leveli > X.

Veta: Algoritmus spĺňa horeuvedené pravidlo.

Dôkaz: Dôkaz je jednoduchý. Zo všetkých plných listov zoberieme vždy ten, ktorý ma nespokojného rodiča na úrovni, ktorá je najnižšia z úrovní všetkých nespokojných uzlov. Táto úroveň je vždy $\leq X$ (úroveň najvyššieho nespokojného uzla) a teda si požičiavam z úrovne vždy menšej rovnej. Ak si požičiam z úrovne X, tak daný uzol na úrovni X prestane byť stálym požičiavaním si prenosovej rýchlosti nespokojný a úroveň X sa môže zvýšiť. Medzičasom sa tiež môže stať že niekto na nižších úrovniach začal byť nespokojný a úroveň X sa tým zníži.

3 Simulácie

Táto kapitola sa venuje empirickým meraniam, ako sa správa protokol TCP na linke za rôznych podmienok. Na simulácie použijeme nástroj ns2 ktorý je voľne šíriteľný [37]. Z bohatej funkcionality programu budeme využívať hlavne časti, ktoré sú podstatné z pohľadu prenosových charakteristík TCP komunikácie.

Nastavenia simulácie sú popísané v súboroch v jazyku TCL. Výsledky simulácie sa zapisujú do súboru, odkiaľ budeme extrahovať čiastočné výsledky a tieto vizualizovať nástrojom *gnuplot*. Niekedy sa pred vizualizáciu dáta vyhladia EWMA filtrom.

Pri vytváraní simulácií budeme nastavovať nasledovné parametre:

Topológia siete

- usporiadanie siete a určenie linky, ktorú budeme merať

Prenosová rýchlosť linky (BW bandwidth)

- linky budú full-duplexné. Prenosová rýchlosť je myslená v jednom smere.

Propagačné oneskorenie (PD propagation delay)

- oneskorenie spôsobené šírením signálu. V simuláciách bude väčšie zdržanie simulovať viacero prechodov v sieti.

Dĺžka výstupnej fronty (QS queue size)

- dôležitý parameter hlavne pri FIFO radení paketov

Implementácia (TCP)

- ns2 ponúka možnosť zvoliť si rôzne implementácie protokolu TCP, ako sú napríklad Tahoe, Reno, Vegas, Sack1, NewReno, Fack

Cwnd vs. wnd

- zvolenie metódy, ktorá bude limitujúcim faktorom vplývajúcim na prenosovú rýchlosť TCP komunikácie. Cwnd znamená, že použijeme kontrolu toku metódou predchádzanie zahltenia, wnd zase metódou klasickej kontroly toku bez prechádzania zahlteniu

Veľkosť prenášaného paketu (PS packet size)

- veľkosť paketu bude pre TCP prenosy konštantná. Keďže taktiež vplýva na charakteristiky prenosu, je jedným z parametrov.

Vyhladzovanie grafov

- čisté dáta získané zo simulácie niekedy pre prehľadnosť vyhladíme EWMA (exponential weighted moving averages) filtrom, ktorý je definovaný ako

$$NH = a * PH + (1-a)MH$$

kde *NH* je nová hodnota, ktorú získame z predchádzajúcej hodnoty *PH* a z nameranej hodnoty *MH*, pričom vyhladzovacia konštanta *a* bude zväčša nastavená na 0.9, ak nebude povedané inak.

Výsledné grafy budú mať na ose x vždy časový údaj v sekundách. Na ose y budú nanášané merané veličiny. Veličiny, ktoré budeme merať sú nasledovné

Merané veličiny

Aktuálna prenosová rýchlosť linky

- pre sledovanú linku je to počet bytov/s ktoré na danej linke v danom čase prechádzajú

Systémová prenosová rýchlosť linky

- pre daný prenos dát je to priemerná rýchlosť prenosu, t.j. to čo reálne z prenosu dostáva užívateľ. V čase t je definovaná ako suma dovtedy prenesených dát deleno t.

Systémová prenosová rýchlosť linky2

- v čase t je definovaná ako suma prenesených dát za prechádzajúce dve sekundy deleno dva.

Delay(Latencia)

- zdržanie paketu pri prenose. Ak je paket úspešne prenesený, tak zdržanie závisí iba od toho, koľko čakal vo výstupnej fronte. Ak je paket zahodený, tak sa k tomu pridáva aj čas preposlatia paketu.

Jitter

- veličina, ktorá charakterizuje zmenu zdržania (je to rozdiel v $J_t = D_t - D_{t-1}$, kde J_t je jitter v čase t a D_t je delay v čase t)

Naplnenosť výstupnej fronty

- sledovanie plnosti výstupnej fronty na zvolenej linke

Cwnd

- sledovanie premennej cwnd pri prenose protokolom TCP

Počet prenesených dát

- numerická veličina udávajúca, koľko dát sa za daný čas podarilo efektívne preniesť

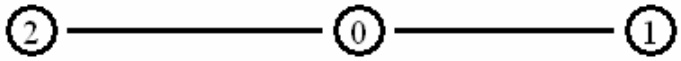
Stratovosť paketov

- numerická veličina udávajúca, koľko paketov bolo zahodených pri preplnení siete. Iné zahadzovania paketov neuvažujeme.

3.1 Jeden tok TCP

Cieľom týchto simulácií je podať obraz, ako sa jeden TCP tok správa na nezahltenej linke ktorú nezdieľa s ostatnými tokmi. Pri simuláciách predpokladáme použitie mechanizmov na predchádzanie zahlteniu, ktoré kontrolujú využitie linky. Preto veľkosť okna vysielateľa nastavíme na vysoké hodnoty, ktoré nebudú obmedzujúcim faktorom pre rýchlosť.

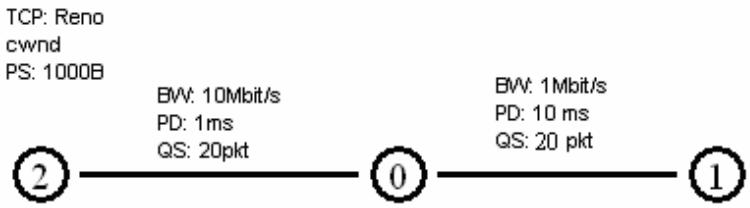
Topológia siete je pre všetky simulácie jedného toku rovnaká a je zobrazená na nasledovnom obrázku. Do uzla 2 sme umiestnili zdroj dát TCP a do uzlu 1 príjemcu dát, ktorý dáta príjme a naspäť pošle potvrdzovaciu správu ACK. Úzkym miestom v sieti je vždy uzol 0, ktorého výstupnú linku 0-1 budeme sledovať.



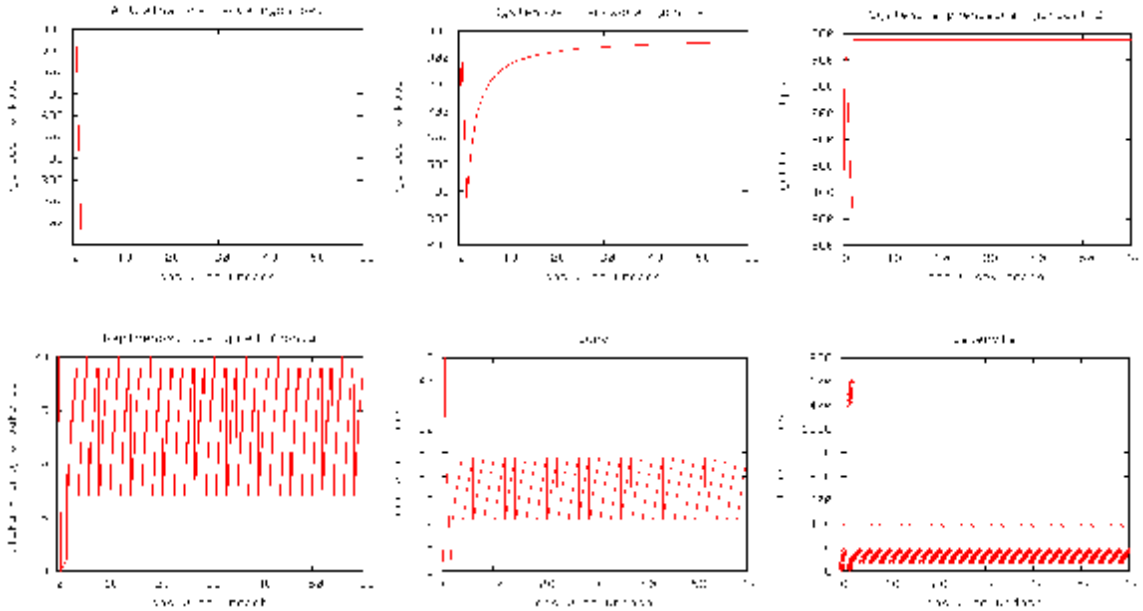
3.1.1 Simulácia 1

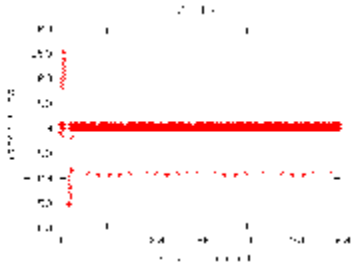
Touto simuláciu chceme ukázať dopad dĺžky výstupnej fronty na efektívnosť prenosu pri použití zahadzovania paketov Tail drop. Jedná sa o typický príklad, kedy sa rýchla linka poskytovateľa internetu pripája na pomalú, alebo shapovanú linku užívateľovi domov. Zároveň chceme demonštrovať dôvody, prečo poskytovatelia nastavujú vysoké dĺžky týchto front.

Nastavenia parametrov vidno na nasledujúcom obrázku. Simulácia trvala 60 sekúnd. Pri kreslení grafov sme nepoužili vyhladzovanie.



Po spustení simulácie sme dostali nasledovné grafy a údaje



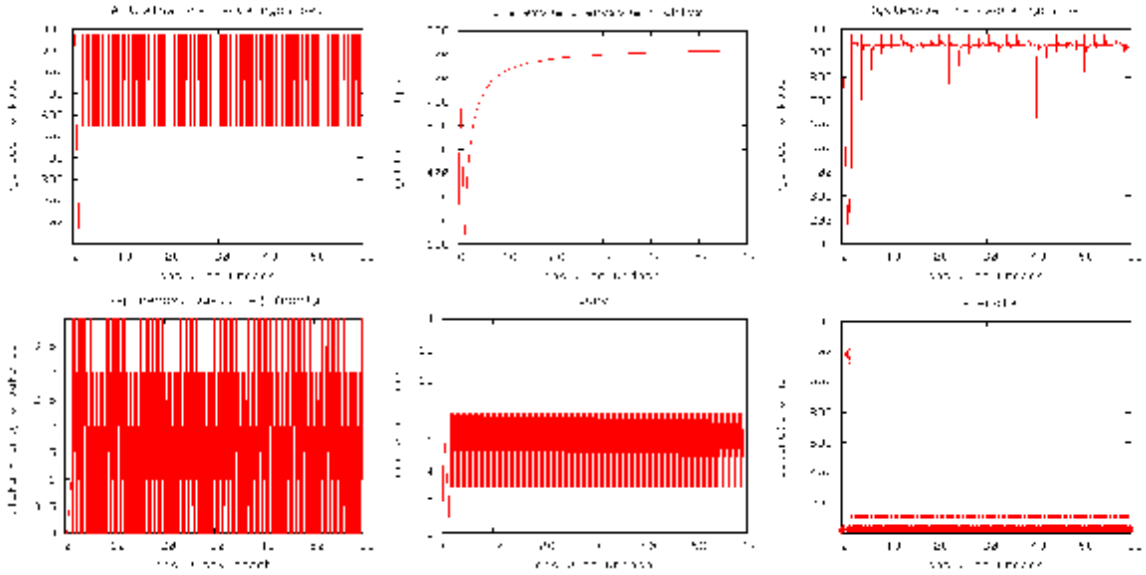


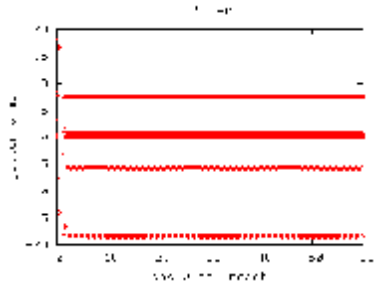
Počet prenesených dát: 7197.773438 KB
Počet poslatých paketov: 7139
Počet zahodených paketov: 51
Percento straty: 0.714386%

Ako vidno zo simulácie, výstupná fronta v uzle 0 mala kapacitu 20 paketov, pričom nikdy nebola prázdna. Na grafe aktuálnej rýchlosti vidno, že linka 0-1 bola vždy vyťažená na maximum. Možno si všimnúť, že latencia paketov závisí od meniacej sa dĺžky fronty. Priebeh cwnd má typický pílovitý charakter. Na začiatku je vidno fázu pomalého štartu, ktorá sa skončila po preplnení fronty a strate viacerých paketov. Celému grafu dominuje fáza predchádzanie zahlteniu s rýchlym preposielaním a rýchlym zotavením. Počiatočné väčšie straty paketov možno pozorovať na začiatku grafu latencie, kde skoro dve sekundy trvalo zistenie straty a úspešné preposlatie paketu. Strata bola spôsobená exponenciálnym nárastom rýchlosti, ktorú síce linka 2-0 zvládla, avšak linka 0-1 spolu s veľkosťou výstupnej fronty (16pkt) nie.

3.1.2 Simulácia 2

Simulácia je rovnaká ako v predošlom prípade, znížime však veľkosť výstupnej fronty na linke 0-1 iba na štyri pakety. Po simulácii dostávame nasledovné výsledky



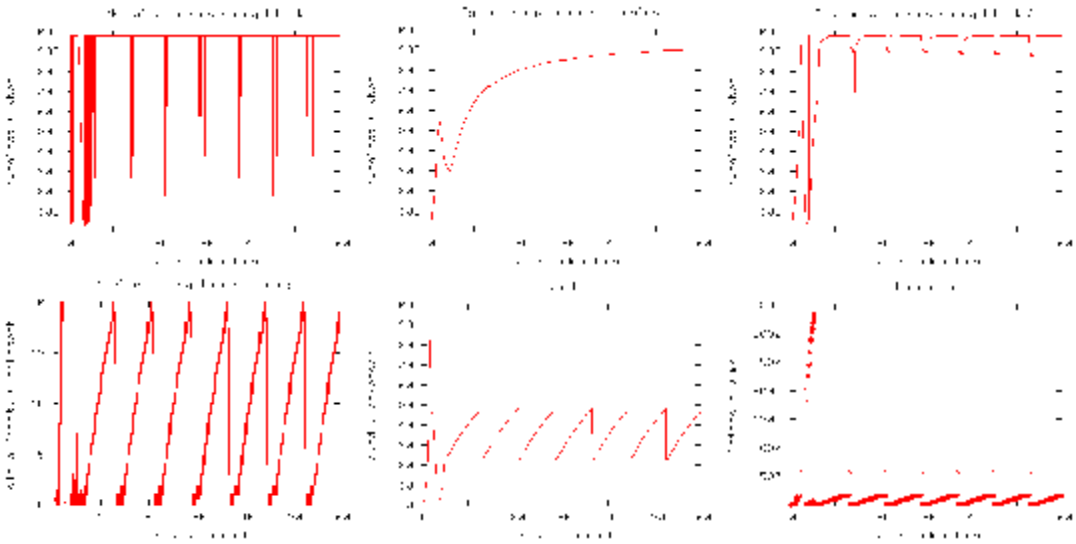


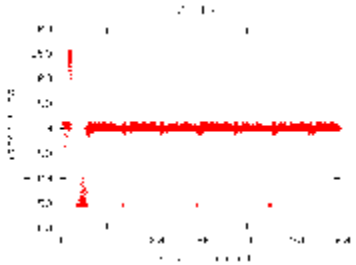
Počet prenesených dát: 6866.679688 KB
Počet poslatých paketov: 6967
Počet zahodených paketov: 205
Percento straty: 2.942443%

Z grafov možno vyčítať, že využitie linky a naplnenosť výstupnej fronty 0-1 veľmi výrazne osciluje. Systémová rýchlosť narastá veľmi pomaly a počet prenesených bytov je nižší. Z grafov vidno potrebu mať dobre nastavenú veľkosť výstupnej fronty na efektívne využitie linky.

3.1.3 Simulácia 3

Pre zaujímavosť zaveďme ešte jednu zmenu do simulácie. Zmeňme parameter PD linky 2-0 na vyššiu hodnotu, aby simulovala zdržanie, ktoré narastá putovaním paketu po Internete (narátavanie propagačného oneskorenia z transkontinentálnych liniek plus oneskorenia spôsobené smerovačmi, ako aj čakaním vo fronách). Nastavenia simulácie sú rovnaké ako v prvom prípade, meníme iba PD pre linku 2-0 na hodnotu 100 ms. Po simuláciách dostávame grafy



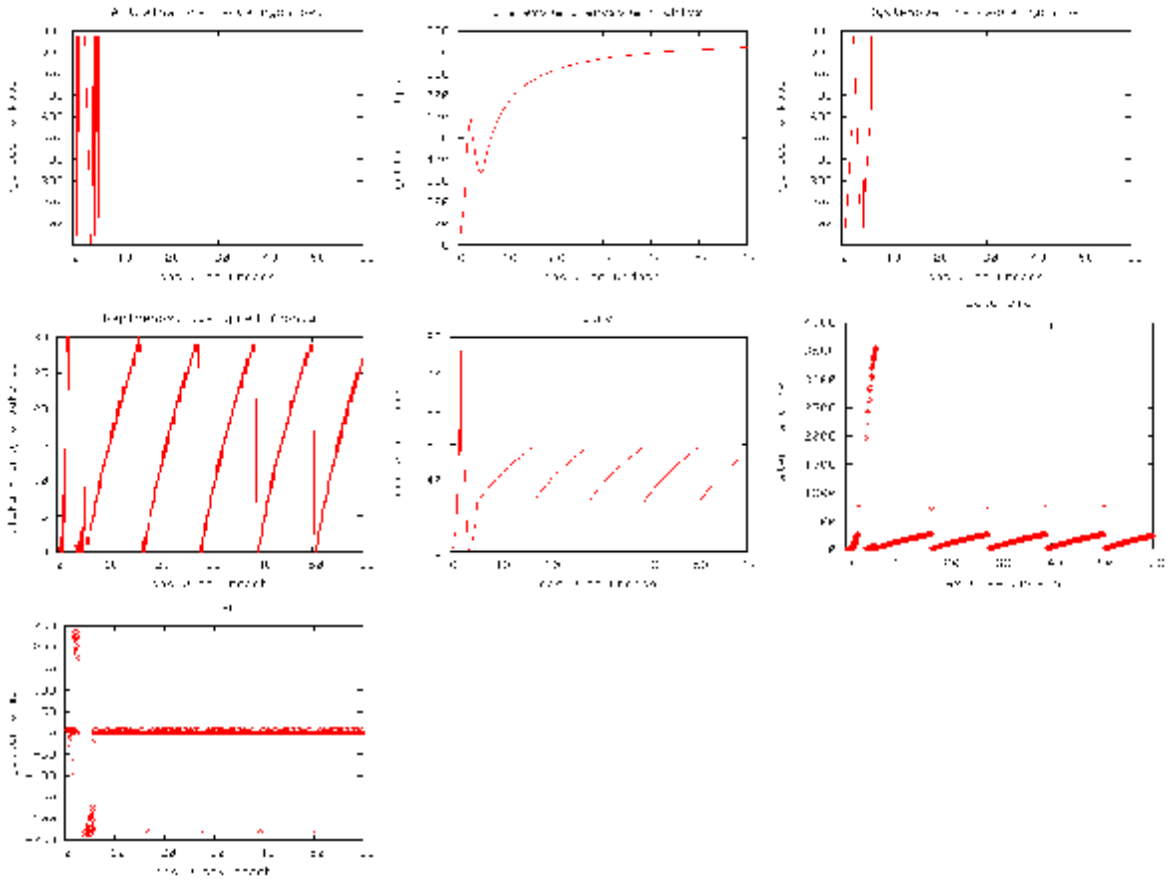


Počet prenesených dát: 6797.617188 KB
Počet poslatých paketov: 6747
Počet zahodených paketov: 53
Percento straty: 0.785534%

Vidno, že s nárastom propagačného oneskorenia sa zvýšili aj nároky na kapacitu výstupnej fronty. Linka 0-1 nie je využitá naplno a jej využitie osciluje. Ďalšími simuláciami sme zistili, že na plné využitie linky je potreba až 30 paketovú kapacitu fronty, ako znázorňujú nasledovné grafy.

3.1.4 Simulácia 4

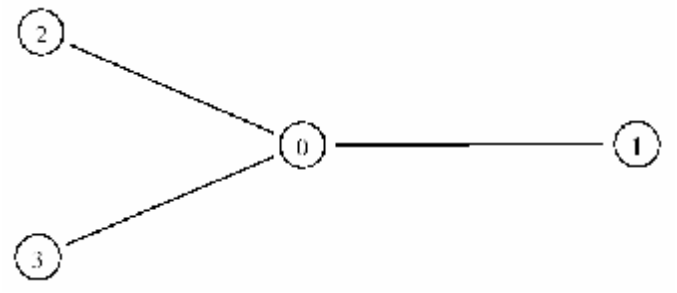
Simulácia je nastavená rovnako ako simulácia 3, až na to, že linka 0-1 má kapacitu výstupnej fronty nastavenú na 30 paketov. Cieľom bolo overiť, že 30 paketová kapacita fronty stačí na plné využitie linky 0-1.



3.2 Viacero tokov TCP súperiacich o zdieľanú linku

Cieľom týchto simulácií je podať obraz, ako viacero tokov TCP súperi o jednu linku, ktorú medzi sebou zdieľajú. Pri simuláciách predpokladáme použitie mechanizmov na predchádzanie zahlteniu, ktoré kontrolujú využitie linky. Preto veľkosť okna vysielateľa nastavíme na vysoké hodnoty, ktoré nebudú obmedzujúcim faktorom pre rýchlosť.

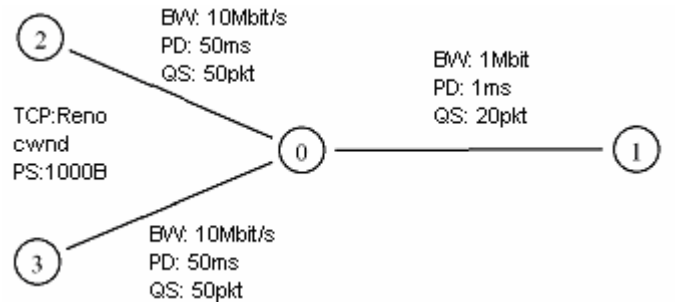
Topológia siete je pre všetky simulácie rovnaká a je zobrazená na nasledovnom obrázku. Do uzlov 2 a 3 sme umiestnili zdroj dát TCP a do uzlu 1 príjemcu dát pre oba zdroje (zvlášť), ktorý dáta príjme a naspäť pošle potvrdzovaciu správu ACK. Úzkym miestom v sieti je vždy uzol 0, ktorého výstupnú linku 0-1 budeme sledovať.



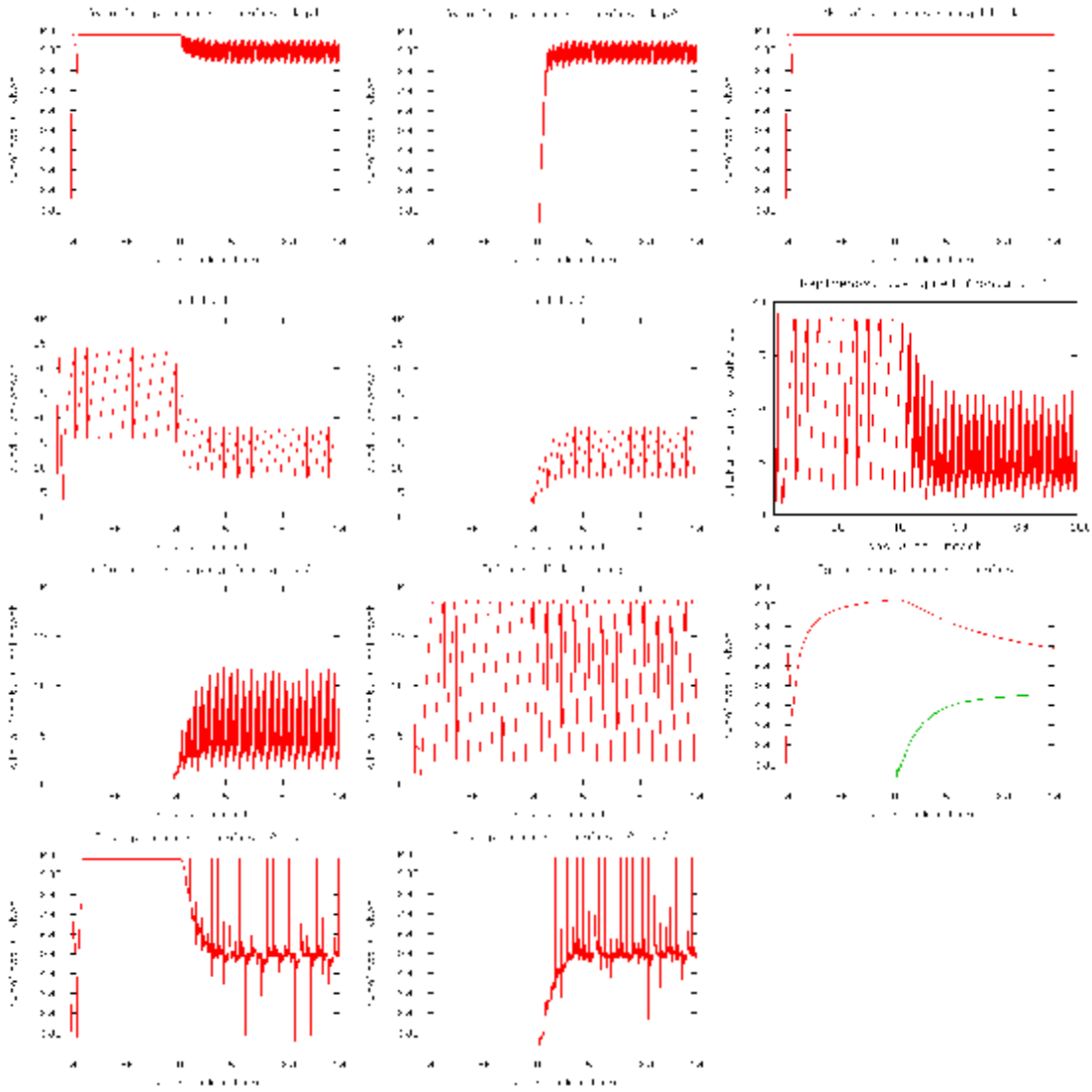
3.2.1 Simulácia 1

Touto simuláciou chceme ukázať ako si dva TCP toky s rovnakými charakteristikami liniek rozdelia spoločnú zdieľanú linku 0-1, pri použití zahadzovania paketov Tail drop.

Nastavenia parametrov vidno na nasledujúcom obrázku. Simulácia trvala 100 sekúnd. Prvý tok označený ako *tcp 1*, prenášal dáta plných 100 sekúnd, druhý tok označený ako tcp 2 sme spustili v čase 40 s. Keďže grafov popisujúcich simuláciu je viac ako v prípade jedného toku, uvádzame z nich iba niektoré. Pri kreslení grafov aktuálnej rýchlosti jednotlivých tokov na linke 0-1 sme použili EWMA vyhladenie s parametrom $a = 0.9$ (pre všetky simulácie v podkapitole 3.2), ostatné grafy sú bez vyhladzovania. Vyhladzovanie spôsobuje skreslenie výsledkov a pri sčítaní rýchlostí načrtnutých v grafe vychádza, že využitie linky presahuje 1 Mbit/s.



Po spustení simulácie sme dostali nasledovné výsledky



Počet prenesených dát tcp 1: 8620.664062 KB
Počet poslatých paketov tcp 1 : 8555
Počet zahodených paketov tcp 1: 66
Percento straty tcp 1: 0.771479%

Počet prenesených dát tcp 2: 3369.882812 KB
Počet poslatých paketov tcp 2 : 3344
Počet zahodených paketov tcp 2: 25
Percento straty tcp 2: 0.747608%

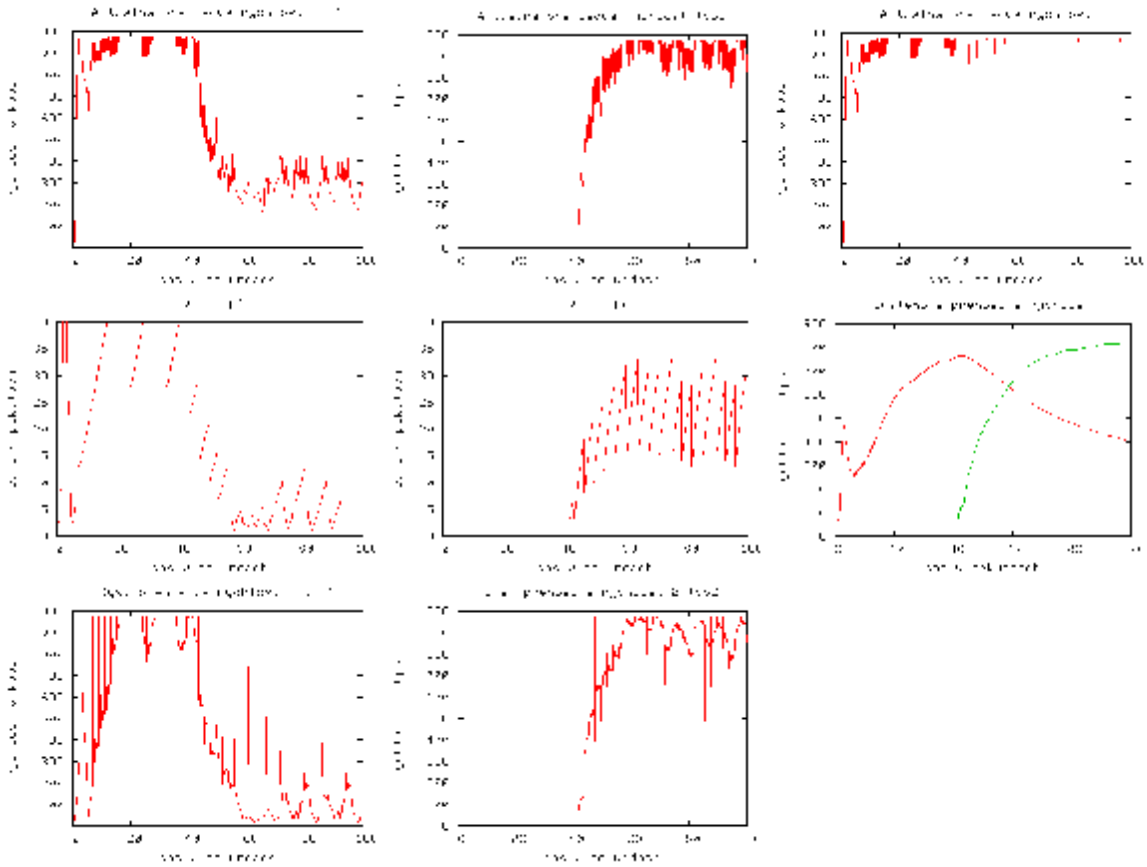
Ako vidno na grafoch systémovej prenosovej rýchlosti ako aj na grafoch cwnd a zabratia FIFO fronty, tak oba toky si zhruba na polovicu rozdelili prenosové pásmo zdieľanej linky. Graf aktuálnej prenosovej rýchlosti po čase 40 s veľmi výrazne osciloval pre oba toky, preto sme použili vyhladzovanie. Toto však skresľuje výsledky a v skutočnosti cez linku striedavo prechádzali pakety raz jedného, raz druhého TCP spojenia. Graf s názvom Aktuálna prenosová rýchlosť je výsledkom súčtu rýchlostí jednotlivých tokov a mimo iné naznačuje, že cez linku 0-1 stále prúdili dáta. Tento fakt vidno aj na obrázku celková dĺžka fronty, ktorá nikdy neklesla na nulu.

3.2.2 Simulácia 2

Nasledovné simulácie slúžia na ukážku rozdelenia linky medzi dva protokoly TCP, ktoré majú rôzne chrakteristiky prístupových ciest. Nastavenia simulácie sú rovnaké ako v simulácii 1 až na to, že linka 2-0 má zvýšenú hodnotu PD na 150ms. Tok *tcp 1* je teda v zjavnej nevýhode oproti toku *tcp 2*, čo sa odzrkadlí aj na zdieľaní linky.

Cieľom ďalších troch simulácií je ukázať ako rôzne techniky kontroly premávky skorigujú uvedenú nevýhodu protokolu *tcp 1* a ako zabezpečia férovosť pri zdieľaní linky. Pri simulácii 2 je na linke 0-1 použitá FIFO metóda, pri simulácii 3 RED metóda a pri simulácii 4 metóda FQ (fair queueing).

Po spustení simulácie pre radenie FIFO sme dostali nasledovné výsledky



Počet prenesených dát tcp 1: 5005.039062 KB
Počet poslatých paketov tcp 1 : 5005
Počet zahodených paketov tcp 1: 76
Percento straty tcp 1: 1.518482%

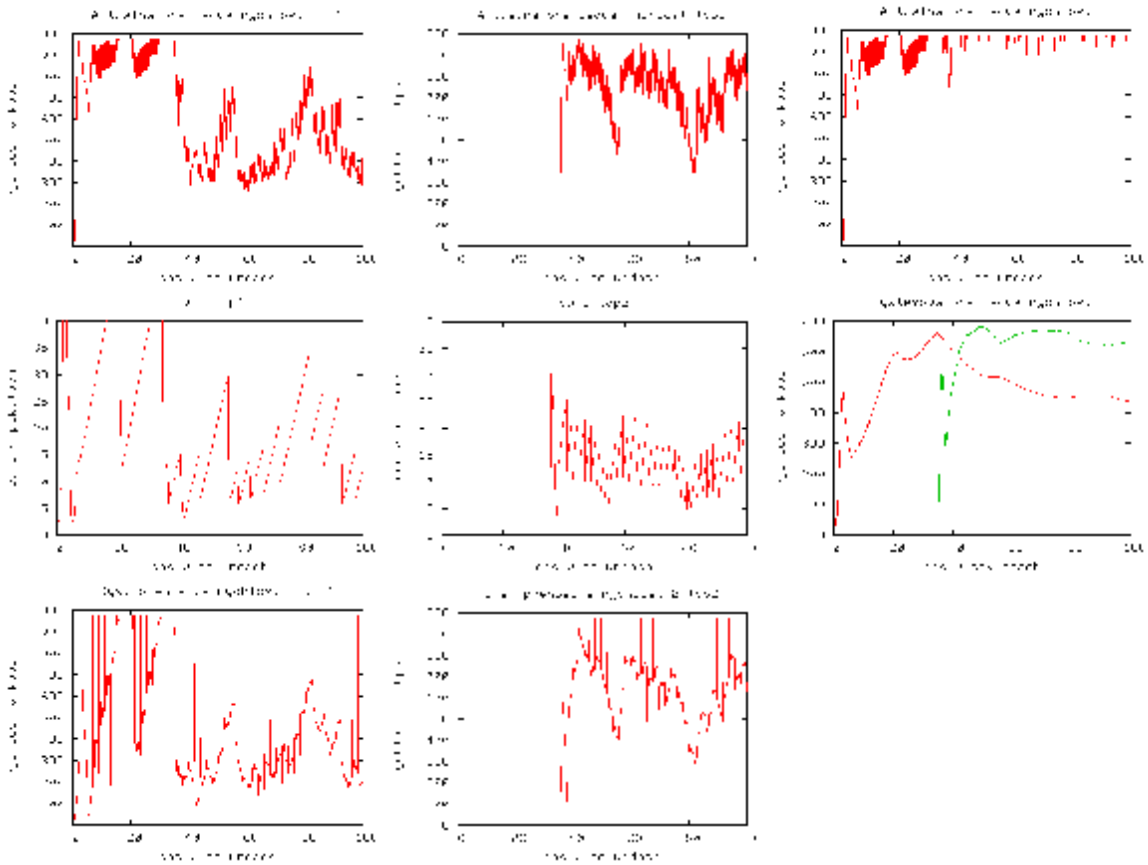
Počet prenesených dát tcp 2: 6054.179688 KB
Počet poslatých paketov tcp 2 : 5980

Počet zahodených paketov tcp 2: 18
Percento straty tcp 2: 0.301003%

Pri radení FIFO vidno zjavnú výhodu toku tcp 2. Hneď po jeho spustení v čase 40s pozorujeme jeho razantný nástup a zabratie väčšej kapacity linky pre seba.

3.2.3 Simulácia 3

V tejto simulácii sú nastavenia rovnaké ako v simulácii 2, na linke 0-1 je však použitá metóda RED. Po spustení simulácie sme dostali nasledovné výsledky



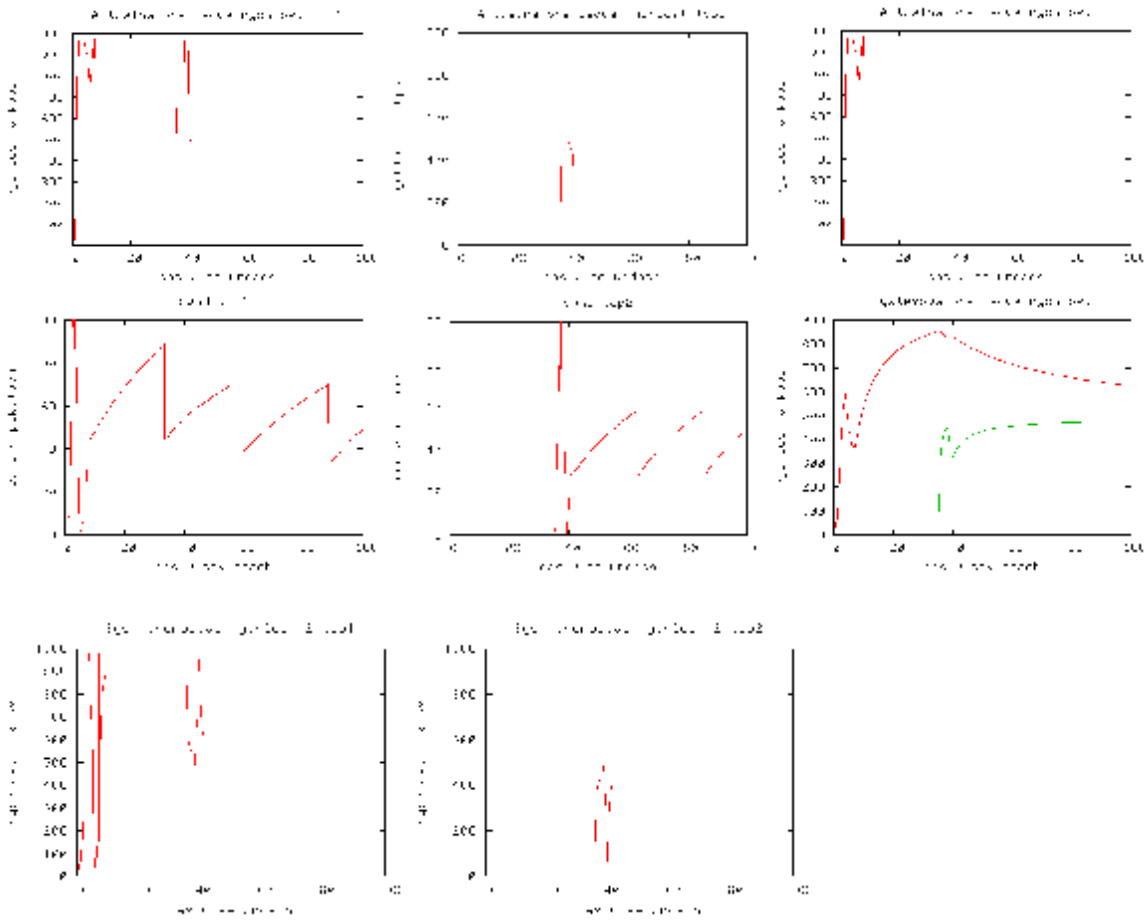
Počet prenesených dát tcp 1: 5436.679688 KB
Počet poslatých paketov tcp 1 : 5421
Počet zahodených paketov tcp 1: 67
Percento straty tcp 1: 1.235934%

Počet prenesených dát tcp 2: 5142.148438 KB
Počet poslatých paketov tcp 2 : 5121
Počet zahodených paketov tcp 2: 57
Percento straty tcp 2: 1.113064%

Náhodnosť zahadzovania paketov v podaní techniky RED zaviedla určitý stupeň férovosti a mierne znevýhodnila tok tcp 2 (vidno zvýšenie stratovosti) a dala šancu aj toku tcp1.

3.2.4 Simulácia 4

V tejto simulácii sú nastavenia rovnaké ako v simulácii 2 a na linke 0-1 je použitá metóda FQ. Rozdielna je ešte kapacita fronty pre každú triedu FQ, ktorá bola nastavená na hodnotu 50 paketov. Po spustení simulácie sme dostali nasledovné výsledky



Počet prenesených dát tcp 1: 7811.210938 KB
Počet poslatých paketov tcp 1 : 7792
Počet zahodených paketov tcp 1: 91
Percento straty tcp 1: 1.167864%

Počet prenesených dát tcp 2: 3887.851562 KB
Počet poslatých paketov tcp 2 : 3904
Počet zahodených paketov tcp 2: 59
Percento straty tcp 2: 1.511270%

Z grafov možno vyčítať úplnú izoláciu jednotlivých tokov a presné rozdelenie prenosového pásma linky 0-1 na polovicu.

4 Kontrola premávky v Linuxe

Linux v sebe zahŕňa bohatú množinu nástrojov na manipuláciu a prenos paketov. Najznámejší nástroj danej problematiky je určite Linuxový firewall ako aj stovky sieťových služieb. Menej známy je subsystém TC (traffic control) na kontrolu premávky, ktorý bol zavedený do jadier *2.2* a vylepšený pre jadrá *2.4*.

4.1 Prehľad konceptov a pojmov

Pred samotným nahliadnutím do implementácie kontroly premávky v Linuxe, treba uviesť zopár základných konceptov a pojmov, ktoré sa v danej problematike používajú. Vychádzame z publikácií [22], [23] a [24]. V texte sú uvedené slovenské pojmy a korenšpondujúce anglické pojmy.

TC kontrola premávky (traffic control): je názov pre množinu mechanizmov radenia do front a mechanizmov ktorými je paket prijatý na smerovači a následne odvysielaný. Záhŕňa to rozhodnutia o tom či prijať paket, akou rýchlosťou odvysielať paket, v akom poradí vysielať pakety. Ako TC sa zvykne označovať aj množina nástrojov na správu front a mechanizmov radenia do front. Aby bol zmätok ešte väčší, tak synonymom pre TC sa zvykne v niektorej literatúre označovať aj pojem QoS(quality of service – kvalita služieb).

Tok(Flow): je spojenie alebo konverzácia medzi dvoma koncovými bodmi resp. stanicami. V TCP/UDP sa ako tok zvyčajne chápu pakety s rovnakou štvoricou: zdrojová IP adresa, cieľová IP adresa, zdrojový port, cieľový port. Koncept tokov sa používa na separáciu premávky na smerovačoch. Toky sa zvyčajne agregujú do rovnakých tried, ktoré definujú určitú úroveň poskytnutej služby a prenášajú sa ako jeden veľký agregovaný tok. Na základe tokov sa zvykne rozdeľovať aj kapcita linky.

Pakety, segmenty, rámce: Pre ujasnenie pojmov treba spomenúť aj termíny, ktoré sa zaužívali pre označenie elementov prenášaných cez tzv. packet switched siete. Tieto termíny záležia od vrstvy o ktorej je reč.
Pre druhú vrstvu (data link layer) sa zaužíval pojem rámec. Pre tretiu vrstvu (network layer) pojem paket a pre štvrtú vrstvu (transport layer) zase pojem segment.

V ďalšom texte nasledujú definície elementov, ktoré sa zvyknú označovať ako tradičné elementy kontroly premávky. V implementácii TC v Linuxe sa používajú iné pojmy. Uvádzame aj korenšpondenciu medzi klasickými elementami TC a elementami v Linuxe.

Fronta(Queue): je jeden z najzákladnejších pojmov kontroly premávky. Je to miesto, kde čaká konečný počet entít (paketov) pred tým, ako budú obslúžené. Najjednoduchším príkladom je typ fronty FIFO, kde sú pakety obslúžené v takom poradí, v akom prišli. Nad frontou sú definované dve operácie – operácia zaradenia paketu do fronty (Enqueue) a operácia vybratia paketu z fronty a vyslanie paketu na výstupné zariadenie (Dequeue). V ďalšom texte budeme používať anglické ekvivalenty uvedených pojmov. Fronta sama o sebe nie je veľmi zaujímavý koncept, ale v spolupráci s ostatnými mechanizmami ktoré pozdržujú, menia poradie a zahadzujú pakety, sa dá robiť kontrola premávky. Z pohľadu

vyššej vrstvy sú skutočne viditeľné iba fronty a vyššia vrstva iba zaradí paket do fronty a očakáva, že bude systémom TC prenesený.

Shaper: Pre tento pojem sme nenašli vhodný slovenský ekvivalent, preto budeme používať iba anglický. Tento element slúži na pozdržovanie paketov pred vyslaním pričom sa tým sleduje cieľ, aby výstupný tok spĺňal dopredu dohodnutú výstupnú rýchlosť. Postranným efektom tohto pozdržovania paketov, je vyhladenie priebehu premávky (priebeh zhlukovitej vstupnej premávky s výraznými výkyvmi sa vyhladí) ako aj zmena zdržania a variácie zdržania charakteristík daného toku. Uvedená metóda sa používa hlavne na splnenie požiadaviek na veľkosť výstupnej rýchlosti toku. Mechanizmy použité na dosahovanie cieľov shaper-a sú non-work-conserving mechanizmy (definované nižšie).

Plánovač (Scheduler): táto entita rozhoduje o zmene poradia odosielania paketov. Pakety prichádzajú na vstup v určitom poradí a úlohou plánovača je určiť ich výstupné poradie.

Klasifikátor (Classifier): Jeho úlohou je diferenciácia premávky. Príchodzie pakety sa zvyknú deliť do tried, ktoré sú rôznym spôsobom obsluhované. Týmto dostávame rôzne úrovne poskytnutej služby.

Policer: Pre tento pojem sme nenašli vhodný slovenský ekvivalent, preto budeme používať iba anglický. Je to v podstate áno/nie otázka, ktorou sa dá charakterizovať premávka. Je daná limitná hodnota charakteristiky premávky a dve akcie. Pre danú premávku, prechádzajúcu zariadením sú merané jej charakteristiky a podľa toho či je uvedená premávka pod alebo nad daným limitom sa vykoná akcia. V praxi býva hraničnou hodnotou prenosová rýchlosť a policer býva implementovaný ako Token Bucket. Premávka, ktorá vstupuje do zariadenia pod daným limitom býva klasifikovaná do určitej triedy a zaradená do výstupnej fronty pre ďalšie spracovanie. Premávka nad limitom býva zväčša zahodená alebo klasifikovaná do inej triedy.

Zahadzovanie (Droping): zahadzovať možno jednotlivý paket, celý tok alebo množinu paketov vyhovujúcu vopred stanovenej podmienke. Zahodiť paket znamená zrušiť ho v zariadení, do ktorého prišiel a neposlať ho na výstupnú linku. V IP sieťach o tomto fakte nezvykne sieťový prvok informovať vysielateľa.

Označovanie (Marking): označovanie je mechanizmus pri ktorom dôjde ku zmene paketu. Zmenou paketu sa myslí buďto samotná zmena obsahu alebo hlavičky paketu, alebo len zmena meta informácie, ktorá je dostupná pre každý paket. Táto meta informácia sa neprenáša s paketom v sieti, je dostupná len v zariadení, cez ktoré paket práve prechádza.

Zachovávajúca prácu, nezachovávajúca prácu (Work conserving, non-work-conserving): pre oba pojmy sme sa rozhodli používať anglické akvivalenty. Entita, ktorá vyberá ďalší paket na výstup sa zvykne označovať ako work-conserving, ak pre každú žiadosť výstupného zariadenia, ktoré oznamuje, že je pripravené pre ďalší prenos paketu mu je ďalší paket poskytnutý (ak je nejaký k dispozícii). Ináč sa označuje ako non-work-conserving.
Ak máme napríklad 10 Mbit/s výstupnú linku a chceme dosiahnuť aby výstupný tok bol 1 Mbit/s, potom musíme zabezpečiť, že výstupná linka bude na 90% nečinná. To znamená, že nutne musíme pozdržovať pakety pri žiadosti hardwaru o poslanie ďalšieho paketu.

4.2 Implementácia a elementy TC v Linuxe

Základnými objektami v implementácii TC v Linuxe sú iba 3 elementy:

- Qdisc (radenie do front - queueing discipline)
- Trieda (class)
- Filter

Problematika TC v Linuxe nie je ani do dnešných dní dobre zdokumentovaná. Oproti minulému roku sa však situácia značne zlepšila a existuje zopár zdrojov, ktoré sa jej venujú. Nasledovná časť bola spracovaná hlavne z materiálov dostupných v [22] až [28].

4.2.1 Radenie do front (Qdisc)

Každé sieťové zariadenie má v Linuxe pridelený qdisc. Táto entita berie na vstup paket (operácia enqueue zaraďuje došlý paket do qdiscu príslušného zariadenia) a dáva ho na výstup (operácia dequeue sa znaží odvysielať paket z qdiscu cez výstupné zariadenie). Najjednoduchším a implicitným qdisc-om pre každé zariadenie je *pfifo_fast* (čo je FIFO fronta s tromi triedami). Qdisc sa považuje za základný stavebný kameň TC v Linuxe.

Qdisc-y rozdeľujeme na *classless* a *classfull* podľa toho, či vo svojom vnútri môžu obsahovať užívateľsky definované triedy alebo nie. Toto rozdelenie však treba upresniť z toho dôvodu, že niektoré classless qdiscy majú vo svojom vnútri triedy. Rozdelenie je myslené ohľadom na možnosť konfigurácie daných tried užívateľom. Ak pre daný qdisc možno pridávať, meniť, uberať triedy – považuje sa tento za classfull. Ako classfull qdiscy sú označované napríklad HTB a CBQ. Pre príklad možno uviesť qdisc PRIO, kde sú vo vnútri definované tri triedy a pre každú triedu samostatná fronta. Tieto triedy však nie je možné nijako meniť a modifikovať - pri vytvorení qdiscu majú dopredu určenú funkcionalitu. Preto je uvedený qdisc zaradený do skupiny classless.

Ďalej treba uviesť, že každé rozhranie v skutočnosti obsahuje dva qdisc-y. Jeden sa používa pre tzv *egress* radenie do front a druhý pre *ingress* radenie do front. Ako egress sa označuje proces posielania paketov z daného zariadenia smerom von do siete a ingress ako proces príjmu paketu zo siete smerom dnu do systému. Príslušné qdisc-y sa označujú ako *root qdisc* a *ingress qdisc*. Root qdisc sa môže pýšiť plnou funkcionalitou TC implementovanou v Linuxe, pokým ingress qdisc ju má obmedzenú. Dôvody sú také, že odchod paketov zo zariadenia má v plnej moci systém a môže na to použiť plnú silu TC, pokým príchod paketu nie je nijako možné regulovať a o tomto rozhoduje vysielajúca strana. Väčšina dokumentácie z tejto oblasti je preto venovaná hlavne root qdisc-om. Ingress qdisc umožňuje definovať iba policery pre príchodzie pakety a nie je možné robiť shaping ani scheduling. Ak chceme nejako regulovať premávku prichádzajúcu do zariadenia musíme, to robiť iba pomocou policerov. Existuje však aj riešenie, ktoré umožňuje aj pre ingress proces definovať všetku funkcionalitu root qdiscu. Je to virtuálne zariadenie IMQ, ktoré však zatiaľ nie je súčasťou štadartných Linuxových jadier.

Zoznam a možnosti nastavení jednotlivých qdisc-ov, ktoré sú implementované v jadre, sa dajú nájsť napríklad v [18], [22], [23], [26], [27] alebo aj v komentároch v zdrojových textoch jadra.

4.2.2 Triedy

Koncepcia tried vychádza z koncepcie hierarchického zdieľania linky, ktorá je uvedený v kpt 2.3. Triedy sú určené na vytváranie hierarchií, preto sa používajú pojmy ako rodič triedy a potomok triedy. Úlohou triedy je poskytnutie určitej úrovne služby.

Triedy existujú iba ako súčasť classfull qdisc-ov. Možno ich vytvoriť ako entity sami o sebe, ale vždy treba uviesť tzv. rodičovskú triedu alebo rodičovský qdisc pri vytváraní. Každá trieda je určitého typu. Platí zásada, že ak ak je trieda napríklad typu HTB, tak jej potomkovia môžu byť iba triedy typu HTB (nemôžu byť napr. typu CBQ).

4.2.3 Filtre

Filtre tvoria jeden z najkomplexnejších komponentov v rámci TC v Linuxe. Môžeme sa na ne pozerať ako na lepidlo, ktoré spája ostatné elementy. Jedným zo základných cieľov filtra je klasifikácia paketov. Celkovo v sebe filter zahŕňa funkcionalitu troch elementov známych z klasických pojmov TC:

- klasifikácia
- zahodenie paketu (drop)
- policer

Filter ako klasifikátor
Filter sa pripája ako súčasť qdisc-u alebo triedy. Keď paket vstúpi do qdisc-u, tak je konfrontovaný filtrami ktoré ho klasifikujú t.j. pošlú do nejakej triedy. V tejto triede môže byť konfrontovaný ďalšími filtrami a následne reklasifikovaný t.j. poslatý do inej triedy alebo qdisc-u. V Linuxe sú filtre, ktoré vedia filtrovať podľa najrozličnejších kritérií, nasleduje ich zoznam:

- **u32** klasifikuje pakety podla informácie v hlavičke paketu
- **route** klasifikuje pakety podľa smerovacej tabuľky (routing table) a znalosti riadku zo smerovacej tabuľky, ktorý bol pre daný paket použitý
- **fw** firewall ako súčať TC v Linuxe umožňuje označkovať pakety (meta informáciou). Podľa tejto značky sa môže taktiež filtrovať.
- **TC_INDEX** je filter určený pre architektúru DiffServ a klasifikuje pakety podľa tohoto štandardu (tejto architektúre sa v práci nevenujeme).

Filter ako policer
Policer v Linuxe vystupuje iba ako súčasť filtra a nie samostatná entita. Pre policer sa definujú *metriky* a *akcie* ktoré sa vykonajú pre daný paket podľa výsledku merania.

Pomocou policera sa dá realizovať aj obmedzovanie premávky, kedy sa ako akcia po prekročení stanoveného limitu zvolí zahodenie paketu. Takýmto spôsobom možno realizovať obmedzovanie došlej premávky v ingress procese spracovania paketu - ak pakety prichádzajú prirýchlo, tak ich jednoducho začneme zahadzovať (a dúfať, že to druhá strana interpretuje ako zahltenie a podľa vzoru TCP spomalí rýchlosť vysielania).

Súčasťou policer-a v linuxe je entita zvaná *merač* (meter), ktorá implementuje metriku. Ako metrika sa používa meranie rýchlosti prechádzajúcich paketov a využíva sa princíp Token

bucket, ktorý sa zvykne označovať aj termínom Leaky bucket. Zvyčajne sa používajú nasledovné štyri typy policer-ov, ktoré sú súčaťou TCNG a popísané napr. v [29]. Líšia sa medzi sebou spôsobom ako je implementovaný merač.

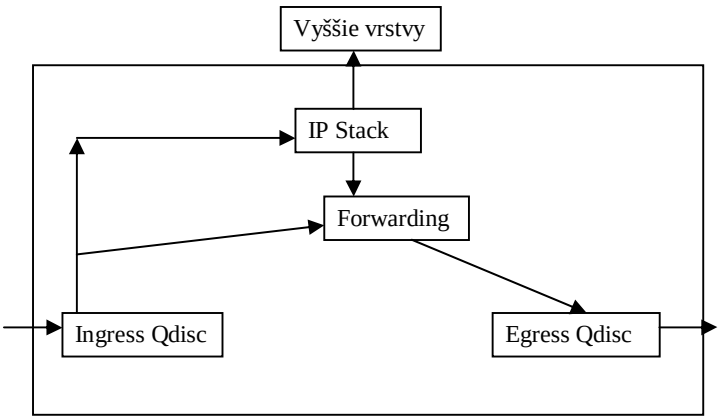
1) single leaky bucket
2) double leaky bucket meter
3) single rate three color meter (RFC 2697)
4) two rate three color meter policer (RFC 2698)

Filter ako zahadzovač paketov

Akcia zahodenia paketu (DROP) je súčasťou policer-a a teda aj súčasťou filtra. Toto je jediné miesto, kde užívateľ môže explicitne špecifikovať zahodenie paketu v celom sytéme TC. Zahadzovanie paketov sa v systéme TC deje aj napríklad pri preplnení front, toto však užívateľ explicitne nemôže špecifikovať a deje sa to ako vedľajší efekt.

4.3 Putovanie paketu cez sytém TC

Pre správne pochopenie systému TC je dôležité si uvedomiť ako je spracovávaný paket pri prechode cez tento systém. Na nasledovnom obrázku sú zachytené časti TC s ktorými je paket na svojej ceste konfrontovaný.



Došlý paket prejde cez Ingress qdisc, ktorý je pridelený vstupnému sieťovému zariadeniu. Potom je rozhodnuté, či je určený pre tento systém a teda sa dostane sa vyšších vrstiev, alebo je určený na preposlatie (forwarding). Paket určený pre poslatie von do siete môže vzniknúť z dvoch dôvodov:

1) systém vygeneruje paket určený pre poslatie
2) príchodzí paket je určený na preposlatie (forwarding)

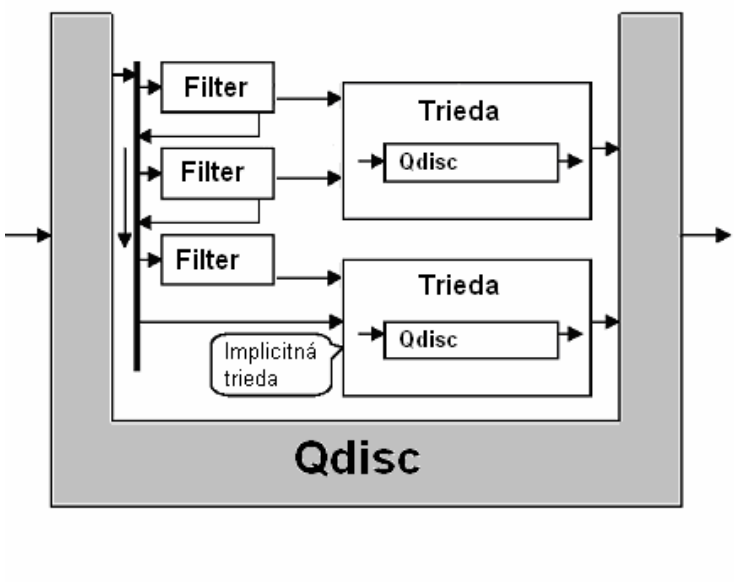
Pre paket určený na poslatie von do siete sa nájde výstupné zariadenie. Každé výstupné zariadenie obsahuje root/egress qdisc. Daný paket je zaradený do root qdiscu operáciou enqueue. Pri oznámení výstupného sieťového zariadenia o možnosti vyslať ďalší paket je na qdisc-u zavolaná operácia dequeue.

4.3.1 Spracovanie paketu qdisc-om

Zvonku funguje root qdisc ako čierna skrinka, ktorá podporuje dve operácie enqueue a dequeue. Pri zaradení paketu do qdiscu sa použije operácia enqueue a pri žiadosti sieťového rozhrania o ďalší paket sa zavolá operácia dequeue.

Enqueue
Vnútri v qdiscu je príchodzí paket postupne konfrontovaný filtrami, ktoré sú zoradené v postupnosti špecifikovanej užívateľom. Prvý vyhovujúci filter paket klasifikuje do triedy, ktorá je súčasťou qdiscu. Vrámci tried môžu byť ďalšie filtre a daný paket reklasifikovať. V rámci každej triedy je ďalší qdisc (implicitne pfifo_fast) kam je paket zaradený operáciou enqueue. Situáciu zobrazuje nasledovný obrázok.

Pri hierarchickom zdieľaní linky pomocou CBQ alebo HTB neplatí, že každá trieda má vo svojom vnútri qdisc. Toto tvrdenie platí iba pre listové triedy. Nelistové triedy slúžia iba na prerozdelenie nadbytočnej kapacity a nemali by obsahovať pakety na posielanie. Ak pri klasifikovaní dôjde k tomu, že sa paket nedostane do listovej triedy, tak je odvysielaný priamo na zariadenie plnou rýchlosťou, alebo ak je uvedená implicitná trieda v qdisc-u, tak do nej.



Dequeue
Ako bolo spomínané, tak každé sieťové zariadenie v Linuxe ma práve jeden root qdisc rQ. Kernel samotný komunikuje iba s týmto qdisc-om rQ. Pri žiadosti sieťového zariadenia o ďalší paket na odoslatie, kernel zavolá operáciu dequeue na rQ. Táto vyberie z pomedzi všetkých paketov čakajúcich na odoslatie ďalší, ktorý bude fyzicky odvysielaný. Ako sa postaví rQ k požiadavke, je špecifické pre každý qdisc.

4.4 Mapovanie tradičných elementov na objekty TC v Linuxe

V podkapitole 4.1 boli spomenuté základné koncepty a pojmy kontroly premávky. Tu spomenieme, ako je rozložená a implementovaná funkčnosť jednotlivých objektov klasickej kontroly premávky (shaping, plánovanie, klasifikácia, policer, zahadzovanie paketov, označovanie paketov) do elementov sytému TC v Linuxe. Pripomeňme ešte, že v Linuxe sú definované iba tri elementy: qdisc, class a filter.

shaping – class

Triedy, ktoré sú súčasťou classfull qdisc-ov HTB, CBQ vedia robiť shaping. Pre triedu sa špecifikuje výstupná rýchlosť, akou z nej majú odchádzať pakety. Okrem týchto tried vie shaping robiť aj TBF qdisc.

plánovanie – qdisc

Qdisc berie paket na vstup operáciou enqueue, a posiela na výstup operáciu dequeue. V rámci týchto oprácií môže dôjsť k preusporiadaniu paketov a zdržovaniu paketov.

klasifikácia – filter

Filtre slúžia na klasifikáciu paketov

policing – filter

Ako bolo spomínané, súčasťou filtra je v Linuxe aj policer a nevystupuje ako entita sama o sebe.

Zahadzovanie paketov – filter

Explicitné zahadzovanie je implementované ako akcia DROP v rámci filtra, nie je na to vyčlenená špeciálna entita

Označovanie paketov – (dsmark) qdisc + firewall

Pre túto funkciu je v systéme TC vyčlenený špeciálny qdisc, ktorý je súčasťou *diffserv* architektúry, ktorej sa v tejto diplomovej práci nevenujeme. Na označovanie paketov sa dá použiť aj napríklad linuxový firewall.

4.5 Software a nástroje

Aby sme mohli využívať plnú silu systému TC, musí byť táto zahrnutá pri preklade do jadra. V zdrojových súborov jadra sa totiž nachádzajú implementácie všetkých qdisc-ov, tried a filtrov dostupných v Linuxe. Ďalej je potrebné nainštalovať balík iproute2. Ten v sebe obsahuje nástroje na nastavovanie parametrov spomínaných elementov. Najzákladnejším nástrojom je utilita tc, ktorá toto umožňuje. V spolupráci s týmto programom sa zvykne používať aj Linuxový firewall, ktorého podporu je tiež nutné zahrnúť pri preklade jadra.

4.5.1 Nástroj tc

Pre nástroj tc a všetky možnosti nastavenia jednotlivých objektov TC v Linuxe chýba kvalitná a úplná dokumentáciu. Ešte v nedávnej dobe (2002/2003) nebola k tejto problematike dokumentácia skoro žiadna, situácia sa však postupne zlepšuje. Vznikla napríklad skupina lartc.org, ktorá sa tejto problematike venuje na svojich stránkach. Dosť kvalitnú dokumentáciu poskytol aj autor HTB Martin Devera na svojich stránkach [19], [20], čo bol nepochybne správny krok a pomohol k úspešnému presadeniu sa tejto disciplíny. Táto je v súčasnosti veľmi obľúbená a slúži ako náhrada za komplikovaný a málo pochopený CBQ. Ďalším zaujímavým dielom je aj diplomová práca [30], ktorá vo svojej dobe bola jediným zdrojom dokumentácie pre utilitu tc.

4.5.2 Pomocné skripty

Utilita tc je síce silný nástroj, ktorým je možné nastaviť veľa detailov qdisc-ov, tried a filtrov, má však veľmi zdĺhavú syntax, ktorá nie je vo veľkej obľube. Chýbajúca dokumentácia taktiež spôsobila, že syntax je chápaná chybne a ľudia ju nevedia používať. Pri nastavovaní parametrov treba mať aj slušnú úroveň znalostí, čo ktoré nastavenie znamená a veľakrát aj značné množstvo skúseností, keďže nastavenia nie vždy prinášajú želané výsledky (CBQ).
Preto vznikali rôzne pomocné skripty, ktoré sa snažili zjednodušiť nastavenia, pričom skrývali rôzne detaily a ponúkali iba časť funkčnosti systému TC v Linuxe. Každé vo svojom vnútri využíva príkazy utility tc. Medzi tieto nástroje patria napr: wondershaper, myshaper, htb.init, cbq.init.

Skripty htb.init [31] a cbq.init [32] sa snažia komplikovanú syntax pri definovaní hierarchickej štruktúry tried pri metóde hierarchického zdieľania linky preniesť do hierarchie súborového systému. V súboroch sa definujú parametre tried, a hierarchia adresárov slúži ako obraz hierarchie tried. Príslušné skripty potom vygenerujú príkazy utility tc.

Cieľom skriptov wondershaper [33] a myshaper [34] je snaha o dosiahnutie rozumného využitia domácej linky do internetu. Riešenie Wondershaper si napríklad stanovilo nasledovné ciele:

1) udržať nízke zdržanie paketov pre všetku dostupnú premávku (t.j. prenos dát zo/do siete)
2) umožnenie surfovania po internete v slušnej rýchlosti ak na pozadí beží napríklad nejaké sťahovanie dát
3) zabezpečiť, aby sa simultánne sťahovanie (download) a nahrávanie (upload) navzájom neblokovali, čo je efekt, ktorý možno pozorovať napríklad pri použití DSL

Autori vidia v súčastnosti *nasledovné problémy s poskytovateľmi pripojenia*: Pre zákazníkov je v súčasnosti veľmi dôležitá rýchlosť sťahovania dát. Aby zákazníci mali výsoké rýchlosti pri sťahovaní, tak poskytovatelia majú nastavené veľké kapacity výstupných front, ktoré majú zabraňovať stratám paketov. Taktiež zariadenia na strane zákazníka majú pred odoslatím paketu nastavenú veľkú kapacitu fronty. Pri naplnení týchto front však dochádza k oneskoreniu paketov ktoré nimi musia prejsť. Napríklad pri sťahovaní súboru je pomalá práca s terminálovými službami typu telnet.

Navrhované riešenie vychádza práve z predpokladu, že oneskorenie paketov je spôsobené dlhými a preplnenými frontami ako na strane poskytovateľa, tak na strane výstupného sieťového zariadenia (ADSL modem). Navrhujú preto kontrolovať dĺžku fronty vo výstupnom sieťovom zariadení tak, že systém TC v Linuxe bude pakety vypúšťať len o málo nižšou rýchlosťou ako je fyzická výstupná rýchlosť zariadenia (t.j. v smere upload sa bude robiť shaping). Regulovanie fronty na strane providera bude robiť taktiež systém TC v Linuxe za pomoci ingress policer-ov (v smere download sa bude robiť policing). Pakety ktoré prekračujú fyzickú vstupnú (download) rýchlosť zariadenia budú jednoducho zahadzovať. Využíva sa tak vlastnosť protokolu TCP, ktorý na toto zareaguje znížením rýchlosti a dáta by mali prúdiť o niečo pomalšie. Za cenu zníženia rýchlosti v oboch smeroch teda dostávame riešenie, ktoré spĺňa všetky uvedené ciele a v dôsledku ktorého nebude dochádzať k prepĺňaniu front – tie sa totiž prenesú do systému TC v Linuxe a môžeme ich menežovať podľa svojich predstáv.

4.5.3 Projekt TCNG (traffic control next generation)

Veľmi zaujímavým projektom sa zdá byť snaha Wernera Almesbergera a vývoj nástroja tcng [35]. Ako sám autor píše, tcng je revízia kontroly premávky v Linuxe. Autor si stanovil nasledovné ciele:

- vylepšiť a prekonať nevýhody existujúcej architektúry a spraviť ju viacej rozšíriteľnou a škálovateľnou
- vyvinúť jazyk, ktorý má syntax podobnú rozšíreným programovacím jazykom ako sú napríklad Java alebo C.
- abstrahovať od konkrétnej implementácie QoS t.j. ak človek napíše skripty v tcng, mali by byť univerzálne použiteľné nielen pre kontrolu premávky v Linuxe

Projekt TCNG sa zdá byť ešte nedokončený. Zatiaľ sa skladá z dvoch častí [29].

1) traffic control compiler (TCC)

jeho úlohou je preložiť vstup v jazyku tcng do rôznych výstupných formátov. Jedným z výstupných formátov je aj preklad do jazyka tc, ako aj možnosť vytvoriť modul do jadra spolu s príkazmi na jeho aktiváciu

2)traffic control simulator(tcsim)

je nástroj slúžiaci na simuláciu správania sa kontroly premávky v Linuxe. Jeho účelom je najmä:

- validácia konfigurácie vygenerovanej nástrojom tcc
- vývoj konfiguračných skriptov
- testovanie komponentov TC v Linuxe

Do tcsim vstupujú skripty na nastavenie kontroly premávky v Linuxe a skripty popisujúce simuláciu. Výstupom je súbor popisujúci priebeh simulácie, z ktorého možno robiť aj grafické výstupy. Implementácia tcsim v sebe kombinuje:

- zdrojový kód kontroly premávky v Linuxe, ktorý je súčasťou jadra (t.j. qdisc, triedy, filtre)
- zdrojový kód utility tc, ktorá je súčasťou balíka iproute2

Pridáva k tomu svoj vlastný simulačný stroj a kód, ktorý uvedené súčasti dáva dokopy. Výsledný kód tcsim beží v user space móde. Simulácie sú však veľmi reálne a simulujú sa

časti kódu v kernel móde, ako je napríklad nahrávanie modulov do jadra. Reálnosť plynie aj z použitia skutočných zdrojových kódov, keď tcsim vykonáva presne ten kód, ktorý je aj súčasťou bežiaceho jadra. Tcsim používa zdrojový kód utility tc napríklad na skontrolovanie nastavení a zachováva pritom rôzne efekty plynúce z implementácií jednotlivých qdiscov (napr zaokrúhľovanie prijatých parametrov a pod.)

4.6 Simulácie

Táto kapitola sa venuje empirickému overeniu presnosti rozdelenia výstupnej kapacity linky pri použití metódy HTB. Na simulácie použijeme nástroje tc, ethloop a gnuplot. Nástroj ns2 žiaľ nemá implementovanú podporu HTB, preto ho nie je možné použiť. Nástrojom tc nastavíme hierarchiu tried a nástrojom gnuplot vizualizujeme výsledky simulácie. Ethloop je program, ktorý generuje CBR tok (constant bit rate – tok s konštantnou rýchlosťou) paketov na zvolené výstupné zariadenie, kde tieto pakety prejdú cez egress spracovanie (shaping) a vrátia sa späť, akoby boli prijaté zo siete. Ethloop tieto pakety zachytáva a do súboru zapisuje vstupnú, výstupnú rýchlosť a oneskorenie (delay).

Pri definovaní hierarchie tried v HTB budeme pre každú triedu definovať nasledovné tri parametre:

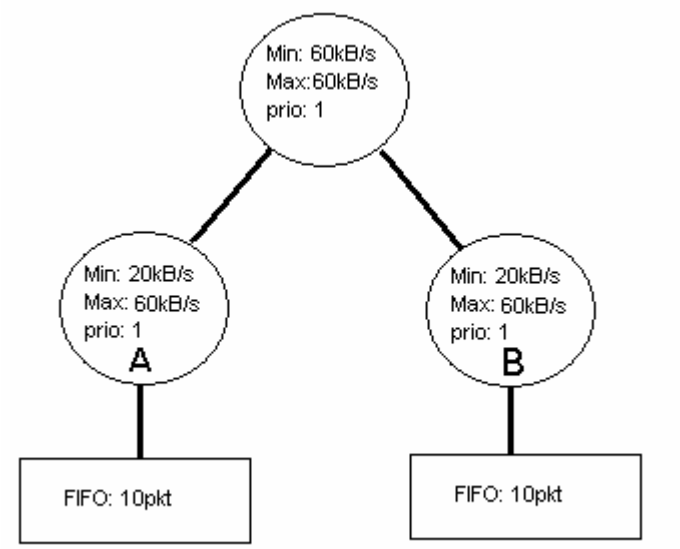
- garantovaná rýchlosť (*min*)
- maximálna rýchlosť (*max*)
- priorita (*prio*) (čím nižšie číslo, tým vyššia priorita. 0 znamená najvyššia priorita. Priority sú absolútne. To značí, že ak majú všetky triedy rovnaké číslo priority, tak sú si rovnocenné. Trieda s nižším číslom priority má absolútnu prednosť pred triedou s vyšším číslom priority.)

Pre každú listovú triedu vytvoríme qdisc, ktorý bude slúžiť na uchovávanie došlých paketov pred odoslatím.

Výsledok simulácie budú vždy tri grafy a jedna tabuľka. Prvý graf bude grafom vstupných rýchlostí jednotlivých tokov CBR tak, ako ich generoval nástroj ethloop podľa nastavenia. Druhý graf bude obsahovať výstupné rýchlosti týchto tokov po spracovaní v egress procese. Sú to tie výsledky merania, ktoré sú pre nás zaujímavé. Tretí graf bude zobrazovať latenciu paketov v egress procese spracovania. V tabuľke bude uvedená presná numerická hodnota vstupných rýchlostí a očakávaná hodnota výstupnej rýchlosti pre daný tok.

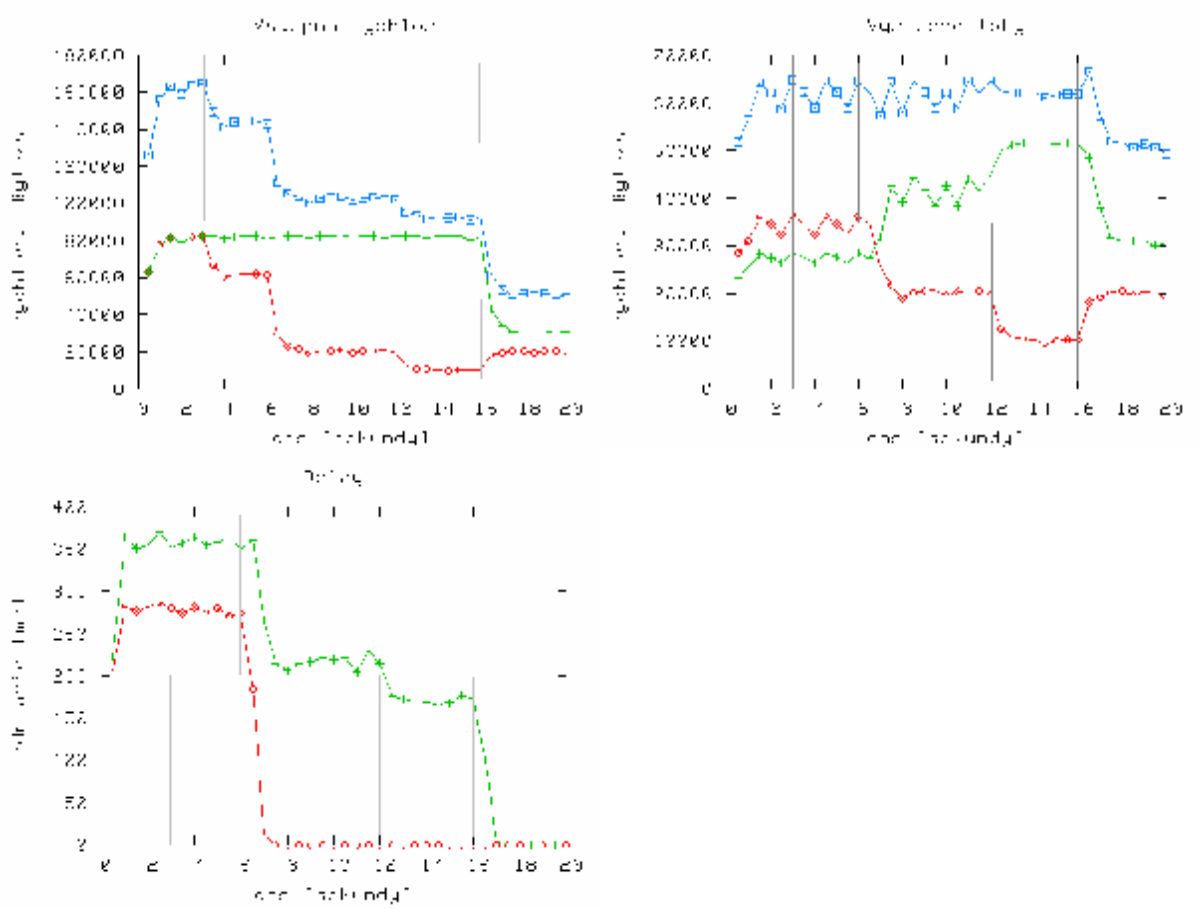
4.6.1 Simulácia 1

Simulácia sa venuje jednoduchému scenáru rozdelenia linky medzi dve organizácie *A* a *B*. Každej organizácii bola pridelená rovnaká garantovaná rýchlosť 20kB/s. Rodičovská trieda má kapacitu 60kB/s a umožňuje požičiavať deťom nevyužitú kapacitu. Všetky triedy majú rovnakú prioritu. Hierarchia je znázornená na nasledovnom obrázku.



Cez hierarchiu boli spustené dva toky CBR. Tok 1 prechádza cez triedu patriacu firme A a tok 2 cez triedu patriacu firme B. Na grafoch krivka s kosoštvorcami patrí toku 1, krivka s krížikmi toku 2 a krivka so štvorčekami predstavuje súčet kriviek oboch tokov. Zvislé čiary symbolizujú zmeny rýchlosti vstupných tokov.

Po spustení simulácie, ktorá trvala 20 sekúnd sme dostali nasledovné výsledky:

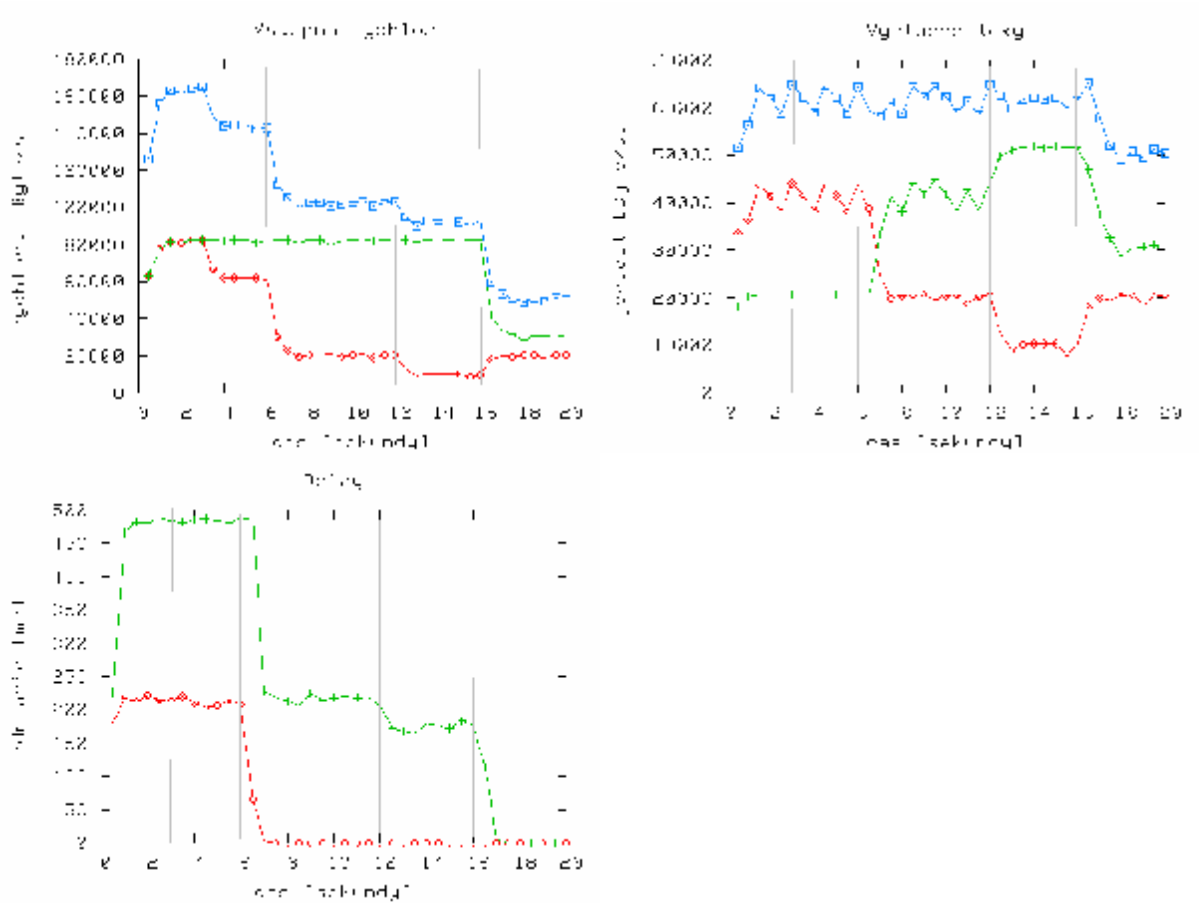


Čas [sek]	0-3	3-6	6-12	12-16	16-20
tok 1/vstup	80	60	20	10	20
tok 1/výstup	30	30	20	10	20
tok 2/vstup	80	80	80	80	30
tok 2/výstup	30	30	40	50	30

Vstupné rýchlosti sú generované programom ethloop a boli nastavené pred simuláciou. Boli zvolené tak, aby toky mali tendenciu využiť viac, ako plnú kapacitu linky (koreňová trieda 60kByte/s) Výstupné rýchlosti sú generované egress procesom spracovania na výstupnom zariadení a predstavujú proces shaping-u. Pri porovnaní tabuľky a grafu výstupných rýchlostí tokov vidno, že rýchlosti dosahujú približne očakávané hodnoty. Takisto sedí aj súčet rýchlostí, ktorý predstavuje najvyššia krivka, ktorá skutočne osciluje okolo hodnoty 60 a predstavuje tak koreňovú triedu. Zaujímavé je zistenie, že v čase 0-6 sekúnd nedošlo k presnému rozdeleniu nadbytočnej kapacity linky a že jeden z tokov mierne dominuje.

4.6.2 Simulácia 2

Pozrime sa teraz ako priorita vplýva na získanie nadbytočnej kapacity linky. Nastavenie hierarchie je rovnaké, iba triede pre firmu B sme nastavili prioritu na hodnotu 2 (t.j. nižšia priorita). Po simulácii dostávame nasledovné výsledky.



Čas [sek]	0-3	3-6	6-12	12-16	16-20
tok 1/vstup	80	60	20	10	20
tok 1/výstup	40	40	20	10	20
tok 2/vstup	80	80	80	80	30
tok 2/výstup	20	20	40	50	30

Výsledky simulácie sa od predošlých výsledkov líšia v časoch 0-6 sekúnd. Vtedy si môžeme všimnúť, že toku 2 bola pridelená iba garantovaná rýchlosť výstupnej kapacity a tok 1 dostal garantovanú plus všetku nadbytočnú kapacitu. Na grafe delay si môžeme všimnúť aj pokles omeškania paketov toku 1 a nárast omeškania paketov toku 2. V ďalších časoch sú grafy totožné, lebo tok 1 nemá záujem o nadbytočnú kapacitu linky.

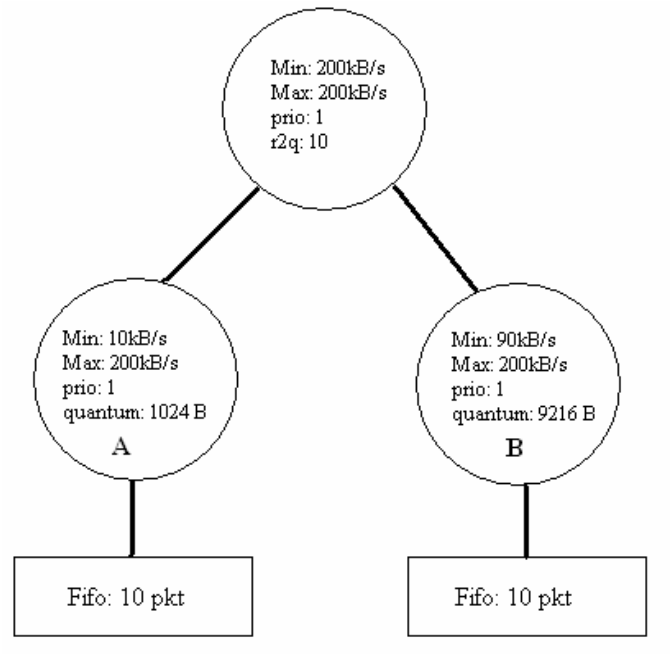
4.6.3 Simulácia 3

Touto simuláciu chceme demonštrovať možnosť nastavenia DWRR algoritmu, ktorý riadi rozdelenie nadbytočnej kapacity linky. Ten sa dá korigovať v HTB dvoma parametrami (takto nastavujeme parameter *Quantum[C]* (parameter DWRR) pre každú triedu C zvlášť):

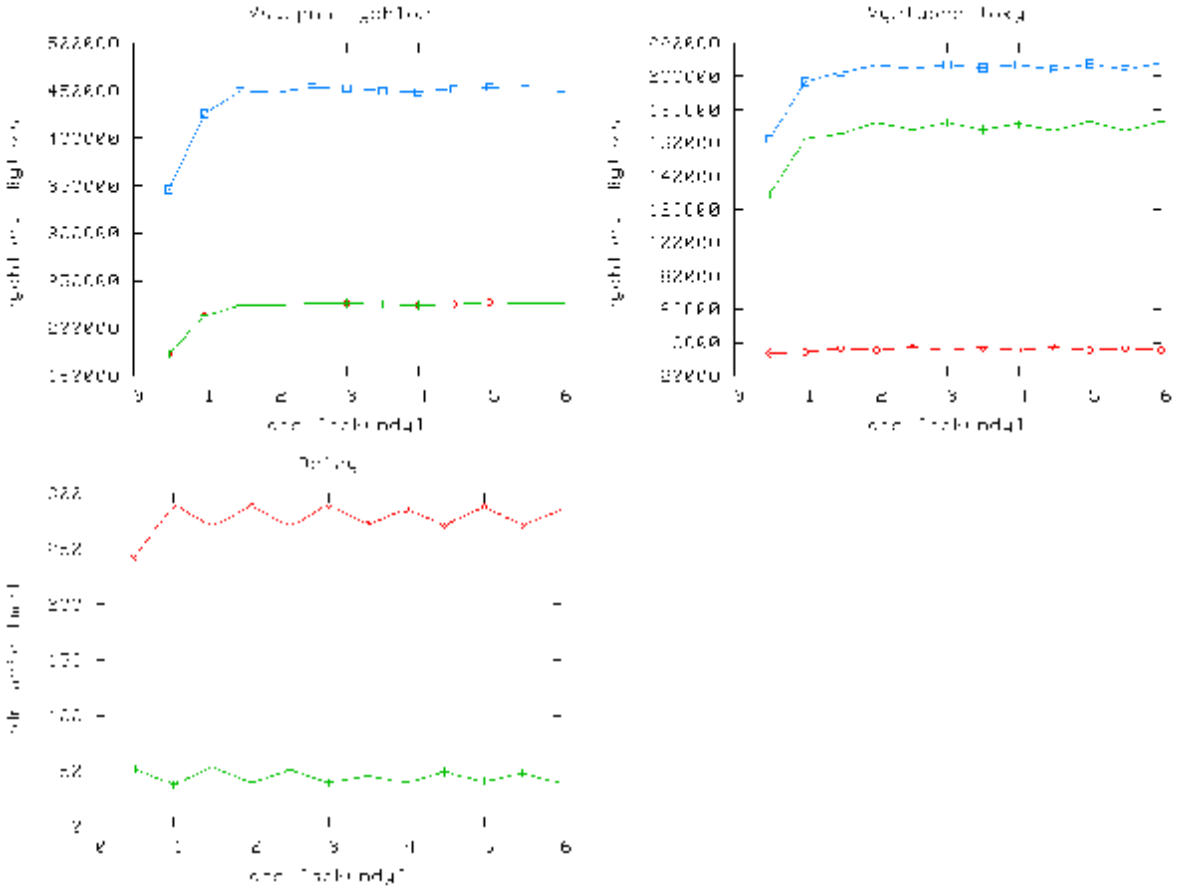
1. globálnym parametrom *r2q*, ktorý sa uvádza pri vytváraní HTB qdisc-u a pomocou ktorého sa počítajú kvantá do DWRR pre všetky triedy ako
   $$Quantum[C] = MinRate[C] / r2q$$
   Kde *MinRate[C]* je garantovaná rýchlosť triedy *C*.
2. pre každú triedu uvedieme parameter *quantum*, ktorým špecifikujeme koľko bytov sa môže v jednom kole DWRR z danej triedy poslať. Pri špecifikácii tohoto parametra, si trieda nastaví *Quantum[c]* podľa uvedenej hodnoty a nie podľa výpočtu uvedeného v 1.

Radi by sme pripomenuli, že podľa výpočtu v bode 1, ktorý sa deje štandartne pri definovaní HTB tried sa nadbytočná kapacita rozdelí v pomere garantovaných rýchlostí jednotlivých tried. Ak chce užívateľ tento pomer zmeniť, mal by na to používať parameter quantum.

Hierarchia tried a označenie tokov bude rovnaké ako v simulácii 1. Zmeníme len pomer rýchlostí. Pre túto simuláciu nastavíme DWRR pomocou globálneho parametra *r2q* na hodnotu 10 (štandartná hodnota, netreba ani uvádzať) a necháme aby HTB vypočítal *Quantum[c]* pre každú triedu sám (hodnoty sú uvedené na obrázku)



Vlastná simulácia trvala 6 sekúnd a vstupné rýchlosti oboch tokov (tok 1 prechádza cez triedu A, tok 2 cez triedu B) boli nastavené na hodnotu 220 kB/s. Výsledky simulácie sú znázornené na grafoch.

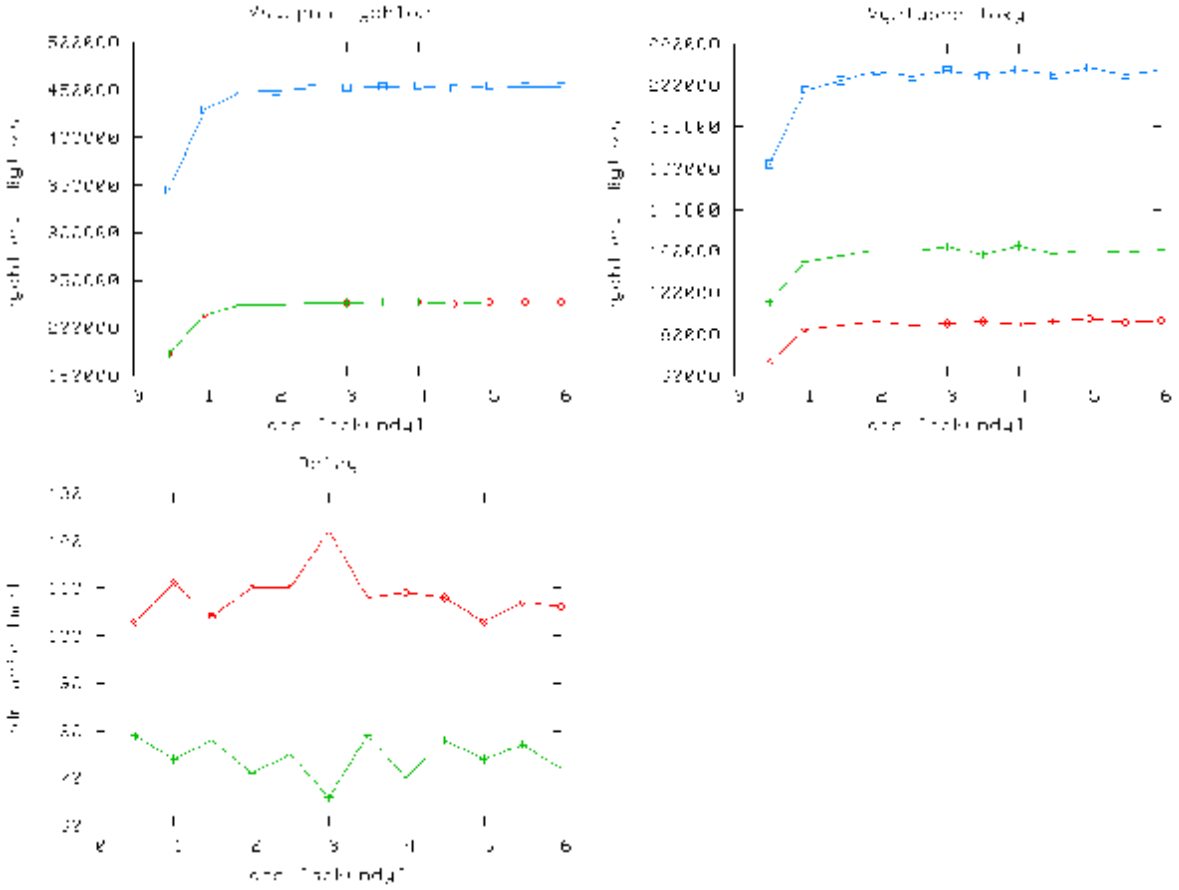


Oba toky získali svoju garantovanú rýchlosť (10kB/s, 90kB/s) a nadbytočnú kapacitu linky (100kB/s) si mali rozdeliť v pomere 1:9. Tok 1 mal dostať kapacitu *20kB/s* a tok 2 *180kB/s*.

Z grafov možno pozorovať, že tok 1 dostal kapacitu okolo *33 kB/s* a tok 2 okolo *172kB/s* – t.j. približne očakávané hodnoty.

4.6.4 Simulácia 4

Táto simulácia bola nastavená rovnako ako simulácia 3. Zmenené boli len hodnoty quantum pre jednotlivé triedy. Pre triedu A bolo nastavené *quantum = 3000 B*, pre triedu B *quantum = 1500 B*. Rozdelenie nadbytočnej kapacity by malo byť v pomere 2:1 v prospech triedy A (t.j. toku 1). Po spustení simulácie sme dostali nasledovné výsledky:



Tok 1 mal získať kapacitu linky 10kB/s (garantovaná rýchlosť) + 66 kB/s (podiel z nadbytočnej kapacity) = *76kB/s* a tok 2 mal získať 90kB/s + 33kB/s = *123 kB/s*. Z výsledkov možno vidieť nasledovné hodnoty: tok 1 *84kB/s*, tok 2 *119kB/s* – znova približne očakávané hodnoty.

5 Záver

Problematika riadenia sieťovej premávky a ovplyvňovania prenosových charakteristík protokolov sieťovej komunikácie je značne rozsiahla. Za hlavný prínos tejto práce považujeme podanie prehľadu o problematike. Prínosom práce je taktiež zmapovanie situácie v operačnom systéme Linux. Je známe, že táto oblasť je veľmi slabo zdokumentovaná, slabo prebádaná a zatiaľ nie veľmi využívaná. Napriek tomu je implementácia kontroly premávky veľmi dobre vyriešená a poskytuje bohaté možnosti pri kontrole prenosu dát. Sústredili sme sa najmä na metódy hierarchického zdieľania linky a algoritmus HTB. Tento je v súčasnosti veľmi obľúbený u správcov sietí, ktorí riadia premávku napríklad pri prístupe užívateľov do internetu alebo chcú zabezpečiť diferenciáciu alebo obmedzenie premávky. Hlavne jednoduchosť a zrozumiteľnosť nastavení a výborná dokumentácia dostupná na autorovej stránke zabezpečili široké uznanie a akceptáciu tohto algoritmu.

Súčasťou tejto práce je aj CD disk spolu so simulačnými skriptami, ktoré sme použili pri vytváraní grafov v simuláciách. Na CD sa nachádzajú aj grafy, ktoré sme v práci neuviedli.

Pre ďalšie smerovanie uvádzam nasledovných niekoľko bodov

- v práci sa venujeme implementácii kontroly premávky iba v operačnom systéme Linux. Bolo by zaujímavé zistiť aj možnosti iných operačných systémov. Z tohto hľadiska možno napríklad spomenúť Windows2000 a možnosti riadenia premávky založené na architektúre Integrovaných služieb (protokol RSVP a podpora diferencovaných služieb) a možnosti centrálnej správy nastavení v doméne. Dobré by bolo zistiť aj možnosti operačného systému Unix FreeBSD, Solaris a iných.
- v práci je spomenutý pojem diferencovaných služieb bez bližšieho vysvetlenia. Jedná sa o architektúru, ktorá sa snaží komplexnejšie poňať riadenie premávky v sieti a garanciu určitých služieb. S touto architektúrou možno spomenúť aj architektúru Integrovaných služieb, ktorá sa snaží o podobné ciele, ale trochu iným spôsobom. Tieto mechanizmy sa začínajú nasadzovať do súčasného internetu, aby zabezpečili určité úrovne služby okrem tej, ktorá je dnes k dispozícii – tj *best effort* kvalita služby, ktorá nezaručuje a negarantuje nič. Popis týchto dvoch architektúr ako aj zmapovanie ich podpory v operačnom systéme Linux by mohol byť aj náplňou ďalšej práce.
- v práci sú spomenuté problémy protokolu TCP s vysokorýchlostnými sieťami a s novými typmi sietí ako sú bezdrátové siete, mobilné siete a rôzne hybridné siete. Zmapovať nové vylepšenia protokolu TCP ako aj preskúmať modely týchto sietí by mohla byť téma pre samostatnú diplomovú prácu.

6 Literatúra

[1] The TCP/IP guide, TCP/IP Transmission control protokol (TCP) http://www.tcpipguide.com/free/t_TCPIPTransmissionControlProtocolTCP.htm
[2] Stevens, W. Richard: TCP/IP Illustrated,Volume 1 The Protocols
[3] Van Jasobson; Karels, Michael J.: Congestion Avoidance and Control, Nov1988
[4] Wright, Gary R.; Stevens, W. Richard: TCP/IP Illustrated, Volume 2 The Implementation
[5] Floyd, S.; Fall K.: Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, August 1999; http://www.aciri.org/floyd/end2end-paper.html
[6] TCP Congestion Control for Next Generation Networks; http://www.hamilton.ie/net/tcp.htm
[7] Scalable TCP, improving performance in highspeed networks; http://www-lce.eng.cam.ac.uk/~ctk21/scalable/
[8] Srikant, R.: TCP Stability and Resource Allocation, University of Illinois; http://www.ima.umn.edu/talks/workshops/1-7-9.2004/srikant/
[9] Mathis, Semke, Madhavi: The Macroscopic Behaviour of the TCP congavoid algorithm; http://www.psc.edu/networking/papers/model_ccr97.ps
[10] Semeria Ch: Supporting differentiated service clases: Queue schdeuling principles; http://www.juniper.net/solutions/literature/white_papers/200020.pdf
[11] Semeria Ch, Supporting differentiated service clases: Active queue memory managemet; http://www.juniper.net/solutions/literature/white_papers/200021.pdf
[12] Semeria Ch, Supporting differentiated service clases: TCP Congestion control mechanizms; http://www.juniper.net/solutions/literature/white_papers/200022.pdf
[13] Feng, W.; Kandlur, D.; Saha, D.; Shin, K.: Blue: A New Class of Active Queue Management Algorithms, U. Michigan, CSE-TR-387-99, April 1999; http://www.thefengs.com/wuchang/work/blue/CSE-TR-387-99.pdf
[14] Sanjeewa Athuraliya; Victor H. Li; Steven H. Low; Qinghe Yin: REM: Active Queue Management, January 15, 2001; http://netlab.caltech.edu/netlab-pub/cbef.pdf
[15] Srisankar Kunniyurz; R. Srikantx: An Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management, December 16, 2002; www.seas.upenn.edu/~kunniyur/papers/avq.pdf
[16] Ratul Mahajan; Sally Floyd; David Wetherall: Controlling High-Bandwidth Flows at the Congested Router; University of Washington, AT&T Center for Internet Research at ICSI (ACIRI), Seattle, WA Berkeley, CA; http://www.cs.washington.edu/homes/ratul/red-pd/paper_icnp.pdf
[17] Sally Floyd; Van Jacobson: Link-sharing and Resource Management Models for Packet Networks, IEEE/ACM Transactions on Networking, Vol. 3 No. 4, August 1995; http://www.icir.org/floyd/papers/link.pdf
[18] Devera, M.: Princip a užití HTB QoS disciplíny
[19] Devera, M.: Hierachical token bucket theory; http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm
[20] Devera, M.: HTB Linux queuing discipline manual - user guide; http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm
[21] Sally Floyd: Notes on Class-Based Queueing: Setting Parameter, LBNL, February 1996

[22] Brown, Martin A.: Traffci control HOWTO; http://linux-ip.net/articles/Traffic-Control-HOWTO/
[23] Kolektív autorov: Linux Advanced Routing & Traffic control; http://lartc.org/
[24] RFC 3290, Kolektív autorov, An Informal Management Model for Diffserv Routers, Máj 2002
[25] Smotlacha, Vladimír: QoS v Linuxu, technická správa; http://www.cesnet.cz/doc/techzpravy/2001/20/
[26] Balliache, Leonardo: Practical QOS; http://www.opalsoft.net/qos/
[27] http://www.docum.org/docum.org/docs/
[28] Almesberger, Werner: Linux Network Traffic Control - Implementation Overview; http://www.almesberger.net/cv/papers.html
[29] Almesberger, Werner: Traffic Control - Next Generation, Reference Manual; http://linux-ip.net/gl/tcng/
[30] Mácha Jakub: Kontorla síťového provozu, Diplomová práce, Masarykova univerzita, 2000
[31] HTB setup script; http://sourceforge.net/projects/htbinit/
[32] CBQ.init traffic management script; http://sourceforge.net/projects/cbqinit/
[33] Hubert, Bert: The Wonder Shaper; http://lartc.org/wondershaper/
[34] MyShaper; http://www.linuxsoft.cz/sw_detail.php?id_item=4466
[35] Almesberger, Werner: Traffic control next generation; http://tcng.sourceforge.net/
[36] Johanna Antila: TCP Performance Simulation using Ns2; http://www.netlab.hut.fi/~jmantti3/pubs/special_study.pdf
[37] The network simulator ns-2; http://www.isi.edu/nsnam/ns/
[38] Web100; http://www.web100.org/
[39] Chunlei Liu; Raj Jain: Enhancement and A New Proposal Based on Congestion Coherence, Approaches ofWireless TCP; http://www.cse.ohio-state.edu/~jain/papers/ftp/hicss.pdf

7 Pojmy a ich anglické ekvivalenty

vysielateľ – sender
prijímateľ – reciever
nevyvárajúci spojenia – connectionless
potvrdzovacia správa ACK - ACK
prenos dát – data transfer
dáta – data
šírka prenosového pásma – bandwidth
preposielanie – retransmission
vyslaný – send
časovač – retransmission timer
interval časovača – retransmission timer interval
CACK, SACK potvrdzovacia politika – CACK, SACK policy
ACK politika – ACKing policy
smerovač – router
linka – link
zahltenie siete – congestion
fronta – queue
vyrovnávajúca pamäť – buffer
kolaps v dôsledku zahltenia - congestion collapse
radenie do výstupných front - output queueing
zhluk – burst
zdržanie - delay
variácia zdržania - jitter
algoritmy QSD - queue schduling disciplines
stupeň férovosti zdieľania linky – fairness
trieda poskytnutej služby – service class
sieťová premávka – traffic
plánovač – scheduler
stanica – host
poskytovateľ pripojenia – provider
systém TC – traffic control
vedro - token bucket
kontrola premávky – traffic control
mechanizmy radenia do front - queueing mechanisms
klasifikátor – classifier
nástroj – tool, utility
smerovacia tabuľka - routing table
sieťové zariadenie – network device
sťahovanie – download
nahrávanie – upload
propagačné oneskorenie – propagation delay

Neprekladané pojmy
policer
shaper
classfull/classless
qdisc

firewall
bandwidth-delay product