Přírodovědecká fakulta
Faculty of Science

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
FAKULTA PŘÍRODOVĚDECKÁ

Katedra informatiky

Efektivita různých variant protokolu TCP ve smyslu jeho použití v oblasti softwarově definovaných sítí a datových center

Bakalářská práce

Autor práce: Jakub Kocum

Vedoucí práce: Ing. Jan Fesl, Ph.D.

České Budějovice
2023

Bibliografické údaje

Kocum J. 2023: Efektivita různých variant protokolu TCP ve smyslu jeho použití v oblasti softwarově definovaných sítí a datových center [The effectiveness of different variants of the TCP protocol in terms of its use in the field of software defined networks and data centers Bc. Thesis, in Czech.] – 41 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Abstract

This work focuses on the effectiveness of TCP variants in terms of congestion control. Since this focus is very comprehensive, this work focuses only on the effectiveness of the amount of data sent via TCP congestion control variants. The main questions that this work deals with are: Which option is the most efficient in terms of sending data? Which option has the least amount of resending data to the network? What is the progression of the size of the congestion window?

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V České Budějovice dne 04.12.2023 Podpis autora**:** Jakub Kocum

Poděkování

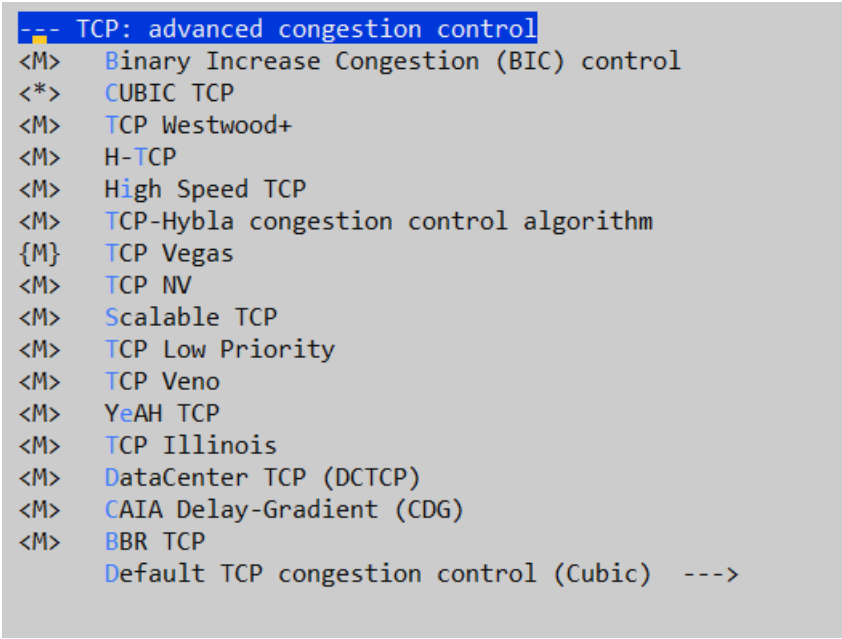
Rád bych na tomto místě poděkoval zejména Ph.D. Ing. Fesl Jan, svému školiteli, za veškerou odbornou podporu a pomoc.

1. Obsah

2. Úvod

Tato bakalářská práce se zabývá problematikou TCP congestion control (CC) a jeho efektivitou odesílání dat do sítě. Především změřením velikosti okna přetížení a poslaní dat v důsledku přetížení sítě.

TCP je v dnešní době již maturovaným protokolem, který se používá ve velkém procentu datových přenosů v internetu. V aktuální době existuje poměrně velké množství variant TCP, které se mezi sebou liší zejména různým mechanismem využitým pro řízení odesílání dat, tzv. congestion control (CC). TCP používá následující varianty CC viz obr.1.



Obrázek 1 – Varianty TCP congestion control v Linux jádře 5.19

Pokud by se neřídilo odeslaní dat do sítě nastávalo by zahlcení sítě. Se zvyšující odezvou sítě se zvyšuje i zpoždění a sníží se celkový výkon sítě. A kdyby to došlo do stavu, kdy zpoždění se zvýší do té míry, že TCP dojde do stavu opětovného přenosu dat tak se situace ještě zhorší. Aby tento stav nenastával tak se implementovaly algoritmy pro řízení přetížení sítě congestion control (CC).

CC se dělí do dvou druhů oba tyto druhy používají frontu. První z těchto druhů má implementovanou konečnou frontu, když se tato fronta naplní tak další packety, které do této fronty přijdou se zahodí. Tento druh CC má konstantní rychlost odesílaní dat do sítě. Druhý druh používá frontu s tokeny to znamená, že pokud ve frontě je token tak se packet odešle do sítě. Pokud ve frontě není token, nedojde k poslání packetu a čeká se na doplnění tokenů do fronty. Tokeny se do fronty

doplňují konstantně v čase. Tento druh CC je flexibilnější, protože nemusí mít konstantní rychlost odesílání packetů do sítě.

3. Cíl práce

Primárním cílem této bakalářské práce bude analýza efektivity jednotlivých variant protokolu TCP, které budou používány pro typické datové přenosy softwarově definovaných sítích v datových centrech. V teoretické části bakalářské práce je popsání mechanismů, které se používají pro řízení CC. V praktické části této práce proběhlo měření variant protokolu TCP na sestaveném modelu pro datový přenos v simulačním programu.

Na daném modelu bylo provedeno měření, výsledky se zaznamenali do této práce. Z těchto dat se bude určovat efektivita konkrétní varianty protokolu TCP z hlediska CC. Vše bude realizováno v datovém centru JU v distribuovaném virtuálním prostředí. V závěru je provedena evaluace všech zjištěných faktů.

4. Teoretická část

4.1. TCP Congestion control

Congestion control využívá okno příjemce tzn. kolik dokáže příjemce přijmout nepotvrzených dat. Jsou dvě okna, jedno na straně příjemce a druhé na straně odesílatele a obě jsou dynamické a jejich velikost určuje množství odeslaných a přijatých dat. Congestion Window (CWND) je okno na straně odesílatele, které má reprezentovat kolik nepotvrzených dat může odesílatel odeslat něž dojde k zahlcení sítě. Odesílatel postupně zvyšuje CWND, avšak nelze ho zvyšovat neomezeně. Hranice, za kterou už je větší pravděpodobnost zahlcení, se označuje SSTHRESH. Odesílatel odesílá vždy jen takové množství nepotvrzených dat, které nepřevyšuje okno inzerované příjemcem ani nepřevyšuje CWND, tj. odesílá jen nejvýše minimální hodnotu CWND nepotvrzených dat [1]. Pokud dojde k SSTHRESH, tak se toto okno CWND zmenší na polovinu. CC rozhodne podle stavu CWND a SSTHRESH jak má dále postupovat.

Když je CWND menší nebo rovno SSTHRESH, jedná se o pomalý start. Pokud varianta CC protokolu TCP vyhodnotí tento stav, pokusí se o odeslání dvojnásobku dat.

V případě, že CWND je již větší než SSTHRESH, pak odeslání dvojnásobku by již pravděpodobně způsobilo zahlcení. V tomto případě se CWND zvyšuje pouze o malou část. Toto „drobné zvětšování" CWND se nazývá algoritmus vyhýbání se zahlcení (Congestion Avoidance Algorithm).

4.2. Datové centrum

Datové centrum je, místo které obsahuje více serverů a tyto servery obsahují velké množství různých dat. Většinou jsou to celé budovy serverů, které uchovávají data, které jsou potom přístupné přes síťové rozhraní pomocí dotazů na data a následnou odpovědí od datového centra. V těchto datových centrech jsou různé typy serverů, které můžeme rozdělit podle toho, jaká data obsahují např. Aplikační servery, databázové servery a atd. Provoz datového centra tvoří dotazy a odpovědi nad nějakými daty. Různá datová centra používají různé varianty CC protokolu TCP pro řízení zahlcení sítě. Právě na tyto varianty CC protokolu TCP se tato bakalářská práce zaměřuje.

4.3. TCP protokoly

4.3.1 Data Center TCP (DCTCP)

Využívá technologii explicitního upozornění na přetížení (ECN) k úpravě okna přetížení odesílatele (CWND) a zpomalení rychlosti odesílání packetů, než se fronta zaplní. Počítáním počtu paketů ACK označených ECN dosahuje DCTCP jemného řízení zahlcení. Technologie ECN je však omezena hardwarem datového centra a problém spočívá v tom, jak správně nastavit práh ECN. DCTCP nesmí být použito pro veřejné sítě bez úpravy.

DCTCP odhaduje zlomek odeslaných bajtů, u kterých došlo k zahlcení. Aktuální odhad je uložen v nové stavové proměnné TCP, DCTCP.Alpha, která je inicializována na 1 a měla by se aktualizovat následovně:

$$DCTCP.Alpha = DCTCP.Alpha * (1 - g) + g * M$$

Kde g je zisk z odhadu, reálné číslo mezi 0 a 1. Výběr g je ponechán na implementaci. M je zlomek odeslaných bajtů, které se setkaly s přetížením během předchozího pozorovacího okna, kde je pozorovací okno zvoleno tak, aby odpovídalo přibližně Round-Trip Time (RTT). Zejména pozorovací okno končí, když byly potvrzeny všechny bajty. K aktualizaci DCTCP.Alpha se používají stavové proměnné TCP definované v [RFC0793] a jsou zavedeny tři další stavové proměnné TCP:

- DCTCP.WindowEnd: práh pořadového čísla TCP, když jedno pozorovací okno končí a druhé má začít; inicializováno na SND.UNA.
- DCTCP.BytesAcked: počet odeslaných bajtů potvrzených během aktuálního pozorovacího okna; inicializováno na 0.
- DCTCP.BytesMarked: počet bajtů odeslaných během aktuálního pozorovacího okna, u kterých došlo k přetížení; inicializováno na 0.

Odhad přetížení na straně odesílatele musí zpracovat přijatelná potvrzení ACK následovně:

1. Vypočítat potvrzené bajty (volby selektivního potvrzení TCP (SACK) [RFC2018] jsou pro tento výpočet ignorovány):

$$BytesAcked = SEG.ACK - SND.UNA$$

2. Aktualizujte odeslané bajty:

$$DCTCP.BytesAcked \mathrel{+}= BytesAcked$$

3. Pokud je nastaven příznak ECE, aktualizujte bajty označené:

$$DCTCP.BytesMarked \mathrel{+}= BytesAcked$$

4. Pokud je číslo potvrzení menší nebo rovno DCTCP.WindowEnd, zastaví se zpracování. Jinak bylo dosaženo konce pozorovacího okna, takže pokračujte v aktualizaci odhadu přetížení následovně:

   4.1. Vypočítat úroveň přetížení pro aktuální okno pozorování:

$$M = \frac{DCTCP.BytesMarked}{DCTCP.BytesAcked}$$

   4.2. Aktualizujte odhad přetížení:

$$DCTCP.Alpha = DCTCP.Alpha * (1 - g) + g * M$$

   4.3. Určí se konec dalšího pozorovacího okna:

$$DCTCP.WindowEnd = SND.NXT$$

   4.4. Vynuluje se počítadla bajtů:

$$DCTCP.BytesAcked = DCTCP.BytesMarked = 0$$

   4.5. Spíše než vždy zmenšit okno přetížení (CWND) na polovinu, jak je popsáno v [RFC3168], odesílatel BY MĚL aktualizovat CWND následovně:

$$cwnd = cwnd * \left(\frac{1 - DCTCP.Alpha}{2}\right)$$

Jak je uvedeno v [RFC3168], DCTCP nereaguje na zahlcení indikace více než jednou pro každé okno dat. Nastavení bit CWR je také podle [RFC3168]. Toto je vyžadováno pro součinnost s klasickými ECN přijímači z důvodu potenciálu chybné konfigurace.[2]

4.3.2 CUBIC

V zásadě používá ACK-clocking k úpravě velikosti okna, nemění algoritmus rychlé obnovy a rychlého opětovného přenosu. V případě, že dojde k události zahození paketu, z důvodu zahlcení sítě, CUBIC začne pracovat. Nastaví maximální velikost okna. Potom CUBIC multiplikativně snižuje velikost okna přetížení v závislosti na rychlosti sítě. Poté CUBIC vstupuje do fáze vyhýbání se přetížení, ve které lze jeho postup růstu okna rozdělit do tří oblastí, přičemž za referenční hodnotu považuje

maximální hodnotu. CUBIC má povoleno udržovat pomalý růst blízko poslední maximální hodnoty, kde došlo k zahození paketu.

Provede multiplikativní snížení okna zahlcení faktorem β, kde β je konstanta poklesu okna a pravidelné rychlé obnovení a opětovné vysílání TCP. Poté, co vstoupí do zamezení přetížení z rychlého zotavení, začne zvětšovat okno pomocí konkávního profilu kubické funkce. Kubická funkce je nastavena tak, aby měla konkávní růst, dokud se velikost okna nedostane na hodnotu W_{max}. Poté se kubická funkce změní na konvexní profil a začíná růst konvexního okna. Tento styl úpravy okna (konkávní a poté konvexní) zlepšuje stabilitu protokolu. Je to proto, že velikost okna zůstává téměř konstantní a tvoří plošinu kolem W_{max}, kde je využití sítě považováno za nejvyšší a za ustáleného stavu se většina vzorků velikosti okna CUBIC blíží W_{max}, což podporuje vysoké využití sítě a stabilitu protokolu. Funkce na růst okna je následující:

$$W(t) = C\ (t - K)^3 + W_{max}$$

Kde C je kubický parametr, t je čas uplynulý od posledního zmenšení okna a K je časový úsek, který výše uvedená funkce potřebuje ke zvýšení W na W_{max}, když nedojde k žádné další ztrátové události, vypočítá se K pomocí následující rovnice:

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

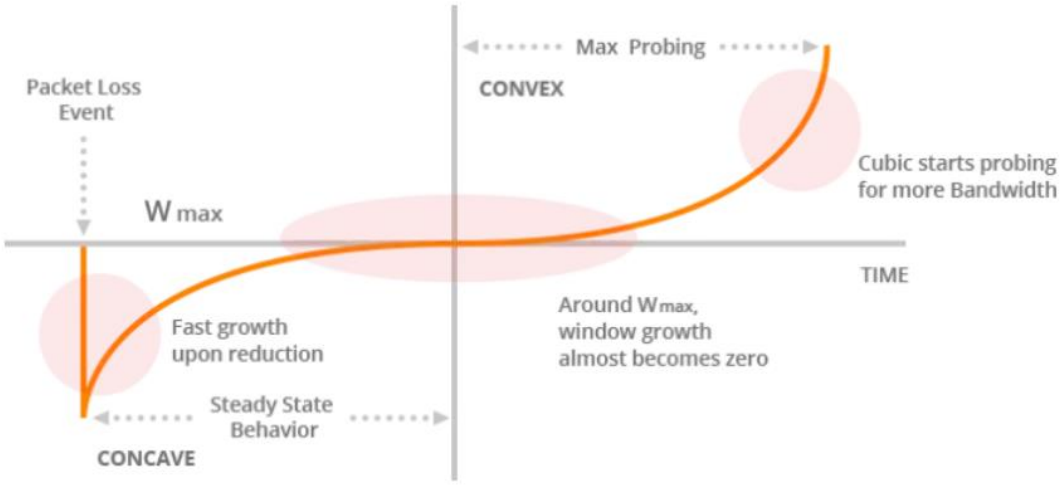
Po obdržení ACK během vyhýbání se přetížení, CUBIC vypočítá rychlost růstu okna během dalšího období RTT . Nastaví W(t + RTT) jako kandidátskou cílovou hodnotu okna přetížení. V závislosti na hodnotě CWND pracuje CUBIC ve třech různých režimech:

1. Pokud je CWND menší než velikost okna, které by (standardní) TCP dosáhl v čase t po poslední ztrátové události, pak je CUBIC v režimu TCP. Velikost okna TCP z hlediska uplynulého času t se vypočítá takto:

$$W_{tcp(t)} = W_{max}(1 - \beta) + 3\frac{\beta}{2 - \beta}\frac{t}{RTT}$$

2. V opačném případě, pokud je CWND menší než W_{max}, pak je CUBIC v konkávní oblasti.
3. Pokud je CWND větší než W_{max}, je CUBIC v konvexní oblasti.

[3] Oblasti jsou znázorněné na obrázku 2.



Obrázek 2 – Okno přetížení je KUBICKÁ funkce s konkávními a konvexními profily

4.3.3 TCP Westwood+

Kde je obecnou myšlenkou použít odhadovanou šířku pásma k nastavení okna přetížení (CWND) a prahu pomalého startu (SSTHRESH) po přetížení. Základní role, kterou hrají CWND a SSTHRESH při kontrole přetížení TCP, je to, že CWND se zvyšuje a snižuje, aby bylo možné sledovat dostupné zpoždění šířky pásma, který by měl představovat SSTHRESH. Přestože TCP Westwood nespoléhá na informace z mezilehlých uzlů, může tyto strategie řazení do fronty, pokud jsou přítomny, využít díky výsledné přesné alokaci šířky pásma flow-by-flow. Celkově se výkon TCP Westwood zlepšuje, pokud je v síti implementována nějaká forma spravedlivého sdílení šířky pásma.[4]

TCP Westwood vypočítává přibližnou dostupnost šířky pásma tak, že na zdroji se přijal ACK v čase t_k, oznamující, že d_k bajty byly přijaty na TCP přijímači. Následující přibližnou šířku pásma spojení můžeme změřit jako:

$$b_k = \frac{d_k}{\Delta_k}$$

$$\Delta_k = t_k - t_k - 1$$

kde $t_k - 1$ je čas, kdy bylo přijato předchozí ACK. Protože k přetížení dochází vždy, když rychlost nízkofrekvenčního vstupního provozu překročí kapacitu spojení, používá se nízkofrekvenční filtr pro průměrování vzorkovaných měření a pro získání nízkofrekvenčních složek dostupné šířky pásma spojení.[5]

4.3.4 TCP Vegas a TCP NV (New Vegas)

Snímá přetížení sítě dříve, než dojde ke ztrátě paketů a okamžitě zmenší velikost okna. TCP Vegas tedy zvládá zahlcení bez ztráty paketů. TCP Vegas používá čas zpáteční cesty pro zvýšení nebo snížení okna přetížení. Měří se očekávaná a aktuální propustnost, jejíž rozdíl je porovnáván s minimální a maximální prahovou hodnotou. Na základě srovnání zvětší, sníží nebo nemění okno přetížení.[6]

TCP-NV (New Vegas) je nástupcem TCP-Vegas optimalizovaným pro použití v datových centrech. Detekuje přetížení dříve, než dojde ke ztrátám paketů. Umožňuje vysokou rychlost a měl by být používán pouze v případě, že všechna připojení používají řízení přetížení TCP-NV.[15]

TCP Veges pracuje s přetížením následujícím způsobem. Nejdříve určí BaseRTT (s) jako minimální naměřenou hodnotu RTT, ta se zvolí na základě šíření zpoždění (jde o RTT segment, který se nachází v přehlceném spojení). Následně je vypočítána očekávaná propustnost (Expected) a aktuální propustnost (Actual), která se počítá pro každé RTT, kde CWND je momentální hodnota okna přetížení a RTT je nejnovější naměřené RTT. Následuje porovnání očekávané hodnoty přetížení a aktuální hodnoty přetížení označené jako diff, která je vždy kladná a slouží k nastavení okna CWND.[7]

$$Expected = \frac{cwnd}{BaseRTT}$$

$$Actual = \frac{cwnd}{RTT}$$

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

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

TCP Vegas má implementované konstanty alfa a beta, podle kterých řídí CWND:

- Pokud diff je menší jak α, zvětší se CWND lineárně během dalšího RTT

- Pokud diff je větší jak β, sníží se CWND lineárně během dalšího RTT
- Pokud je diff mezi α a β, ponechá se CWND nezměněné

$$cwnd = \begin{cases} cwnd + 1 & diff < \alpha \\ cwnd & \alpha \leq diff \leq \beta \\ cwdn - 1 & diff > \beta \end{cases}$$

4.3.5 TCP Veno

Vylepšuje mechanismus řízení přetížení TCP Reno, aby zvládl vícenásobné ztráty paketů přes bezdrátové připojení pomocí technik TCP Vegas, pro vyhodnocení přetížení na úzkém místě, aby bylo možné rozlišit, zda ke ztrátě paketů došlo v důsledku přetížení nebo v důsledku náhodné ztráty. Předurčuje událost zahození paketu v důsledku přetížení, pokud ho najde při přetíženém spojení, jinak považuje ztrátu za náhodnou.

TCP Veno vyhodnocuje změnu hodnoty ve velikosti CWND analytickým porovnáním odhadovaného N a nastavené prahové hodnoty β pro něj. Když hodnota N překročí nastavenou prahovou hodnotu β, byla spotřebována celá šířka pásma sítě. V případě této události TCP Veno rozšíří velikost okna přetížení (CWND) o **1/CWND** za každé nové obdržené potvrzení. To umožňuje TCP Veno zlepšit svůj výkon ve fázi zamezení přetížení ve srovnání s TCP Reno udržováním stabilních spojení po delší dobu. Také když je hodnota N menší než nastavená prahová hodnota β, chová se stejně jako TCP Reno, to znamená, že zvyšuje velikost CWND o **i/CWND** pro každé další nové přijaté potvrzení.

Když je TCP Veno v nekongestivním stavu (N<β), metoda multiplikativního snížení sníží jeho CWND velikost pouze o 1/5 jeho hodnoty, protože je pozorován pokles paketů, je pravděpodobnější, že je to kvůli náhodné chybě.

TCP Veno, stejně jako TCP Reno, snižuje práh pomalého startu na polovinu, když je N větší než β. V opačném případě, když dojde k přetížení (N≥β), algoritmus multiplikativního snížení TCP Veno sníží velikost CWND o polovinu své hodnoty, aby se uvolnila šířka pásma v síti. [7]

4.3.6 HighSpeed TCP a H-TCP

HighSpeed TCP a H-TCP je modifikací mechanismu řízení přetížení TCP Reno pro použití s připojenými TCP s velkými okny přetížení. H-TCP je algoritmus založený na ztrátě paketů, který k řízení okna přetížení TCP používá aditivní zvýšení anebo multiplikativní snížení. Je to jeden z mnoha algoritmů pro zamezení přetížení TCP, který se snaží zvýšit agresivitu TCP na cestách produktu s vysokou šířkou pásma (BDP) při zachování „přívětivosti TCP“ pro malé cesty BDP. H-TCP zvyšuje svou agresivitu (zejména rychlost aditivního nárůstu) s tím, jak se zvyšuje doba od předchozí ztráty paketů.[8]

Konkrétně H-TCP upravuje konvenční TCP následujícím způsobem. Ve vysokorychlostním režimu funkce zvýšení zdroje i následovně $\alpha_i^H(\Delta_i)$ a v režimu nízké rychlosti α_i^L. Přepínání režimů se řídí:

$$\alpha_i = \begin{cases} \alpha_i^L & \Delta_i \leq \Delta^L \\ \alpha_i^H(\Delta_i) & \Delta_i \geq \Delta^L \end{cases}$$

Kde Δ_i je čas, který uplynul od poslední události přetížení, kterou zaznamenal i-tý zdroj, α_i^L je parametr zvýšení pro režim nízké rychlosti (jednotka pro zpětnou kompatibilitu), $\alpha_i^H(\Delta_i)$ je funkce zvýšení pro vysokorychlostní režim, β_i je parametr poklesu jako obvykle a Δ^L je práh pro přepnutí z nízko rychlostního režimu na vysokorychlostní. Funkce zvýšení α_i^H je konstrukční parametr, který lze zvolit podle požadovaných cílů. Ve zbytku tohoto popsání TCP nastavíme α_i^H podle:

$$\alpha_i^H(\Delta_i) = 1 + 10\left(\Delta_i - \Delta^L\right) + \left(\frac{\Delta_i - \Delta^L}{2}\right)^2$$

Tato funkce řídící rychlost, kterou α_i je možné vyladit, aby bylo zajištěno, že H-TCP bude fungovat jako standardní TCP v konvenčních sítích, kde je doba mezi po sobě jdoucími událostmi přetížení malá, a aby se vyvíjel agresivněji ve vysokorychlostních sítích a sítích na dlouhé vzdálenosti, kde může být doba mezi událostmi přetížení dlouhá. Přepínání režimů je založeno na čase od posledního zahození packetu, to znamená, že packety, které jsou již ve vysokorychlostním režimu, nezískají dlouhodobou výhodu oproti novým spouštěným tokům.

Při přetížení funguje úzké místo sítě jako kapacita linky a celková datová propustnost přes linku je dána:

$$R(k)^{-} = \sum_{i}^{n} \frac{w_i(k)}{RTT_{max,i}}$$

kde β je kapacita linky, n je počet síťových zdrojů a $RTT_{max,i}$ je maximální RTT, kterou zaznamená i-tý zdroj. Po backoff je datová propustnost přes odkaz dána:

$$R(k)^{+} = \sum_{i}^{n} \frac{\beta_i w_i(k)}{RTT_{min,i}}$$

Jednoduchý způsob, jak zajistit maximální propustnost, je srovnat obě rychlosti, čímž vznikne následující rovnice pro β_i :

$$\beta_i = \frac{RTT_{min,i}}{RTT_{max,i}}$$

Na základě výše uvedené rovnice je navržena adaptivní strategie, podle níž každý zdroj odhaduje a používá tuto veličinu k určení β_i tak, aby se propustnost shodovala před a po stažení, čímž se zajistí, že se vyrovnávací paměť po přetížení pouze vyprázdní a linka zůstane v provozu na kapacitě. Alternativně může být β_i faktor vyjádřen jako

$$\beta_i(k+1) = min_j\, \beta_i(j) \frac{B_i^{-}(j)}{B_i^{+}(j)}$$

Obě veličiny $B_i^{-}(j)$ a $B_i^{+}(j)$ jsou snadno měřeny z paketů ACK přes RTT. To eliminuje potřebu měřit poměr $RTT_{min,i}/RTT_{max,i}$ přímo a je to přístup, který se v současnosti používá v testovacích implementacích. V souhrnu je adaptivní resetovací algoritmus následující:

- Neustále sledujte hodnotu B_i^{-}.
- Když se naměřená hodnota B_i^{-} posune mimo prahové pásmo, resetujte hodnotu β_i na β_{reset}.

Jakmile se B_i^- vrátí do prahového pásma (např. po konvergenci k novému ustálenému stavu, který lze vypočítat z β_{reset}), znovu je povolen adaptivní algoritmus stažení. $\beta_i = \frac{RTT_{min,i}}{RTT_{max,i}}$ [9]

4.3.7 TCP-BIC

Řízení binárního zvýšení přetížení (BIC) je implementace TCP s optimalizovaným algoritmem řízení přetížení pro vysokorychlostní sítě s vysokou latencí. BIC má jedinečný algoritmus zahlcení okna, který používá binární vyhledávací algoritmus ve snaze najít největší okno zahlcení, které bude trvat maximální dobu. [8]

BIC se přepne z aditivního zvýšení na binární zvýšení, když je vzdálenost od aktuální velikosti okna k cílovému oknu menší než S_{max}. Protože cílové okno je středem mezi W_{max} a aktuální velikostí okna, lze říci, že BIC přepnutí na binární vyhledávání se zvětší, když je vzdálenost od aktuální velikosti okna k W_{max} menší než $2S_{max}$. Pokud je vzdálenost mezi aktuálním oknem a W_{max} menší než $2S_{max}$, nedochází k žádnému aditivnímu zvýšení. Nechť N_1 a N_2 jsou počty kol RTT aditivního zvýšení a zvýšení binárního vyhledávání. N_1 je určeno následovně

$$N_1 = \max\left(\left\lceil \frac{W_{max}\beta}{S_{max}} \right\rceil - 2, 0\right)$$

Potom může být celkové zvýšení okna během zvýšení binárního vyhledávání vyjádřeno jako $W_{max}\beta$-N_1S_{max}. Za předpokladu, že toto množství je dělitelné S_{min}, lze N2 získat následovně.

$$N_2 = log_2\left(\frac{W_{max}\beta - N_1 S_{max}}{S_{min}}\right) + 2$$

Ve výše uvedené rovnici 2 odpovídá prvnímu a poslednímu RTT zvýšení binárního vyhledávání. Během aditivního nárůstu okno roste lineárně se sklonem $1/S_{max}$. Takže celkový počet paketů během aditivního zvýšení, Y_1, lze získat následovně.

$$Y_1 = \frac{1}{2}\left(W_{max}(1-\beta) + W_{max}(1-\beta) + (N_1 - 1)S_{max}\right)N_1$$

Během zvýšení binárního vyhledávání okno roste logaritmicky. Celkový počet paketů během zvýšení binárního vyhledávání, Y_2, lze tedy vyjádřit následovně.

$$Y_2 = W_{max}N_2 - 2(W_{max}\beta - N_1 S_{max}) + S_{min}$$

Celkový počet RTT je $N = N_1 + N_2$ a celkový počet paketů je $Y = Y_1 + Y_2$. Níže jsou uváděny uzavřené výrazy W_{max} pro dva speciální případy.

1. Nejprve předpokládáme, že $W_{max}\beta > 2S_{max}$ a $W_{max}\beta$ je dělitelné S_{max}. Potom $N_1 = W_{max}\beta/S_{max} - 2$. Nyní můžeme získat

$$W_{max} = \frac{-b + \sqrt{b^2 + 4a\left(c + \frac{1}{p}\right)}}{2a}$$

Kde $a = \beta\frac{2-\beta}{2S_{max}}$, $b = log_2\left(\frac{S_{max}}{S_{min}}\right) + \frac{(2-\beta)}{2}$ a $c = S_{max} - S_{min}$.

Průměrná rychlost odesílání R je pak

$$R = \frac{Y}{N * RTT} = \frac{\frac{1}{RTT}(2-\beta)\frac{1}{p}}{\sqrt{b^2 + 4a\left(c + \frac{1}{p}\right) + (1-\beta)b + \frac{\beta(2-\beta)}{2}}}$$

2. V případě, že $W_{max}\beta >> 2S_{max}$, pro pevnou S_{min}, $N_1 >> N_2$. Rychlost odesílání BIC proto závisí hlavně na části lineárního nárůstu a pro malé hodnoty p lze rychlost odesílání aproximovat takto:
3.

$$R \approx \frac{1}{RTT}\sqrt{\frac{S_{max}}{2}\frac{2-\beta}{\beta}\frac{1}{p}} when\ W_{max}\beta >> S_{max}$$

U velmi velkého okna se rychlost odesílání stane nezávislou na S_{min}.

V souhrnu je rychlost odesílání BIC úměrná 1/p d, přičemž $1/2 < d < 1$. S rostoucí velikostí okna se d snižuje z 1 na 1/2. Pro pevné β, když je velikost okna malá, je rychlost odesílání funkcí S_{min} a když je velikost okna velká, funkcí S_{max}.[10]

4.3.8 TCP-Hybla

TCP-Hybla byla navržena s primárním cílem působit proti nespravedlivému výkonu TCP spojení s delšími RTT. TCP-Hybla je určen k překonání výkonnostních problémů, se kterými se setkávají TCP spojení přes pozemní a satelitní rádiová spojení. Tyto problémy pramení ze ztráty paketů v důsledku chyb v přenosovém spoji, které jsou mylně považovány za přetížení, a dlouhého RTT, které omezuje velikost okna přetížení. [8]

Tohoto cíle lze dosáhnout ve dvou krocích: zaprvé tím, že $W_{(t)}$ se stane nezávislým na RTT prostřednictvím modifikace časového měřítka, potom kompenzací vlivu dělení RTT. Oba kroky lze lépe popsat po zavedení normalizovaného zpátečního času ρ definováno jako ρ = RTT/RTT_0, kde RTT_0 je doba zpáteční cesty referenčního spojení, ke kterému se snažíme vyrovnat náš výkon. První krok se provádí vynásobením času ρ (nebo doby uplynulé od dosažení SSTHRESH), čímž se dosáhne $W_{(t)}$ nezávislého na RTT. To druhé vynásobením ρ okna přetížení vyplývající z prvního kroku (včetně původního SSTHRESH, g), aby $B_{(t)}$ bylo nezávislé na RTT. Na závěr máme

$$W^H(t) = \begin{cases} \rho 2^{\frac{\rho t}{RTT}}, & 0 \le t < t_{\gamma,0}, \quad SS \\ \rho\left[\rho\dfrac{t - t_{\gamma,0}}{RTT} + \gamma\right], & t \ge t_{\gamma,0}, \quad CA \end{cases}$$

kde horní index H označuje návrh Hybla. V důsledku modifikace zavedené ve druhém kroku se doba sepnutí $t_{\gamma,0}$, definováno jako čas, kdy okno přetížení dosáhne hodnoty ρ_γ je stejný pro každou RTT, je $t_{\gamma,0}$= RTT_0 $\log_2\gamma$. Vývoj okna přetížení v čase daný vztahem (výše), pro stejné hodnoty RTT. Ze vztahu (výše) je segmentová přenosová rychlost

$$B^H(t) = \frac{W^H(t)}{RTT}$$

TCP Hybla předpokládá následující výraz:

$$B^H(t) = \begin{cases} \dfrac{2^{t/RTT_0}}{RTT_0}, & 0 \le t < t_{\gamma,o}, \quad SS \\ \dfrac{1}{RTT_0}\left[\dfrac{t - t_{\gamma,0}}{RTT_0} + \gamma\right], & t \ge t_{\gamma,0}, \quad CA \end{cases}$$

která je jasně nezávislá na RTT a rovná se segmentové přenosové rychlosti reference standardního připojení TCP. Z tohoto vztahu lze jednoduše odvodit analytické vyjádření počtu přenesených segmentů od začátku spojení TCP Hybla:

$$T_d^H(t) = \int_0^t B^H(\tau)d\tau = \begin{cases} \dfrac{2^{\frac{t}{RTT_0}} - 1}{\ln(2)}, & 0 \le t < t_{\gamma,0} \quad SS \\ \dfrac{\gamma - 1}{ln(2)} + \dfrac{\left(t - t_{\gamma,0}\right)^2}{2RTT_0^2} + \dfrac{\gamma\left(t - t_{\gamma,0}\right)}{RTT_0}, & t \ge t_{\gamma,0} \qquad CA \end{cases}$$

Výkon TCP Hybla již nezávisí na skutečné hodnotě RTT. Zatímco množství přenášených dat u standardních TCP spojení s RTT klesá, u TCP Hybla mají všechna spojení stejný výkon jako referenční. Dynamiku okna zahlcení Hybla lze získat úpravou pravidel aktualizace okna zahlcení takto:

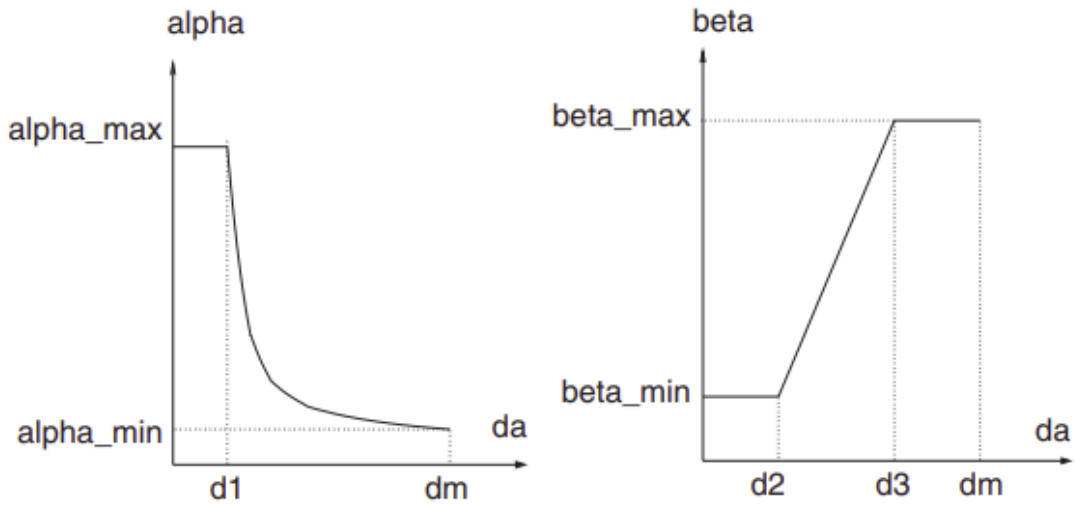
$$W_{i+1}^H = \begin{cases} W_i^H + 2^\rho - 1, & SS \\ W_i^H + \dfrac{\rho^2}{W_i^H}, & CA \end{cases}$$

Dalším důležitým rozdílem je, že ve vztahu výše, TCP Hybla nastaví minimální hodnotu pro ρ na 1. V takovém případě se TCP Hybla jednoduše chová jako standardní TCP pro „rychlá“ připojení (tj. spojení s RTT≤RTT_0) jejich standardní rychlost nárůstu. Kromě vztahu výše musí být počáteční CWND i původní SSTHRESH vynásobeny ρ a také velikost přenosové vyrovnávací paměti (pro zvládnutí většího shluku přenosu TCP Hybla).[11]

4.3.9 TCP-Illinois

TCP-Illinois je zaměřen na vysokorychlostní sítě na dlouhé vzdálenosti. TCP-Illinois je algoritmus založený na ztrátě zpoždění, který používá ztrátu paketů jako primární signál přetížení k určení směru změny velikosti okna a používá zpoždění ve frontě jako sekundární signál přetížení k úpravě tempa změny velikosti okna. [8]

TCP-Illinois je speciální případ algoritmů C-AIMD, které používají následující možnosti pro f1(·) a f2(·):



Obrázek 3. α a β křivky vs d_a

máme:

$$\alpha = f_1(d_a) = \begin{cases} \alpha_{\max} & if\ d_a \le d_1 \\ \dfrac{\kappa_1}{\kappa_2 + d_a} & jinak \end{cases}$$

$$\beta = f_2(d_a) = \begin{cases} \beta_{min} & if\ d_a \le d_2 \\ \kappa_3 + \kappa_4 d_a & if\ d_2 < d_a < d_3 \\ \beta_{max} & jinak \end{cases}$$

Necháme-li f1(·) a f2(·) spojité funkce a tedy $\frac{\kappa_1}{\kappa_2 d_1} = \alpha_{max}$, $\beta_{min} = \kappa_3 + \kappa_4 d_2$ a $\beta_{max} = \kappa_3 + \kappa_4 d_3$. Předpoklad je, že d_m je maximální průměrné zpoždění ve frontě a nechť $\alpha_{min} = f1(d_m)$, pak máme také $\frac{\kappa_1}{\kappa_2 d_m} = \alpha_{min}$. Z těchto podmínek máme

$$\kappa_1 = \frac{(d_m - d_1)\alpha_{min}\alpha_{max}}{\alpha_{max} - \alpha_{min}} \quad and \quad \kappa_2 = \frac{(d_m - d_1)\alpha_{min}}{\alpha_{max} - \alpha_{min}} - d_1$$

$$\kappa_3 = \frac{\beta_{min} d_3 - \beta_{max} d_2}{d_3 - d_2} \quad and \quad \kappa_4 = \frac{\beta_{max} - \beta_{min}}{d_3 - d_2}$$

Tato konkrétní volba je zobrazena v Obr. 3.

Protokol TCP-Illinois nyní bude popsán podrobněji:

- Všechny funkce TCP-New Reno kromě algoritmu AIMD jsou zachovány.
- Ve fázi zamezení přetížení změří odesílatel RTT T pro každé potvrzení a zprůměruje měření RTT za poslední W potvrzení (jeden interval

RTT), aby odvodil průměr RTT T_a. Odesílatel zaznamená maximální a minimální (průměrné) RTT_4, jaké kdy byly zaznamenány, jako T_{max} a T_{min}, a vypočítá maximální (průměrné) zpoždění ve frontě $d_m = T_{max} - T_{min}$ a aktuální průměrné zpoždění ve frontě $d_a = T_a - T_{min}$.

- Odesílatel vybere následující parametry: $0 < \alpha_{min} \leq 1 \leq \alpha_{max}$, $0 < \beta_{min} \leq \beta_{max} \leq 1/2$, $W_{thresh} > 0$, $0 \leq \eta_1 < 1$, $0 \leq \eta_2 \leq \eta_3 \leq 1$. Odesílatel nastaví s $d_i = \eta_i d_m$ (i = 1, 2, 3), vypočítá κ_i (i = 1, 2, 3, 4) a vypočítá hodnoty α a β.
- $\alpha \leftarrow 1$ a $\beta \leftarrow 1/2$ pokud $W < W_{thresh}$.
- Hodnoty κ_i (i = 1, 2, 3, 4) se aktualizují, pokud se aktualizuje T_{max} nebo T_{min}. Hodnoty α a β jsou aktualizovány jednou za RTT.
- $W \leftarrow W + \alpha/W$ pro každé ACK.
- $W \leftarrow W - \beta W$, pokud je v posledním RTT detekována ztráta paketu prostřednictvím trojitého duplikátu ACK.
- Jakmile vyprší časový limit, odesílatel nastaví práh pomalého startu na W/2, vstoupí do fáze pomalého startu a resetuje $\alpha = 1$ a $\beta = 1/2$ a hodnoty α a β se nemění, dokud jeden RTT po skončení fáze pomalého startu neskončí.

4.3.10 TCP Low Priority (TCP-LP)

TCP-Low Priority (TCP-LP) je algoritmus řízení přetížení, jehož cílem je využít pouze nadměrnou šířku pásma sítě ve srovnání se „spravedlivým podílem“ šířky pásma, na který se zaměřuje TCP-Reno. Klíčovými mechanismy jedinečnými pro řízení přetížení TCP-LP jsou použití jednosměrných zpožděních paketů pro indikaci přetížení a zásada zabránění přetížení transparentním TCP. [8]

TCP-LP měří jednosměrné zpoždění paketů a využívá jednoduchou metodu založenou na prahu zpoždění pro včasné odvození přetížení. Označte d_i jako jednosměrné zpoždění paketu se sekvenčním číslem i a d_{min} a d_{max} jako minimální a maximální zpoždění jednosměrného paketu po celou dobu životnosti spojení. Minimální a maximální zpoždění jednosměrných paketů se zpočátku odhadují během slow-start fáze a jsou používány po první ztrátě paketů, tj. ve fázi zabránění zahlcení. Proměnná d_{min} je tedy odhad jednosměrného zpoždění šíření a $d_{max} - d_{min}$ je odhad maximálního zpoždění ve frontě. Dále označte γ jako parametr vyhlazování zpoždění

a sd_i jako vyhlazené jednosměrné zpoždění. Exponenciálně vážený klouzavý průměr se vypočítá jako

$$sd_i = (1 - \gamma)sd_{i-1} + \gamma d_i$$

Včasná indikace přetížení je odvozena z toku TCP-LP, kdykoli vyhlazené jednosměrné zpoždění překročí prahovou hodnotu v rozsahu minimálního a maximálního zpoždění. Jinými slovy, stav časné indikace přetížení je

$$sd_i > d_{min} + (d_{max} - d_{min})\delta$$

kde $0 < \delta < 1$ označuje prahový parametr. ECN používá zvyšující se velikosti fronty k upozornění na toky přetížení před tím, než dojde ke ztrátě výskytu přetížení.[12]

4.3.11 Yet Another Highspeed TCP (TCP-YeAH)

Yet Another Highspeed TCP (TCP-YeAH) je vysokorychlostní algoritmus pro řízení přetížení TCP na straně odesílatele, který k výpočtu okna přetížení používá smíšený přístup ztráty anebo zpoždění. Cílem je dosáhnout vysoké účinnosti, malého RTT a TCP Reno spravedlnosti a odolnosti vůči ztrátě spojení při co nejnižším zatížení síťových prvků. [8]

YeAH-TCP předpokládá dva různé typy režimů: „Fast“ a „Slow“, jako Africa TCP. Během „Fast“ režimu YeAH-TCP zvyšuje okno přetížení podle agresivního pravidla (pravidlo STCP). V „Slow“ režimu se chová jako TCP Reno. O stavu se rozhoduje podle odhadovaného počtu paketů ve frontě úzkých míst. Nechť RTT_{base} je minimální RTT naměřené odesílatelem (tj. odhad zpoždění šíření) a RTT_{min} minimální RTT odhadované v aktuálním datovém okně CWND paketů. Aktuální odhadované zpoždění ve frontě je $RTT_{queue} = RTT_{min} - RTT_{base}$. Z RTT_{queue} je možné odvodit počet paketů zařazených do fronty token jako:

$$Q = RTT_{queue} * G = RTT_{queue} * \left(\frac{cwnd}{RTT_{min}}\right)$$

kde G je goodput. Můžeme také vyhodnotit poměr mezi RTT ve frontě a zpožděním šíření $L = RTT_{queue}/RTT_{base}$, který udává úroveň zahlcení sítě. Všimněte si,

že RTT_{min} se aktualizuje jednou za datové okno dat. Pokud $Q < Q_{max}$ a $L < 1/\varphi$, algoritmus je v režimu „Fast“, jinak je v režimu „Slow“. Q_{max} a φ jsou dva laditelné parametry; Q_{max} je maximální počet paketů, které může jeden tok ponechat ve vyrovnávací paměti. $1/\varphi$ je maximální úroveň zahlcení vyrovnávací paměti s ohledem na BDP. Během „pomalého“ režimu je implementován preventivní algoritmus dekongesce. Dekongesce se používá pouze tehdy, když YeAH-TCP nekonkuruje tokům TCP Reno. Kdykoli $Q > Q_{max}$, okno zahlcení se zmenší o Q a SSTHRESH se nastaví na CWND/2. Protože RTT_{min} se počítá jednou za RTT, je granularita dekongesce jeden RTT. Při startu je ve stavu „Slow“ inicializováno na CWND/2, je zvýšeno o každé RTT v „Slow“ režimu a když je detekována ztráta paketů, je velikost okna poloviční. Proměnná je resetována na aktuální CWND/2 vždy, když je početní rychlost vyšší než prahová hodnota. Současně je počítadlo resetováno na 0.

YeAH-TCP má implementováno postupné snižování okna přetížení, ale pokud dojde k tomu, že přetížení se nezmenšuje tak přepne do způsobu „Fast“ a potom sdílí šířku pásma způsobem TCP Reno.[13]

4.3.12 Scalable TCP

Scalable TCP je jednoduchou změnou tradičního algoritmu řízení přetížení TCP (RFC2581), který dramaticky zlepšuje výkon TCP ve vysokorychlostních rozlehlých sítích. Scalable TCP změní algoritmus pro aktualizaci okna přetížení TCP na následující: *CWND:=CWND+0,01* pro každé přijaté potvrzení, když není v obnově ztráty dat, potom *CWND:=0,875*CWND* při každé události ztráty. [8]

Zobecněný algoritmus pro aktualizaci okna Scalable TCP reaguje na každé potvrzení přijaté v době zpáteční cesty, ve které nebylo při aktualizaci zjištěno přetížení.

$$cwnd_r \mapsto cwnd_r + a$$

kde a je konstanta s $0 < a < 1$. Dále, při první detekci přetížení v dané zpáteční době se okno přetížení změní o

$$cwnd_r \mapsto cwnd_r - \lceil b * cwnd_r \rceil$$

kde b je konstanta s 0 < b < 1. Doby obnovy ztráty paketů pro tradiční TCP spojení jsou úměrné velikosti okna připojení a době zpětného přenosu. Scalable TCP má doby obnovy ztráty paketů, které jsou úměrné pouze zpáteční době připojení; tato invariance k velikostem spojení umožňuje Scalable TCP překonat tradiční TCP ve vysokorychlostních rozlehlých sítích. Vlastnost škálování platí pro libovolnou volbu konstant a a b tyto konstanty určují omezení implementace a nasazení.

Algoritmus aktualizace okna zahlcení uvádí do vztahu velikost okna zahlcení s rychlostí signalizace mezi koncovými body prostřednictvím křivky odezvy. Zobecněný algoritmus Scalable TCP má křivku odezvy, kterou lze aproximovat pro malé rychlosti poklesu mezi koncovými body. To lze odvodit zvážením velikosti okna zahlcení v rovnováze pomocí modelu diferenciální rovnice CWND nebo očekáváním stochastického modelu CWND.

$$cwnd_r \approx \frac{a}{b}\frac{1}{P_r}$$

Aby byla zajištěna spojitá a klesající křivka odezvy, musí křivka odezvy Scalable TCP procházet bodem, který dává následující omezení pro a a b

$$\frac{a}{b} = p_l * lwnd \approx \sqrt{1{,}5}p_l$$

Počet volných proměnných je nyní snížen na jednu; volba b opravuje a. Okamžitá rychlost připojení TCP se pohybuje kolem střední hodnoty, což jí dává podíl na dostupné kapacitě. Koeficient rozptylu pro okamžitou rychlost odesílání je

$$CoV(x_r) = CoV\left(\frac{cwnd_r}{T_r}\right) \sim \sqrt{\frac{b}{2}}$$

To naznačuje, že b by mělo být zvoleno co nejmenší, aby se snížila okamžitá změna rychlosti, což je závěr, který souhlasí s intuitivními argumenty založenými na dobách obnovy ztráty paketů. Aby algoritmus neměl variace rychlosti větší než tradiční TCP, takže b by mělo splňovat $b \leq ½$. Algoritmus Scalable TCP reaguje na události zahlcení maximálně jednou za dobu zpáteční cesty. Proto je nutné, aby cyklus roztahování a smršťování okna trval déle než doba zpáteční.[14]

4.3.13 BBR - Bottleneck Bandwidth and RTT congestion control algorithm

Tento algoritmus byl vyvinut ve společnosti Google a nasazen pro použití na YouTube. Jedná se o algoritmus založený na modelu, který měří šířku pásma úzkého hrdla a dobu šíření. Když BBR začne odesílat datový tok, nejprve se pokusí zvýšit přenosovou rychlost. Poté, co ze ztráty potvrzení (ACK) zjistí, že pásmo je plné, vyprázdní frontu a změní stav na šířku pásma sondy. Klíčovým prvkem algoritmu je sledování doby zpáteční cesty (RTT) během přenosu. Toto je hlavní stav BBR, tráví v něm většinu času. Vzhledem k tomu, že šířka pásma úzkého hrdla sítě a doba šíření se mohou v čase dynamicky měnit, musí je algoritmus periodicky přepočítávat. Takže z ACK přepočítá obousměrný RTT cesty (RTProp). Pokud není aktualizován, přejde BBR do stavu sondy RTT. V tomto stavu nastaví okno TCP na malou velikost, aby zkontrolovalo, zda je možné nižší RTT. Nevýhodou BBR je chybějící funkce škálovatelnosti a ukončení souběžných toků založených na ztrátách v ekosystému toků TCP.[15]

TCP aktuálně sleduje RTT (časový interval od odeslání datového paketu do jeho potvrzení), protože je vyžadován pro detekci ztráty. Kdykoli t,

$$RTT_T = RTprop_t + \eta_t$$

kde $\boldsymbol{\eta} \geq 0$ představuje „šum“ zaváděný frontami podél cesty, strategií zpožděného potvrzení přijímače, agregací potvrzení atd. RTprop je fyzická vlastnost cesty připojení a mění se pouze při změně cesty. Vzhledem k tomu, že ke změnám dráhy dochází na časových škálách >> RTprop, nestranný a účinný odhad v čase T je

$$\widehat{RTprop} = RTprop + \min(\eta_t) = \min(RTT_t) \qquad \forall t \in [T - W_R, T]$$

Tj. běžící min v časovém okně WR (které jsou obvykle desítky sekund až minut).

Na rozdíl od RTT nic ve specifikaci TCP nevyžaduje implementace pro sledování šířky pásma úzkého místa, ale dobrý odhad vyplývá ze sledování rychlosti doručení. Když potvrzení pro nějaký paket dorazí zpět k odesílateli, předá RTT tohoto paketu a oznámí doručení datového letu, když paket odejde. Průměrná rychlost doručení mezi odesláním a potvrzením je poměr doručených dat k uplynulému času: rychlost

doručení = Δ*delivered*/Δt. Tato míra musí být ≤ míra úzkého hrdla (množství doručení je přesně známo, takže veškerá nejistota je v Δt, což musí být ≥ skutečný interval doručení; poměr tedy musí být ≤ skutečná rychlost doručení, což je naopak, horní ohraničená kapacitou úzkého hrdla). Proto je maximální rychlost doručení v okně efektivním a nezaujatým odhadem BtlBw:

$$\widehat{BtlBw} = \max(deliveryRate_t) \qquad \forall t \in [T - W_R, T]$$

kde časové okno WB je obvykle šest až deset RTT. TCP musí zaznamenat čas odchodu každého paketu pro výpočet RTT. BBR rozšiřuje, že zaznamenává s celkovými dodanými daty, takže každý příjem potvrzení poskytuje měření RTT i rychlosti doručení, které filtry převádějí na odhady RTprop a BtlBw.[17]

4.3.14 CDG - CAIA Delay-Gradient congestion control algorithm

CDG funguje na základě gradientu zpoždění. Gradient zpoždění se používá jako indikátor přetížení a řízení toku je založeno na ztrátě segmentů. Je citlivý a snižuje množství odesílaných segmentů, když jsou pakety ztraceny kvůli zahlcení, ale toleruje pakety ztracené kvůli událostem v síti bez zahlcení. Algoritmus používá pro vyhlazení klouzavý průměr gradientů minimálních a maximálních hodnot RTT ve specifikovaném okně.[15]

CDG používá maximální RTT (τ_{max}) zaznamenanou v naměřeném intervalu RTT spolu s minimální RTT (τ_{min}) zaznamenanou v naměřeném intervalu RTT. Na základě toho se udržují dvě míry gradientu (změna měření RTT na interval RTT), kde n je n-tý interval RTT:

$$g_{min,n} = \tau_{min,n} - \tau_{min,n-1} \qquad g_{max,n} = \tau_{max,n} - \tau_{max,n-1}$$

Maximální a minimální měření jsou méně hlučná než měření RTT na paket. Přesto aplikujeme vyhlazování klouzavým průměrem rovnice

$$\overline{g_n} = \sum_{i=n-a}^{n} \frac{g_i}{a}$$

kde $g_i = g_{min,i}$ pro výpočet $\bar{g}_{min,n}$ nebo $g_i = g_{max,i}$ pro výpočet $\bar{g}_{max,n}$, které lze vypočítat iterativně pomocí rovnice níže (kde a je počet vzorků v okně klouzavého

průměru). Při provozu v režimu pomalého startu se $g_{min,n}$ a $g_{max,n}$ používají bez vyhlazování pro včasnější odezvu na rychlé zvýšení datového okna.

$$\overline{g_n} = \overline{g}_{n-1} + \frac{g_n - g_{n-a}}{a}$$

V režimu zabránění přetížení aktualizuje CDG okno přetížení (w) jednou za každý RTT podle rovnice

$$w_{n+1} = \begin{cases} w_n\beta & X < P[backoff] \wedge \overline{g_n} > 0 \\ w_n + 1 & otherwise \end{cases}$$

kde w je velikost okna zahlcení TCP v paketech, n je n-tý RTT, X = [0, 1] je rovnoměrně rozložené náhodné číslo a β je multiplikativní faktor snížení. Indikace přetížení s gradientem zpoždění tedy způsobí, že CDG ustoupí maximálně jednou za dva intervaly RTT. V režimu pomalého startu se w zvyšuje stejně jako u New Reno. Rozhodnutí snížit w a vstoupit do režimu zabránění přetížení se provádí na RTT podle rovnice výše nebo na ztrátu paketů jako v New Reno, podle toho, co nastane dříve.[16]

5. Praktická část

5.1. Analýza

Pro tuto bakalářskou práci se udělá měření přenesených dat v závislosti na congestion control, která nabízí systém Linux ve simulačním programu GNS3, kde bude probíhat měření efektivity TCP congestion control. Měřením TCP congestion control se zjistí, jaké protokoly congestion control nejlépe zvládají přetížení sítě z hlediska přenesených dat a podle toho se vyhodnotí závěr této bakalářské práce. V GNS3 se vytvoří testovací topologie sítě, na které bude probíhat měření účinnosti congestion control.

5.2. GNS3 simulační program

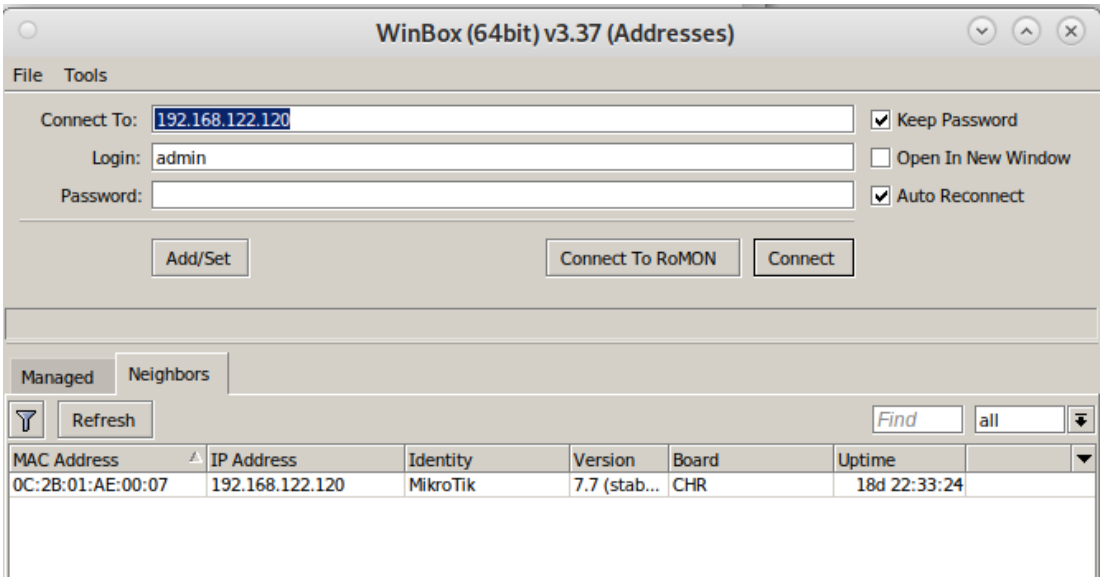
V tomto programu se vytvoří topologie sítě, kde se bude následně měřit úspěšnost congestion control, které jsou popsány výše.

Topologie se bude skládat z prvků:

- Router
- Switch
- Počítače s operačním systémem Linux Debian verze 11.6
- Cloud pro připojení topologie k internetu pro konfiguraci routeru a stažení všech věcí které jsou pro měření potřeba

5.2.1 Ukázková konfigurace router MikroTik

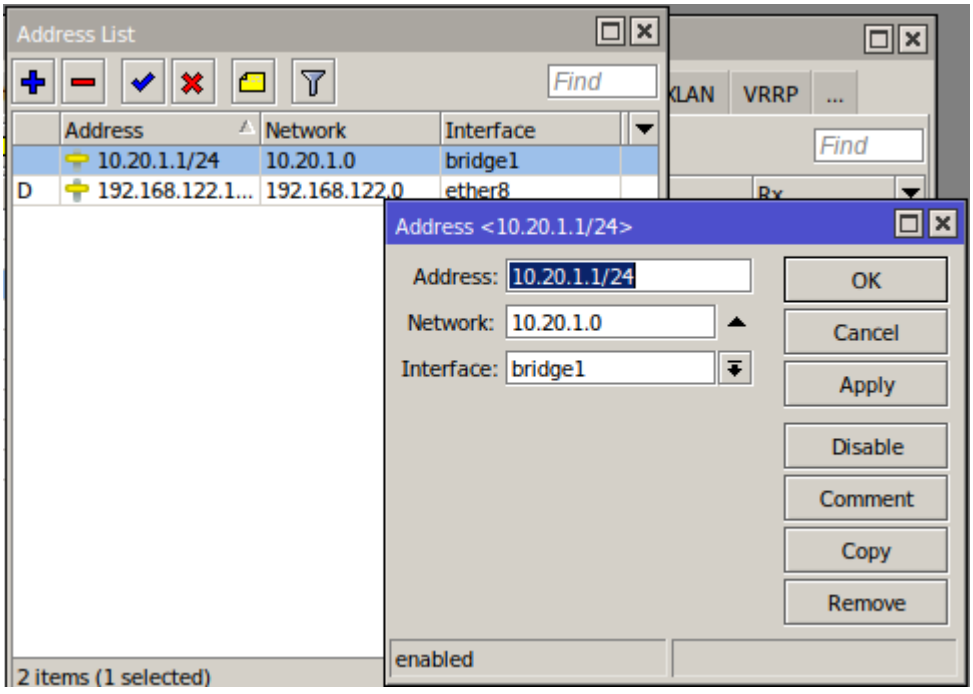
Router MikroTik se konfiguruje pomoci programu WinBox. Nejdřív se připojí k routeru přes program WinBox viz obr.



Obrázek 4 připojovací okno k MikToku

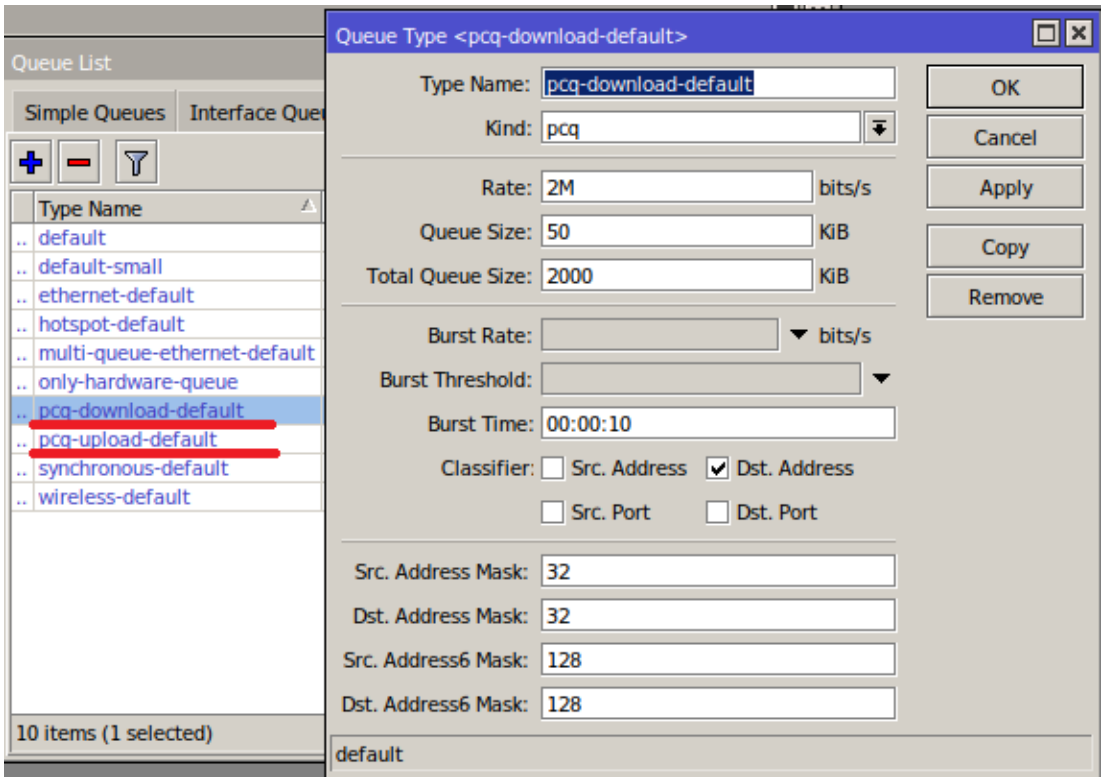
V konfiguraci se vytvoří bridge pro propojení dvou virtuálních počítačů na routeru, pro měření bude stačit bridge s defaultním nastavením.

Když už je vytvořen bridge do nastavíme síť v konfiguraci „IP address“ kde se vytvoří síť viz obr.



Obrázek 5 MikTok nastaveni adresniho rozsahu pro bridge

Nastaví se IP rozsah s maskou sítě a interface se zvolí bridge který byl vytvořen. Další se nastaví queue pro interfaces které propojují dva virtuální počítače. Pro měření se použijí queues viz obr. s konfigurací.



Obrázek 6 nastavení propustnosti linky

Tyto queue jsou nastaveny se stejným ratem a přiřadí se k interfacům na kterých jsou připojeny virtuální počítače.

5.2.2 Ukázková konfigurace počítače s OS Debian verze 11.6

Pro nastavení PC je třeba nastavit IP adresu podle sítě nastavené na routeru. Pokud je na routeru nastavená síť 192.168.1.0/24 tak PC musí mít IP adresu z této sítě. Pomocí textového editoru změníme nastaveni IP adresy v „etc/network/interfaces“ kde se přidají následující řádky viz obr.

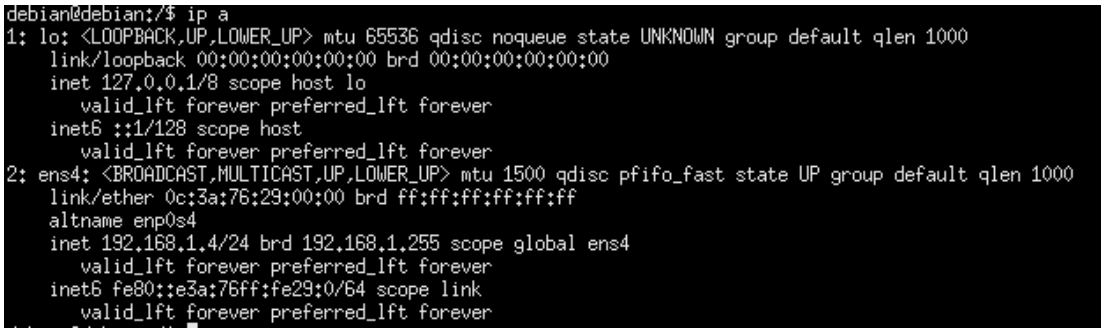
```
auto lo
iface lo inet loopback

auto ens4
iface ens4 inet static
address 192.168.1.4
netmask 255.255.255.0
gateway 192.168.1.1
dns-nameservers 8.8.8.8
```

Obrázek 7 interface v Debian

Potom je třeba restartovat službu pro internet pomoci příkazu „sudo systemctl restart NetworkManager.service“, který nastaví statickou adresu pro PC. Kontrolu, zda se IP

adresa nastavila bude provedeno pomocí příkazu „ip a“ která zobrazí nastaveni IP adres viz obr.



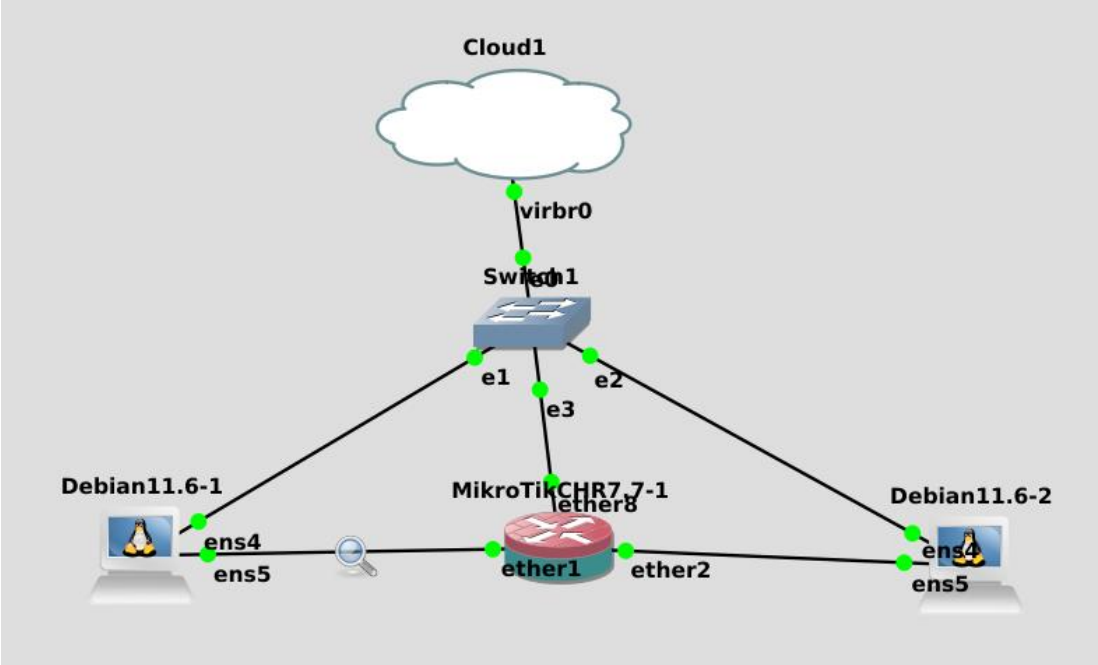
Obrázek 8 Nastavení IP adresy

Pomocí příkazu „ping“ na gateway se zkontroluje, že PC je v síti a že cesta ke gateway funguje.

Pro získání dalších congestion control je třeba zkompilovat linuxové jádro. Potom pomocí příkazu „modprobe {algoritmus_congestion_control}“ povolíme jednotlivé congestion control.

5.3. Topologie sítě pro měření

Ve simulačním programu GNS3 se vytvoří topologie sítě, na které bude prováděno měření pomocí scriptu v jazyce Python. Tato topologie je následující:



Obrázek 9 Topologie sítě v GNS3

Skládá se ze dvou virtuálních počítačů s operačním systémem Linux Debian a routeru MikroTik který je pro toto měření ideální, jednotlivé věci jsou v tomto routeru konfiguratelné.

Dále v této topologii je i připojení k reálnému internetu pro doinstalovaní všech potřebných věcí pro virtuální počítače hlavně linuxové jádro, aby se mohli změřit všechny varianty TCP congestion control, které jsou popsány výše. Dále toto spojení slouží i k připojení k routeru MikroTik pro nakonfigurování toho routeru pro účel testování.

5.4. Software

V této bakalářské práci bude použit software typu klient-server, kde serverová část bude provozován na virtuálním PC ve GNS3 a klient bude provozován na dalším virtuálním počítači v síti ve GNS3, který bude posílat dotazy na server. Klient bude zjišťovat a ukládat statistiku socketu, kde je velikost okna přetížení a další údaje o socketu. Během měření se všechna potřebná data budou ukládat do souboru, který

bude potřeba na konečnou analýzu a vyhotovení závěru. Jako software pro klienta a server je použit program „iperf3“, který zajistí neustálou komunikaci během měření.

5.4.1 Přepínání TCP Congestion Control

Pro přepínání mezi congestion control bude použit nasledujíci přikaz: „sysctl -w net.ipv4.tcp_congestion_control={cc_algorithm}“ kde {cc_algorithm} představuje jeden z výše popsaných protokolů.

5.4.2 Návrh softwaru

5.4.2.1 Návrh softwaru pro získaní statistiky socketu

Tento program bude získávat statistiku socketu a bude tyto údaje ukládat do souboru pro konečnou analýzu. K tomu bude stačit jednoduchý pythonovský script, který bude pouštět linuxoví nástroj ss (socket statistic) s parametrem potřebným pro získaní dat pro měření. Tento script bude spuštěn a 3600krát zavolá nástroj ss. Výstup z tohoto linuxového nástroje se bude ukládat do souboru.

5.4.2.2 Klient server

Pro posílání dat bude použit nástroj „iperf3“, který zajistí jak serverovou část, tak klientskou část. Tento nástroj posílá periodicky data na server a poslouží pro měření. Pro měření se použije parameter tohoto nástroje, který zajistí že se data budou posílat po určitou dobu. Má tu výhodu že posílá vždy maximální množství dat která sítová linka může poslat a není třeba testovacího souboru.

5.5. Analýza naměřených hodnot

Naměřená data uložená v souborech se vyhodnotí a podle dat bude vyhodnocen závěr.

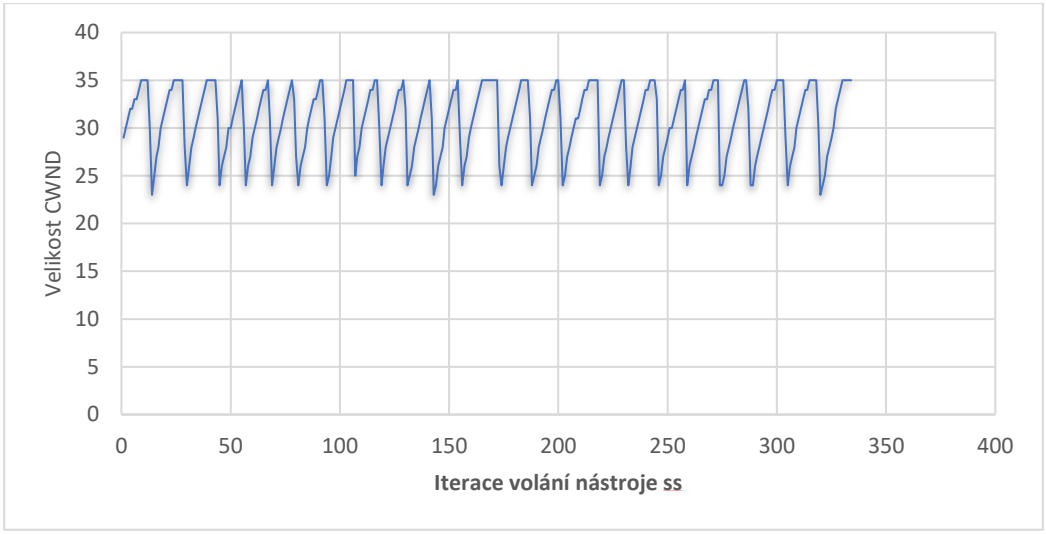
Naměřená data jsou dána do grafu pro lepší čitelnost naměřených dat. Data zobrazují velikost okna přetížení při zavolání nástroje ss. Pro viditelnější průběh je v grafu pouze vybraný interval z naměřených dat.

Dále se vezme z naměřených dat, kolik bytů bylo posláno za měření pro zjištění kolik dat dokázal protokol TCP se zvoleným CC poslat za určitou dobu.

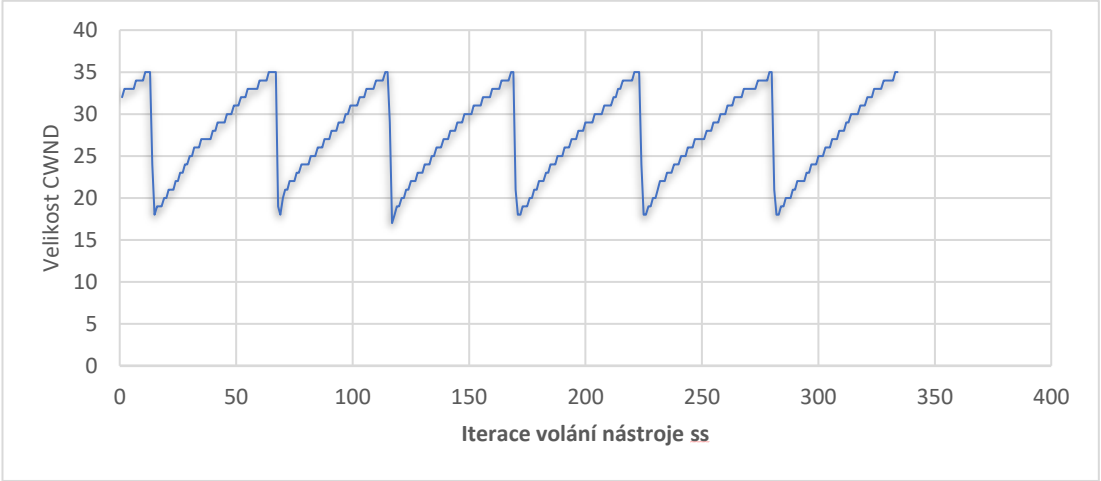
Každý protokol CC byl měřen padesátkrát a jejich výsledky a průběhy byli téměř totožné. Každé měření trvalo 10 minut a celkově se jeden protokol CC měřil 8 hodin a 20 minut.

5.5.1 Průběh velikosti CWND na základě měření různých variant protokolu TCP

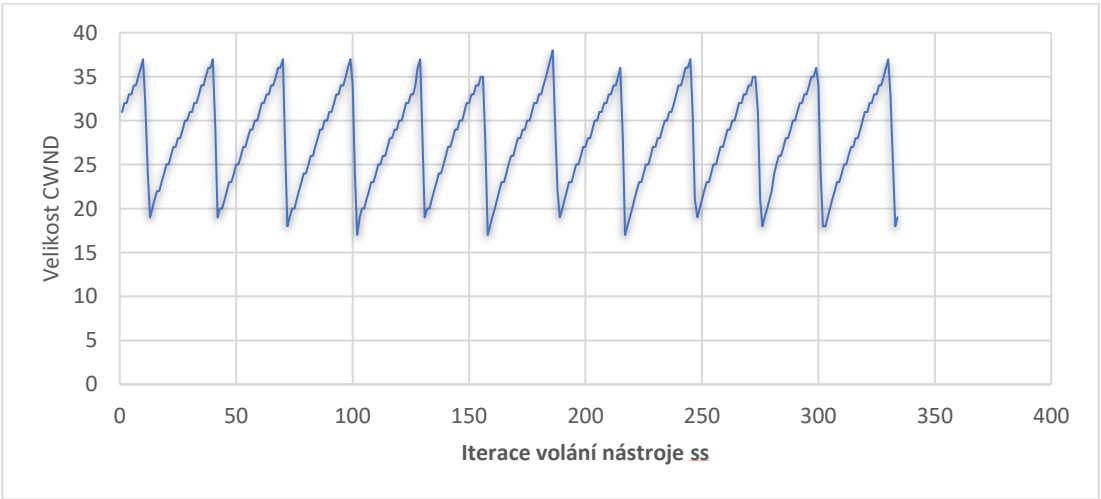
V grafech jsou vyobrazeny průběhy velikostí CWND za jednu iteraci nástroje socket statistics během měření. Jak je vidět na grafech průběhy velikosti CWND je různé v závislosti na variantě TCP.



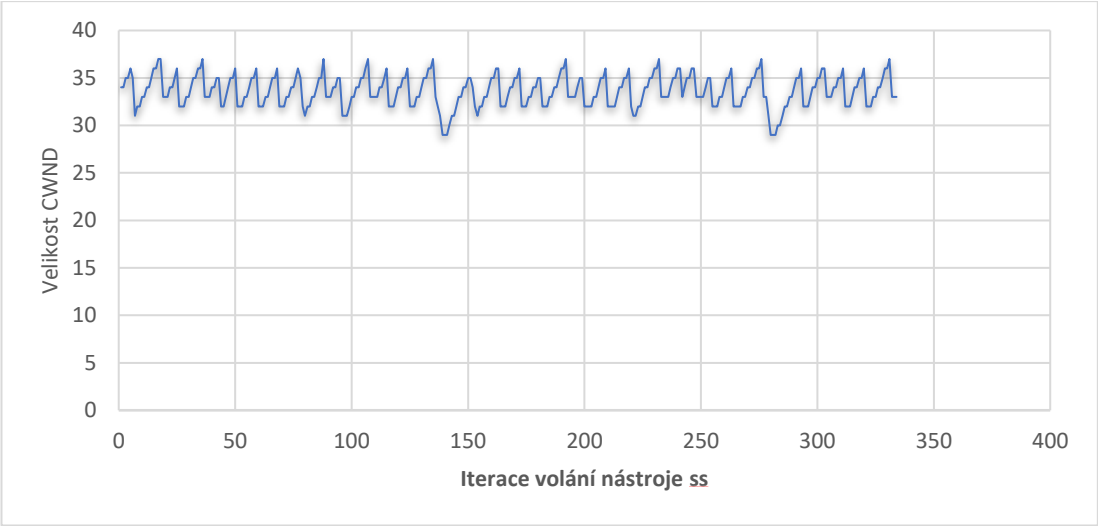
Obrázek 10 Cubic - průběh velikosti okna CWND za jednu iteraci nástroje ss



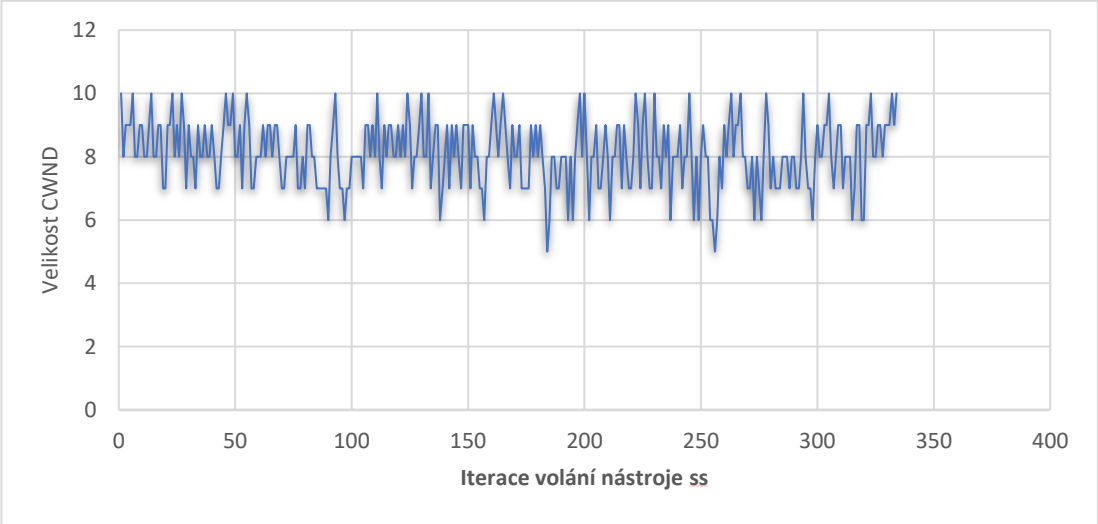
Obrázek 11 Hybla - průběh velikosti okna CWND za jednu iteraci nástroje ss



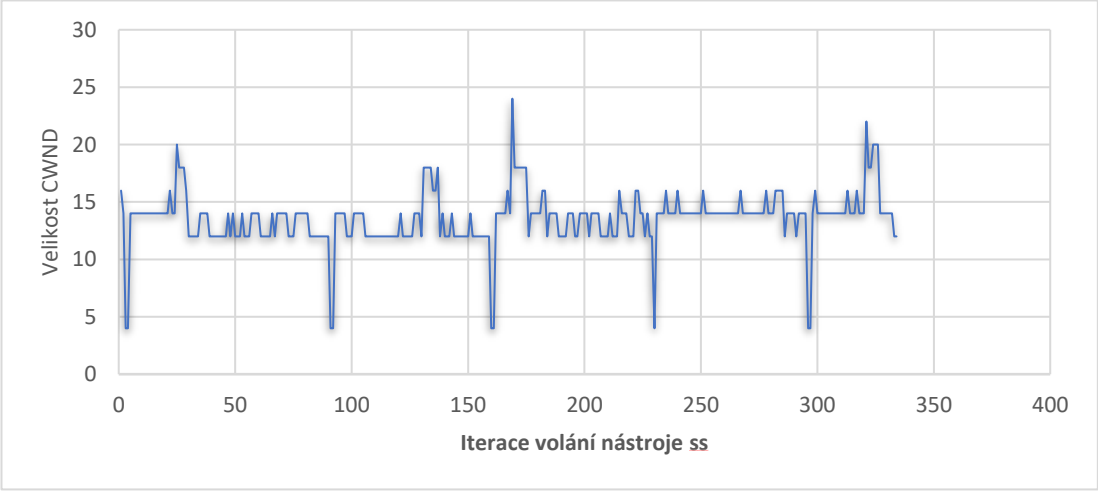
Obrázek 12 H-TCP - průběh velikosti okna CWND za jednu iteraci nástroje ss



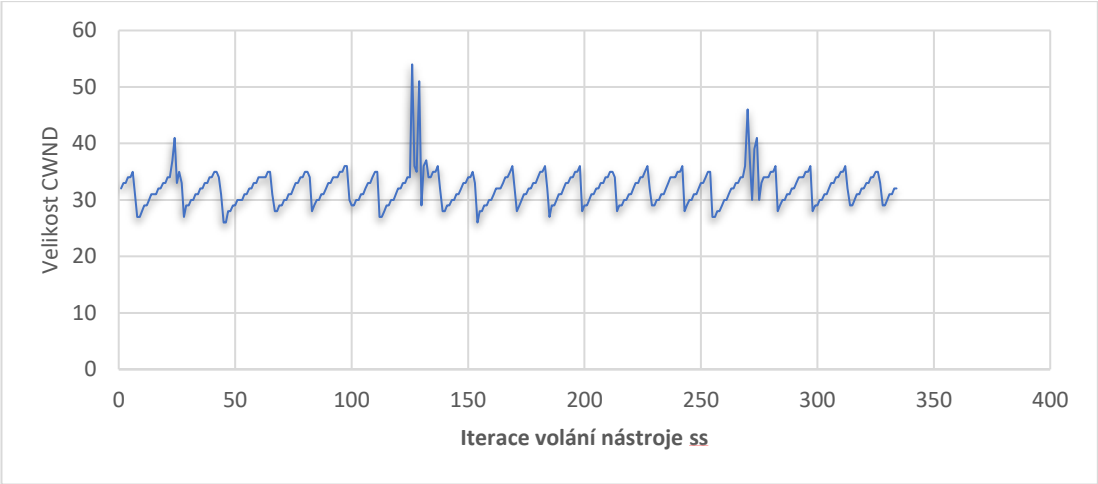
Obrázek 13 Scalable - průběh velikosti okna CWND za jednu iteraci nástroje ss



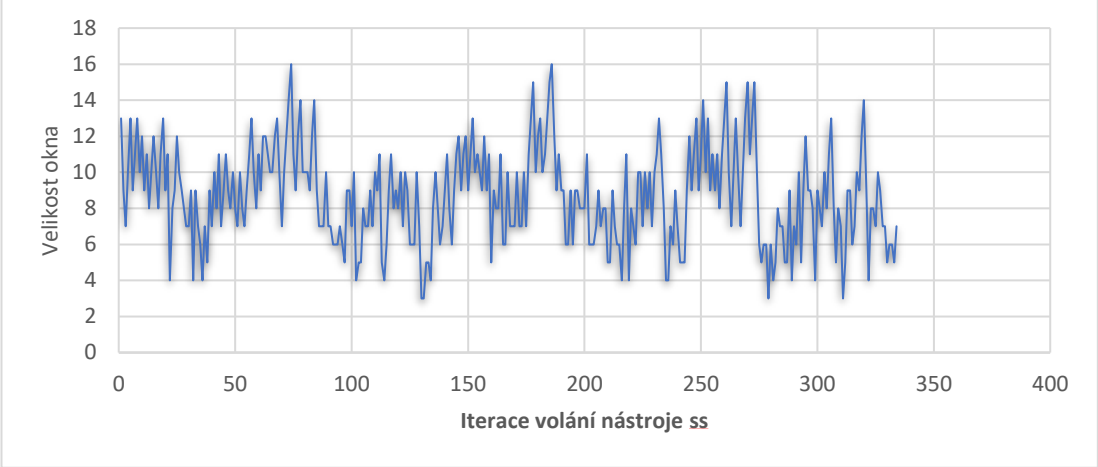
Obrázek 14 Vegas - průběh velikosti okna CWND za jednu iteraci nástroje ss



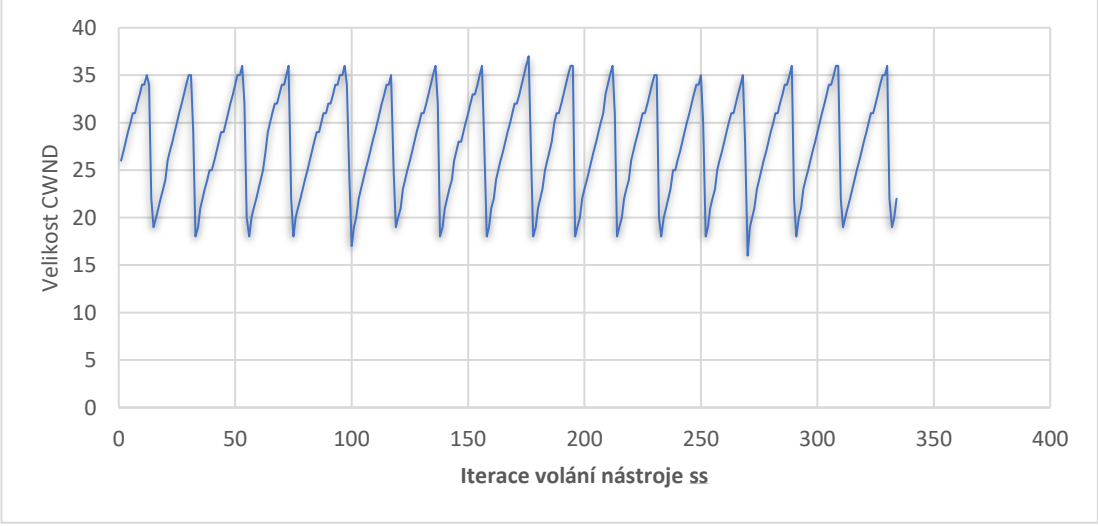
Obrázek 15 BBR - průběh velikosti okna CWND za jednu iteraci nástroje ss



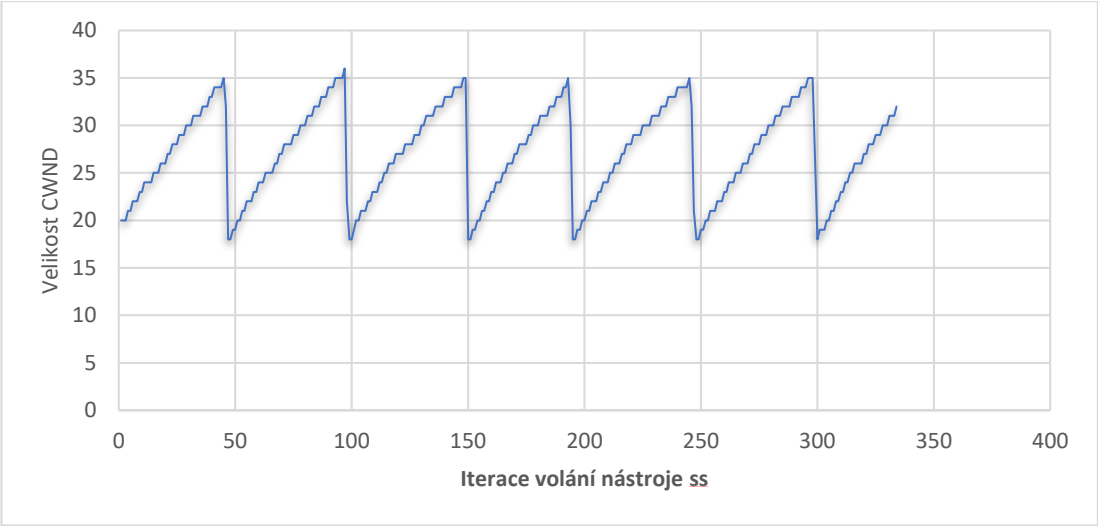
Obrázek 16 BIC - průběh velikosti okna CWND za jednu iteraci nástroje ss



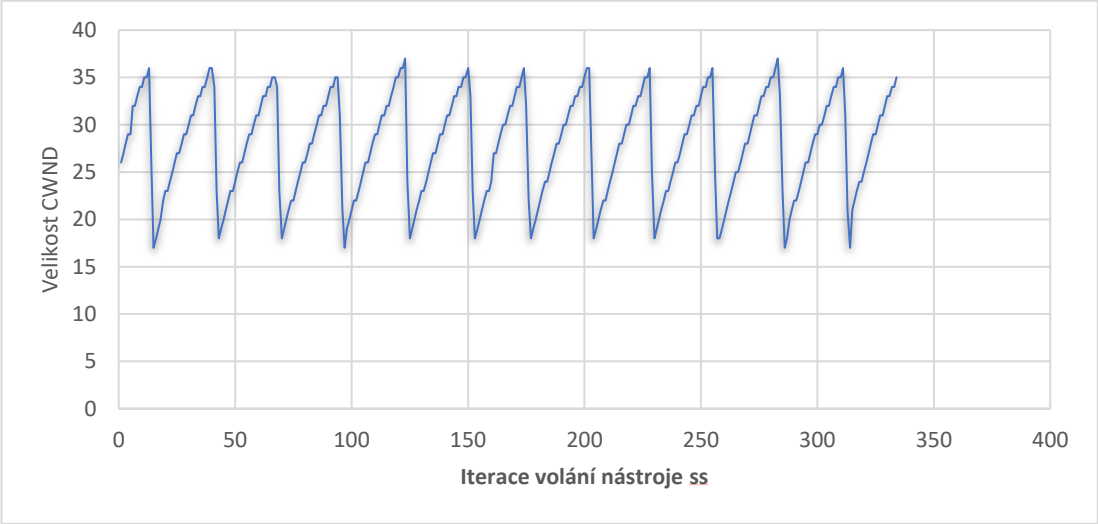
Obrázek 17 CDG - průběh velikosti okna CWND za jednu iteraci nástroje ss



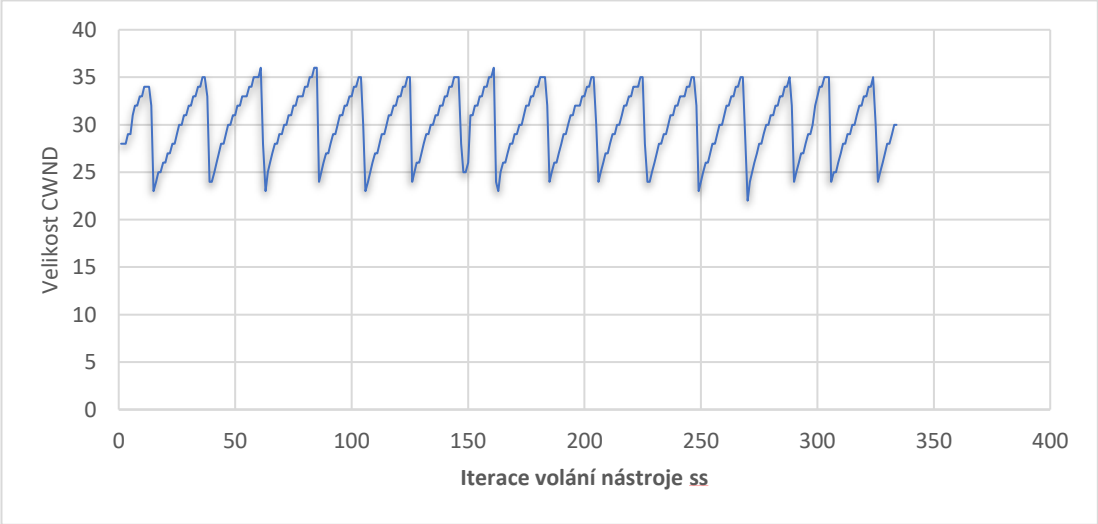
Obrázek 18 DCTCP - průběh velikosti okna CWND za jednu iteraci nástroje ss



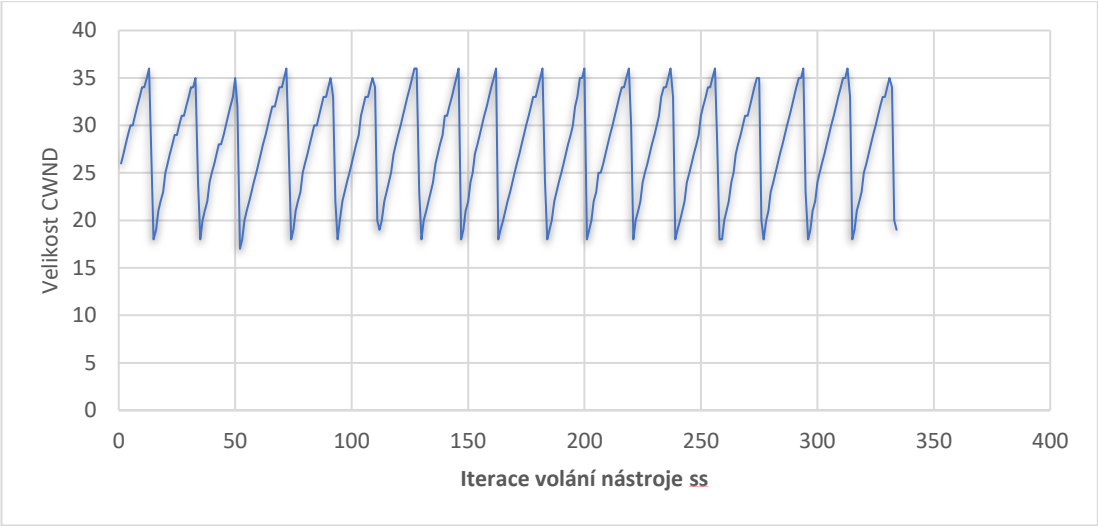
Obrázek 19 Highspeed - průběh velikosti okna CWND za jednu iteraci nástroje ss



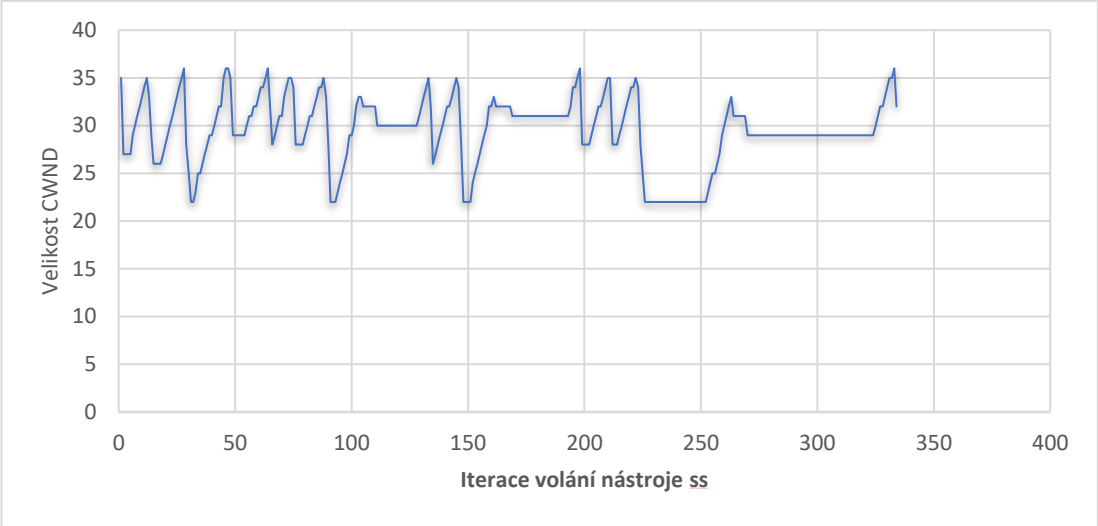
Obrázek 20 Reno - průběh velikosti okna CWND za jednu iteraci nástroje ss



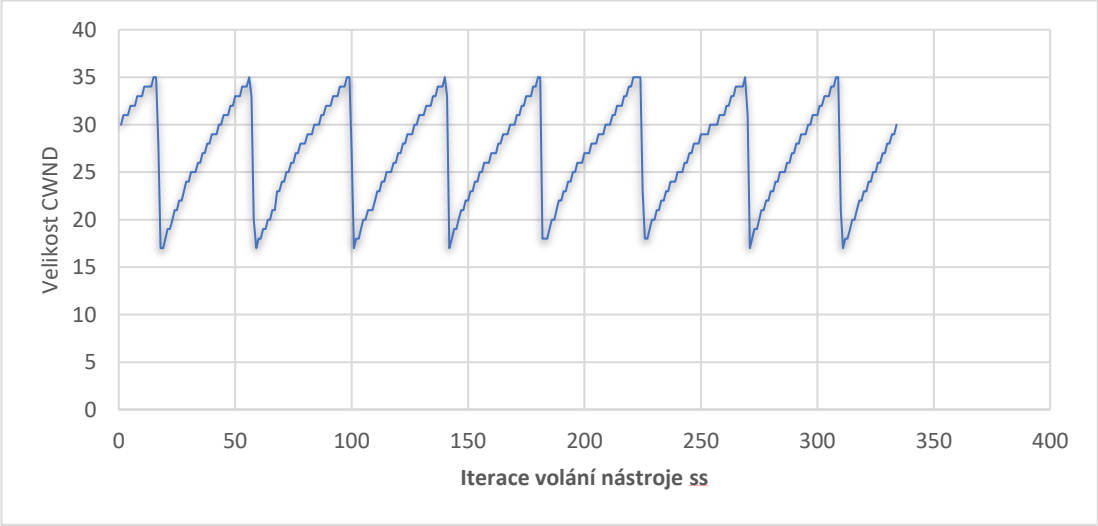
Obrázek 21 Illinois - průběh velikosti okna CWND za jednu iteraci nástroje ss



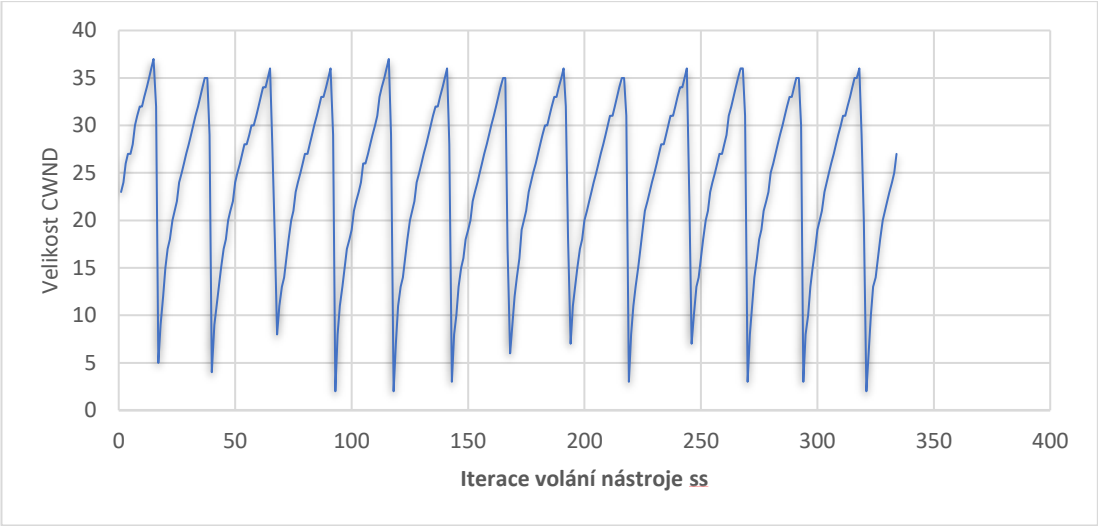
Obrázek 22 Low priority - průběh velikosti okna CWND za jednu iteraci nástroje ss



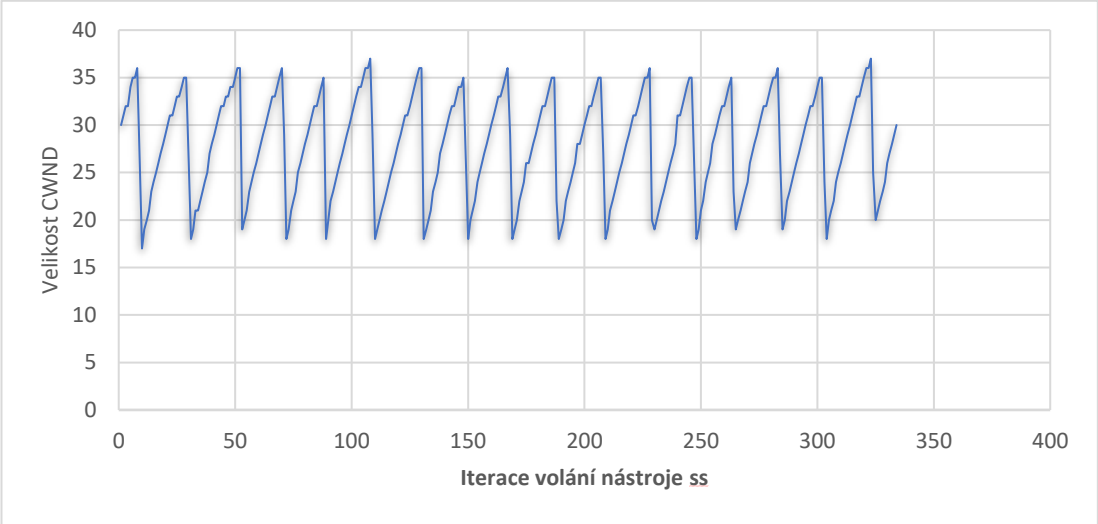
Obrázek 23 New vegas - průběh velikosti okna CWND za jednu iteraci nástroje ss



Obrázek 24 Veno - průběh velikosti okna CWND za jednu iteraci nástroje ss



Obrázek 25 Westwood - průběh velikosti okna CWND za jednu iteraci nástroje ss



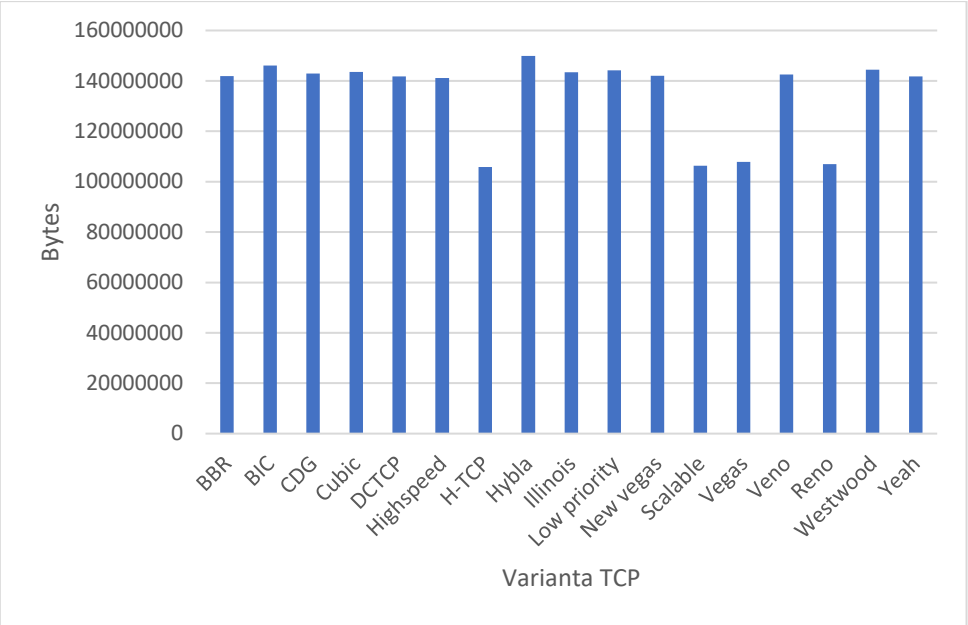
Obrázek 26 Yeah - průběh velikosti okna CWND za jednu iteraci nástroje ss

5.5.2 Počet přenesených byte a velikost okna CWND

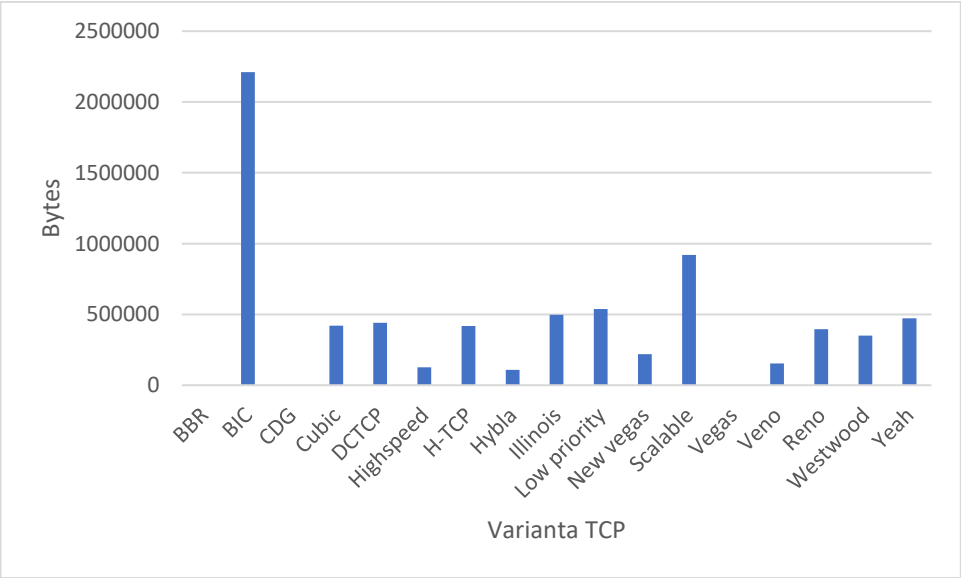
V tabulce jsou zanalyzovaná data přenesených dat (bytes) za provedené měření rozdělena podle congestion control. Dále jsou v tabulce průměr, medián a modus velikosti CWND. Tyto hodnoty (poslané bytes, průměr a medián) jsou přeneseny do grafu pod tabulkou pro lepší čitelnost.

Congestion control	Bytes send	Bytes resend	CWND průměr	CWND medián	CWND modus
BBR	141935856	0	13,898056	14	14

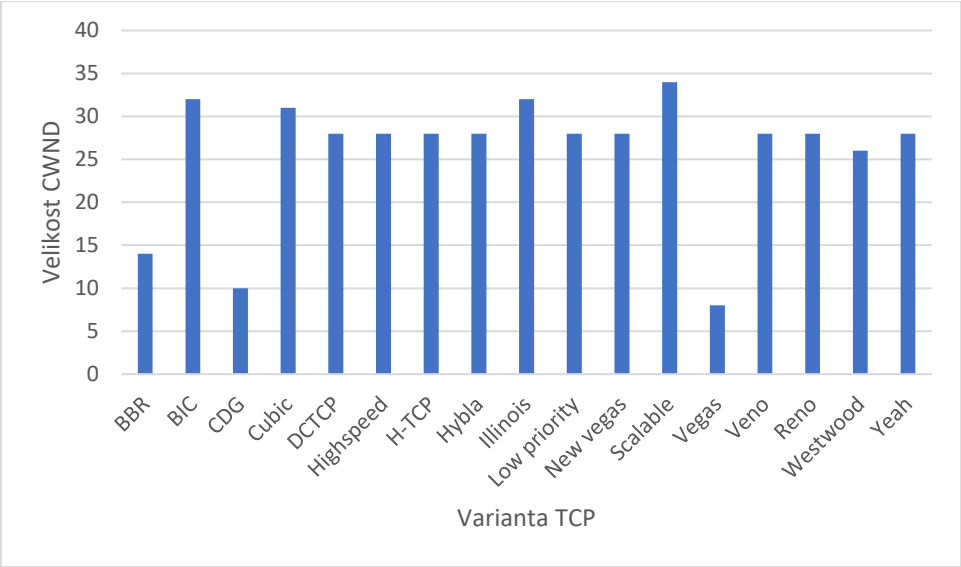
BIC	146055416	2209648	32,0911111	32	34
CDG	142884296	0	11,72389	10	7
Cubic	143531552	419920	30,65722	31	35
DCTCP	141828704	440192	27,67861	28	32
Highspeed	141193032	127424	27,30972	28	32
H-TCP	105779296	418472	27,75222	28	32;34
Hybla	149850624	108600	27,31417	28	34
Illinois	143412816	496664	31,14028	32	33
Low priority	144222248	537208	27,85	28	33
New vegas	141974952	218648	27,4075	28	28
Scalable	106361392	920928	33,62944	34	33
Vegas	107900616	0	7,951667	8	8
Veno	142519400	153488	27,09722	28	34
Reno	106939144	395304	27,976944	28	34
Westwood	144423520	350416	24,2075	26	31;32
Yeah	141750512	472048	27,82278	28	35



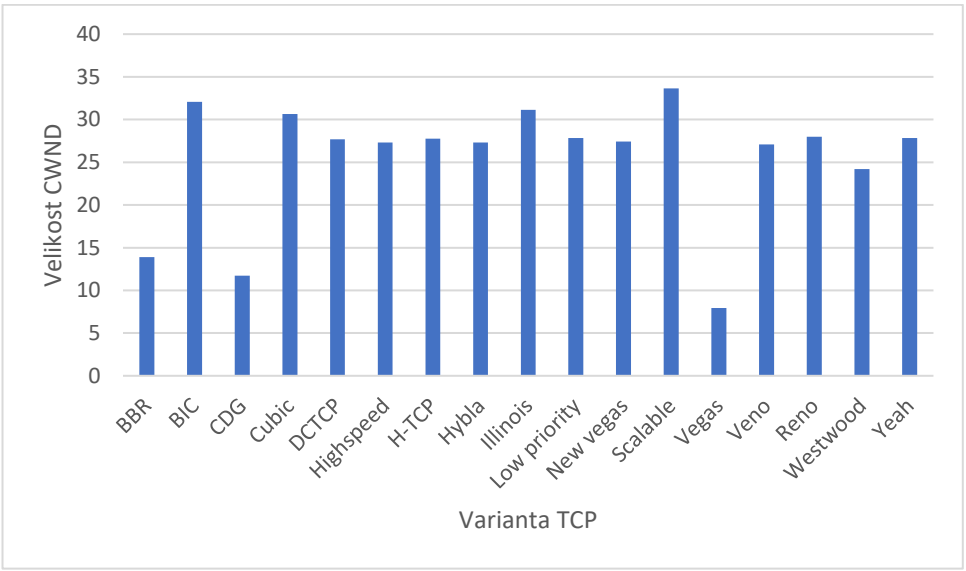
Obrázek 27 Srovnání objemu přenesených dat (bytes) u různých variant TCP



Obrázek 28 Objem znovu odeslaných dat (bytes) u jednotlivých variant TCP



Obrázek 29 Srovnání mediánu velikosti CWND u různých variant TCP



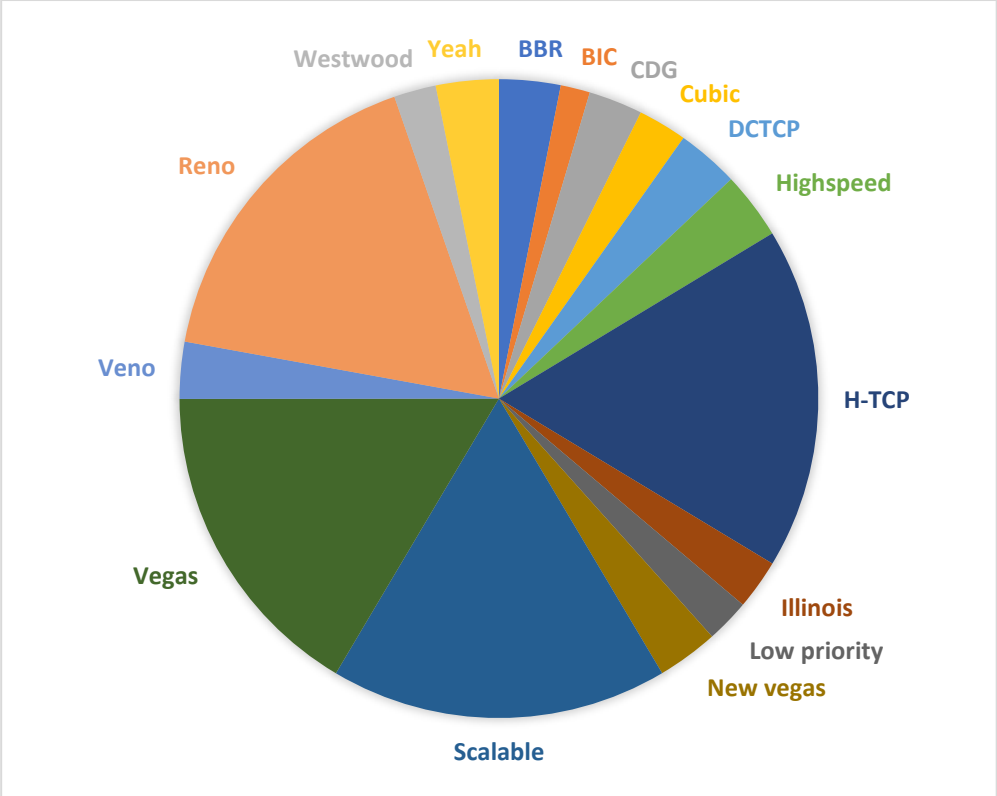
Obrázek 30 Srovnání průměrné velikosti CWND u různých variant TCP

Procentuální porovnání objemu přenesených dat v procentech oproti TCP Hybla, kdy TCP Hybla je sto procent, protože měla největší objem přenesených dat.

Varianta TCP	Procentuální rozdíl v objemu přenesených dat oproti TCP Hybla
BBR	5,2817718
BIC	2,532660792
CDG	4,648848176
Cubic	4,216914038
DCTCP	5,353277675

Highspeed	5,777481447
H-TCP	29,41017316
Illinois	4,296150278
Low priority	3,755991033
New vegas	5,255681818
Scalable	29,02172233
Vegas	27,99455009
Veno	4,892354669
Reno	28,63617038
Westwood	3,621675943
Yeah	5,405457638

Tabulka procentuálního rozdílu v objemu přenesených dat přenesená do grafu



Obrázek 31 Procentuální rozdíl v objemu přenesených dat oproti TCP Hybla

6. Závěr

Tato bakalářská práce se zabývala problematikou TCP congestion control (CC) a jeho efektivitou odesílání dat do sítě. Především změřením velikosti CWND z hlediska průběhu a přenesení dat v důsledku přetížení sítě. Měření probíhalo v programu GNS3, který nabízí vytvoření vlastní topologie sítě a nastavení jednotlivých zařízení jako ve skutečném světě.

V teoretické části se tato bakalářská práce věnovala popsaní jednotlivých protokolů TCP congestion control a jejich matematickému výpočtu velikosti CWND.

Měřená problematika byla jaká varianta TCP má největší objem přenesených dat a zda to nějak ovlivňuje velikost CWND.

Pro tento účel byla vytvořena testovací topologie v programu GNS3, na které proběhlo měření jednotlivých protokolů TCP congestion control. Všechny protokoly byli měřené za stejných podmínek: stejná přenosová kapacita linek, stejná doba měření a použití implicitního nastavení aditivních parametrů (pro specifické varianty TCP).

Z naměřených hodnot je viditelné, že velikost okna byla pro většinu protokolů CC okolo hodnoty 28 až 32 a ostatní jsou v rozmezí 8 až 14, ale tyto hodnoty neurčují, že tyto protokoly, které měli menší velikost okna CWND poslali méně dat než ostatní. Největší průměrnou velikost okna měl protokol TCP Scalable, který měl průměrnou hodnotu 33,6 ale medián tohoto protokolu je 34.

Z naměřených dat vyšlo, že nejvíce přenesených dat bylo posláno při zvoleném protokolu TCP Hybla. Ostatní protokoly na tom byli velice podobně až na pět měřených protokolů, které měli výrazně míň poslaných dat. Jeden z těchto protokolů je již zmiňovaný Scalable, který má největší okno CWND, ale je ve skupině, která poslala míň dat než ostatní.

Z hlediska znovu přenesených dat na tom byl nejhůř protokol TCP BIC, který měl největší objem znovu poslaných dat, ale v celkovém porovnání objemu přenesených dat na tom nebyl nejhůř, protože byl po protokolu TCP Hybla další, který poslal nejvíce dat.

Z hlediska efektivity objemu přenesených dat a jejich objemu znovu přenesených dat pro protokoly TCP congestion control je nejefektivnější protokol TCP Hybla.

Seznam použité literatury

[RFC3168] https://datatracker.ietf.org/doc/html/rfc3168

[1] Velký průvodce protokoly TCP-IP a systémem DNS autor Libor Dostálek

[2] Data Center TCP (DCTCP): TCP Congestion Control for Data Centers Internet Engineering Task Force (IETF) S. Bensley Request for Comments: 8257 D. Thaler Category: Informational P. Balasubramanian ISSN: 2070-1721 Microsoft L. Eggert NetApp G. Judd Morgan Stanley October 2017

[3] CUBIC: A New TCP-Friendly High-Speed TCP Variant ∗ Sangtae Ha, Injong Rhee Dept of Computer Science North Carolina State University Raleigh, NC 27695 {sha2,rhee}@ncsu.edu Lisong Xu Dept of Comp. Sci. and Eng. University of Nebraska Lincoln, Nebraska 68588 xu@cse.unl.edu

[4] TCP Westwood:congestion control with faster recoveryUCLA CSD TR #200017S. Mascolo, C. Casetti, M. Gerla, S. S. Lee, M. SanadidiComputer Science Department - UCLALos Angeles, CA, 90024

[5] Performance Evaluation of Westwood+ TCP Congestion Control ?,?? S. Mascolo ∗, L. A. Grieco, R. Ferorelli, P. Camarda, G. Piscitelli Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona, 4 – 70125 BARI, Italy {mascolo;a.grieco;ferorelli;camarda;piscitel}@poliba.it

[6] COMPARATIVE ANALYSIS OF NEWER CONGESTION CONTROL ALGORITHMS IN HIGH BDP NETWORKS Sheshank Gupta*1, Yashvardhan Singh*2 *1Ex-Undergraduate Student, Computer Science And Engineering, National Institute Of Technology Rourkela, Rourkela, Odisha, India. *2Master's Student, Mathematics And Physics, University Of Sussex, Brighton, East Sussex, United Kingdom

[7] TCP VEGAS ALGORITHM FOR CONTROLLING THROUGHPUT AT THE TRANSPORT LAYER Michal Pánek Master Degree Programme (2), FEEC BUT E-mail: xpanek01@feec.vutbr.cz Supervised by: Jaroslav Koton E-mail: koton@feec.vutbr.cz

[8] TCP Congestion Control Comparison A. Esterhuizen and A.E. Krzesinski Department of Mathematical Sciences University of Stellenbosch, 7600 Stellenbosch, South Africa aek1@cs.sun.ac.za Tel.: +27 21 808 4232 Fax: +27 21 882 9865

[9] H-TCP: TCP for high-speed and long-distance networks D. Leith∗ , R. Shorten∗ Hamilton Institute, NUI Maynooth

[10] Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks Lisong Xu, Khaled Harfoush, and Injong Rhee Department of Computer Science North Carolina State University Raleigh, NC 27695-7534 lxu2, harfoush, rhee@csc.ncsu.edu

[11] TCP Hybla: a TCP enhancement for heterogeneous networks Carlo Cainin,y and Rosario Firrincieli DEIS/ARCES, Bologna University, Viale Risorgimento 2, 40136, Bologna, Italy

[12] TCP-LP: Low-Priority Service via End-Point Congestion Control Aleksandar Kuzmanovic and Edward W. Knightly

[13] YeAH-TCP: Yet Another Highspeed TCP Andrea Baiocchi, Angelo P. Castellani and Francesco Vacirca INFOCOM Department - University of Roma ”Sapienza”, Via Eudossiana 18, 00184 Roma, Italy e-mail: {baiocchi,castellani,vacirca}@infocom.uniroma1.it

[14] Scalable TCP: Improving Performance in Highspeed Wide Area Networks This work is submitted to the PFLDnet 2003 workshop for the purposes of discussion. The original paper was submitted in December 2002 to a journal for

publication. Tom Kelly CERN∗ / University of Cambridge† ctk21@cam.ac.uk 21 December 2002

[15] Performance evaluation of massively parallel and high speed connectionless vs. connection oriented communication sessions Zoltán Gál, Gergely Kocsis, Tibor Tajti, Robert Tornai July 2021 Article 103010

[16] Revisiting TCP Congestion Control Using Delay Gradients David A. Hayes and Grenville Armitage Centre for Advanced Internet Architectures Swinburne University of Technology, Melbourne, Australia {dahayes,garmitage}@swin.edu.au

[17] BBR Congestion-Based Congestion Control, Measuring bottleneck bandwidth and round-trip propagation time, NEAL CARDWELL YUCHUNG CHENG C. STEPHEN GUNN SOHEIL HASSAS YEGANEH VAN JACOBSON, september-october 2016

Seznam Obrázků