České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra Řídící techniky

Diplomová práce

Tester 10 Gigabit Ethernet sítí

Bc. Jáchym Šimák

Vedoucí práce: Ing. Jan Kubr

Studijní program: Otevřená informatika, strukturovaný, Navazující magisterský

Obor: Počítačové inženýrství

2. ledna 2012

Poděkování

Mé poděkování patří především rodině, za jejich podporu nejen během psaní diplomové práce, školiteli Ing. Janu Kubrovi, Ing. Alexandru Mouchouvi za trpělivost při mém lamentování nad funkčností laboratorních zařízení, Bc. Morisovi Bangourovi, za pomoc při sestavováním testovací sítě a Bc. Peterovi Šomló za korektury.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem podkladů uvedených v přiloženém seznamu.

Souhlasím s užitím tohoto školního díla ve smyslu §60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 2. 1. 2012 .....

Abstrakt

Diplomová práce se zabývá analýzou OpenSource nástrojů pro testování parametrů komplexních sítí, ale i jednotlivých zařízení. Rozebírá parametry, pro něž je třeba navrhnout metodiku měření a proměnné tyto parametry ovlivňující. Specifikuji hlavní typy aplikací a jejich požadavky na přenosový řetězec. Testuji OpenSource programy k testování sítí, včetně popisu nutných optimalizací k dosažení nejvyšších výkonů.

Abstract

This thesis deals with the assessment of OpenSource tools for testing complex networks and networking devices. It discusses parameters and methodologies for network performance measurement. The influence of variables on the network parameters and the main network utilization patterns are described. Performance optimization of the measurement tools are suggested.

Obsah

Seznam Ilustrací

Seznam tabulek

1. Úvod

1.1. Motivace

10 Gigabitové sítě dnes již nejsou výsadou pouze poskytovatelů internetu. Stále větší množství i menších firem při budování své infrastruktury o této nejvyšší kategorii pro interní sítě přemýšlí. Nicméně je to stále pro mnoho techniků novinka, u které chybí potřebné znalosti.

Všeobecně lze říci, že i celé testování sítí je věc, která se v České republice nevykonává zrovna pečlivě. Obvykle totiž zákazníkovi stačí protokoly z testování jednotlivých zařízení výrobcem a předvedení funkčnosti infrastruktury s jeho aplikacemi a to nejen z důvodů, že profesionální certifikované testery jsou velice drahé. Tyto nízké nároky na testování se ale později mohou velmi vymstít, protože problémy s nedostatkem prostředků nastanou až přesně ve chvílích, kdy jsou tyto prostředky potřeba. Proto se pokouším popsat jakým způsobem sestavit necertifikovaný tester sítí, který poskytne alespoň přibližné výsledky o jednotlivých parametrech sítě.

1.2. Popis struktury diplomové práce

Diplomovou práci jsem rozčlenil do několika kapitol, jejich obsah je uveden v seznamu níže:

- Kapitola 1. - Tato kapitola obsahuje motivaci k sepsání diplomové práce a její strukturu
- Kapitola 2. - Specifikuje úkony a cíle diplomové práce
- Kapitola 3. - Třetí kapitola obsahuje popis zapojení mého laboratorního prostředí a to jak formou schémat, tak i informací o výkonnostních parametrech jednotlivých zařízení.
- Kapitola 4. - V kapitole Měřené parametry testované sítě popisuji jednotlivé parametry sítě, jejichž měřením se při testování sítí musíme zabývat.
- Kapitola 5. - V této části popisuji proměnné ovlivňující měření, jejich seznam členěný podle vrstvy OSI modelu, ke které přísluší. Definuji zde také maximální teoretickou propustnost užitečných dat, které se budu snažit při stavbě generátoru dosáhnout.
- Kapitola 6. – V této kapitole charakterizuji základní kategorie síťového provozu, které se na sítích vyskytují a popisuji jejich požadavky na přenosovou cestu. Toto je nezbytně nutné, aby bylo možné navrhnout co testovat.
- Kapitola 7. - Zde jsou popsány způsoby jak optimalizovat operační systém k získání dobrých výsledků
- Kapitola 8. - Popisuje jakým způsobem byly jednotlivé programy testovány a obsahuje též naměřené hodnoty a jejich význam.
- Kapitola 9. - V této kapitole jsou vyhodnoceny dosažené výsledky
- Kapitola 10. - Obsahuje seznam použité literatury

- Kapitola 11. neboli příloha 1. - Popisuje obsah přiloženého CD
- Kapitola 12. neboli příloha 2. - Obsahuje popis souborů použitých na testy
- Kapitola 13. neboli příloha 3. - Obsahuje popis instalace testovacích programů
- Kapitola 14. neboli příloha 4. - Obsahuje seznam testovacích programů vhodných pro další testování
- Kapitola 15. neboli příloha 5 – Obsahuje seznam použitých zkratek

2. Specifikace cíle

Cílem práce je otestovat OpenSource programy pro testování parametrů komplexní sítě. Jako první část práce je tedy nutné specifikovat, pro které parametry je potřeba navrhnout systém měření. Abych mohl měřit skutečné chování zařízení v sítích, je třeba navrhnout testování na vyšších vrstvách OSI modelu než na druhé vrstvě OSI modelu.

Z hlediska optimalizace a akcelerace na síťových zařízení je nejzajímavější čtvrtá vrstva OSI modelu, protože na ni je možné již využít téměř veškeré akcelerace implementované v zařízeních. Na vyšších vrstvách už z optimalizací dochází pouze k akceleracím aplikačním (například SSL offloading či komprese přenášených dat) a protože tyto vlastnosti nejsou obsaženy ve standardních síťových zařízeních ale pouze ve specializovaných zařízeních, lze navrhovat systém měření univerzálně pro čtvrtou vrstvu. Na této vrstvě je definováno celé spektrum protokolů, proto je nutné sepsat základní typy provozu, aby bylo možné určit jaké protokoly tyto typy provozu využívají a jaké parametry od sítě požadují.

Po tomto kroku je nutné případně upravit testovací zařízení do podoby, kdy jsou schopné v co nejvyšším výkonu parametr změřit. Pro vybraný operační systém Linux existuje velké množství návodů jak má být upraven, bude tedy náročnější je projít a použít pouze správné úpravy.

Jako finální krok bude již samostatné testování jednotlivých programů a zhodnocení jejich použitelnosti a výkonnosti pro měření daných parametrů.

3. Testovací sítě

Tato kapitola obsahuje popis zapojení mého laboratorního prostředí a to jak formou schémat, tak i informací o výkonnostních parametrech jednotlivých zařízení.

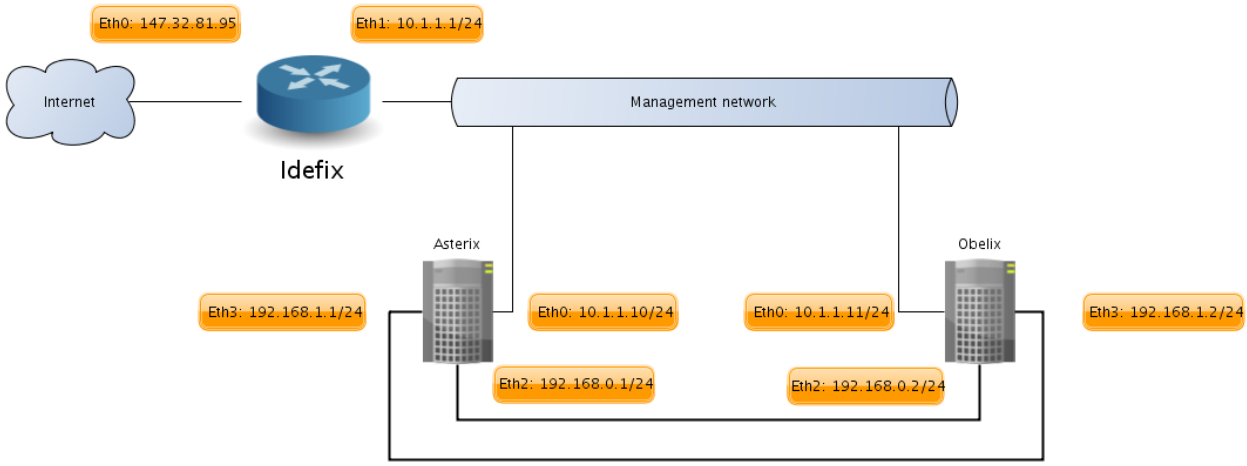
3.1. Popis

Pro testování budu využívat dva počítače opatřené 10Gb síťovými kartami. Jejich role budou v závislosti na druhu testování určeny takto:

1. Asterix (PC1) – server , Obelix (PC2) – klient
2. Asterix (PC1) – klient a server zároveň, Obelix (PC2) – klient a server zároveň

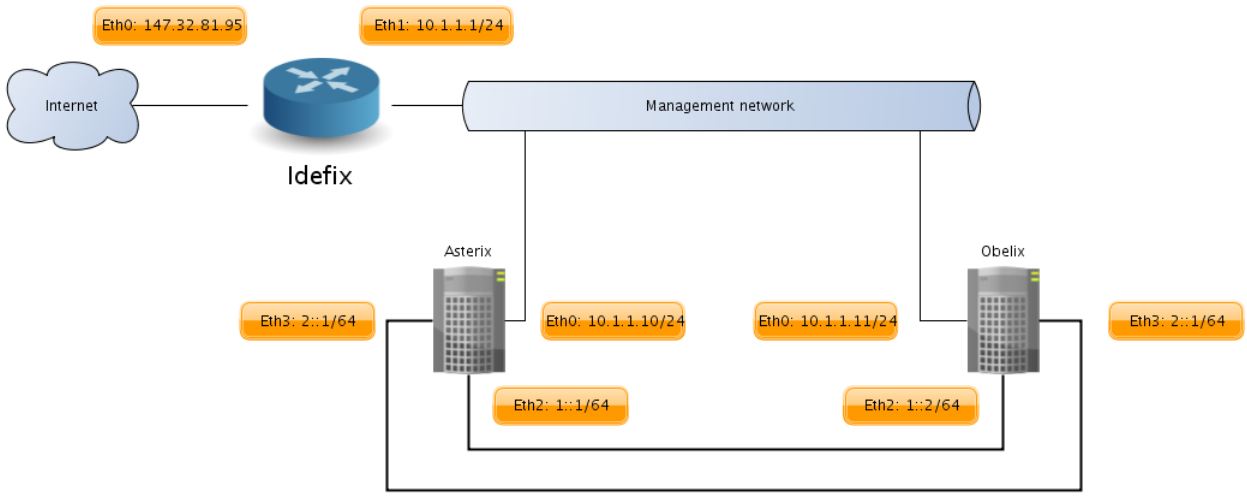
Jednotlivá zařízení pro jednotlivé testy jsou zapojena dle schémat níže.

3.1.1) Schéma testovací sítě pro protokol IP verze 4



Ilustrace 1: Schéma testovací sítě pro protokol IP verze 4

3.1.2) Schéma testovací sítě pro protokol IP verze 6



Ilustrace 2: Schéma testovací sítě pro protokol IP verze 6

3.2. Testovací zařízení

3.2.1) Asterix - PC1

- 1x SuperMicro MB X8DAH+ (2xLGA1366, 18xDDR3 ECC, 6xSATA, 7xPCIe)
- 2x Intel Xeon E5606 (4c/2.13GHz/8MB/LGA1366)
- 2x INTEL heatsink
- 6x Kingston 2GB 1333MHz DDR3 ECC CL9 DIMM
- Intel® 82599EB 10 Gigabit Ethernet Controller SFI/SFP+ Dual Port

3.2.2) Obelix - PC2

- 1x SuperMicro MB X8DAH+-F (2xLGA1366, 18xDDR3 ECC, 6xSATA, 7xPCIe, IPMI remote access)
- 2x Intel Xeon E5620 (4c/2.40GHz/12MB/LGA1366)
- 2x INTEL heatsink
- 6x Kingston 2GB 1333MHz DDR3 ECC CL9 DIMM
- Emulex Corporatio OneConnect 10GbE Network Adapters SFI/SFP+ Dual Port

3.2.3) Idefix – Router

Tento stroj je použit pouze pro management a přístup k internetu. Z hlediska měření nejsou parametry tohoto stroje relevantní.

3.2.4) Operační systém

Jako operační systém byl použit Debian Lenny pro AMD64. Jediná nestandardnost kromě úprav popsaných v sekci tunning, bylo použití vlastního jádra 3.1.2-xsimi oproti defautnímu jádru 2.6.32-5. Domníval jsem se, že s využitím nejnovějšího jádra dosáhnu lepších výsledků, nicméně testy pomocí programu Netperf neukázaly žádné zlepšení výkonnosti, ale protože zde nebyl ani pokles výkonnosti, tak toto nové jádro zůstalo a veškeré testování bylo prováděno na něm.

Pozn. instalační balík jádra, hlaviček a config souboru jsou přiloženy v souborech na CD. Popis v kapitole 11. Příloha 1. - Obsah přiloženého DVD

4. Měřené parametry testované sítě

Následující text popisuje jednotlivé parametry sítě, jejichž měřením se při testování sítí musíme zabývat.

4.1. Propustnost

Dá se říci, že propustnost provozu je nejdůležitější parametr dnešních sítí. U tohoto měření půjde o změření průměrného množství přenesených dat/rámců/paketů za sekundu. Je nutné provádět měření s více proměnnými parametry (popsáno v kapitole 5. Parametry ovlivňující provoz) a to jak v jednosměrném provozu, tak i duplexním. Propustnost je z důvodu měření protokolem vyšší vrstvy rovněž ovlivněna u mnohých protokolů jinými parametry jako je ztrátovost, sekvenčnost a latence.

4.2. Latence

Vysokorychlostní sítě jsou budovány nejen z důvodů vysoké propustnosti, ale i z důvodu zrychlení odezvy. Pro některé typy provozu (viz kapitola 6. Hlavní typy aplikací) je latence kritickým parametrem.

Typy latence k měření:

- Průměrná latence
- Maximální latence
- Jitter (rozptyl latencí)

Bohužel latenci nemůžeme jednoduše měřit a udávat pouze jako tyto tři hodnoty. Latence je totiž u komplexnějších sítí závislá na použitém protokolu a operaci. Např. jinou odezvu bude mít nově vytvářené spojení při cestě přes firewall, či load balancer oproti již navázaným spojením.

4.3. Ztrátovost

Ve chvílích, kdy už propustnost sítě nestačí a došly i všechny buffery, takže není kam již data umisťovat, je nutné je zahodit. Tento výpadek dat bývá pak detekován vyšším protokolem, který omezí množství tekoucích dat. Pokud přesně určíme místo, kde dochází ke ztrátám dat, lokalizujeme i nejslabší místo sítě, na kterém pak mohou být provedeny optimalizace. Zároveň i každý paket, u kterého kontrolní součty nesouhlasí s vnitřnímu daty, je zahozen, a je možné ho započítat jako ztracený. Tento parametr má přímý vliv na propustnost a sekvenčnost.

4.4. Sekvenčnost

U rozsáhlejších sítí může docházet také k tomu, že pakety docházejí v jiném pořadí než v pořadí, ve kterém byly odeslány, to může některým aplikacím (převážně těm určeným pro komunikaci) způsobovat problémy. Může také dojít k situaci, kdy jsou kvůli prohození pořadí některá data odeslána několikrát.

4.5. Zdroje

Důležité bude monitorovat využití zdrojů (procesory, obsazenost paměti a bufferů), abychom zjistili, jestli úzké hrdlo není náš systém.

4.6. Reakce na chyby

Většina komerčních testerů má mód pro injektování chyb, kdy jsou do sítě pouštěna poškozená data a je sledováno, jak si s tím jednotlivá zařízení poradí.
Např:

- Poškození typu zpoždění, zahození, jitter, chyba CRC, fragmentace
- Vkládání ztráty segmentu, ztráty synchronizace, bitových chyb na L2 a vyšších vrstvách.
- Fixní/periodické nebo náhodné rozložení výskytu poruch, jednotlivé nebo shlukové, mnohonásobné poruchy.

[1]

5. Parametry ovlivňující provoz

Výsledky měření jsou ovlivněny množstvím proměnných. V následující kapitole je jejich seznam členěný podle vrstvy OSI modelu, ke které přísluší. Definuji zde také maximální teoretickou propustnost užitečných dat, které se při stavbě generátoru budu snažit dosáhnout.

5.1. Parametry příslušející 1. vrstvě OSI modelu

Do této kategorie patří především propustnosti a výkon jednotlivých prvků (především generátorů a příjemců provozu). Z pohledu injektovacích strojů záleží především na:

- Propustnosti sběrnice
- Chipu síťové karty
- Správně nastaveném DMA
- Velikost paměti
- Počet a výkon CPU

Pozn. Softwarové vybavení jako parametr je popsán v části: 5.5. Mimo OSI model.

5.2. Parametry příslušející 2. vrstvě OSI modelu

5.2.1) Odstup rámců (Interframe Gap)

Časový rozestup mezi koncem rámce a začátkem následujícího rámce. Jednotka měření je sekunda. Na Ethernetu je tento odstup definován jako minimálně čas pro odeslání 96 bitů – to odpovídá 9.6 μs u 10 Mbit/s Ethernetu, 960 ns pro 100 Mbit/s (fast) Ethernetu, 96 ns na 1 Gbit/s (gigabit) Ethernetu a 9.6 ns na 10 Gbit/s (10 gigabit) Ethernetu. Další zvětšení tohoto rozestupu pak může být způsobeno HW. V případě, že je vysíláno velké množství velmi malých paketů, snižuje vysoká hodnota tohoto parametru propustnost a zvyšuje i latenci, nízká hodnota může u některých sítí způsobit nárůst kolizí. Lze nastavit v OS.

Nicméně jsou povoleny standardizováné výjimky z pravidla a tento 96bitovy interval je možno snížit z 96 bitů na:

- 10 Gigabit Ethernet - 40 bit interval time (5 bytes).
- Gigabit Ethernet - 64 bit interval time (8 bytes).
- Fast Ethernet – není specifikována úprava oproti 96 bitovým intervalům
- Ethernet - 47 bit interval time

[2]

5.2.2) Zpoždění rámců

Časový interval mezi odesláním poslední části rámce a přijetím jeho první části na straně příjemce. Jednotka měření je sekunda.

5.2.3) Ztrátovost rámců

Procento rámců, které jsou vyslány do síťové infrastruktury, ale nedorazí k cíli.

5.3. Parametry příslušející 3. vrstvě OSI modelu

5.3.1) Maximum Transmission Unit (MTU)

Tento parametr definuje maximální velikost přenosové jednotky na úrovni síťové vrstvy. Nižší hodnota tohoto parametru efektivně snižuje propustnost linky, neboť každá fragmentace způsobí nutnost vyslat navíc hlavičku a navíc čekat mezi rámci, nehledě na náročnost této operace. Při testování je třeba se zaměřit i na Jumbo Frame.

5.3.1.A Maximální teoretická propustnost užitečných dat

Z L2 pohledu platí

$$Propustnost[Mb/s] = \frac{Rychlost\ linky[Mb/s] * Užitečná\ data\ v\ paketu[b]}{Velikost\ paketu[b]}$$

Tedy

$$Propustnost[Mb/s] = \frac{Rychlost\ linky[Mb/s] * 8 * (MTU[B] - Transportní\ a\ služební\ data[B])}{8 * (MTU[B] + Interframe\ Gap[B])}$$

Velikost Transportních a ostatních dat vypočteme takto:

$$Transportní\ a\ služební\ data = Preambule + SFD + MAC\ cíle + MAC\ zdroje + ¿$$
$$Pole\ typ/délka + CRC32 + Interframe\ Gap + hlavičky\ vyššího\ protokolu$$

Po dosazení hodnot:

Preambule = 7B
SFD (Start of Frame delimiter) = 1B
Mac cíle = 6B
Mac zdroje = 6B
Pole typ/délka = 2B
CRC32 = 4B
Po dosazení Interframe Gap = 12B
Rychlost linky = 10 Gb/s

Poznámka: Platí pro netagovaný provoz, při využití 802.1q je hlavička zvětšena o 4B

Získáme vzorec:

$$Propustnost[Mb/s] = \frac{10000[Mb/s]*8*(MTU[B]-38[B]-Velikost\ hlaviček\ vyšších\ protokolů)}{8*(MTU+12[B])}$$

5.3.1.B IP verze 4

Pro TCP:

$$Propustnost[Mb/s] = \frac{10\,000*(MTU-66)}{MTU+12}$$

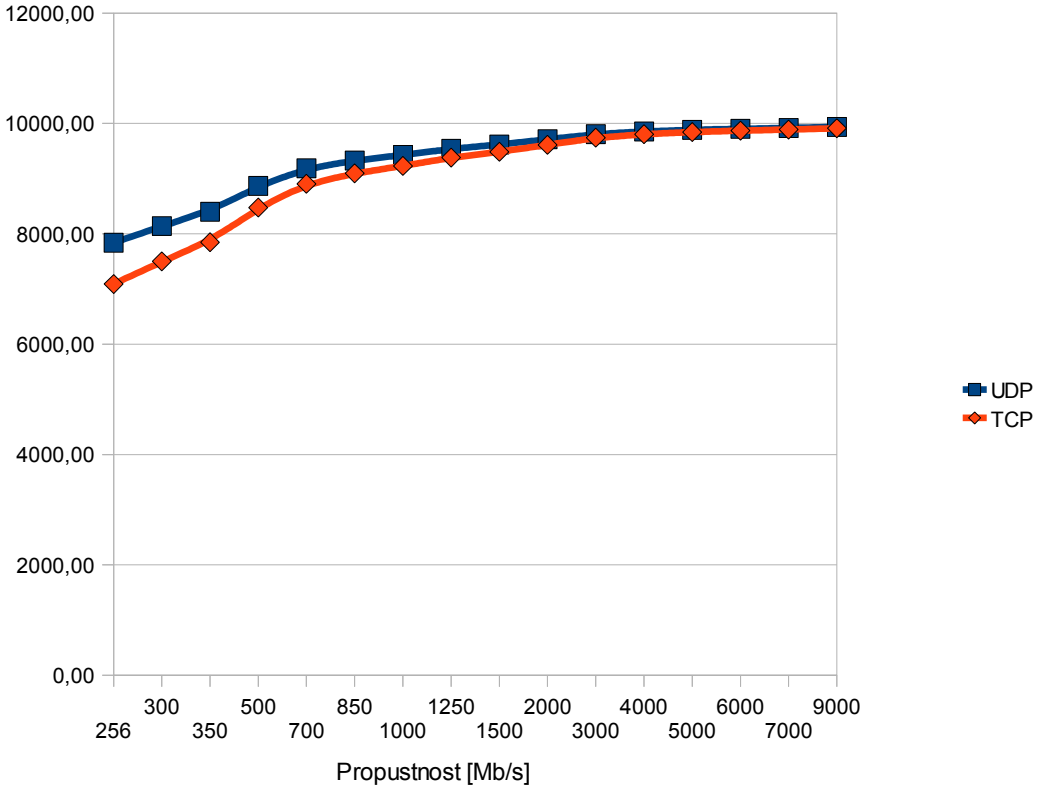
Pro UDP

$$Propustnost[Mb/s] = \frac{10\,000*(MTU-46)}{MTU+12}$$

Maximální teoretická propustnost [Mb/s]		
MTU [B]	UDP	TCP
256	7835,82	7089,55
300	8141,03	7500,00
350	8397,79	7845,30
500	8867,19	8476,56
700	9185,39	8904,49
850	9327,15	9095,13
1000	9426,88	9229,25
1250	9540,41	9381,93
1500	9616,40	9484,13
2000	9711,73	9612,33
3000	9807,44	9741,04
4000	9855,43	9805,58
5000	9884,28	9844,37
6000	9903,53	9870,26
7000	9917,28	9888,76
9000	9935,64	9913,45

Tabulka 1: Maximální teoretická propustnost IP verze 4 v závislosti na MTU

Ve specifikaci je určena minimální podporovaná MTU pro IPv4 jako 578 B, Linux umožňuje mimo specifikace využít hodnoty již od 256B. Následující body k měření MTU jsem určil ve vhodném rozpětí.



Ilustrace 3: Maximální teoretická propustnost IP verze 4 v závislosti na MTU

5.3.1.C IP verze 6

Pro TCP:

$$Propustnost\,[Mb/s] = \frac{10\,000 * (MTU - 86)}{MTU + 12}$$

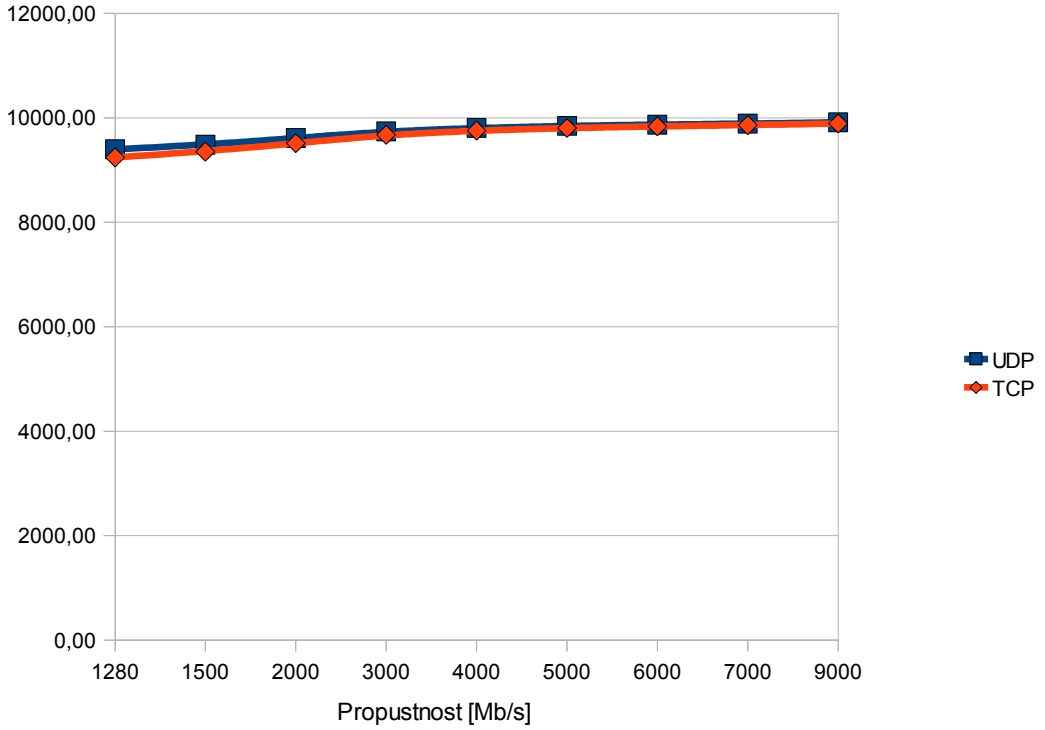
Pro UDP

$$Propustnost\,[Mb/s] = \frac{10\,000 * (MTU - 66)}{MTU + 12}$$

Maximální teoretická propustnost [Mb/s]		
MTU [B]	UDP	TCP
1280	9396,28	9241,49
1500	9484,13	9351,85
2000	9612,33	9512,92
3000	9741,04	9674,63
4000	9805,58	9755,73
5000	9844,37	9804,47
6000	9870,26	9836,99
7000	9888,76	9860,24
9000	9913,45	9891,26

Tabulka 2: Maximální teoretická propustnost IP verze 6 v závislosti na MTU

IPv6 má ve specifikaci určeno minimální MTU jako 1280B a menší hodnotu nelze nastavit, proto též všechna měření začínají od této hodnoty.



Ilustrace 4: Maximální teoretická propustnost IP verze 6 v závislosti na MTU

Pozn. Upozorňuji, na zdánlivě nižší pokles výkonnosti u IP verze 6 oproti grafu verze 4, ten je způsoben vykreslením grafu od MTU 1280 – minimum pro IPv6.

5.3.2) Protokolová závislost

U některých typů prvků může docházet k jinému chování v závislosti na používaném protokolu L3, tedy například (IP verze 4, IP verze 6, Raw či IPX). Proto je nutné rozšířit testy. Zde se budou výsledky lišit i na základě jiné velikosti hlavičky, ale i na základě naprosto jiného způsobu přenosu. Zvláštní kapitolou by pak zasluhovalo testování Multicast protokolů, bohužel pro toto není k dispozici dostatek strojů.

5.4. Parametry příslušející 4. vrstvě OSI modelu

5.4.1) Protokolová závislost

Otestovat základní protokoly TCP a UDP. Předkládám, že UDP přenosy budou daleko efektivnější z důvodu menší hlavičky, neexistujících polí zajištujících kontrolu přenosu. Nicméně zde bude daleko náročnější data zpracovat z důvodu netransakčního způsobu přenosu u protokolu UDP.

5.5. Mimo OSI model

5.5.1) Softwarové vybavení

Do této kategorie patří:

- Operační systém, jeho architektura, nastavení a verze
- Kvalita ovladačů
- Program generující a přijímající data a jeho nastavení

6. Hlavní typy aplikací

V této kapitole charakterizuji základní kategorie síťového provozu, které se nám na sítích vyskytují a popisuji jejich požadavky na přenosovou cestu. Toto je nezbytně nutné, aby bylo možné navrhnout co testovat.

6.1. Datové streamy

Datové streamy jsou nosným typem provozu ve všech transportních službách, ale i P2P sítí. Při testování čistého datového streamu bude nejjednodušší dosáhnout nejvyšších výkonů. Každý paket má maximální možnou velikost a je vyslán bez čekání okamžitě, když je to možné.

Charakteristika datového streamu:

- Protokoly: TCP/IP, UDP/IP
- Velikost paketu: Maximální možná, konstantní
- Počet paketů: Vysoký, po ustálení konstantní tok
- Náchylnost na zpoždění: Žádná či minimální
- Náchylnost na kolísání rychlosti: Nulové

6.2. Signalizace/servisní data

Do této kategorie patří přenášení veškeré signalizace a servisní data, jako například přenos informací routovacích protokolů, či signalizace např. Signalizační kanál transportních služeb.

Příklad: Routovací informace (BGP/OSPF)

Charakteristika servisního streamu:

- Protokoly: TCP, UDP, ICMP
- Velikost paketu: Obvykle malá, nekonstantní
- Počet paketů: Obvykle velmi nízký ve srovnání s transportním tokem, nekonstantní tok
- Náchylnost na zpoždění: Vyšší
- Náchylnost na kolísání rychlosti: Střední

6.3. Transportní služby

Transportní služby se obvykle skládají ze dvou druhů spojení: kontrolního(servisního) a datového streamu (datový stream popsán výše). Kontrolní stream bývá náchylný na zpoždění a jsou na něm poskytovány informace o úkonech, zatímco po datovém streamu se provádí vlastní přenos užitných dat. Aby bylo měření průkazné, je třeba zamezit vlivu pomalých médií (HDD), například pomocí použití cache nebo RAM disku.

Příklad protokolů: FTP, RSYNC

Charakteristika transportních služeb:

- Charakteristika servisního streamu: v podkapitole 6.2 Signalizace/servisní data
- Charakteristika datového streamu: v podkapitole 6.1 Datové streamy

6.4. Interaktivní přenosy a komunikace

Do této sekce spadá komunikace vyžadující rychlou odezvu od obou stran například komunikace, hry, Ajax či Webex přenosy. Je nutné zajistit stálou latenci a přenosovou rychlost, jinak dochází k výraznému snížení kvality poskytovaných služeb.

Příklady:

- Ajax
- Online hry
- Instant messaging (Icq, Jabber)
- Přenos mluvených zpráv: Teamspeak, Voip (kodeky G711 a G722 - 64 kbit/s, skutečná zátěž je zde cca 90 kbit/s))
- Přenos video chatu: Skype
- Vzdálená plocha, Webex session

Obecná charakteristika interaktivních přenosů:

- Protokoly: UDP/IP
- Velikost paketu: Konstantní, malá
- Počet paketů: Obvykle velmi nízký ve srovnání s transportním tokem
- Náchylnost na zpoždění: Vysoká
- Náchylnost na kolísání rychlosti: Vysoká

6.5. Komunikace neinteraktivní

Do neinteraktivní komunikace patří například email.

Obecná charakteristika neinteraktivní komunikace:

- Technické požadavky jsou stejné jako pro datové streamy (podkapitola: 6.1 Datové streamy)

6.6. Tunelování

Slouží k transportu provozu mezi dvěma sítěmi/nody, přes jinou síťovou strukturu. Tato struktura nemusí podporovat tunelovaný protokol, nemusí znát routování za endpointama. Transportovaný provoz může být i šifrován pro zvýšení bezpečí. Principiálně tunelování funguje tak, že se původní paket obalí tunelovací a transportní hlavičkou a odešle, na druhé straně se pak tyto přidané hlavičky odstraní a vznikne původní paket.

Příklad: IPoverIP

Technické požadavky jsou určené tunelovanou aplikací.

6.7. Přístup k souborovým systémům a databázím

Slouží pro vzdálený přístup k datům. Vysoce náročné na všechny zdroje.

Příklad: ATA přes Ethernet, SAN, NFS

Charakteristika přístupu k souborovým systémům a databázím:

- Protokoly: Nespecifikovatelné, velké množství
- Velikost paketu: Nekonstantní
- Počet paketů: Vysoký
- Náchylnost na zpoždění: Vysoká
- Náchylnost na kolísání rychlosti:Vysoká

6.8. Vyhodnocení

Většina druhů provozu využívá přenosových protokolů TCP a UDP a těmi se proto budu prioritně zabývat. Bohužel se mi nepodařilo sehnat lepší zdroj poměru použitých protokolů než [3]. Nicméně data z tohoto zdroje ukazují, že zhruba 97% dat je přenášeno pomocí TCP a UDP.

7. Optimalizace operačního systému

Ač 10G sítě dnes již nejsou úplnou novinkou a operační systémy jsou na jejich využití většinou dobře připraveny, nejsou distribuce optimalizovány. Avšak obvykle tyto nedostatky nejsou způsobeny nedostatky ve znalostech správců distribucí, ale cílením na určitou průměrnou skupinu použití. Přeci jen nastavení bufferu pro TCP spojení na 16Mb má úplně jiné následky u serveru s desítkami Gigabytů pamětí, než u malého zařízení, kde se paměť počítá v desítkách Megabytů. V následující sekci popisuji úpravy, které je nutné vykonat, aby došlo ke zvýšení výkonů k maximu možného.

7.1. Zvětšení TCP bufferu

Operační systémy mají vyhraněny pro operace týkající se TCP streamů paměťové limity. Tento limit je určen velikostí přijímacího a odesílacího okna pro jedno spojení. Pokud jsou v systému obsažena rychlejší interface, či je na lince větší latence, je nutné toto nastavení zkontrolovat a případně poupravit.

U standardního TCP protokolu je maximální velikost okna 2^16 tedy 65KB (65535B). Tato velikost určuje, kolik dat může být najednou na cestě bez potvrzení a tím tedy v závislosti na latenci určuje propustnost TCP/IP spojení.

Tedy propustnost v závislosti na velikosti okna:

$$Propustnost[B/s] = \frac{Velikost\ TCP\ okna[B]}{Roundtrip\ zpoždění[s]}$$

Ku příkladu na lince s latencí 1 sekunda můžeme jedním TCP/IP streamem přenášet pouze 65KB/s dat. Ve standardu RFC1323 je popsáno rozšíření okna na až 2^32 B, kdy při latenci 1 sekunda dochází k možnému nárůstu rychlosti jednoho TCP streamu na více jak 4,2 GB/s.

Zapnutí této funkcionality:

```
echo "1" >/proc/sys/net/ipv4/tcp_window_scaling
```

Všeobecně lze říci, že je optimální nastavovat dle vzorce:

$$Velikost\ buffer = 2 * šířka\ pásma * zpoždění$$

Pozn.: Úprava maximální velikosti TCP okna. Jako maximum je doporučeno 16MB pro normální 10G linky, 32MB linky s větší latencí a 40G linky.

Nastavení přijímacího/odesílacího bufferu (minimum, defaultní a maximální hodnota [B]) – používá se autotunning – nicméně pro takhle rychlé linky mívá malé maximum, takže upravíme:

```
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
```

[4]
[5]

7.2. TCP výběrové potvrzování dat

Výběrové potvrzování dat (Selective Acknowledgments Option) povolí možnost informovat odesílatele o tom, přesně které pakety z TCP streamu chybí, či jsou poškozeny. Při používání tedy nedochází k přenosu všech dat od chyby, ale jen chybějících částí. [6]

```
echo "1" > /proc/sys/net/ipv4/tcp_sack
```

Dnes již defaultně v Linuxových distribucích aktivováno.

7.3. TCP_NODELAY

Defaultně je u TCP přenosů využit Nagleho algoritmus. Ten velmi zefektivňuje komunikaci u aplikací generujících data po malých sekvencích, menších než je MSS (např. jeden znak) tím, že pozdrží vyslání dat, které plně nezaplní segment do doby, než je potvrzeno úspěšné přijetí předchozích dat. Bohužel toto čekání způsobí zvětšení latence. Pro některé typy provozu je tedy výhodné tento algoritmus vypnout.

Pseudokód Nagleho algoritmu pro lepší pochopení: [7]

```
if there is new data to send
  if the window size >= MSS and available data is >= MSS
    send complete MSS segment now
  else
    if there is unconfirmed data still in the pipe
      enqueue data in the buffer until an acknowledge is received
    else
      send data immediately
    end if
  end if
end if
```

7.4. Path MTU

Pro nejvyšší výkon je nejvhodnější v cestě použít největší MTU, které je možné. V případě MTU nastaveného na větší hodnotu bude při přenosu docházet k nutnosti pakety fragmentovat a naopak v případě nižšího MTU budou vysílány nadbytečné hlavičky na úkor užitečných dat.

Pro tyto účely vznikly algoritmy, které detekují nejmenší velikost po cestě a využije ji při přenosu. Tento mechanizmus funguje tak, že nastaví v hlavičce paketů příznak „Don't Fragment (DF) bit“, když tento paket dojde na zařízení, kde by bylo nutno fragmentovat, je přeposlána zpět zpráva o maximální velikosti, která projde bez fragmentování. Tato hodnota je na lokálním systému načtena a uplatněna na stream. Podrobnější popis viz ([8], [9], [10]). Při mém měření budu tento parametr MTU měnit.

7.5. TCP Recycle/Reuse

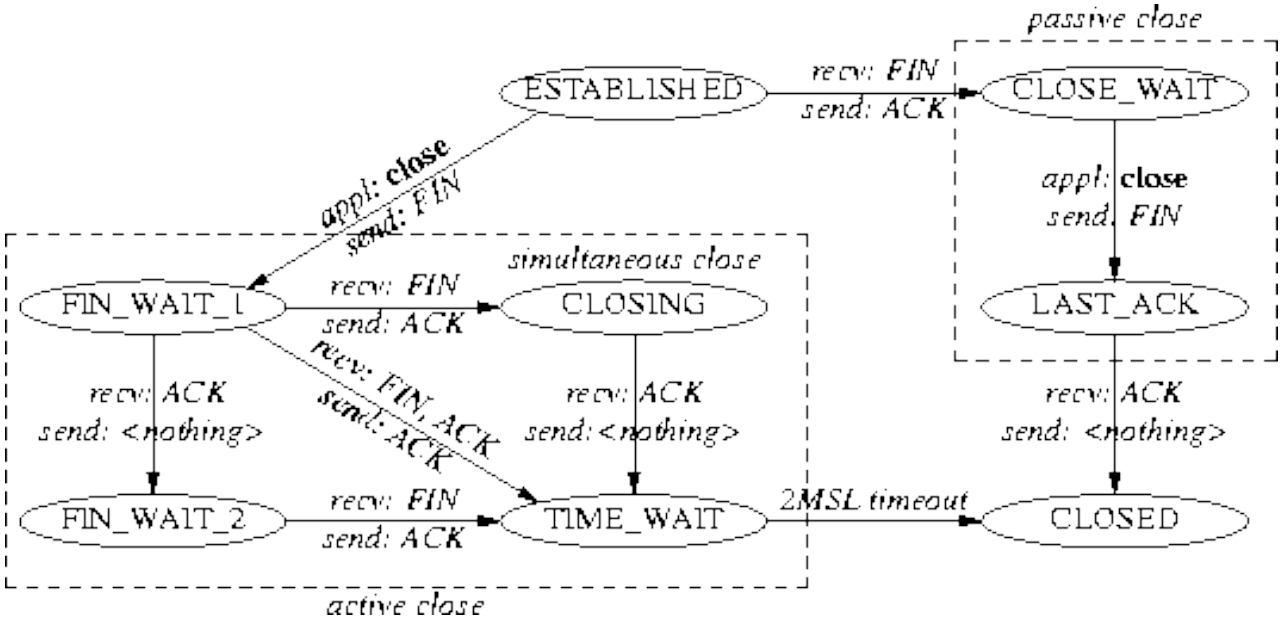
Aktivace těchto voleb umožní využít soket v Time_wait stavu pro nové spojení. Samozřejmě, pokud v té chvíli žádný k dispozici není, tak se alokují nové zdroje, jak je tomu defaultně. Odstranění nutnosti vytváření/rušení soketů zvýší u systémů generující velké množství požadavků výkonnost.

Pozn.: Time_wait stav slouží například k plnému duplexnímu uzavření (situace způsobena tím, že se poslední paket s ACK se ztratí) a k odstranění situace, kdy by dorazivší duplexní paket způsobil otevření nového spojení, i když logickou posloupností patří do zavíraného spojení.

Volba Recycle je oproti Reuse více agresivní a může způsobovat problémy při použití s překladem adres, aplikačními firewally, nebo s load balancery (ty však v testovací síti nemáme).

```
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse
```

Defaultně jsou tyto vlastnosti vypnuty (0)



Ilustrace 5: Schéma stavů u TCP spojení

[11]
[12]

7.6. Úprava bufferů

Mezi síťovým subsystémem a ovladačem síťové karty se nacházejí kruhové buffery pro příchozí provoz a pro odchozí provoz a ty je potřeba navýšit, protože při plném provozu se zaplní příliš rychle a následující provoz je zahozen.

Zvýšení odchozí fronty pro síťový adaptér EthX

```
/sbin/ifconfig ethX txqueuelen 10000
```

Defautní velikost je 1000 paketů

Zvýšení velikosti příchozí fronty

```
sysctl -w sys.net.core.netdev_max_backlog=400000
```

[13]

7.7. Rozšíření počtu otevřených descriptorů

Operační systémy mají omezení na množství descriptorů na jeden proces, proto když budeme chtít spustit testovací program, který simuluje více klientů (desítky tisíc), musíme tento limit navýšit.

```
ulimit -n 40000
```

[14]
[15]
[16]

8. Programy

Následující část popisuje testování jednotlivých programů, jejich přizpůsobení a dosažené výsledky.

Struktura u jednotlivých programů je:

- Název
- Popis programu
- Jednotlivé testy a jejich výsledky
- Zhodnocení výsledků

8.1. Netperf

8.1.1) Popis

Program určený pro měření propustností sítí. Primárně se zaměřuje na TCP a UDP přenosy, nicméně podporuje také DLPI, Unix Domain Sockets, SCTP, atd..

Netperf využívá klient/server model. Klient se jmenuje netperf a server netserver – netserver je obvykle spuštěn pomocí RC scriptů v systému. Při spuštění programu netperf se jako první krok naváže TCP spojení na vzdálený systém, po kterém se přenáší nastavení a výsledky.

Při testování jsem měnil MTU a měřil jsem propustnost streamů a rychlosti odezev TCP a UDP protokolů. Zkusil jsem vliv Nagleho algoritmu a využití IP verze 6 oproti verzi 4.

[17]
[18]

8.1.2) Seznam možných testů

- TCP_STREAM
- TCP_SENDFILE
- TCP_MAERTS
- TCP_RR
- TCP_CRR
- UDP_STREAM
- UDP_RR
- DLCO_STREAM
- DLCO_RR
- DLCL_STREAM

- DLCL_RR
- STREAM_STREAM
- STREAM_RR
- DG_STREAM
- DG_RR
- LOC_CPU
- REM_CPU

Všechna měření jsem prováděl 10 krát po dobu 60 sekund pro jednotlivé MTU. Výsledek pak zprůměroval a použil.

8.1.3) Příprava na měření

Tato podsekce se zabývá zjištěním nejlepších parametrů a verze na testování tohoto software.

8.1.3.A Parametr – použito CPU

Určoval jsem, kolik bude nutno pro testování programu Netperfu přiřadit CPU. Každé měření TCP_STREAM při použití IP verze 4 o době trvání 10 sekund provedeno 5 krát, pro snížení chyby, poté jsem použil aritmetický průměr měření a odchylku pro MTU jsem porovnal s jinými měřeními.

Testovaný rozsah MTU [256 300 350 500 700 850 1000 1250 1500 2000 3000 4000 5000 6000 7000 9000]
Rozsah nastavovaných CPU [1 2 4 8 16 31 64 128]

Příklad scriptu určeného pro testování :

```
#!/bin/bash

#vypis hodnot kterych bude promena v cyklu nabyvat
for cpu in 1 2 4 8 16 32 64 128
do

     echo "Pouzito CPU: $cpu"
     for mtu in 256 300 350 500 700 850 1000 1250 1500 2000 3000 4000 5000
6000 7000 9000
     do

           echo  "MTU: $mtu"

           #nastaveni mtu na interface
           ifconfig eth10 mtu $mtu
```

```
            #testuj petkrat
            for iterace in   1 2 3 4 5
            do

                  #spust vlastni test s zadanymi paramerty
                  netperf -H 192.168.0.1 -n $cpu
            done

      done
done
```

Počet CPU	Rozdíl propustnosti oproti CPU=1 [%]
2	-0,4946459783
4	-0,5345546828
8	-1,0271034274
16	-0,6340905196
32	-0,6688590582
64	-0,7025370436
128	-0,6985325445

Tabulka 3: Procentuální rozdíl od propustnosti při počtu CPU 1 [%]

Výsledky ukázaly, že přidělení dalších CPU nemá prakticky vliv na výkonnost, čemuž odpovídají i výsledky z monitorování systému, kdy zatížení jednoho jádra při přidělení jednoho CPU nepřerostlo 5%. Při přidělení dalších CPU již nezpůsobilo zrychlení, ale naopak mělo negativní vliv a proto jsem všechny testy pouštěl na jednom CPU.

8.1.3.B Testované verze:

Porovnal jsem výkon v módu TCP_Stream ve verzích 2.4.4 – verze aktuálně obsažená v distribuci Debian a verzi 2.5.0 – což je nejnovější stabilní verze. Výsledky propustnosti se nelišily, proto jsem použil verzi 2.5.0.

8.1.4) Testy

8.1.4.A TCP Stream Performance IP verze 4

Popis

Vytvoří TCP/IP stream mezi lokálním systémem a cílovým systémem definovaným pomocí parametru -H, výsledek je propustnost.

Ukázka použití: 10 sekundový jednosměrný TCP Stream performance test.

```
netperf -H 147.32.125.182
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 147.32.125.182
(147.32.125.182) port 0 AF_INET : demo
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    10.00    6117.41
```

Testoval jsem pomocí deseti 60 sekundových testů pro každé jednotlivé MTU. Výsledky jsem zprůměroval a vykreslil z nich graf. Otestoval jsem také s parametrem NO_DELAY – tedy při vypnutém Nagleho algoritmu (popis algoritmu s ukázkou v kapitole tunning).

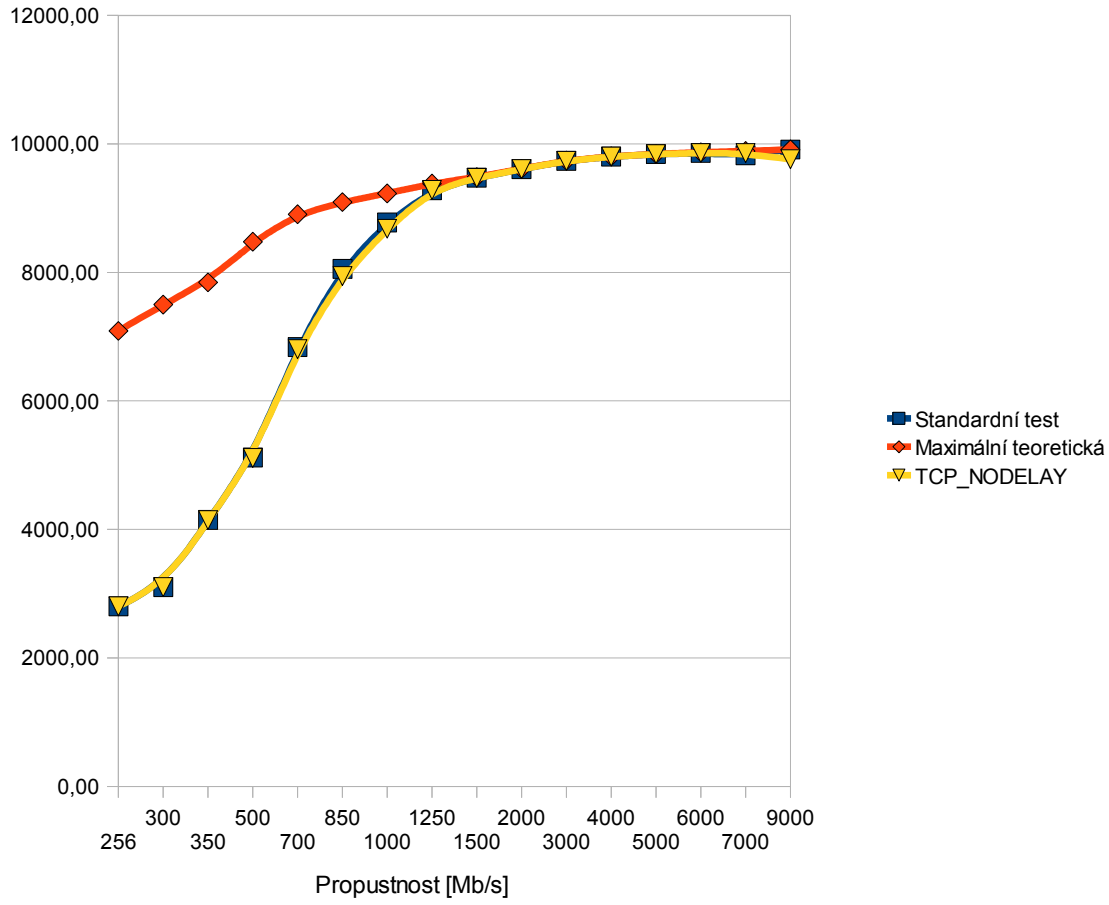
Výsledky měření IP verze 4

	Propustnost [Mb/s]	Aritmetický průměr propustnosti [Mb/s]	
MTU [B]	Maximální teoretická	TCP_NODELAY	Standardní test
256	7089,55	2803,94	2804,42
300	7500,00	3101,60	3102,42
350	7845,30	4145,77	4147,41
500	8476,56	5111,75	5117,38
700	8904,49	6804,27	6838,05
850	9095,13	7942,87	8052,70
1000	9229,25	8679,11	8775,82
1250	9381,93	9283,86	9280,26
1500	9484,13	9470,45	9473,96
2000	9612,33	9606,96	9607,00
3000	9741,04	9737,88	9737,90
4000	9805,58	9801,94	9801,89
5000	9844,37	9842,24	9842,17
6000	9870,26	9858,25	9858,35
7000	9888,76	9856,08	9824,68
9000	9913,45	9759,32	9912,32

Tabulka 4: Netperf: Změřená propustnost pro TCP/IP (v4) streamu v závislosti na MTU

	Odchylka od maximálního teoretického [%]	
MTU [B]	Standardní test	TCP_NODELAY
256	60,44	60,45
300	58,63	58,65
350	47,14	47,16
500	39,63	39,70
700	23,21	23,59
850	11,46	12,67
1000	4,91	5,96
1250	1,08	1,05
1500	0,11	0,14
2000	0,06	0,06
3000	0,03	0,03
4000	0,04	0,04
5000	0,02	0,02
6000	0,12	0,12
7000	0,65	0,33
9000	0,01	1,55

Tabulka 5: Netperf: Procentuální odchylka propustnosti pro TCP/IP (v4) streamu v závislosti na MTU



Ilustrace 6: Netperf: Graf změřené propustnosti pro TCP/IP (v4) stream v závislosti na MTU

Závěr měření IP verze 4

Toto měření ukazuje možnosti jednoho TCP/IP streamu. Výsledky ukazují, že se vzrůstajícím MTU se zvyšuje množství přenesených dat a tyto hodnoty se přibližují maximální teoretické datové propustnosti. Z výsledků měření vyplývá, že tento způsob injektuje v rozmezí MTU 1250-9000 B linku na 99% teoretického maxima. Pro nižší MTU nejsou již hodnoty i přeze všechny optimalizace tak vynikající, nicméně musíme vzít na vědomí, že nejnižší specifikované MTU pro IPv4 je 578 B – tedy provoz skládající se z rámců s nižší MTU je spíše nereálná záležitost. Nejčastější MTU z Ethernet prostředí je 1500 B, kde se podařilo dosáhnout propustnosti pouze o 2 MB nižší než je maximální teoretická (99,9% využití přenosového kanálu), což u aplikačního testu považuji vyloženě za úspěch.

Měřil jsem dvakrát pro otestování rozdílu aktivace Nagleho algoritmu. Předpokládal jsem, že tato vlastnost nebude mít u datového streamu vliv. Výsledky ukázaly vliv minimální, konkrétně průměrné zhoršení je menší než 0,2%. Toto zhoršení přisuzuji zařazením tohoto algoritmu do cesty výpočtů práce s pakety.

8.1.4.B TCP Stream Performance IP verze 6

Popis

Po změření TCP streamů u protokolu IP verze 4 jsem se rozhodl otestovat, jaký vliv bude mít použití IP verze 6.

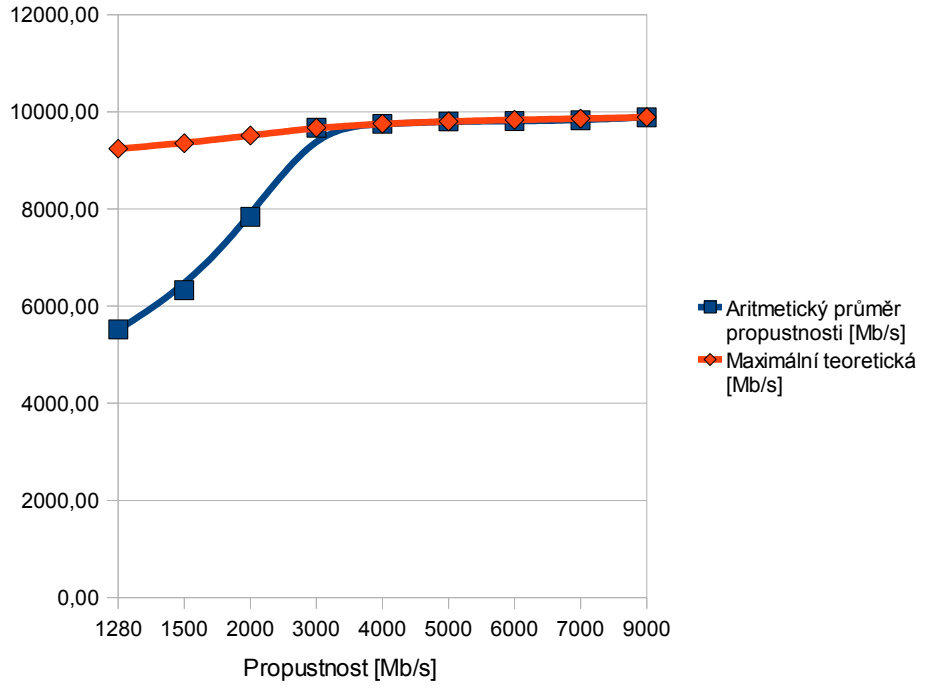
Výsledky měření IP verze 6

MTU [B]	Aritmetický průměr propustnosti [Mb/s]	Maximální teoretická [Mb/s]
1280	5520,48	9241,49
1500	6327,81	9351,85
2000	7840,42	9512,92
3000	9671,86	9674,63
4000	9752,33	9755,73
5000	9802,55	9804,47
6000	9808,40	9836,99
7000	9825,97	9860,24
9000	9890,22	9891,26

Tabulka 6: Netperf: Změřená propustnost pro TCP/IP (v6) streamu v závislosti na MTU

Odchylka od maximálního teoretického [%]	
MTU [B]	Standardní test
1280	40,26
1500	32,34
2000	17,58
3000	0,03
4000	0,03
5000	0,02
6000	0,29
7000	0,35
9000	0,01

Tabulka 7: Netperf: Procentuální odchylka propustnosti pro TCP/IP (v6) streamu v závislosti na MTU



Ilustrace 7: Netperf: Graf změřené propustnosti pro TCP/IP (v6) stream v závislosti na MTU

Závěr měření TCP/IP verze 6

Při měření propustnosti streamem TCP/IP verze 6 má křivka stejný tvar jako u verze 4, nicméně k optimálnímu využívání linky dochází až při MTU o velikost 3KB. Prozkoumal jsem zdrojový kód Netperfu a nenašel jsem žádné rozdíly při vytváření soketu pro protokol TCP/IP verze 4 a verze 6. Takže jsem zkoušel najít problém níže, v jádře. Ani v jádře jsem nenašel žádné nastavitelné optimalizace týkající se přímo IP verze 6. Po vyloučení přetížení HW, kdy množství obsazené paměti nepřekročilo 300MB (z 16GB) a využití jednoho z 16 procesorů nepřekročilo 5%, přisuzuji propad výkonnosti při nižších MTU spíše nižší optimalizaci HW síťové karty a jádra.

8.1.4.C UDP Stream Performance

Popis

Využívá pro testování UDP stream, principiálně je tento test stejný jako testování TCP streamem, nicméně UDP není spojově orientované, tedy roztřídění příchozího provozu je pro server daleko náročnější. Je třeba dát pozor na správné nastavení velikosti paketu, protože UDP nepodporuje fragmentaci.

```
netperf -H 147.32.125.182 -t UDP_STREAM -- -m 1024
UDP UNIDIRECTIONAL SEND TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
147.32.125.182 (147.32.125.182) port 0 AF_INET : demo
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

114688    1024   10.00     1154174      0     945.43
114688           10.00     1149567            941.66
```

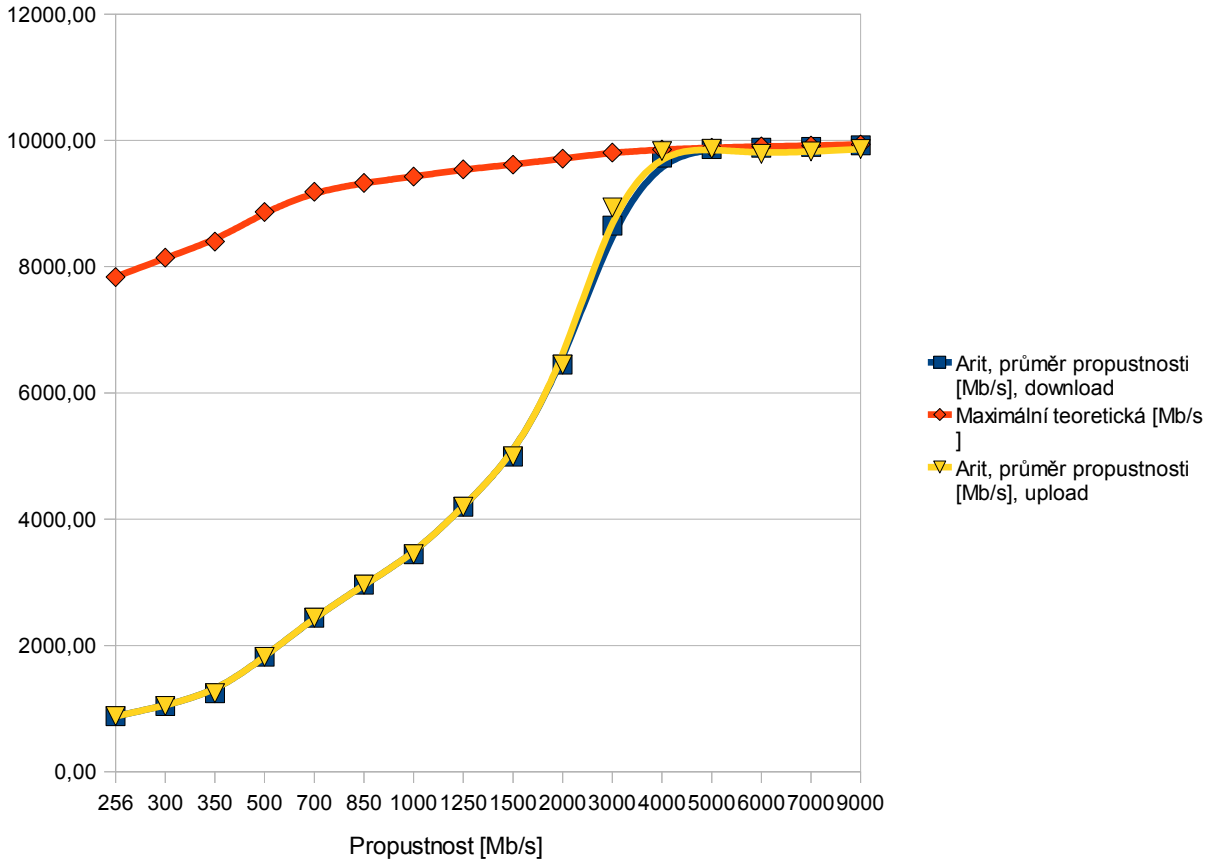
Výsledky měření

	Propustnost [Mb/s]	Změřený arit. průměr propustnosti [Mb/s]	
MTU [B]	Maximální teoretická	Upload	Download
256	7835,82	879,93	879,88
300	8141,03	1042,01	1041,99
350	8397,79	1244,78	1244,74
500	8867,19	1820,11	1820,01
700	9185,39	2439,57	2439,48
850	9327,15	2961,52	2960,96
1000	9426,88	3445,13	3444,94
1250	9540,41	4196,71	4196,50
1500	9616,40	4993,55	4993,22
2000	9711,73	6450,37	6449,80
3000	9807,44	8938,66	8650,98
4000	9855,43	9831,55	9726,78
5000	9884,28	9867,83	9861,50
6000	9903,53	9785,56	9882,47
7000	9917,28	9823,88	9895,46
9000	9935,64	9862,39	9923,82

Tabulka 8: Netperf: Změřená propustnost pro UDP/IP (v4) streamu v závislosti na MTU

	Odchylka od maximálního teoretického [%]	
MTU [B]	Upload	Download
256	790,50	790,56
300	681,28	681,30
350	574,64	574,66
500	387,18	387,21
700	276,52	276,53
850	214,94	215,00
1000	173,63	173,64
1250	127,33	127,34
1500	92,58	92,59
2000	50,56	50,57
3000	9,72	13,37
4000	0,24	1,32
5000	0,17	0,23
6000	1,21	0,21
7000	0,95	0,22
9000	0,74	0,12

Tabulka 9: Netperf: Procentuální odchylka propustnosti pro UDP/IP (v4) streamu v závislosti na MTU



Ilustrace 8: Netperf: Graf změřené propustnosti pro UDP/IP (v4) stream v závislosti na MTU

Závěr měření UDP streamů

Z výsledků prováděných testy u UDP streamu vyplývá, že efektivně jsem schopen linku utilizovat až při vysokých MTU. Toto je způsobeno tím, že testy jsou aplikační a měřím skutečnou propustnost – tedy co se podaří vyslat a co se podaří přijmout. Konkrétně u UDP přenosů jsem měl obrovské potíže s přijímáním paketů (příkaz: netstat -su, položka UDP, packet receive errors). Přírůstek tohoto počitadla při testech s malými MTU rostlo nade všechny meze. Částečně se mi podařil jeho růst eliminovat zvětšením netdev_max_backlog (zvětšení příchozí fronty – tunning popsán v podkapitole 7.6.Úprava bufferů), ale bohužel propustnosti srovnatelnou s TCP streamem se mi u nízkých MTU nepodařilo dosáhnout.

8.1.4.D TCP_SENDFILE Performance

Popis

Oproti TCP Stream Performance nepoužívá náhodná data, ale data ze souboru zadaném v parametru. Tedy nepřenáší se celý soubor, ale kousky dat z něho – kdyby tomu bylo jinak, docházelo by k problémům s rychlostí (v testovacím stroji byl disk s rozhraním SATA 1, jehož maximální teoretická rychlost je 150MB, což je 12% teoretické propustnosti 10G sítě). Já pro testování využil soubor DVD image instalátoru Debianu (popis v příloze v sekci soubory).

```
root@Obelix:~# netperf -H 192.168.0.1 -t TCP_SENDFILE -l 60 -C -c -F debian-
6.0.3-amd64-DVD-1.iso
TCP SENDFILE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.0.1
(192.168.0.1) port 0 AF_INET
Recv   Send    Send                          Utilization       Service Demand
Socket Socket  Message  Elapsed              Send     Recv     Send    Recv
Size   Size    Size     Time     Throughput  local    remote   local   remote
bytes  bytes   bytes    secs.    10^6bits/s  % S      % S      us/KB   us/KB

 87380  16384  16384    60.00      9378.96   1.35     7.55     0.189   0.528
```

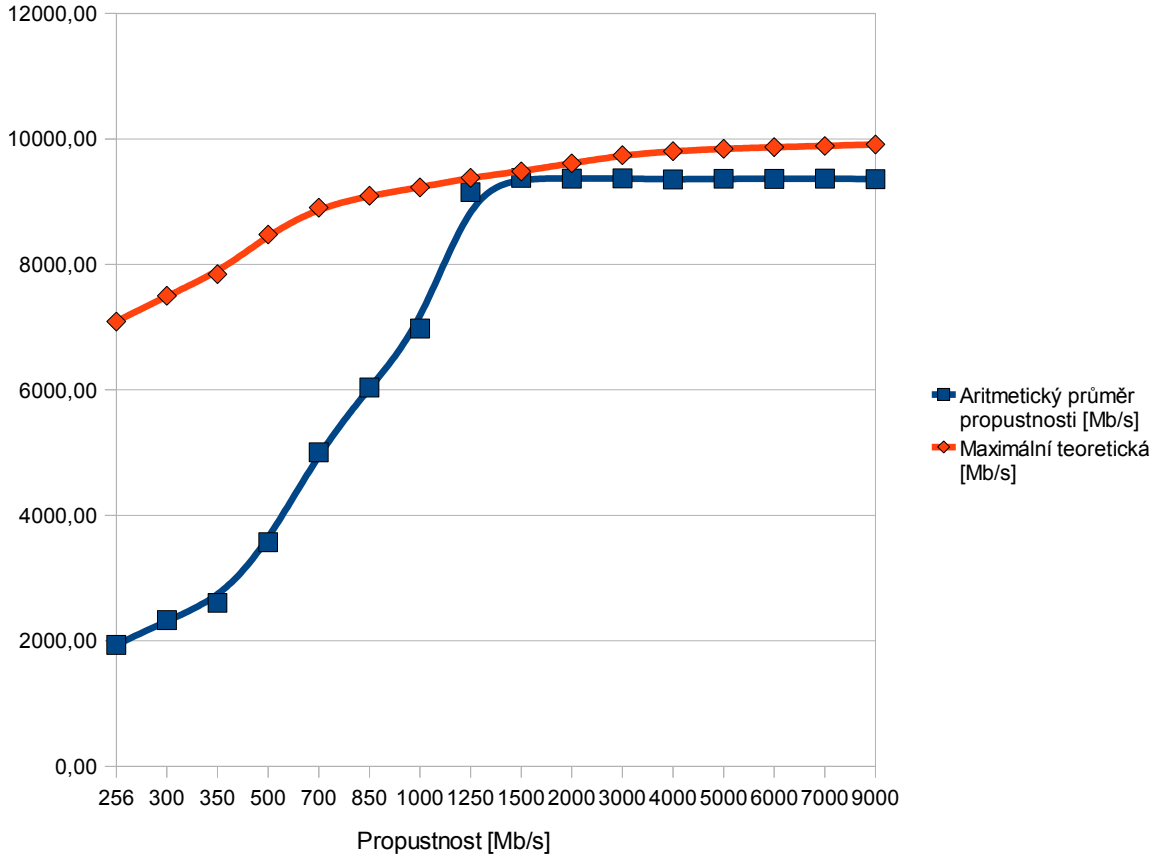
Výsledky měření

	Propustnost [Mb/s]	
MTU [B]	Maximální teoretická	Aritmetický průměr
256	7089,55	1932,58
300	7500,00	2328,17
350	7845,30	2603,79
500	8476,56	3573,12
700	8904,49	5005,05
850	9095,13	6037,43
1000	9229,25	6978,46
1250	9381,93	9152,97
1500	9484,13	9378,40
2000	9612,33	9366,37
3000	9741,04	9371,61
4000	9805,58	9354,23
5000	9844,37	9365,02
6000	9870,26	9363,26
7000	9888,76	9368,41
9000	9913,45	9360,91

Tabulka 10: Netperf: Změřená propustnost pro TCP/IP (v4) SENDFILE streamu v závislosti na MTU

MTU [B]	Odchylka od maximálního teoretického [%]
256	266,84
300	222,14
350	201,30
500	137,23
700	77,91
850	50,65
1000	32,25
1250	2,50
1500	1,13
2000	2,63
3000	3,94
4000	4,83
5000	5,12
6000	5,41
7000	5,55
9000	5,90

Tabulka 11: Netperf: Procentuální odchylka propustnosti pro TCP/IP (v4) Sendfile streamu v závislosti na MTU



Ilustrace 9: Netperf: Graf změřené propustnosti pro TCP/IP (v4) Sendfile streamu v závislosti na MTU

Závěr měření módu Sendfile

Očekával jsem stejnou propustnost jako u čistého TCP streamu, nicméně výsledky jsou zhruba o 5% nižší, což je patrně způsobeno neoptimalizacemi v Neperfu. Vzhledem k tomu, že funkční systém pro měření propustnosti TCP/IP streamu již máme, není třeba tento způsob testování používat.

8.1.4.E TCP Request/Response performance

Popis

Posílá 1B žádosti a očekává 1B odpověď, výsledkem je počet transakcí za sekundu. Vhodné použití jako ping, či ověření jak rychle je systém schopen zpracovat a reagovat na přijatá data. Bylo by zajímavé vložit do testovací sítě aplikační firewall a pozorovat změny v latenci.

```
root@Obelix:~# netperf -H 192.168.1.2 -t TCP_RR  -l 60
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.2 (192.168.1.2) port 0 AF_INET : demo
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

65536  87380  1        1       60.00    57826.16
65536  87380
```

Maximální teoretické množství transakcí je z hlediska vyslání paketu:

$$Maximální\ teoretické\ množství\ transakcí = \frac{Rychlost\ linky[b/s]}{(\sum Velikost\ paketu + \sum Interframe\ Gap)}$$

(tento vzorec bere v úvahu jen čas na vyslání paketu, nikoli záležitosti v OS a ovladači, výsledek je tedy stav, kdy by systém na paket odpověděl hned po přijetí, dobu přenosu paketu zanedbávám)

$$Maximální\ teoretické\ množství\ transakcí = \frac{10^9[b/s]}{2154[b]*4} \approx 464253$$

Výsledky měření

Bylo měřeno po dobu 60 sekund a průměrně se uskutečnilo 57826.16 transakcí za sekundu.

Závěr

57826.16 transakcí za sekundu, odpovídá průměrné latenci 1/57826.16 = 17,293µs. Tato hodnota odpovídá 12,5% teoretické latence, tento stav by šel zlepšit změnou rychlosti zpracovávání přerušení.

8.1.4.F UDP Request/Response performance

Popis

Princip testu je stejný jako u TCP REQUEST/RESPONSE testu, nicméně je použit protokol UDP – tedy není zde navázané spojení mezi systémy.

```
root@Obelix:~# netperf -H 192.168.1.2 -t UDP_RR  -l 60
UDP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.2 (192.168.1.2) port 0 AF_INET : demo
Local /Remote
```

```
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

126976 126976 1        1       60.00    71970.16
126976 126976
```

Maximální teoretické množství transakcí je z hlediska vyslání paketu:

$$Maximální\ teoretické\ množství\ transakcí = \frac{Rychlost\ linky[b/s]}{(\sum Velikost\ paketu + \sum Interframe\ Gap)}$$

(tento vzorec bere v úvahu jen čas na vyslání paketu, nikoli záležitosti v OS a ovladači, výsledek je tedy stav, kdy by systém na paket odpověděl hned po přijetí, dobu přenosu paketu zanedbávám)

$$Maximální\ teoretické\ množství\ transakcí = \frac{10^9[b/s]}{344[b]*2} \approx 1453488$$

Výsledky měření

Bylo měřeno po dobu 60 sekund a průměrně se uskutečnilo 71970.16 transakcí za sekundu.

Závěr

71970.16 transakcí za sekundu, odpovídá průměrné latenci 1/71970.16 = 13,895µs. Ač se může zdát zvýšení této hodnoty oproti TCP jako zrychlení, reálně jde o snížení způsobené nižší velikostí paketů a jejich polovičním počtem na transakci. Tento výkonnostní rozdíl přisuzuji rozdílným optimalizacím u TCP a UDP. Při testu s UDP je dosahujeme 5% teoretické hodnoty.

8.1.4.G TCP CRR Request/Response performance

Popis

Oproti TCP REQUEST/RESPONSE testu, kdy jsou všechna data přenášena přes jedno spojení, je zde pro každou iteraci otevřeno a uzavřeno spojení, díky tomu lze otestovat optimalizaci systému a jiných zařízení jako firewallů po cestě na zpracovávání nových spojení.

```
root@Obelix:~#netperf -H 192.168.1.2 -t TCP_CRR -l 60
TCP Connect/Request/Response TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.2 (192.168.1.2) port 0 AF_INET : demo
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
```

```
bytes  Bytes  bytes    bytes   secs.    per sec

65536  87380  1        1       60.00    17470.66
65536  87380
```

No.	Time	Source	Destination	Protocol	Length	Info
28	0.001072	127.0.0.1	127.0.0.2	TCP	74	24546 > 52101 [SYN] Seq=0 Win=32792 Len=0 MSS=16396 SACK_PERM=1 TSval=17688754 TSecr=0 WS=64
29	0.001081	127.0.0.2	127.0.0.1	TCP	74	52101 > 24546 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=16396 SACK_PERM=1 TSval=17688754 TSecr=17688754
30	0.001090	127.0.0.1	127.0.0.2	TCP	66	24546 > 52101 [ACK] Seq=1 Ack=1 Win=32832 Len=0 TSval=17688754 TSecr=17688754
31	0.001131	127.0.0.1	127.0.0.2	TCP	67	24546 > 52101 [PSH, ACK] Seq=1 Ack=1 Win=32832 Len=1 TSval=17688754 TSecr=17688754
32	0.001152	127.0.0.2	127.0.0.1	TCP	66	52101 > 24546 [ACK] Seq=1 Ack=2 Win=32768 Len=0 TSval=17688754 TSecr=17688754
33	0.001162	127.0.0.2	127.0.0.1	TCP	67	52101 > 24546 [PSH, ACK] Seq=1 Ack=2 Win=32768 Len=1 TSval=17688754 TSecr=17688754
34	0.001168	127.0.0.2	127.0.0.1	TCP	66	52101 > 24546 [FIN, ACK] Seq=2 Ack=2 Win=32768 Len=0 TSval=17688754 TSecr=17688754
35	0.001206	127.0.0.1	127.0.0.2	TCP	66	24546 > 52101 [ACK] Seq=2 Ack=2 Win=32832 Len=0 TSval=17688754 TSecr=17688754
36	0.001226	127.0.0.1	127.0.0.2	TCP	66	24546 > 52101 [FIN, ACK] Seq=2 Ack=3 Win=32832 Len=0 TSval=17688754 TSecr=17688754
37	0.001235	127.0.0.2	127.0.0.1	TCP	66	52101 > 24546 [ACK] Seq=3 Ack=3 Win=32768 Len=0 TSval=17688754 TSecr=17688754

Ilustrace 10: Netperf: Ukázka jedné CRR transakce

Maximální teoretické množství transakcí je z hlediska vyslání paketu:

$$Maximální\ teoretické\ množství\ transakcí = \frac{Rychlost\ linky\,[b/s]}{(\sum Velikost\ paketu + \sum Interframe\,Gap)}$$

(tento vzorec bere v úvahu jen čas na vyslání paketu, nikoli záležitosti v OS a ovladači, výsledek je tedy stav, kdy by systém na paket odpověděl hned po přijetí, dobu přenosu paketu zanedbávám)

$$Maximální\ teoretické\ množství\ transakcí = \frac{10^9[b/s]}{5056[b]} \approx 197785$$

Výsledky měření

Bylo měřeno po dobu 60 sekund a průměrně se uskutečnilo 17470.66 transakcí za sekundu.

Závěr

Měření ukázalo, že testovací stroje zvládnou vytvořit spojení, přenést data a uzavřít spojení průměrně za 57,239 µs. Což je mimochodem 8,8% z teoretického maxima daného linkovou rychlostí.

8.1.5) Závěr testování programu Netperf

Program Netperf se ukázal jako skvělý pro testování propustnosti komplexních sítí TCP/IPv4 streamem, kdy v rozmezí nejčastějšího MTU (tedy 1250-9000 B) linku vytěžoval na více jak 99% teoretického maxima. Při použití protokolu UDP jsem narazil na problémy při příjmu paketů, jenž způsobovaly dosažení plné propustnosti až při vyšších MTU.

Z hlediska testování latence byly dosažené výsledky skvělé a pohybovaly se okolo 10% z teoretického maxima daného linkovou rychlostí.

8.2. Iperf

8.2.1) Popis

Nástroj na testování sítě – vytváří TCP a UDP datové streamy. Architektura klient/server. Verze pro Windows i Linux.
Možnosti:

- Ipv4
- IPv6
- Více paralelních streamů
- Změna velikosti TCP okna
- Vypnutí Nagle algoritmu

[19]

8.2.2) Použití

Spuštění serveru (parametr -s), řádky později ukazují již běžící test z druhé strany (tedy připojení klienta na server)

```
iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 127.0.0.1 port 5001 connected with 127.0.0.1 port 57252
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5001
TCP window size:   560 KByte (default)
------------------------------------------------------------
[  6] local 127.0.0.1 port 57253 connected with 127.0.0.1 port 5001
[  5] local 127.0.0.1 port 5001 connected with 127.0.0.1 port 57253
Waiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  3.50 GBytes  3.00 Gbits/sec
[  6]  0.0-10.0 sec  3.42 GBytes  2.94 Gbits/sec
[  5]  0.0-10.0 sec  3.42 GBytes  2.93 Gbits/sec
[SUM]  0.0-10.0 sec  6.92 GBytes  5.93 Gbits/sec
```

Spuštění klienta

```
iperf -c 127.0.0.1
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5001
TCP window size: 49.4 KByte (default)
------------------------------------------------------------
[  4] local 127.0.0.1 port 57252 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  3.50 GBytes  3.00 Gbits/sec
```

8.2.3) Výsledky měření

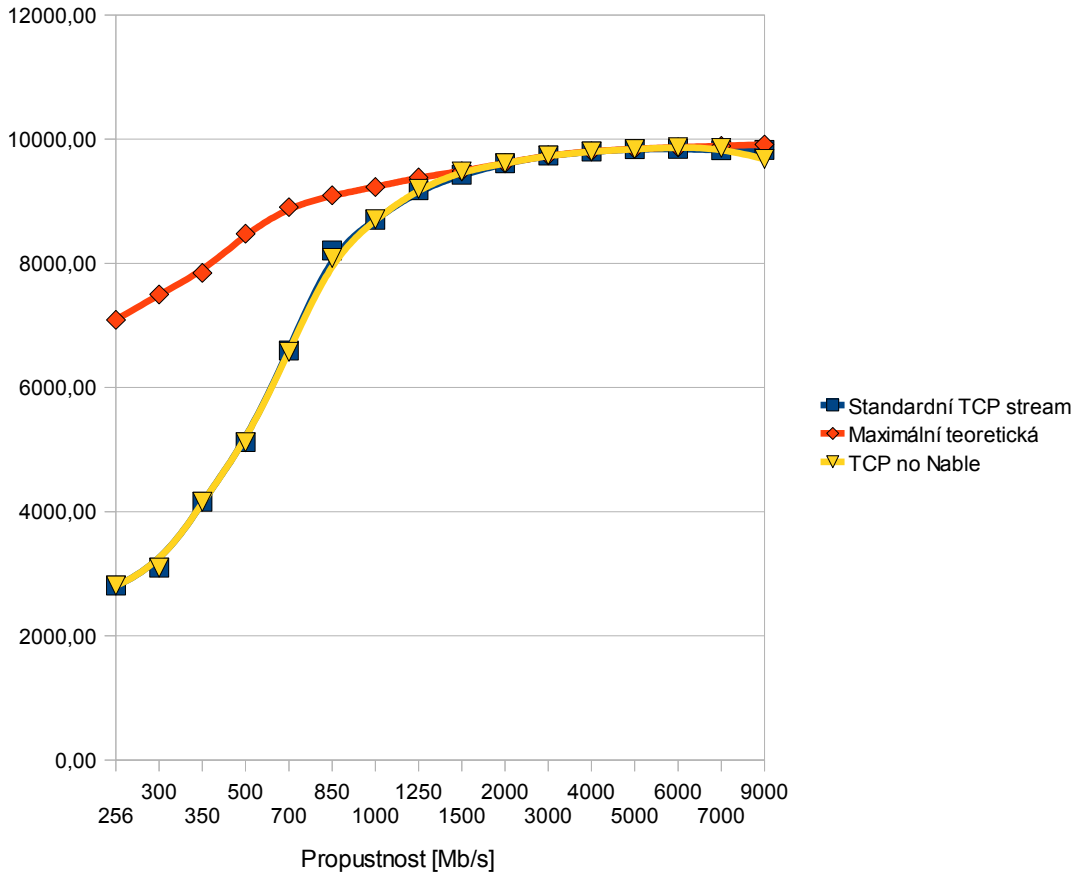
Měřil jsem TCP stream s aktivovaným a deaktivovaným Nagle algoritmem. Každá hodnota je tvořena aritmetickém průměrem z 10 měření trvající 60 sekund.

	Propustnost [Mb/s]	Aritmetický průměr propustnosti [Mb/s]	
MTU [B]	Maximální teoretická	TCP no Nable	Standardní TCP stream
256	7089,55	2810,00	2810,00
300	7500,00	3100,00	3100,00
350	7845,30	4160,00	4160,00
500	8476,56	5118,00	5120,00
700	8904,49	6574,00	6596,67
850	9095,13	8088,00	8207,78
1000	9229,25	8710,00	8702,22
1250	9381,93	9201,00	9177,78
1500	9484,13	9480,00	9425,56
2000	9612,33	9610,00	9610,00
3000	9741,04	9738,00	9736,67
4000	9805,58	9800,00	9800,00
5000	9844,37	9840,00	9840,00
6000	9870,26	9870,00	9845,56
7000	9888,76	9858,00	9821,11
9000	9913,45	9679,00	9827,78

Tabulka 12: Iperf: Změřená propustnost pro TCP/IP (v4) streamu v závislosti na MTU

	Odchylka od maximálního teoretického [%]	
MTU [B]	TCP_NODELAY	Standardní test
256	152,30	152,30
300	141,94	141,94
350	88,59	88,59
500	65,62	65,56
700	35,45	34,98
850	12,45	10,81
1000	5,96	6,06
1250	1,97	2,22
1500	0,04	0,62
2000	0,02	0,02
3000	0,03	0,04
4000	0,06	0,06
5000	0,04	0,04
6000	0,00	0,25
7000	0,31	0,69
9000	2,42	0,87

Tabulka 13: Iperf: Procentuální odchylka propustnosti pro TCP/IP (v4) streamu v závislosti na MTU



Ilustrace 11: Iperf: Graf změřené propustnosti pro TCP/IP (v4) stream v závislosti na MTU

8.2.4) Závěr

Výsledky ukazují, že se vzrůstajícím MTU se zvyšuje množství přenesených dat a tyto hodnoty se přibližují maximální teoretické datové propustnosti. V rozmezí MTU 1500-9000 B přenášíme linkou více jak 99% teoretického maxima. Lze říci, že naměřené hodnoty jsou pro rozmezí 1500-9000B lepší než u Netperfu, nicméně výsledky nejsou tak stabilní. Pro nižší hodnoty MTU je zde propastný rozdíl vzhledem k výkonnosti Netperfu. Při testování tohoto programu se žádný negativní vliv aktivace Nagleho algoritmu se neprojevil.

8.3. Přenos pomocí FTP –Wget a Curl vs. Proftpd

8.3.1) Popis

Zde zkouším využít standardní unixové aplikace FTP server Proftpd oproti klientovy Wget a Curl na vytvoření aplikačního testu. Na straně serveru jsou soubory umístěny v cache – vynuceno vhodnou úpravou scriptu. Na straně klienta jsou data souboru přeposílána do /dev/null.

GNU Wget 1.12
Proftpd 1.3.3a
Curl 7.20.0

8.3.2) Testovací soubory:

Pro testování jsem zvolil soubory tří velikostí. U velikosti souboru jsem narazil na limit paměti, protože ve chvíli kdy by soubor nebyl poskytován z cache, už bych neměřil propustnost sítě, ale propustnost datového úložiště, které svou propustností nebylo schopné 10G síťovým kartám konkurovat. Největší soubor proto má 647 MB. Zbylé soubory mají odstupňované velikosti. Jednotlivé soubory popsány v kapitole: 13. Příloha 3. - Soubory použité na testy.

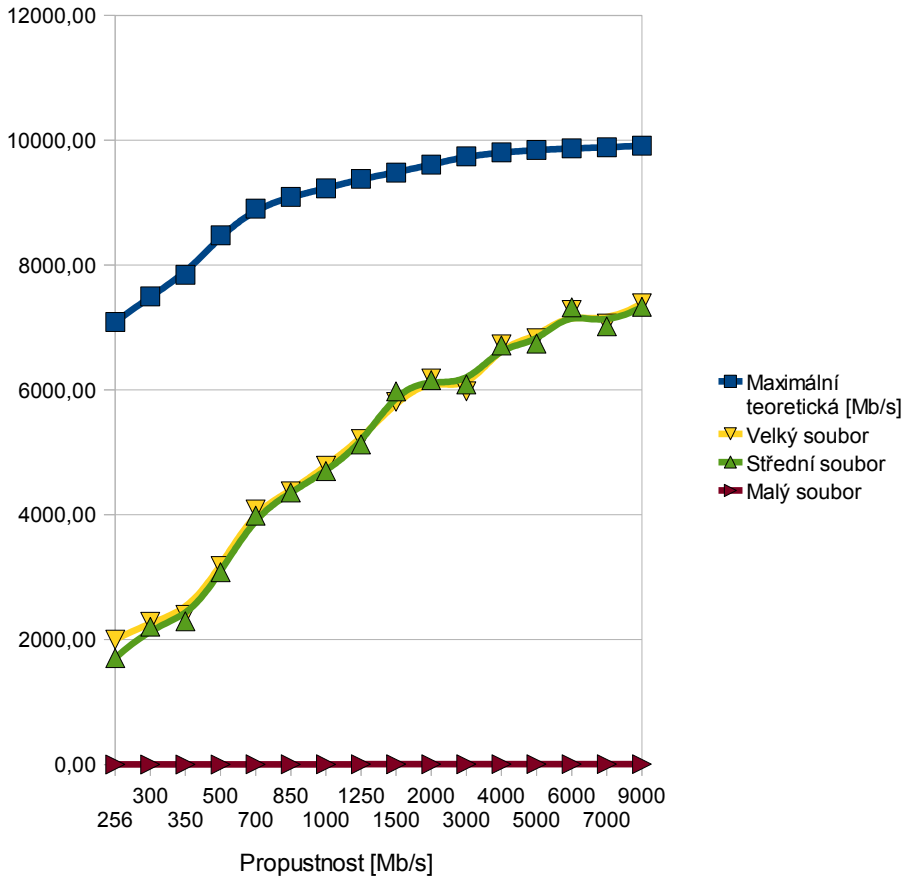
8.3.3) Naměřené hodnoty pro Wget

		Arit, Průměr propustnosti [Mb/s]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	2000,00	1704,80	1,07
300	7500,00	2286,40	2208,00	1,17
350	7845,30	2398,40	2294,40	1,81
500	8476,56	3180,80	3079,20	2,16
700	8904,49	4086,40	3986,40	3,44
850	9095,13	4375,20	4360,00	3,93
1000	9229,25	4784,80	4699,20	4,18
1250	9381,93	5214,40	5126,40	5,12
1500	9484,13	5810,40	5976,80	5,23
2000	9612,33	6181,60	6158,22	5,52
3000	9741,04	5979,20	6088,00	5,87
4000	9805,58	6730,40	6713,60	6,41
5000	9844,37	6830,40	6742,22	6,76
6000	9870,26	7284,00	7324,00	7,12
7000	9888,76	7064,00	7019,20	6,58
9000	9913,45	7387,20	7332,00	6,37

Tabulka 14: Wget vs. Proftpd - Změřená propustnost pro TCP/IP (v4) v závislosti na MTU

		Odchylka od maximální teoretické rychlosti [%]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	254,48	315,86	663986,15
300	7500,00	228,03	239,67	638566,79
350	7845,30	227,11	241,93	432181,24
500	8476,56	166,49	175,28	392541,79
700	8904,49	117,91	123,37	258756,64
850	9095,13	107,88	108,60	231049,52
1000	9229,25	92,89	96,40	220627,05
1250	9381,93	79,92	83,01	182986,66
1500	9484,13	63,23	58,68	181203,91
2000	9612,33	55,50	56,09	174170,54
3000	9741,04	62,92	60,00	165800,77
4000	9805,58	45,69	46,06	152983,29
5000	9844,37	44,13	46,01	145499,99
6000	9870,26	35,51	34,77	138583,76
7000	9888,76	39,99	40,88	150109,40
9000	9913,45	34,20	35,21	155531,96

Tabulka 15: Wget vs. Proftpd - Procentuální odchylka propustnosti pro TCP/IP (v4) v závislosti na MTU



Ilustrace 12: Wget vs. Proftpd - Graf změřené propustnosti pro TCP/IP (v4) v závislosti na MTU

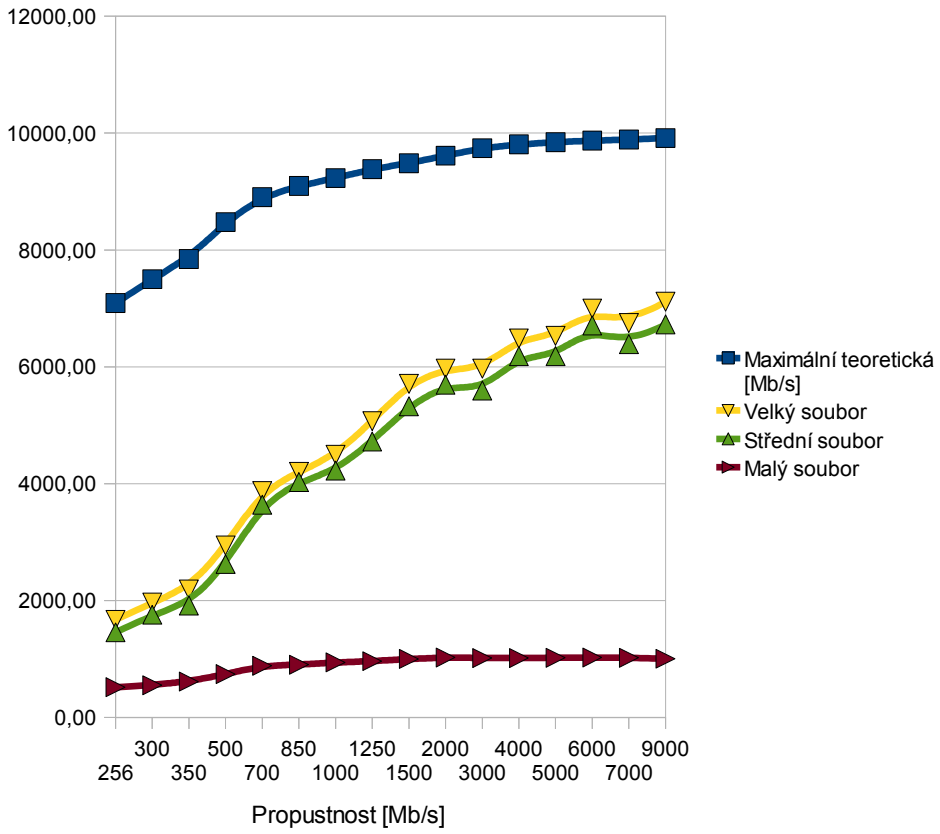
8.3.4) Naměřené hodnoty pro CURL

		Aritmetický průměr propustnosti [Mb/s]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	1672,00	1456,80	519,52
300	7500,00	1965,60	1759,20	553,28
350	7845,30	2188,80	1915,20	615,20
500	8476,56	2940,80	2625,60	742,24
700	8904,49	3874,40	3642,40	885,60
850	9095,13	4200,80	4028,80	904,00
1000	9229,25	4494,40	4227,20	942,40
1250	9381,93	5069,60	4724,00	964,00
1500	9484,13	5708,00	5324,80	1001,60
2000	9612,33	5968,00	5693,60	1027,20
3000	9741,04	5968,00	5594,40	1019,20
4000	9805,58	6483,20	6183,20	1018,40
5000	9844,37	6528,00	6181,60	1023,20
6000	9870,26	6997,60	6708,00	1028,00
7000	9888,76	6748,80	6390,40	1022,40
9000	9913,45	7115,20	6724,80	1006,40

Tabulka 16: Curl vs. Proftpd - Změřená propustnost pro TCP/IP (v4) v závislosti na MTU

		Odchylka od maximální teoretické rychlosti [%]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	324,02	386,65	1264,63
300	7500,00	281,56	326,33	1255,55
350	7845,30	258,43	309,63	1175,24
500	8476,56	188,24	222,84	1042,02
700	8904,49	129,83	144,47	905,48
850	9095,13	116,51	125,75	906,10
1000	9229,25	105,35	118,33	879,33
1250	9381,93	85,06	98,60	873,23
1500	9484,13	66,16	78,11	846,90
2000	9612,33	61,06	68,83	835,78
3000	9741,04	63,22	74,12	855,75
4000	9805,58	51,25	58,58	862,84
5000	9844,37	50,80	59,25	862,12
6000	9870,26	41,05	47,14	860,14
7000	9888,76	46,53	54,74	867,21
9000	9913,45	39,33	47,42	885,04

Tabulka 17: Curl vs. Proftpd - Procentuální odchylka propustnosti pro TCP/IP (v4) v závislosti na MTU



Ilustrace 13: Curl vs. Proftpd - Graf změřené propustnosti pro TCP/IP (v4) v závislosti na MTU

8.3.5) Závěr

Bohužel jsem dospěl k závěru, že nelze takto jednoduše testovat prostupnost. Dosažená utilizace linky je maximálně 61% u MTU 9KB. Tento test nicméně prokázal, že se zvyšujícími MTU stoupá propustnost a efektivita přenosů i u reálných aplikací jako je FTP server Proftpd použitý společně s downloadery a nikoli jen u programů určených k testování sítí.

Původně jsem také zvažoval testovat s multithread downloader Axel. Bohužel jsem narazil na několik problémů, kvůli nimž se ukázal nepoužitelný. Například neumí zahodit výstup (request #312194 z 29.12.2009) a tedy se snaží soubor ukládat fyzicky na disk, nehledě na to, že při nastavení vyššího množství vláken padal se seqmentation fail – debugovací výpisy pro nahlášení této chyby mám připraveny a chybu budu reportovat.

8.4. Přenos pomocí HTTP – přenos Wget a Curl vs. Apache

8.4.1) Popis

Zde zkouším využít standardní unixovské aplikace http server Apache oproti klientovy Wget a Curl na vytvoření aplikačního testu. Na straně serveru jsou soubory umístěny v cache – vynuceno vhodnou úpravou scriptu. Na straně klienta jsou data souboru přeposílána do /dev/null.

GNU Wget 1.12
Apache 2.2.16
Curl 7.20.0

8.4.2) Testovací soubory:

Jako testovací soubory byly použité stejné postupy jako v kapitole 8.3.2 Přenos pomocí FTP –Wget a Curl vs. Proftpd - Testovací soubory:. Tedy tři testovací soubory s odstupňovanou velikostí. Podrobně jsou soubory popsány v kapitole: 13. Příloha 3. - Soubory použité na testy.

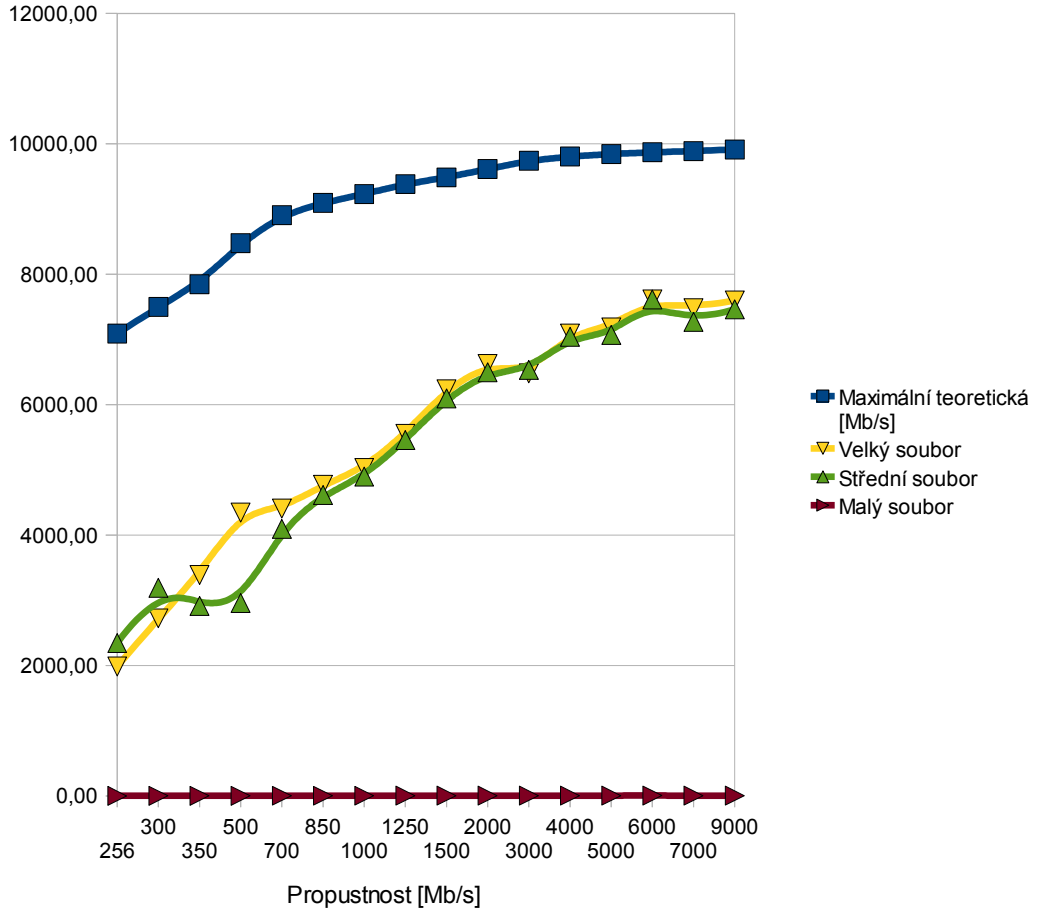
8.4.3) Naměřené hodnoty pro Wget

		Aritmetický průměr propustnosti [Mb/s]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	1984,80	2343,20	1,07
300	7500,00	2723,20	3192,00	1,81
350	7845,30	3387,20	2912,00	2,14
500	8476,56	4346,40	2957,60	2,14
700	8904,49	4404,00	4095,20	2,81
850	9095,13	4768,00	4616,00	3,00
1000	9229,25	5028,00	4898,40	3,27
1250	9381,93	5552,80	5456,00	3,53
1500	9484,13	6233,60	6096,80	3,65
2000	9612,33	6623,20	6497,60	3,90
3000	9741,04	6481,60	6532,00	4,48
4000	9805,58	7090,40	7041,60	4,38
5000	9844,37	7185,60	7068,00	4,45
6000	9870,26	7614,40	7612,00	5,02
7000	9888,76	7477,60	7268,80	4,24
9000	9913,45	7597,60	7460,80	4,15

Tabulka 18: Wget vs. Apache - Změřená propustnost pro TCP/IP (v4) v závislosti na MTU

		Odchylka od maximální teoretické rychlosti [%]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	257,19	202,56	663986,15
300	7500,00	175,41	134,96	413154,98
350	7845,30	131,62	169,41	367339,05
500	8476,56	95,02	186,60	396904,47
700	8904,49	102,19	117,44	316453,61
850	9095,13	90,75	97,03	303136,83
1000	9229,25	83,56	88,41	282464,97
1250	9381,93	68,96	71,96	265824,14
1500	9484,13	52,15	55,56	259735,27
2000	9612,33	45,13	47,94	246423,52
3000	9741,04	50,29	49,13	217151,00
4000	9805,58	38,29	39,25	223924,33
5000	9844,37	37,00	39,28	221211,98
6000	9870,26	29,63	29,67	196614,55
7000	9888,76	32,25	36,04	232952,53
9000	9913,45	30,48	32,87	238878,98

Tabulka 19: Wget vs. Apache - Procentuální odchylka propustnosti pro TCP/IP (v4) v závislosti na MTU



Ilustrace 14: Wget vs. Apache - Graf změřené propustnosti proTCP/IP (v4) v závislosti na MTU

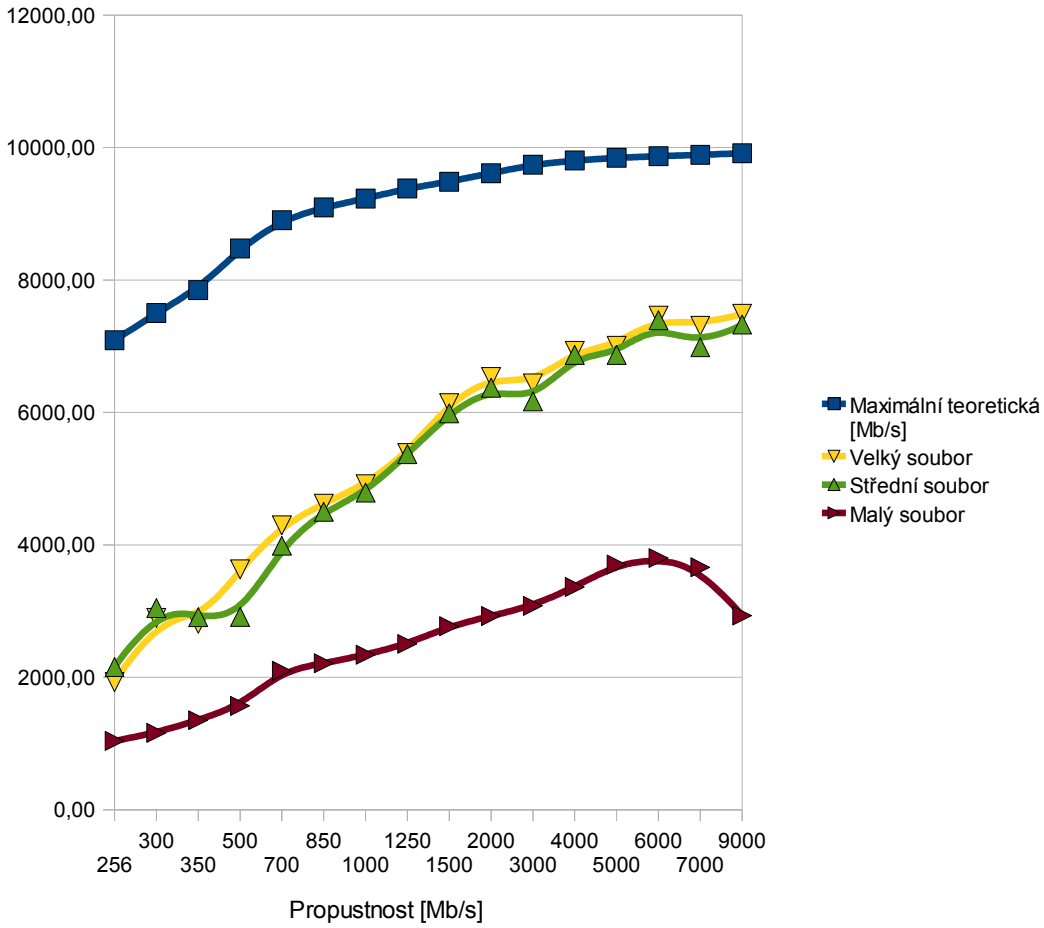
8.4.4) Naměřené hodnoty pro CURL

		Aritmetický průměr propustnosti [Mb/s]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	1925,33	2155,20	1039,20
300	7500,00	2898,67	3047,20	1160,80
350	7845,30	2808,89	2907,20	1352,80
500	8476,56	3629,33	2912,80	1569,60
700	8904,49	4293,33	3986,40	2098,40
850	9095,13	4622,22	4500,80	2211,20
1000	9229,25	4915,56	4790,40	2340,00
1250	9381,93	5387,56	5371,20	2504,00
1500	9484,13	6141,33	5987,20	2771,20
2000	9612,33	6533,33	6373,60	2926,40
3000	9741,04	6436,44	6168,00	3080,00
4000	9805,58	6928,00	6867,20	3363,20
5000	9844,37	7006,22	6869,60	3701,60
6000	9870,26	7461,33	7392,80	3801,60
7000	9888,76	7308,44	6987,20	3660,80
9000	9913,45	7487,11	7327,20	2932,00

Tabulka 20: Curl vs. Apache - Změřená propustnost pro TCP/IP (v4) v závislosti na MTU

		Odchylka od maximální teoretické rychlosti [%]		
MTU [B]	Maximální teoretická [Mb/s]	Velký soubor	Střední soubor	Malý soubor
256	7089,55	268,22	228,95	582,21
300	7500,00	158,74	146,13	546,11
350	7845,30	179,30	169,86	479,93
500	8476,56	133,56	191,01	440,05
700	8904,49	107,40	123,37	324,35
850	9095,13	96,77	102,08	311,32
1000	9229,25	87,76	92,66	294,41
1250	9381,93	74,14	74,67	274,68
1500	9484,13	54,43	58,41	242,24
2000	9612,33	47,13	50,81	228,47
3000	9741,04	51,34	57,93	216,27
4000	9805,58	41,54	42,79	191,56
5000	9844,37	40,51	43,30	165,95
6000	9870,26	32,29	33,51	159,63
7000	9888,76	35,31	41,53	170,13
9000	9913,45	32,41	35,30	238,11

Tabulka 21: Curl vs. Apache - Procentuální odchylka propustnosti pro TCP/IP (v4) v závislosti na MTU



Ilustrace 15: Curl vs. Apache - Graf změřené propustnosti pro TCP/IP (v4) v závislosti na MTU

8.4.5) Závěr

Také při tomto testování oproti HTTP serveru Apache jsem se dostal k prakticky stejným výsledkům jako u testování serveru Proftpd. A to tedy - nelze takto jednoduše testovat prostupnost. Dosažená utilizace linky je mezi 60-70% u vysokých MTU. Opět je zde vidět zrychlení přenosů v závislosti na MTU. Zajímavý je propad rychlosti při vyšších MTU u malého souboru, pro které jsem nenašel vysvětlení, ač se tento propad vyskytoval i při opakování testů.

8.5. Curl-loader

8.5.1) Popis

Open-source nástroj napsaný v jazyku C licencovaný pod GPLv2 je určen k simulování zátěže generovaného velkým množstvím klientů. Srovnává se s profesionálními nástroji jako Spirent Avalanche a IXIA IxLoad – toto srovnání bohužel nemohu potvrdit – tyto nástroje jsem neměl to štěstí použít, nicméně z mých pokusů vyplývá, že je to velmi mocný nástroj, který dovede při spuštění i na slabším klientu přetížit či úplně zahltit velké farmy serverů.

- podporuje 2 500 - 100 000 simultálních klientů, ale toto číslo je určeno hardwarem nikoli omezením v programu, je tedy možné zvýšením výkonu HW použít i více klientů. Měl by zvládat cca 10 000 klientů na jednom jádru 3.2 GHz Intel nebo 2.2 GHz AMD s 1.5-2 GB RAM.
- Jednotliví klienti pak mohou vykonávat aktivity jako načítání stránek, procházení FTP serveru, přihlašování/odhlašování se atd.
- Tito klienti mohou všichni sdílet jednu adresu, či vyhraněný pool adres či mít každý unikátní adresu.
- Přidávání uživatelů lze provést jak popisem jejich přírůstku ve scriptu, tak jejich počet manuálně upravovat při běhu testu
- Podpora IPv4 a IPv6
-
- Podpora většiny z funkcí protokolů: HTTP, HTTPS, FTP, FTPS , tedy:
  - HTTP login, logout (POST, GET+POST) zvlášt pro každého klienta
  - HTTP Web Proxy Autentizace (Basic, Digest a NTLM)
  - HTTP 3xx redirekce
  - HTTP cookies
  - HTTP/FTP download/upload passive/active
  - Možnost plného specifikování HTTP/FTP hlavičky
  - Datové limity pro každého klienta/URL
  - Znovupoužití TCP spojení
  - Libovolné kombinování jednotlivých requestů
  - - Statistika každého klienta zalogována

8.5.2) Použití

Příklad konfiguračního souboru:

po spuštění curl-loader -t 4 -f <soubor>

-parametr -t určuje kolika vlákny má Curl-loader běžet

```
########### GENERAL SECTION ###############################
```

```
BATCH_NAME= bulk
CLIENTS_NUM_MAX=1000 # Same as CLIENTS_NUM
CLIENTS_NUM_START=50
CLIENTS_RAMPUP_INC=50
INTERFACE   =eth0
NETMASK=24
IP_ADDR_MIN= 147.32.125.182
IP_ADDR_MAX= 147.32.125.182
IP_SHARED_NUM=1
CYCLES_NUM= 1
URLS_NUM= 1

########### URL SECTION ###################################

URL=http://www.seznam.cz
URL_SHORT_NAME="index.html"
REQUEST_TYPE=GET
TIMER_URL_COMPLETION = 5000
TIMER_AFTER_URL_SLEEP = 500
```

Tento script začne s 50 klienty (CLIENTS_NUM_START) a každou sekundu přidá 50 dalších (CLIENTS_RAMPUP_INC) a bude pokračovat dokud nedosáhne hodnoty 1000 klientů (CLIENTS_NUM_MAX). Provede celkem jedno opakování (CYCLES_NUM). Každý klient požádá o stránku určenou parametrem (URL) a po jejím odeslání si poznamená návratový kód

Příklad konfigurace pro vytvoření jednoho klienta, který bude uploadovat soubor na server.

```
########### GENERAL SECTION ###############################
BATCH_NAME= ftp-upload
CLIENTS_NUM_MAX=1
INTERFACE   = eth0
NETMASK=20
IP_ADDR_MIN= 192.168.1.1
IP_ADDR_MAX= 192.168.5.255  #Actually - this is for self-control
CYCLES_NUM= 3
URLS_NUM = 1

########### URL SECTION ###################################

#Don't forget to allow over-write of files in your FTP-server

URL=ftp://ftp:heslo@xsimi.sh.cvut.cz:61616/data.dat
FRESH_CONNECT=1   # At least my proftpd has problems with connection re-use
UPLOAD_FILE=./data.dat
TIMER_URL_COMPLETION = 0 # In msec. When positive, Now it is enforced by
cancelling url fetch on timeout
TIMER_AFTER_URL_SLEEP =10000
```

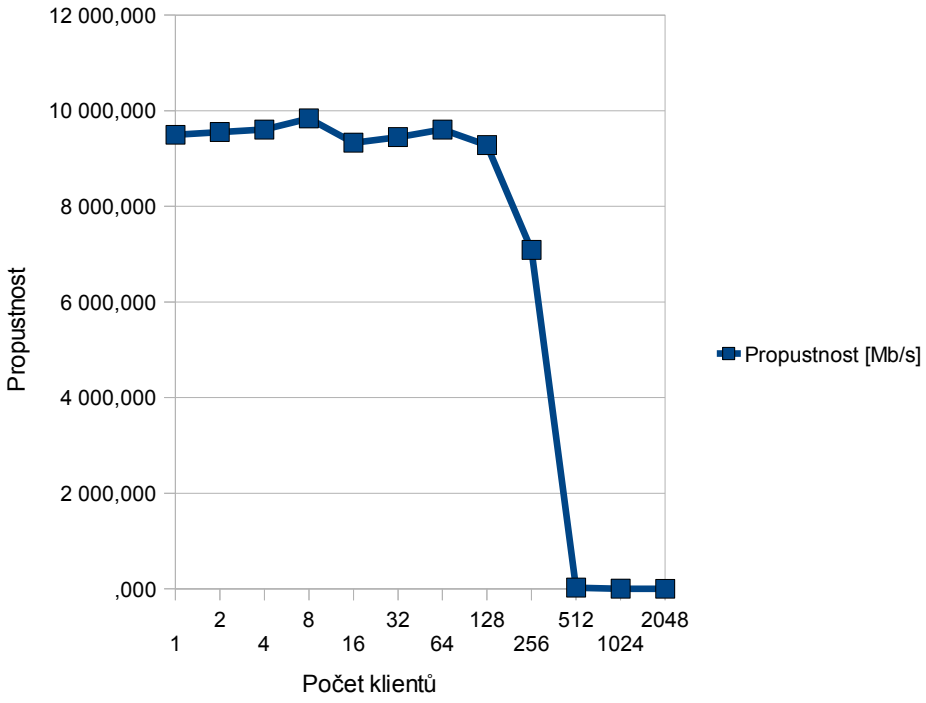
8.5.3) Testovaní

Testoval jsem stahování souboru: Střední soubor (popis na stránce 85. - kapitola 13. Příloha 3. - Soubory použité na testy) ze serveru Apache pomocí protokolu http pro proměnné množství klientů. MTU jsem u všech testů nechal na 9KB. Obor hodnot testovaných klientů byl [1 2 4 8 16 32 64 128 256 512 1024 2048]

8.5.4) Změřená data

Počet klientů	Propustnost [Mb/s]
1	9 496,303
2	9 553,309
4	9 606,055
8	9 837,956
16	9 332,766
32	9 449,969
64	9 606,918
128	9 280,549
256	7 090,493
512	25,229
1024	2,969
2048	1,398

Tabulka 22: Curl-loader - Změřená propustnost



Ilustrace 16: Curl-loader - Graf změřené propustnosti

8.5.5) Závěr

Při měření propustnosti jsem se dostal k problémům se serverovou stranou. Při překročení cca 200 klientů začala propustnost příkře klesat. Toto nebylo způsobeno nedostatkem výkonu ani na straně serveru, ani na straně klienta – zatížení během testů nepřesahovalo 10%. Zkoušel jsem měnit množství vláken, které má Apache k dispozici, také jsem vyzkoušel překompilovávat s jiným maximem klientů, ale nedočkal jsem se žádného zlepšení. Chybu jsem neodstranil ani ve spolupráci s kamarádem zabývajícím se webhostingem. Na klientu se mi podařilo vytvořit přes 100 tisíc klientů a limitujícím parametrem nebyl výkon procesoru, ale velikost paměti RAM. Nicméně vzhledem k tomu, že z druhé strany neodpovídal server, zatížení neodpovídalo plnému testování. Dovoluji si odhadnout, že na našem laboratorním HW můžeme tímto způsobem simulovat 60-70 tisíc klientů.

Z praxe mám vyzkoušeno, že provoz generovaný jedním postarším notebookem (Asus A6Jc -Intel Core Duo T2300 1.66GHz, paměť 512MB RAM), jenž zvládne simulovat 3 tisíce klientů, postačí k úplnému zahlcení tří velmi výkonných serverů a datový tok při tom nepřesáhne 20Mb. Pro servery se od určité hranice počtu klientů jeví provoz jako DOS/DDOS útok a pokud jsou bez ochran, nejen služby na nich běžící nepřežijí.

8.6. Ping

8.6.1) Popis

Ping je standardní systémový příkaz pro dostupnosti některého bodu v síti a měření latence případné odezvy, zkusil jsem ho tedy použít jako utilizátor linky a též měřič latencí.

8.6.2) Měření

Měřil jsem dobu potřebnou pro provedení 1000000 dvojic příkazů žádost a odpověď. MTU jsem nastavil na 9KB.

```
root@Obelix:~/na_noc# ping 192.168.0.1 -f -s 8972 -c 1000000
PING 192.168.0.1 (192.168.0.1) 8972(9000) bytes of data.

--- 192.168.0.1 ping statistics ---
1000000 packets transmitted, 1000000 received, 0% packet loss, time
171057ms
rtt min/avg/max/mdev = 0.065/0.132/0.360/0.049 ms, ipg/ewma 0.171/0.090 ms
```

8.6.3) Závěr

Pro měření jsem se pokusil použít i ping v režimu Flood (záplava), kdy je posílán ping bez čekání na odpověď, tento způsob není vhodný pro měření propustnosti, protože dat pro zaplnění 10G linky je generováno příliš málo.

Při zasílání ICMP paketu o velikosti 9KB se má za sekundu teoreticky přenést:
Počet možných přenesených paketů = rychlost linky [B] / velikost paketu [B]
10 000 000 000/8/(9000+12) = 138 703 paketů jedním směrem. Můj test posílal 1000000 paketů každým směrem, což by při teoretickém maximálním využití linky vyžadovalo 7.2 sekund, nicméně při měření se data se přenesla za 171 sekund – tedy využití linky 4-5%.

Tento test nicméně ukazuje zajímavé informace o kolísání zpoždění min/avg/max/ = 0.065/0.132/0.360 ms, pro které nemám vysvětlení.

Důležitá informace je, že během testu bylo jedno jádro na obou strojích plně utilizované a nepodařilo se mi mu žádným způsobem odlehčit. I přes tyto potíže se během testu neztratil ani jeden paket.

9. Závěr

9.1. Zhodnocení diplomové práce

V této diplomové práci jsem se zaměřil na testování OpenSource programů pro testování parametrů komplexních 10 Gigabitových sítí.

Základem práce je teoretická studie o měřených parametrech sítě a určení hlavních typů aplikací využívaných v sítích. Z této studie je určeno jaké základní testy a pro jaké protokoly se testy mají navrhovat.

Dále jsem se zabýval jednotlivými parametry ovlivňujícími provoz, kde jsem definoval maximální teoretickou propustnost na 4. vrstvě, vůči níž jsem prováděl srovnání propustnostních hodnot dosažených při testování jednotlivých programů.

Asi nejtěžší bylo vyladit systém pro plné využití 10 Gigabitových síťových karet, zde jsem otestoval velké množství nastavení, z nich pouze několik málo mělo skutečně pozitivní vliv na výkonnost, tyto změny jsou popsány v kapitole 7. Optimalizace operačního systému. Zde jsem také dělal největší chybu diplomové práce a to bylo použití zbytečně přesné testování. Po každé změně jsem totiž pro ověření jejího efektu prováděl testování pomocí pěti minutových testů pro 15 rozdílných hodnot MTU, což v součtu dává 75 minut potřebných pro ověření zlepšení výkonnosti. Což vzhledem k velkému množství potřebných testů velmi zpomalovalo práci. Z dodatečných testů doporučuji pro jakékoli budoucí testy využít jeden desetisekundový test pro každou hodnotu MTU pro určení vzrůstu výkonu. I při tomto snížení délky testu jsou naměřené výsledky naprosto srovnatelné s výsledky delších testů.

Z hlediska výsledků testovaných aplikací vyplynula nemožnost použít standardních aplikací pro testování parametrů. Za standardní aplikace nyní považuji standardní downloadery (Wget a Curl) spuštěným oproti serverů Apache a Proftpd. V tomto případě byla totiž utilizace linek příliš nízká. Oproti tomu při využití specializovaných programů se mi podařilo při měření propustnosti TCP streamů při použití IP verze 4 v rozmezí MTU 1250-9000 B utilizovat linku na více jak 99% teoretického maxima . Při použití UDP streamů a TCP/IP verze 6 jsem dokázal linku utilizovat na 99% teoretického maxima propustnosti až při vysokých MTU. Toto nebylo způsobeno nefunkčností generátoru, který pakety generoval plnou rychlostí, ale problémy s příjmem paketů na přijímací straně, které jsem nedokázal plně eliminovat.

Z testů na měření latence jsem uskutečnil měření výměny dat na TCP a UDP protokolu a dále měření latence provedení plné TCP transakce. Dále jsem testoval systém Curl-loader, který slouží jako simulátor až několika desítek tisíc klientů připojujících se ke službě.

Kromě popsání metodiky testování a určení správných parametrů nastavení systému jsou i všechny tyto testy připravené jako scripty ke spuštění pro použití v navazující práci Michaela Rohrbachera, který se bude zabývat porovnáním výkonu standardních síťových karet oproti síťovému akcelerátoru.

9.2. Navazující práce

Kromě již zmíněné navazující práce Michaela Rohrbachera zabývající se porovnáním výkonu standardních síťových karet oproti síťovému akcelerátoru, bude nutné lokalizovat jaké chyby způsobují problémy na Intel síťové kartě, která při zadání některých systémových příkazů přestává odpovídat, stejný problém se opakoval i v druhém stroji a ani aktualizace driveru nevedla k odstranění chyby.

Pro navazující práce také navrhuji otestovat některé testovací programy z kapitoly 14. - Příloha 4. - Seznam programů vhodných k dalšímu testování.

10. Seznam použité literatury

[1] S. Bradner and J. McQuaid, “Benchmarking Methodology for Network Interconnect Devices.” [Online]. Available: http://www.ietf.org/rfc/rfc2544.txt. [Accessed: 18-Dec-2011].

[2] Slash The Seat, “Interframe gap | Slash The Seats.” [Online]. Available: http://slashtheseats.com/rrpedia/Interframe_gap. [Accessed: 18-Dec-2011].

[3] “Observing routing asymmetry in Internet traffic.” [Online]. Available: http://www.caida.org/research/traffic-analysis/asymmetry/. [Accessed: 30-Dec-2011].

[4] “How do I troubleshoot slow Myri10GE or MX-10G performance? - Documentation.” [Online]. Available: http://www.myricom.com/kb/index.php?title=How_do_I_troubleshoot_slow_Myri10GE_or_MX-10G_performance%3F. [Accessed: 18-Dec-2011].

[5] “README - Myricom 10GbE driver for Linux.” [Online]. Available: http://www.myricom.com/scs/READMES/README.myri10ge-linux. [Accessed: 18-Dec-2011].

[6] “RFC 2018 - TCP Selective Acknowledgment Options (RFC2018).” [Online]. Available: http://www.faqs.org/rfcs/rfc2018.html. [Accessed: 20-Dec-2011].

[7] Citrix, “CTX130976 - Nagle’s Algorithm on a NetScaler Appliance - Citrix Knowledge Center.” [Online]. Available: http://support.citrix.com/article/CTX130976. [Accessed: 17-Dec-2011].

[8] J. M. S. Deering, “RFC 1191 - Path MTU Discovery.” [Online]. Available: http://www.ietf.org/rfc/rfc1191.txt. [Accessed: 28-Dec-2011].

[9] J. Mogul, “RFC1981 - Path MTU Discovery for IP version 6.” [Online]. Available: http://www.ietf.org/rfc/rfc1981.txt. [Accessed: 28-Dec-2011].

[10] J. H. M. Mathis, “RFC 4821 - Packetization Layer Path MTU Discovery.” [Online]. Available: http://www.ietf.org/rfc/rfc4821.txt. [Accessed: 28-Dec-2011].

[11] “The TIME-WAIT state in TCP and Its Effect on Busy Servers.” [Online]. Available: http://www.isi.edu/touch/pubs/infocomm99/infocomm99-web/. [Accessed: 20-Dec-2011].

[12] “[Groupe comp-protocols-tcp-ip] : time-wait recycling vs. reuse - Conversation.” [Online]. Available: http://www.frameip.com/nntp/comp-protocols-tcp-ip/21912-comp-protocols-tcp-ip-time-wait-recycling-vs-reuse.htm. [Accessed: 20-Dec-2011].

[13] Vivek Gite, “Linux Increasing The Transmit Queue Length (txqueuelen).” [Online]. Available: http://www.cyberciti.biz/faq/gentoo-centos-rhel-debian-fedora-increasing-txqueuelen/. [Accessed: 18-Dec-2011].

[14] “Kernel.org - Documentation - Networking - IP-Sysctl.” [Online]. Available: http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt. [Accessed: 18-Dec-2011].

[15] “Enabling High Performance Data Transfers [PSC].” [Online]. Available: http://www.psc.edu/networking/projects/tcptune/. [Accessed: 18-Dec-2011].

[16] “Linux Tuning.” [Online]. Available: http://fasterdata.es.net/fasterdata/host-tuning/linux/. [Accessed: 18-Dec-2011].

[17] “The Netperf Homepage.” [Online]. Available: http://www.netperf.org/netperf/. [Accessed: 18-Dec-2011].

[18] “Netperf: A Network Performance Benchmark.” Information Netwotks Division Hewlett-Packard Company.

[19] “IPERF - The Easy Tutorial.” [Online]. Available: http://openmaniak.com/iperf.php. [Accessed: 18-Dec-2011].

[20] “Seagull: an Open Source Multi-protocol traffic generator.” [Online]. Available: http://gull.sourceforge.net/. [Accessed: 28-Dec-2011].

11. Příloha 1. - Obsah přiloženého DVD

11.1. Výpis souborů

Umístění souborů s daty

```
./
|-- ing1.odt
|-- ing1.pdf
|-- manual
|   `-- netperf.ps
|-- mereni
|   |-- curl - apache
|   |   |-- data-cista
|   |   |-- data-raw
|   |   |-- mereni.ods
|   |   `-- script
|   |-- Curl-loader
|   |   |-- data
|   |   `-- mereni.ods
|   |-- curl - proftpd
|   |   |-- data-cista
|   |   |-- data-raw
|   |   |-- mereni.ods
|   |   `-- script
|   |-- Iperf
|   |   |-- data
|   |   |-- data.cista
|   |   |-- mereni.ods
|   |   `-- script
|   |-- netperf
|   |   |-- Connecting
|   |   |   |-- CRR.png
|   |   |   |-- data
|   |   |   `-- script
|   |   |-- ke zpracovani
|   |   |-- mtu.ods
|   |   |-- parametry
|   |   |   |-- data
|   |   |   |-- netperf-core-mtu-1.4.4.ods
|   |   |   `-- netperf-core-mtu-1.5.ods
|   |   |-- TCP_SENDFILE
|   |   |   |-- data
|   |   |   |-- mereni.ods
|   |   |   `-- script
|   |   |-- TCP_STREAM
|   |   |   |-- data
|   |   |   |-- IPv6
|   |   |   |   |-- ipv6
```

```
|   |   |   |   |-- ipv6.cista
|   |   |   |   `-- mereni.ods
|   |   |   |-- mereni.ods
|   |   |   |-- script
|   |   |   `-- TCP_NODELAY
|   |   |       |-- data
|   |   |       `-- mereni.ods
|   |   `-- UDP_STREAM
|   |       |-- data
|   |       |-- mereni.ods
|   |       `-- script
|   |-- wget-apache
|   |   |-- data-cd.raw
|   |   |-- data-cista
|   |   |-- data-netinstall.raw
|   |   |-- data.raw
|   |   |-- data-vmlinux.raw
|   |   |-- mereni.ods
|   |   `-- script
|   `-- wget - proftpd
|       |-- data
|       |-- data-cista
|       |-- data-ciste-stredni
|       |-- data-mala
|       |-- data-mala-cista
|       |-- data-mala2
|       |-- data-stredni
|       |-- data-stredni2
|       |-- grep
|       |-- mereni.ods
|       `-- script
|-- ostatni
|   `-- Kernel
|       |-- linux-headers-3.1.2-xsimi_3.1.2-xsimi-10.00.Custom_amd64.deb
|       `-- linux-image-3.1.2-xsimi_3.1.2-xsimi-10.00.Custom_amd64.deb
|-- script
|   `-- kompilujjadro.sh
|-- script-ping
|-- schema
|   |-- http:__www.isi.edu_touch_pubs_infocomm99_infocomm99-web_.gif
|   `-- pouzit
|       |-- testovací sítIPv4.graphml
|       |-- testovací sítIPv4.png
|       |-- testovací sít IPv6.graphml
|       `-- testovací sít IPv6.png
`-- teorie
    |-- mtu-teoreticka propustnost-ipv6.ods
    `-- mtu-teoreticka propustnost.ods
```

11.2. Popis přiložených souborů

11.2.1) ing1.odt

Soubor s textem Diplomové práce

11.2.2) ing1.pdf

Vygenerovaný PDF soubor s textem Diplomové práce

11.2.3) Manual

Složka obsahuje manuály k testovacím programům.

11.2.3.A netperf.ps

Manuál testovacího systému Netperf

11.2.4) Složka mereni

Obsahuje podsložky s názvy jednotlivých testovaných programů případně i jednotlivých testů.

Příklad podsložka „curl - proftpd „ obsahuje:

- script – soubor spouštějící jednotlivé testy
- data-raw – data z výstup ze scriptu
- data-cista – data vyčištěná o nadbytečné informace z měření
- mereni.ods – soubor s výsledky, výpočty a grafy

Pro budoucí měření pak stačí spustit pouze script a výsledky vložit do souboru mereni.ods, po čemž se přepočítají výsledné tabulky a překreslí grafy.

11.2.5) script/kompilujadro.sh

Je jednoduchý script, který automatizuje sestavení jádra verzí 3.x.x na systémech založených na distribuci Debian. Script připraví systém na kompilaci jádra, následně stáhne a

rozbalí zdrojové kódy vybraného jádra.

Vytvoří novou verzi konfigurací s využitím voleb z již běžícího jádra a spustí program pro úpravy konfigurací. V následujícím kroku jádro přeloží a pokud vše proběhne úspěšně, tak i nainstaluje.

Použití:

K sestavení jádra 3.1.5:

```
sh kompilujadro.sh 3.1.5
```

11.2.6) Složka schema

Obsahuje schémata použité v diplomové práci.

Konkrétně jde o http:__www.isi.edu_touch_pubs_infocomm99_infocomm99-web_.gif [11], obsahující schéma stavů TCP spojení a následně soubory se schématy sítě, jak zdrojové soubory pro program yEd Graph Editor, tak vygenerovanou grafiku:

- testovací sit IPv4.graphml
- testovací sit IPv4.png
- testovací sit IPv6.graphml
- testovací sit IPv6.png

11.2.7) Složka teorie

Obsahuje dokumenty s vypočítanou propustností a grafy pro TCP/UDP přenos na protokolu IP verze 4 a verze 6.

Soubory:

- mtu-teoreticka propustnost-ipv6.ods
- mtu-teoreticka propustnost.ods

11.2.8) Složka ostatni

Obsahuje instalační balíčky použitého jádra a headers soubory pro případné dokompilování ovladačů.

Soubory:

- linux-headers-3.1.2-xsimi_3.1.2-xsimi-10.00.Custom_amd64.deb
- linux-image-3.1.2-xsimi_3.1.2-xsimi-10.00.Custom_amd64.deb

12. Příloha 2. - Instalace testovacích programů

12.1. Instalace Curl-loader

Instalace pro Debian Squeeze 6.0 (i386/AMD64) spočívá v těchto krocích:

Stažení zdrojových kódů z adresy: http://sourceforge.net/projects/curl-loader/files/curl-loader-stable/ (zdrojové kódy v ./nastroje/curl/instalace/curl-loader-0.52.tar.gz)

```
cd /usr/src/
wget  http://sourceforge.net/projects/curl-loader/files/curl-loader-
stable/curl-loader-0.52/curl-loader-0.52.tar.gz/download
```

Jejich rozbalení a nastavení se do nově vzniklé složky

```
tar xvfz curl-loader-0.52.tar.gz
cd curl-loader-0.52
```

Spuštění kompilace s deaktivovanou možností debugování a maximální optimalizací (zrychluje program)

```
make optimize=1 debug=0
```

Nechání si vygenerovat instalační balíček a jeho následnou instalací – vytvořený balíček jsem přiložil do ./nastroje/curl/instalace/curl-loader_0.52-1_i386.deb

```
checkinstall
```

Ještě uděláme pár úprav na systému pro vyšší dosažitelné výkony (popsány v kapitole: 7.Optimalizace operačního systému)

```
ulimit -n 20000
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse
```

12.2. Instalace Netperf

```
apt-get install netperf
```

12.3. Instalace Iperf

```
apt-get install iperf
```

13. Příloha 3. - Soubory použité na testy

13.1. DVD instalační image Debian 6.0.3

Instalační DVD Debianu Squeeze 3. revize pro platformu AMD64.
Dostupný na:
http://cdimage.debian.org/debian-cd/6.0.3/amd64/iso-dvd/debian-6.0.3-amd64-DVD-1.iso
Velikost: 4.4GB
md5: 231ef6d946ac3864a61c36ab631f6405
sha512:83c0562d95dca3f266a5331cea8c0e0889b167b55f4bad816dcd7e9bad7aad6c698f60de9838dc4beb3485b60dafcf18b4008afc2d18c1ec40fb38705dbfc139

Použit pro testy s programem Netperf

13.2. Pro aplikační testy

Pro testování jsem zvolil soubory tří velikostí. S velikostí souboru jsem nemohl jít do nekonečna, protože ve chvíli, kdy by soubor nebyl poskytován z cache či ramdisku, už bych neměřil propustnost sítě, ale propustnost úložiště, které svou propustností nebylo schopné 10G síťovým kartám konkurovat. Největší soubor proto má 647 MB. Zbylé soubory mají odstupňované velikosti.

13.2.1) Velký soubor

Instalační CD Debianu Squeeze 3. revize pro platformu AMD64.
Dostupný na:
http://cdimage.debian.org/debian-cd/6.0.3/amd64/iso-cd/debian-6.0.3-amd64-CD-1.iso
Velikost: 647MB
md5: 924dd1350c8845d7320105247ce09b47
sha256: 62828fb75c820b13bbca57d9ca77d94255561782added2d979ca8472a30f699e

13.2.2) Střední soubor

Instalační CD Debianu Squeeze 3. revize pro platformu AMD64 - netinstall verze.
Dostupný na:
http://cdimage.debian.org/cdimage/daily-builds/daily/arch-latest/amd64/iso-cd/debian-testing-amd64-netinst.iso
Velikost: 203M
Md5: 44a033faf56068d1c8999d2c2ec79ddc

Sha256: f8882e6af93ddac1bdeb29be1ad7db24f11f066550c8f3848fa8ee38af076af6

13.2.3) Malý soubor

Vmlinux - Linux kernel x86 boot executable bzImage
Dostupný: po rozbalení balíčku jádra systému (přiloženo na CD)
Velikost: 2.6MB
sha256: ad4b9d9eee6e442796358c0e218b6b7a1901487da3574d08baa56084765dd8a9
md5sum: d18fce19e33c462afb60512a94d5bcee

14. Příloha 4. - Seznam programů vhodných k dalšímu testování

Zde jsou uvedené některé programy, které by bylo vhodné v příštích pracích vyzkoušet.

14.1. Seagull

Podobný programu Curl-loader, je to Open Source (GPL v2) generátor provozu. Určený primárně pro testování IMS (3GPP, TISPAN, CableLabs)
Podpora pro testování provozu:

- Diameter base aplikací dle RFC 3588 - IMS Cx, Dx, Ro, Rf, TLS
- TCAP ITU, ANSI libovolný protokol přes TCAP (Camel, GSM MAP, IS41, Win, …) i přes SS7 (E1/T1) či SIGTRAN
- XCAP přes HTTP
- HTTP
- H248/Megaco ASCII form (UDP, TCP, SCTP)
- Radius

[20]

Domovská stránka: http://gull.sourceforge.net/

14.2. Tcpreplay

Tcpreplay se skládá z několika GPLv3 nástrojů umožňující zachytit a přehrát provoz..

Tcpreplay se skládá z těchto nástrojů:

- **tcprewrite** - editor pcap souborů, umožňuje přepsat L2-L4 hlavičky
- **tcpreplay** – přehraje zaznamenaný soubor do sítě
- **tcpreplay-edit** – sloučený tcpreplay a tcprewrite
- **tcpbridge** – slouží pro vytvoření bridge mezi dvěma segmenty síťě s využitím tcprewrite
- **tcpcapinfo** – decoder a debuger raw pcap souborů
- **tcpprep** – slouží k předzpracování dat pro tcpreplay and tcprewrite

Domovská stránka: http://tcpreplay.synfin.net/

14.3. Multi mechanize

Testovací rozhraní, které umožňuje spustit Python script simulující virtuálního klienta. Z výsledků scriptů jsou poté vygenerovány reporty.

Domovská stránka: http://code.google.com/p/multi-mechanize/

14.4. Ostinato

Ostinato je open-source, generátor a analyzátor trafiku. Na stránkách je označen jako "Wireshark in Reverse".

Základní vlastnosti:

- Otevření, editace, přehrátí a uložení PCAP souborů
- Ethernet/802.3/LLC SNAP
- Podpora VLAN
- ARP, IPv4, IPv6, IP-in-IP a.k.a IP Tunelování (6over4, 4over6, 4over4, 6over6)
- TCP, UDP, ICMPv4, ICMPv6, IGMP, MLD
- všechny protokoly založené na textu (HTTP, SIP, RTSP, NNTP ...)

Domovská stránka: http://code.google.com/p/ostinato/

14.5. Další programy již jen výčtem:

- Bwping
- Faban [http://faban.sunsource.net/]
- FunkLoad
- Hping
- Httping
- Jmeter.apache.org [http://jmeter.apache.org]
- Netcps
- Nnetperfmeter
- Netpipe
- Netsend
- Netsniff-ng
- Nuttcp
- Pktgen
- SIPp [http://sipp.sourceforge.net/]
- Ttcp,

15. Příloha 5. - Seznam použitých zkratek

CRC - Cyclic redundancy check
DDOS - Distributed Denial of Service
DMA - Direct Memory Access
DOS - Denial of Service
FTP - File Transfer Protocol
HTTP - Hypertext Transfer Protocol
ICMP - Internet Control Message Protocol
IPv4 - IP protokol verze 4
IPv6 - IP protokol verze 6
L2 - Druhá vrstva OSI modelu
MSS - Maximum segment size
MTU - Maximum Transmission Unit
OSI - Open Systems Interconnection
SFD - Start of Frame delimiter
SSL - Secure Sockets Layer
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
URL - Uniform Resource Locator