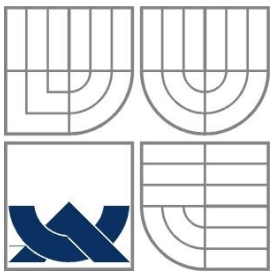
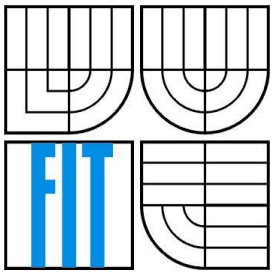


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FILTROVÁNÍ PAKETŮ V POČÍTAČOVÝCH SÍTÍCH

PACKET FILTERING IN COMPUTER NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Petr Šrůtka
AUTHOR

VEDOUCÍ PRÁCE — Ing. Michal Kajan
SUPERVISOR

BRNO 2013

Abstrakt

Tato bakalářské práce představuje klasifikaci paketů, různé její způsoby a metriky. Jsou zde popsány různé algoritmy, pomocí kterých je klasifikace paketů realizována. V práci jsou uvedeny vlastnosti algoritmu Recursive Flow Classification společně s popisem jeho implementace. Jsou zde představeny různé konfigurace algoritmu RFC a popsány jejich výhody a nevýhody. Na závěr práce jsou provedeny experimenty porovnávající jednotlivé algoritmy popsané v rámci této práce.

Abstract

This bachelor thesis contains an introduction to packet classification, types of packet classification techniques and different metrics. It describes different algorithms used for packet classification and implementation of Recursive flow classification algorithm is also part of this bachelors thesis. It presents different configuration parameters of RFC algorithm and describes its advantages and disadvantages. This work is concluded with a set of experiments describing characteristics of implemented algorithm together with comparison of other classification approaches.

Klíčová slova

klasifikace paketů, filtrování paketů, RFC, Recursive Flow Classification, porovnání

Keywords

packet classification, packet filtering, RFC, Recursive Flow Classification, comparison

Citace

Petr Šrůtka: Filtrování paketů v počítačových sítích, bakalářská práce, Brno, FIT VUT v Brně, 2013

Filtrování paketů v počítačových sítích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Kajana.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Šrůtka
15.5.2013

Poděkování

Chtěl bych poděkovat vedoucímu této práce, Ing. Michalu Kajanovi, za podporu a rady při řešení této práce.

Obsah

Obsah 1
1 Úvod 3
2 Počítačové sítě 4
2.1 Model ISO/OSI 4
2.1.1 Fyzická vrstva 5
2.1.2 Linková vrstva 5
2.1.3 Síťová vrstva 5
2.1.4 Transportní vrstva 5
2.1.5 Relační vrstva 6
2.1.6 Prezentační vrstva 6
2.1.7 Aplikační vrstva 6
2.2 TCP/IP 6
2.2.1 Vrstva síťového rozhraní 6
2.2.2 Síťová vrstva 7
2.2.3 Transportní vrstva 7
2.2.4 Aplikační vrstva 7
3 Klasifikace paketů 8
3.1 Způsoby klasifikace 8
3.2 Metriky klasifikačních algoritmů 9
4 Algoritmy pro klasifikaci paketů 11
4.1 Algoritmus Hierarchical Intelligent Cuttings 11
4.2 Algoritmus HyperCuts 13
4.3 Algoritmus Bit-Vector 14
4.4 Algoritmus Distributed crossproducting of field labels 15
4.5 Algoritmus Multi-subset crossproduct 16
5 Algoritmus Recursive Flow Classification 19
5.1 Výkon algoritmu Recursive Flow Classification 21
5.1.1 Fáze předzpracování 21
5.1.2 Klasifikace paketů 23
6 Implementace 25
6.1 Knihovna Netbench 25
6.2 Implementace algoritmu Recursive Flow Classification 26
7 Experimenty 29
7.1 Předzpracování 29

7.2 Paměť 29
7.3 Rychlost klasifikace paketů 30
8 Závěr 32

1 Úvod

Tato bakalářská práce popisuje problém klasifikace paketů. S velikým rozmachem internetu a počítačových sítí obecně se právě klasifikace paketů stává čím dál důležitější. Nejběžnějším použitím klasifikace paketů jsou směrovače, firewally a oblast zajištění kvality služeb. Zatímco směrovače potřebují pakety klasifikovat pouze podle jedné položky z hlavičky paketu u firewallů může být toto číslo mnohem vyšší. Protože nechceme klasifikací brzdit provoz na síti, je potřeba stále zvyšovat její rychlost ať už optimalizací algoritmu nebo vylepšením hardwaru.

V druhé kapitole se podíváme obecně na rozvoj sítí a vznik internetu. Jsou tam popsány vrstvové modely ISO/OSI a TCP/IP, díky kterým se sítě vyvinuly do dnešní podoby, a jejich jednotlivé vrstvy.

Ve třetí kapitole je obecně popsána klasifikace paketů, včetně jejich způsobů a metrik používaných k hodnocení algoritmů, které klasifikaci zajišťují. Kapitola obsahuje popis jednotlivých fází fungování klasifikačních algoritmů a jejich průběh.

Čtvrtá kapitola je zaměřená na algoritmy samotné. Pro popis v této práci bylo vybráno pět algoritmů, které již byly implementovány v experimentální knihovně Netbench. Je zde jednoduše popsáno a na obrázcích znázorněno jejich fungování.

Pátá kapitola obsahuje detailní popis algoritmu Recursive Flow Classification (RFC), který bude v rámci této práce implementován, pozorování, která vedla k jeho vzniku, hlavní myšlenku algoritmu a jeho možné konfigurace. Dále jsou v této implementaci srovnány implementace dvou různých konfigurací algoritmu. Srovnání je založeno na základě rychlosti fáze předzpracování, využití paměti a rychlosti klasifikace.

Šestá kapitola se zabývá samotnou implementací algoritmu RFC. Je zde popsána knihovna Netbench, ze které byly při implementaci využity některé funkcionality. V této kapitole jsou popsány nutné změny, které byly potřebné udělat v třídách knihovny Netbench pro správné fungování algoritmu, ale i samotné třídy vzniklé pro správné fungování algoritmu a testy.

V poslední kapitole budou provedeny experimenty nad algoritmy z knihovny Netbench a algoritmem RFC. Bude změřena a porovnána rychlost fáze předzpracování, paměťové nároky a rychlost klasifikace paketů jednotlivých algoritmů.

2 Počítačové sítě

Na konci šedesátých let minulého století, pod vedením společnosti ARPA (Advanced Reasearch Projects Agency), začala vznikat první velká síť. Tato síť vznikala v období studené války, aby umožnila americkým důstojníkům zůstat v kontaktu i v případě atomového výbuchu. Síť, nazývaná ARPAnet, vznikla na pěti základních požadavcích [1].

- Žádný z uzlů této sítě nesmí být kritičtější než jiný. Vzhledem k tomu, že síť vznikla za hrozby války, bylo nežádoucí, aby síť obsahovala jedno kritické místo, vyřazením kterého by byla vyřazena celá síť.
- Více cest k jakémukoliv uzlu - komunikace mezi dvěma uzly nikdy nesměla být omezena pouze jednou cestou. Tento požadavek úzce souvisí s předchozím požadavkem. Bez nadbytečných cest by se jakýkoliv uzel mohl stát kritickým a jeho vyřazením by byla omezena nebo zcela ukončena funkčnost sítě.
- Přesměrování dat za provozu v případě výpadku kterékoliv části sítě, musí síť dokázat přesměrovat data na jednu z alternativních cest během velmi krátkého času.
- Schopnost připojit se k různým druhům počítačů na různých typech sítí - bylo nežádoucí, aby síť byla svázána s pouze jedním operačním systémem a byla svázána s nákupem nového hardwaru.
- Nesmí být ovládána jedinou korporací. Cílem této sítě bylo spojit úsilí mnoha pracovníků tak, aby pracovali v zájmu sítě samotné a ne pro prospěch nějaké korporace.

Pro účely této sítě byl vytvořen protokol NCP, který ale s rostoucí velikostí sítě pomalu přestával plnit své původní účely. Postupně byl tento protokol nahrazen sadou protokolů TCP/IP. ARPAnet se postupným rozšiřováním rozvinul v celosvětovou síť nazývanou Internet.

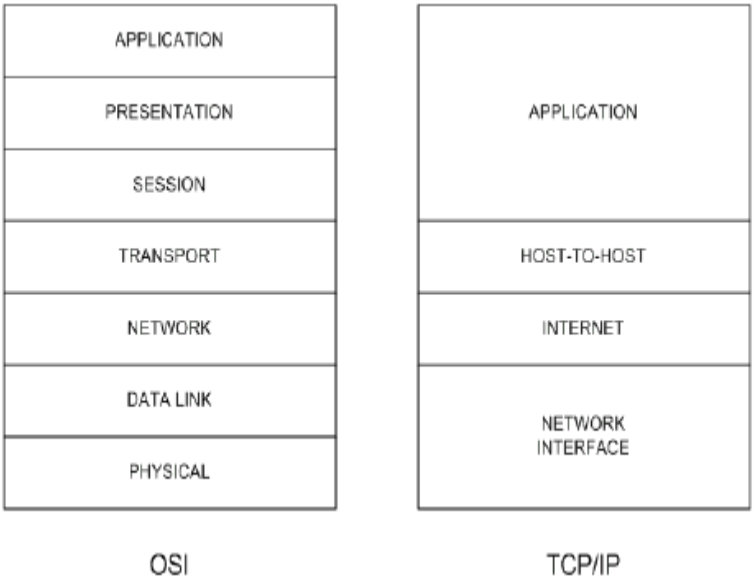
Od té doby sítě prošly mnoha změnami a byla ustanovena spousta standardů. V druhé polovině osmdesátých let dvacátého století vznikl model ISO/OSI definovaný mezinárodní organizací OSI. Dalším, v praxi mnohem více využívaným modelem, je model TCP/IP. Jejich porovnání můžete vidět na obr. 1.

2.1 Model ISO/OSI

Model ISO/OSI [2] vznikl v roce 1984. Jedná se o referenční model pro popis síťové architektury. V praxi tento model nikdy nebyl, kvůli své komplexnosti, celý implementován. Je rozdělen do sedmi vrstev a to fyzické, linkové, síťové, transportní, relační, prezentační a aplikační. Každá vrstva zajišťuje jinou část komunikace mezi dvěma počítači.

2.1.1 Fyzická vrstva

Tato vrstva definuje mechanické a funkční vlastnosti, jako tvar konektorů, použité přenosové médium, přenos bitů po vedení a elektrické specifikace. Stará se o navázání a ukončení spojení, převod bitů na signály a rozložení zátěže mezi uživatele. Na této vrstvě, jako jediné, dochází k přenosu bitů komunikačním kanálem. Vyšším vrstvám nabízí způsob odeslání a přijímání bitů. Mezi protokoly řazené do této vrstvy patří bluetooth, USB, IEEE 802.11, IEEE 802.15, DSL.



Obr. 1 Porovnání modelů ISO/OSI a TCP/IP, převzato z [10]

2.1.2 Linková vrstva

Linková vrstva popisuje přenos dat po datových linkách. Přeposílá datové rámce mezi síťovou a fyzickou vrstvou. Při přenosu z fyzické vrstvy uspořádává data do logických celků, datových rámců. Dokáže odhalit, případně i opravit, bitové chyby. Pokud chybu nedokáže opravit, odesílá vyšším vrstvám oznámení o odhalení neopravitelné chyby. Tato vrstva zahrnuje protokoly a standardy ATM, PPP, Ethernet, HDLC, SDLC.

2.1.3 Síťová vrstva

Síťová vrstva zajišťuje přenos dat s proměnnou délkou mezi dvěma uzly. Stará se o adresování a směrování v síti. Na této vrstvě fungují směrovače. Nejznámějším protokolem fungujícím na této vrstvě je IP.

2.1.4 Transportní vrstva

Transportní vrstva zajišťuje transparentní přenos dat mezi uživateli. V případě odesílání dat zajišťuje transportní vrstva jejich rozložení na jednotlivé pakety. Při příjmu dat naopak zajišťuje složení všech paketů. Mezi nejznámější protokoly patřící do této vrstvy se řadí protokoly TCP, který zajišťuje

spolehlivý přenos dat. Dalším je protokol UDP, který je mnohem rychlejší než TCP, ale nezajistí doručení odeslaného paketu.

2.1.5 Relační vrstva

Relační vrstva vytváří, řídí výměnu dat a ukončuje spojení mezi dvěma uzly. Umožňuje také synchronizaci, obnovení spojení a oznamování výjimečných stavů. Poskytuje tři druhy spojení: tzv. full-duplex (současná komunikace oběma směry), half-duplex (postupná komunikace oběma směry) a simplex (komunikace pouze jedním směrem).

2.1.6 Prezentační vrstva

Hlavní funkcí prezentační vrstvy je převod dat do formátu, kterému rozumí aplikace. Musí zajistit správné pořadí bitů, použití správného kódování znaků a korektní reprezentaci času. Dále zabezpečuje šifrování a kompresi dat. Zabývá se strukturou dat, ale nezná jejich význam.

2.1.7 Aplikační vrstva

Aplikační vrstva je nejvyšší vrstvou modelu ISO/OSI. Tato vrstva komunikuje přímo s aplikací. Zahrnuje například protokoly FTP, DNS, DHCP, SMTP a SSH.

2.2 TCP/IP

TCP/IP [1] je sadou protokolů umožňujících komunikaci mezi počítači. Architektura této sady protokolů je mnohem jednodušší než architektura ISO/OSI, díky čemu se, na rozdíl od ISO/OSI, stala po světě daleko více využívanou. Po celém světě se používá už více než třicet let a stala se hlavní sadou protokolů využívanou na internetu. TCP/IP je podporováno všemi hlavními operačními systémy používanými v dnešní době. TCP/IP se skládá ze čtyř vrstev, které se dají do modelu ISO/OSI volně přemapovat podle obrázku na začátku kapitoly (obr. 1). Obdobně jako v modelu ISO/OSI má i zde každá vrstva na práci jinou část přenosu dat a navzájem mezi sebou komunikují. Nejníže položenou vrstvou je vrstva síťového rozhraní a nad ní následují síťová, transportní a aplikační vrstva.

2.2.1 Vrstva síťového rozhraní

Vrstva síťového rozhraní definuje, jak je počítač připojen do sítě. Funkci této vrstvy zajišťuje síťová karta. Při odesílání dat balí datagramy do rámců. V hlavičce každého rámce je obsažena zdrojová a cílová MAC adresa. MAC adresa je 48-bitové číslo, zpravidla zapsané v šestnáctkové soustavě. První polovina MAC adresy je unikátní pro výrobce síťových karet a zbytek je unikátní pro všechny síťové karty od daného výrobce. Ve výsledku by žádné dvě síťové karty neměly mít stejnou MAC

adresu. Konkrétní MAC adresa může vypadat takto: 3C:97:0E:3F:CD:F7. Aby tato vrstva předala paket dále, musí být v hlavičce obsažena cílová MAC adresa, které odpovídá MAC adresa síťové karty, která paket přijala. Jedinou výjimkou jsou pakety všesměrového vysílání (tzv. broadcast), které obsahují MAC adresu FF:FF:FF:FF:FF:FF a jsou vždy předány vyšším vrstvám.

MAC adresa se v algoritmech pro klasifikaci paketu používá pouze výjimečně.

2.2.2 Síťová vrstva

Hlavní funkcí síťové vrstvy je adresování. Nejčastějším protokolem fungujícím na této vrstvě je protokol IP. Tento protokol pro adresaci používá IP adresu. Ve verzi 4 je IP adresa 32 bitové číslo a ve verzi 6 je její délka 128 bitů. IP adresa je počítači přiřazena automaticky systémem DHCP po připojení do sítě, nebo manuálně administrátorem. Dalšími protokoly fungujícími na této vrstvě jsou ARP, ICMP a IGMP.

Síťová vrstva je z hlediska klasifikace paketů daleko zajímavější než vrstva síťového rozhraní. Zpravidla se na této vrstvě ke klasifikaci paketů používá zdrojová a cílová IP adresa a použitý protokol.

2.2.3 Transportní vrstva

Transportní vrstva zajišťuje spojení mezi dvěma počítači a určuje, jak často si budou navzájem zasílat potvrzení o spojení. Také zajišťuje fragmentaci odesílaných dat, chybovou kontrolu, kontrolu toku, kontrolu zahlcení sítě a adresování jednotlivých aplikací v rámci počítače za pomocí čísla portu. Transportní vrstva obsahuje pouze dva protokoly, TCP a UDP. Hlavní rozdíl mezi těmito dvěma protokoly byl popsán na konci kapitoly 2.1.4.

Pro klasifikaci paketů jsou na této vrstvě zajímavé zdrojové a cílové číslo portu. Číslo portu je 16bitové číslo v rozsahu 0 až 65535, které unikátně identifikuje aplikaci běžící v rámci jednoho počítače. Čísla portů se dále dělí na rezervovaná (0-1023), registrovaná (1024-49151) a dynamická (49152-65535). Toto dělení zajišťuje snadnou dostupnost standardních služeb (například DNS serveru je přiřazen port 53).

2.2.4 Aplikační vrstva

Nejvyšší vrstvou je vrstva aplikační. Tato vrstva je složena z aplikací, které voláním nižších vrstev komunikují s jinými aplikacemi po síti. Aplikace na této vrstvě se dají rozdělit do dvou skupin: uživatelské a systémové. Uživatelské aplikace jsou využívané přímo uživatelem a řadí se sem například Telnet, IRC a SSH. Systémové aplikace komunikují s jinými aplikacemi a rozšiřují tak jejich funkčnost. Mezi systémové aplikace patří například DNS a SNMP.

Tato vrstva je u klasifikace paketů zajímavá pouze u firewallu typu aplikační brána, který se dnes, kvůli své pomalosti a náročnosti, jen ve velmi specializovaných případech [14].

3 Klasifikace paketů

Klasifikace paketů [3][4] je jednou ze základních činností většiny síťových zařízení. Jedná se o algoritmický problém, který řeší rozpoznání příchozích paketů pomocí definované množiny pravidel. Využívá se například při filtrování paketů ve firewallech, pro adresování paketů ve směrovačích nebo pro zajištění kvality služeb. S rostoucí rychlostí počítačových sítí je třeba optimalizovat algoritmy pro klasifikaci paketů, aby vyhledávaly co nejrychleji a zároveň jejich datové struktury zabíraly co nejméně místa v paměti zařízení. Tyto dva požadavky bohužel ve většině algoritmů kolidují, takže je pro optimální výkon třeba do každého zařízení zvolit algoritmus, podle jeho očekávaného zatížení a velikosti paměti.

Klasifikace se v zásadě může dělit na dvě části, fázi předzpracování pravidel a fázi klasifikace paketů. Ve fázi předzpracování dochází k vytvoření datových struktur, které jsou později využívány pro klasifikaci paketů. Tato fáze neprobíhá velmi často, při každé změně sady pravidel, a obvykle tedy není důležité, aby probíhala co nejrychleji. Ovšem může nastat situace, kdy je potřeba co nejrychleji změnit pravidla obsažená v předpočítaných strukturách, a proto je nutné dobu předzpracování držet v rozumných mezích. Ve fázi klasifikace dochází k samotnému přiřazení paketu k pravidlu. Při této fázi dochází ke zpracování dat, která prochází zařízením. Zde už je velmi důležité, aby vyhledávání v datových strukturách, vytvořených v předchozí fázi, probíhalo co nejrychleji a provoz v síti nebyl klasifikací paketů brzděn.

3.1 Způsoby klasifikace

Pro klasifikaci paketů se používají následující způsoby porovnání jednotlivých položek paketů. [4]

- Přesné porovnání - u přesného porovnání musí položka z paketu přesně odpovídat položce z pravidla.
- Porovnání na shodu prefixu - při porovnávání na shodu prefixu musí být položky pravidla prefixem položky z paketu. V případě, že položka z paketu odpovídá v sadě pravidel více prefixům, zpravidla se používá ten, kde došlo k nejdelší prefixové shodě.
- Intervalové porovnání - u intervalového porovnání musí položka z paketu náležet do intervalu definovaného v pravidle.
- Porovnání regulárního výrazu - porovnávání pomocí regulárního výrazu se používá k analýze obsahu paketu. Je velmi náročné na zpracování.
- Wildcard znak - speciálním případem je takzvaný wildcard znak, který se interpretuje jako „any“. Odpovídá tedy jakékoliv hodnotě dané položky a je možné ho použít u všech výše uvedených druhů porovnání. Jako wildcard znak se v tabulkách pravidel používá znak *.

Dále by se klasifikace dala rozdělit podle počtu porovnávaných položek z hlavičky paketu na jednodimenzionální a multidimenzionální.

- Jednodimenzionální - u jednodimenzionálního porovnání dochází k porovnání pouze jedné položky z paketu. Tento druh porovnávání se využívá například při směrování, kdy se porovnává pouze cílová IP adresa.
- Multidimenzionální - při multidimenzionálním porovnávání se porovnává *n* položek. Ve firewallech se nejčastěji používá standardní pětice složená ze zdrojové IP adresy, cílové IP adresy, zdrojového portu, cílového portu a protokolu. Klasifikace ale může probíhat nad prakticky jakýmkoliv polem z libovolné hlavičky nebo i samotného těla paketu.

3.2 Metriky klasifikačních algoritmů

Pro hodnocení klasifikačních algoritmů se používají následující metriky [3][4].

- Rychlost - algoritmus musí být schopen zpracovávat neustálý proud paketů. Na 10 Gb/s lince to, v případě 40 bytů velikých paketů a plného vytížení linky, tvoří přibližně 31 miliónů paketů za vteřinu. V případě přenosu po síti typu ethernet je i při maximálním využití linky přenášena zhruba polovina paketů,- z důvodu režie zajišťující správný přenos po síti.
- Paměťová náročnost - je žádoucí, aby paměťová náročnost algoritmu byla co nejmenší. Při malé paměťové náročnosti můžeme pro hardwarovou implementaci algoritmu, kde čip zpravidla nemá k dispozici veliké množství paměti, využít rychlejší, ale dražší paměť typu SRAM místo levnějších pamětí.
- Složitost implementace - složitost implementace má přímý vliv na to, jak náročné bude implementovat algoritmus v hardwaru. Obecně použití specializovaného hardwaru dosahuje daleko rychlejšího vykonání algoritmu než použití naprogramovaného mikročipu. Protože rychlost klasifikačního algoritmu je jedním z klíčových aspektů, snaha o jeho implementaci v hardwaru je logická.
- Rychlost předzpracování - důležitost této metriky, na rozdíl od metrik výše uvedených, je velmi závislá na druhu použití. Například pro firewally, kde se tabulky s pravidly nemění tak často, není velice důležitá a tak se vyplatí vybrat algoritmus s pomalejší rychlostí předzpracování a vyšší rychlostí. Na druhou stranu pro směrování je možné, že se ve směrovacích tabulkách budou měnit stovky prefixů během vteřiny a je tedy vhodné zvolit algoritmus, který bude zvládat fázi předzpracování dostatečně rychle. Směrování je ale speciální případ, kdy dochází ke klasifikaci pouze nad jednou dimenzí (obvykle je to cílová IP adresa) a fáze předzpracování je tedy mnohem rychlejší než u firewallu.

- Rozšiřitelnost - rozšiřitelnost udává, zda je možné pro algoritmus zvolit počet dimenzí. V ideálním případě by algoritmus měl povolovat zvolit si libovolný počet dimenzí pro porovnání a to z hlavičky libovolné vrstvy.

4 Algoritmy pro klasifikaci paketů

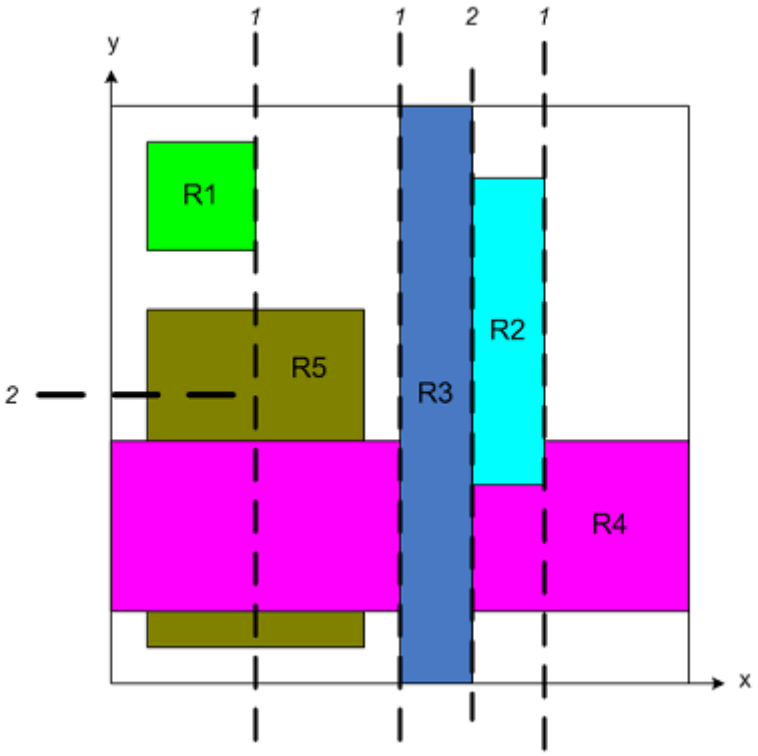
Pro klasifikaci paketů se používá více algoritmů, které se výrazně liší v metrikách uvedených v minulé kapitole, ale i stylem klasifikace. Pro každé konkrétní využití klasifikace paketů je třeba se zamyslet, který z existujících algoritmů je vhodný. Špatně zvoleným algoritmem může velmi vzrůst cena systému nebo se znatelně zpomalit průchod sítí. Pro popis v této kapitole byly záměrně vybrány algoritmy implementované v knihovně Netbench [12], vyvíjené výzkumnou skupinou ANT@FIT, působící na Fakultě informačních technologií na Vysokém učení technickém v Brně, kam by měl v jedné z příštích verzí přibýt i algoritmus Recursive Flow Classification implementovaný v rámci této bakalářské práce.

4.1 Algoritmus Hierarchical Intelligent Cuttings

HiCuts [7][11] je algoritmus založený na použití rozhodovacích stromů (tzv. decision tree based algoritmy). HiCuts algoritmus ve fázi předzpracování vytvoří rozhodovací strom, který při každém příchozím paketu projde až k listu. List obsahuje oproti celkové sadě pravidel podstatně menší počet pravidel, které jsou lineárně projity a z nich se vybere pravidlo, které nejlépe vyhovuje příchozímu paketu. Použitím kombinace stromového a lineárního prohledávání se tento algoritmus snaží dosáhnout co nejvýhodnějšího poměru mezi rychlostí algoritmu a velikostí paměti potřebné ke správnému fungování.

Strom využívaný pro klasifikaci paketů algoritmem HiCuts se vytváří rozřezáním stavového prostoru ve fázi předzpracování. Výkonnost tohoto algoritmu je velmi závislá na správném počtu řezů. Veliký počet řezů, sníží velikost stromu, čímž se zvýší rychlost prohledávání, ale také se tím zvýší potřebná paměť k uložení stromu. Celá fáze předzpracování probíhá ve čtyřech částech, které jsou zpracovány heuristickými algoritmy. Příklad vstupního stavového prostoru je na obr. 2.

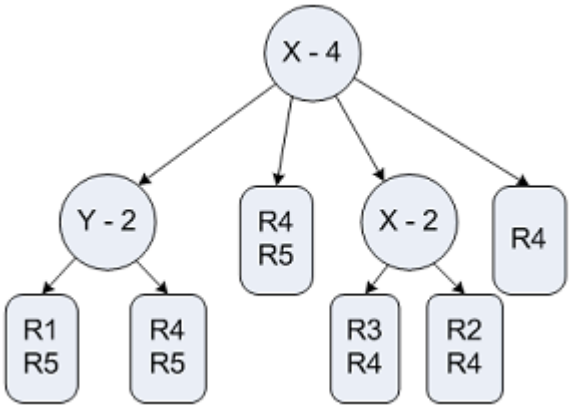
1. V první části heuristický algoritmus rozhodne, kolik řezů se má provést, aby výsledný strom byl optimálně veliký.
2. Dále algoritmus rozhoduje, jakými dimenzemi řez povede v každém vnitřním uzlu. V této fázi jsou možné čtyři přístupy k tomuto problému. Můžeme si vybrat, jestli každým následujícím řezem chceme co nejvíce snížit počet potomků daného uzlu a tím snížit nejvyšší počet úrovní stromu, jestli chceme ve výsledku mít co nejrovnoměrnější strom, jestli chceme co nejnižší součet pravidel všech potomků uzlů s aktuálním počtem řezů anebo můžeme vybrat dimenzi s největším počtem pravidel.



Obr. 2 Dělení stavového prostoru algoritmem HiCuts, převzato z [11].

3. V praxi bylo vypozorováno, že spousta potomků obsahuje stejnou sadu pravidel. Dalším logickým krokem tedy je, tato pravidla sloučit a ušetřit tak na paměťové náročnosti algoritmu.
4. Posledním krokem fáze předzpracování je odstranění nadbytečných uzlů. Při provádění řezu uzlem se může stát, že jedno nebo více pravidel v tomto řezu může být překryto jiným pravidlem s vyšší prioritou. V tomto případě je možné opět ušetřit paměť potřebnou k uchování výsledného stromu odstraněním těchto pravidel.

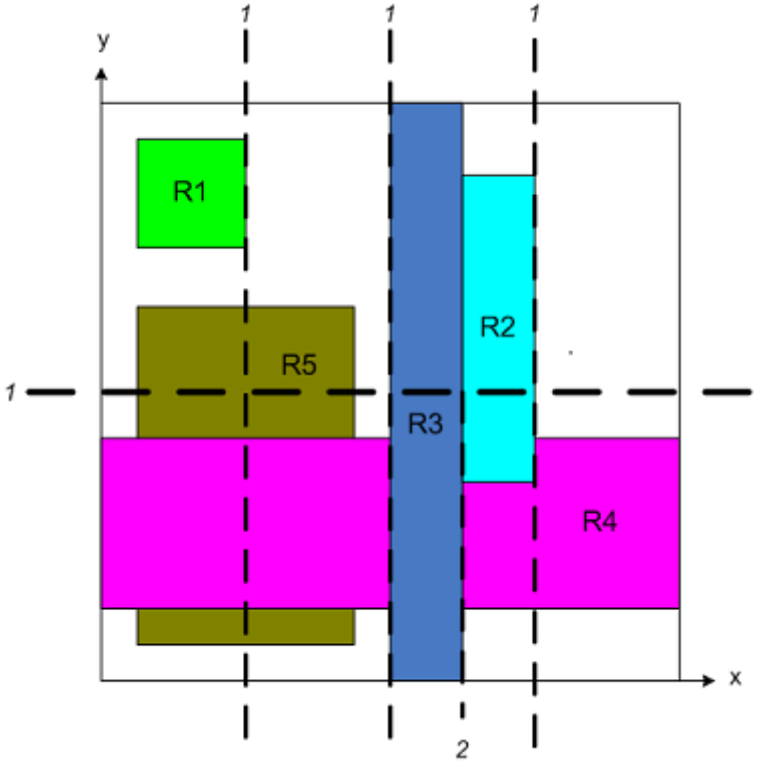
Na obr. 3 je zobrazen strom vzniklý ze stavového prostoru zobrazeného na obr. 2. Tento strom je ovšem pouze možným zobrazením, konkrétní zobrazení je závislé na zvolených metodách ve fázi předzpracování.



Obr. 3 Výsledný strom algoritmu HiCuts, převzato z [11].

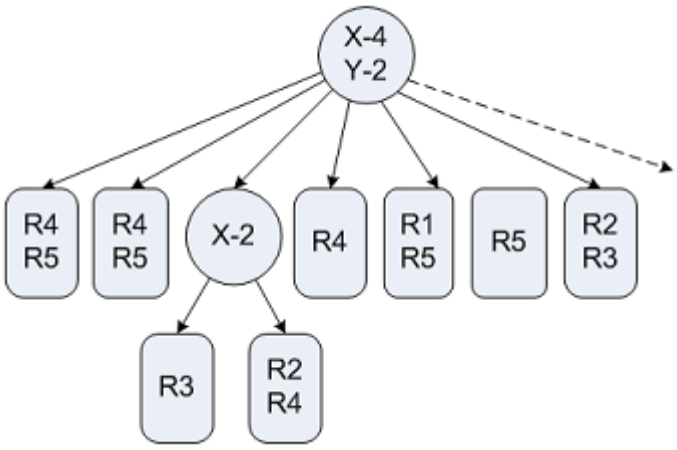
4.2 Algoritmus HyperCuts

Algoritmus HyperCuts [9][11] je velice podobný algoritmu HiCuts. Novou částí implementovanou v rámci algoritmu HyperCuts jsou řezy v rámci více dimenzí. Stejný stavový prostor, použitý v algoritmu HiCuts (obr. 2), by tedy po rozřezání algoritmem HyperCuts mohl vypadat takto (obr. 4).



Obr. 4 Dělení stavového prostoru algoritmem HyperCuts, převzato z [11].

A jeho následné zpracování do stromu je zobrazené na obr. 5.



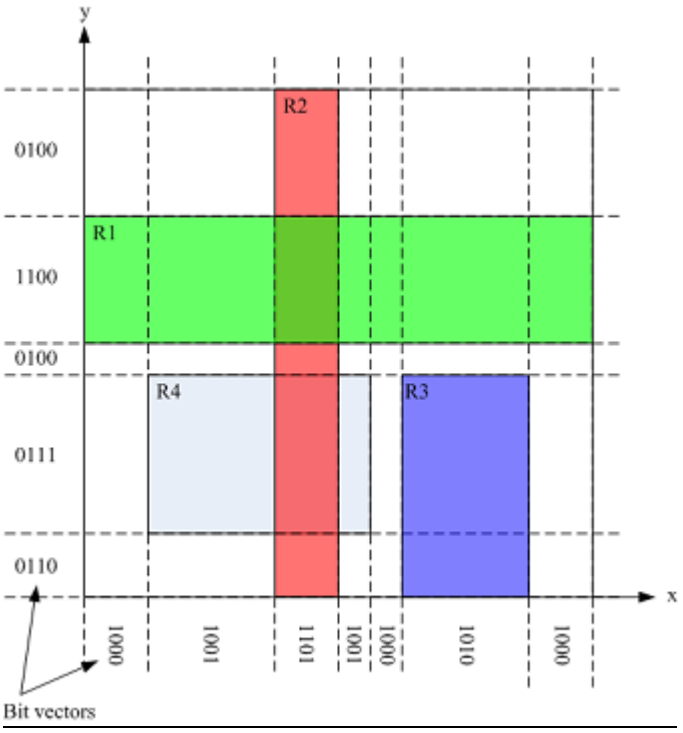
Obr. 5 Výsledný strom algoritmu HyperCuts, převzato z [11].

4.3 Algoritmus Bit-Vector

Bitový vektor [7][11] je algoritmus založený na bitovém paralelismu, který patří do třídy algoritmů využívající dekompozici. Tento algoritmus obsahuje fázi předzpracování a jeho využití je tedy vhodné na místech, kde se sada pravidel nemění příliš často. Fáze předzpracování je složená ze dvou kroků.

1. Pro každou dimenzi se vytvoří množina intervalů z položek všech pravidel dané dimenze. Těchto intervalů může být maximálně $2n+1$, kde n je počet pravidel vybrané dimenze. $2n+1$ intervalů vznikne v momentě, kdy se žádná dvě pravidla nepřekrývají.
2. V druhém kroku se každému intervalu vzniklému v předchozím kroku vytvoří bitový vektor o stejné velikosti, jako je počet pravidel dané dimenze. Bitovému vektoru se poté nastaví 1 na pozice pravidel, která odpovídají danému intervalu. Ve výsledku by vektor pro dimenzi obsahující 6 pravidel, kde zpracovávaný interval odpovídá prvnímu a čtvrtému pravidlu vypadal takto: 100100. Pro případ shody paketu s více pravidly je vhodné, aby byla pravidla v bitovém vektoru řazena podle priority a nemusela se dále prohledávat.

Na obr. 6 je zobrazen stavový prostor, předzpracovaný pro vyhledávání pomocí algoritmu využívajícího bitové vektory. Stavový prostor obsahuje 2 dimenze a 4 pravidla.



Obr. 6 Dělení stavového prostoru na bitové vektory, převzato z [11].

Klasifikace konkrétních paketů by se pak dala shrnout do tří jednoduchých kroků.

1. Pro každou dimenzi najdeme interval, do kterého příchozí paket spadá. Tato část je vykonána libovolným vyhledávacím algoritmem.
2. Nad všemi intervaly vybranými prvním krokem provedeme logický součin (AND), čímž získáme pouze pravidla, pokud nějaká existují, která vyhovují všem dimenzím zpracovávaného paketu.
3. Poslední krok závisí na tom, zda jsme si ve fázi předzpracování seřadili pravidla v intervalech podle priority. Pokud jsou pravidla seřazena, vybereme pravidlo s nejvyšší prioritou. V případě, že pravidla nebyla seřazena, je nutno všechna vyhovující pravidla prohledat a vybrat to, které má nejvyšší prioritu.

4.4 Algoritmus Distributed crossproducting of field labels

Distributed crossproducting of field labels [5] je algoritmus primárně určený k hardwarové implementaci. Ke zrychlení běhu tohoto algoritmu se k prohledávání jednotlivých dimenzí využívá paralelismus. Díky paralelismu můžeme v první fázi identifikovat pravidla pro jednotlivé dimenze paketu zároveň.

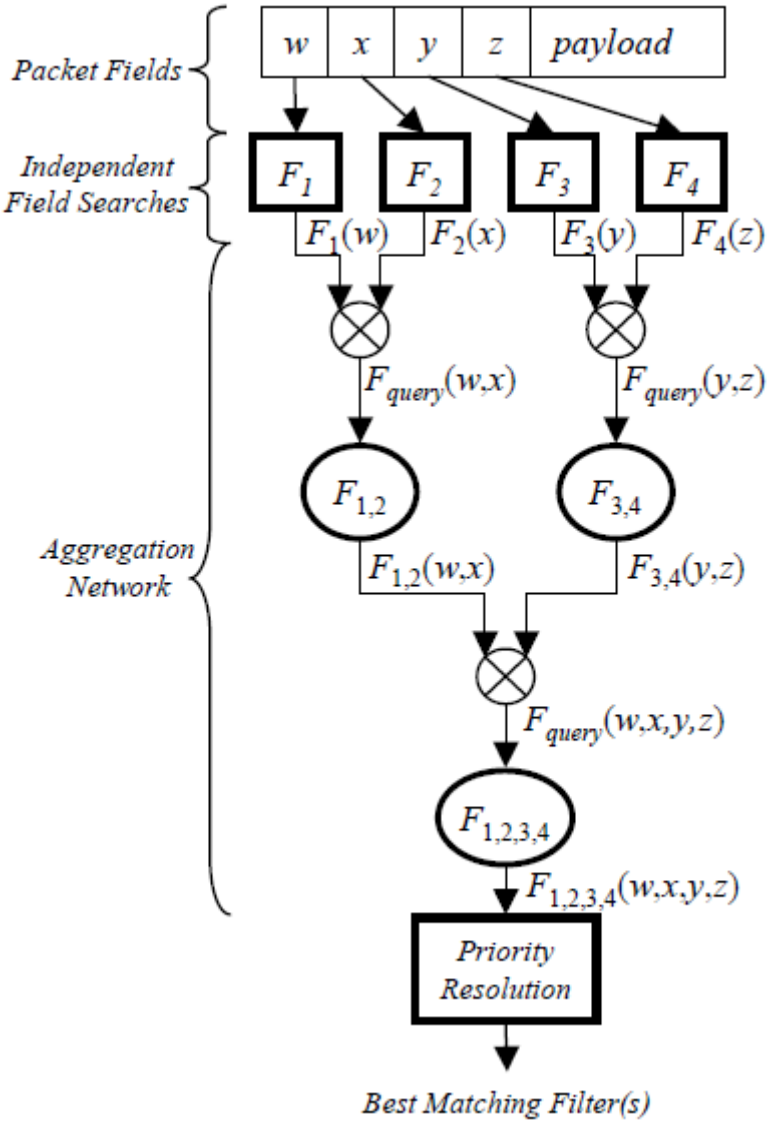
Ve fázi předzpracování se vytváří agregační síť. Tato síť má zásadní vliv na rychlost vyhledávání paketu. Každý uzel agregační sítě obsahuje kartézský součin všech unikátních výsledků předchozího kroku, odpovídajících jednomu nebo více pravidlům. Při tvorbě agregační sítě se snaží jednotlivé uzly zvolit tak, aby uzly měly uložený co nejmenší počet možných hodnot a tím se minimalizoval počet přístupů do paměti. Tato snaha vede ke snaze vyhnout se uzlům spojujícím více než dva výsledky.

Na obr. 7 je zobrazen klasifikační proces pomocí DCFL algoritmu vyhledávajícího nad čtyřmi dimenzemi. Tento konkrétní příklad by se dal rozdělit na tři fáze vyhledávání.

1. V prvním kroku algoritmus identifikuje pravidla pro jednotlivé dimenze (označené jako F1, F2, ...). Pro identifikaci pravidel v prvním kroku může využít pro každou dimenzi jiný algoritmus. Toto je velikou výhodou, protože pro 32 bitové IP adresy je určitě vhodné zvolit jiný vyhledávací mechanismus, než například pro protokoly, kterých je omezené množství a mnohem častěji se v definicích sad pravidel opakují.
2. Ve druhém kroku algoritmus sloučí výsledky pro dimenze F_1 a F_2, respektive F_3 a F_4. Toto sloučení probíhá tak, že se vytvoří kartézský součin množin dvojic dimenzí a pro každou takto vzniklou zkontroluje, zda není obsažena v předpočítané množině unikátních hodnot $F_{1,2}$ pro dimenze F_1 a F_2, respektive $F_{3,4}$ pro dimenze F_3 a F_4. Pokud je obsažena v množině unikátních hodnot daného kroku, přidáme tuto hodnotu do

množiny výsledků prováděného kroku. Stejně postupujeme při sloučení dimenzí $F_{1,2}$ a $F_{3,4}$.

3. Posledním krokem přiřazení pravidla k paketu je identifikování pravidla s nejvyšší prioritou z množiny výsledků.



Obr. 7 Průběh klasifikace paketu algoritmem DCFL, převzato z [5].

4.5 Algoritmus Multi-subset crossproduct

Multi-subset crossproduct [6] je algoritmus který využívá hašovací tabulky a Bloomův filtr [6]. Ve fázi předzpracování jsou zde vytvořeny hašovací tabulky. V naivní variantě byla použita pouze jedna tabulka, což vedlo k vytvoření velkého množství pseudo-pravidel a velikým paměťovým nárokům algoritmu. Na obr. 8 je zobrazena naivní verze algoritmu, která využívá naivní variantu algoritmu pro 2 dimenze a 4 bity a je zobrazená pomocí trie obr. 8. Proto, aby byl algoritmus v praxi použitelný, se přistoupilo k vytvoření více množin. Verze algoritmu s rozdělením do více množin je zobrazena na

obr. 9. Po porovnání obou obrázků je vidět, že i v čtyřbitovém prostoru a na pouhých šesti pravidlech rozdíl činí 8 záznamů.

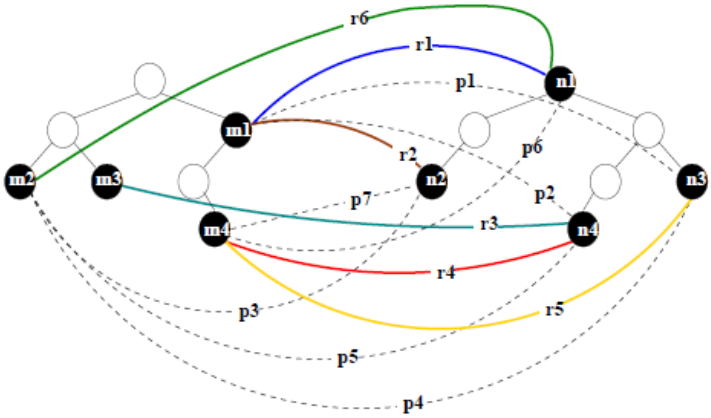
Algoritmus díky rozdělení do více podmnožin podstatně ušetří na velikosti potřebné paměti, bohužel ovšem za cenu více přístupů do ní. Při vyhledávání nejdelšího prefixu je potřeba přistupovat do každé z množin, protože nejdelší prefix nemusí nutně být nejdelší i v jiné.

Bloomův filtr je datová struktura používaná k ověření, zda prvek patří do množiny. Jedná se pouze o pravděpodobnostní strukturu, díky čemu při ověřování může dojít k chybě. Chyba ovšem může nastat pouze při dotazu, zda prvek do množiny patří. Odpovědi na dotaz, jestli prvek do množiny nepatří, jsou vždy správné. Čím více je v množině prvků, tím větší je pravděpodobnost chyby.

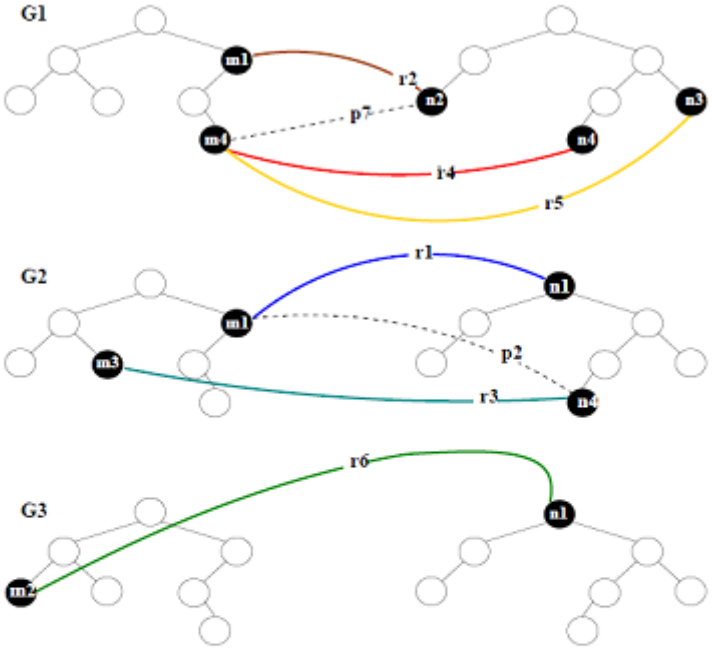
Tabulka 1 Pravidla k příkladu algoritmu MSC

R1	1*	*
R2	1*	00*
R3	01*	100*
R4	101*	100*
R5	101*	11*
R6	00*	*

	1*	*	r1
	1*	00*	r2
p1	1*	11*	r1
p2	1*	100*	r1
	00*	*	r6
p3	00*	00*	r6
p4	00*	11*	r6
p5	00*	100*	r6
	01*	*	
	01*	00*	
	01*	11*	
	01*	100*	r3
p6	101*	*	r1
p7	101*	00*	r1 r2
	101*	11*	r5
	101*	100*	r4



Obr. 8 Naivní verze algoritmu MSC, převzato z [6].



	G1	G2	G3
1*	1	1	–
00*	–	–	2
01*	–	2	–
101*	3	1	–

LPM Table for field 1

	G1	G2	G3
*	–	0	0
00*	2	0	0
11*	2	0	0
100*	3	3	0

LPM Table for field 2

Obr. 9 Algoritmus MSC po rozložení na více tabulek, převzato z [6].

5 Algoritmus Recursive Flow Classification

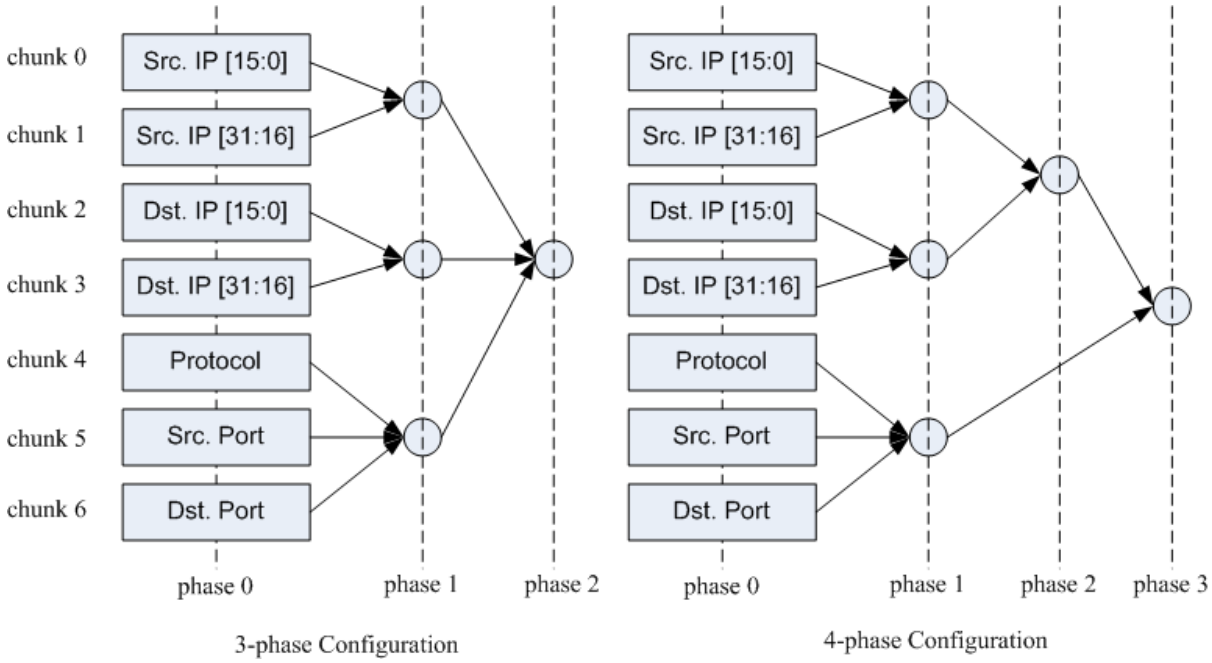
Recursive Flow Classification [13][11] je algoritmus pro klasifikaci paketů založený na dekompozici. Jedná se o rychlý, flexibilní algoritmus s velikými paměťovými nároky, který ačkoliv byl vyvinut pro implementaci v hardwaru, je vhodný i pro softwarovou implementaci.

Vytvoření tohoto algoritmu předcházela analýza pravidel používaných v reálných sítích. Bylo posbíráno více než 40 000 pravidel od více než 100 různých poskytovatelů internetu. Pro tato pravidla byly zjištěny následující charakteristiky.

- Většina klasifikátorů neobsahuje veliký počet pravidel. Pouze 0.7 % klasifikátorů obsahuje více než 1000 pravidel a průměrný počet pravidel je 50.
- Syntaxe klasifikátorů povoluje maximálně 8 položek z hlavičky paketu. Zdrojovou a cílovou IP adresu, zdrojový a cílový port, druh služby (type-of-service), protokol transportní vrstvy a příznaky protokolu transportní vrstvy. Okolo šedesáti procent klasifikátorů ovšem pracuje pouze se čtyřmi položkami.
- Protokol transportní vrstvy neobsahuje velké množství unikátních hodnot.
- Porty obsahují specifikace zahrnující velké množství hodnot. Například hodnota „větší než 1023“ se objevila v zhruba 9 % případů.
- Asi 14 % klasifikátorů obsahuje alespoň jednu nesouvislou masku. Nesouvislá maska je taková, která nemá všechny bity s hodnotou 1 souvisle za sebou.
- Je běžné, že různá pravidla sdílejí stejné hodnoty v jedné nebo více dimenzích.
- Průměrně 15 % pravidel v klasifikátoru je nadbytečných.

Hlavní myšlenkou tohoto algoritmu bylo, že kombinací všech položek z paketu se vytvoří index, přes který se přímo přistoupí do tabulky obsahující výsledná pravidla. Bohužel pro standardní pětici by tato tabulka byla velmi paměťově náročná. Z tohoto důvodu byl zaveden způsob vyhledávání používaný v algoritmu využívajícím kartézský součin hodnot a vyhledání cílového pravidla bylo rozděleno do více fází. Pro klasifikaci standardní pětice se doporučují 3 nebo 4 fáze. Možný postup vyhledávání podle počtu fází je zobrazen na následujícím obrázku (obr. 10).

Dalším problémem tohoto algoritmu bylo, že tabulka první fáze obsahuje tolik záznamů, jako je velikost dané dimenze. To pro IP adresu činí 2^{32} záznamů. Tento problém byl vyřešen vytvořením takzvaných bloků (v originále nazývaných chunks), které rozdělují IP adresy na dvě tabulky, každá o 2^{16} záznamech.



Obr. 10 Postup vyhledávání algoritmu RFC v závislosti na počtu fází, převzato z [11].

Recursive Flow Classification obsahuje výpočetně náročnou fázi předzpracování.

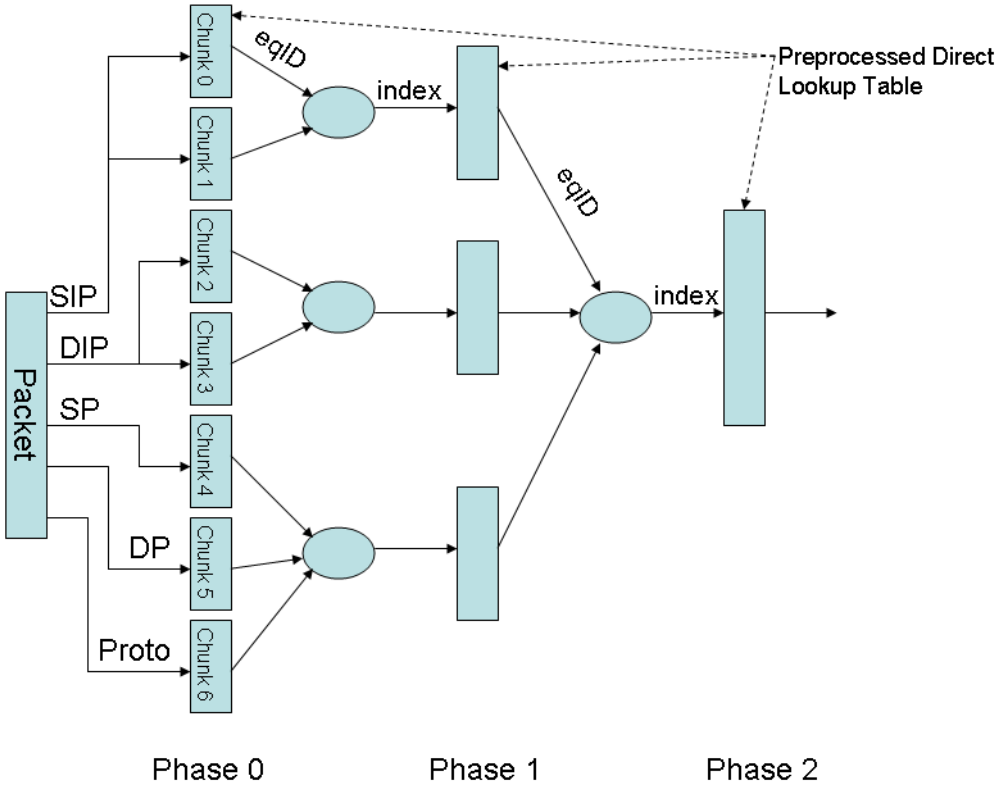
1. V první části předzpracování se pro každý blok, vypočítává tabulka obsahující tolik záznamů jako je velikost bloku. Každá unikátní sada pravidel dostane přiřazené své `eqId`. Část předzpracování pracující přímo s hodnotami z klasifikátoru je nazývána fází 0. Fáze 0 je zobrazena na pseudokódu 1.
2. V každé další fázi je, obdobným způsobem jako ve fázi 0, spočítáno `eqId` pro kombinaci dvou nebo více bloků. Záznam vytvořený v této fázi je uložen do paměti pomocí jednoduché hašovací funkce, která zajistí, že se pravidla navzájem nepřepíší. Indexem do paměti, pokud kombinujeme dva bloky, tedy je následující funkce `eqId1*početEqId2+eqId2`. Počet vyvolání tohoto kroku je závislý na zvolené konfiguraci algoritmu.

Pseudokód 1: Fáze 0 předzpracování klasifikátoru.

```
for firstRule in rules:
    for secondRule in rules:
        if firstRule.covers(secondRule):
            if not ruleList.contains(secondRule):
                ruleList.addRule(secondRule)
        i = i+1
    if not eqIdList.contains(ruleList):
        eqIdList.createNewEntry(ruleList)
return eqIdList
```

Klasifikace paketů se dá rozdělit na tři části. Průběh klasifikace paketu je zobrazen na obr. 11.

1. V první fázi jsou hodnoty paketů používané pro klasifikaci rozděleny na bloky. Pomocí binární hodnoty těchto bloků je pro každý z nich z paměti vytáhnuto odpovídající `eqId`.
2. V dalších částech se, pomocí jednoduché hašovací funkce zmiňované v druhém kroku fáze předzpracování, z paměti načítají `eqId` pro následující fáze.
3. Po úspěšné kombinaci všech dimenzí nám zůstane jeden výsledek, označující číslo odpovídajícího pravidla.



Obr. 11 Průběh klasifikace paketu pomocí algoritmu RFC, převzato z [11].

5.1 Výkon algoritmu Recursive Flow Classification

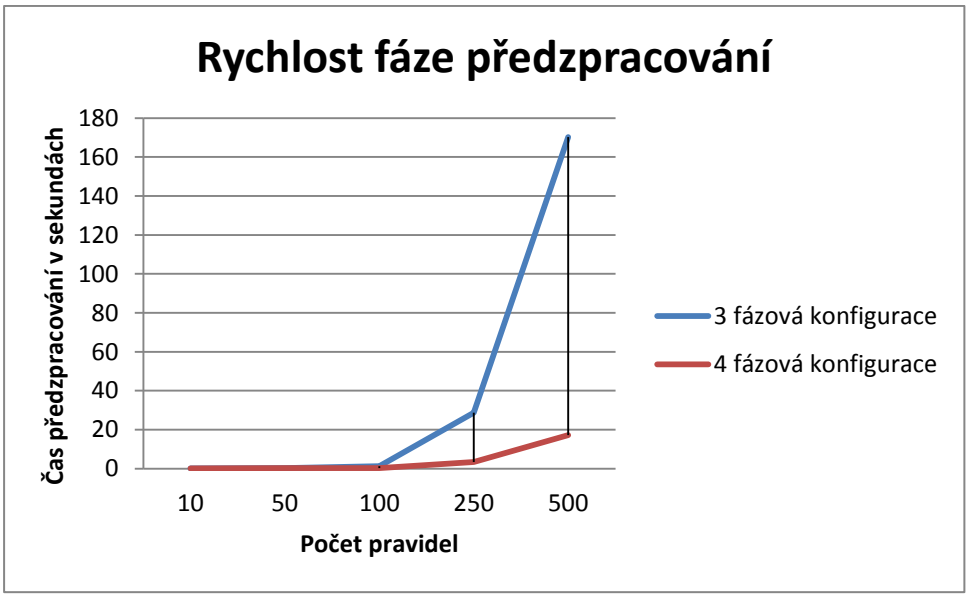
V této kapitole se podíváme na výkonost algoritmu Recursive Flow Classification a to hlavně z hlediska předzpracování a z hlediska samotné klasifikace paketu. Všechny použité sady pravidel a paketů jsou obsaženy na přiloženém CD.

5.1.1 Fáze předzpracování

Na výkon algoritmu má veliký vliv počet fází klasifikace, ale i druh použitého stromu. Algoritmus RFC neumožňuje rychlé úpravy předzpracovaných struktur a, v nejhorším případě, je při úpravě pravidel potřeba přepočítat všechny struktury. Při odebrání pravidla lze upravit pouze sady pravidel uložené ve finální fázi algoritmu. Přidání pravidla vždy vyžaduje přepočítání všech struktur. Pro

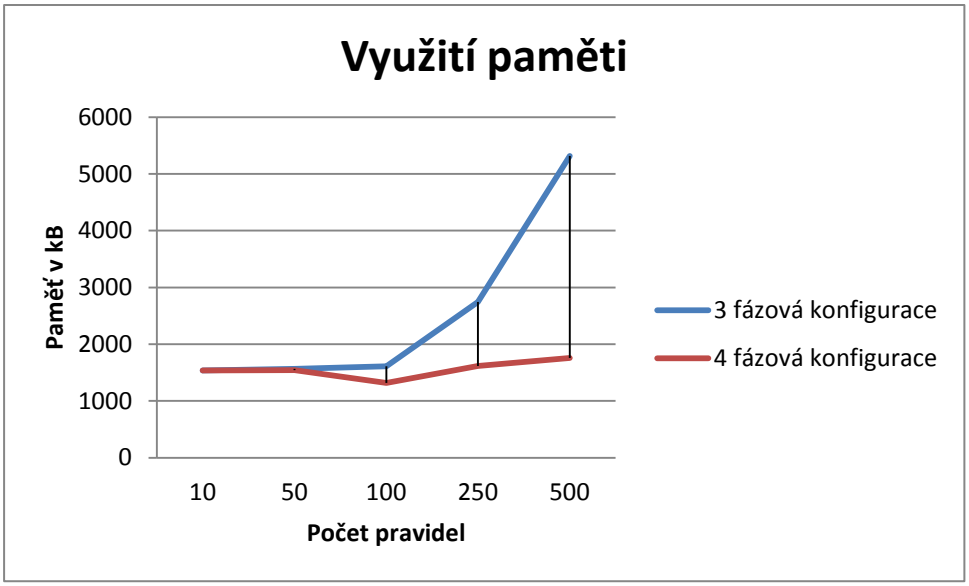
měření v této kapitole byla použita standardní pětice (zdrojová IP adresa, cílová IP adresa, zdrojový port, cílový port a protokol) rozdělená do 7 bloků. Měření byla provedena na konfiguracích zobrazených na obr. 10. Při větším počtu pravidel jsou rozdíly mezi 3 fázovou a 4 fázovou konfigurací patrné.

Rozdíly v rychlosti výpočtu datových struktur ve fázi předzpracování jsou vidět na obr. 12. Ačkoliv rychlost fáze předzpracování není kritická, rozdíl v rychlosti pro 500 pravidel činí 153 sekund. Předzpracování 3 fázové konfigurace pro 500 pravidel tedy trvá skoro deset krát déle než předzpracování 4 fázové konfigurace.



Obr. 12 Závislost rychlosti fáze předzpracování na počtu fází.

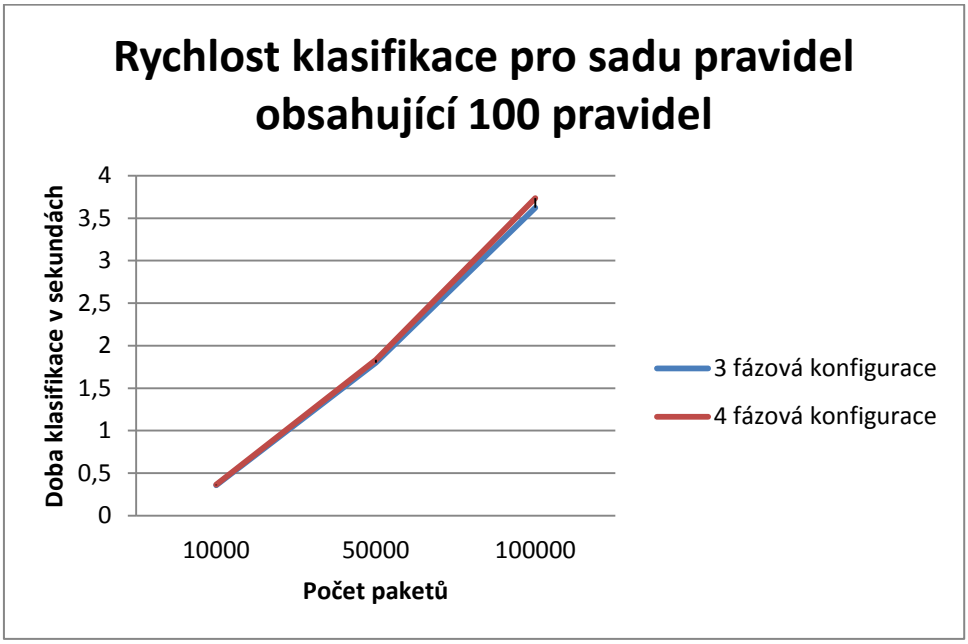
Na obr. 13 jsou znázorněny rozdíly ve velikosti datových struktur vzniklých ve fázi předzpracování. Využití paměti začíná těsně na 1.5 megabytu pro 10 pravidel. Toto číslo se může zdát vysoké, ale je způsobeno fází 0 předzpracování, kde je potřeba pro každou možnou hodnotu uložit `eqId` sady pravidel, která této hodnotě náleží. To činí 6 tabulek, které obsahují 65535 hodnot o 4 Bytech, to je velikost proměnné typu `integer` pro jazyk C, a 1 tabulku pro protokol, která zpravidla obsahuje mnohem méně záznamů. Pro jazyk Python, by toto číslo bylo mnohem větší, protože objekt typu `integer` zde zabírá 24 bytů. Prvních 6 bloků (bez protokolu) tedy v nejhorším případě zabírá 9215,86 kB a v reálných případech se její velikost o moc neliší od nejhorších případů, protože jakýkoliv wildcard znak způsobí její úplné naplnění.



Obr. 13 Velikost potřebné paměti pro uložení předzpracovaných pravidel v závislosti na počtu fází.

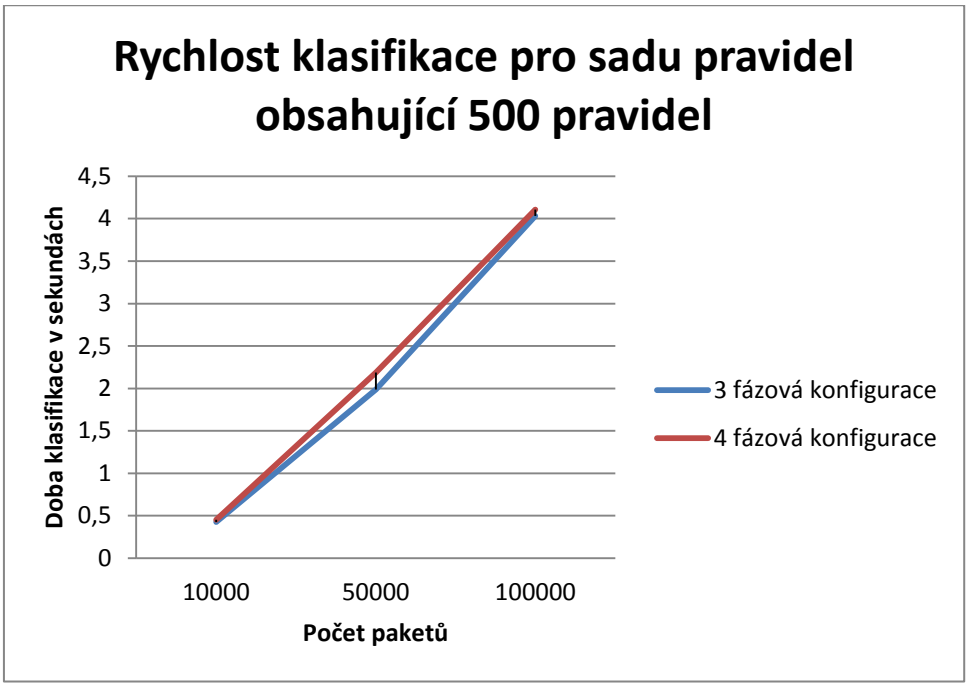
5.1.2 Klasifikace paketů

U této fáze je kladen veliký důraz na rychlost. Pokud by klasifikace neprobíhala dostatečně rychle, provoz na síti by mohl být značně omezen. Na obr. 14 je zobrazen rozdíl v rychlosti klasifikace mezi 3 a 4 fázovou konfigurací pro sadu pravidel obsahující 100 pravidel a na obr. 15 pro sadu pravidel obsahující 500 pravidel. Na obr. 14, obr. 15 a v tabulce 2 a tabulce 3 je vidět porovnání výkonu jednotlivých konfigurací algoritmu RFC. Při bližším zkoumání výsledků je potřeba zohlednit fakt, že výsledky byly vytvořeny pouze jako porovnání výkonu dvou zvolených konfigurací algoritmu RFC v jazyce Python.



Obr. 14 Rychlost klasifikace paketů algoritmem Recursive Flow Classification v sekundách.

Při porovnání obou konfigurací algoritmu RFC lze vidět, že, na rozdíl od rychlosti fáze předzpracování a využití paměti, rozdíl v rychlosti klasifikace paketů není veliký.



Obr. 15 Rychlost klasifikace paketů algoritmem Recursive Flow Classification v sekundách.

Tabulka 2 Výsledky klasifikace 3 fázové konfigurace algoritmu Recursive Flow Classification.

3 fázová konfigurace				
Počet pravidel	Počet paketů	Rychlost klasifikace v sekundách	Paketů za sekundu	Propustnost v kB/s
100	10000	0,356	28089	548,61
100	50000	1,801	27762	542,22
100	100000	3,621	27616	539,37
500	10000	0,425	23529	459,55
500	50000	1,988	25150	491,21
500	100000	4,033	24795	484,28

Tabulka 3 Výsledky klasifikace 4 fázové konfigurace algoritmu Recursive Flow Classification.

4 fázová konfigurace				
Počet pravidel	Počet paketů	Rychlost klasifikace v sekundách	Paketů za sekundu	Propustnost v kB/s
100	10000	0,365	27397	535,1
100	50000	1,832	27292	533,05
100	100000	3,735	26773	522,91
500	10000	0,453	22075	431,15
500	50000	2,187	22862	446,52
500	100000	4,108	24342	475,43

6 Implementace

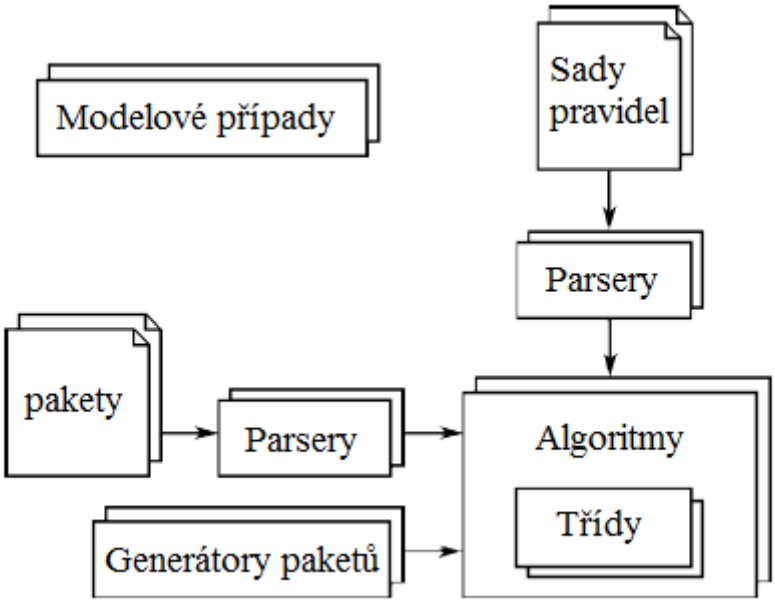
Implementace probíhala v jazyce Python verze 2.6. Jazyk Python byl zvolen kvůli knihovně Netbench, která poskytla pomocné třídy pro implementaci.

6.1 Knihovna Netbench

Experimentální knihovna Netbench [12], vyvíjená na Fakultě informačních technologií na Vysokém učení technickém v Brně, byla vytvořena s následujícími pěti cíli.

- Sloužit jako nezávislá platforma pro vědecké pracovníky, aniž by byla pevně spjata s jednou výzkumnou skupinou.
- Umožnit rychlé prototypování algoritmů.
- Zjednodušit úkoly, které jsou prováděny při mnoha experimentech
- Poskytovat komplexní sady dat pro vyhledávání IP adres, klasifikaci paketů a vyhledávání pomocí regulárních výrazů.
- Sloužit pro vzdělávací účely.

Na obr. 16 je zobrazena struktura knihovny.



Obr. 16 Struktura knihovny Netbench, převzato z [12].

6.2 Implementace algoritmu Recursive Flow Classification

Protože algoritmus RFC pracuje s bloky místo přímo s hodnotami jednotlivých dimenzí, bylo potřeba pomocí dědičnosti upravit několik tříd z knihovny Netbench.

Jako první bylo potřeba přetížit třídu `SimpleNificParser`, která se stará o načítání pravidel ze souboru a upravit metodu pro ukládání pravidel do datové struktury tak, aby pravidla rozdělila do bloků. Tento krok mohl být proveden až po načtení pravidel, ale musel by se provést cyklus, projíždějící všechna pravidla.

Dále bylo třeba upravit třídu `Prefix`, opět z důvodu rozdělení pravidel do bloků. Při tvorbě tříd, které vycházely ze třídy `Prefix`, bylo potřeba dát pozor na zpětnou kompatibilitu, aby se daly tyto třídy využít v dalších částech algoritmu implementovaných pomocí knihovny Netbench.

Bohužel úprava třídy `Prefix` způsobila, že nešla využít třída `PrefixSet`, která při přidávání objektu kontroluje, zda je objekt typu `Prefix`. Třída `ChunkSet`, která dědí od třídy `PrefixSet` tedy přetěžuje pouze jednu metodu, `add_prefix`, a odstraňuje z ní výše uvedené omezení. Odstraněním tohoto omezení je sice docíleno větší dynamičnosti třídy, při využití je ale potřeba dát si pozor na třídy vstupující do `ChunkSet`, aby nevznikla žádná chyba za běhu programu.

Hlavní část algoritmu byla implementována ve třídě, nazvané zkratkou algoritmu, `RFC`. Tato třída dědí od `BClassification`, která je abstraktním předkem všech klasifikačních algoritmů implementovaných v knihovně Netbench. Algoritmus se tedy skládá z následujících metod.

- `__init__` - Jedná se o konstruktor třídy. Vstupním parametrem je počet fází použitých v předzpracování. V rámci této práce byly zapracovány pouze 3 a 4 fázové varianty zobrazené na obr. 10.
- `load_ruleset` - Touto metodou je zahájena fáze předzpracování. Předzpracování je dále složeno z volání několika dalších metod. Jako první se volá metoda `_phase0`, která zajišťuje úvodní zpracování všech vstupních bloků a přiřazení příslušných `eqId`. Po provedení nulté fáze se sjednocují jednotlivá `eqId` pomocí metod `_process2Chunk` a `_process3Chunk`. Pořadí sjednocování jednotlivých `eqId` bylo zvoleno podle obr. 10. Zda bude algoritmus provádět 3 fázovou klasifikaci nebo 4 fázovou klasifikaci je možné zvolit parametrem, který se zadává do konstruktoru algoritmu. Vstupem této metody je třída `ChunkSet` vytvořená načtením pravidel ze souboru pomocí `ChunkParser`.
- `report_memory` - Po ukončení předzpracování je volána metoda `report_memory`, jejíž úkol je vypsat na obrazovku paměť využitou datovými strukturami uloženými ke

správnému fungování algoritmu. Tato metoda provolává všechny třídy uložené v rámci fungování algoritmu a každá třída vrací svou velikost, popř vypíše údaje na obrazovku.

- `Classify` - Metoda `classify` je používána ke klasifikaci jednotlivých paketů. Vstupem této metody je paket, jehož hodnoty slouží jako vstupy do tabulek vytvořených v nulté fázi předzpracování. Výstupy z této tabulky jsou pomocí jednoduché hašovací funkce transformovány na vstupy do tabulky vytvořené ve fázi první. Takto se pokračuje dále, dokud neprojdeme všechny fáze a nedojde ke klasifikaci paketu. Výstupem této metody je pravidlo, které bylo přiřazeno paketu přijatého na vstupu.
- Dále zde byly vytvořeny metody určené pro odladění, vypisující obsah jednotlivých tabulek na standardní výstup. Ve finální verzi bylo volání těchto metod zakomentováno.

Pro uložení datových struktur vytvořených v rámci předzpracování byly vytvořeny následující třídy.

- `EqIdTable` - třída `EqIdTable` obsahuje list `eqId`, které jsou přiřazovány jednotlivým pravidlům a proměnnou typu `integer`, která si pamatuje číslo posledního bodu v tomto listu.
- `RuleList` - tato třída ukládá do listu čísla pravidel a do proměnné typu `integer` počet těchto pravidel.
- `RuleTable` - třída `RuleTable` obsahuje obě výše zmíněné třídy a zaštiťuje přístup k nim. Nejzajímavější z metod obsažených ve třídě `RuleTable` jsou metody `addToRuleList` a `addToRuleListFast`. Obě tyto metody mají za účel přidat objekt typu `RuleList` do tabulky, liší se svým přístupem k tomuto problému. Metoda `addToRuleList` kontroluje, zda tabulka již neobsahuje podmnožinu přidávaného pravidla a v případě že ano přepíše `eqId` na pozici podmnožiny. Tato kontrola je potřeba pouze v nulté fázi předzpracování, kde se pracuje s hodnotami z jednotlivých dimenzí paketů. V následujících fázích zpracování ale tato kontrola není nutná, proto byla vytvořena metoda `addToRuleListFast`. V metodě `addToRuleListFast` se kontroluje pouze zda tato množina pravidel již není v listu obsažena.

Pro účely testování zde byly vytvořeny 2 další jednoduché třídy a jeden samostatně spustitelný skript.

- `Nificconvertor` - Toto je samostatně spustitelný skript, který dokáže převést pravidla z formátu používaného na stránce shrnující informace o klasifikačních algoritmech do formátu nific.
- `PacketParser` - `PacketParser` je třída, která dokáže ze souboru načítat pakety ve stejném formátu jako třída `PcapFile` používaná v knihovně Netbench. Na rozdíl od

třídy `PcapFile` ale načítá pakety z textového formátu a dává tedy větší prostor pro manuální úpravu přicházejících paketů.

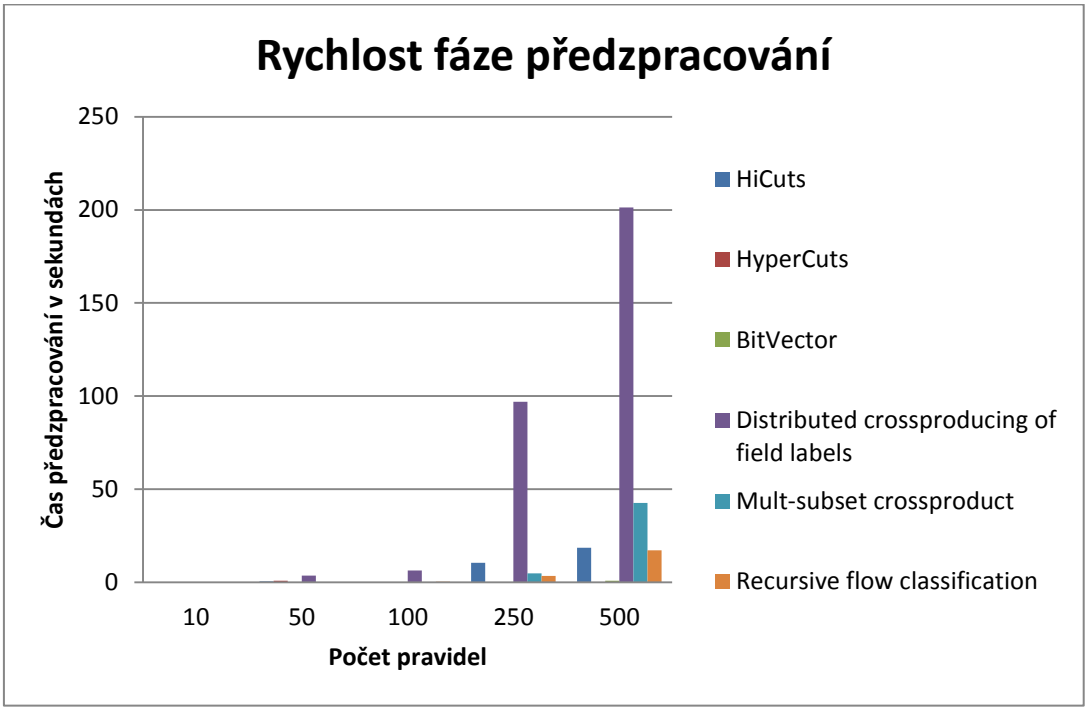
- `FakePacket` - tato třída vznikla v závislosti na třídě `PacketParser`. `Packet` vytvářený třídou `PcapFile` neumožňoval vložení ručně vytvořených hlaviček. Byla mu tedy přidána metoda, umožňující vkládání hlaviček a zároveň zachována zpětná kompatibilita.

7 Experimenty

V této kapitole jsou ukázány rozdíly ve výkonnosti algoritmů popsaných výše. Podíváme se na rozdíl v rychlosti fáze předzpracování, velikosti potřebné paměti a rychlost samotné klasifikace paketů. Všechny použité sady pravidel a paketů jsou obsažené na přiloženém CD. Pro porovnání byla použita 4 fázová konfigurace algoritmu RFC. Pro algoritmus HyperCuts byly vypuštěny testy se sadami pravidel o 250 a 500 pravidlech, z důvodu omezení rekurze v Pythonu.

7.1 Předzpracování

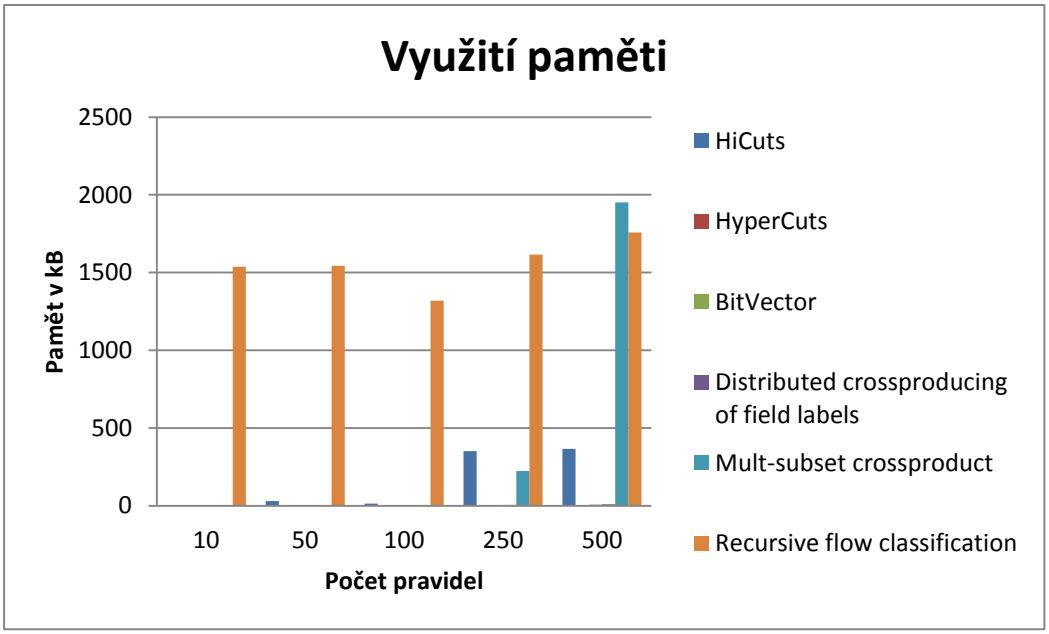
Předzpracování nepatří mezi velmi kritické metriky. Ale na obr. 17 lze vidět, že rozdíly mezi algoritmy i v této metrice jsou obrovské. Nejhůře v tomto experimentu dopadl algoritmus DCFL, kterému předzpracování 500 pravidel trvalo o málo více než 3 minuty.



Obr. 17 Porovnání rychlosti předzpracování různých algoritmů.

7.2 Paměť

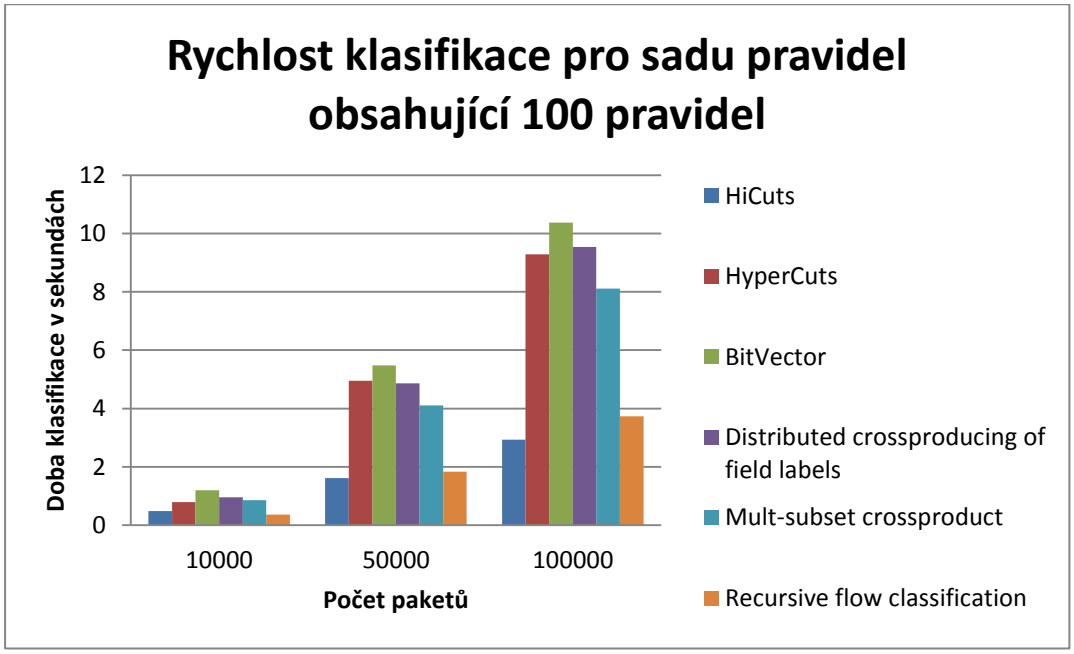
Paměťová náročnost algoritmu je důležitá zejména pro implementaci algoritmu v hardwaru. V tomto jsou porovnány paměťové nároky různých algoritmů. Na obr. 18 lze vidět, že RFC patří k více paměťově náročným algoritmům, ale na rozdíl od algoritmů HiCuts a DCFL jeho paměťová náročnost tolik nenarůstá s přibývajícími pravidly.



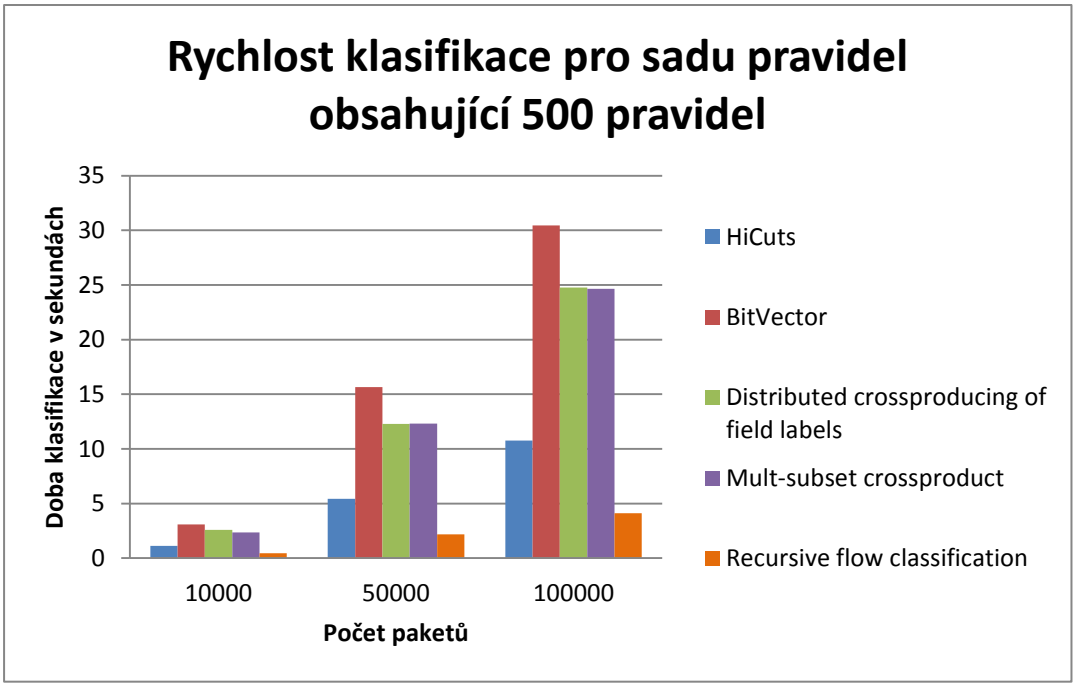
Obr. 18 Porovnání využití paměti pro různé algoritmy.

7.3 Rychlost klasifikace paketů

Rychlost klasifikace paketů je jednou z nejdůležitějších metrik protože přímo ovlivňuje propustnost sítě. Na obr. 19 a obr. 20 je vidět porovnání rychlosti klasifikace paketů různých algoritmů. Na obr. 19 je vidět, že pro menší počet pravidel nejrychleji klasifikuje algoritmus HiCuts, ale pro větší sadu pravidel zobrazenou grafem na obr. 20 už lépe funguje algoritmus RFC.



Obr. 19 Porovnání rychlosti klasifikace různých algoritmů se sadou 100 pravidel.



Obr. 20 Porovnání rychlosti klasifikace různých algoritmů se sadou 500 pravidel.

8 Závěr

V této bakalářské práci byl popsán rozvoj počítačových sítí a jejich vrstvové modely. Byla zde rozebrána problematika klasifikace paketů, její způsoby a metriky na porovnání klasifikačních algoritmů.

Dále byly popsány vybrané algoritmy z experimentální knihovny Netbench a jejich fungováním. Podrobněji byl popsán algoritmus RFC, který byl vybrán k implementaci v rámci této bakalářské práce. Pro algoritmus RFC byly rozebrány dvě z mnoha jeho možných konfigurací a tyto konfigurace byly experimentálně porovnány z hlediska rychlosti fáze předzpracování, paměťové náročnosti a rychlosti klasifikace paketů.

Byla popsána implementace algoritmu RFC, třídy, které vznikly lehkými úpravami tříd poskytnutých knihovnou Netbench, třídy které vznikly pro správné fungování algoritmu, ale i třídy které vznikly čistě z testovacích důvodů.

Na závěr práce byly provedeny experimenty nad algoritmy, které byly popsány v rámci této práce. Z výsledků experimentů je vidět, že algoritmus RFC sice ke správnému fungování potřebuje relativně velké množství paměti, díky tomu ale dokáže pakety klasifikovat velikou rychlostí.

Bylo zjištěno, že algoritmus RFC je, v případě že máme k dispozici dostatek paměti, výkonným algoritmem pro klasifikaci paketů. Jako možná vylepšení byly navrženy optimalizace, které nejsou součástí této práce. Například použitím některého z rychlých algoritmů pro vyhledávání, místo ukládání veškerých hodnot do tabulky, při fázi 0 předzpracování algoritmu RFC je možné snížit jeho paměťové nároky.

Literatura

[1] Blank A G.: *TCP/IP foundations*. San Francisco, Calif.: Sybex, c2004, xv, 284 p. ISBN 07-821-4370-9.

[2] Matoušek P.: *Architektura sítí, adresování, konfigurace TCP/IP*. FIT VUT Brno, 2012.

[3] Gupta P., McKeown N.: *Algorithms for packet classification*. Computer systems laboratory, Stanford University

[4] Matoušek P.: *Klasifikace paketů a filtrování dat*. FIT VUT Brno, 2012.

[5] Taylor E. D., Turner S. J.: *Scalable Packet Classification using Distributed Crossproducting of Field Labels*, IEE INFOCOM 2005.

[6] Dharmapurikar S., Song H., Turner J., Lockwood J.: *Fast packet classification using bloom filters.* In *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, December 03-05.12, 2006, San Jose, California, USA

[7] Gupta P., McKeown N.: *Packet classification using Hierarchical Intelligent Cuttings*. Computer systems laboratory, Stanford University, 2000.

[8] Lakshman T.V., Stiliadis D.: *High-speed policy-based packet forwarding using efficient multi-dimensional range matching.* In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, p.203-214, 31.8-04.9, 1998, Vancouver, British Columbia, Canada [doi>10.1145/285237.285283]

[9] Singh S, Baboescu F, Varghese G, Wang j: *Packet Classification Using Multidimensional Cutting*. In *Proceedings of the 2003 conference on Applications, technologies, architectures and protocols for computer communications*, SIGCOMM 2003, New York, NY, USA. ISBN 1-58113-735-4, s. 213-224.

[10] TCP/IP Overview. CISCO. *Cisco* [online]. 10.8.2005. [cit. 2013-05-14]. Dostupné z: http://www.cisco.com/en/US/tech/tk365/technologies_white_paper09186a008014f8a9.shtml

[11] *Evaluation of Packet Classification Algorithms* [online]. 2007 [cit. 2013-05-13]. Dostupné z: http://www.arl.wustl.edu/~hs1/PClassEval.html

[12] *Accelerated Network Technologies* [online]. 2009, 2013 [cit. 2013-05-13]. Dostupné z: http://merlin.fit.vutbr.cz/ant/netbench/index.html

[13] Pankaj G.: *Algorithms for Routing Lookups and Packet Classification*. USA, 2000. Dostupné z: http://klamath.stanford.edu/~pankaj/phd.html. PhD Dissertation. Standford University.

[14] Scott D., Sharp R.: *Abstracting Application-level Web Security*. In *Proceedings of the 11th international conference on World Wide Web*.,07-11.5.2002, Honolulu, Hawaii, USA [doi>10.1145/511446.511498]

Seznam příloh

K práci je přiloženo CD, které obsahuje tento text, všechny zdrojové kódy vytvořené v rámci implementační části a všechna testovací data, která byla použita pro testování a experimenty.