VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SOFTWAROVÝ MODEL FIREWALLU

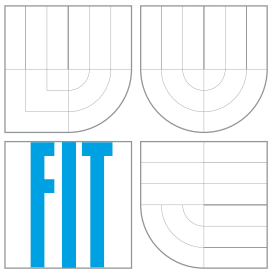
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL ŠVEC

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SOFTWAROVÝ MODEL FIREWALLU

SOFTWARE MODEL OF FIREWALL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL ŠVEC
AUTHOR

VEDOUCÍ PRÁCE — Ing. VIKTOR PUŠ
SUPERVISOR

BRNO 2011

Abstrakt

Tato bakalářská práce se zabývá problematikou bezestavových paketových filtrů a algoritmů klasifikace paketů. Hlavním úkolem je vytvořit softwarovou implementaci firewallu a změřit rychlost klasifikace paketů. Implementovaný klasifikátor využívá pro klasifikaci algoritmus Perfect-Hash. Klasifikátor je implementován v několika verzích. Jako implementační jazyk je použitý jazyk C, pro samotný klasifikátor a jazyk Python, pro vytvoření pomocných datových struktur. Jazyk C byl zvolen kvůly jeho rychlosti.

Abstract

This bachelor thesis deals with stateless packet filters and packet classification algorithms. The main task is to implement a software firewall and measure the speed of packet classification. The implemented classifier uses the Perfect Hash Classification Algorithm. The classifier is implemented in several versions. The C language is used for classifier implementation, Python language is used to create auxiliary data structures. The C language was chosen because of its speed.

Klíčová slova

Firewall, klasifikace paketů, paketový filtr, Perfect-hash, klasifikační algoritmy

Keywords

Firewall, packet classification, packet filter, Perfect-hash, classification algorithms

Citace

Michal Švec: Softwarový model firewallu, bakalářská práce, Brno, FIT VUT v Brně, 2011

Softwarový model firewallu

Prohlášení

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

.....
Michal Švec
15. mája 2011

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Viktorovi Pušovi za ochotu a nápomoc při řešení problémů.

Obsah

Kapitola 1

Úvod

Táto práca bola vytvorená na Fakulte informačných technológií na Vysokém učením technickém v Brně. Práca sa zameriava na problematiku bez stavových paketových filtrov a algoritmov klasifikácie paketov. So stálym vývojom a zrýchlovaním sieťových technológií je nutné vyvíjať aj nové, rýchlejšie a efektívnejšie algoritmy pre klasifikáciu paketov. Ďalším faktorom je pribúdajúci počet jadier na čipe, ktoré je možné pri klasifikácii efektívne využiť.

Hlavnou úlohou tejto práce bolo navrhnúť softwarový model tak, aby skutočne odpovedal funkcii reálneho firewallu. Následne vytvoriť jeho softwarovú implementáciu a previesť sadu experimentov, čím sa mali overiť vlastnosti samotnej implementácie. Dôraz bol kladený najme na rýchlosť klasifikácie jednotlivých paketov. Cieľom bolo určiť, ktorá časť algoritmu je najviac pomalá, ako by sa dala táto rýchlosť zvýšiť a tiež uplatniť v praxi. Pri experimentoch sa bral ohľad tiež na pamäťové nároky, pretože samotná implementácia nájde svoje uplatnenie najme v hardwari (ešte vyššia rýchlosť), kde je pamäť pomerne obmedzená. Ako vhodný implementačný jazyk pre samotný algoritmus klasifikácie bol zvolený *jazyk C* vďaka svojej rýchlosti a relatívne dobrej výške abstrakcie. Pre vytvorenie pred pripravenej dátovej štruktúry s pravidlami bol použitý jazyk *Python*. Jeho rýchlosť je síce menšia, ale tabuľka sa vytvorí dopredu a vo vlastnej klasifikácií sa už do nej len pristupuje.

1.1 Prehľad kapitol

Druhá kapitola sa venuje problematike firewallu. Obsahuje základne informácie a príklad použitia. V nasledujúcich podkapitolách sú priblížené jednotlivé druhy firewallov a záver kapitoly je venovaný demilitarizovanej zóne. Tretia kapitola sa zaoberá klasifikáciou paketov. Popisuje základný princíp a problémy, ktoré pri klasifikácii vznikajú. Obsahuje podkapitolu, v ktorej sú opísané hodnotiace kritéria pri porovnávaní klasifikačných algoritmov. Štvrtá kapitola opisuje jednotlivé klasifikačné algoritmy, ich základné vlastnosti a možnosti použitia. Obsahuje podkapitoly, v ktorých sú podrobnejšie popísané niektoré zmienené algoritmy. Dôraz sa však kladie na algoritmus použitý v implementácii. Piata kapitola sa bližšie venuje klasifikačnému algoritmu, ktorý je použitý v samotnej implementácií. Podobne ako predchádzajúca kapitola, popisuje jeho základné vlastnosti a princíp činnosti. Obsahuje podkapitolu zaoberajúcu sa Perfect-Hash funkciou, ktorá je použitá v implementovanom klasifikačnom algoritme. Šiesta kapitola sa zaoberá samotným návrhom implementácie klasifikátoru, ktorý využíva Perfect-Hash funkciu. Opisuje celkový priebeh klasifikácie a obsahuje blokovú schému klasifikačného algoritmu. Tiež bližšie po-

pisuje jednotlivé implementované verzie klasifikátoru a použité moduly v nich. Siedma kapitola je venovaná samotnému meraniu rýchlosti implementácie. Popisuje postup merania a meraciu zostavu. Obsahuje tabuľky s nameranými hodnotami a grafiké závislosti. Posledná kapitola obsahuje záver a celkové zhodnotenie práce. Poukazuje na dosiahnuté výsledky v rýchlosti klasifikácie paketov.

Kapitola 2

Firewall

Firewall je sieťové zariadenie alebo softvér, ktorého úlohou je oddeliť siete s rôznymi prístupovými právami (typicky napr. Internet – globálna sieť a Intranet – súkromná sieť) a kontrolovať tok dát medzi týmito sieťami.

Kontrola údajov prebieha na základe aplikovania pravidiel, ktoré určujú podmienky a akcie. Rozlišujeme dva základné typy pravidiel:

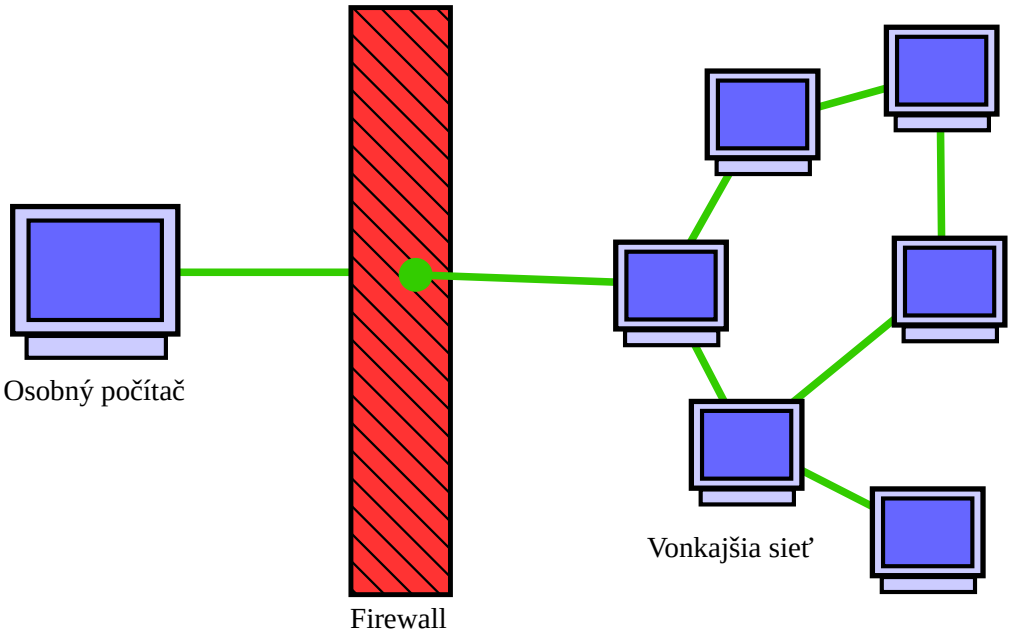
- *Statické* – sú platné stále
- *Dynamické* – sú platné za určitých podmienok (napr. podľa dennej doby)

Podmienky sa stanovujú pre údaje, ktoré možno získať z dátového toku (napr. zdrojová, cieľová adresu, zdrojový alebo cieľový port a rôzne iné). Úlohou firewallu je vyhodnotiť podmienky, a ak je podmienka splnená, vykonať príslušnú akciu. Dve základné akcie sú „povoliť kontrolovaný paket“ a „odmietnuť kontrolovaný paket“. Po vykonaní takejto akcie firewall prestane paket spracovávať. Existujú však aj iné akcie, ktoré neurčujú osud paketu a slúžia napr. na logovanie hlavičiek paketu, zmenu hlavičiek paketu a podobne. Pri rozhodovaní sa používajú filozofie ACCEPT (čo nie je zakázané je povolené), alebo REJECT (čo nie je povolené je zakázané). [1]

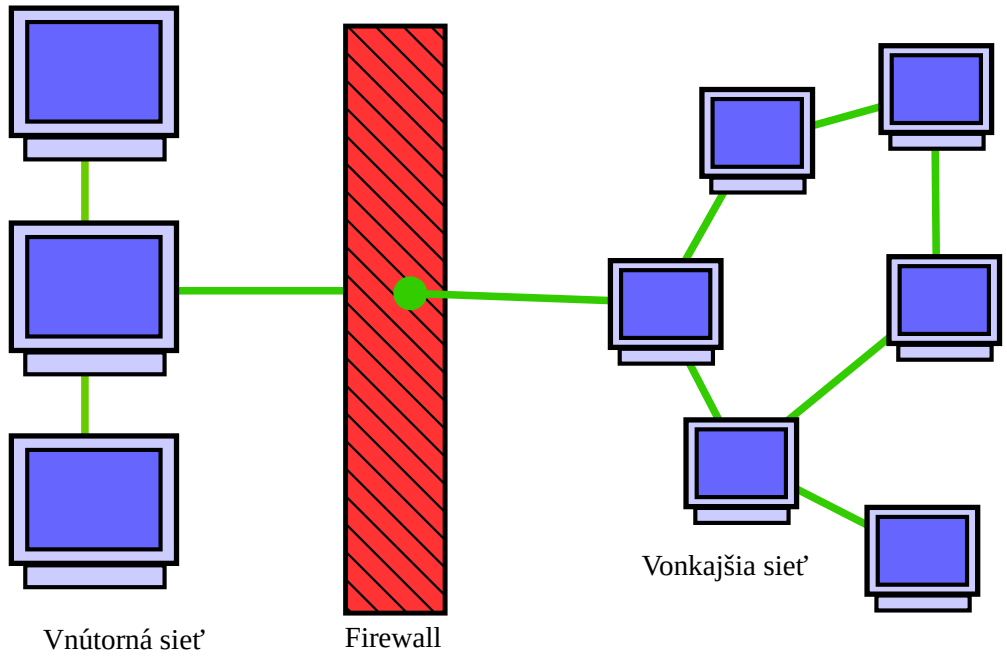
Ďalšou vlastnosťou firewallu, ktorá sa často používa, i keď nejde o filtrovanie, je schopnosť prekladu adries (Network Address Translation - NAT). NAT umožňuje zmeniť zdrojové a cieľové adresy v paketoch, čím sa najčastejšie umožňuje komunikácia so sieťami s privátnymi adresami (napr. 10.0.0.0/8). Preklad adries prebieha tiež pomocou pravidiel. Príklady možného použitia firewallu sú na obrázkoch 2.1 a 2.2. [2]

2.1 Rozdelenie firewallov podľa hĺbky kontroly

- Paketové filtre
- Stavové paketové filtre
- Aplikačné firewally spolu s paketovým filtrom a kontrolou protokolov



Obrázok 2.1: Firewall - oddelenie osobného počítača od vonkajšej siete



Obrázok 2.2: Firewall - oddelenie vnútornej siete od vonkajšej siete

2.2 Paketový filter

Jedná sa o najjednoduchší typ firewallu. Filtrovanie je postavené na základe kontroly informácií v hlavičkách protokolov. Kontrola sa zameriava na 3. a 4. vrstvu modelu ISO/OSI, teda na sieťovú a transportnú vrstvu. Pravidlá je možné zamerať na typ protokolu, zdrojovú a cieľovú adresu, alebo zdrojový a cieľový port. Zvyčajne na rozhodovanie požíva nejakú formu tabuľky, ktorú *zhora-nadol* prechádza a v prípade zhody uplatňuje akcie, ktoré v tabuľke nájde. Ak v tabuľke nenájde odpovedajúci záznam, tak sa použije implicitná politika pre danú tabuľku. Filter môže pracovať aj s niekoľkými tabuľkami, kde sa vo väčšine prípadov uplatňuje jedna tabuľka pre jeden smer komunikácie. K výhodám paketového filtru patria: jednoduchá implementácia, nízke požiadavky na výpočtový výkon, vysoká rýchlosť spracovania paketov a s tým aj súvisiaca vysoká paketová priepustnosť (možnosť použitia ako *hraničné firewally*). Medzi nevýhody patrí: nedostatočná hĺbka kontroly a problém pri použití neštandardných portov. Niekedy musia na serveri fungovať služby, ktoré pri komunikácii využívajú náhodne vybrané porty (napr. pri komunikácií pomocou Skype, nie je možné detekovať na základe čísla portu). Paketový filter nevidí súvislosti medzi paketmi a každý analyzuje samostatne. Vtedy je nutné, buď povoliť celý rozsah portov (príliš benevolentné), alebo ho zakázať a takéto služby nepoužívať (niekedy nerealizovateľné). [2, 3]

2.3 Stavový paketový filter

Vychádza z paketových filtrov, ale je obohatený o možnosť ukladať aktuálny stav spojenia. Pri filtrovaní sa rozhoduje, či komunikácia patrí už k povolenému spojeniu, alebo je nutné previesť rozhodovací proces od začiatku. K tomuto rozhodnutiu je potrebné udržovať tabuľku, ktorá obsahuje naviazané spojenia – *contract table*. Je možné použiť rôzne politiky pre prichádzajúce a odchádzajúce spojenia. K výhodám stavového paketového filtru patria: jednoduchšia konfigurácia a obsluha, rádovo menší počet pravidiel a vyššia miera bezpečnosti oproti paketovým filtrom. Stav spojenia možno využiť v pravidlách, a tak napríklad automaticky povoliť odpovede na všetky odoslané pakety, povoliť spojenia, ktoré súvisia s daným spojením počas jeho trvania atď. Toto sa s výhodou používa napr. na riešenie už spomenutého problému s FTP. Jeho nevýhody sú spôsobené nutnosťou udržovať stavovú informáciu, čo spôsobuje problémy pri výpadkoch a dynamickom smerovaní. Ak má sieť viac vstupných bodov, môže sa stať, že pakety odpovede neprídu do siete tou istou cestou ako vyšli požiadavky. Tu vzniká potreba zdieľania stavovej informácie medzi viacerými firewallmi, čo môže značne ovplyvniť výkon a spoľahlivosť stavových firewallov. Podobné problémy vznikajú pri paralelizácií a násobnosti s cieľom zvýšiť výkon, alebo spoľahlivosť. Implementácia stavových filtrov je značne náročná, a preto je výsledná cena produktov relatívne vysoká v porovnaní s bez stavovými filtrami. [2, 3, 4]

2.4 Aplikačný firewall

Často sa označuje ako *proxy server*. Aplikačná brána sa líši od paketových filtrov hlavne v tom, že vôbec nesmeruje pakety. Kým paketové filtre pracujú na sieťovej (IP) vrstve s prípadnou analýzou transportnej vrstvy (TCP/UDP), aplikačná brána pracuje na najvyššej aplikačnej vrstve. Aplikačná brána nie je pre sieť transparentná. Aplikácie a používatelia musia vedieť, že aplikačná brána existuje, aby mohli získať prístup do chránenej časti siete. Príkladmi aplikačných brán sú veľmi rozšírené *HTTP-Proxy* servery, alebo *Mail Relay* ser-

very. Ako už bolo povedané, aplikačná brána nie je transparentná. Každá aplikácia musí mať na aplikačnej bráne svoj modul, ktorý je schopný porozumieť aplikačnému protokolu a implementovať relevantnú časť bezpečnostnej politiky vzhľadom na daný aplikačný protokol. Štandardné aplikácie, ako sú napríklad klientské programy služieb FTP a Telnet, je síce s väčšinou aplikačných brán možne používať ďalej, ich použitie je však nepohodlné. Používateľ musí vedieť adresu brány, musí sa prihlásiť najprv na bránu a potom až na cieľový uzol, spojenia sa ťažko automatizujú vzhľadom na rôzne druhy aplikačných brán atď.

Na druhej strane však aplikačná brána poskytuje najvyššiu relatívnu bezpečnosť, keďže priamo nesmeruje pakety a môže dokonale analyzovať aplikačný protokol. Okrem toho sa pakety prechádzajúce aplikačnou bránou regenerujú - fragmenty sú zlúčené. Prípadný útok na implementáciu rodiny protokolov TCP/IP dopadne len na aplikačnú bránu a nie na žiaden uzol za ňou. Medzi nevýhody aplikačných brán patria: vyššie nároky na výpočtový výkon, pomerne nízka priepustnosť, vysoká latencia a potreba vlastných modulov pre jednotlivé protokoly. [2, 3, 4]

2.5 Paketový filter s kontrolou protokolov

Tiež označované ako *paketové analyzátory*. Pracujú na 3. až 7. vrstve modelu ISO/OSI. Dokážu dynamicky otvárať porty pre riadiace aj dátové spojenia známych protokolov. Implementujú v sebe aj hĺbkovú kontrolu spojenia. Sú schopné kontrolovať aj korektnosť prechádzajúcich dát (kontrola hlavičiek, príkazov) a sú schopné zabrániť tunelovaniu dát. Tiež je možné analyzátory použiť na filtrovanie obsahu (skripty, reklamy na webe a iné). Pomocou databázy signatúr je tiež možné odhaľovať možné útoky. Výhody: oproti aplikačným bránam je rýchlosť spracovania vyššia a neznáme protokoly je možne filtrovať na princípe paketového filtra. Nevýhody: vysoká komplexnosť, pri nájdení bezpečnostnej diery je možné kompromitovať celý systém a dátová priepustnosť oproti paketovým filtrom je nižšia. [3]

2.6 Firewall s demilitarizovanou zónou

Nejedná sa o zvláštny druh firewallu, ale o jeho spôsob použitia. Pri väčších sieťach je typické, že firewall rozdeľuje sieť na 3 časti:

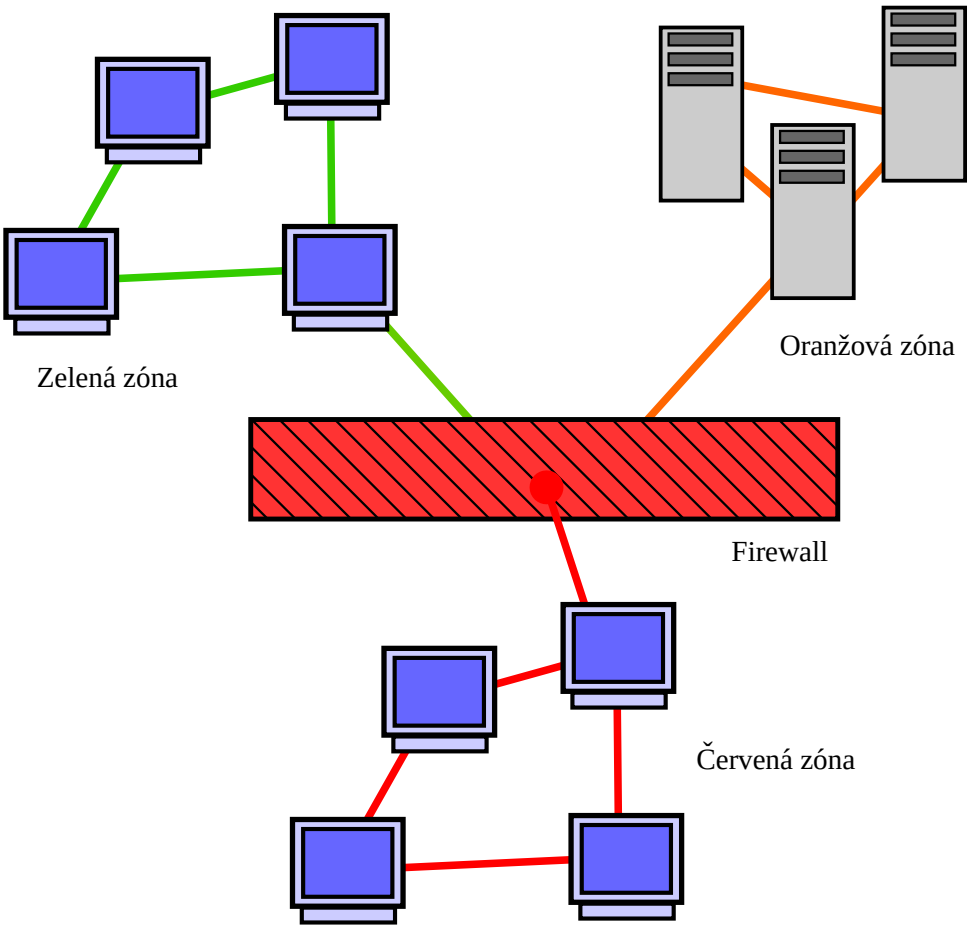
(označenie farbami je platné pre obrázok 2.3, neoficiálne)

Zelená zóna:
Bezpečná, celá komunikácia je oddelená od vonkajšieho sveta.

Oranžová zóna:
Demilitarizovaná zóna, umiestnenie pre servery, na ktoré je povolený prístup z vonka.

Červená zóna:
Vonkajšia, nebezpečná sieť, klienti sa môžu spojiť so servermi v demilitarizovanej zóne, ale do vnútornej zóny prístup nemajú.

Takýto princíp je možné aplikovať i v rámci vnútornej siete za účelom vytvoriť miesto s vyšším zabezpečením. [3]



Obrázok 2.3: Firewall - použitie firewallu s demilitarizovanou zónou

Kapitola 3

Klasifikácia paketov

Klasifikácia paketov dovoľuje sieťovým zariadeniam (firewall, smerovač a iné) poskytovať rozšírené sieťové služby. Ako príklad môže byť zvýšená forma bezpečnosti, QoS smerovanie (so zaručením kvality služby), rezervácia zdrojov a iné. Oblasť klasifikácie paketov je v neustálom vývoji. Stále je snaha vytvárať rýchlejšie a účinnejšie algoritmy pre klasifikáciu. Na jednej strane to spôsobujú faktory ako sieťová bezpečnosť a QoS, na strane druhej neustále sa zvyšujúca rýchlosť sietí. V súčasnosti existuje pomerne veľké množstvo používaných klasifikačných algoritmov. Pri výbere algoritmu je nutné vychádzať priamo z oblasti použitia tzn., že väčšina algoritmov je navrhnutá pre špecifické použitie, a preto je veľmi problematické porovnávať vlastnosti jednotlivých algoritmov. Princíp klasifikácie spočíva v triedení paketov do jednotlivých tried, pre ktoré platia určité pravidlá. V konečnom dôsledku to znamená, že v jednej triede sú len pakety, ktoré spĺňajú požiadavky tejto triedy. Pakety sa väčšinou klasifikujú podľa obsahu hlavičiek (adresy, porty, protokoly, ...) jednotlivých protokolov, ale nie je to pravidlo, klasifikácií môže podliehať aj dátový obsah. Postup klasifikácie je znázornený na obrázku 3.1. [5, 6]

3.1 Požiadavky na klasifikačné algoritmy

Rýchlosť vyhľadania:
Rýchlejšie siete potrebujú rýchlejšie algoritmy pre klasifikáciu.

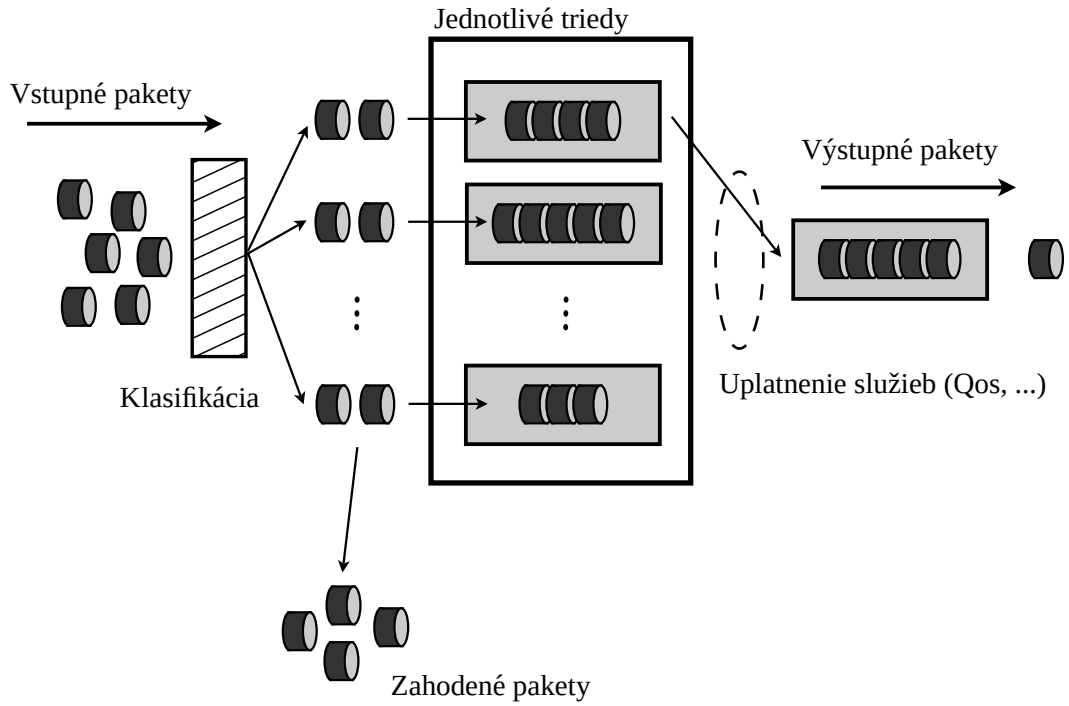
Nízke pamäťové nároky:
Možnosť použitia rýchlejších SRAM pamätí, uplatnenie algoritmov v hardwari.

Schopnosť zvládať rozsiahle klasifikátory (real-life):
Reálne klasifikátory môžu obsahovať stovky až tisícky pravidiel.

Rýchly update:
V prípade zmeny klasifikátoru je nutné zmeniť dátové štruktúry. Rozoznávame štruktúry, ktoré je možné preorganizovať (pridať prvok, odobrať prvok, meniť prvok) a štruktúry, ktoré je nutné vytvoriť znovu.

Možnosť meniť počet kontrolovaných polí hlavičky

Flexibilita v špecifikácii:
Podpora všeobecných pravidiel, operátorov a zástupných znakov.



Obrázok 3.1: Diagram klasifikácie

Kapitola 4

Klasifikačné algoritmy

Klasifikačný algoritmus má za úlohu predspracovať dané pravidlá do dátovej štruktúry, ktorá je potom použitá pri klasifikácií jednotlivých paketov. Klasifikačné algoritmy je možné rozdeliť do niekoľkých skupín (tabuľka, zdroj [6]).

Kategórie	**Algoritmy**
Základné dátové štruktúry	Linear search, caching, hierarchical trees, set-pruning trees
Geometricky založené	Grid-of-tries, AQT, FIS
Heuristické	RFC, hierarchical cuttings, tuple-space search
Hardvérové	Ternary CAM, bitmap-intersection

Tabuľka 4.1: Rozdelenie klasifikačných algoritmov

4.1 Základné dátové štruktúry

4.1.1 Linear Search

Najjednoduchšia dátová štruktúra je lineárne viazaný zoznam, v ktorom sú pravidlá uložené na základe priority. Každý paket je porovnávaný sekvenčne s jednotlivými pravidlami, až kým sa nenájde zhoda, alebo neprejdú všetky pravidlá. Algoritmus je jednoduchý a pamäťovo nenáročný, ale jeho časová zložitosť rastie lineárne s počtom pravidiel, takže je značne nevýhodný. [6]

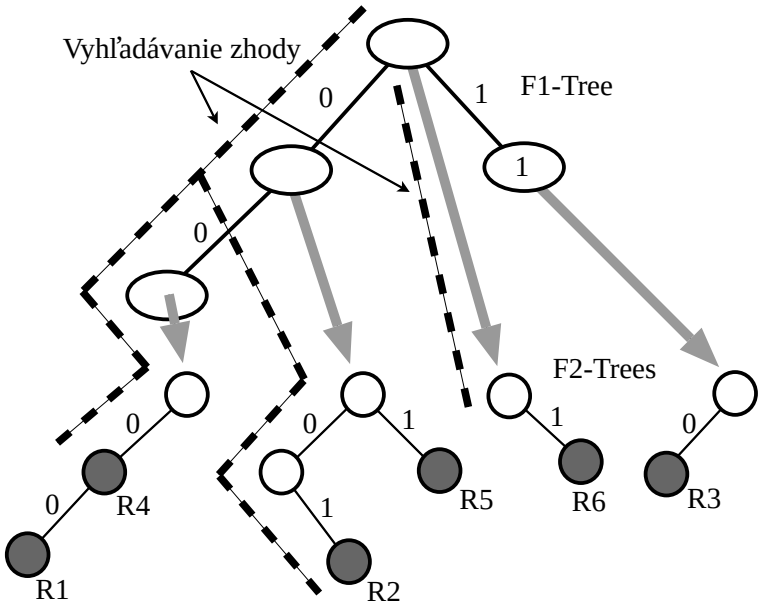
4.1.2 Hierarchical trees

D-dimenzionálny hierarchický strom je jednoduché rozšírenie 1-dimenzionálnej stromovej dátovej štruktúry. Strom je tvorený rekurzívne. Ak je d väčšie ako 1, najskôr sa vytvorí 1-dimenzionálny strom, nazvaný $F1$-tree, skupina prefixov $\{R_{j1}\}$ patriaca dimenzii $F1$ zahrňujúca všetky pravidlá klasifikátoru $C = \{R_j\}$. Pre každý prefix p v strome $F1$ vytvoríme rekurzívne $(d-1)$-dimenzionálny strom, T_p, na tie pravidlá, ktoré špecifikujú presne p v dimenzii $F1$. Prefix p je pripojený k stromu T_p použitím ukazovateľa na ďalší strom.

Tieto stromy sú tiež označované ako „multi-level trees“, „backtrackingsearch trees“, alebo „tree-of-trees“.

Klasifikácia paketov prebieha pomocou algoritmu, ktorý prechádza jednotlivé uzly a porovnáva ich z bitmi. Čas spracovania jedného paketu teda závisí od počtu dimenzií stromu a od dĺžky prefixu. Každé jedno pravidlo je uložené v maximálnej hĺbke daného stromu. Ukážka stromu je na nasledujúcom obrázku (šedé uzly predstavujú pravidlá, šedé šípky odkaz do ďalšieho stromu, cesta je zobrazená pre pakety $000, 010$). [6]

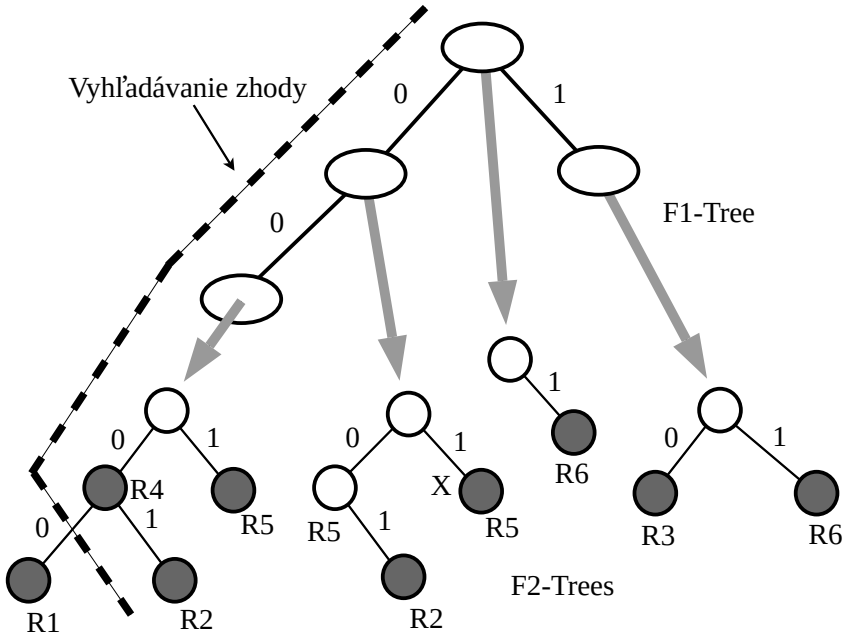
Klasifikačné algoritmy veľmi často využívajú LPM (Longest Prefix Match). LPM sa používa, buď na zníženie počtu potencionálnych pravidiel (pravidiel, kde je pravdepodobná zhoda), alebo pri riešení pravidiel, ktoré v sebe zahrňujú rozsahy.



Obrázok 4.1: Príklad stromovej štruktúry Hierarchical trees

4.1.3 Set-Prunning trees

Podobná štruktúra ako bola použitá v predchádzajúcom algoritme, ale klasifikačný čas je menší, pretože sú použité replikované pravidlá, ktoré zabraňujú zbytočnému prehľadávaniu. Replikácia zaistí, aby všetky pravidlá, ktoré prichádzajú do úvahy boli na jednej ceste v strome. Vzhľadom k tomu, že sa pravidlá opakujú, sú pamäťové nároky mierne vyššie. Táto štruktúra tiež funguje len pre relatívne statické klasifikátory. Ukážka stromu je na nasledujúcom obrázku (šedé uzly predstavujú pravidlá, šedé šípky odkaz do ďalšieho stromu, cesta je zobrazená pre pakety $000, 010$). [6]

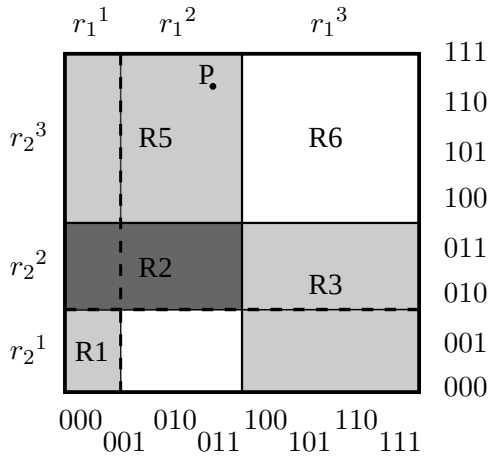


Obrázok 4.2: Príklad stromovej štruktúry Set-Prunning trees

4.2 Geometricky založené

4.2.1 Cross-producting

Algoritmus je schopný pracovať s neobmedzeným počtom dimenzií. Pakety sú klasifikované na základe výsledkov z oddelených 1-dimenzionálnych vyhľadávaní. Najprv je nutné vytvoriť špeciálnu dátovú štruktúru (*Crossproduct table*). Štruktúra zahŕňa výpočet sady rozsahov, ktoré projektujú zadané pravidlá v každej dimenzii. Takto vznikne tabuľka, ktorá obsahuje položky prepočítané tak, aby priamo vyhovovali pravidlám. Klasifikáciou v každej dimenzii získame n-ticu. Na základe n-tice vyhľadáme miesto v tabuľke, ktoré mapuje pravidlá. Algoritmus je vhodný len pre menšie a relatívne statické klasifikátory. [6]



Obrázok 4.3: Mapovanie pravidel

$(r_1{}^1, r_2{}^1)$	**R1**
$(r_1{}^1, r_2{}^2)$	**R2**
$(r_1{}^1, r_2{}^3)$	**R5**
$(r_1{}^2, r_2{}^1)$	—
$(r_1{}^2, r_2{}^2)$	**R2**
	R5
$(r_1{}^3, r_2{}^1)$	**R3**
$(r_1{}^3, r_2{}^2)$	**R3**
$(r_1{}^3, r_2{}^3)$	**R5**

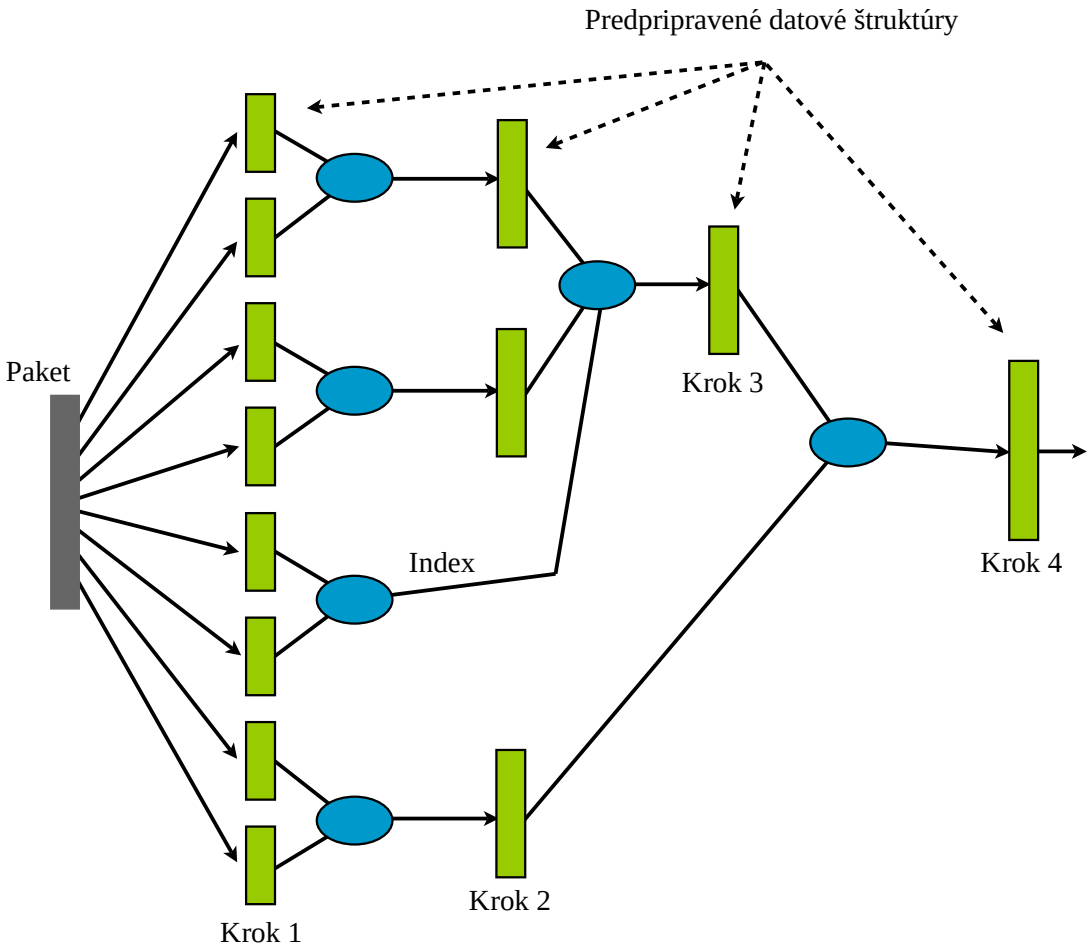
Obrázok 4.4: Crossproduct table

4.3 Heuristické

4.3.1 Recursive flow classification:

Dôvod k vytvoreniu tohto algoritmu bol prieskum reálnych klasifikátorov, ktorý poukázal na to, že množina pravidiel klasifikátorov obsahuje obrovské množstvo *redundantných* pravidiel, ktoré môžu byť využité pri výbere správneho klasifikačného postupu.

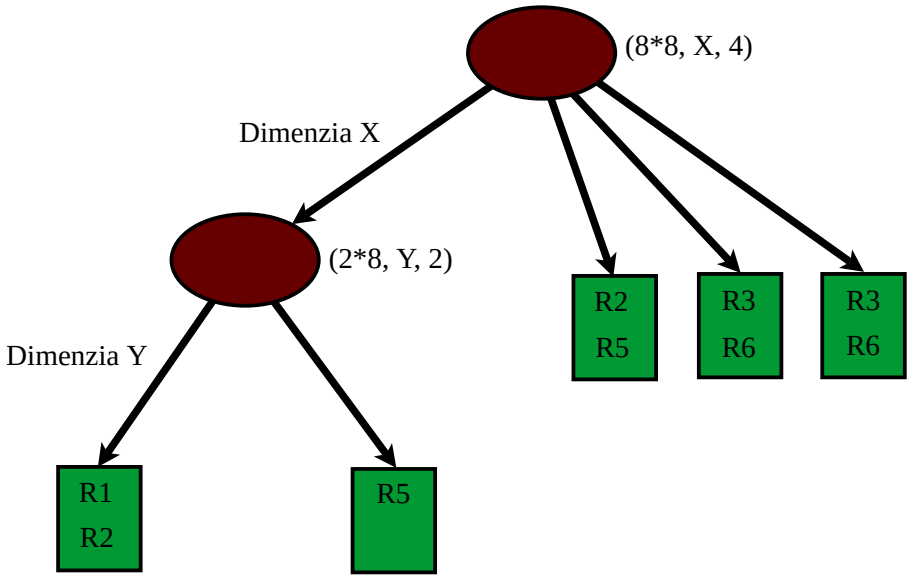
Na základe zistení prieskumu vyplynulo, že celý proces klasifikácie môže byť značne zjednodušený pri správnom návrhu algoritmu. Algoritmus RFC na klasifikáciu pozerá ako na mapovanie bitov hlavičky na odpovedajúce pravidlo. Toto mapovanie sa prevádza v niekoľkých krokoch, pričom sa množina každým krokom redukuje. V prvom kroku sú všetky sledované polia z hlavičky rozdelené do niekoľkých častí, ktoré sú potom využívané ako ukazovatele do pamäti. V nasledujúcom kroku sú potom indexy do pamäti vytvárané ako lineárne kombinácie výsledkov vyhľadávania z predchádzajúceho kroku. V poslednom kroku ostane identifikátor odpovedajúceho pravidla. [7]



Obrázok 4.5: Algoritmus RFC

4.3.2 Hierarchical Intelligent Cuttings

Algoritmus je schopný prispôsobiť svoju dátovú štruktúru konkrétnemu klasifikátoru, a teda čo najlepšie vykonávať klasifikáciu. Pri svojej činnosti využíva jednoduché heuristiky na rozdelenie prehľadávaného priestoru na niekoľko častí podľa jednotlivých dimenzií. Algoritmus vytvára rozhodovací strom, ktorého vnútorné uzly obsahujú informácie potrebné pre činnosť algoritmu a listy obsahujú určité množstvo pravidiel. Pri spracovaní paketu najprv algoritmus prejde stromom a potom zoznamom pravidiel v liste. Prehľadávanie listov je sekvenčné. Maximálny počet pravidiel v jednom liste je obmedzený. Parametre dátovej štruktúry stromu (hĺbka, tvar, spôsob prehľadávania, ...) závisia od konkrétneho klasifikátoru. V prípade n dimenzii, koreňový uzol predstavuje celý geometrický priestor, každý uzol predstavuje časť tohto priestoru. Každý uzol sa ďalej rekurzívne rozdeľuje na časti, až kým sa nesplní podmienka maximálneho počtu pravidiel v jednom liste. Na obrázku je príklad stromovej štruktúry algoritmu. [7]



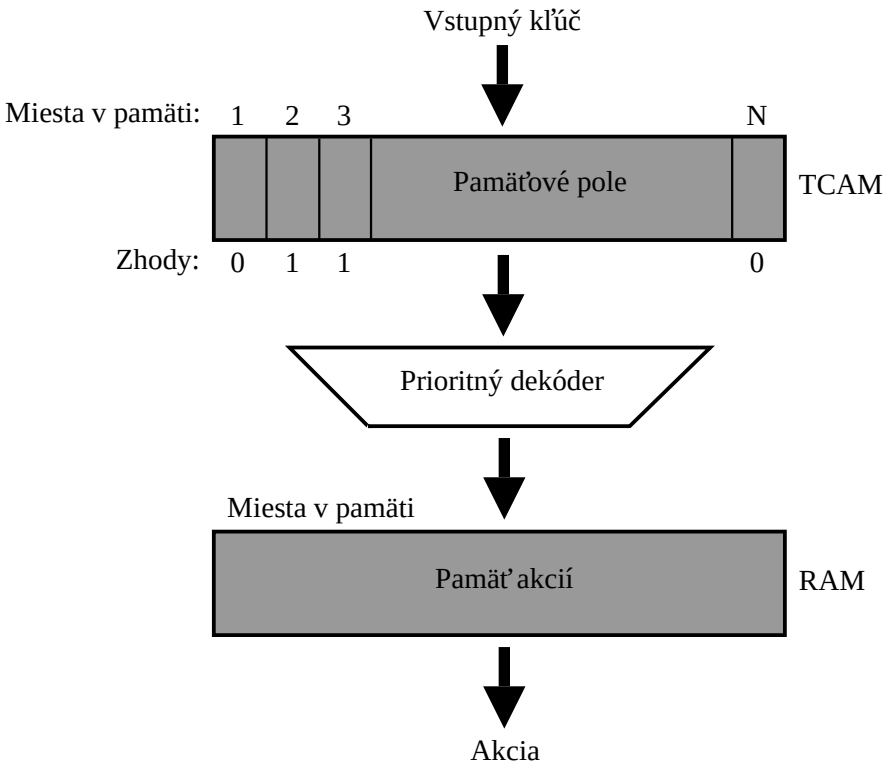
Obrázok 4.6: Algoritmus HiCuts

4.4 Hardvérové

4.4.1 Ternary CAMs

TCAM (špeciálna paralelne pracujúca pamäť) ukladá každé w-bitové pole ako pár (hodnota, maska), kde sú obe hodnoty w-bitové čísla. Napríklad ak sa $w = 5$ a prefix je $10*$, TCAM to uloží ako pár (10000, 11000). Zhoda sa vyhodnocuje tak, že sa porovnávajú vstupné prvky s hodnotou páru s použitím masky. (porovnávajú sa len tie bity kde je maska 1) Použitie TCAM je zachytené na nasledujúcom obrázku. V TCAM pamäťovom poli sú uložené pravidlá vzhľadom na prioritu. Vstupný prvok je porovnaný s každým pravidlom paralelne. Výsledkom je N-bitový vektor zhody pre všetky pravidlá. Následne N-bitový prioritný dekodér indikuje adresu s najvyššou prioritou zhody. Adresa je potom použitá

ako ukazovateľ do pamäti RAM (Random Access Memory), kde je uložená akcia pridelená tomuto prefixu. Tento algoritmus dosahuje veľmi vysokých rýchlostí. [6]



Obrázok 4.7: Algoritmus TCAMs

Kapitola 5

Klasifikácia pomocou Perfect-Hash

Tento algoritmus patrí medzi najnovšie algoritmy, pomocou ktorých je možné klasifikovať pakety. Riešenie je zamerané na rozklad problému na pod problémy a je koncipované tak, aby ho bolo možné využiť aj pri vysokých rýchlostiach dnešných sieťových technológií. Unikátnou vlastnosťou tohto algoritmu je konštantná časová zložitosť, pokiaľ ide o prístup do externej pamäti. Pri svojej činnosti algoritmus vykonáva presne dva prístupy do externej pamäti. Algoritmus pri klasifikácií využíva SRAM (Static Random Access Memory), pre uloženie potrebných dát a FPGA (Field-Programmable Gate Array), pre implementáciu algoritmu. Rýchlosť algoritmu je možné zvýšiť použitím rýchlejších SRAM pamätí.

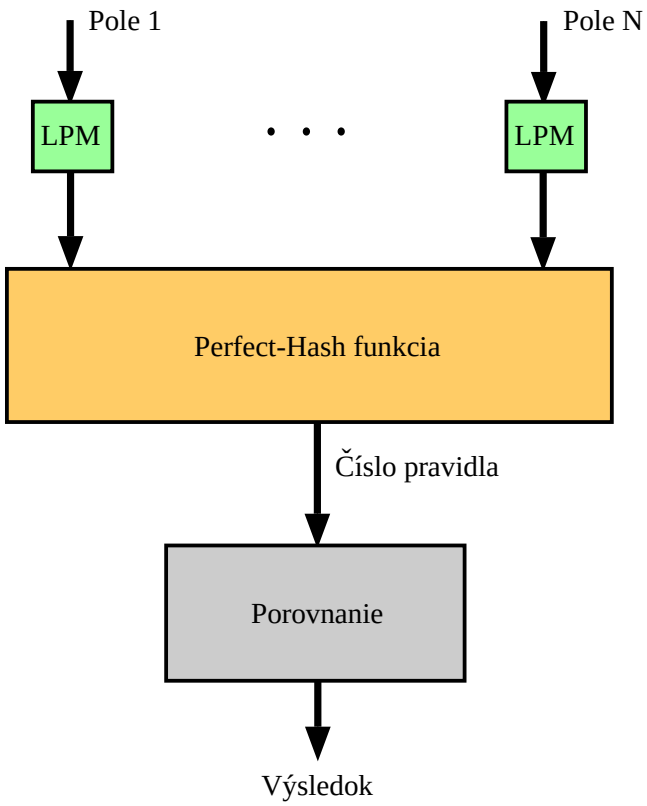
Samotná klasifikácia je rozdelená do niekoľkých krokov. Prvým je operácia LPM (Longest Prefix Match), ktorá sa vykonáva nezávisle pre všetky dimenzie. Zo zadanej sady prefixov rôznej dĺžky, funkcia nájde ten, ktorý najlepšie vyhovuje porovnávanej hodnote. Pravidlá, ktoré obsahujú rozsahy sú konvertované na prefixy. LPM je často využívaná operácia pri smerovaní paketov, kde sa však kontroluje len jedna dimenzia a tou je cieľová IP adresa. Bloková schéme klasifikácie pomocou Perfect-Hash funkcie je na obrázku 5.1.

Po operácii LPM nasleduje algoritmus vyhľadania samotného pravidla. Výhodou tejto koncepcie je, že v prípade lepšieho riešenia môžu byť jednotlivé časti vymenené nezávisle. Perfect-Hash algoritmus má výhodu v konštantnej časovej zložitosti a vo využití *off-chip* pamäte na uloženie tabuľky. V prípade, že sa paket nezhoduje so žiadnym pravidlom hashovacia funkcia vráti číslo ľubovoľného pravidla. Preto sa v poslednom kroku pomocou komparátora kontroluje zhoda paketu s vráteným pravidlom hashovacej funkcie. Algoritmus pre prípravu dátových štruktúr (implementovaný v Pythone), ktoré sa pri klasifikácii využívajú, je optimalizovaný pre použitie v hardwari. Z tohoto dôvodu sú pravidlá, ktoré generujú najviac pseudopravidiel a teda zaberajú najväčšie miesto v hash tabuľke odstránené. Tieto pravidlá sa označujú ako *spoilers*. V prípade, že paket neodpovedá žiadnemu pravidlu uloženému v hash tabuľke, je nutné lineárne prehľadať zoznam s odstránenými pravidlami. Tabuľka pravidiel je uložená na *on-chip* pamäti, tá ma však obmedzenú kapacitu, preto je nutné použiť metódu kompresie a uložiť tabuľku čo možno najefektívnejšie. Tabuľka pravidiel obsahuje iba indexy do tabuliek prefixov. Čísla portov sú uložené priamo v tabuľke pravidiel, pretože sa jedná o hodnoty, ktoré nezaberajú veľkú časť pamäte. Tým sa ušetrí veľkosť pamäte minimálne o polovicu. Pri tejto schéme (obrázok 5.2) je tiež možné využiť hardvérového paralelizmu, pretože všetky tabuľky prefixov je možné kontrolovať súčasne. [8]

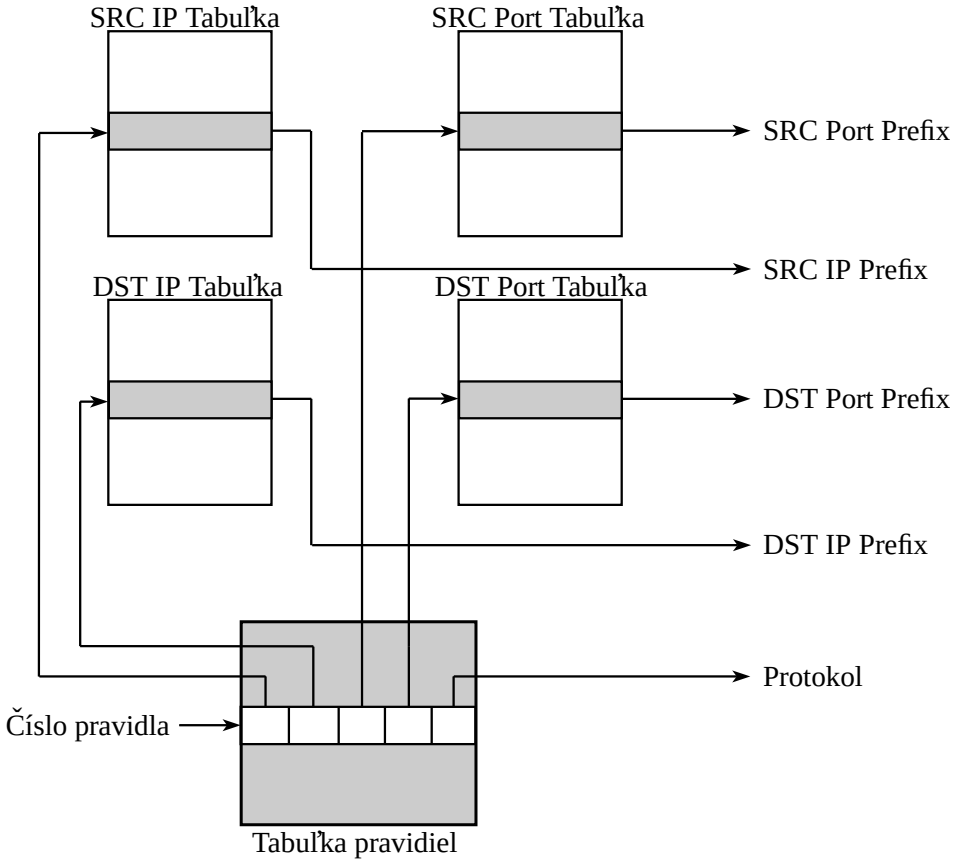
5.1 Perfect-Hash Funkcia

Perfect-Hash konštrukčný algoritmus vytvára acyklické grafy, kde hrany predstavujú kľúče a vrcholy sú výsledkami dvoch rôznych hashovacích funkcií. Vrcholom sú priradené hodnoty tak, aby ich súčet odpovedal požadovanej hodnote hashu. Algoritmus pozostáva z nasledujúcich krokov: [8]

1. Vstup: K kľúče, každý asociovaný s číslom čo bude hashované
2. Vytvorenie grafu s $N = cK$ uzlami, kde $c > 1$
3. Výber dvoch rôznych bežných hash funkcií s výstupnými hodnotami $0..N–1$
4. Pre každý kľúč spočítať hodnotu predchádzajúcich hash funkcií (h_1, h_2), nakresliť hranu medzi vrcholmi h_1 a h_2 grafu a priradiť jej požadovanú hash hodnotu.
5. Kontrola či je graf acyklický. V prípade, že nie zvýšiť c a vrátiť sa ku kroku 2
6. Priradiť hodnoty každému uzlu tak, že pre každú hranu môžete sčítať hodnoty oboch vrcholov a dostanete požadovanú hodnotu hrany. Toto sa vykonáva algoritmom prehľadávania do hĺbky pretože graf je acyklický.
7. Hash funkcie f_1 a f_2 z bodu 3 spolu s hodnotami vrcholov vytvárajú požadovanú Perfect-Hash funkciu.



Obrázok 5.1: Bloková schéma klasifikácie pomocou Perfect-Hash



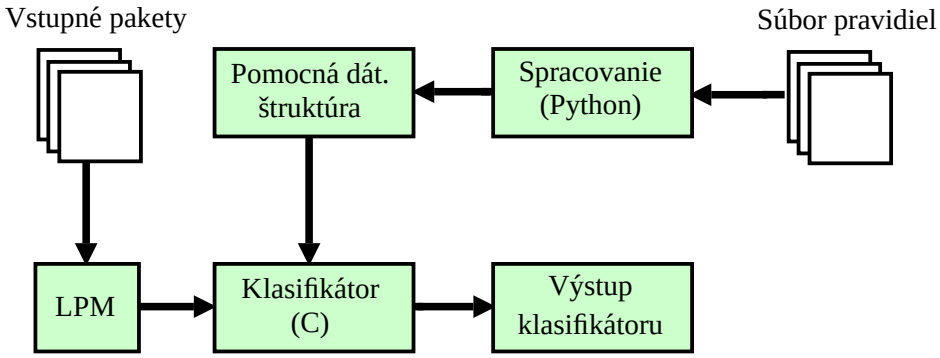
Obrázok 5.2: Schéma spracovania

Kapitola 6

Návrh implementácie

Implementovaný algoritmus využíva pri klasifikácií už spomínanú Perfect-Hash funkciu. Implementácia sa zameriava len na samotnú klasifikáciu paketov preto je vhodne zjednodušená o niektoré prvky, ktoré sa v plnohodnotných paketových filtroch vyskytujú.

Celkový návrh sa dá rozdeliť na niekoľko menších častí. Prvou časťou je modul LPM. V jednej z verzií klasifikátoru bol tento modul prevzatý z bakalárskej práce študenta, na ktorú táto práca nadväzuje. Ďalšou časťou je vytvorenie pomocných štruktúr, ktoré budú použité pri samotnej klasifikácií. Táto časť je implementovaná pomocou skriptovacieho jazyka Python. Jazyk Python je v porovnaní s jazykom C pomalší, ale vzhľadom k tomu, že štruktúry sa pripravia jednorázovo podľa zadaných pravidiel pred samotnou klasifikáciou je jeho použitie pre túto časť klasifikátoru výhodné. Samotný klasifikátor, ktorý je implementovaný pomocou jazyka C, tieto štruktúry následne využíva a na základe hodnôt Perfect-Hash funkcie vracia číslo hľadaného pravidla. Algoritmus sa zameriava na rýchlosť klasifikácie a neuvažuje teda následnú akciu spracovania paketu podľa príslušného pravidla. Celkový priebeh klasifikácie znázorňuje nasledujúci obrázok:



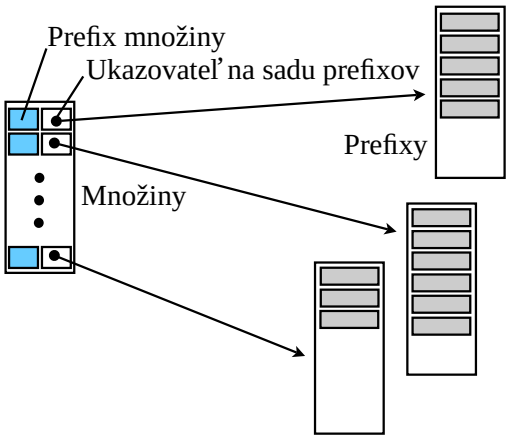
Obrázok 6.1: Blokový diagram algoritmu

Ako už bolo spomenuté vyššie klasifikátor bol implementovaný v niekoľkých verziách. V každej vyššej verzii bola snaha o zvýšenie rýchlosti klasifikácie. Zvýšenie rýchlosti sa dosahovalo

úpravou jednotlivých modulov. Upravované boli moduly LPM (modul vykonávajúci LPM) a PHCA (modul vykonávajúci Perfect-Hash).

Klasifikátor v1.0 obsahuje jednoduchý lineárny LPM modul a jednoduchý PHCA modul. LPM modul pri vyhľadávaní zhodného prefixu prechádza jednotlivé prefixy lineárne. Čas pre vyhľadávanie je teda závislý na počte vstupných prefixov. PHCA modul spočíta na základe kľúča, ktorý je výstupom LPM modulu, hodnoty dvoch hashov. Následne získa číslo odpovedajúceho pravidla a vráti výsledok. Na záver skontroluje, či výsledné pravidlo odpovedá vstupnému paketu. Toto je prvotná implementácia, z ktorej vychádzajú všetky ostatné.

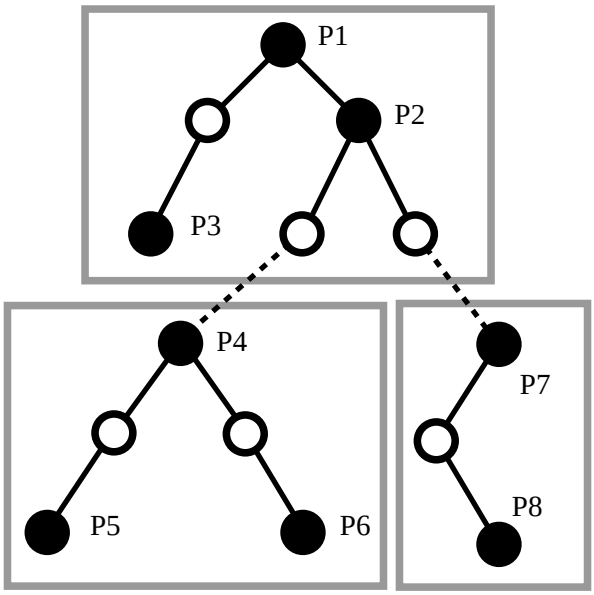
Klasifikátor v2.0 má modul LPM tvorený 2-úrovňovou stromovou štruktúrou (obrázok 6.2), ktorá umožňuje zrýchlenie oproti v1.0 až o 2-násobok a v určitých prípadoch aj viac. Táto rýchlosť je však závislá na tvare vstupných prefixov. Modul rozdelí IP prefixy do množín podľa prvých 16 bitov. V prípade prefixov kratších ako 16 bitov sa vytvoria množiny, ktoré odpovedajú dĺžke prefixov (ak má prefix dĺžku 3, vytvorí sa množina pre dĺžku 3). Vyhľadávanie zhody prefixu následne pozostáva z 2 krokov. V prvom kroku algoritmus vyberie množinu, do ktorej vstupný prefix patrí. Tento výber je realizovaný lineárnym prehľadávaním všetkých množín. V prípade, že bola nájdená odpovedajúca množina, v druhom kroku algoritmus v tejto množine lineárne vyhľadáva odpovedajúci prefix. Týmto spôsobom je možné pri vyhľadávaní vynechať všetky prefixy, ktoré patria do určitej množiny, ak vstupný prefix do tejto množiny nepatrí.



Obrázok 6.2: LPM stromová štruktúra

Klasifikátor v3.0 využíva v module LPM algoritmus Tree Bitmap (modul prevzatý od [9]). TBM (Tree Bitmap) vychádza z konceptu binárnej Trie. Príklad stromovej štruktúry TBM je na obrázku 6.3. Tento koncept sa snaží zefektívniť tým, že neprechádza a nereprezentuje Triu po jednom uzle, ale tieto uzly spája a vytvára tzv. multi-uzly. Každý z nich pritom reprezentuje tvarovo rovnaký podstrom, konkrétne úplný binárny strom s danou výškou (*stride*). Každý uzol je reprezentovaný pomocou 2 bitmáp. Jedna z nich reprezentuje vnútorné uzly (interná) a druhá reprezentuje všetky existujúce hrany k následníkom daného uzlu (externá). Veľkosť bitmáp je 2^S, kde S je výška binárneho stromu *stride*. Algoritmus je potom pri vyhľadávaní schopný prechádzať stromom po viacerých poschodiach naraz, čo výrazne znižuje počet prístupov do pamäte a réžiu s tým spojenú. [9]

Klasifikátor v4.0 má rovnaký LPM modul ako verzia 3.0, ale PHCA modul je vláknový. Využíva dve vlákna, ktoré klasifikujú vstupné pakety paralelne. Každé vlákno vykonáva úplnú klasifikáciu paketov (načítanie paketu, LPM modul, PHCA modul, spracovanie výsledku). Týmto spôsobom sa dosiahne takmer 2-násobné zrýchlenie oproti verzii 3.0.



Obrázok 6.3: Stromová štrukúra TBM

6.1 Testovanie správnosti implementácie

Toto testovanie malo za úlohu dokázať, že implementovaný algoritmus funguje správne a teda je schopný klasifikovať vstupné pakety podľa zadaných pravidiel. Testovanie spočívalo v porovnávaní výsledkov jednoduchého lineárneho klasifikátoru s výsledkami implementovaného Perfect-Hash klasifikátoru. Bolo testovaných niekoľko vstupných súborov pre rôzne sady pravidiel. Ako vstupné súbory s paketmi boli použité aj súbory generované nástrojom *Ostinato* [10], ktorý slúži pre generovanie a analýzu paketov. Pakety boli generované tak, aby presne vyhovovali zadaným pravidlám (súbory *Test.pcap* a *Test.rul*).

Kapitola 7

Meranie výkonnosti

Meranie výkonnosti algoritmu PHCA bolo hlavným účelom tejto bakalárskej práce. Výkonnosť bola meraná pri všetkých implementovaných verziách klasifikátoru za použitia rovnakých vstupných súborov s paketmi a pravidlami.

7.1 Princíp merania

Pri meraní sa počas klasifikácie zaznamenávajú jednotlivé časy modulu LPM a modulu PHCA. Meranie sa teda zameriava len na rýchlosť klasifikácie a jej modulov. Počiatočná inicializácia klasifikátoru (načítanie pomocných dátových štruktúr potrebných pre klasifikáciu) sa pri meraní do výsledkov nezahŕňa. Tiež sa zanedbáva čas, ktorý trvá načítanie paketu (priemerne $5\,\mu s$). Celkový čas potrebný pre klasifikáciu sa potom vyhodnotí ako súčet časov jednotlivých modulov. Priemerné časy sú počítané ako podiel celkového času a počtu klasifikovaných paketov. Pre získavanie časových údajov je použitá funkcia `clock_gettime()`. Ako vstupné pakety sú použité pakety uložené v *.pcap* súboroch.

V prípade klasifikátoru v4.0 je meranie výkonnosti vzhľadom k použitiu vlákien mierne odlišné. Pre zjednodušenie sa meria iba celkový čas klasifikácie všetkých paketov. Tento čas zahŕňa aj čas potrebný pre načítanie paketov (rozdiel oproti predchádzajúcim verziám). Priemerný čas klasifikácie jedného paketu sa následne získa ako celkový čas klasifikácie lomeno počet klasifikovaných paketov. Pri porovnávaní s predchádzajúcimi verziami sa ešte odpočíta čas, ktorý trvá priemerné načítanie daného počtu paketov (tieto hodnoty boli merané pre zadané vstupné súbory).

7.2 Zostava merania

Všetky implementované verzie klasifikátoru boli testované s použitím rovnakých vstupných súborov s paketmi aj pravidlami.

100 paketov:
súbor so vstupnými paketmi: *example.com-1.pcap*

1 000 paketov:
súbor so vstupnými paketmi: *example.com-1.pcap*

10 000 paketov:
súbor so vstupnými paketmi: *example.com-2.pcap*

100 000 paketov:
súbor so vstupnými paketmi: *example.com-8-overnight.pcap*

1 000 000 paketov:
súbor so vstupnými paketmi: *A.pcap*

Ako súbory s pravidlami boli pri testovaní použité súbory: *100.rul, 250.rul, 500.rul, 900.rul.* Počet pravidiel obsiahnutých v jednotlivých súboroch odpovedá ich názvom. Parametre jednotlivých počítačov, na ktorých bolo uskutočnené meranie obsahuje tabuľka 7.1.

Počítač	**Procesor**	**RAM**
PC1	Intel®Core^{TM} i3 - 370M - 2,4 GHz, 3 M Cache	1,5 GB
PC2	Intel®Core^{TM} 2 Duo - P8400 - 2,26 GHz, 3 M Cache	3 GB
Merlin	Quad-Core AMD Opteron^{TM} Processor 2387 - 2,8 GHz, 2 M Cache	16 GB

Tabuľka 7.1: Parametre počítačov

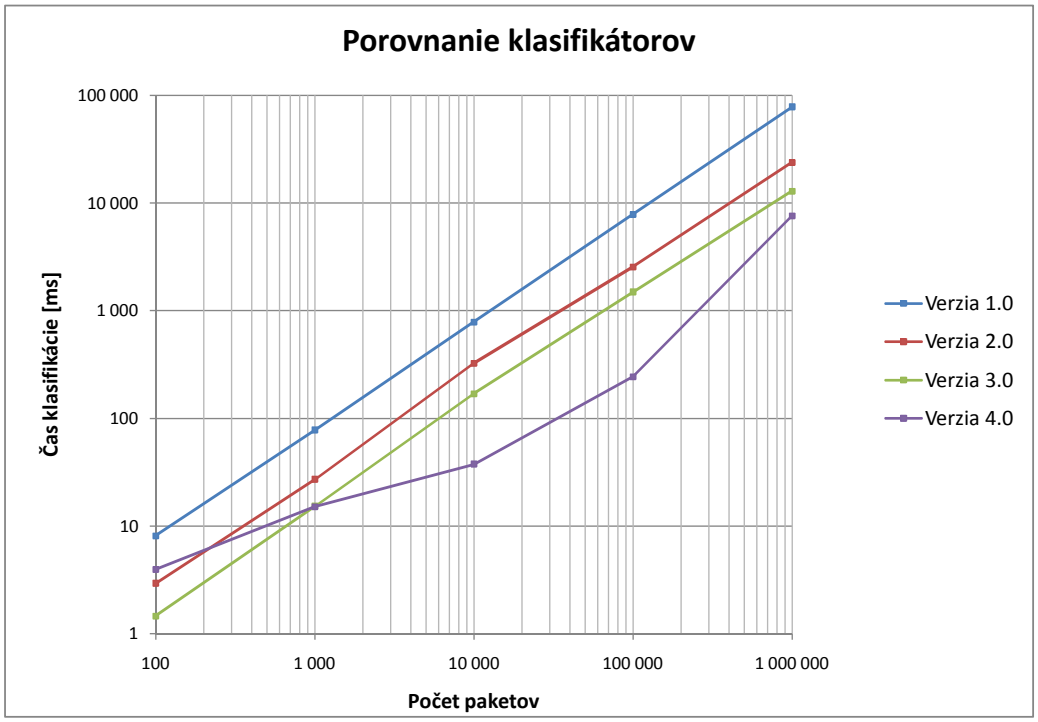
7.3 Výsledky merania

Výsledky merania pre každú implementovanú verziu klasifikátoru sú zaznamenané v tabuľkách (obrázky 7.5, 7.7, 7.9, 7.11) a graficky znázornené na grafoch (obrázky 7.6, 7.8, 7.10, 7.12). Stručné porovnanie rýchlosti jednotlivých verzií zobrazujú grafy na obrázkoch 7.1 a 7.2. V grafoch sú výsledky klasifikácie pre 900 pravidiel. Postupné zrýchlovanie implementovaného klasifikátora je na grafoch jasne viditeľné. Pri klasifikátore v4.0 sú pre porovnanie odpočítané časy potrebné pre načítanie vstupných paketov. Tieto výsledky boli namerané na počítači PC1 z tabuľky 7.1 (meranie na PC1 prebiehalo na virtuálnom stroji).

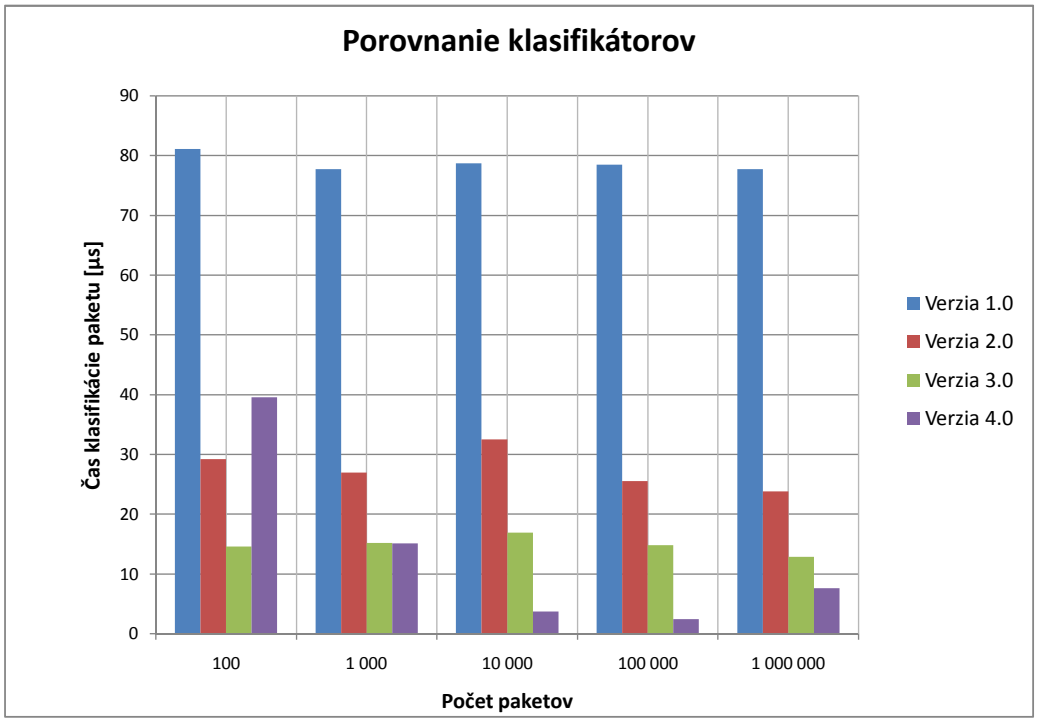
Pre porovnanie bolo meranie spustené aj na viacerých počítačoch s rôznymi výkonovými parametrami (tabuľka 7.1). Výsledky tohoto merania zobrazujú grafy na obrázkoch 7.3 a 7.4. Z týchto výsledkov vyplýva, že rýchlosť klasifikácie a teda aj výsledky merania sú závislé od výkonu počítača, na ktorom je klasifikátor spustený. Tento výsledok je však pochopiteľný, pretože vyšší výkon poskytuje vyššiu rýchlosť.

Graf na obrázku 7.3 zobrazuje porovnanie merania na rôznych počítačoch pri klasifikátore v3.0. Toto meranie malo za úlohu porovnať vplyv výkonu počítača na rýchlosť klasifikácie.

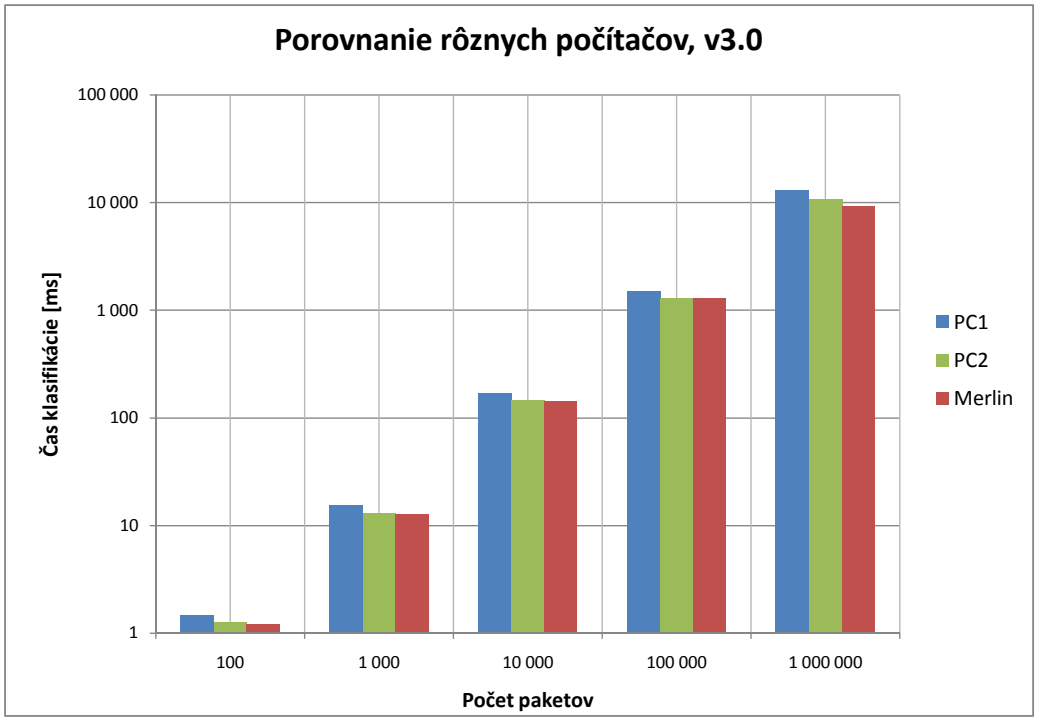
Graf na obrázku 7.4 zobrazuje porovnanie merania na rôznych počítačoch pri klasifikátore v4.0. Toto meranie malo za úlohu porovnať výkon jednotlivých procesorov pri vláknovom spracovaní. Pri tomto meraní nebol použitý počítač PC1, pretože na virtuálnom stroji je efektivita vláknového spracovania veľmi nízka.



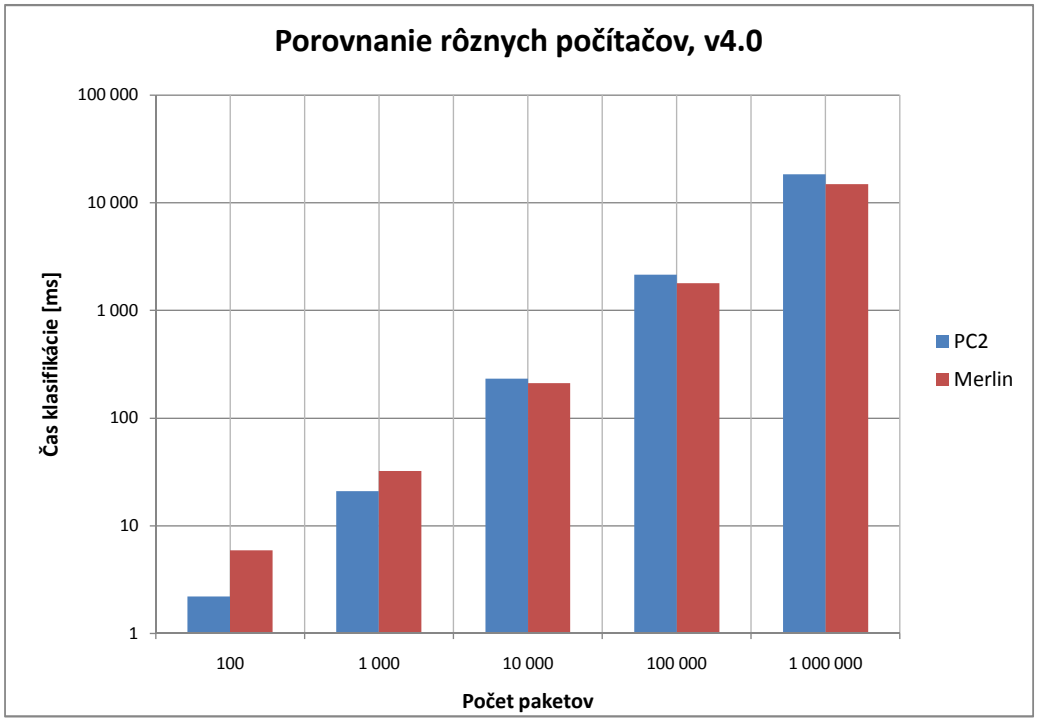
Obrázok 7.1: Grafické porovnanie časov klasifikácie jednotlivých verzií



Obrázok 7.2: Grafické porovnanie rýchlosti klasifikácie paketu jednotlivých verzií



Obrázok 7.3: Grafické porovnanie rýchlosti klasifikácie na rôznych počítačoch



Obrázok 7.4: Grafické porovnanie rýchlosti klasifikácie na rôznych počítačoch

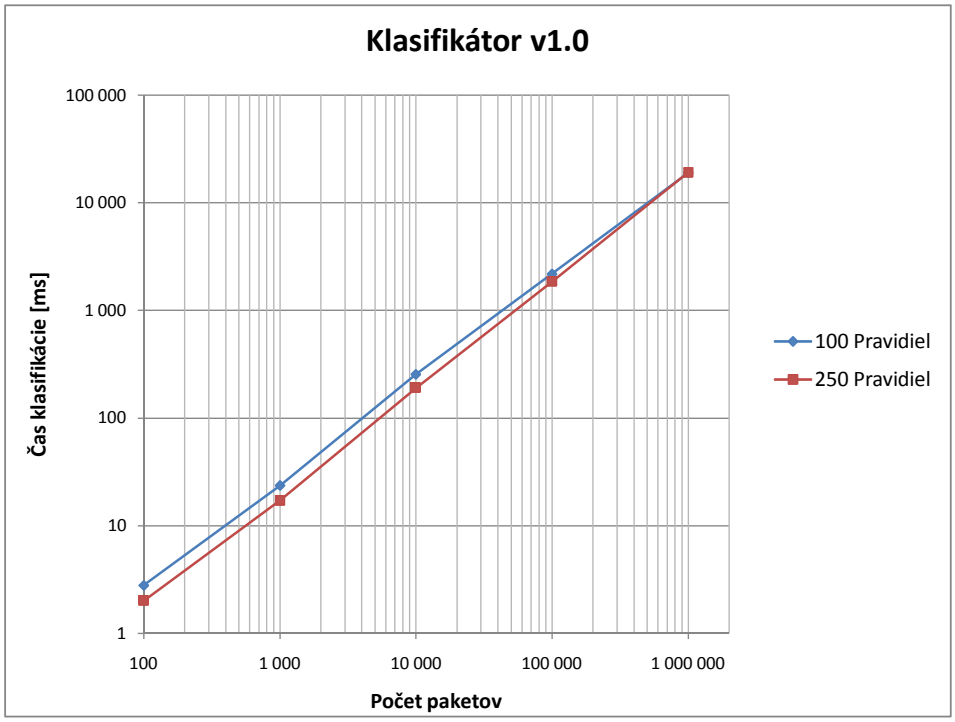
PHCA v1.0 - 100 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	1.929605ms	15.609865 ms	182.438537 ms	1.490382 s	14.517032 s
Čas PHCA	832.003 µs	7.991711 ms	72.955520 ms	690.873061 ms	4.454476 s
Čas Klasifikácie	2.799877 ms	23.601576 ms	255.394057 ms	2.181255 s	18.971508 s
LPM Priemer	19.678740 µs	15.609865 µs	18.243854 µs	14.903821 µs	14.517032 µs
PHCA Priemer	8.320030 µs	7.991711 µs	7.295552 µs	6.908731 µs	4.454476 µs
Klasifikácia paketu	27.998770 µs	23.601576 µs	25.539406 µs	21.812551 µs	18.971508 µs

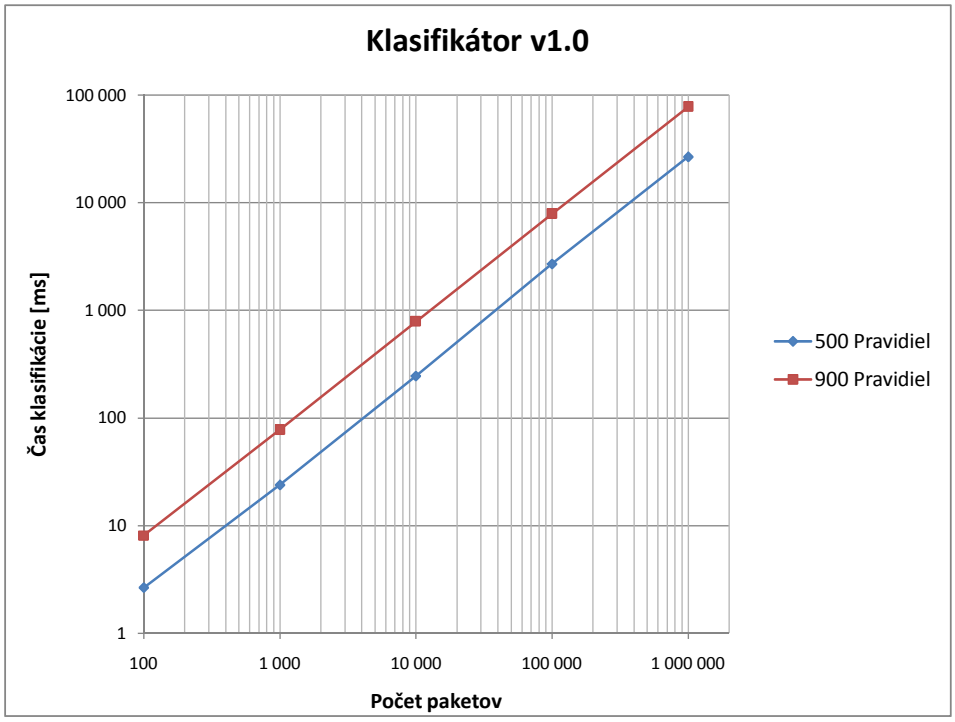
PHCA v1.0 - 250 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	1.906507 ms	16.138385 ms	179.925035 ms	1.745556 s	17.953606 s
Čas PHCA	106.761000 µs	974.501000 µs	10.782657 ms	100.564581 ms	1.098795 s
Čas Klasifikácie	2.013268 ms	17.112886 ms	190.707692 ms	1.846120 s	19.052401 s
LPM Priemer	19.065070 µs	16.138385 µs	17.992503 µs	17.455557 µs	17.953606 µs
PHCA Priemer	1.067610 µs	975 ns	1.078266 µs	1.005646 µs	1.098795 µs
Klasifikácia paketu	20.132680 µs	17.112886 µs	19.070769 µs	18.461203 µs	19.052401 µs

PHCA v1.0 - 500 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	2.530820 ms	22.672850 ms	234.943652 ms	2.588652 s	25.603980 s
Čas PHCA	121.021000 µs	1.252872 ms	11.136002 ms	115.461329 ms	1.041460 s
Čas Klasifikácie	2.651841 ms	23.925722 ms	246.079654 ms	2.704113 s	26.645440 s
LPM Priemer	25.308200 µs	22.672850 µs	23.494365 µs	25.886520 µs	25.603980 µs
PHCA Priemer	1.210210 µs	1.252872 µs	1.113600 µs	1.154613 µs	1.041460 µs
Klasifikácia paketu	26.518410 µs	23.925722 µs	24.607965 µs	27.041133 µs	26.645440 µs

PHCA v1.0 - 900 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	7.345902 ms	69.018932 ms	689.313917 ms	7.038801 s	72.612941 s
Čas PHCA	759.625000 µs	8.658405 ms	97.600363 ms	807.533080 ms	5.077415 s
Čas Klasifikácie	8.105527 ms	77.677337 ms	786.914280 ms	7.846334 s	77.690356 s
LPM Priemer	73.459020 µs	69.018932 µs	68.931392 µs	70.388007 µs	72.612941 µs
PHCA Priemer	7.596250 µs	8.658405 µs	9.760036 µs	8.075331 µs	5.077415 µs
Klasifikácia paketu	81.055270 µs	77.677337 µs	78.691428 µs	78.463337 µs	77.690356 µs

Obrázok 7.5: Tabuľky nameraných hodnôt verzie 1.0





Obrázok 7.6: Grafické porovnanie výsledkov verzie 1.0

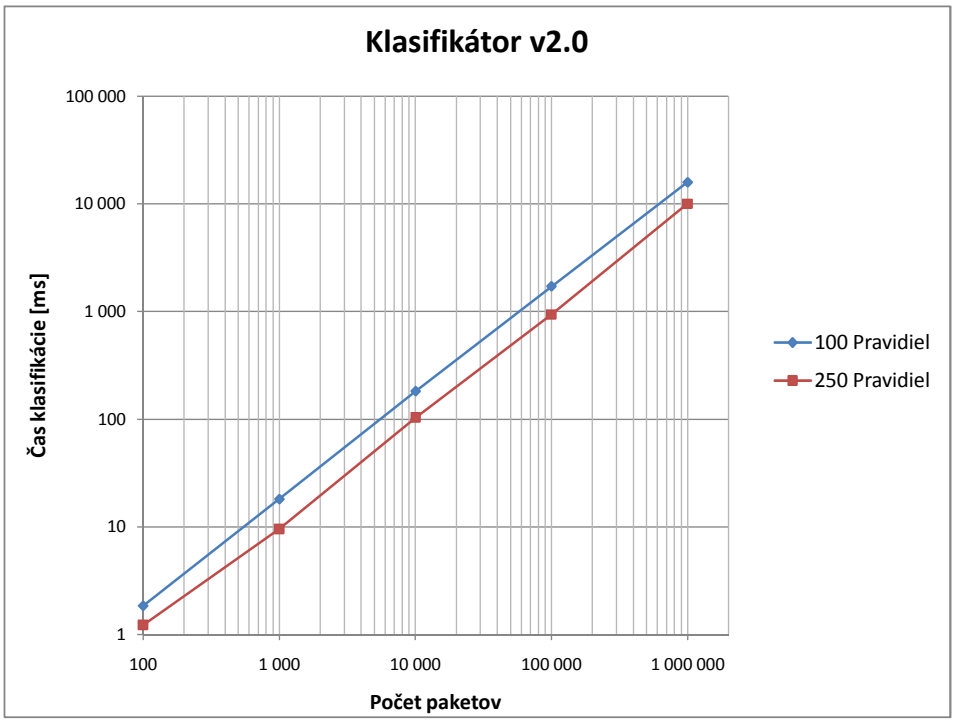
PHCA v2.0 - 100 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	1.113889 ms	10.855869 ms	111.313962 ms	1.050815 s	11.433086 s
Čas PHCA	738.362000 µs	7.203171 ms	70.656468 ms	657.879057 ms	4.484333 s
Čas Klasifikácie	1.852251 ms	18.059040 ms	181.970430 ms	1.708694 s	15.917419 s
LPM Priemer	11.138890 µs	10.855869 µs	11.131396 µs	10.508151 µs	11.433086 µs
PHCA Priemer	7.383620 µs	7.203171 µs	7.065647 µs	6.578791 µs	4.484333 µs
Klasifikácia paketu	18.522510 µs	18.059040 µs	18.197043 µs	17.086941 µs	15.917419 µs

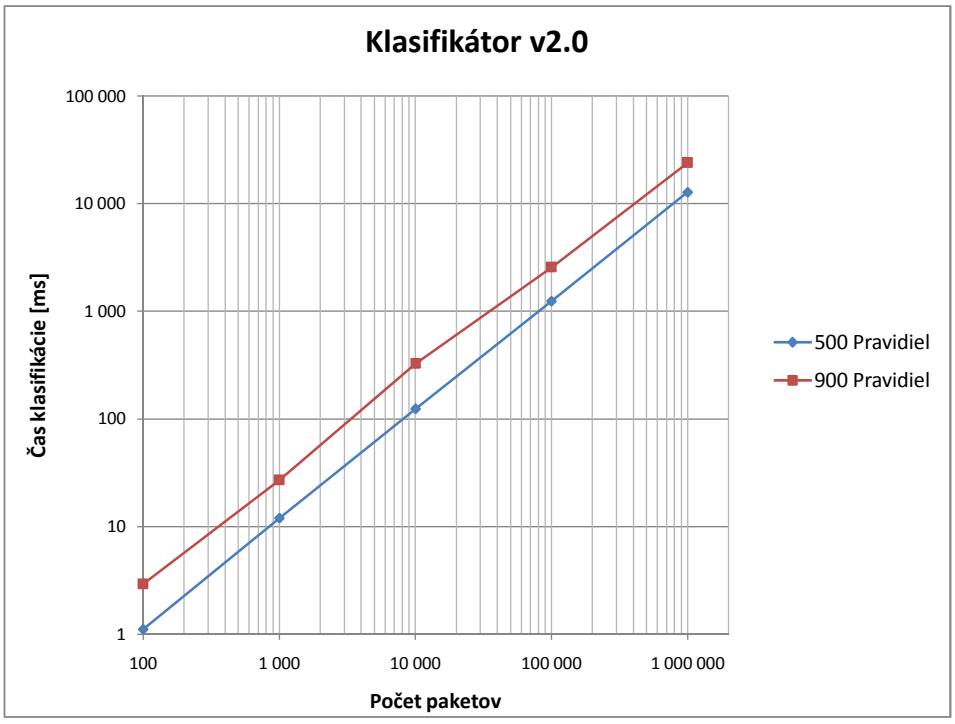
PHCA v2.0 - 250 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	1.050248 ms	8.121169 ms	87.844363 ms	849.014577 ms	8.870608 s
Čas PHCA	171.734000 µs	1.395972 ms	15.349514 ms	92.583000 ms	1.183421 s
Čas Klasifikácie	1.221982 ms	9.517141 ms	103.193877 ms	941.597577 ms	10.054029 s
LPM Priemer	10.502480 µs	8.121169 µs	8.784436 µs	8.490146 µs	8.870608 µs
PHCA Priemer	1.717340 µs	1.395972 µs	1.534951 µs	926 ns	1.183421 µs
Klasifikácia paketu	12.219820 µs	9.517141 µs	10.319388 µs	9.415976 µs	10.054029 µs

PHCA v2.0 - 500 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	1.022463 ms	10.851223 ms	112.963371 ms	1.148248 s	11.878499 s
Čas PHCA	89.243000 µs	1.071271 ms	10.464880 ms	91.508962 ms	893.018756 ms
Čas Klasifikácie	1.111706 ms	11.922494 ms	123.428251 ms	1.239757 s	12.771518 s
LPM Priemer	10.224630 µs	10.851223 µs	11.296337 µs	11.482481 µs	11.878499 µs
PHCA Priemer	892 ns	1.071271 µs	1.046488 µs	915 ns	893 ns
Klasifikácia paketu	11.117060 µs	11.922494 µs	12.342825 µs	12.397570 µs	12.771518 µs

PHCA v2.0 - 900 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	2.107177 ms	18.005399 ms	216.196676 ms	1.780062 s	18.538999 s
Čas PHCA	815.554000 µs	8.990432 ms	109.268568 ms	777.474793 ms	5.301054 s
Čas Klasifikácie	2.922731 ms	26.995831 ms	325.465244 ms	2.557537 s	23.840053 s
LPM Priemer	21.071770 µs	18.005399 µs	21.619668 µs	17.800619 µs	18.538999 µs
PHCA Priemer	8.155540 µs	8.990432 µs	10.926857 µs	7.774748 µs	5.301054 µs
Klasifikácia paketu	29.227310 µs	26.995831 µs	32.546524 µs	25.575367 µs	23.840053 µs

Obrázok 7.7: Tabuľky nameraných hodnôt verzie 2.0





Obrázok 7.8: Grafické porovnanie výsledkov verzie 2.0

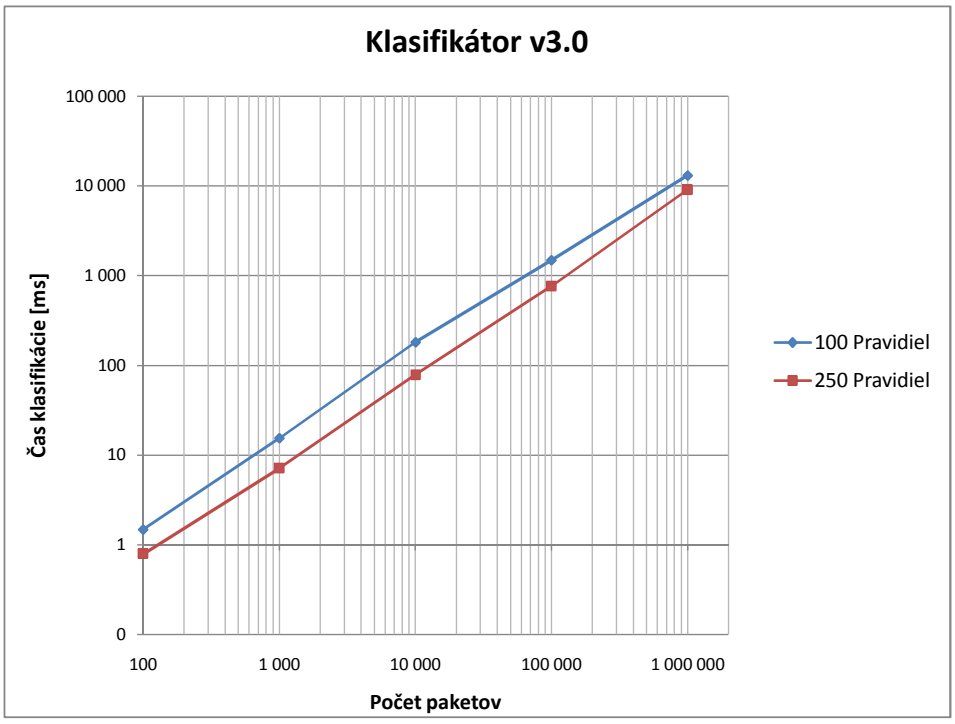
PHCA v3.0 - 100 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	687.133000 µs	6.499264 ms	79.035485 ms	683.131202 ms	7.899076 s
Čas PHCA	788.955000 µs	8.862098 ms	102.243596 ms	806.986755 ms	5.168662 s
Čas Klasifikácie	1.476088 ms	15.361362 ms	181.279081 ms	1.490118 s	13.067738 s
LPM Priemer	6.871330 µs	6.499264 µs	7.903549 µs	6.831312 µs	7.899076 µs
PHCA Priemer	7.889550 µs	8.862098 µs	10.224360 µs	8.069868 µs	5.168662 µs
Klasifikácia paketu	14.760880 µs	15.361362 µs	18.127908 µs	14.901180 µs	13.067738 µs

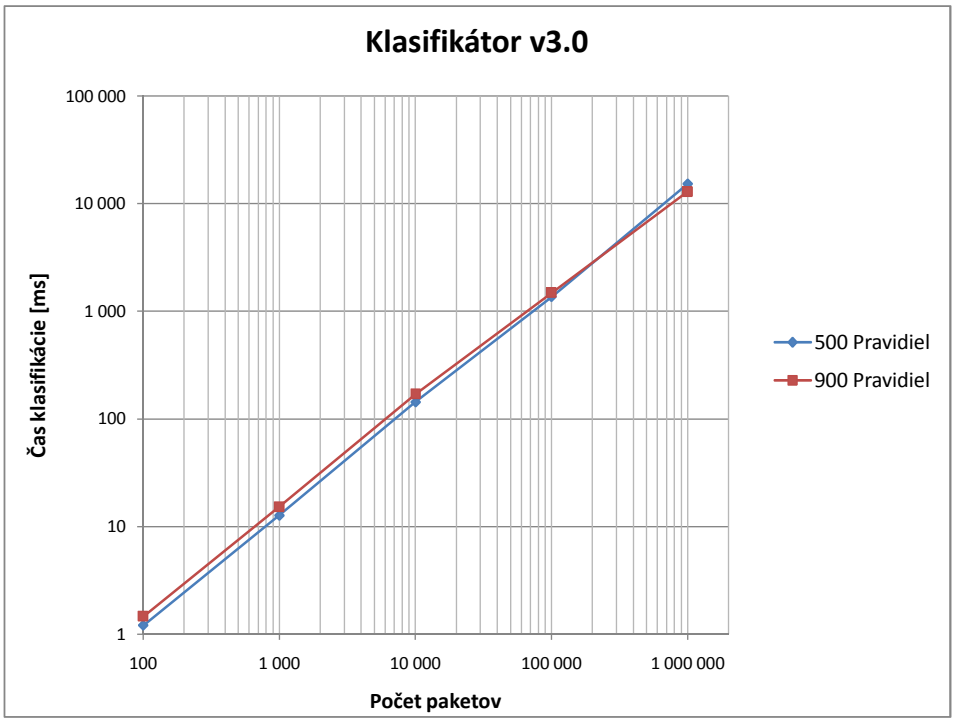
PHCA v3.0 - 250 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	638.303000 µs	5.930598 ms	67.538367 ms	655.769059 ms	7.781486 s
Čas PHCA	148.669000 µs	1.155366 ms	10.663117 ms	106.897676 ms	1.244713 s
Čas Klasifikácie	786.972000 µs	7.085964 ms	78.201484 ms	762.666735 ms	9.026199 s
LPM Priemer	6.383030 µs	5.930598 µs	6.753837 µs	6.557691 µs	7.781486 µs
PHCA Priemer	1.486690 µs	1.155366 µs	1.066312 µs	1.068977 µs	1.244713 µs
Klasifikácia paketu	7.869720 µs	7.085964 µs	7.820148 µs	7.626667 µs	9.026199 µs

PHCA v3.0 - 500 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	1.068809 ms	11.194648 ms	132.445698 ms	1.234493 s	14.081204 s
Čas PHCA	134.992000 µs	1.487078 ms	11.423248 ms	125.556651 ms	1.146617 s
Čas Klasifikácie	1.203801 ms	12.681726 ms	143.868946 ms	1.360050 s	15.227821 s
LPM Priemer	10.688090 µs	11.194648 µs	13.244570 µs	12.344931 µs	14.081204 µs
PHCA Priemer	1.349920 µs	1.487078 µs	1.142325 µs	1.255567 µs	1.146617 µs
Klasifikácia paketu	12.038010 µs	12.681726 µs	14.386895 µs	13.600497 µs	15.227821 µs

PHCA v3.0 - 900 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas LPM	651.626000 µs	5.992946 ms	69.622432 ms	665.580529 ms	7.662723 s
Čas PHCA	809.496000 µs	9.241165 ms	100.054009 ms	817.699667 ms	5.213689 s
Čas Klasifikácie	1.461122 ms	15.234111 ms	169.676441 ms	1.483280 s	12.876412 s
LPM Priemer	6.516260 µs	5.992946 µs	6.962243 µs	6.655805 µs	7.662723 µs
PHCA Priemer	8.094960 µs	9.241165 µs	10.005401 µs	8.176997 µs	5.213689 µs
Klasifikácia paketu	14.611220 µs	15.234111 µs	16.967644 µs	14.832802 µs	12.876412 µs

Obrázok 7.9: Tabuľky nameraných hodnôt verzie 3.0





Obrázok 7.10: Grafické porovnanie výsledkov verzie 3.0

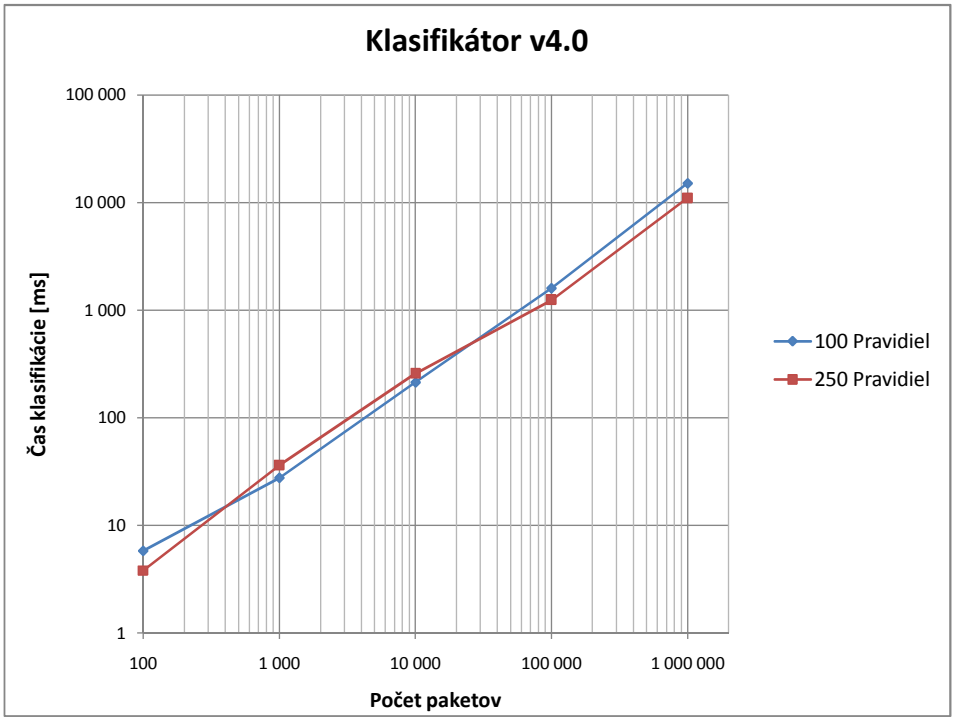
PHCA v4.0 - 100 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas Klasifikácie	5.807155 ms	27.627808 ms	213.833021 ms	1.602885 s	15.073455 s
Klasifikácia paketu	58.07155 µs	27.627808 µs	21.3833 µs	16.0288 µs	15.0734 µs

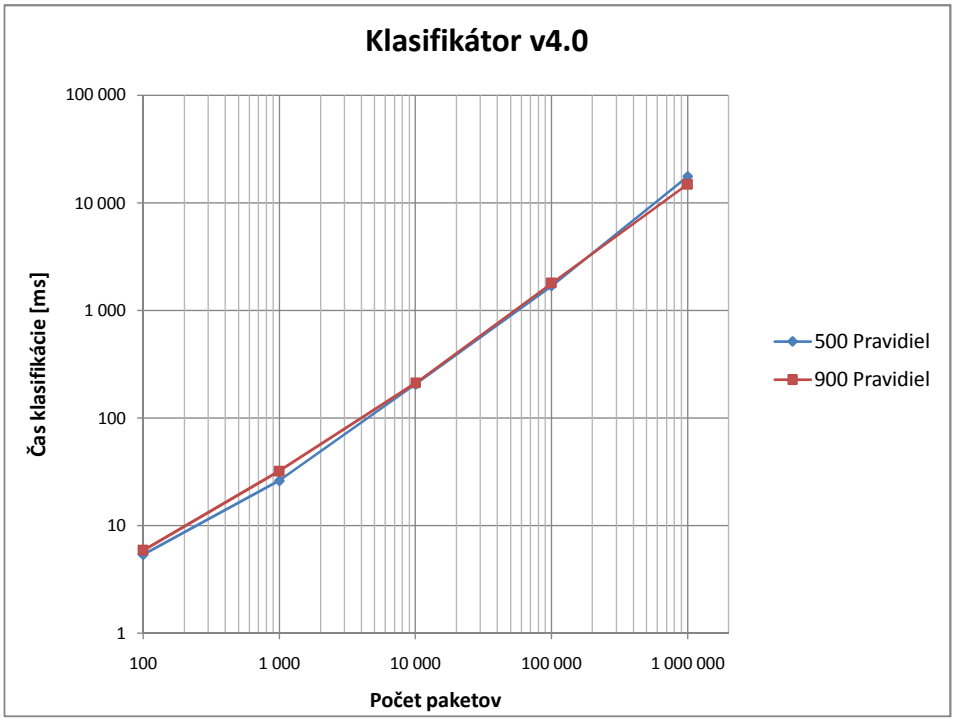
PHCA v4.0 - 250 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas Klasifikácie	3.822510 ms	36.139151 ms	258.226693 ms	1.244909 s	10.989466 s
Klasifikácia paketu	38.2251 µs	36.139152 µs	25.8226 µs	12.44909 µs	10.989466 µs

PHCA v4.0 - 500 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas Klasifikácie	5.396040 ms	26.168403 ms	205.814300 ms	1.685552 s	17.481745 s
Klasifikácia paketu	53.9604 µs	26.16840 µs	20.58143 µs	16.85552 µs	17.481745 µs

PHCA v4.0 - 900 Pravidiel					
Počet paketov	**100**	**1 000**	**10 000**	**100 000**	**1 000 000**
Čas Klasifikácie	5.904502 ms	32.040812 ms	210.723464 ms	1.785990 s	14.804730 s
Klasifikácia paketu	59.04502 µs	32.04081 µs	21.072346 µs	17.85990 µs	14.804730 µs

Obrázok 7.11: Tabuľky nameraných hodnôt verzie 4.0





Obrázok 7.12: Grafické porovnanie výsledkov verzie 4.0

Kapitola 8

Záver

V tejto práci bol navrhnutý softwarový model tak, aby jeho operácie odpovedali skutočnej funkcii firewallu. Model bol implementovaný v niekoľkých verziách, pričom sa dôraz kládol na zvýšenie rýchlosti klasifikácie. Všetky verzie klasifikátoru boli pred samotným meraním výkonnosti podrobené testom správnosti implementácie. Po testovaní správnosti implementácie bolo spustené meranie výkonnosti. Pri tomto meraní boli použité rôzne sady pravidiel a vstupných súborov s paketmi.

Z hodnôt získaných meraním je možné pri každej z verzií vyvodiť určité závery. Pri klasifikátore v1.0 je použitý jednoduchý lineárny modul LPM. Z tohoto dôvodu je rýchlosť klasifikácie veľmi závislá na tvare vstupných prefixov a vstupných paketov. U v2.0 je tento fakt mierne kompenzovaný použitím 2-úrovňovej stromovej štruktúry pre uloženie jednotlivých prefixov. Rýchlosť klasifikácie je síce vyššia, ale zrýchlenie je opäť závislé na tvare vstupných prefixov. Vo v3.0 je použitý LPM modul, ktorý využíva TBM (Tree Bitmap) algoritmus. Tento modul priniesol až 2-násobné zrýchlenie oproti v2.0. Aj napriek tomuto zrýchleniu je však stále čas modulu LPM takmer 10-násobne dlhší ako čas modulu PHCA. Čas modulu PHCA je v prípade, že paket odpovedá nejakému pravidlu, ktoré je uložené v hash tabuľke, konštantný. Problém nastáva v prípade, že paket neodpovedá žiadnemu pravidlu z hash tabuľky a je nutné prechádzať zoznamom odstránených pravidiel (pravidlá, ktoré sú pri tvorbe pomocných dátových štruktúr odstránené kôly optimalizácii, spoilers). Tento zoznam je lineárny a čas PHCA je potom závislý od jeho dĺžky. Veľký rozdiel v časoch modulov LPM a PHCA sa snaží riešiť klasifikátor v4.0 tým, že používa 2 klasifikačné vlákna. Prvotný návrh používal jedno vlákno, ktoré vykonávalo modul LPM a druhé vlákno, ktoré vykonávalo modul PHCA. Z nameraných výsledkov klasifikátoru v3.0 vyplynulo, že vlákno vykonávajúce modul PHCA by nebolo efektívne využité, pretože čas modulu LPM je oveľa dlhší ako čas modulu PHCA. Preto bola nakoniec zvolená a implementovaná varianta, v ktorej dve vlákna klasifikujú vstupné pakety paralelne. Týmto sa rýchlosť klasifikácie opäť takmer 2-násobne zvýšila. Meranie ukázalo, že pri klasifikácii menšieho počtu paketov (do 1000), je efektivita vláknového spracovania pomerne nízka. Tento fakt zrejme spôsobuje čas, ktorý trvá inicializácia vlákien a ich vzájomná synchronizácia. Rýchlosť klasifikácie významne ovplyvňuje aj výkon počítača, na ktorom je klasifikátor spustený. Meraním sa tiež overila vlastnosť klasifikačného algoritmu využívajúceho Perfect-Hash a síce, že čas klasifikácie nie je závislý na počte pravidiel.

Pokračovaním tejto práce by mohla byť optimalizácia algoritmu Perfect-Hash pre software alebo meranie vplyvu veľkosti cache pamäti na rýchlosť klasifikácie.

Literatúra

[1] Firewall [online]. Wikipédia, [cit. 23.11.2010].
URL http://sk.wikipedia.org/wiki/Firewall

[2] NORIS, I.: Firewall In: Príručka systémového administrátora [online]. [cit. 23.11.2010].
URL http://deja-vix.sk/sysadmin/firewall.html#firewall_whatis

[3] OČENÁŠEK, P.: Firewally, systémy IDS a IPS In: IBS, studijní materiál. VUT Fakulta informačných technológií, 2010.

[4] SEMANCIK, R.: *Stavový paketový filter*. Diplomová práca, Slovenská technická univerzita v Bratislave, 2000.
URL http://storm.alert.sk/work/papers/dipl-projekt/html/node35.html

[5] Song, H.: Evaluation of Packet Classification Algorithms [online]. 2007 [cit. 23.11.2010].
URL http://www.arl.wustl.edu/~hs1/

[6] GUPTA, P.; MCKEOWN, N.: Algorithms for packet classification. *Network, IEEE*, ročník 15, č. 2, mar/apr 2001: s. 24 – 32, ISSN 0890-8044, doi:10.1109/65.912717.

[7] GUPTA, P.: *Algorithms for Routing Lookups and Packet Classification*. Phd thesis, Stanford University, 2000.

[8] PUŠ, V.; KOŘENEK, J.: Fast and scalable packet classification using perfect hash functions. In *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, New York, US: ACM, 2009, ISBN 978-1-60558-410-2, s. 229–236.

[9] KEKELY, L.: *Hardwarová akcelerace operace hledání nejdelšího společného prefixu*. Bakalárska práca, VUT Fakulta informačných technológií, 2011.

[10] ostinato - Packet/Traffic Generator and Analyzer [online]. c2011, [cit. 23.04.2011].
URL http://code.google.com/p/ostinato/