VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

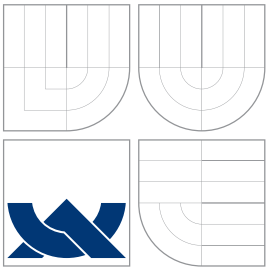
ALGORITMY PRO KLASIFIKACI PAKETŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

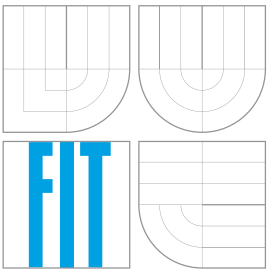
AUTOR PRÁCE
AUTHOR

MARTIN FOUKAL

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ALGORITMY PRO KLASIFIKACI PAKETŮ

PACKET CLASSIFICATION ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN FOUKAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. VIKTOR PUŠ
SUPERVISOR

BRNO 2012

ABSTRAKT

Tato práce se zabývá algoritmy pro klasifikaci paketů, které jsou určené pro filtrování provozu v počítačových sítích. Pojednává o různých oblastech využití klasifikace paketů. Popisuje množství algoritmů včetně paměťových a rychlostních charakteristik. Dále práce popisuje implementaci dvou vybraných algoritmů založených na bitovém paralelismu a bitových vektorech, které byly integrovány do Netbench, experimentálního frameworku pro testování síťových algoritmů. Jsou popsány paměťové požadavky obou algoritmů, které byly doloženy testováním na různých sadách pravidel. Tyto požadavky jsou porovnány s dalšími algoritmy v Netbench.

KLÍČOVÁ SLOVA

klasifikace paketů, filtrování, firewally, algoritmy

ABSTRACT

This work deals with the packet classification algorithms for traffic filtering in computer networks. It contains summary of different areas where packet classification is used. It describes various algorithms and their memory and speed characteristics. Then this work describes implementation of two chosen algorithms based on bit paralelism and bit vectors which were integrated into Netbench, framework for evaluation and experiments with packet processing algorithms. There are described memory requirements of both these algorithms which were tested for different sets of rules. These requirements are compared with other algorithms in Netbench.

KEYWORDS

packet classification, filtering, firewalls, algorithms

FOUKAL, Martin *Algoritmy pro klasifikaci paketů*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, Ústav počítačových systémů, 2012. 39 s. Vedoucí práce byl Ing. Viktor Puš

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Algoritmy pro klasifikaci paketů“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Viktoru Pušovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

ÚVOD

Tato bakalářská práce se zabývá problematikou algoritmů pro klasifikaci paketů v počítačových sítích, primárně v prostředí globální počítačové sítě Internet. Klasifikace paketů se stala postupem času stále důležitější součástí fungování Internetu. Požadavek na spolehlivý a rychlý přenos interaktivních dat, streamingu multimédií spolu s mnohonásobným nárůstem síťového provozu, přinesl řešení v podobě různých úrovní kvality služeb, od původního nespolehlivého best effort přístupu až po garantované služby. Původně podceňovaná problematika bezpečnosti doznala obrovského pokroku, a jedním z pilířů je rozvoj firewallů a technik filtrování dat. Všechny tyto aspekty s sebou přinesly rozvoj algoritmů, jejichž cílem je klasifikace co nejrychlejší, časově úsporná a zároveň paměťově nenáročná.

Ve své práci nejprve čtenáře seznámím s úvodními prerekvizitami, jejichž znalost je nutná pro další rozbor problematiky klasifikace paketů. Popíši vrstvový model sítí a jednotlivé vrstvy protokolového zásobníku TCP/IP a osvětlím, které části jsou zajímavé z hlediska klasifikace. Poté rozeberu jednotlivé oblasti, kde nachází algoritmy pro klasifikaci paketů uplatnění. Zaměřím se stručně popis kvality služeb (QoS), využití klasifikace ve směrování a především v oblasti filtrování. Rozeberu princip, fungování a jednotlivé typu firewallů.

Ve třetí kapitole se již zabývám problematikou samotných algoritmů. Nejprve teoreticky popisuji problém klasifikace a klasifikátoru. V další části se věnuji rozboru jednotlivých algoritmů, které klasifikují podle vícenásobných kritérií, a jejichž využití je hlavně v oblasti bezstavových firewallů. Soustředím se na princip jejich činnosti a také paměťovou a prostorovou složitost.

Poslední část práce popisuje mé praktické programátorské úsilí při implementaci dvou algoritmů založených na paralelismu a bitových vektorech. Rozeberu implementaci algoritmu v prostředí Netbench, specializovaného frameworku určeného pro zjednodušenou implementaci a testování algoritmů pro zpracování paketů napsaného v jazyce Python.

1 VRSTVOVÝ MODEL INTERNETU A FORMÁT PAKETU

První kapitola této práce stručně pojednává o vrstvovém modelu v počítačových sítích. Rozebírá nejprve referenční model ISO/OSI a poté protokolový zásobník TCP/IP, který je základem Internetu. Zdůrazňuji jaké části paketů jsou důležité z hlediska jejich klasifikace.

Základní myšlenkou přenosu dat v počítačových sítích je jejich postupné zapouzdřování do jednotlivých vrstev, kde každá vrstva má na starosti určitou funkci a využívá přitom služeb vrstev nižších. Při poslání určitých dat (poslání zprávy) na úrovní aplikace dochází k postupnému zapouzdřování až na úroveň vrstvy nejnižší. Při příjmu dat se jedná o opačný proces odpouzdřování a aplikaci je předána pouze přenášená zpráva. Realizace jednotlivých vrstev, jejich charakteristika a funkce jsou popsány daným síťovým modelem. Zpočátku vývoje počítačových sítí vznikala celá řada proprietárních architektur. V dnešní době se používají v oblasti internetu modely ISO/OSI a TCP/IP.

1.1 Referenční model ISO/OSI

Referenční model ISO/OSI vypracovala standardizační organizace ISO a v roce 1984 ho přijala jako mezinárodní normu. Model je výsledkem snah o standardizaci počítačových sítí. Popisuje jednotlivé vrstvy (jejich funkce a služby), které jsou vzájemně nahraditelné a nezávislé. OSI nespecifikuje implementaci, pouze uvádí všeobecné principy architektury, která je rozdělena na sedm vrstev. Každá ze sedmi vrstev vykonává skupinu jasně definovaných funkcí potřebných pro komunikaci. Pro svou činnost využívá služeb své sousední nižší vrstvy. Své služby pak poskytuje sousední vyšší vrstvě.

1.1.1 Popis vrstev a jejich stručná stručná charakteristika

- **Aplikační vrstva** – nejblíže ke koncových uživatelům, interaguje přímo s konkrétními aplikacemi a síťovými službami
- **Prezentační vrstva** – zajišťuje nezávislou reprezentaci přenášených dat, stará se o kódování či kompresi
- **Relační vrstva** – ustanovuje a zajišťuje spojení mezi komunikujícími aplikacemi
- **Transportní vrstva** – zajišťuje spolehlivý přenos dat mezi koncovými uživateli

- **Síťová vrstva** – stará se o adresování a směrování dat
- **Linková vrstva** – přenos dat mezi jednotlivými síťovými zařízeními, adresování, vytváření rámců, detekce chyb
- **Fyzická vrstva** – definuje elektrické a fyzikální charakteristiky přenosu, vlastnosti přenosového média

1.2 Model TCP/IP

TCP/IP je základním modelem Internetu a kořeny jeho vývoje sahají až k předchůdci internetu ARPAnet. Oproti OSI je architektura výrazně jednodušší, tvořená čtyřmi vrstvami : aplikační, transportní, internetovou a vrstvou síťového rozhraní. Cílem bylo vytvořit decentralizovanou, paketově orientovanou síť odolnou vůči nepřátelskému napadení. Protokoly IP a TCP byly standardizovány v roce 1981.

V následujících podsekcích se zaměřím na popis jednotlivých vrstev, s příklady komunikačních protokolů a názvů protokolových datových jednotek (PDU). Také uvedu jaké části protokolů jsou důležité z hlediska klasifikace paketů.

1.2.1 Aplikační vrstva

Vrstva na úrovni jednotlivých aplikací. Z modelu OSI zahrnuje i vrstvy prezentační a relační. Aplikační komunikační protokoly se dělí na ty, které běží nad spolehlivou transportní službou TCP a ty, které využívají nespojovaného,nespolehlivého přenosu UDP. Mezi protokoly využívající TCP patří např. HTTP,FTP a SMTP. Nad UDP běží např. RTP, DHCP nebo DNS. Protokolová datová jednotka (PDU) aplikační vrstvy se nazývá **zpráva**. Využití této vrstvy při klasifikaci paketů nalezneme především u firewallů typu **aplikační brána** (2.3.2)

1.2.2 Transportní vrstva

Transportní vrstva zajišťuje přenos dat ze zdrojového počítače na cílový. Transportní protokol se stará o rozdělení zpráv určených aplikacím (komunikujícím skrze protokol aplikační vrstvy), přičemž vytváří logické spojení mezi síťovými procesy běžících na vzdálených počítačích. Tyto procesy jsou adresovány čísly portů. Zdaleka nejpoužívanějšími protokoly transportní vrstvy jsou TCP (*Transmission Control Protocol*) a UDP (*User Datagram Protocol*). Protokolové datové jednotky se nazývají **segmenty**.

TCP zajišťuje spolehlivý přenos dat mezi vzdálenými procesy, jedná o tzv. spojovanou službu, kde před začátkem samotného datové přenosu proběhne ustanovení

spojení pomocí synchronizačních paketů a na konci komunikace zase korektní ukončení. Spolehlivý přenos je realizován sekvenčními čísly paketů a jejich potvrzováním. TCP také řídí tok a snaží se o prevenci zahlcení. Z těchto důvodu je struktura hlavičky TCP segmentu poměrně složitá (základní velikost 20 B).

UDP realizuje nespolehlivou a nespojovanou službu pro přenos dat mezi aplikacemi. Vzhledem k tomu je jeho režie menší (velikost hlavičky 8 B) a používá se tam, kde je kladen důraz na rychlost přenosu před spolehlivostí doručení. Proto jeho využití nalezneme u služeb zajišťujících přenos multimédií (protokol RTP), u služby DNS nebo SNMP.

Základními údaji v hlavičkách transportních protokolů jsou čísla zdrojového a cílového portu. Významné jsou také z pohledu klasifikace paketů. Číslo portu se používá jako jeden z parametrů klasifikace toků u realizace kvality služby (více v sekci 2.2). Stejně tak je důležité z hlediska filtrování paketů (více v sekci 2.3). Číslo portu je tvořeno 16bitovou hodnotou (0 až 65535), která identifikuje síťovou službu bežící na daném počítači. Rozlišují se porty rezervované (0–1023), registrované (1024–49151) a dynamické. O rezervaci a registraci jednotlivých služeb se stará organizace IANA[1]. Rezervace čísel portů u jednotlivých služeb je klíčová kvůli jejich snadné dostupnosti (např. web server běžící na portu 80, DHCP server na portu 67 nebo DNS na portu 53), což se dá také využít pro jejich snadné povolení či zakázání v nastavení filtrovacích pravidel firewallu (3.1).

1.2.3 Internetová vrstva

Internetová resp. síťová vrstva odpovídá síťové vrstvě OSI modelu. Má především na starosti adresování a směrování přenášených dat v internetu. Zajišťuje tzv. doručování dat s největším úsilím (*best effort delivery*), což znamená, že sice není zajištěna spolehlivost přenosu dat, ale je vyvinuta snaha o doručení nejvhodnější cestou. Protokolová datová jednotka se nazývá **datagram**. Datagram při své cestě k cíli prochází množinou směrovačů, které se starají o jeho směrování na další síťový uzel dle svých směrovacích tabulek (sekce 2.1). Základním protokolem je protokol IP (Internet Protocol), který se používá verzi 4 a nyní dochází k postupnému přechodu na verzi 6. Oba dva protokoly ve svých hlavičkách definují způsob adresování, povolují fragmentaci pro nižší vrstvu síťového rozhraní a umožňují nastavit kvalitu služby (sekce 2.2). Kromě IPv4 a IPv6 jsou součástí síťové vrstvy také protokoly ICMP pro řízení a kontrolu, protokoly pro překlad IP adres na MAC adresy a protokoly pro podporu multicastu.

[1] http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml

Nejvýznamnějšími položkami IP datagramu při klasifikaci jsou zdrojová a cílová IP adresa. U protokolu IPv4 se adresuje podle 32bitových adres zapisovaných ve formátu tečkou oddělených hodnot jednotlivých bytů v desítkové soustavě (např. 152.12.170.4). To umožňuje teoreticky adresovat 2^{32} (přes 4 miliardy) unikátních adres. Tento počet se brzy ukázal s prudkým rozvojem internetu jako dlouhodobě nedostačující a stal se tak hlavní motivací pro vytvoření verze IPv6. IPv6 adresy mají 128 bitů a zapisují se v šestnáctkové soustavě s dvojicemi bytů oddělených dvojtečkou (např. 2001:0db8:0000:0000:0000:0000:1428:57ab). Klasifikaci podle cílové IP adresy nalezneme u směrování (kap.), zdrojová i cílová adresa se používá při klasifikaci toků (2.2), filtrování (2.3), i multicastového směrování.

1.2.4 Vrstva síťového rozhraní

Vrstva síťového rozhraní (*Network Interface Layer*) odpovídá fyzické a linkové vrstvě modelu OSI. Definuje charakteristiky přenosového media a přenosových signálů. Zajišťuje řízení přenosového media a zapouzdřování IP datagramů do rámců. Funkci této vrstvy zajišťuje síťová karta. Pro síťová rozhraní v místních a metropolitních sítích je definována rodina standardů IEEE 802 (např. 802.3 pro Ethernet a 802.11 pro WLAN). Každý rámec v těchto sítích obsahuje ve své hlavičce zdrojovou a cílovou MAC adresu (každá 48 bitů) a délku, resp. typ přenášených dat (16 bitů). MAC adresa je identifikátor daného síťového rozhraní, prvních 24 bitů tvoří přidělený identifikátor výrobce a dalším 24 bitů pak unikátní číslo daného zařízení (např. 01:23:45:67:89:AF). Pomocí MAC adres se adresují zařízení v daném síťovém segmentu, přičemž se obvykle používají switche k přepínání jednotlivých rámců. Z pohledu klasifikace má tato vrstva nejmenší význam, ale mohou se využívat paketové filtry blokující rámce přicházející z konkrétního síťového rozhraní (např. na hranici podnikové sítě).

2 VYUŽITÍ KLASIFIKACE PAKETŮ

V této kapitole rozeberu jednotlivé oblasti, v nichž klasifikace paketů nachází uplatnění. Krátce popíši problematiku směrování, oblast kvality služeb a nakonec se budu zabývat filtrováním a firewally, kde se v další kapitole popisované algoritmy nejvíce uplatňují.

2.1 Směrování

Internet je globální síť založená na přepínání paketů. Zařízení, které pracuje na síťové vrstvě a má za úkol přeposílání paketů mezi propojenými sítěmi se nazývá směrovač (router). Směrovač propojuje dvě a více sítí a v případě příchodu paketů na vstupní rozhraní se rozhoduje na jaké výstupní rozhraní bude paket odeslán. Tomuto procesu se říká **směrování** a rozhodovacím kritériem je cílová IP adresa. Proces směrování je realizován pomocí směrovací tabulky, která obsahuje seznam cílových adres a k nim přiřazené adresy dalšího skoku (*next hop routery*). Při příchodu paketů do směrovače se pomocí daného algoritmu vyhodnotí odpovídající záznam ve směrovací tabulce a paket může být odeslán na další adresu uvedenou v záznamu (přes dané síťové rozhraní). Takto datagram prochází množinou navzájem propojených směrovačů dokud nedorazí ke svému cíli, pokud není po cestě zahozen např. při zahlcení front či při snížení položky TTL v hlavičce datagramu na nulu.

Ke vzájemné komunikaci mezi směrovači se využívají směrovací protokoly, pomocí nichž si směrovače vytvářejí informace o síťové topologii a mohou se tak rozhodovat, jakou nejvhodnější cestou mají směrovat. Směrovací protokoly obvykle fungují na principu Distance vector algoritmu (založený na Bellman-Ford algoritmu), kde se k nalezení nejvhodnější cesty používá pouze komunikace mezi sousedními routery, nebo na principu Link-State algoritmu, který vychází z Dijkstrova algoritmu pro nalezení nejkratší cesty v orientovaném grafu a v němž musí mít směrovače znalost o celkové topologii. Směrovací protokoly dále můžeme dělit na ty, které fungují v rámci autonomního systému a na ty, které mezi nimi. Autonomním systémem je myšlena obvykle WAN síť pod jednotnou správou a s jednotnou směrovací politikou. Příklady protokolů v rámci AS jsou RIP (Distance vector) a OSPF (Link–state). Mezi AS pak především BGP (Link–state).

2.1.1 Třídní adresování

V původním návrhu internetového protokolu bylo zavedeno tzv. třídní adresování, kde mají jednotlivé třídy jednoznačně vymezený rozsah a masku:

- A - 0.0.0.0 - 127.255.255.255, maska 255.0.0.0

- B - 128.0.0.0 - 191.255.255.255, maska 255.255.0.0
- C - 192.0.0.0 - 223.255.255.255, maska 255.255.255.0
- D - 224.0.0.0 - 239.255.255.255, multicast
- E - 240.0.0.0 - 255.255.255.255, rezervováno

Daná maska určuje rozsah adresy sítě (bity v jedničce) a rozsah adresy adresy uzlu (bity v nule). Třídní adresování má výhodu v tom, že u každé adresy je zřejmá maska sítě a proto tato informace v prvních směrovacích protokolech nebyla přítomna. Jak se však brzy ukázalo s boomem Internetu, třídní adresování neumožňovalo adekvátní škálovatelnost a kvůli nemožnosti agregace rostla neúměrně velikost směrovacích tabulek. Samotné vyhledávání ve směrovacích tabulkách však bylo jednoduché. Směrovače měli typicky pro každou třídu zvláštní směrovací tabulku. Při příchodu paketu se identifikovala třída cílové adresy a extrahovala síťová část. Tato část pak sloužila jako hesh index do konkrétní tabulky. Tuto metodu označujeme jako přesné porovnávání (*exact matching*)

2.1.2 Beztřídní adresování (CIDR)

Z výše popsaných důvodů se hledal lepší způsob adresování a došlo k zavedení tzv. beztřídního adresování (*Classless Inter-Domain Routing*). CIDR nejnověji popsáno v RFC 4632 ruší předchozí přiřazování intervalů adres do tříd a místo toho zavádí adresování s využitím prefixové notace. Každá adresa má přiřazený prefix, jehož hodnota určuje počet bitů masky v logické jedničce, resp. délku síťové části adresy (např. adresa sítě 200.155.60.128/25 určuje síť s maskou 255.255.255.128, která může obsahovat $(2^{32-25} - 2)$ síťových uzlů). Tento přístup umožňuje přidělovat adresy s libovolným prefixem bez ohledu na třídu (v praxi se nepoužívají prefixy menší než 8). Výhodou CIDR je tedy lepší využitelnost adresového prostoru přesně podle potřeb konkrétní organizace. Kromě toho umožňuje hierarchické přidělování adres. Hierarchické přidělování adres umožňuje poskytovatelům Internetu agregovat směrovací záznamy o sítích, které jsou k nim připojeny, díky čemuž dochází k redukci velikosti směrovacích tabulek. Mezi problémy patří vznik „děr“ v adresovém prostoru, když organizace změní ISP, ale ponechá si své adresy.

Vzhledem k tomu, že směrovací protokoly původně nepracovaly se síťovou maskou, protože ta byla podle třídy implicitně dána, muselo dojít k jejich přepracování. Stejně se musely přizpůsobit i samotné směrovače. U CIDR směrovací záznamy obsahují adresu sítě (či jejich agregace) s uvedeným prefixem (např. 198.10.0.0/22) a adresu *next-hop* routeru. Z tohoto důvodu je nalezení správného záznamu komplikovanější. Když paket přijde do směrovače, hledá se takový záznam v tabulce, který se v prvních n bitech shoduje s prvními n bity paketu, tak aby n bylo co největší. Tento proces se nazývá hledání **nejdelšího shodného prefixu** (*longest prefix*

match). Hledání nejdelšího shodného prefixu je algoritmický problém, kdy hledáme nejvíce specifický záznam, o tom kam příchozí paket směrovat. LPM algoritmy se snaží o co nejvýkonnější porovnávání a jejich implementace nachází uplatnění ve vysokorychlostních směrovačích. Tato práce se ovšem těmito algoritmy nezabývá.

2.2 Kvalita služeb (QoS)

Další oblastí, kde se uplatňuje klasifikace paketů je zajišťování kvality služeb (*Quality of Service*). Služby QoS zajišťují diferencované úrovně přenosu dat. Požadavky na kvalitu služeb jsou většinou mezi zákazníkem a ISP definovány v dohodě zvané SLA (*Service Level Agreement*), která definuje základní parametry připojení k síti jako jsou šířka přenosového pásma, ztrátovost, zpoždění či rozptyl (jitter). Aby byla tato dohoda naplněna, musí se využívat určitá politika realizace kvality služeb. K té patří klasifikace paketů do různých toků a na základě toho přidělování určité priority při průchodu směrovači (*packet marking*). Ve směrovačích se používají různé mechanismy plánování a řazení paketů do výstupních front. Mohou se používat fronty různých typů: klasické FIFO fronty, prioritní fronty, cyklické nebo váhové fronty. K regulaci rychlosti a rozkládaní provozu na výstupních rozhraních se používá metoda zvaná *traffic shaping*. K omezování provozu *traffic policing*.

V dnešním Internetu rozlišujeme tři základní mechanismy QoS: best-effort služby, diferencované služby (DiffServ) a integrované služby (IntServ).

2.2.1 Best-effort služby

Best-effort je standardní a původní službou v Internetu. Snaží se o doručovaní paketů s největším úsilím, tedy o co nejrychlejší a nejefektivnější doručení, ovšem neposkytuje jakékoli garance, že se tak stane. Hodnota kvality služeb (QoS) je nastavena na nulu.

2.2.2 Integrované služby (IntServ)

Prvním používaným modelem kvality služeb byly integrované služby. Tyto služby zajišťují pro dané toky rezervaci přenosového pásma. Směrovač podporující IntServ nejprve provede klasifikaci příchozího paketu a na základě definovaného toku mu přiřadí danou třídu služeb, která definuje parametry přenosu a plánovací politiku na výstupních frontách. IntServ definuje garantovanou službu, která zajišťuje požadovanou rychlost a zpoždění, službu řízené zátěže, která již neposkytuje stoprocentní garance parametrů přenosu, a základní službu největšího úsilí (best-effort).

K rezervaci zdrojů na jednotlivých směrovačích se používá signalizační protokol RSVP, který umožňuje, aby si přímo aplikace rezervovaly pásmo s danou kvalitou služeb a specifikací provozu. RSVP funguje v multicastových stromech, kde rezervační požadavky jsou vysílány od příjemců ke zdrojům dat.

Architektura integrovaných služeb se příliš nevyužívá. Nevýhodou je malý počet tříd služeb a především výpočetní náročnost. K rezervaci zdrojů dochází pro jednotlivé toky zvlášť, což je pro vytíženější směrovače neúnosné.

2.2.3 Diferencované služby (DiffServ)

Diferencované služby jsou dnes nejpoužívanější metodou zajišťování kvality služeb. Oproti integrovaným službám neprovádějí rezervaci zdrojů na základě jednotlivých toků, ale rozdělují pakety do diferencovaných tříd provozu s definovaným chováním a úrovní QoS. Výpočetně náročné operace provádějí pouze hraniční směrovače s podporou DiffServ, které nejprve příchozí paket klasifikují podle jednotlivých parametrů. Těmi jsou nejčastěji zdrojová a cílová IP adresa, zdrojový a cílový port a typ transportního protokolu. Následně se provede označkování, od kterého se odvíjí definovaný typ chování. Další směrovače v síti poté kontrolují přicházející provoz a měří, zda-li nedošlo k překročení provozu pro definovanou třídu. Směrovače na rozhraní různých ISP obvykle provádějí přeznačkování podle své interní politiky.

Ke značkování paketů se využívá oktet Type of Service (ToS) v hlavičce IPv4 resp. oktet Traffic Class v hlavičce IPv6. ToS byl původně rozdělen na 3 bity označující prioritu paketu a 4 bity pro typ služby, toto rozdělení se ale příliš nepoužívalo. Se zavedením DiffServ se oktet změnil a prvních 6 bitů určuje diferencované třídy provozu (DSCP). Každá třída provozu má definované chování podle tzv. PHB (Per hop behavior). PHB určuje jak nakládat s pakety dané třídy, ale nepopisuje jak provádět skutečnou implementaci (plánování front apod.). Rozlišuje se tzv. přednostní přeposílání (EF) s maximální garancí bez možnosti zahazování a omezování, garantované přeposílání (AF) s různými prioritami zahazování při zahlcení linky a implicitní best-effort bez jakékoli garance.

2.3 Filtrování paketů a firewally

Firewally jsou síťová zařízení určená k filtrování a zabezpečení provozu mezi sítěmi. Obvykle jsou integrovány ve směrovačích, kde poskytují zabezpečení hranic sítě kontrolou vstupního a výstupního provozu. Mohou pracovat také jako specializovaný software na serverech i klientských stanicích. Kontrolovaný provoz je s využitím filtrovacích pravidel a klasifikace paketů propouštěn nebo zahazován. Vývoj firewallu a zabezpečení sítí prodělal v posledních dekádách udělal obrovský skok vpřed. Od jednoduchých paketových filtrů až po detekční (IDS) a prevenční systémy (IPS) schopných detekovat, resp. odklonit prováděný útok. V této sekci popíši základní typy firewallů, mezi které patří bezstavové a stavové paketové filtry, překladače síťových adres a proxy servery.

2.3.1 Paketové filtry

Bezstavové paketové filtry jsou nejstarším zabezpečením typu firewall. Jejich funkcí je klasifikovat pakety podle určených kritérií a pomocí filtrovací pravidel je buď propustit nebo zahodit. Mohou fungovat jako samostatná hardwarová zařízení nebo jako součást směrovačů. Většina operačních systémů také poskytuje podporu pro filtrování paketů jako součást své implementace TCP/IP zásobníku. Filtruje se podle nejdůležitějších údajů síťové a transportní vrstvy:

- Zdrojové a cílové IP adresy
- Zdrojového a cílové portu
- Protokolu transportní vrstvy (TCP/UDP)

Filtrovat se může i pomocí dalších kritérií, např. k filtrování provozu přicházejícího z konkrétního rozhraní lze filtrovat i na úrovni linkové vrstvy.

Nejdůležitějším článkem paketových filtrů je správné nastavení filtrovacích pravidel. Pakety jsou filtrovány pomocí daného algoritmu pro klasifikaci, kde klasifikátor tvoří filtrovací pravidla s akcemi povolit a zahodit. Pravidla jsou řazena podle priority, takže v případě, že se dá pro daný paket aplikovat více pravidel, uplatní se to s nejvyšší prioritou. Formát filtrovacích pravidel:

```
<number> <action> <protocol>
   from <src IP> to <dest IP>
   [src-port <srcPort>] [dst-port <dstPort>]
   [<flags>] [<options>]
```

Příklad pravidla, které blokuje přístup na web server pro danou IP:

```
10 deny TCP from 88.203.34.2 to 147.229.9.23 dst-port 80
```

U pravidel se používá dvojí přístup: optimistický, který blokuje pakety podle vypsaných pravidel a všechny ostatní propouští, a pesimistický, který naopak určuje

pravidla pro povolování a ostatní provoz blokuje. Výhodou paketových filtrů je jednoduchost a rychlost zpracování, což je činí vhodnými i pro vytížené linky. Nevýhodou bezstavových filtrů je obtížnost nastavení pravidel. Zdrojové porty s vyššími čísly (nad 1024) bývají povoleny kvůli klientským požadavkům, což způsobuje bezpečností rizika (např. riziko naslouchání trojských koňů). Tyto nedostatky odstraňuje další generace firewallů: paketové filtry s kontrolou stavu.

Paketové filtry s kontrolou stavu si zaznamenávají informace o ustanovení komunikace. U TCP zaznamenají úvodní SYN paket, u bezstavového UDP mohou ukládat pseudostavy. Pro tyto ustavená spojení poté povolují komunikaci a ostatní přístup blokují. Tím je zajištěn bezpečný přístup klientů z interní sítě k serverům ve vnější sítí. Přímý přístup k vyšším portům do interní sítě bývá blokován, pokud to není explicitně povoleno. Stavové filtry výrazně zvyšují bezpečnost, ale nejsou schopny analýzy samotných dat v aplikační vrstvě.

2.3.2 Proxy servery

Dalším typem firewallů jsou proxy servery. Proxy se servery se původně používaly jako vyrovnávací paměti v dobách, kdy většina stránek byla statických a spojení do internetu velice pomalé. Nakonec se osvědčila jejich bezpečností role. *Aplikační proxy* (aplikační brány) mají funkci prostředníka na aplikační vrstvě, který přijímá požadavky od klienta a generuje úplně nové požadavky na cílový počítač, přijaté odpovědi poté přeposílá klientům zpět. Tím se klienti úplně odstiňují od probíhající komunikace, což zvyšuje bezpečnost interní sítě. Každá aplikační proxy zajišťuje činnost pouze pro síťovou službu (nejčastěji web–protokol HTTP). Kromě toho existují i tzv. *generické proxy* (např. SOCKS), které přesměrovávají komunikaci na transportní vrstvě. Proxy mohou kontrolovat datovou část paketů a filtrovat nebezpečný obsah. Mezi nevýhody patří nutnost nastavit klienty pro spolupráci s proxy. Tento nedostatek řeší tzv. transparentní proxy. Klienti nemusí nic nastavovat a o přesměrování se stará příslušný firewall.

3 ALGORITMY PRO KLASIFIKACI PAKETŮ

V minulé kapitole jsme si ukázali, že klasifikace paketů nachází četné uplatnění. Routery na rozhraní sítí kromě samotného směrovaní mohou zajišťovat filtrování, značkovat pakety, zajišťovat kvalitu služeb vhodným řazením do front, měřit, zda-li nedošlo k překročení provozu pro danou službu, a omezovat provoz. Pro všechny tyto činnosti je nutná nějaká forma klasifikace.

Přicházející pakety jsou klasifikovány pomocí sady pravidel zvané **klasifikátor**. Každé pravidlo specifikuje určitý tok založený na kritériích, se kterými se přicházející paket může shodovat, a příslušnou akci, která je pro paket aplikována v případě nejvýznamnější shody. Pravidla mají také určenou prioritu, která je určující v případech, kdy se může pro příchozí paket aplikovat více pravidel.

Mějme klasifikátor C obsahující N pravidel R_j, kde $1 \leq j \leq N$. Klasifikátor, tak můžeme popsat jako množinu pravidel $R = \{R_1, \ldots, R_N\}$. Každé pravidlo R_j v klasifikátoru C je tvořeno z d částí, které označujeme jako **dimenze**. Pravidlo je tedy n-tice, kde jako $R[i]$ označujeme položku pravidla v dimenzi i. V obecném smyslu můžeme $R[i]$ chápat jako regulární výraz. Příchozí paket považujeme za n-tici $P = (P_1, \ldots, P_n)$, kde n je počet dimenzí a $P[i]$ odpovídá položce paketu v í-té dimenzi. Paket P odpovídá pravidlu R, jestliže pro všechny jeho položky $P[i]$ platí, že regulární výraz v $R[i]$ pokrývá $P[i]$.

Položky pravidel v jednotlivých dimenzích nejsou obvykle zadány jako obecný regulární výraz. Porovnávání položek pravidel s hlavičkami paketů se dělí na přesné porovnávání, porovnávání prefixů a porovnávání rozsahů:

- **Přesné porovnávání** (*exact matching*) – Pravidlo je specifikováno jako konkrétní hodnota. Kontroluje se, zda-li je daná část paketu shodná s pravidlem. Obvykle se tímto způsobem zadává přesný port nebo protokol transportní vrstvy (TCP/UDP)
- **Porovnávání prefixů** (*prefix matching*) – Rozsah pravidla je určen prefixem, jehož hodnota určuje zleva počet významných bitů. Rozsah pravidla je tedy určen vztahem $rozsah = 2^{c-p}$, kde c je celkový počet bitů pravidla (u IPv4 32 bitů) a p délka prefixu. Zápis v prefixové notaci je obvykle u IP adres (viz CIDR 2.1.2), např. zápis 150.160.64.0/23 určuje celkem $2^{32-23} = 2^9 = 512$ adres v rozsahu od 150.160.64.0 do 150.160.65.255.
- **Porovnávání rozsahů** (*range matching*) – Přesně specifikovaný rozsah pravidla, obvykle u portů, ale může být i u IP adres (např. 150.160.64.128 až 150.160.65.64). Platí, že spojité prefixy lze převést na rozsah, stejně tak libovolný rozsah lze převést na spojité prefixy (viz 3.0.3).

Tabulka 3.1 ukazuje příklad několika jednoduchých filtrovacích pravidel. Příchozí paket se porovná s pravidly a pokud dojde k nalezení, výsledkem je číslo pravidla

Pravidlo	Zdrojová IP	Cílová IP	Zdrojový port	Cílový port	Protokol	Akce
1	*	188.155.132.60/32	*	80	TCP	Povolit
2	201.168.16.0/24	188.155.132.0/24	*	20-21	TCP	Povolit
3	201.168.16.0/24	188.155.132.0/24	*	23	TCP	Zahodit
4	201.168.16.3/32	188.155.132.0/24	*	23	TCP	Povolit
5	*	188.155.132.21/32	*	53	UDP	Povolit
6	*	*	*	*	*	Zahodit

Tab. 3.1: Příklad klasifikátoru - tabulka filtrovacích pravidel

s příslušnou akcí. Za zmínku stojí implicitní pravidlo 6, které pokrývá všechny pakety, tzn. zahazuje všechny pakety, pro než se neuplatilo žádné z pravidel s vyšší prioritou. Pravidla 3 a 4 ukazují, že určující je nalezení nejdelšího shodného prefixu. Pokud se adresa 201.168.16.3 pokusí připojit na Telnet, aplikuje se pravidlo 4, i když má nižší prioritu než pravidlo 3, které daný paket také pokrývá.

3.0.3 Vztah mezi prohledáváním prefixů a rozsahů

Požadavky na algoritmy, aby byly schopny pracovat s rozsahy i s prefixy, dělají problém více komplexní. Ukážeme si, že existuje vztah, který umožňuje převést libovolný rozsah na prefixy. Definujme si problém vyhledávání v rozsazích v dimenzi D o bitové šířce $W(2)$:

Definice 1. *Mějme množinu disjunktních rozsahů* $R = \{R_i = [l_i, u_i]\}$*, z nichž každý tvoří část číselného rozsahu* $\langle 0, 2^W - 1\rangle$ *a platí, že* $l_1 = 0, l_i \leq ui, l_{i+1} = u_i + 1, u_N = 2^W - 1$*. Potom vyhledávací problém spočívá v nalezení takového rozsahu* R_P*, který obsahuje daný bod* P*.*

Rozsah	Prefixy
[1,3]	0001,001*
[4,7]	01*
[1,14]	0001,001*,01*,10*,110*,1110
[0,15]	*

Tab. 3.2: Příklad převodu rozsahů na prefixy

Každý rozsah můžeme konvertovat na množinu vzájemně spojitých prefixů. Samotný prefix délky s v dimenzi o délce W tvoří rozsah $[l, u]$, kde l má $(W - s)$ bitů hodnotu 0 a u má $(W - s)$ bitů hodnotu 1. Tabulka 3.2 ukazuje příklad převodu rozsahů na prefixy v dimenzi o délce 4 bity (0-15). Prefixy jsou zapsány v notaci,

která se bude používat i u příkladů dalších klasifikátorů. Délku prefixu určuje počet signifikantních bitů zleva (0 nebo 1), zápis hvězdičky pak libovolné bity. Zápis rozsahu [0,15], tj.celý rozsah dimenze se dá zapsat nejobecnějším prefixem (*). Rozsah [1,14] ukazuje případ, kdy je rozsah reprezentován maximálním počtem prefixů podle vztahu $Max = 2W - 2 = 5 * 4 - 2 = 6$.

3.1 Požadavky pro klasifikační algoritmy

V této sekci shrneme požadavky pro klasifikační algoritmy z pohledu jejich výkonu a praktické použitelnosti.

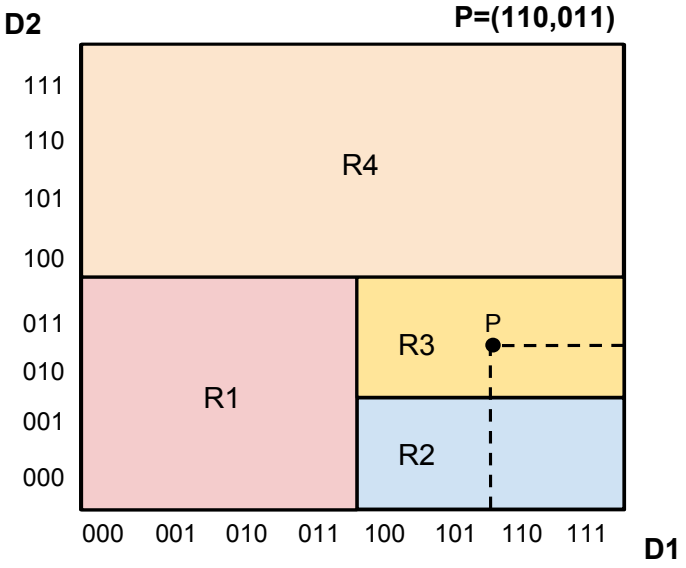
- Rychlost klasifikace – Filtrovací paketů je první činností při vstupu do směrovače. Z tohoto důvodu je nutné, aby zpracování proběhlo co nejrychleji a nebrzdilo další činnost. Algoritmy by měly zpracovávat pakety v nejhorším případě na úrovni provozu linky (nikoliv v průměru), aby se předešlo řazení paketů do front. Řazení do front je kritické, protože před samotnou klasifikací nelze určit příslušnost do daného toku a nastavit tak plánovací politiku. Nelze zajistit kvalitu služeb a hrozí zahlcení linky. Dalším faktorem je, že při zmenšující se velikosti paketů rostou nároky na zpracování, byť celkový průtok dat zůstává nezměněn.
- Způsob implementace – Algoritmy mohou být implementovány softwarově nebo hardwarově v závislosti na výkonnostních požadavcích. Pro vysokorychlostní zpracování je požadována hardwarová implementace, přičemž algoritmy musí být pro tuto implementaci navrženy.
- Paměťové požadavky – Malé paměťové nároky umožňují použít rychlé, ale dražší paměti SRAM.
- Schopnost účinně pracovat s klasifikátory velkého rozsahu
- Rychlost předzpracování (*preprocessing*) a rychlost aktualizací – Rychlost preprocessingu určuje čas nutný k vytváření datových struktur pro potřeby algoritmů. Důležitá je také schopnost vyrovnat se s pozdějšími změnami klasifikátorů (např. změna filtrovacích pravidel, aktualizace směrovací tabulky). Obvykle jsou tyto změny o několik řádů méně časté než samotná frekvence klasifikace. Statické algoritmy vyžadují při změně klasifikátoru kompletní přepočítání datových struktur. Dynamické umožňují aktualizace, takže rychlost preprocessingu může být pomalejší.
- Schopnost klasifikovat různý počet dimenzí a různé položky hlaviček paketů.
- Flexibilní specifikace pravidel – klasifikátory by měly umožňovat zadávání pravidel ve více formátech (prefixy,rozsahy,přesné hodnoty,použití operátorů).

3.2 Geometrická reprezentace pravidel

Pravidlo	Osa X	Osa Y
1	0*	0*
2	1*	00*
3	1*	01*
4	*	1*

Tab. 3.3: Příklad dvoudimenzionálního klasifikátoru

Každý klasifikátor obsahující množinu pravidel, lze znázornit v eukleidovském prostoru. Jednotlivé prefixy nebo rozsahy představují úsečky v ose dimenze d. Klasifikátor obsahující pravidla ve dvou dimenzích, lze tudíž nakreslit jako dvojrozměrný eukleidovský prostor, kde jednotlivá pravidla tvoří obdélníky. Analogicky, klasifikátor s pravidly v n dimenzích přestavuje n-rozměrný prostor, kde pravidla reprezentují n-rozměrné objekty. Klasifikovaný paket pak představuje n-rozměrný bod v tomto prostoru. Tabulka 3.3 obsahuje příklad klasifikačních pravidel ve dvou dimenzích. Geometrické znázornění těchto pravidel ukazuje obrázek 3.1. V tomto obrázku je zakreslen i klasifikovaný paket reprezentovaný bodem $P = (110, 011)$, což představuje dvě hodnoty polí v jeho hlavičkách. Bod spadá do obdélníku $R3$, tudíž pravidlo $R3$ je na paket aplikováno. Pokud bod paketu spadá do prostoru s více navzájem se překrývajícími obdélníky, aplikuje se pravidlo s nejvyšší prioritou. Tuto situaci lze geometricky znázornit tak, že pravidlo s nejvyšší prioritou překrývá ostatní.



Obr. 3.1: Příklad geometrické reprezentace pravidel ve dvourozměrném prostoru

3.3 Třídění klasifikačních algoritmů

Klasifikační algoritmy můžeme dělit podle několika kritérií:

3.3.1 Dělení podle počtu dimenzí

- Prohledávání v jedné dimenzi – Nachází využití především v unicastovém směrování u algoritmů pro nalezení nejdelšího shodného prefixu (LPM).
- Prohledávání ve dvou dimenzích – Klasifikuje je se podle zdrojové a cílové adresy. Využití např. u multicastového smětování.
- Prohledávání ve více dimenzích – Většinou podle zdrojové a cílové adresy,čísel portů a protokolu transportní vrstvy.

3.3.2 Dělení podle techniky vyhledávání a použitých datových struktur

Existuje celá řada typů algoritmů:

- Lineární vyhledávání
- Algoritmy založené na stromových strukturách
- Geometrické algoritmy
- Heuristické algoritmy (např. RFC 3.10, HiCuts 3.8)
- Hardwarové algoritmy (např. BitVector 3.6)
- Algoritmy založené na dekompozici (např. DCFL 3.9)

3.4 Lineární vyhledávání

Lineární vyhledávání je nejjednodušší formou klasifikace paketů. Pravidla jsou seřazena v seznamu sestupně podle priority. Každý příchozí paket je při klasifikaci postupně porovnáván s položkami v seznamu pravidel dokud se paket s nějakým pravidlem neshoduje. Přístup lineárního prohledávání je sice jednoduchý a paměťové nenáročný, hodí se ovšem pouze pro malý počet klasifikačních pravidel. S rostoucím počtem pravidel roste lineárně i časová složitost algoritmu a ten se tak stává nepoužitelným.

3.5 Algoritmy založené na stromových strukturách

3.5.1 Stromová struktura *trie*

Binární stromová struktura zvaná *trie* je binární strom určený pro klasifikaci v jedné dimenzi. Konkrétní uzly stromu představují bity daného prefixu a celý prefix je určen jako cesta od kořenu stromu k danému uzlu. Stromová struktura umožňuje snadné a rychlé prohledávání uzlů, přidávání nových nebo jejich rušení. Strom určený pro dimenzi šířky W bitů má maximální hloubku W uzlů. Časová složitost prohledávání je tedy $O(W)$. Při vložení nového prefixu je v nejhorším případě nutné vložit W nových uzlů. Prostorová složitost při vložení N prefixů o maximální šířce W bitů je $O(NW)$.

Jeden strom nám umožní naleznout nejdelší shodný prefix v jedné dimenzi (LPM). Pro klasifikaci paketů ve více dimenzích můžeme použít hierarchické spojování stromů. Prohledávání začíná u nejvýše postaveného stromu obsahujícího uzly tvořící prefixy pravidel v první dimenzi. Z jednotlivých uzlů pak vedou ukazatele do dalších stromů, představující další dimenze pravidel. Kromě toho lze využít kombinaci stromového prohledávání (např. pro zdrojovou a cílovou IP adresu) a lineárního prohledávání (pro zbývající dimenze).

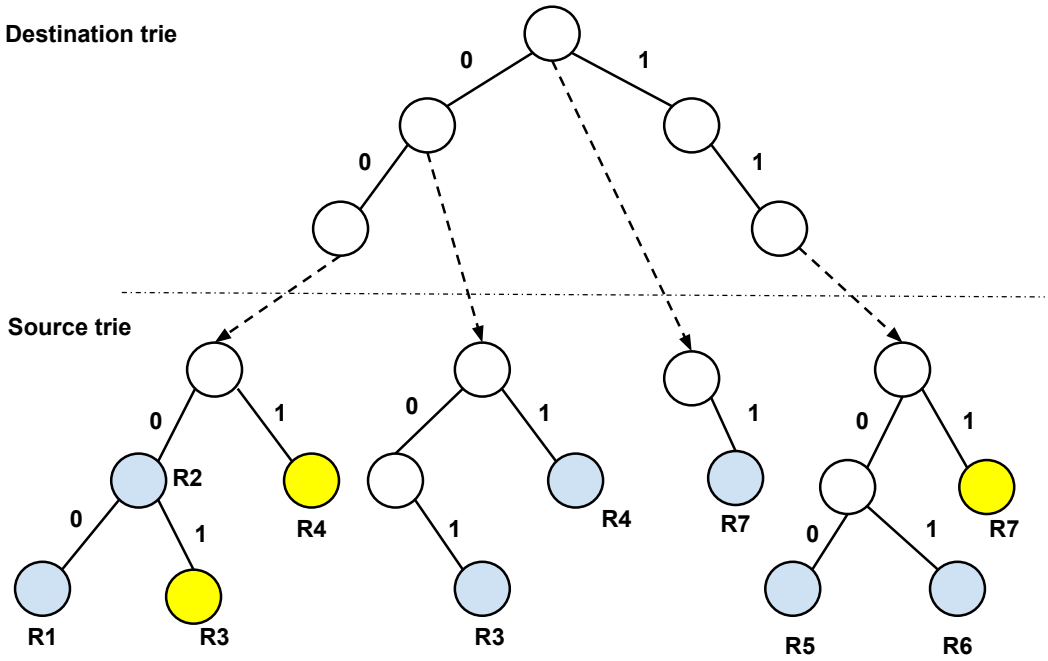
Pravidlo	Cílová adresa	Zdrojová adresa
R1	00*	00*
R2	00*	0*
R3	0*	01*
R4	0*	1*
R5	11*	00*
R6	11*	01*
R7	*	1*

Tab. 3.4: Pravidla znázorněná v ukázkách stromových struktur

3.5.2 Hierarchické stromy s duplicitami

Hierarchické stromy (znázorněné na obrázku 3.2) jsou tvořeny následujícím způsobem: uzly podstromu patřícímu k danému uzlu jsou replikovány do podstromů jeho přímých potomků. Tato replikace je provedena rekurzivně pro podstromy všech dimenzí. V praxi to znamená, že podstromy níže položených uzlů obsahují i pravidla, která mají v dané dimenzi obecnější prefix než prefix představovaný daným uzlem.

Toto uspořádání umožňuje nejrychlejší prohledávání bez nutnosti zpětného navracení (časová složitost $O(dW)$, kde d je počet dimenzí a W jejich rozsah v bitech), ovšem za cenu exponenciální paměťové složitosti. Změna datové struktury je také exponenciální, což dělá tento typ prakticky statickým.

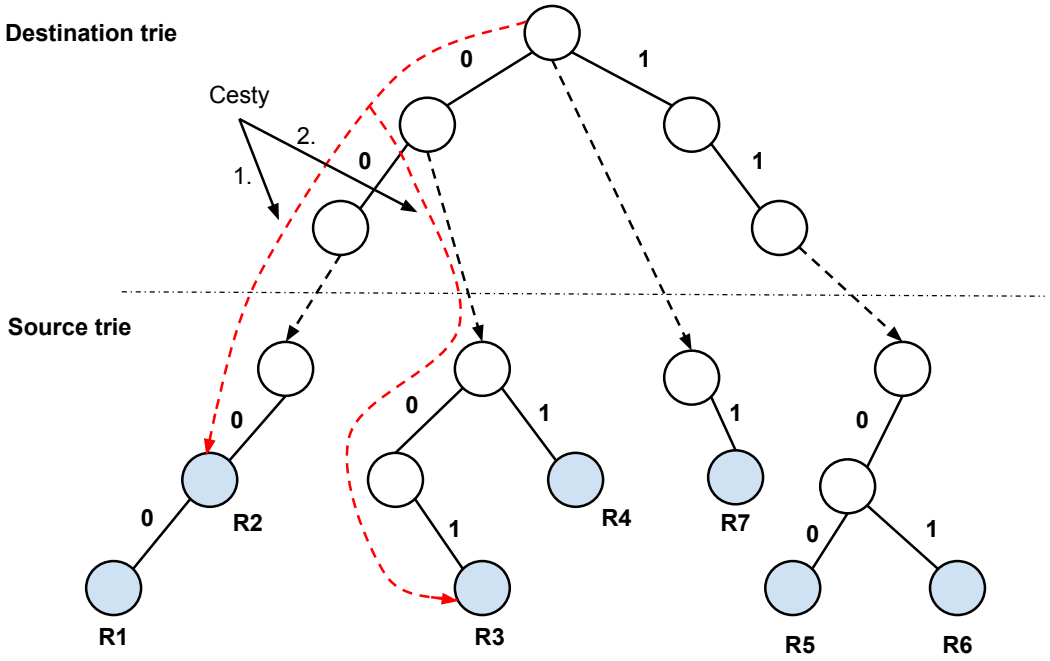


Obr. 3.2: Příklad hierarchických stromů s duplicitními uzly (vyznačené žlutě)

3.5.3 Hierarchické stromy bez duplicit se zpětným navracením

Při sestrojování této struktury pro d dimenzí se nejprve vytvoří první strom tvořený prefixy pravidel v první dimenzi. Ke každému uzlu, který reprezentuje část pravidla v této dimenzi se poté napojí podstrom o $d-1$ dimenzích vytvořený z pravidel, které korespondují s prefixem v první dimenzi. Tento proces se rekurzivně opakuje pro všechny podstromy. Z této konstrukce plyne, že při zanořování nemusíme hned poprvé narazit na aplikovatelné pravidlo. Proto je při neúspěšném vyhledávání nutné prohledat i všechny podstromy napojené k nadřazeným uzlům (tj. podstromy s pravidly, které mají obecnější prefix). To je provedeno technikou zpětného navracení (*backtrackingu*). Více ozřejmuje obrázek 3.3 s vyznačenými cestami vyhledávání. Algoritmus má oproti stromům s duplicitami zvýšenou časovou složitost na $O(d^2W)$,

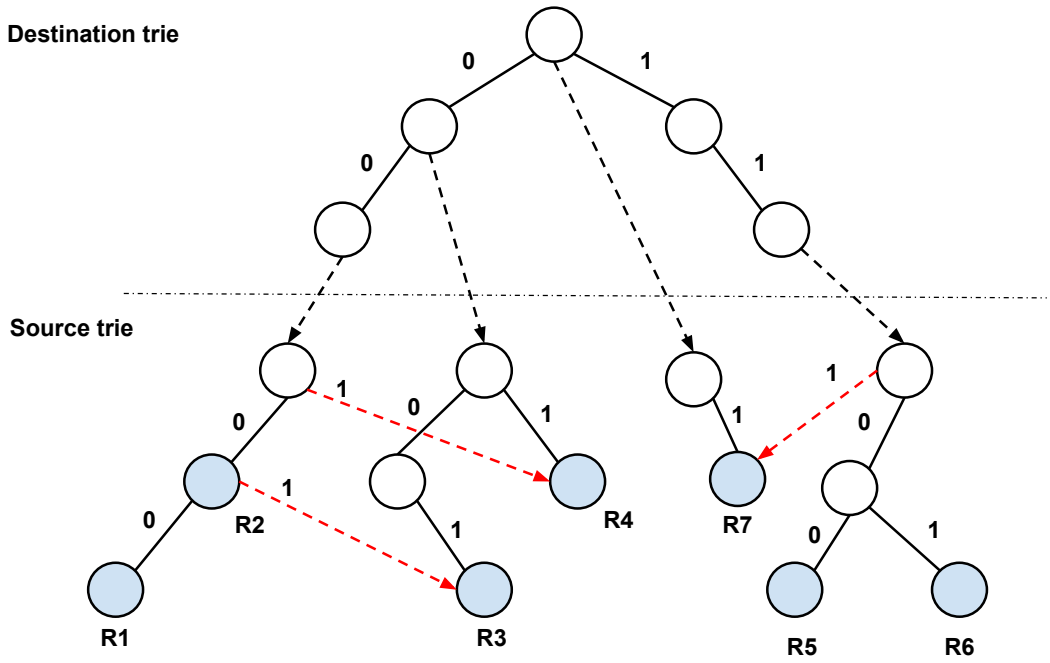
ale prostorová složitost je nižší a přidání nových pravidel výrazně rychlejší (pravidla jsou uložená pouze jednou).



Obr. 3.3: Příklad hierarchických stromů bez duplicit se zpětným navracením. Vyznačené cesty prohledávání pro paket tvořený prefixy (00*,01*).

3.5.4 Hierarchické stromy bez duplicit s použitím ukazatelů

Hierarchické stromy bez duplicit a ukazateli jsou rozšířením předchozího algoritmu, které je určeného pro dvě dimenze (tzv. *grid of tries*). K tomu, aby se snížila časová náročnost daná nutností zpětného navracení, využívá tento přístup použití ukazatelů. Ukazatele jsou sestrojeny od uzlů, kterým chybí potomek, a míří přímo na uzel, ke kterému by se došlo metodou zpětného navracení. Tyto uzly tedy vždy náleží k podstromu napojenému na nadřazený uzel ukazujícího uzlu (více na obrázku 3.4). Optimalizace přináší redukovanou časovou složitost a shodné paměťové nároky jako základní verze s backtrackingem. Využití ukazatelů ovšem komplikují změny ve struktuře stromu, takže je nutné je znovu přepočítávat po každé aktualizaci pravidel.



Obr. 3.4: Příklad stromové struktury *trie* bez duplicit se zpětným navracením a ukazateli (červené šipky)

3.6 Algoritmus založený na bitovém paralelismu (*BitVector*)

Tento algoritmus popsaný v 1 je určen k hardwarové implementaci a rychlé klasifikaci využívající paralelní zpracování v jednotlivých dimenzích. Pracuje s klasifikačními pravidly ve formě rozsahů.

3.6.1 Popis Algoritmu

Před samotnou klasifikací musí dojít k předzpracování (preprocessingu), tj. vytvoření datových struktur. Algoritmus patří mezi statické algoritmy, proto při každé změně klasifikačních pravidel dochází ke kompletnímu přepočítání.
Fáze *preprocessingu*:

1. Pro každou dimenzi $d, 1 \leq j \leq k$, kde k je počet dimenzí, vytvoř množinu disjunktních intervalů z položek všech pravidel v dimenzi d. Těchto intervalů může být maximálně $2n + 1$, kde n je počet pravidel. Tím získáme k množin $I_j, 1 \leq j \leq k$ takovýchto intervalů.

2. Ke každému intervalu z množiny I_j přiřaď bitový vektor délky n, ve kterém se pro každé pravidlo $R_i, i \leq j \leq n$, jež v dané dimenzi j interval překrývá, nastaví příznak 1 na pozici určenou příslušným pravidlem. Předpokládáme, že pravidla jsou seřazena dle priority a proto první příznak v bitovém vektoru značí pravidlo s nejvyšší prioritou.

Předpokládáme, že pakety $P_1, P_2, \ldots, P_m$ přicházejí do systému. Klasifikace probíhá následovně:

1. Pro každou dimenzi najdi takový interval z množiny I_j, do něhož spadá položka paketu P_i v j-té dimenzi. Tento krok je proveden s využitím binárního vyhledávání nebo jiného vyhledávacího algoritmu. Porovnávání v každé dimenzi může být provedeno paralelně. Pro každou dimenzi získáme korespondující bitový vektor.
2. Vytvoř průnik bitových vektorů získaných v předchozím bodě. Toto může být jednoduše provedeno bitovým součinem (operátor AND). Získáme tak bitový vektor, jenž obsahuje příznaky těch pravidel, která lze na pro daný paket uplatnit.
3. Aplikuj pravidlo s nejvyšší prioritou na klasifikovaný paket.

Při hardwarové realizaci máme jednotlivé bitvektory (bitmapy) uložené v paměti o bitové šířce w. Pro získání daných bitvektorů v k dimenzích se musí provést $k*n/w$ paměťových přístupů.

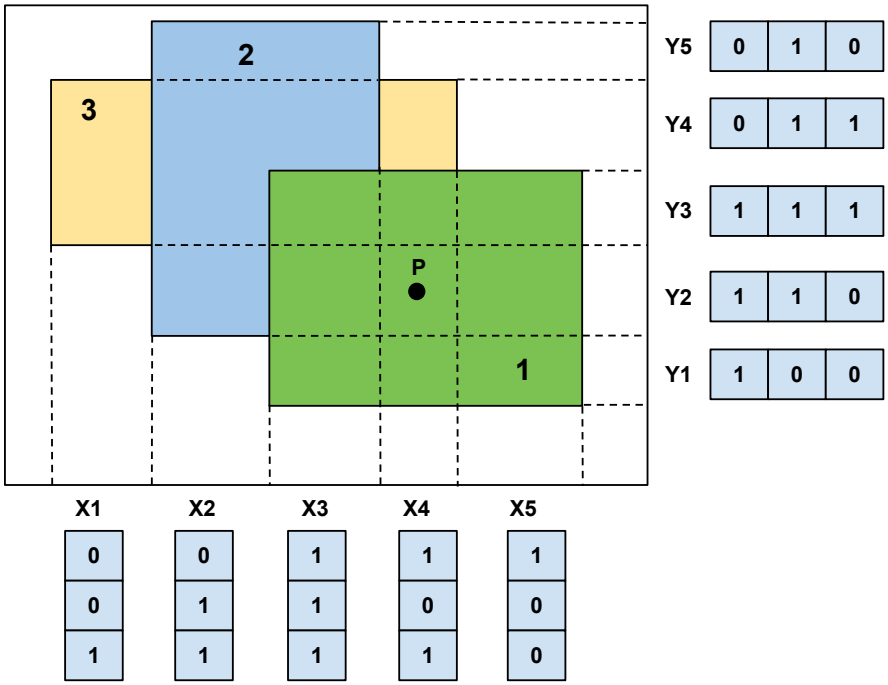
3.6.2 Příklad

Na obrázku 3.5 vidíme příklad jednoduchého klasifikátoru se třemi pravidly, které mají rozsahy ve dvou dimenzích. V první fázi algoritmu došlo k separaci všech nepřekrývajících se intervalů. V ose x to jsou intervaly $X1 \ldots X5$, v ose y intervaly $Y1 \ldots Y5$. Vidíme, že intervaly jsou sestrojeny tak, že končí resp. začínají na hranách jednotlivých pravidel. K intervalům jsou přiřazeny bitové vektory, jejichž šířka je podle celkového počtu pravidel 3, např. interval $X1$ má bitový vektor 001, protože jím prochází pouze pravidlo 3.

Nyní přijde paket P do systému. Paralelně proběhne vyhledávání, které určí, že paket patří do intervalu $X4$ v ose x a do intervalu $Y2$ v ose y. Získáme 2 bitové vektory 101 pro x a 110 pro y. Operací (101 AND 110) spočítáme průnik 100, ze kterého vyplývá, že aplikovaným pravidlem bude pravidlo 1.

3.6.3 Rozšíření algoritmu s využitím ukazatelů

Rozšíření základního algoritmu přináší snížení paměťových nároků za cenu určitého zvýšení časové složitosti. Algoritmus vychází z myšlenky, že při $2n + 1$ interva-



Obr. 3.5: Příklad dvourozměrného klasifikátoru s vyznačenými rozsahy a bitovými vektory

lech se mezi přilehlými intervaly mění bitové vektory pouze v jednom bitu. Proto, pokud máme uloženou bitmapu pouze pro první interval, můžeme zrekonstruovat druhý bitvektor jen s využitím jednoho ukazatele na změněný bit, aniž bychom ho museli mít uložený. Poslední bitmapa se liší o všech n bitů, proto její zrekonstruování potřebujeme aplikovat všech n ukazatelů. Když tuto myšlenku zobecníme, tak bez ohledu na počet intervalů zůstává počet ukazatelů vždy n a z jedné bitmapy můžeme zrekonstruovat bitmapu libovolného intervalu. Prostorová složitost klesne z $O(n^2)$ na $O(n \log n)$, ale počet paměťových přístupů se zvýší až na $(2n \log n)/w$. Abychom zajistili co nejlepší poměr mezi paměťovou a časovou složitostí algoritmu, místo pouze jedné uložené bitmapy, uložíme bitmapy v intervalu daném hodnotou l. Tato hodnota určuje, že jakmile se počet bitů v daném intervalu změní oproti poslední uložené bitmapě o l bitů, uložíme místo ukazatelů opět celou bitmapu.

Shrneme si fázi *preprocessingu*:

1. Pro každou dimenzi vytvoř intervaly stejně jako v základním algoritmu.
2. Ulož bitvektor pro první interval a následně ukládej bitvektory pro ty intervaly, u kterých se počet bitů změnil alespoň o l oproti předchozímu bitvektoru.
3. Mezi intervaly s bitvektory je nyní maximálně $l - 1$ intervalů. Pro ty, které leží v levé polovině, vypočti ukazatele z bitvektoru s menším indexem, pro ty, které v pravé polovině, vypočti ukazatele z bitvektoru s vyšším indexem.

Průběh klasifikace:

1. Nalezení intervalu. Stejné jako v základním algoritmu.
2. Pokud má daný interval uloženou bitmapu, výpočet je dále shodný se základním algoritmem. V opačném případě vypočti bitmapu pro daný interval aplikováním ukazatelů ve všech intervalech mezi daným intervalem a nejbližší bitmapou. Po získání nového bitvektoru výpočet pokračuje jako základní algoritmus.

3.6.4 Paměťové požadavky

Analýzu paměťových požadavků obou algoritmů naleznete v sekci 4.3 v praktické části práce.

3.7 Algoritmus založený na kartézském součinu (*Crossproduct*)

Mezi základní algoritmy patří algoritmus využívající kartézský součin. Byť tento algoritmus je pro svá negativa prakticky nepoužitelný, jeho koncept nachází využití u paměťově efektivnějších algoritmů. Anglický termín *crossproduct* sice označuje vektorový součin, ale z matematického hlediska je korektnější termín kartézský součin, tj. množina všech n-tic složená z unikátních hodnot n množin. Algoritmus umožňuje

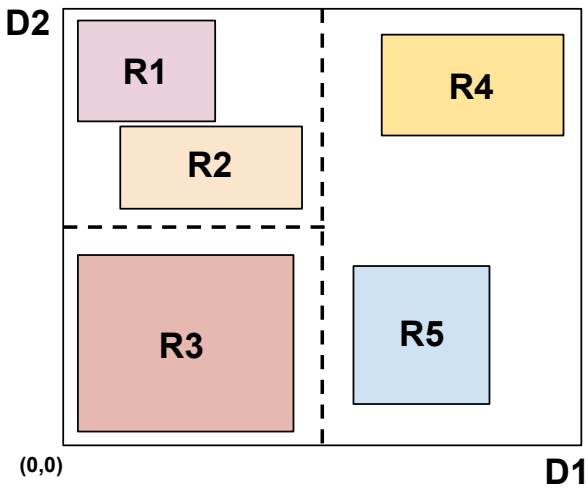
Číslo	D1	D2	Odpovídající pravidlo
1	0*	0*	R1
2	0*	00*	R1
3	0*	01*	R1
4	0*	1*	R4
5	1*	00*	R2
6	1*	01*	R3
7	1*	1*	R4
8	*	1*	R4

Tab. 3.5: Tabulka vytvořená kartézským součinem klasifikátoru 3.1. Záznamy 1,5,6,8 odpovídají pravidlům a záznamy 2,3,4,7 tvoří pseudopravidla.

dekompozici problému a paralelní výpočet. Prvním krokem je prohledávání pro každou dimenzi zvlášť. Typ tohoto prohledávání může být odlišný podle typu dimenze. Pro prohledávání prefixů se nejčastěji používá struktura *trie* (3.5.1) a výsledkem

je nalezení nejdelšího shodného prefixu (LPM). Při prohledávání se mohou použít hešovací tabulky pro přesné porovnávání nebo algoritmus pro porovnávání rozsahů. Tyto prohledávání mohou být provedeny paralelně a získáme tak pro každou dimenzi výsledné hodnoty. Nalezené hodnoty poté slouží jako klíč, pomocí kterého se s využitím hešovací funkce získá index do tabulky, kterou tvoří všechny součiny daného klasifikátoru a k nim přiřazená pravidla. Každý součin odpovídá kombinaci unikátních hodnot všech dimenzí, kterým se dá přiřadit aplikovatelné pravidlo. Je zřejmé, že kromě samotných pravidel tabulka obsahuje i položky s hodnotami, které žádné pravidlo netvoří, ale nějakému pravidlu odpovídají. Tyto záznamy se nazývají *pseudopravidla*. Jak vidíme u takto zkonstruované tabulky 3.5, největší bolestí algoritmu jsou obrovské paměťové nároky kvůli počtu pseudopravidel. V krajním případě může velikost tabulky dosáhnout až exponenciální složitosti $O(N^d)$. Stejně tak aktualizace pravidel vyžaduje kompletní přepočítání, proto je tento algoritmus použitelný pouze pro malé a statické filtry. Největší výhodou algoritmu je naopak jeho rychlost, která závisí pouze na době prohledávání v jednotlivých dimenzích. Poté stačí pouze jeden přístup do paměti (do tabulky) k získání výsledku.

3.8 HiCuts algoritmus

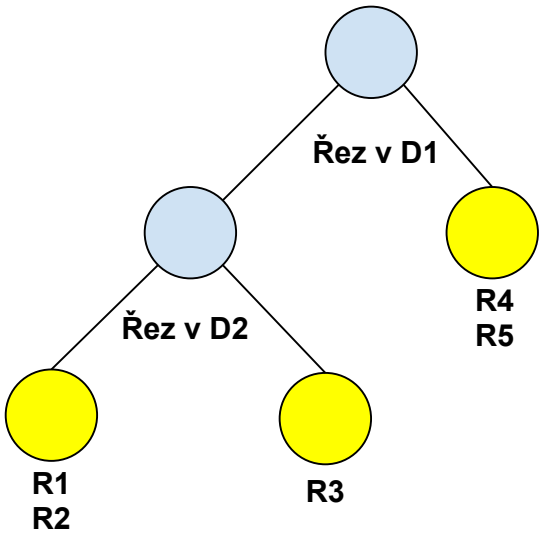


Obr. 3.6: Příklad klasifikátoru s vyznačenými řezy. Práh HiCuts algoritmu nastavený na 2.

HiCuts (*Hierarchical Intelligent Cuttings* 8) je algoritmus založený na rozhodovacích stromech. Vzhledem k tomu, že nalezení optimálního rozhodovacího stromu patří mezi NP-úplné problémy, k jeho sestavení se využívá heuristických algoritmů. Ve fázi preprocessingu HiCuts sestavuje rozhodovací strom postupným dělením

prostoru podle vyhodnocených dimenzí. Kořenový uzel stromu představuje prohledávaný prostor celého klasifikátoru. Jeho přímí potomci pak část prostoru, která vznikla rozdělením podle dané dimenze (provedení řezů). Tyto řezy jsou rekurzivně prováděny dokud se počet pravidel překrývající daný prostor či obsažených v tomto prostoru nesníží pod úroveň nastaveného prahu (*bin-threshhold*). Takovýto uzel se stává listem se seznamem pravidel, které se při klasifikaci již mohou porovnávat sekvenčně. Každý uzel ve vybudovaném stromu má uložený rozsah všech svých dimenzí, tj. jaký prostor představuje. Dále má určenou dimenzi, ve které bude proveden řez, a také hodnotu určující kolik potomků bude mít, tzn. na kolik částí bude prostor rozdělen (řezy jsou rovnoměrné). Stromy budované s větší šířkou, tedy s větším počtem řezů v každém uzlu, budou mít rychlejší prohledávání, ale paměťové nároky vzrostou. Druhým významným parametrem, který ovlivňuje tvorbu stromu, je hodnota prahu.

Obrázek 3.6 ukazuje dvourozměrný klasifikátor s pěti pravidly a k němu vybudovaný rozhodovací strom na obrázku 3.7. Hodnota prahu je 2 a dělení prostoru probíhá na 2 části. Nejprve proběhne řez v dimenzi D1 (osa x). V pravé části jsou pouze dvě pravidla R4,R5, takže daný uzel se již nedělí a stává se listem. V levé části se prostor dále dělí v dimenzi D2 (osa y). Počet pravidel kolidujících v těchto prostorech je menší nebo roven nastavenému prahu, takže je tvorba stromu ukončená.



Obr. 3.7: Příklad rozhodovacího stromu klasifikátoru 3.6

3.9 DCFL algoritmus

DCFL (*Distributed Crossproducting of Field Labels* 6) je moderní algoritmus založený na dekompozici a kombinující několik přístupů. Vychází ze základního *Crossproduct* algoritmu, ale oproti němu výrazně redukuje paměťové nároky, díky čemuž může prakticky pracovat s paketovými filtry o vysokém počtu pravidel. Algoritmus rozděluje činnost do několika fází. V první je provedeno nezávislé prohledávání v jednotlivých dimenzích (LPM), což je stejné jako u základního *Crossproduct* algoritmu. V této fázi se pro každou dimenzi zvlášť uchovávají unikátní položky pravidel, např. množina F_1 pro první dimenzi atd. V reálných klasifikátorech je počet unikátních hodnot v jednotlivých dimenzích výrazně menší než celkový počet pravidel, takže paměťové nároky pro toto uložení nejsou vysoké. Zároveň se k těmto položkám ukládá kolik pravidel je obsahuje. Při přidání nového pravidla, které má v konkrétní dimenzi již uloženou hodnotu, se pouze zvýší tento počet o jedna. Stejně tak při mazání pravidel se počet v konkrétních položkách sníží o jedna a pokud dosáhne nuly, položka se může smazat. Položky v jednotlivých dimenzích jsou indexovány tzv. *labely*.

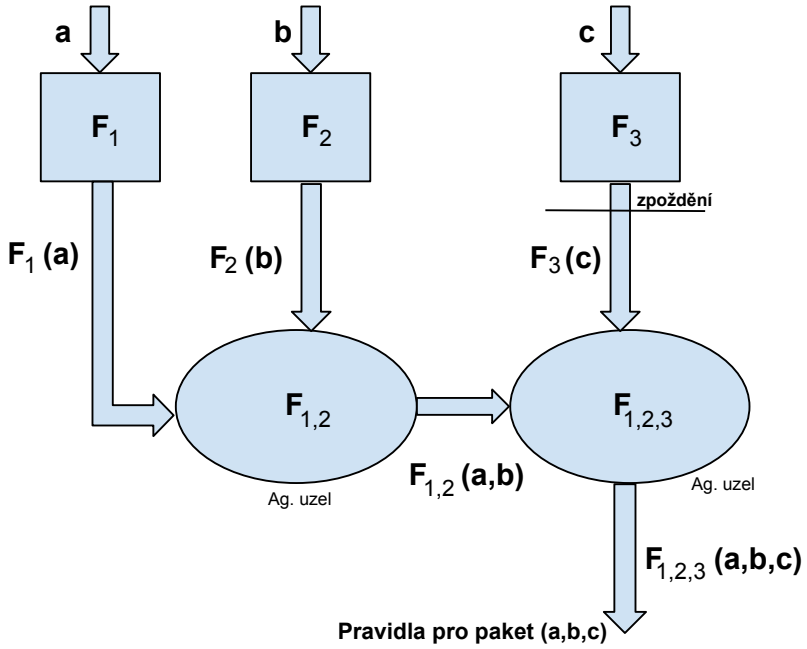
Další fází algoritmu je takzvaná **agregační síť**. Agregační síť je tvořena **agregačními uzly**, které slučují unikátní hodnoty pravidel v několika dimenzích. Např. uzel $F_{1,2}$ obsahuje všechny unikátní páry hodnot obsažené dimenzích $d1$ a $d2$ daného klasifikátoru. Agregované hodnoty v $F_{1,2}$ představují podmnožinu kartézského součinu množin F_1 a $F2$.

Na obrázku 3.8, který znázorňuje agregační síť pro tři dimenze, můžeme vidět, jak probíhá klasifikace paketu s hodnotami (a, b, c). Po vyhledání v prvních dvou dimenzích vstupují do agregačního uzlu $F_{1,2}$ množiny hodnot $F_1(a)$ a $F_2(b)$, což jsou všechny prefixy pokrývající daný paket v první resp. v druhé dimenzi. Provedením kartézského součinu nad těmito množinami získáme tzv. *query set*, tj. množinu, kde pro všechny její prvky testujeme příslušnost v množině $F_{1,2}$. Všechny prvky splňující tuto příslušnost přidáváme do množiny $F_{1,2}(a, b)$, která tvoří výstup daného uzlu. $F_{1,2}(a, b)$ představuje položky pravidel v dimenzích $d1$ a $d2$, které pokrývají vstupní paket v těchto dimenzích. Stejný proces probíhá u druhého uzlu $F_{1,2,3}$, kde *query set* tvoří součin párů v $F_{1,2}(a, b)$ s prefixy v $F_3(c)$. Výsledkem prohledávání budou trojice $F_{1,2,3}(a, b, c)$ představující aplikovatelná pravidla. Poslední fází je zjištění pravidla s nejvyšší prioritou.

Před samotnou klasifikací je nutné vybudovat agregační síť s důrazem na co nejvyšší efektivitu. Síť je nutné sestavit tak, aby byl maximální *query set* v jednotlivých uzlech co nejmenší. Maximální mohutnost této množiny je dána hloubkou zanoření prefixů v dimenzích, resp. maximálním počtem překrytí rozsahů. V reálných klasifikátorech je ale toto zanoření poměrně malé, takže provedené kartézské součiny

produkují nesrovnatelně méně kombinací než součin provedený nad všemi dimenzemi v jednom kroku u základního algoritmu. Další záležitostí je realizace samotného agregačního uzlu, aby bylo dotazování se na příslušnost prvků z *query setu* v množině uzlu co nejefektivnější. Jednou z možností je použití datové struktury *bloom filter* pro testování příslušnosti prvků na množině. Dalším přístupem je indexování do pole seznamů pomocí tzv. *meta labelů*, které představují již agregované hodnoty vstupující do daného uzlu. *Meta-label* má přiřazený seznam *labelů*, reprezentující hodnoty v další dimenzi, se kterými je možné provést výstupní agregaci.

Algoritmus DCFL má při vhodně vytvořené agregační síti vysokou rychlost prohledávání i nízké paměťové nároky. Umožňuje rychlé přidávání i mazání pravidel, aktualizace datových struktur je provedena jako atomická operace během samotné klasifikace.



Obr. 3.8: Příklad agregační sítě pro 3 dimenze

3.10 RFC algoritmus

Algoritmus RFC (*Recursive Flow Classification*) je algoritmus kombinující přístupy algoritmů *Crossproduct* 3.7 a *BitVector* 3.6. Vychází z ideje, že pro klasifikaci paketů, kde S je délka všech klasifikovaných položek v bitech, bychom mohli mít tabulku o velikosti 2^S. Každý příchozí paket by sloužil jako přímý index do této

hash tabulky a výsledkem by bylo odpovídající pravidlo. Klasifikace by sice požadovala pouze jeden paměťový přístup pro každý paket, ale paměťového nároky by byly neúnosně vysoké. RFC ve snaze redukovat paměťovou složitost tento přístup rozčleňuje do více fází a buduje prohledávací síť v využitím tzv. **tříd ekvivalence**.

V počáteční fázi probíhá paralelní prohledávání pro každou dimenzi. Každá dimenze šířky W má svou RFC tabulku o velikosti 2^W. Protože by i tyto tabulky mohly být příliš velké, rozdělují se dimenze do tzv. *chunků*, např. dimenze zdrojové IP adresy o šířce 32 bitů se rozdělí na 2 *chunky* pro horních a dolních 16 bitů. RFC tabulka přiřazuje každé možné hodnotě v dimenzi třídu ekvivalence. Stejná třída ekvivalence určuje, že všechny hodnoty z této třídy pokrývá stejná množina pravidel, tj. jedná se o všechny intervaly, resp. n-tice intervalů, které mají shodný bitový vektor (bitmapu). Třídy ekvivalence mají při budování RFC sítě uloženou svou bitmapu, která slouží k vyhodnocování tříd ekvivalence v dalších fázích (složených z více dimenzí/chunků). Složené třídy ekvivalence získané v první fází slouží jako indexy do tabulek v další fázi. Kombinací vstupních tříd ekvivalence a s využitím bitového součinu daných bitmap se určují nové třídy ekvivalence pro složené dimenze/chunky v dalších fázích (všechny kombinace, které mají stejnou bitmapu, vytvoří jednu třídu). V závěru celé sítě je tabulka, která agreguje celý prohledávací prostor, a pomocí přiřazených bitmap se vyhodnotí výsledné pravidlo pro daný paket.

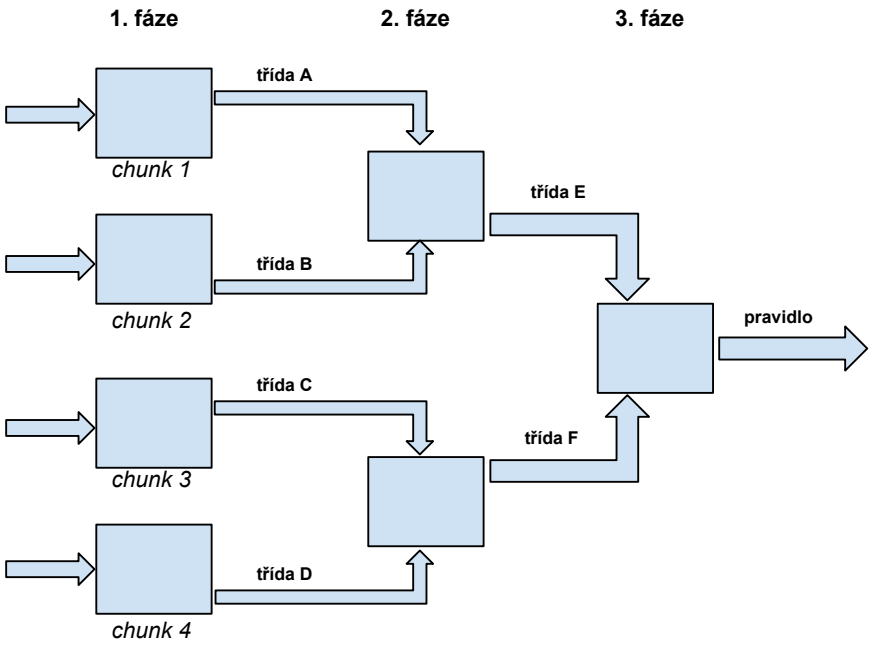
RFC algoritmus oproti DCFL nepodporuje dynamické změny pravidel a má vyšší paměťové nároky. V porovnání s *BitVector* algoritmem má vyšší rychlost prohledávání. Stejně jako u DCFL, výkonnost algoritmu ovlivňuje konfigurace prohledávací sítě. Proto se hledá takové nastavení, aby v další fázi vzniklo co nejméně tříd ekvivalence. Další faktorem je počet fází. Rozdělení do většího počtu sice snižuje paměťové nároky, ale rychlost prohledávání se snižuje, protože vzroste počet přístupů do paměti.

3.11 Další algoritmy

Součástí frameworku Netbench je celá řada dalších klasifikačních algoritmů, jejichž podrobný popis není součástí této práce.

3.11.1 MSCA - *Multi Subset Crossproduct Algorithm*

Algoritmus MSCA 7 stejně jako DCFL modifikuje základní *crossproduct* algoritmus s cílem snížit paměťové nároky. Používá heuristiky k rozdělení pravidel do několika podmnožin, v rámci kterých je provedeno prohledávání (LPM). Podmnožiny jsou sestrojeny tak, aby se co nejvíce zredukoval počet pseudopravidel. Nalezené



Obr. 3.9: Příklad sítě RFC algoritmu. Vstupní paket rozdělen podle 4 chunků/dimenzí. Získané třídy ekvivalence v první fázi slouží jako indexy do tabulky v druhé fázi. Z nalezené třídy ekvivalence v poslední fázi je určeno výsledné pravidlo.

hodnoty v různých podmnožinách slouží jako index do hešovací tabulky. K urychlení klasifikace se využívá *bloom filter*.

3.11.2 PHCA - *Perfect Hashing Crossproduct Algorithm*

PHCA 11 vylepšuje MSCA algoritmus. Používá speciálně sestrojenou tzv. perfektní hešovací funkci (bezkolizní). Kombinace prefixů získané v první fázi vyhledavaní (LPM) jsou pomocí této funkce namapovány přímo do paměti s výsledným pravidlem. Oproti MSCA výrazně redukuje paměťové nároky.

3.11.3 PCCA - *Prefix Coloring Classification Algorithm*

Algoritmus PCCA dále vylepšuje PHCA. Ve fázi LPM prohledávání rozšiřuje jednotlivé prefixy o abstraktní barvu, kterou mají přiřazenou. Prefixy dále obsahují bitmapu složenou z dovolených a potlačených barev jiných dimenzí. S použitím jednoduché logiky se může vyfiltrovat většina nechtěných kombinací z LPM prohledávání, čímž se sníží paměťové nároky zatímco rychlost zůstává zachována.

3.11.4 MSPCCA

Kombinace MSCA a PCCA, která ještě dále snižuje paměťové nároky, přičemž rychlost zůstává stejná jako u PCCA.

4 PRAKTICKÁ ČÁST: IMPLEMENTACE ALGORITMU *BITVECTOR*

Poslední kapitola této práce se zabývá mou praktickou činností, kterou byla implementace dvou algoritmů založených na bitových vektorech od autorů T.Lakshmana a D.Stiliadise 1 (popis v sekci 3.6).

4.1 Netbench

Implementace obou zadaných algoritmů byla realizována pomocí experimentálního frameworku **Netbench**. Netbench slouží jako nezávislá platforma pro rychlou implementaci a testování algoritmů a jejich srovnávání s jinými přístupy. Framework je zaměřen na algoritmy k prohledávání nejdelších prefixů (*prefix matching*), na algoritmy pro klasifikaci paketů a algoritmy pracující s regulárními výrazy (*pattern matching*).

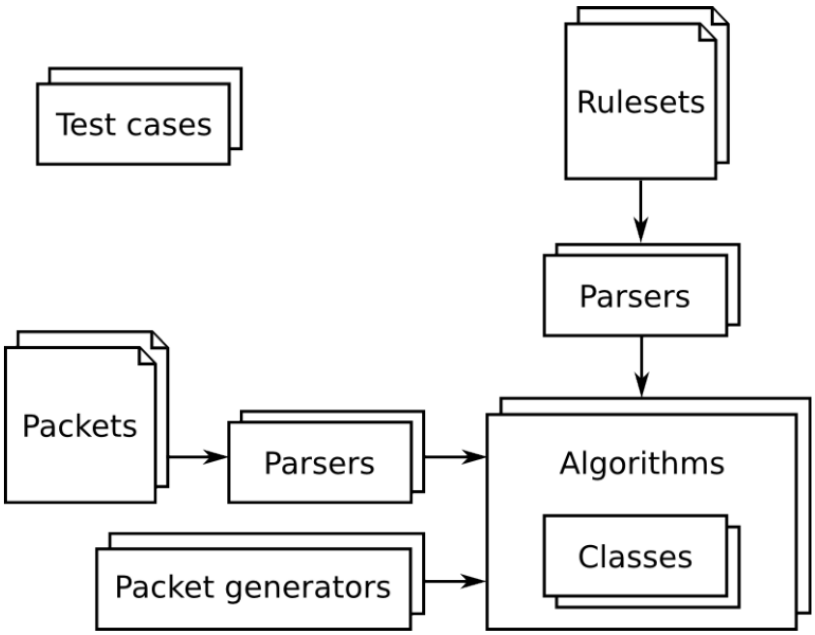
Netbench je objektově orientovaný framework implementovaný v dynamicky typovaném a interpretovaném jazyce Python (verze 2.6), který pro svou jednoduchou syntax umožňuje programátorům rychlé zorientovaní v problematice síťových algoritmů a celkovém návrhu knihovny (4.1). Pro testování klasifikačních algoritmů framework obsahuje syntetické množiny pravidel vygenerované nástrojem Classbench, jejichž rozsah se pohybuje mezi 100 až 1000 pravidly. Jak ukazuje obrázek 4.1 s celkovou architekturou, Netbench používá parsery pro zpracovaní sad pravidel i pro soubor s pakety v *pcap* formátu k převedení do příslušných datových struktur.

4.1.1 Klasifikace paketů

Část knihovny určená k experimentům s klasifikačními algoritmy je obsažená ve složce `netbench/classification`. Obsahuje syntetické sady pravidel (složka `rulesets`), parsery pravidel (složka `parsers`), základní třídy a implementace samotných algoritmů (složka `algorithms`). Základní strukturu tříd zobrazuje obrázek 4.2.

Základní třídy:

- `RuleSet` - Třída reprezentující sadu pravidel.
- `Rule` - Třída reprezentující jedno pravidlo.
- `PrefixSet` - Třída pro reprezentaci několika prefixů. U pravidla definovaného prefixem obsahuje pouze jeden prefix, u pravidel daných rozsahy obsahuje složené prefixy.
- `Prefix` - Třída představující jeden prefix.



Obr. 4.1: Architektura Netbench frameworku (zdroj: 10)

Implementované algoritmy dědí od bázové abstraktní třídy `BClassification`. Třída obsahuje tři základní metody, které musejí odvozené třídy implementovat:

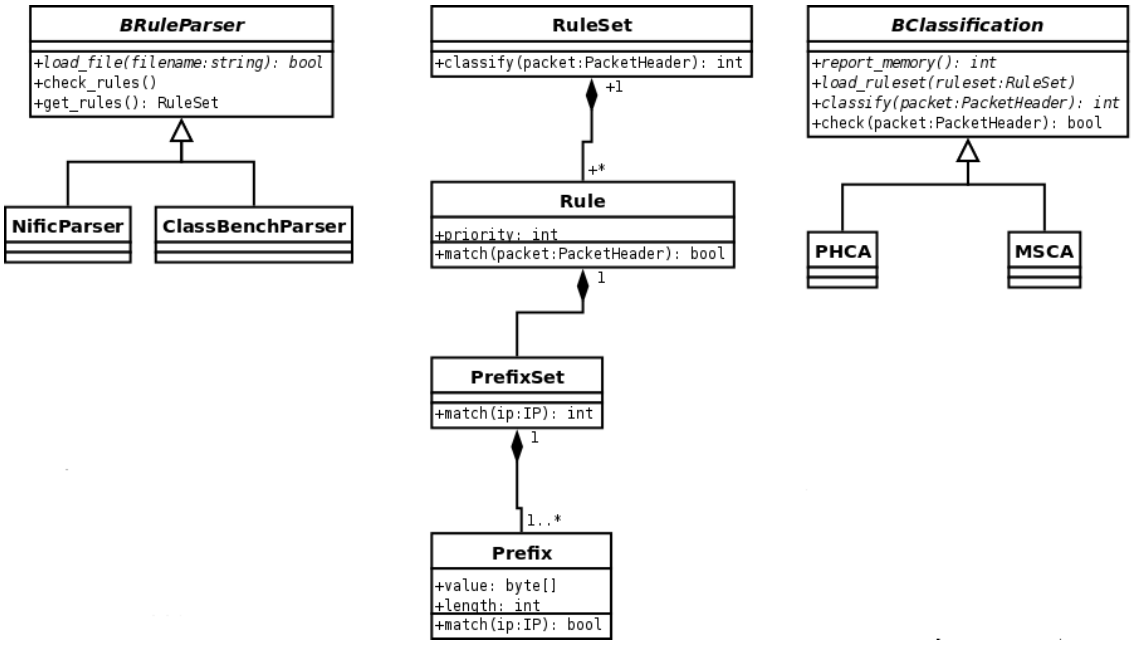
- `load_ruleset()` - Slouží ke zpracování pravidel a vytvoření potřebných datových struktur daných návrhem algoritmu. Jako vstupní parametr přijímá instanci třídy `RuleSet`.
- `classify()` - Klasifikuje daný paket pomocí vytvořených datových struktur a vrací odpovídající pravidlo podle priority.
- `report_memory()` - Vypíše informace o paměťových nárocích algoritmu v závislosti na použitém klasifikátoru.

4.2 Implementace

Oba algoritmy implementují rozhraní abstraktní třídy `BClassification` a jsou obsažené ve složce `netbench/classification/algorithms` frameworku Netbench spolu s ostatními algoritmy:

- `bva.py` - Modul obsahující třídu BVA, která realizuje základní *BitVector* algoritmus.
- `bva2.py` - Modul obsahující třídu BVA2, která realizuje rozšíření základního algoritmu.

Oba algoritmy používají další nově vytvořené třídy:



Obr. 4.2: Diagram tříd pro tvorbu a testování klasifikačních algoritmů (zdroj: 10)

- `Range` - Třída reprezentující rozsah hodnot v zadané dimenzi. Tvoří ji tři atributy: `domain_size` pro šířku dimenze v bitech, rozsah tedy nesmí překročit meze dané touto dimenzí, atribut `begin` určující počátek intervalu a `end` určující konec intervalu. Kromě atributů třída obsahuje dvě metody: `display()` pro tisk rozsahu a `covers()`, která vrací `True`, jestliže interval překrývá interval daný vstupním parametrem.
- `RangeBV` - Třída, která dědí od `Range` a která rozšiřuje daný rozsah o bitový vektor (atribut `bitvector`), resp. o ukazatele v `BVA2` algoritmu (atribut `pointers`). V BVA se používá pouze atribut `bitvector`, u BVA2 pak vždy jeden z těchto dvou atributů podle toho, co je danému rozsahu přiřazeno. K tomuto rozlišení slouží metody `has_bitvector()` a `has_pointers`. Ukazatele (`pointers`) jsou realizovány jako asociativní pole (slovník), kde klíče tvoří indexy bitvektoru na změněné bity oproti předchozímu intervalu.

4.2.1 BVA - implementace základního algoritmu

Popis metod třídy `BVA`:

- `load_ruleset()` - Vstupním parametrem je `ruleset`, který se zkopíruje do interního atributu, odstraní se duplikáty pravidel a doplní případná univerzální pravidla. Následně se z rulesetu separují pro jednotlivé dimenze hodnoty pravidel ve formě prefixů a převedou se na intervaly (instance třídy `Range`). Všechny intervaly se uloží do asociativního pole `_rules` (klíčem je dimenze a hodnotou

pole instancí `Range`). Na konci metody se zavolá metoda `_process_range` pro další zpracování.

- `_process_ranges()` - Pomocná metoda, kterou volá `load_ruleset` a která pro rozsahy v jednotlivých dimenzích (v as. poli `_ranges`) volá `_create_bitvectors`. Vrácené seznamy intervalů s bitvektory jsou ukládány v as. poli `_ranges` (klíč: dimenze, hodnota: seznam instancí `RangeBV`). Tato metoda je překryta v `BVA2`.
- `_prefixes_to_range()` - Pomocná metoda , kterou volá `load_ruleset`. Z několika spojitých prefixů vrátí začátek a konec intervalu.
- `_create_bitvectors()` - Přijímá všechny rozsahy pravidel v dané dimenzi a její šířku v bitech. Celou dimenzi následně rozděluje na množinu vzájemně spojitých intervalů, které mají meze v místech kde pravidla začínají či končí. Tyto intervaly reprezentují instance třídy `RangeBV`, které metoda vrací v seznamu.
- `classify()` - Klasifikuje vstupní paket. Pomocí metody půlení intervalů (bisekce) nalezne v jednotlivých dimenzích intervaly, do kterých paket spadá. Provede bitový součin nad korespondujícími bitovými vektory a vrátí pravidla podle priority.
- `report_memory()` - Vypíše informace o paměťových nárocích algoritmu. Více o analýze paměťových nároků v 4.3.

4.2.2 BVA2 - implementace rozšířeného algoritmu

Třída BVA2 je potomkem třídy BVA. U některým metod používá implementaci svého rodiče, jiné překrývá a také obsahuje několik pomocných metod:

- `load_ruleset()` - Metoda volá rodičovskou metodu.
- `_process_ranges()` - Překrývá rodičovskou metodu. Nejprve volá pro všechny dimenze `_create_intervals` pro vytvoření spojitých intervalů. Poté volá `_create_bitvectors_and_pointers` pro vytvoření bitových vektorů nebo ukazatelů.
- `_create_intervals()` - Rozdělí dimenzi na spojité intervaly.
- `_create_bitvectors_and_pointers()` - V určitých intervalech podle vypočítané hodnoty l vytvoří bitové vektory, k ostatním intervalům pak vypočítává ukazatele na změněná pravidla.
- `classify()` - Klasifikuje vstupní paket. Pomocí metody půlení intervalů nalezne v jednotlivých dimenzích intervaly, do kterých paket spadá. Pokud má vyhledaný interval přiřazené ukazatele, dopočítá bitový vektor z nejbližšího uloženého bitvektoru a všech dalších ukazatelů mezi těmito intervaly. Provede bitový součin nad získanými bitovými vektory a vrátí pravidla podle priority.

- `report_memory()` - Vypíše informace o paměťových nárocích algoritmu. Více o analýze paměťových nároků v 4.3.

4.3 Analýza paměťových požadavků

4.3.1 BVA

U základního algoritmu musíme ukládat meze všech intervalů v jednotlivých dimenzích a jejich bitové vektory. Paměťové nároky pro uložení všech mezí se spočítají podle vztahu

$$\sum_{d=1}^{dims} 2W_d c_d$$

, kde W_d je šířka v bitech dimenze d a c_d je počet intervalů v dimenzi d. Paměťové nároky pro uložení všech bitových vektoru můžeme vyčíslit podle vztahu

$$\sum_{d=1}^{dims} nc_d$$

, kde n je délka bitových vektorů (resp. počet pravidel) a a c_d je počet intervalů v dimenzi d. Celkové paměťové požadavky algoritmu BVA můžeme tedy vyjádřit jako:

$$\sum_{d=1}^{dims} (2W_d c_d + nc_d)$$

.

Paměť potřebná pro uložení mezí má přinejhorším lineární prostorovou složitost $O(n)$. Celková paměť pro bitové vektory závisí jak na celkovém počtu pravidel (délce vektoru), tak na počtu všech intervalů v dimenzích. V nejhorším případě můžeme získat při n pravidlech $2n + 1$ různých intervalů, což dává vztah $n(2n + 1)$ pro potřebnou paměť v dané dimenzi. Tj. algoritmus má kvadratickou horní paměťovou složitost $O(n^2)$.

4.3.2 BVA2

Stejně jako u BVA musíme ukládat meze všech intervalů podle výše uvedeného vztahu. Algoritmus snižuje paměťové nároky ukládáním bitvektorů pouze v některých intervalech. V ostatních intervalech jsou uložené pouze ukazatele na změněné bity mezi jednotlivými intervaly. Důležitá je proto hodnota l, která definuje, jak často se budou ukládat celé bitvektory. Tato hodnota určuje, že po právě l změněných bitech od posledního uloženého bitvektoru se opět uloží celý bitvektor. Všechny intervaly mezi doplní ukazatele. Jeden ukazatel bude mít velikost $\log_2 n$ bitů, kde n je počet pravidel. Pokud bychom uložili pouze první bitvektor a v ostatních měli

ukazatele ($l = 2n+1$), paměťová složitost se zredukuje až na $O(n \log_2 n)$, ale budeme potřebovat až $2n$ paměťových přístupů k rekonstrukci bitvektoru.

Autoři algoritmu 1 navrhují pro hardwarovou implementaci umístit celé bitvektory do paměti na čipu, kde šířka této paměti bude α-krát větší než paměť mimo čip, kam se uloží ukazatele. Počet přístupů nutných k získání celého bitvektoru na čipu bude $t = n/\alpha w$ a počet přístupů k získání $l/2$ ukazatelů (víc není potřeba) mimo čip bude $t = l \log_2 n/2w$. Aby byl počet přístupů roven (tj. trvaly stejný čas) získáváme doporučený vztah pro určení hodnoty l:

$$l = \frac{2}{\alpha} \frac{n}{\log_2 n}.$$

4.3.3 Testování

Pravidla	Počet	Unikátní hodnoty (Zdrojová IP/Cílová IP/Zdrojový port/Cílový port/Protokol)
fw2_05_m05_100	99	75/37/8/1/1
fw2_05_m05_250	219	55/53/9/1/1
fw2_05_m05_500	394	65/57/9/1/1
fw2_05_m05	947	746/307/9/1/1

Tab. 4.1: Charakteristika použitých sad pravidel po odstranění duplikátů

Pro testování celkových paměťových požadavků algoritmů jsem použil několik sad pravidel. Charakteristiku sad zobrazuje tabulka 4.1. Jednotlivé sady se liší počtem pravidel a počtem unikátních hodnot v dimenzích.

Celková spotřeba paměti pro jednotlivé sady je zobrazena v tabulce 4.2, která srovnává paměťové požadavky algoritmů BVA a BVA2 společně s dalšími algoritmy z Netbench. Tabulka dokladuje, že spotřeba paměti BVA2 je výrazně menší než u BVA. S rostoucím počtem pravidel se rozdíl v paměťových nárocích dále prohlubuje.

Pravidla	**BVA**	**BVA2**	DCFL	MSCA	PHCA	PCCA
fw2_05_m05_100	31	16	15	200	62	23
fw2_05_m05_250	42	20	15	1 026	265	169
fw2_05_m05_500	65	30	25	1 928	480	332
fw2_05_m05	2 074	185	85	44 846	63 108	14 590

Tab. 4.2: Srovnání celkových paměťových požadavků různých algoritmů z Netbench v kilobitech

5 ZÁVĚR

Cílem mé práce bylo nastudování problematiky algoritmů pro klasifikaci paketů a implementace vybraného algoritmu. Nejprve jsem se věnoval praktickým oblastem, kde nachází klasifikace paketů využití. Nastudoval jsem teoretickou rovinu klasifikace a popsal různé typy porovnávání. Shrnul jsem požadavky na klasifikační algoritmy určené k filtrování provozu v počítačových sítích.

Největší část práce obsahuje popis vybraných algoritmů. Vysvětlil jsem jejich činnost a zabýval se jejich paměťovými nároky a rychlostí. V této části také popisuji k implementaci vybraný algoritmus založený na bitovém paralelismu a bitových vektorech (*BitVector*). Rozebral jsem činnost obou jeho variant.

V poslední fázi mé práce jsem se seznámil s experimentálním frameworkem pro testování algoritmů - Netbench. Podařilo se mi implementovat oba vybrané algoritmy, úspěšně je otestovat a integrovat do nové verze Netbench. V práci jsem popsal všechny fáze implementace a navržené datové struktury. V závěrečné části pak zhodnocuji paměťové nároky obou algoritmů, které jsem testoval pro několik klasifikátorů, a srovnávám je s ostatními algoritmy z Netbench.

LITERATURA

[1] Lakshman T.V., Stiliadis D. High-Speed Policy-base Packet Forwarding Using Efficient Multi-dimensional Range Matching. Bell Laboratories, 101 Crawfors Corder Rd. Holmdel, NJ 07733.

[2] Gupta Pankaj, McKeown Nick. Algorithms for Packet Classification. Computer Systems Laboratory, Stanford University Stanford, CA 94305-9030.

[3] Gupta Pankaj. Algorithms for Routing Lookups and Packet Classification. The Department of Computer Science and The Committee on Graduate Studies of Stanford University. December 2000.

[4] Strebe Matthew, Perkins Charles. Firewally a proxy-servery Praktický průvodce. Computer Press. Brno 2003.

[5] Ingham Kenneth, Forrest Stephanie. A History and Survey of Network Firewalls. University of New Mexico, Computer Science Department. 2002.

[6] Taylor D.E., Turner J.S. Scalable Packet Classification using Distributed Crossproducting of Fields Labels. Applied Research Laboratory, Washington University in Saint Louis. 2005.

[7] Dharmaputikar S., Haoyu S., Turner J., Lockwood J. Fast Packet Classification Using Bloom filters. Department of Computer Science and Engineering, Washington University in Saint Louis. December 2006.

[8] Gupta Pankaj, McKeown Nick. Packet Classification using Hierarchical Intelligent Cuttings. Computer Systems Laboratory, Stanford University Stanford, CA 94305-9030.

[9] ANT@FIT research group. Netbench: Framework for Evaluation of Packet Processing Algorithms, Technical report. Faculty of Information Technology, Brno University of Technology. March 10, 2012.

[10] Puš V., Košař V., Tobola J. Netbench-Design Document. Faculty of Information Technology, Brno University of Technology. Brno 2012.

[11] Puš V., Kořenek J.: Fast and Scalable Packet Classification Using Perfect Hash Functions. ACM/SIGDA Symposium on Field Programmable Gate Arrays, New York, 2009.

SEZNAM PŘÍLOH

K práci je přiloženo CD, které obsahuje technickou zprávu, všechny zdrojové kódy a dokumentaci.