VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

FILTR IP TOKŮ

IP FLOW FILTER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **IMRICH ŠTOFFA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN WRONA,**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20008/2016/xstoff02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Štoffa Imrich**
Obor: Informační technologie
Téma: **Filtr IP toků**
IP Flow Filter
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s protokolem IPFIX a kolektorem IPFIXcol.
2. Navrhněte modul pro efektivní filtraci záznamů o IP tocích pro kolektor.
3. Navržený modul implementujte jako softwarovou knihovnu. Zaměřte se na podporu IPFIX i nfsen jazyka filtru.
4. Vytvořenou implementaci důkladně otestujte a změřte výkonnostní parametry.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Wrona Jan, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Predmetom tejto práce je snaha o zjednotenie filtračných jazykov používaných v kolektore IP tokov a v knižnici pre ich analýzu. Momentálne tieto programy používajú rozdielne filtračné moduly a formáty súborov, z čoho plynú isté nezrovnalosti. Návrh filtra vznikol ako súčasť práce na zlepšení integrácie týchto dvoch programov a to poskytnutím implementácie spoločného filtračného modulu podporujúceho populárny filtračný jazyk. Obsahuje teoretický úvod do monitorovania tokov v sieti, zmieňuje aj algoritmy z prostredia klasifikácie tokov použiteľné pri vyhodnocovaní podmienok nad záznamami. Vlastná práca sa venuje návrhu a implementácii filtračného modulu spolu s adaptérmi k spomínanému kolektoru a knižnici. Záver je venovaný vyhodnoteniu výkonu a funkčnosti filtra.

Abstract

This thesis is focused on unification of filtering languages used by IP flow collecting program and library for their analysis. At the moment these implementations use different filtering modules and file formats. Because of this, inconsistencies in results arise and as a response to this, creation of one filtering module was proposed as part of effort to better integrate collection and analysis of IP flows using these programs. The one filtering module aims to provide one implementation and support for popular filtering language for use in the programs. Thesis contains theoretical introduction to flow monitoring in networks, describes algorithms useful for evaluation of conditions on flow records and packets. The core of authors work is design and implementation of the filtering module and its wrappers for the collector and analysis library. Results of performance tests and evaluation of features can be found in the thesis's conclusion.

Kľúčové slová

filtrovanie, filtračný modul, filtračný jazyk, filtrovanie tokov, sieťové toky, IP toky, NetFlow, nfdump, IPFIX, IPFIXcol, libnf

Keywords

filtering, filtering module, query language, flow filtering, network flows, IP flows, NetFlow, nfdump, IPFIX, IPFIXcol, libnf

Citácia

ŠTOFFA, Imrich. *Filtr IP toků*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Wrona,

Filtr IP toků

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Jána Wrony. Ďalšie informácie mi poskytli Martin Žádník a Lukáš Huták. Uviedol som všetky literárne zdroje a publikácie z ktorých som čerpal.

.....
Imrich Štoffa
17. mája 2017

Poďakovanie

Ďakujem vedúcemu mojej bakalárskej práce Ing. Janovi Wronovi za jeho odborné vedenie, a čas ktorý venoval pripomienkovaniu mojej práce. Tiež ďakujem Bc. Lukášovi Hutákovi za jeho odbornú pomoc a postrehy.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Internet od svojho vzniku prešiel obrovským vývojom. Od siete niekoľkých počítačov na veľkých univerzitách sa z neho stala celosvetová sieť s miliónmi užívateľov. Jednoducho povedané stala sa z neho ďalšia vrstva životného prostredia ľudí, virtuálny svet kde si vymieňame informácie.

Z pohľadu správy je internet rozdelený do menších autonómnych sietí ktoré spravujú samostatné skupiny administrátorov. Bežné monitorovacie prostriedky ako napríklad SNMP a základné aktívne nástroje neprezrádzajú dostatok informácií o dianí na sieti. Časom, ako sa siete vyvíjali došlo k potrebe sledovať a reagovať na aktuálne dianie v sieti z rôznych dôvodov. Medzi nimi napríklad bezpečnosť, plánovanie a sledovanie kvality služieb. Monitoring na základe tokov je jedna z možností ako merať sieť. Informácie ktoré monitoring poskytuje je možné využiť rôznym spôsobom. Pre plánovanie siete je dôležité vedieť obvyklé toky, kde linky narážajú na limity a kde ešte ich kapacita postačuje. Pre poskytovateľov je dôležité vedieť či niektorý z užívateľov nevyčerpal dohodnutú prenosovú kapacitu, a dáta nad limit patrične spoplatniť. Administrátori takto môžu zistiť či na sieti nedochádza k nepovolenej činnosti ako prevádzka nepovoleného serveru, šírenie vírusov, nevyžiadanej pošty. Je nutné dodať že k zberu dát obyčajne nedochádza všade na sieti ale len na vhodne zvolených linkách alebo uzloch.

Pre zber dát a ich analýzu je potrené na sieti nasadiť softvérové a hardvérové vybavenie, ktoré toto realizuje. Existuje veľké množstvo softvérových kolektorov a analyzátorov. Medzi voľne dostupné riešenia patrí napríklad NfSen, založený na sade nástrojov nfdump, avšak špecializácia návrhu analyzátora nfdump je veľmi úzko spätá s dátovým formátom ktorý používa a ktorý predstavuje jeho najväčšiu nevýhodu. Nfdump je schopný filtrovať nemennú množinu položiek protokolu NetFlow v9, čo mimo iné znamená žiadnu podporu pre položky s rôznou dĺžkou a veľmi komplikované rozšírenie o vlastné položky. Táto práca má za cieľ vytvoriť podobne výkonné filtračné jadro a zároveň prekonať limitáciu spojenú s úzkou väzbou na formát dát. Motiváciou pre vytvorenie tohoto jadra je potreba filtrovania a profilovania záznamov o tokoch počas ich zberu v kolektore IPFIXcol a ich analýza pomocou knižnice libnf. IPFIXcol podporuje novší, komplikovanejší protokol pre export dát (IPFIX), čo znamená väčšie množstvo položiek rôznych typov ktoré je možno filtrovať. Libnf knižnica sa pokúša riešiť problémy s rozšírením formátu nfdump a používa jeho interné moduly. Ak sa majú dáta uložené IPFIXcolom analýzovať pomocou libnf filtra je treba zariadiť zjednotenie jazykov ktoré používajú. V súčasnej dobe to tak nie je a snahou tejto práce bude vytvorenie jednej implementácie filtra pre oba tieto programy.

V kapitole 2 je popísaný teoretický úvod do monitorovania ip tokov v sieti a jeho architektúra, kapitola 3 obsahuje metriky algoritmov pre klasifikáciu a obsahuje popis niek-

torých dátových štruktúr pre použitie v klasifikátoroch. Kapitola 4 je navrhnutý modul a gramatika pre filtrovanie spolu s popisom problémov a detailov implementácie ktoré táto práca priniesla. Testovanie a výkonnostná analýzu možno nájsť v poslednej kapitole 5.

Kapitola 2

Monitorovanie IP tokov

Účelom tejto kapitoly je načrtnúť problematiku monitorovania siete a identifikácia miest v systéme kde sa filtrovanie využíva. Požiadavky na metódu filtrovania sa budú líšiť v závislosti od zariadenia, povahy a účelu. Monitorovanie ako také sa vyvíjalo už od vzniku internetu. Dôležité siete je treba udržiavať v behu, čo bez efektívneho spravovania neide. Administrátori pre svoju prácu potrebujú vedieť v akom stave sú sieťové prvky a monitoring im tieto informácie zaisťuje. Existuje viacero spôsobov získavania aktuálneho stavu siete [12].

Aktívne nástrojmi ktoré generujú prenosy a sledujú odozvu zariadení na sieti, za účelom otestovania dostupnosti liniek, uzlov alebo služieb. Patria sa sem napríklad `telnet`, `ping` a `traceroute`, aplikácie pracujúce s ICMP[1], SNMP[2] protokolmi.

Pasívne logovaním systémových udalostí pomocou Syslog, získavaním hlásení – asynchrónnych správ o stave siete SNMP či pomocou záznamov o tokoch. Získavanie záznamov komunikácii prebieha zachytávaním paketov v sledovacích bodoch, ich predspracovaním a prenosom na server kde sa uložia na potrebnú dobu. Takého dáta poskytujú komplexný náhľad na dianie v sieti a v závislosti na nastavení predspracovania, a vôbec celej infraštruktúry zberu, sa môžu zachytávať časti hlavičiek až celý obsah. Záchyt celých paketov[3] je pamäťovo intenzívna činnosť a predstavuje podstatný zásah do súkromia užívateľov. Miesto zachytávania celých paketov sa obyčajne sledujú len niektoré parametre a zaznamenávajú sa len raz pre celú množinu paketov s rovnakými vlastnosťami za určitý čas daný nastavením. Monitorovanie IP tokov je viac fázový proces ktorý začína zachytávaním a agregáciou paketov do tokov a končí analýzou týchto dát.

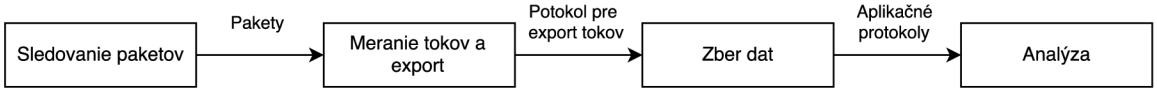
2.1 Historické pozadie

Najstaršie zverejnené práce týkajúce sa exportu tokov sú z 90 tých rokov. Roku 1991 bola popísaná metóda agregácie paketov do tokov na základe informácii v hlavičke pracovnou skupinou spadajúcou pod Internet Engineering Task Force (IETF). V 1995 vzrástol záujem o analýzu tokov pre účely profilovania premávky na internete. O rok neskor, 1996, započala IETF skupina Realtime Traffic Flow Measurement (RTFM) výskum problémov pri meraní prenosov a zariadení, výsledkom čoho bol lepší model prenosu tokov a architektúra pre vylepšené meranie tokov. To trvalo do roku 2000. Znova, kvôli nedostatočnému

[1] Internet Control Message Protocol

[2] Simple Network Management Protocol, dovoľuje aktívne aj pasívne získavanie stavu siete. Umožňuje zber a organizáciu informácii o zariadeniach v IP sieti a tiež ich zmenu pre vzdialenú konfiguráciu.

[3] Na bežných počítačoch možno pomocou aplikácie Wireshark analyzovať pakety z lokálnej siete



Obr. 2.1: Typická architektúra monitorovania tokov

záujmu zo strany výrobcov nedošlo k vytvoreniu štandardu. V dobe práce skupiny RTFM firma Cisco pracovala na vlastnej verzii technológie pre export tokov zvanej NetFlow. Svoj počiatok našla pri zdokonaľovaní smerovania. Jednalo sa o vylepšenie smerovača[4], kde sa údaje o tokoch udržiavajú vo *flow-cache* a rozhodnutie kam preposlať paket sa vykoná raz, vždy pre prvý paket v toku, v *control plane*. Nasledujúce pakety sú smerované už len v *data plane*. Užitočnosť obsahu *flow-cache* bola sekundárny objav a využitie vyžadovalo už len zabezpečenie prenosu do centrálneho zariadenia – export. Cisco patentovala NetFlow v roku 1996. Okolo roku 2002 sa NetFlow v5 stala prvou široko akceptovanou verziou aj napriek tomu, že sa jednalo o proprietárnu technológiu. Rozšíreniu prispel najmä fakt ze Cisco sprístupnilo používaný formát dát. Novšia verzia NetFlow v9 priniesla flexibilnejší spôsob uloženia dát v záznamoch pomocou šablón a podporu technológií IPv6, Virtual Local Area Networks (VLAN) a Multiprotocol Label Switching (MPLS). Roku 2002 IETF rozhodlo, že vytvorí štandard pre export tokov a založilo pracovnú skupinu IP Flow Information Export (IPFIX). Po vyhradení požiadavok a vyhodnotení existujúcich protokolov bol ako nový základ vybraný NetFlow v9. IPFIX je ale odlišný protokol s novými možnosťami. Prvá špecifikácia bola dokončená na začiatku roku 2008. Po štyroch rokoch bola práca skupiny ukončená. Táto špecifikácia sa stala základom dnešného IPFIX Internet Standard koncom roka 2013, avšak, vývoj pokračuje dodnes.

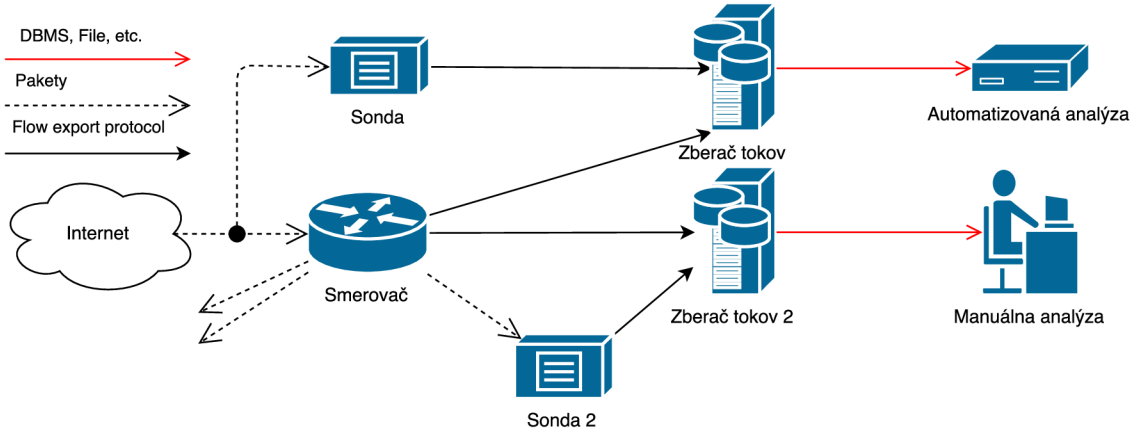
2.2 Architektúra monitorovania tokov

Typická architektúra monitorovania tokov podľa [8] pozostáva z niekoľkých krokov, znázornené na obrázku 2.1. Je treba poznamenať, že jednotlivé fázy spracovania v tejto architektúre sú vecou návrhu a môžu byť delené a kombinované rôznymi spôsobmi. Sledovanie paketov a meranie tokov býva často skombinované do jedného zariadenia zvaného exportér tokov.

Prvá fáza je sledovanie paketov (packet observation), pakety sú zachytávané v sledovacom bode (observation point) a následne dochádza k ich predspracovaniu. Pre každý zachytený paket dôjde k pridaniu časovej značky, ďalšie kroky sú už voliteľné, orezaniu, vzorkovaniu a filtrovaniu.

Druhou fázou je meranie tokov a export(flow metering & export), pozostáva z meracieho a exportného procesu. V meracom procese sú pakety agregované do tokov, po ukončení môže opäť dôjsť ku vzorkovaniu a filtrovaniu záznamov. Vyhovujúce záznamy sú zaradené do datagramu exportného procesu. Záznamy o tokoch môžme vnímať ako tabuľky, kde každý riadok reprezentuje jeden záznam obsahujúci stĺpce pre každú meranú vlastnosť. V praxi merací a exportný proces spolu úzko súvisia. Pokiaľ je exportér tokov kombinovaný so sledovacím zariadením ide o dedikované zariadenie, označuje sa aj ako sonda (flow probe). IPFIX architektúra bola vyvinutá s ohľadom na variantu exportu, kde je žiadúce preposielať pakety na ďalšie zariadenia (forwarding), teda je možné, aby viacero exportných

[4]Dnešná technika napr. Flow based switching – Cisco Calalyst 6500



Obr. 2.2: Príklad systému monitorovania [8]

procesov posielalo dáta na viacero zberacích procesov. Po predspracovaní sú dáta dostupné na analýzu, či už automatickú alebo manuálnu.

Treťou fázou je zber dát (data collection). Jeho hlavnou úlohou je príjem, ukladanie a predspracovanie dát o tokoch. Bežné operácie predspracovania zahŕňajú agregáciu, filtrovanie, kompresiu a vytváranie súhrnných informácií.

Poslednou fázou je analýza dát, jedná sa o samotné spracovanie dát o tokoch. Bežné analýzy zahŕňajú koreláciu a agregáciu tokov, profilovanie premávky, klasifikáciu, charakterizáciu, detekciu anomálií či vniknutí a prehľadávanie uložených dát pre forenzné alebo výskumné účely.

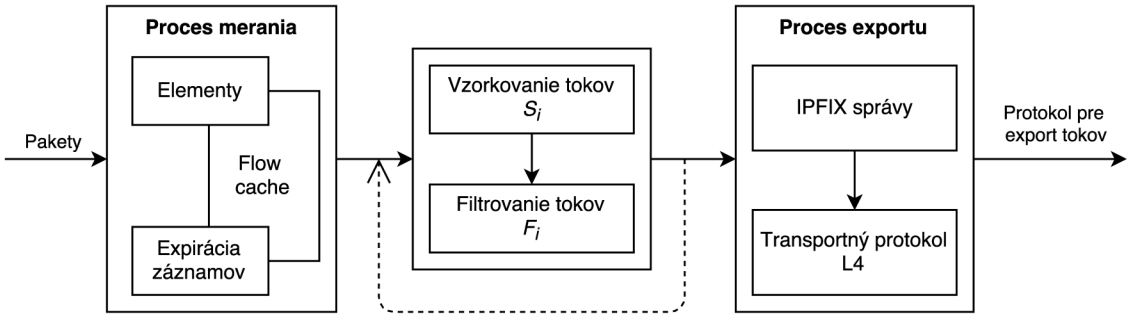
Sledovanie paketov

Táto operácia rieši zachytávanie paketov z linky a zahŕňa aj ich predspracovanie pre proces merania tokov. Je to kľúčový aspekt monitorovania. Sledovacie zariadenie v móde *in-line* sa umiestňuje na linku medzi dva zariadenia ako most alebo pasívna „kopírka premávky" (TAP[5]). Oba varianty poskytujú dedikovaný spoj pre zariadenie na záchyt. *Mirroring-mode* využíva zariadenie už prítomné na sieti schopné smerovať pakety pre preposielanie paketov z 1 a viac portov na port vyhradený na záchyt. Tento spôsob ale ovplyvňuje časovanie a poradie paketov. Sonda na obrázku 2.2 používa TAP, Sonda 2 zase zrkadlenie zo smerovača. Samotný záchyt typicky vykonáva sieťová karta (NIC).

Než sa paket po prijatí uloží do prijímacej medzipamäte NIC, musí prejsť niekoľkými kontrolami ako overenie kontrolných súčtov CRC atď., následne sa paketom pridá časová značka. Presné časové značky sú dôležité pre presnosť analýz. Značkovanie priamo v hardvéri predchádza nepresnostiam, ku ktorým dochádza pri predávaniu paketov do softvéru, čo umožňuje dosiahnuť presnosť až 100 nanosekúnd. Avšak, táto technika je dostupná len v špeciálnych NIC založených na FPGA, bežné NIC využívajú značkovanie v softvéri so synchronizáciou protokolmi NTP alebo SNTP, čo dovoľuje presnosť v ráde stoviek mikrosekúnd.

Záchyt a značkovanie, sú povinné pre všetky pakety, následujúce kroky už sú voliteľné. Orezanie vyberie iba časť paketu ktorý sa zmestí do prednastavenej dĺžky, čo šetrí výpočtové zdroje. Bežne sa pakety orezávajú len na L3–L4 hlavičky, čiže obalky bez aplikačných dát, záleží ale na aplikácii.

[5]Test Access Port



Obr. 2.3: Architektúra meracieho a exportného procesu

Vzorkovanie a filtrovanie umožňujú vybrať podmnožinu všetkých paketov a znížiť tak objem dát pre ďalšie spracovanie. *Vzorkovanie* možno realizovať periodicky alebo stochasticky, s rôznymi pomermi. Pomer $1 : N$ vyjadruje pravdepodobnosť, že každý N-tý (periodicky) alebo priemerne 1 z N paketov (stochasticky) sa akceptuje. Tento prístup umožní znížiť objem dát za cenu straty informácií so snahou štatisticky odhadnúť vlastnosti celého toku. To predstavuje ďalšie výzvy počas analýzy a dokonca znemožňuje niektoré aplikácie[6]. *Filtrovanie* deterministicky oddeľuje pakety ktoré majú istú vlastnosť od ostatných. Podobne, môže byť použité pre redukciu množstva dát. Možno odlíšiť dva hlavné druhy: **filtrovanie podľa vlastností** pracuje tak, že vyberá pakety ktorých polia vyhovujú požadovaným hodnotám alebo spadajú do rozsahu. Typicky filtrovanie pre záchyt komunikácie konkrétnych IP adries, aplikácii atď. Druhým spôsobom je **filtrovanie založené na hash funkcií**, ktorá sa aplikuje na paket alebo jeho časť, výsledok sa porovnáva buď na rozsah alebo konkrétnu hodnotu. To môže slúžiť ako efektívny spôsob výberu paketov so spoločnými vlastnosťami, alebo ak je hash funkcia aplikovaná na veľkú časť paketu, môže vyberať pakety kvázi-náhodne.

Meranie tokov a export

Do meracieho procesu vstupujú pakety transformované predošlou fázou. Tok je definovaný ako *IP pakety ktoré prešli sledovacím bodom v sieti počas istého časového intervalu* majúce *spoločnou n-ticu rovnakých hodnôt* [3]. Záznamy o tokoch su definované ako *informácie o jednotlivých tokoch, zachytené sledovacím bodom* [3], a môžu zahŕňať ako charakteristickú n-ticu toku, tak merané vlastnosti. K agregácií zariadenia využívajú tabuľku v pamäti, (*flow cache*) do ktorej sa ukladajú rozšírené záznamy reprezentujúce aktívne toky. Tieto záznamy obsahujú kľúče tokov spolu so zvyšnými charakteristikami[7]. Kľúč je bežne tvorený zdrojovou, cieľovou IP adresou, zdrojovým, cieľovým portom a číslom IP protokolu. Flow cache sa väčšinou implementuje ako hash tabuľka so zreťazením synoným v ktorej sa vyššie spomínaný kľúč používa pre prístup do tabuľky. Pokiaľ záznam pre spracovávaný paket neexistuje, vytvorí sa nový na jeho základe, inak sa len aktualizujú hodnoty daného počítadiel vyhľadaného záznamu[8]. Vyššie uvedený popis je obecný, pre určité špecifické aplikácie sa hodia iné typy flow cache napr.: *immediate cache* pre takzvané single-packet flows a *perzistentné cache* v ktorých toky nikdy neexpirujú.

[6]Poskytovatelia internetového pripojenia používajú monitorovanie tokov pre účtovanie prenesených dát jednotlivých zákazníkov v rôznych tarifoch.

[7]Počítadlá bajtov, TCP príznaky, čas prvého a posledného paketu toku etc.

[8]Sumárne políčka ako počet paketov v toku, počet prenesených bajtov atď.

Záznamy o toku zotrvávajú v cache, kým tok ktorý reprezentujú nieje považovaný za ukončený (expirovaný), potom je spracovaný exportným procesom, teda zaradený do *datagramu* (IPFIX datagram je načrtnutý na obrázku **??**) nasadeného exportného protokolu. IPFIX doporučuje pre expiráciu použiť niektoré z nasledujúcich spôsobov:

- Limit doby aktivity – Limitovanie doby aktivity predčasne ukončí tok, záznam ostáva v cache a časy začiatku a konca toku sa aktualizujú a počítadlá sa reštartujú. Hodnota sa pohybuje medzi 120 sekundami az 30 minútami. Limitovanie sa používa aby sa predišlo javu, kedy sa extrémne dlhé toky s potenciálne veľkým dopadom na sieť nahlásia až po ich skončení.
- Limit doby nečinnosti – pokiaľ sa definovaný dobu neobjaví paket k danému toku, je považovaný za ukončený.
- Obmedzenie zdrojov – heuristika, redukcia limitov počas behu môže spôsobiť predčasnú expiráciu tokov.
- Prirodzené ukončenie toku – pokiaľ sa v TCP komunikácii objaví paket s príznakom FIN alebo RST
- Núdzová expirácia – istý počet tokov bude expirovaný pokiaľ dôjde k naplneniu cache
- Vyprázdnenie cache – v nečakaných situáciach, kedy musia byť všetky toky expirované. Napríklad pri väčšej zmene času po synchronizácii.

Limity dôb aktivity a nečinnosti majú vplyv na maximálny počet záznamov v cache. Dlhšia doba aktivity zlepšuje pomer agregácie paketov, čo je žiadúce, ale predlžuje dobu počas ktorej je tok neviditeľný pre analýzu.

Vzorkovanie a filtrovanie v tejto fáze pracuje s záznamami o tokoch – paketmi po agregácii, čo možno vidieť na obrázku 2.3, ale nasleduje rovnaké princípy ako vo fáze sledovania paketov. Polia ktoré možno exportovať sa inač nazývajú *information elements* (IE). Organizácia IANA spravuje zoznam *IPFIX information element registry* [9] so štandardnými IE. Okrem štandardných je možné tiež definovať podnikové elementy špecifické pre potreby konkrétnej aplikácie v oddelenom zozname. Elementy v zozname majú priradené meno, číselne id, typ, dĺžku, status (platné/zastaralé), popis a pokiaľ sa jedná a podnikové elementy tak aj *enterprise ID*. IPFIX umožnuje definovať ľubovoľné elementy a exportovať ich, kvôli tomu je aplikovateľný aj mimo počítačových sietí (napr. monitoring dopravy, zber dát zo senzorických sietí). Bežný záznam o toku vo formáte IPFIX obsahuje následovné polia:

Elementy časových značiek sa môžu vyskytnúť vo verziách s rôznou presnosťou, taktiež IP adresy vo verzii 6.

V dobe písania zoznam IANA registruje už viac ako 400 položiek. Takýto počet elementov naznačuje, že existuje obrovské množstvo kombinácií ktoré je možné exportovať. Tento problém je v IPFIX riešený následovne. Datagramom protokolu IPFIX je IPFIX správa, obsahuje hlavičku a sady. Existujú tri typy sád a sú určené ID číslom sady: šablóna, šablóna vlastností a dáta. Sada oddeľuje región so záznamami jedného typu a určuje jeho dĺžku. Sada typu šablóna, ID sady 2, obsahuje ID šablóny a definuje n-ticu elementov. Sada typu šablóna vlastností, ID sady 3, poskytuje mechanizmus exportu metadát o záznamoch, zariadení, prostredí atď. Sada typu data, ID sady ≥ 256, začína ID číslom šablóny podľa ktorej boli záznamy v tejto konkrétnej sade uložené a nasledujú dáta jednotlivých záznamov o tokoch. Sada typu šablóna obsahuje zoznam šablón, kde každá popisuje štruktúru

ID	Položka	Popis
152	flowStartMilliseconds	Časová značka prvého paketu v toku.
153	flowEndMilliseconds	Časová značka posledného paketu v toku.
8	sourceIPv4Address	IPv4 zdrojová adresa v hlavičke.
12	destinationIPv4Address	IPv4 cieľová adresa v hlavičke.
7	sourceTransportPort	Zdrojový port v transportnej hlavičke.
11	destinationTransportPort	Cieľový port v transportnej hlavičke.
4	protocolIdentifier	Číslo IP protokolu v hlavičke.
2	packetDeltaCount	Počítadlo paketov v toku.
1	octetDeltaCount	Počítadlo prenesených bajtov v toku.

Tabuľka 2.1: Bežne používané IPFIX elementy

záznamu a je jej priradené ID > 256 a zoznam IPFIX elementov usporiadaných podľa ich výskytu v dátovom zázname. Štruktúra datagramu je načrtnutá na obrázku 2.4. Aby došlo k obmedzeniu fragmentácie pri prenose, exportéri vytvárajú IPFIX správy tak, aby nepresiahli 1500 B (MTU). Vynímkou je situácia keď dôjde k prekročeniu limitu kvôli variabilnej položke[9] [3]. Exportný proces odosiela datagramy na kolektor podľa špecifikovaného nastavenia. Pre prenos datagramov do kolektora sa používajú L4 protokoly TCP, UDP alebo SCTP. Najoptimálnejšou voľbou je protokol SCTP pomocou neho sa dáta prenášajú v 1 a viac paralelných kanáloch spojenia, resp. asociáciách, ako tzv. chunky. V rámci kanálu je zabezpečené doručenie v poradí a pozdržanie chunku jedného kanálu neovplyvní ostatné. Ak STCP nie je natívne podporovaný, je možné zapuzdriť ho do UDP a využiť preň upravené API protokolu TCP. Ak bude export prebiehať na dedikovaných linkách kde sa nepredpokladá zahltenie a programové vybavenie nepodporuje SCTP, je možné efektívnejšie použiť UDP s výhodou oproti TCP.

Zber dat

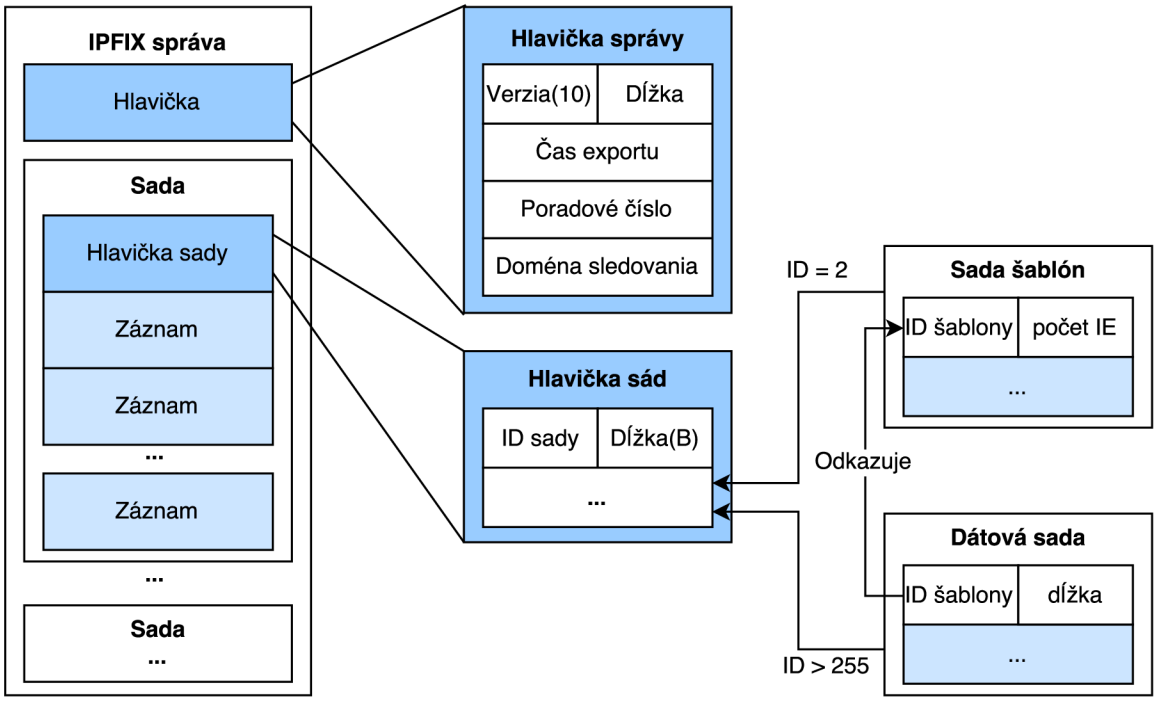
Kolektor tokov je ďalší základný kameň monitorovania, keďže je to časť v ktorá prijíma, predspracováva a ukladá dáta o tokoch z exportérov v sieti pre neskoršiu analýzu. Zber sa vykonáva v takzvanom zberacom procese a jeden kolektor môže vykonávať viac takýchto procesov. Medzi bežné úlohy v predspracovaní patrí komprimácia, agregácia, anonymizácia sád záznamov, filtrovanie a vytváranie súhrnných štatistík.

Hlavnou činnosťou zberacieho procesu je príjem a ukladanie v požadovanom formáte. Pre každý exportný proces spravuje šablóny patriace k jeho dátovým sadám, takže si musí pamätať čísla domény sledovania (ODID) a prenosovú inštanciu (transport session), pretože táto dvojica definuje kontext platnosti danej šablóny. Inštancia pre TCP prenos je identifikovaná zdrojovou IP a portom a cieľovou IP a portom. Spôsob spravovania šablón sa líši podľa protokolu L4 použitého pre prenos. TCP zberací proces spravuje priestor pre šablóny podľa n-tice: zdrojová IP, cieľová IP, zdrojový port, cieľový port a odid. Kolektory si sami spravujú sady elementov[10], keďže šablóny prichádzajúce z exportérov obsahujú len ID a dĺžku IE. Bez týchto definícií kolektor nie je schopný rozlíšiť položky v prichádzajúcich záznamoch.

Sledovanie IP tokov oproti záchytu celých paketov podstatne menej zasahuje do súkromia užívateľov, lebo nenesú obsahy paketov, predmet komunikácie koncových užívateľov

[9] Maximálna dĺžka variabilnej položky je $2^{32} - 1$ B

[10] Dátové typy, názvy, sémantika... ako v IANA registri.

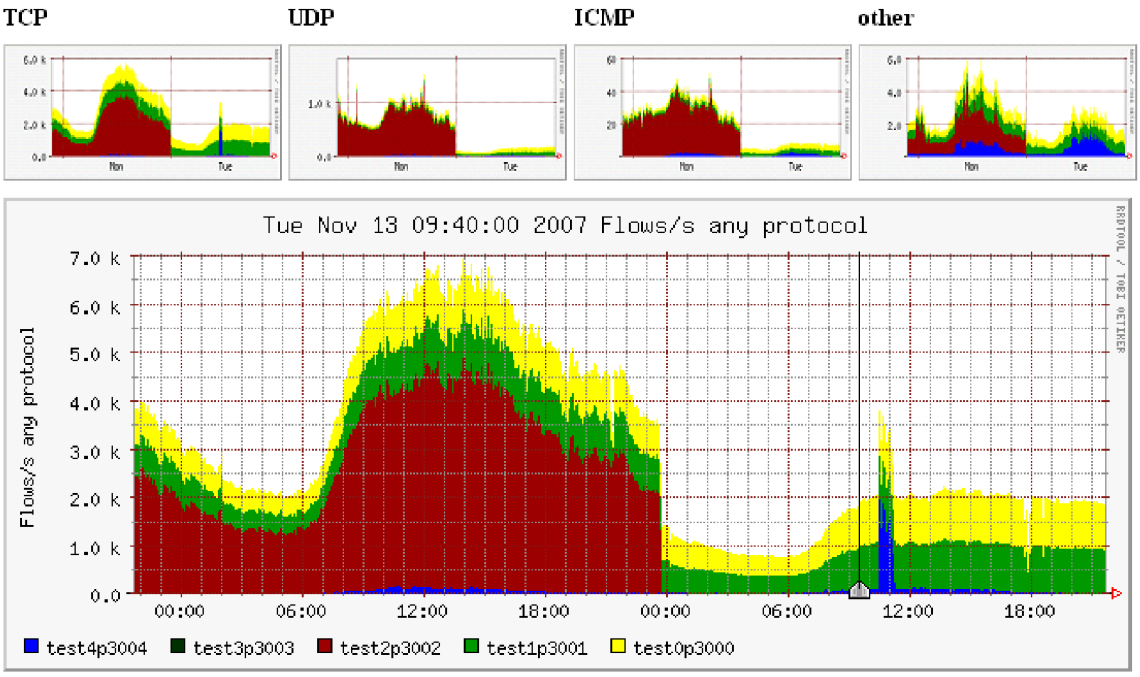


Obr. 2.4: Zjednodušená štruktúra datagramu IPFIX

ostáva skrytý. Avšak dáta o tokoch stále predstavujú hrozby pre súkromie, pretože z nich možno identifikovať jednotlivcov a sledovať ich činnosť. Anonymizácia záznamov zhoršuje ich využiteľnosť v následných analýzach. Vždy záleží na type anonymizácie a type analýzy. Napríklad odstránenie IP adries zo záznamov znemožní akúkoľvek analýzu závislú na priradení tokov ku konkrétnym koncovým bodom. Crypto-PAn algoritmus anonymizuje IP adresy tak, že jednotlivý klienti zdieľajú spoločnú podsieť, čo zachováva použiteľnosť dát, za cenu zmenšenia neurčitosti kvôli zmenšeniu priestoru možných pôvodných adries. Aj napriek anonymizácii dáta obsahujú kúsky informácii ktorých zneužiteľnosť rastie s objemom, takže zverejňovanie stále predstavuje potencionálny únik informácií.

Dáta ktoré kolektor prijíma sa môžu ukladať: **perzistentne**, čo vyhovuje ak je treba dáta o tokoch uchovávať dlhodobo a **volatilne**, kde výhodou je rýchly prístup. Prevod dát z volatilnej do perzistentnej pamäte môže vytvárať úzke hrdlo, kvôli rozdielu rýchlostí jednotlivých médii. Formáty uloženia na perzistentných médiach sa delia na:

- Ploché súbory, uložené textové alebo binárne, zvyčajne veľmi rýchle (nfdump). Jednoduché dotazy.
- Riadkové databázy, kde sú dáta uložené v riadkoch jednotlivých tabuliek, obyčajne sa používa v systémoch riadenia báze dat (MariaDB, MySQL, postgreSQL). K dátam sa pristupuje vždy po celých riadkoch aj keď je potrebný len jeden stĺpec.
- Stĺpcové databázy, majú výhodu oproti SRBD, pretože pre výpočet dotazu sa pristupuje len do potrebných stĺpcov (FastBit).



Obr. 2.5: Príklad grafu NfSen s náhľadom na početnosť tokov v čase [2, sekcia 6.]

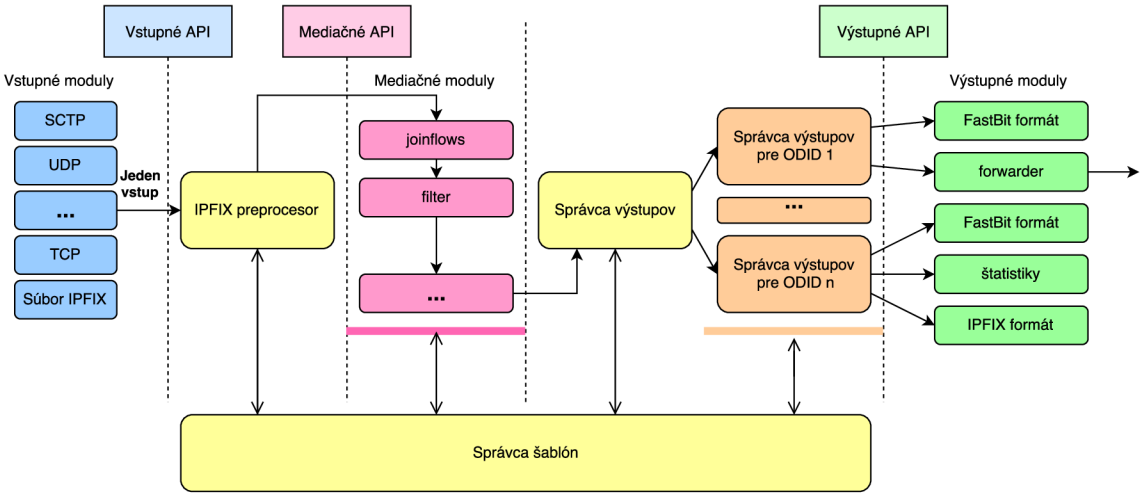
Analýza

Finálna fáza monitorovania, tu sa schádzajú výsledky všetkých predošlých fáz. Rozlišujú sa tri oblasti analýzy: Analýza tokov a generovanie hlásení, Detekcia hrozieb a Monitorovanie výkonu. Táto sekcia načrtne, čo všetko sa dá robiť s dátami o tokoch.

Za predpokladu, že dáta o tokoch sa zvyčajne zberajú zo strategicky zvolených bodov v sieti, kde je možné sledovať veľké množstvo klientov, tieto dáta poskytujú obsiahly súhrn spojení. Analýza tokov a generovanie hlásení je najzákladnejšia funkcia aplikácii na monitorovanie a obyčajne poskytuje následovne:

- **Prehľadávanie a filtrovanie dát**
- **Náhľad štatistík** – napr. *Top-talkers* – entity ktoré preniesli najviac dát.
- **Generovanie hlásení a poplachov** – Hlásenia o využití šírky pásma, počte vytvorených pripojení, použití zakázaných aplikácii alebo protokolov jednotlivými zákazníkmi, pričom prekročenie limitov môže viesť na generovanie poplachu.

Zvyčajne analýza prebieha sledovaním súhrnných grafov premávky podobných ako na obrázku 2.5. Grafy obyčajne sledujú množstvo prenesených tokov/paketov/bajtov za zvolené časové obdobie. Špičky v týchto grafoch signalizujú, že premávka sa líši od toho čo sa považuje za normálne. Či sa jedná o planý poplach alebo škodlivé správanie je treba ďalej preskúmať. Z intervalu v ktorom sa anomália objavila sa vyberú uložené top štatistiky. Preskúmaním najaktívnejších klientov vzhľadom na počet tokov možno odhaliť potencionálne skenovanie siete, slovníkové útoky na SSH, distributed denial of service (DDoS) a ďalšie. Po identifikácii typu útoku je možno získať lepší prehľad prezrením surových dát. Podobné analýzy s generovaním hlásení možno automatizovať [7].



Obr. 2.6: Modulárna schéma IPFIXcol kolektora

2.3 Nástroje pre monitorovanie

Nástroje tu uvedené súvisia s praktickou častou mojej práce. Filtrovací jazyk bol pôvodne vyvíjaný ako mediačný modul do IPFIXcolu pre potreby profilovania záznamov o tokoch. Jazyk je založený na syntaxi akú používa NfSen.

IPFIXcol

Tento kolektor [1] je nástroj schopný príjmu dát o tokoch z jedna a viac exportérov vyvíjaný podľa dokumentu RFC7011 [3]. Jeho primárnou úlohou je ukladanie dát o tokoch v požadovanom formáte. Jadro kolektora sa stará o spracovanie IPFIX správ, správu šablón chybové kontroly. Rieši životnosť šablón (UDP prenosy), detekciu záznamov bez šablón a kontrolu sekvenčných čísel. Jeho modulárna architektúra, znázornená na obrázku 2.6, ho robí flexibilným a rozšíriteľným.

Vstupné moduly zaisťujú príjem dat pre kolektor, typicky sa jedná a moduly prijímajúce pripojenia od exportérov alebo čítajúce dáta z disku. Protokoly TCP, UDP, SCTP, a dalšie používajú vlastné moduly pre daný protokol určený. Kolektor je schopný prijímať aj NetFlow v5 a v9 datagramy vďaka jednoduchému prevodu.

Výstupné moduly riešia záverečné spracovanie dát, teda presun do perzistentnej pamäte. Pre uloženie môže slúžiť napríklad databáza, modul postgres, alebo konkrétny formát dát. Momentálne podporuje json, nfdump, unirec, fastbit a IPFIX. Dáta ale môžu být preposlané po sieti na iné kolektory (v rôznych formátoch), čo z IPFIXcolu robí mediátor.

Kolektor umožňuje transformovať IPFIX dáta, než opustia volatilnú pamäť, pomocou mediačných modulov. Modul prijme dáta z jadra alebo predošlého modulu, vykoná svoj výpočet a odošle správu ďalej. Výpočet/modifikácia môže byť akákoľvek, avšak podmienkou je, že výstupom musí byť validný IPFIX záznam. Tieto moduly možno za seba rôzne reťaziť a ich úlohou môže byť napríklad: *profilovanie dát*, výpočet štatistík, geolokácia, anonymizácia, atď. [10].

NfSen

Je to grafické webové rozhranie, nadstavba, pre nfdump sadu nástrojov [6, 7]. Sprístupňuje užívateľovi silu príkazového riadku pri filtrovaní v GUI, vizualizáciu dát v čase prenesených tokov/paketov/bajtov a ich jednoduchú navigáciu. Pomocou profilov užívateľ môže vytvoriť históriu a profily dát. V grafoch je možné vyznačiť časove okno a analyzovať len dáta ktoré doň spadajú. Umožňuje automatizovať analýzy a stanoviť podmienky pre hlásenia incidentov. Poskytuje rôzne rozhrania k dátam pre rôzne úlohy.

Program `nfcapd/sfcapd` slúži ako kolektor. Je schopný prijímať NetFlow v5, v7 a v9 resp. sFlow. Pre každý exportér je potrebná jedna inštancia. Dáta ukladá priamo do súboru v formáte nfdump a rotuje po nastavenom časovom intervale. Rysy filtračného jazyka budú rozobraté v sekcii 3.3.

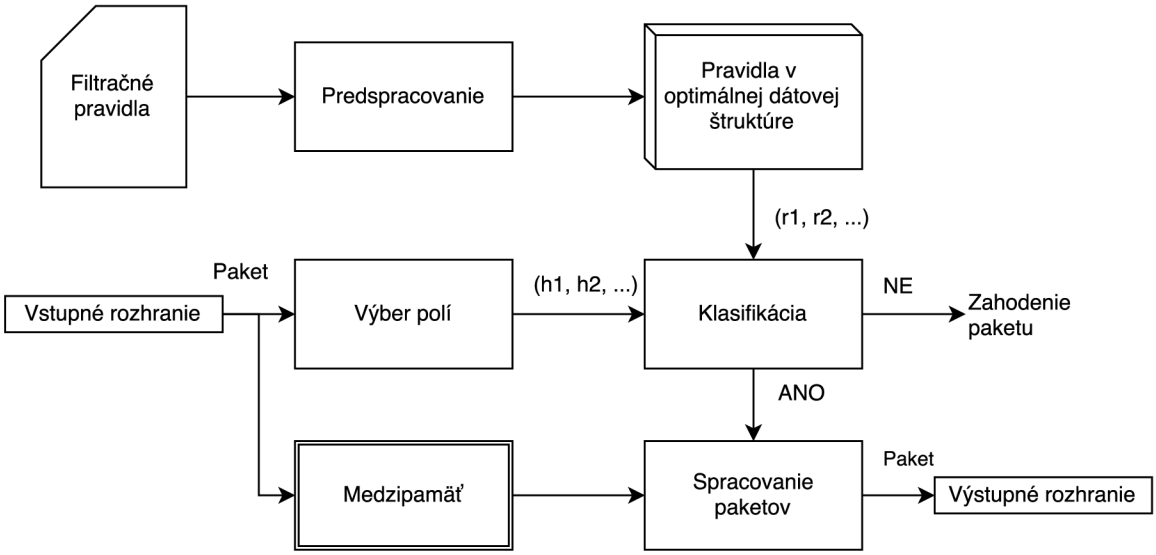
Kapitola 3

Klasifikácia a filtrovanie dát

V počítačových sieťach je často potrebné vykonávať akcie s paketom na základe polí jeho hlavičky. Niekoľko sieťových služieb závisí na klasifikácii paketov napr.: v smerovačoch sa vyhľadáva najdlhšia zhoda cieľovej IP podľa smerovacej tabuľky, vo firewalloch dochádza k zákazu/povoleniu prechodu paketu zariadením na základe rôznych polí, podľa zadaných pravidiel. Ďalej pre poskytovanie prioritnej obsluhy pre rôzne typy prenosov (QoS) dochádza k regulácií tokov na základe značiek ktoré boli pridelené predom, klasifikáciou podľa zadaných skupín, pri účtovaní a faktúrovaní za rôzne typy služieb dochádza ku klasifikácii paketov do jednotlivých tried, napr. VoIP sú náročnejšie a teda drahšie ako HTTP prenosy, kvôli požiadavkám na odozvu a plynulosť v sieti. Vo všetkých týchto prípadoch je na vykonanie rozhodnutia dostupné obmedzené časové kvantum. Pakety môžu byť napríklad presmerované do dočasnej rýchlej pamäte s obmedzenou kapacitou a pokiaľ by rozhodovanie nebolo dostatočne rýchle došlo by k výpadkom kvôli zahlteniu.

Klasifikácia je algoritmický problém, kedy sa snažíme roztriediť záznamy podľa zadaných pravidiel. Kritéria pre klasifikáciu sú označované ako *databázy pravidiel* alebo *politík* ktorých súčasťou bývajú aj *akcie* priradené jednotlivým pravidlám určujúce operáciu s paketom. Cieľom je ku každému záznamu vyhľadať pravidlo ktoré spĺňa. Proces klasifikácie môžme rozdeliť do dvoch krokov: **predspracovanie pravidiel** zabezpečuje prevod do vhodnej dátovej štruktúry, tento krok sa vykonáva čiastočne alebo úplne, (záleží na type štruktúry) pri pridávaní, mazaní či modifikácii pravidiel. Druhým krokom je samotná **klasifikácia**, v ktorej dochádza k spracovaniu záznamov a vyhľadávaniu polí potrebných pre klasifikáciu pričom sa prechádza dátová štruktúra hľadá sa zhoda (výber sa líši podľa algoritmu). O filtrovaní možno tiež uvažovať ako o klasifikácii záznamov do dvoch tried (vyhovuje/nevyhovuje) takže sa jedná o rovnakú úlohu.

Nech máme paket s d rôznymi poliami $H_1, H_2, \ldots, H_d$ a každé pole je reprezentované sekvenciou bitov. Pre IPv4 paket môže byť reprezentovaný poliami: zdrojová adresa a port, cieľová adresa a port, číslo protokolu a príznaky protokolu. Paket z adresy 192.16.0.24:2421 do 24.14.15.7:80 protokolu TCP môže byť vyjadrený ako kombinácia (192.168.0.24, 24.14.15.7, 2421, 80, TCP). Platné polia hlavičky ale nemusia byť obmedzené na spomenutú množinu, teda L2 až L7 polia ako MAC alebo URL môžu byť tiež použité pre klasifikáciu. Klasifikátor smerovača pozostáva zo sady pravidiel značených ako $R_1, R_2, \ldots, R_N$. Každé pravidlo z postupnosti je kombináciou hodnôt polí z d, pre každé pole jedna hodnota. Množiny záznamov ktoré môžu pod tieto pravidlá spadať sa môžu prekrývať. V klasifikácii sa stretávame s následujúcimi druhmi porovnaní:



Obr. 3.1: Priebeh klasifikácie paketov

Presné porovnanie: Musí sa zhodovať hodnota poľa H_k s hodnotou v pravidle $R_i(k)$. Požíva sa napríklad pre pole typu protokolu L4 alebo príznakov TCP (hodnoty ACK, FIN, SYN...)

Provnanie na zhodu prefixu: Hodnota pravidla $R_i(k)$ musí byť prefixom hodnoty paketu H_k. Požíva sa najmä pri smerovaní kde smerovacia tabuľka obsahuje v $R_i(k)$ adresu siete a v poli H_k je adresa konkrétneho počítača.

Intervalove porovnanie: Hodnota H_k musí byť v intervale zadanom pravidlom $R_i(k)$. Využitie nájde napríklad pri obmedzení rozsahu povolených portov. $H_k = 80$ vyhovuje pravidlu $R_i(k) = \langle 0; 511 \rangle$ Intervalové porovnanie je možno previesť na prefixové.

Porovnanie regulárneho výrazu: Používa sa pri analýze obsahu paketov, pri vyhľadávaniu reťazcov v poliach aplikačných protokoloch. Pri analýze vrstiev hlbších ako L4 hovoríme o Deep Packet Inspection (DPI).

Databáza pravidiel môže mať štruktúru ako v tabuľke 3.1. Aby sa pri klasifikácii vyhlo viacnásobným zhodám, poradie pravidiel určuje ich prioritu. Úlohou klasifikácie je teda nájsť zhodné pravidlo s najvyššou prioritou. Vyššie popísanou politikou sa riadia napríklad access control listy vo firewalloch, smerovače používajú najdlhšiu zhodu.

Pravidlo n	Cieľová adresa/prefix	Zdrojová adresa/prefix	Cieľový port	Protokol	Akcia
1	201.15.17.21/32	201.15.75.4/32	< 1024	*	zahoď
2	201.18.20.25/24	201.15.100.10/32	$= 80$	TCP	šifruj
3	201.15.20.25/24	201.15.100.10/32	< 1024	UDP	povoľ
4	201.21.12.1/16	201.75.75.75/16	< 1023	TCP	dešifruj
5	201.21.12.1/24	201.75.75.75/24	*	UDP	zahoď
6	*	*	*	*	povoľ

Tabuľka 3.1: Príklad databázy pravidiel

3.1 Algoritmy

Ako bolo spomenuté, množstvo sieťových zariadení využíva v istom bode svojej činnosti klasifikáciu paketov. Keďže na rozhodnutie spravidla býva obmedzené množstvo času a rýchla pamäť je drahá, pre efektívnu činnosť hľadáme optimálny algoritmus danú aplikáciu. Kvôli dôležitosti a zložitosti klasifikácie existuje veľa algoritmov pre jej realizáciu. Pre ich hodnotenie použijeme nasledujúce metriky zhrnuté nižšie: [4, sekcia 16.2.2]

Rýchlosť: Zvyčajne sa meria v počte prístupov do pamäte, kvôli jej časovej náročnosti. Réžia klasifikácie paketu nezáleží na jeho veľkosti, ale počet prichádzajúcich paketov áno. Rýchlosť klasifikácie by teda mala odpovedať maximálnej možnej rýchlosti príchodu paketov na danej linke (wire speed), obzvlášť na dôležitých vysokorýchlostných linkách.

Pamäťová náročnosť: Rýchlosť vyžaduje rýchlu pamäť pre dátovú štruktúru s pravidlami. Veľkosť potrebnej pamäte je priamo ovplyvnená jej typom a teda nepriamo určuje rýchlosť. SRAM pre nízku pamäťovú náročnosť a prístupovú dobu do 1 ns, DRAM pre vyššiu kapacitu a dobu 40-60 ns. Pokiaľ povaha prenosu na sieti vykazuje malý časový rozptyl medzi paketmi toho istého toku, je dobrá šanca že použitie cache prístup k pravidlám urýchli.

Náročnosť aktualizácie: Treba počítať s tým že u niektorých zariadení môže dochádzať k nutným zmenám v pravidlách. Je teda užitočné ak algoritmus umožňuje čiastočne upraviť dátovú štruktúru. Zápis do pamäte si vyžaduje podstatne viac času a to v ráde milisekúnd. Spôsob aktualizácie pravidiel sa líši podľa typu dátovej štruktúry, u niektorých dôjde len k pridaniu pravidla a rozšíreniu architektúry a iných je nutné jú znova celú vytvoriť, čo je časovo a priestorovo náročné. Typ cieľového zariadenia teda naznačuje výber algoritmu.

Dimenzia: Podľa počtu hlavičiek ktoré algoritmus dokáže spracovať rozdeľujeme metódy na jednodimenzionálne, dvojdimenzionálne a viacdimenzionálne. Ideálny algoritmus nieje obmedzený počtom dimenzií.

Škálovateľnosť: Algoritmus je použiteľný jak pre desiatky, tak pre tisícky pravidiel. Dôležité je aby nedošlo k výraznej degradácii výkonu pri náraste počtu pravidiel.

Zložitosť implementácie: Môže znemožniť implementáciu v hardvéri, čo môže zmariť nasadenie kvôli chýbajúcemu výkonu. Atraktívny algoritmus je implementovateľný ako v hardvéri tak softvéri.

Nasledujúce riadky obsahujú prieskum možných aplikovateľných algoritmov pre klasifikáciu paketov a teda aj tokov. Klasifikácia prebieha na základe porovnávania položiek pravidiel s položkami v hlavičke paketu. Keďže toky, teda agregácie paketov, obsahujú podobnou množinou položiek, možno usúdiť, že principálne sa jedná o rovnaký problém. Avšak stratégie vyhodnocovania pre pakety a toky sa líšia. Klasifikácia paketov na sieti musí byť rýchla a teda algoritmy optimalizované na konkrétne položky a počet dimenzií (IP prefixy, rozsahy portov, malé množstvo hodnôt pre protokoly), čo často zhoršuje flexibilitu, ktorá je nie menej potrebná pri analýze. Klady a zápory jednotlivých algoritmov budú spomenuté a zhodnotené.

Naivný prístup

Najjednoduchšia možnosť je usporiadať pravidlá do lineárneho zoznamu od najmenšej priority. Pre každý paket sa testujú pravidlá od začiatku zoznamu, kým nedôjde k zhode. Náročnosť klasifikácie rastie s počtom pravidiel je teda $O(N)$, avšak algoritmus sa zle škáluje a počet prístupov do pamäte pri klasifikácii rastie s počtom pravidiel. Pridanie a odobranie pravidla prebieha ako u lineárneho zoznamu.

Rýchlosť možno vylepšiť použitím cache pre výsledky posledných klasifikácií. Než sa teda pristúpi ku klasifikácii najprv sa prehľadá táto cache (kľúčom je hlavička paketu), čo vyžaduje jeden prístup do pamäte, na rozdiel od plnej klasifikácie ktorá vyžaduje niekoľko prístupov. Principálne, naivný prístup môže existovať aj na vyššej úrovni, miesto pravidiel môže jednotku zoznamu tvoriť napríklad klasifikátor pre jednu z N dimenzií a pod.

Trie

Skratka pre **re*trie*val**, ide najjednoduchší algoritmus stromového typu. Každý uzol môže mať dvoch až žiadneho potomka, prechod do ľavého podstromu je označený 0 do pravého 1. Je to štruktúra umožňujúca prirodzenú organizáciu IP prefixov a používa bity z polí (v poradí od MSB) pre smerovanie vetvenia pri vyhľadávaní. Uzol ku ktorému cesta od koreňa tvorí platný prefix obsahuje alebo odkazuje na informácie o danom pravidle.

Časová zložitosť vyhľadávania je $O(W)$ kde W je maximálna dĺžka prefixu v tabuľke pravidiel. Rovnaká zložitosť platí pre vkladanie a rušenie pravidiel. Pamäťová zložitosť je $O(NW)$ kde N je počet prefixov v tabuľke.

Hojne sa využíva pre vyhľadávanie v jednej dimenzii, neškáluje sa veľmi dobre čo sa týka počtu dimenzií. Fakt že polia porovnáva bit po bite predstavuje limitáciu pre rýchlosť. V najhoršom prípade bude vyžadovať vyhodnotenie jednej IP adresy 32 prístupov. Z toho dôvodu existuje optimalizácia na n-arny trie.

U **trie s viacbitovým krokom**, *multibit trie*, má každú uzol 2^k potomkov, kde k je krok (počet bitov porovnania). Krok môže byť pevný (každá úroveň stromu má rovnaký), alebo premenlivý. Použitie multibit trie vyžaduje tzv. rozšírenie prefixu – transformáciu pravidiel tak, aby prefixy boli obmedzené na dĺžky zložiteľné z kroku. Kde prefix túto vlastnosť nesplňuje dôjde k rozpísaniu pravidla na konkrétnejšie napr.: Majme krok 3 bity a prefixy $P_1 : 0011*$, $P_2 : 001111*$, u P_1 dĺžka prefixu nesedí s krokom. Musí byť expandovaný tak aby pravidlo zahŕňalo všetky hodnoty, ktoré pred tým. $P_1 \Rightarrow P_{1_1} : 001100*$, $P_{1_2} : 001101*$, $P_{1_3} : 001110*$. Pretože hľadáme najdlhšiu zhodu, pravidlo P_2 má prednosť pred expanziou. To predstavuje réžiu naviac pri vkladaní a mazaní pravidiel, zaujímavé ale je, že tieto operácie menia len určitý podstrom, sú lokálne. Pri mazaní je treba nielenže odstrániť pravidlo, mohlo byť aj expandované, ale aj zistiť, či neexistuje obecnejšie pravidlo ktoré by nahradilo to mazané.

Táto varianta má časovú zložitosť vyhľadávania $O(W/k)$, kde k označuje počet bitov jedného kroku. Nech pri aktualizácii je s krok podstromu, určuje koľko uzlov bude maximálne zmenených, $(2^s - 1)$, potom časová náročnosť aktualizácie je $O(W/k + 2^s)$. Pri implementácií je možno zlepšiť pamäťovú náročnosť použitím modifikácie trie s kompresiou.

Existujú ďalšie úpravy trie algoritmu: pre 2 dimenzie, so spätným vyhľadávaním, s ukazateľmi atď. podrobnosti o nich su ale mimo rozsah tejto práce.

Lucent Bit Vektor

Lineárne prehľadávanie pomocou bitového vektora patrí medzi algoritmy typu rozdeľuj a panuj. U týchto algoritmov existuje následovný spôsob: klasifikácia prebieha pre každú dimenziu samostatne a následne sú výsledky zložené pomocou istej operácie a požadované pravidlo identifikované. Klasifikácia v jednej dimenzii môže kľudne prebiehať algoritmom trie. Tento spôsob umožňuje implementovať klasifikáciu dimenzií paralelne. Majme demonštračnú tabuľku pravidiel. Pre každú dimenziu sa vyberie disjunktná množina hodnôt/prefixov a k ním bitový vektor vyjadrujúci ktoré pravidla v danej dimenzii túto hodnotu akceptujú. Každý bit vektoru označuje jedno pravidlo, teda vektor musí mať aspoň toľko bitov aký je počet pravidiel. Výsledky 1 dimenzionálnych klasifikácii sa následne spoja bitovým súčinom AND a pozície jednotkových bitov vo výsledku označujú čísla pravidiel ktorým záznam vyhovel.

Pri použití tejto metódy je časová náročnosť lineárna, nezávislá na počte dimenzií, $O(N)$, kde N je veľkosť bitového vektoru.

Existujú sofistikovanejšie metódy spájania čiastkových výsledkov ako klasifikácia pomocou karteziánskych súčinov alebo rekurzívna klasifikácia tokov (RFC), vychádzajúce z istých vlastností databáz klasifikátorov.

Algoritmus RFC [5] rieši exponenciálny nárast záznamov karteziánskych súčinov pri narastajúcom počte prehľadávaných dimenzií. Metóda vychádza s pozorovania, že veľké množstvo karteziánskych súčinov odpovedá rovnakým klasifikačným pravidlám. Možno povedať, že dva prefixy sú spolu v triede ekvivalencie keď vyhovujú rovnakým klasifikačným pravidlám. Princíp tejto metódy spočíva v tom, že výpočet čiastočných karteziánskych súčinov (napr. z dvoch dimenzií) je efektívnejší, teda dáva menej kombinácií, ako výpočet úplných karteziánskych súčinov. Pri vytváraní štruktúry sa rovnako, ako v lucent bit vektor algoritme, najprv vytvoria tabuľky jedinečných prefixov pre jednotlivé dimenzie, priradia sa im bitové vektory a vytvoria sa tabuľky čiastkových karteziánskych súčinov. Bitové vektory sa spoja AND bitovou operáciou a riadky s rovnakými vektormi tvoria jednu triedu ekvivalencie. Celkový výsledok sa netvorí skladaním dielčích kar. súčinov ale kombináciou tried ekvivalencie.

Pamäťová zložitosť tejto metódy závisí na variabilite hodnôt v jednotlivých dimenziách. Pre dva dimenzie je zložitosť $O(|M_i|.|M_j|)$, kde $|M_i|$ je počet prvkov v dimenzií i resp. j. V najhoršom prípade degraduje na $O(N^K)$ pre K-dimenzionálny klasifikátor a N hodnôt v každej dimenzii. Autori však uvádzajú, že pre všetky študované databázy pravidiel (počet pravidiel v každej cca 8000), mal RFC lineárnu pamäťovú zložitosť $O(N)$.

Výrazový strom

Jazyk popísaný v sekcií 3.3 sa skladá z logických výrazov obsahujúcich polia a podmienky pre popis pravidiel klasifikácie. Algoritmus implementovaný vo filtri používa binárnu stromovú štruktúru kde uzly stromu reprezentujú logické vzťahy medzi výsledkami podstromov a listy jednotlivé podmienky spolu s identifikáciou položky. Tento strom spravidla nemusí byť vyvážený, pretože skôr ako na vyhodnocovanie sa hodí na hierarchickú reprezentáciu výrazov. Výsledok filtrácie (ano/nie) sa získa prehľadávaním stromu do hĺbky, teda vyhodnocovanie sa vnára do podstromu vľavo, kým nenarazí na list, v ňom sa pristúpi k požadovanej položke záznamu a vyhodnotí sa na základe operátora a druhého operandu (prefixu/hodnoty) v liste. Booleovský výsledok sa spracuje v rodičovskom uzle a z neho sa pokračuje do pravého podstromu. Hodnota navrátená z koreňového uzlu je výsledkom filtrácie. K vnáraniu samozrejme dochádza len v prípadoch kedy daný podstrom existuje.

Tento spôsob sa dá vylepšiť použitím skratového vyhodnocovania, čo znamená, že v uzle nedôjde k zanoreniu do pravého podstromu pokiaľ je výsledok jasný, napr.: uzol AND, ľavý podstrom vrátil false a na hodnote pravého podstromu už nezáleží pretože sa uplatňuje princíp agresivity $0 \wedge x = 0$, takže výsledok uzlu sa nezmení.

Zložitosť vyhodnotenia jedného filtračného výrazu je najhoršie $O(2^n - 1)$, kde n je počet porovnávacích polí vo výraze, teda bude maximálne n prístupov do pamäte záznamu. Pamäť potrebná pre uloženie filtračného výrazu neprekročí $O(2^n - 1)$ uzlov. Zložitosti záležia na použitom filtri, uvedené platia pre schému "src ip X and src port < Y and $\cdots$"Výhodou je že tento algoritmus nie je obmedzený počtom dimenzií a umožňuje relatívne jednoduchý prevod z textového zápisu. Binárny strom výrazu možno považovať aj za vyhodnotiteľnú reprezentáciu filtra ktorú je možno previesť na optimálnejšiu a vyhodnotiť iným algoritmom. Tvorí tak pomyslený medzikód pri kompilácii jazyka filtra do klasifikačného programu.

3.2 Jazyky pravidiel

Vyššie popísané algoritmy transformujú pravidla v istej forme do špecifických dátových štruktúr. Zápis týchto pravidiel musí mať danú štruktúru, *jazyk*, inak by nebolo možné transformovať ich. Sieťové klasifikátory pracujú s pevným počtom dimenzií (v pravidlách majú vždy všetky polia) a teda kompletný popis jedného pravidla reprezentuje n-tica (vektor) dvojíc zložených z operátora hodnoty (operátor môže byť implicitný), teda pre každú dimenziu je rozhodnuteľné či daná hodnota vyhovuje. Majme klasifikátor pracujúci s N dimenziami, potom kompletné pravidlo bude $(\bigodot_1, x_1), (\bigodot_2, x_2), \cdots, (\bigodot_N, x_N)$, kde x_N je operand, $\bigodot_N$ reprezentuje binárny operátor ktorého výsledkom je booleovská hodnota. Záznam s hodnotami $y_1, y_2, \cdots, y_N$ vyhovuje pravidlu vtedy keď platí následovne: $\prod_{n=1}^{N} (y_n \bigodot_n x_n) = 1$. Medzi jednotlivými dimenziami pravidla teda implicitne platí súčin $\wedge$. Príklad takéhoto zápisu možno vidieť v tabuľke 3.1. Tento zápis však môže obsahovať opakujúce sa polia a pokiaľ nemáme záujem rozlišovať medzi skupinou pravidiel kde k tomu dochádza (degradujeme klasifikátor na filter)[1], existuje kompaktnejší zápis, avšak znemožňuje použitie mechanizmu ohodnotenia pravidiel pokiaľ je prevod realizovaný z plnej databázy do zjednodušeného jazyka. Zjednodušený zápis zvádza výrazy typu $\vee$ v rámci jednotlivých dimenzií čo umožní zapísať pravidla ako je načrtnuté v tabuľke 3.2. Existuje aj lepší popis pravidiel však, prevod z nich si vyžaduje istú znalosť jazykov.

D_1	D_2
10∗	1101
0110	1101

$\rightarrow (\bigodot_1, 10* \vee 0110), (\bigodot_2, 1101)$

Tabuľka 3.2: Príklad tabuľky pravidiel

Jazyk nástroja nfdump podporuje ľubovoľné skladanie výrazov pričom výskyt všetkých dimenzií vo filtre nie je nutný. Používa ešte voľnejšiu syntax ako vyššie spomenutá disjunktívna normálna forma $(u \vee v \cdots \vee x) \wedge (y) \cdots (z)$. Pre jednoduchý popis možno použiť nasledujúcu gramatiku:

Formálnym jazykom sa v počítačovej vede rozumie sada reťazcov zložených zo znakov spolu s pravidlami špecifickými danému jazyku. Abecedou je akákoľvek množina znakov/symbolov z ktorých daný jazyk skladá reťazce – slová. Formálne jazyky sa často definujú pomo-

[1] Klasifikátor rozlišuje pakety do N skupín, filter len do dvoch.

⟨*PRAVIDLO*⟩ ::= ⟨*VÝRAZ*⟩

⟨*VÝRAZ*⟩ ::= '`(`' ⟨*VÝRAZ*⟩ '`)`'
| '`not`' ⟨*VÝRAZ*⟩
| ⟨*VÝRAZ*⟩ '`and`' ⟨*VÝRAZ*⟩
| ⟨*VÝRAZ*⟩ '`or`' ⟨*VÝRAZ*⟩

⟨*VÝRAZ*⟩ ::= ⟨*pole/dimenzia*⟩ ⟨*operátor*⟩ ⟨*hodnota*⟩
| ⟨*pole/dimenzia*⟩ ⟨*hodnota*⟩
| ⟨*konštanta*⟩

Obr. 3.2: Zjednodušená gramatika jazyka nfdump

cou gramatík rôznych úrovní. V tejto práce sú využité deterministické regulárne a bezkontextové gramatiky pre definíciu filtračného jazyka. Preklad z jazyka do cieľového kódu (dátovej štruktúry) sa rieši prekladačom. Preklad môže vyzerať rôzne, avšak pri použití syntaxou riadeného prekladu bezkontextového jazyka bude postup približne následovný:

Lexikálna analýza má za úlohou rozsekať (tokenizovať) reťazec zdrojového programu na základné elementy (tokeny), teda musí ich nájsť, rozpoznať a zakódovať. Token je štruktúra reprezentujúca lexému, a slúži ako kategorizácia nedeliteľnej skupiny znakov pre účely syntaktickej analýzy. Jazyky bežne definujú svoje tokeny pomocou regulárnych výrazov, z týchto sa vytvorí stavový automat ktorý sa použije pre klasifikáciu reťazcov. V praxi býva lexikálny analyzátor realizovaný ako podprogram, volaný vždy keď je potrebný ďalší token.

Syntaktická analýza má za úlohu určiť či reťazce tokenov zdrojového programu tvora správne postupnosti. Inak povedané, pre danú postupnost tokenov sa snaží zostrojiť derivačný strom podľa naprogramovanej gramatiky.

Sémantická analýza overuje obsahy tokenov a ich zmysel v kontexte daného jazyka. Pri použití syntaxou riadeného prekladu sa zároveň so synt. analýzou sa spúšťajú sémantické akcie (napr. kontroly, konverzie, generovanie kódu/DS ...). Býva najkomplexnejšou fázou prekladu. Interpretácia preloženého programu, v našom prípade klasifikačnej databázy, už je vecou algoritmu vyhodnotenia.

Programovanie lexikálnej a syntaktickej analýzy býva náročná práca. Keďže sa pri písaní malých analyzátorov sústavne riešia tie isté problémy, boli vytvorené tzv. syntaktické konštruktory (napr. flex pre lexikálny a bison pre syntaktickú analýzu). Jedná sa o programy pre tvorbu analyzátorov ktoré zo špeciálneho fomátu pre popis (založenom na formálnom modely) vytvoria moduly pre použitie v rôznych jazykoch.

3.3 Nástroje pre filtrovanie

Kvôli popularite nástroja nfdump bude nový obecný filter navrhnutý tak, aby podporoval jeho syntax nfdumu, prípadne ho aj rozšíril o užitočné konštrukcie. Z toho dôvodu je v tejto sekcií uvedená aj podmnožina výrazov ktoré nfdump podporuje. Nasledujúce nástroje spadajú do kroku *analýza dát* v IPFIX architektúre a súvisia s obsahom tejto práce.

nfdump

Program je súčasťou veľmi populárnej sady open source nástrojov *nfdump* [6] s CLI rozhraním pre analýzu tokov v sieti a je voľne šíriteľný pod licenciou BSD. Nástroj je rozdelené na dve časti: prvá je zodpovedná za príjem dát z kolektorov a ukladanie do súborov. Druhá, reprezentovaná utilitou **nfdump**, umožňuje čítať a vypisovať v zrozumiteľnom tvare NetFlow v5 a v9 záznamy z týchto uložených súborov. Medzi podporované operácie so záznamami patrí:

- Filtrovanie pomocou flexibilného jazyka, zjednodušená gramatika na obrázku 3.2 slúži ako jeho náčrt. Jazyk sa syntaxou podobá na jazyk nástroja tcpdump.
- Agregácia na základe polí, pre určenie kľúča slúži zápis: $pole_1, pole_2, \cdots, pole_n$
- Výpis štatistík typu top N pre rôzne polia s možnosťou voľby zoradenia od najnižších resp. najvyšších hodnôt. Kľuč zoradenia sa zapisuje: $pole_1/pole_2/\cdots/pole_n$.
- Úpravy formátu výpisu záznamov.

Podrobný popis všetkých možností nástroja možno nájsť v jeho manuálových stránkach (inštalujú sa spolu s nástrojom).

Výstupom programu, po aplikácii filtra ak bol zadaný, je výpis výsledku dotazu alebo nový binárny súbor obsahujúci len požadované záznamy. Nástroj používa efektívne filtračné jadro založené na stromovom algoritme, napísané v jazyku C. Slúži (nie len) ako backend pre analýzy a profilovanie dát v nástroji NfSen. Jeho prednosťou je rýchlosť spracovania a flexibilita dotazov, avšak limitáciou je fakt, že nástroje sú úzko naviazané na tento formát súboru, čo spôsobuje možnú nutnosť s každou novou verziou reintegrovať vlastné vylepšenia formátu. Okrem toho, nfdump dotazovanie nie je pripravené pre viac vláknové vyhodnocovanie.

Gramatika nfdumpu je pomerne restriktívna. Pre každú položku net/ip/mac dovoľuje len nižšie špecifikované kombinácie, aj keď nič nebráni tomu aby napr. mac podporoval kľúčové slovo „in [list]". Vo filtračnom jazyku nfdumpu nájdeme nasledujúce výrazy[2] :

⟨*IP VERZIA*⟩ ::= ‘`inet`’ | ‘`ipv4`’ | ‘`inet6`’ | ‘`ipv6`’

Konštanty pre výber verzie ip protokolu

⟨*PROTOKOL*⟩ ::= ‘`proto`’ ⟨*int*⟩ |
‘`proto`’ ⟨*literál*⟩

Pole protokolu IP možno filtrovať bud podľa čísla alebo mena napr. TCP/UDP.

⟨*IP ADRESA*⟩ ::= ⟨*smer*⟩ ‘`ip`’ ⟨*ip adresa*⟩ |
⟨*smer*⟩ ‘`ip`’ in ‘`[`’ ⟨*zoznam ip*⟩ ‘`]`’ |
⟨*smer*⟩ ‘`host`’ ⟨*ip adresa*⟩ |
⟨*smer*⟩ ‘`host`’ in ‘`[`’ ⟨*zoznam ip*⟩ ‘`]`’ |
⟨*smer*⟩ ‘`net`’ ⟨*ip adresa*⟩ ⟨*ip maska*⟩ |
⟨*smer*⟩ ‘`net`’ ⟨*ip skrátená*⟩‘`/`’⟨*int*⟩

[2] Neterminály s názvom s veľkými písmenami.

⟨*smer*⟩ ::= ϵ |
‘`src`’ | ‘`dst`’ |
‘`src`’ ‘`and`’ ‘`dst`’ | ‘`dst`’ ‘`and`’ ‘`src`’ |
‘`src`’ ‘`or`’ ‘`dst`’ | ‘`dst`’ ‘`or`’ ‘`src`’

Umožňuje zadať v4/v6 adresu transparentne, dotazované pole sa odvodí podľa zadanej adresy. Kľúčové slovo ïn [list]"dáva možnosť dotazovať sa naraz na viac adries. Pri filtrovaní siete je možné zadať plnú IP s masku siete alebo jednoznačne určenú časť IP a prefix. 192.168/24 nestačí, chýba 3. oktet, 192.168/13 je jednoznačné. Slovo *net* implementuje prefixovú zhodu.

⟨*PORT*⟩ ::= ⟨*smer*⟩ ‘`port`’ ⟨*operátor*⟩ ⟨*int*⟩

⟨*operátor*⟩ ::= ϵ |
‘`<`’ | ‘`LT`’ |
‘`=`’ | ‘`==`’ | ‘`EQ`’ |
‘`>`’ | ‘`GT`’

Umožňuje aplikácie špecifikovaného operátora pre porovnanie poľa portu voči hodnote. Pokiaľ je operátor vynechaný, jedná sa o konkrétne (EQ) porovnanie.

⟨*PRÍZNAKY TCP*⟩ ::= ‘`flags`’ ⟨*príznaky*⟩

⟨*príznaky*⟩ ::= ‘`A`’ | ‘`S`’ | ‘`F`’ | ‘`R`’ | ‘`P`’ | ‘`U`’ | ‘`X`’ | ⟨*príznaky n*⟩

⟨*príznaky n*⟩ ::= ϵ | ⟨*príznaky*⟩

Na poradí príznakov nezáleží, tento výraz kontroluje či sú nastavené spomenuté príznaky, stav ostatných výsledok neovplyvňuje.

⟨*MAC ADRESA*⟩ ::= ⟨*smer1*⟩ ⟨*smer2*⟩ ‘`mac`’ ⟨*mac adresa*⟩

⟨*smer1*⟩ ::= ϵ | ‘`in`’ | ‘`out`’

⟨*smer2*⟩ ::= ϵ | ‘`src`’ | ‘`dst`’

Položka MAC môže byť bližšie špecifikovaná dvoma smerovými kvantifikátormi, bez nich kontroluje 4 polia.

⟨*MPLS LABEL*⟩ ::= ‘`mpls label`’⟨*index*⟩ ⟨*operátor*⟩ ⟨*int*⟩

⟨*MPLS EOS*⟩ ::= ‘`mpls eos`’ ⟨*operátor*⟩ ⟨*index*⟩

⟨*MPLS EXP*⟩ ::= ‘`mpls exp`’⟨*index*⟩ ⟨*operátor*⟩ ⟨*int*⟩

⟨*index*⟩ ::= [0-9] | 10

Multi protocol label switching (MPLS) položky majú viacslovné mená. Interne je mpls stack kompozintý dátový typ, pozostáva z 10 značiek. `mpls label1` filtruje číselný identifikátor prvej značky, `mpls eos 5` porovnáva či má značka 5 nastavený eos bit a `mpls exp1` filtruje 3 bitovú hodnotu príznakov na prvej značke.

⟨*POČÍTADLÁ*⟩ ::= ‘`packets`’ ⟨*operátor*⟩ ⟨*int*⟩ ⟨*jednotka*⟩ |
‘`bytes`’ ⟨*operátor*⟩ ⟨*int*⟩ ⟨*jednotka*⟩ |
‘`flows`’ ⟨*operátor*⟩ ⟨*int*⟩ ⟨*jednotka*⟩ |
‘`pps`’ ⟨*operátor*⟩ ⟨*int*⟩ ⟨*jednotka*⟩ |
‘`bps`’ ⟨*operátor*⟩ ⟨*int*⟩ ⟨*jednotka*⟩ |
‘`bpp`’ ⟨*operátor*⟩ ⟨*int*⟩ ⟨*jednotka*⟩ |
‘`duration`’ ⟨*operátor*⟩ ⟨*int*⟩

⟨*jednotka*⟩ ::= ϵ | ‘`k`’ | ‘`m`’ | ‘`g`’

Duration – trvanie sa zadáva v milisekundách. Faktor násobenia pre jednotky je 1000. Nasledujúce položky su počítané z: pps = $1000 * packets/duration$, bps = $1000 * bytes/duration$, bpp = $bytes/packet$.

libnf

Pôvodne mal libnf projekt riešit otázku rozhrania pre prácu s nfdump dátami v jazyku perl. Postupom času sa projekt vyčlenil do samostatného API v jazyku C [14], ktoré rozširuje možnosti nástroja nfdump, a jeho primárnou funkciou sa stalo práve poskytnutie silného rozhrania pre nadradené aplikácie. Jednou z motivácií k jeho vytvoreniu bol fakt, že možné úpravy nfdumpu nebudú kompatibilné až dôjde k aktualizácii, a de fakto ich človek musí implementovať znova. V budúcnosti umožní zmeniť formát súboru za flexibilnejší.

fdistdump

Fdistump je program pre distribúciu dotazovania pomocou knižnice libnf. Pracuje v móde master/slave a umožňuje beh na teoreticky neobmedzenom množstve výpočtových uzlov a dokonca vlákien v rámci každého uzlu. Okrem iného umožňuje vypísať, zoradiť a agregovať záznamy s možnosťou aplikácie mocného filtra záznamov. Program je napísaný v jazyku C s použitím knižnice Message passing interface (MPI) pre komunikáciu medzi procesmi. Je súčasťou sady programov pre distribuovaný IPFIXcol a analýzu dát.

Kapitola 4

Filter IP tokov

Filter bol implementovaný ako obecný modul pre filtrovanie dat pomocou jednoduchého jazyka. Bol zamýšľaný ako modul do programou IPFIXcol a knižnice libnf pre zjednotenie filtračného jazyka. Musí byť prispôsobiteľný, teda pre každú z aplikácii bude existovať vlastný adaptér – prispôsobenie rozhrania. Filter má poskytovať nasledujúce:

- Jazyk a prevod filtra do internej reprezentácie.
- Filtráciu záznamov podľa pravidiel v internej reprezentácii.
- Sadu interných dátových typov ktoré možno filtrovať.
- Rozhranie pre tvorbu adaptérov pre aplikácie.

4.1 Analýza

Cieľom práce bolo navrhnúť, implementovať a otestovať univerzálny interpret filtračných výrazov. Jazyk filtra má zachovať kompatibilitu so syntaxou jazyka nfdump [7], avšak musí umožniť filtrovať aj položky definované IPFIX protokolom [9]. Musí teda existovať mechanizmus, ktorý umožní nezávislosť filtra na konkrétnom formáte dat. Filter má podporovať skladanie výrazov pomocou základných logických operátorov and, or a not. Sémantika výrazov v nfdump a novom filtri má byť čo najpodobnejšia aby sa predošlo rozdielom vo výsledkoch v spoločnej sade polí.

Z teoretickej časti o klasifikácii poznáme jej architektúru na obrázku 3.1. Nový filter bude fungovať na základe podobnej mierne upravenej architektúre, tak aby sa dosiahlo požadovanej flexibility. Filtračný výraz bude preložený do stromovej dátovej štruktúry. Pre vyhodnotenie sa záznam spolu s filtrom predá vyhodnocovacej funkcii a návratovou bude vyhovuje/nevyhovuje. Pomocou funkcií pribalených v dátovej štruktúre pri preklade sa lookup funkciou priradia zadané názvy polí na unikátne ID. Pri vyhodnocovaní sa zo záznamu volaním data funkcie, na základe týchto ID, vyberú patričné dáta (záznam pre danú dimenziu) a vykoná sa operácia s daným dátovým typom. To sa opakuje pre každé pole v pravidlách.

Filter sa delí na jadro a adaptér. Úspešná implementácia filtra si bude vyžadovať:

jazyk Návrh kompatibilného jazyka, lexémy a syntax.

jadro Návrh jadra a jeho rozhrania. Definícia reprezentácie filtra po kompilácii, dostupných dátových typov a funkcii pre ich operátory, konverziu reťazcov z lexém. Návrh spôsobu predávania dát z aplikácie do evaluačnej funkcie.

adapter Prispôsobenie rozhrania špecifickej aplikácii a definícia funkcií ktoré sa predajú do dátovej štruktúry filtra – voľby.

testy Testy ktoré poslúžia počas implementácie pre overenie správnej funkčnosti a detekcii regresie pri ďalšom vývoji.

4.2 Návrh

Modul použije upravenú architektúru klasifikátora na obrázku 4.1. Filter bude rozdelený na **jadro** ktoré poskytuje preklad, filtračnú funkcionalitu a dátové typy a **adaptér** v ktorý zabezpečí prispôsobenie filtra na konkrétnu aplikáciu. Jadro bude poskytovať funkcie pre preklad do internej dátovej štruktúry, definície dátových typov a implementácie operátorov pre nich. Adaptér poskytne zoznam platných mien položiek/dimenzií, každému priradí ID, dátový typ ďalšie dôležité informácie a zabezpečí kompatibilitu externých a interných typov. Rieši tiež napojenie jednotlivých ID na dátové položky aplikácie. Poskytuje dva základné funkcie: lookup callback a data callback. Lookup funkcia sa bude volať pri preklade a jej úlohou bude vyhľadať dané meno filtračnej položky v tabuľke a vrátiť informácie o dimenzii (položke). Data funkcia bude volaná vždy pri vyhodnocovaní záznamu, a jej úlohou bude na základe id položky vybrať dáta zo záznamu, prípadne aplikovať konverziu či kombináciu (napr. polia `duration` a `pps` sa vypočítavajú z viacerých základných polí) a predať ich vyhodnocovacej funkcii.

Návrh jazyka je založený na nfdump filtračnom jazyku [7]. Následujúca gramatika zovšeobecňuje črty jazyka nfdump:

⟨*FILTER*⟩ ::= ⟨*VÝRAZ*⟩

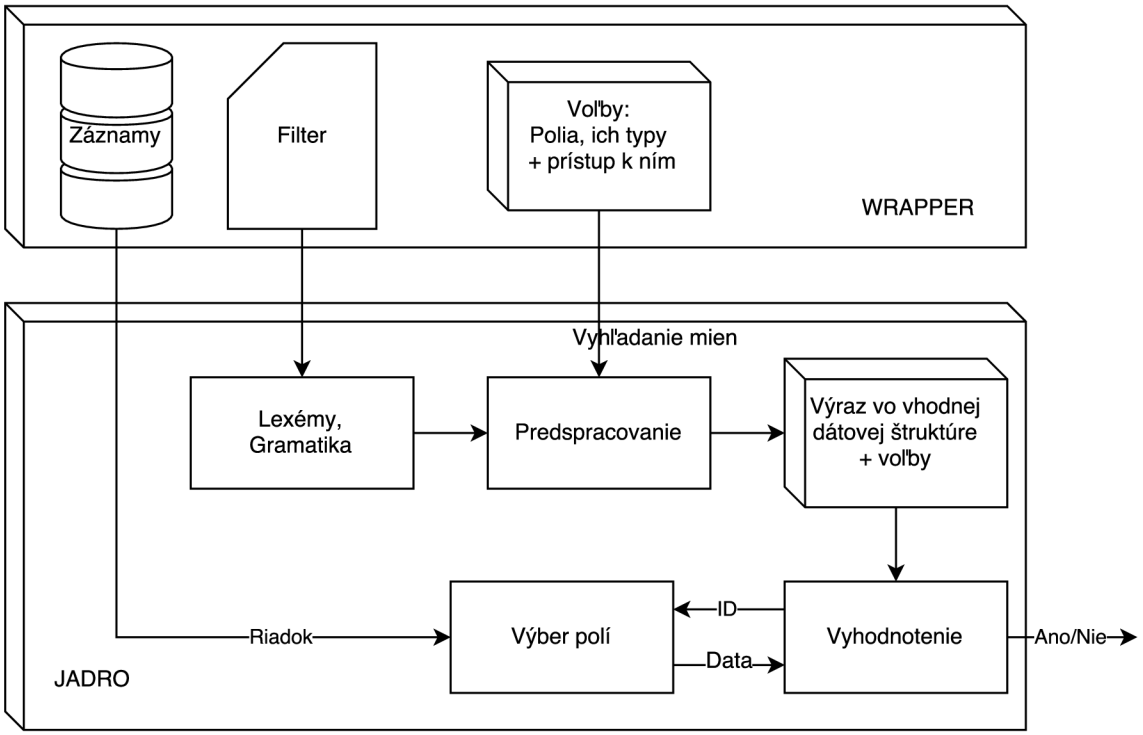
⟨*VÝRAZ*⟩ ::= '`(`' ⟨*VÝRAZ*⟩ '`)`'
| '`not`' ⟨*VÝRAZ*⟩
| ⟨*VÝRAZ*⟩ '`and`' ⟨*VÝRAZ*⟩
| ⟨*VÝRAZ*⟩ '`or`' ⟨*VÝRAZ*⟩

⟨*VÝRAZ*⟩ ::= ⟨*POLE*⟩ ⟨*OPERÁTOR*⟩ ⟨*HODNOTA*⟩
| ⟨*POLE*⟩ '`in`' '`[`' ⟨*ZOZNAM HODNÔT*⟩ '`]`'
| '`any`'
| '`exist`' ⟨*POLE*⟩

⟨*POLE*⟩ ::= ⟨*názov poľa*⟩
| ⟨*smerový kvantifikátor*⟩ ⟨*názov poľa*⟩

⟨*OPERATOR*⟩ ::= ϵ
| '`<`' | '`LT`'
| '`=`' | '`==`' | '`EQ`'
| '`>`' | '`GT`'
| '`&`'

⟨*HODNOTA*⟩ ::= číslo
| reťazec
| tcp_príznaky
| ip_adresa | mac adresa



Obr. 4.1: Bloková schéma filtra

Všetkým poliam sú povolené všetky spôsoby zápisu hodnôt a je na sémantickej časti analýzy zistiť na základe typu, či má daná kombinácia zmysel. Smerový kvantifikátor možno považovať za súčasť názvu poľa. Pokiaľ je položka *párová* (pod jedným menom sa skrýva porovnanie viacerých polí napr. "port "src port "dst port") filter zabezpečí že sa vyhodnotia všetky relevantné polia. Gramatika je rozšírená o výraz exist a operátor &. `Exist` kľúčové slovo je zavedené pre umožnenie kontroly prítomnosti dát daného poľa v zázname keďže IPFIX záznamy spravidla neobsahujú všetky políčka. Konštrukcia „`in [ zoznam ]`" skracuje zápis pre porovnanie viacerých hodnôt a umožňuje optimalizáciu vyhľadávania v jednej dimenzii. Like operátor „&", bude slúžiť pre približné porovnanie hodnôt ako napríklad vyhľadanie podreťazca, nastaveného bitu atď. záleží na type položky.

Zdrojový reťazec filtra (filter) sa pred prvým použitím musí inicializovať, teda preložiť do internej reprezentácie, výsledkom čoho je štruktúra obsahujúca strom výrazu, funkcie adaptéra a pamäť pre chybovú hlášku. Do inicializačnej funkcie vstupuje filter a funkcie implementované adaptérom (štruktúra volieb). Filter sa pomocou lexikálneho analyzátora rozseká na tokeny, tie sa predávajú do syntaktickej analýzy, ktorá overí správnosť postupností. Platné postupnosti ďalej prejdu sémantickou analýzou v ktorej dochádza ku:

- Rozpoznaniu mena položky a získaniu informácií o ňom volaním lookup funkcie. Neplatné meno položky spôsobí sémantickú chybu.
- Konverziám reťazcových hodnôt na dátový typ podľa typu priradeného danému poľu. Pokiaľ hodnotu nie je možno konvertovať, dochádza k sémantickej chybe.

Každý vyraz (neterminál) tvorí minimálne jeden uzol. And/or/not výrazy tvoria uzly stromu a porovnávacie výrazy listy. Z platných postupností sa zostavia uzly a listy z ktorých sa postupne vygeneruje celý strom výrazu.

Uzol stromu popisujúceho filter (strom výrazu) bude musieť uchovávať potomkov a typ operátora ktorý reprezentuje and/or/not. Listy sú komplikovanejšie, každý listový uzol reprezentuje porovnanie položky v jednej dimenzii záznamu. List potrebuje uchovávať hodnotu zadanú vo filtri, jej dátový typ, operátor/odkaz na funkciu s ktorou sa má aplikovať na hodnoty zo záznamu a listu. Štruktúra uzlov reprezentuje logické vzťahy medzi listami.

Medzi interné dátové typy filtra budú patriť základné celočíselné typy int a uint pevnej dĺžky 8, 4, 2 a 1 B, reálne čísla s pohyblivou des. čiarkou, reťazec, štruktúra pre uchovanie IPv4/6 adries zároveň, MPLS stack a časová značka. Typ MPLS je komplikovanejší, pretože vo výraze filtra v reťazci hodnoty je očakávaný dátový typ uint 4 B, ale zo strany záznamov sa očakáva celý MPLS stack (10 značiek), z ktorého sa voči danej uint hodnote vyhodnotí len relevantná časť. Časová značka mení očakávaný vstupný formát políčka z čísla na dátum a ten sa následné prevádza na číslo. Potreba týchto dátových typov vychádza z typov dát v záznamoch nfdump a IPFIX. Pre každý typ bude potrebná konverzná funkcia pre prevod hodnoty z reťazca do daného dátového typu.

Jadro bude poskytovať nasledujúce funkcie rozhrania:

Init funkcia pre danú implementáciu adaptéra preloží filter a vráti jeho objekt.

Eval nad daným objektom filtra vyhodnotí záznam, predaný ako obecný ukazateľ.

Free uvoľní zdroje alokované pre objekt filtra.

Error získa reťazec chyby z objektu filtra.

Set error nastaví reťazec chyby pre objekt filtra ak k nejakej počas prekladu došlo. Táto funkcia je určená pre použitie v adaptéri.

Pre preklad filtrov jadro potrebuje poskytnúť inicializačnej funkcií implementácie data a lookup funkcií. Tieto implementácie spolu s ďalším podporným kódom sa nazývajú adaptér. Úlohou adaptéra je za pomoci jadra preložiť filter, vypísať problém ak preklad zlyhal a vyhodnocovať záznamy. Adaptér teda bude poskytovať funkcie:

Lookup vráti ľavú hodnotu pre platný názov položky a inak nastaví chybu s objekte filtra. Sprístupňuje tabuľku platných názvov položiek a ich vlastností (ID, dátový typ, párovosť, zmena prednastaveného operátora prípadne ďalšie).

Data má za úlohu vyhľadávať položky v záznamoch počas vyhodnocovania. Zabezpečuje prevod typov aplikácie na typy jadra, pokiaľ nie su kompatibilné.

Adaptér by mal obaliť rozhranie jadra aby lepšie vyhovovalo danej aplikácii. Vyššie spomínané funkcie sa po preložení filtra začlenia do jeho objektu. Jadro poskytuje štruktúru pre ľavú hodnotu poľa, a teda musí byť schopné dekódovať v ňom prítomné informácie o položke. Ľavá hodnota sa nastaví v lookup funkcii a dekóduje vo funkcii ktorá vytvára list stromu.

Testy

Pre overenie správnosti fungovania filtra a pre predchádzanie regresiám budu navrhnuté testy. Pre overenie funkčnosti konverzií literálov vo filtri bude potrebné navrhnúť testy prevodných funkcií. Na tieto stačí jadro, bez potreby implementácie adaptéru. Komplexnejšie testy však potrebujú testovací adaptér. Jednoduchý adaptér môže fungovať následovne: pretože bude treba overiť minimálne dátové typy, ich operátory a funkčnosť párových položiek

vytvoria sa vhodné mená položiek, tak aby spĺňali požiadavky jazyka. Vytvoril som lookup funkciu, v nej sa položkám priradia patričné typy, ID a vlastnosti. Navrhol som štruktúru s položkami, ktorá bude suplovať formát záznamu a cez data funkciu sa jednotlivé ID naviažu na položky. Vytvoril som pomocné funkcie pre nastavovanie hodnôt rôznych typov v tomto zázname. Navrhol som sadu testovacích filtrov, ktoré overia funkčnosť/nefunkčnosť všetkých variant výrazov ktoré gramatika umožňuje. Pre test funkčnosti operátorov bude treba pre každý typ testovať všetky operátory.

4.3 Implementácia

Podľa popísaného návrhu bol filter implementovaný v jazyku C. Jednotlivé časti implementácie budú rozobraté nižšie, v sekciách.

Vstupný jazyk

Časť zdrojového kódu filtra je generovaná pomocou dvojice nástrojov flex (lexikálna) a bison (syntaktická analýza) [11]. Nimi vygenerovaný kód rieši prácu s reťazcom filtra a spúšťa sémantické akcie.

Lexikálna analýza rozlišuje nasledujúce typy tokenov: mená položiek, hodnoty, kľúčové slová. Názvy položiek sú obmedzené regulárnemu výrazu `[a-zA-Z][a-zA-Z0-9\-]*`, čo sú vlastné reťazce podobné C identifikátorom. Pokiaľ nastane potreba viacslovných mien, môžu sa zadávať v úvozdovkách. Hodnoty sú reťazce vyhovujúce regexu `[a-ZA-Z0-9:/\.\-]+`. Hodnota môže obsahovať maximálne jednu medzeru. Pre zložitejšie hodnoty sa použiije regex `\"[^\"]+\"`, čo umožňuje zadať akúkoľvek hodnotu. Kľúčové slová sa v identifikátore alebo hodnote môžu objaviť len vo forme s úvodzovkami. Reťazce smerových kvantifikátorov je možné do filtra písať oddelene od mena položky, ale, interne dôjde k jeho spojeniu s menom. To treba mať na pamäti pri tvorbe zoznamu platných položiek pre adaptér.

Bison generuje kód pre analyzátor syntaxe. Tento podľa zadanej gramatiky v BNF (bezkontextová) priraďuje ku každému pravidlu sémantickú akciu v podobe bloku C kódu a sprístupňuje v ňom jednotlivé lexémy (tokeny aj neterminály) pod menami $\$1, \$2, \cdots, \$n$. Tento blok kódu sa vykoná v momente redukcie pravidla. Ako prebieha analýza a dochádza k shift/reduce akciám na zásobníku analyzátora, postupne su spúšťané akcie pre vytvorenie derivačného stromu – stromu výrazu filtra. Tvorba uzlov a listov je parametrizovaná. Do funkcie vytvorenia uzlu vstupujú jeho potomkovia a typ logickej operácie. Do funkcie listu vstupuje meno položky, operátor porovnania a reťazec hodnoty.

Preklad na strom

Nech je filter reprezentovaný reťazcom `"src ip 192.168.0.1 and port < 1024"`, ip a port sú platné názvy polí, a hodnoty majú platnú konverziu do typu poľa. Zjednodušene prebehne syntaktická analýza z dola hore následovne: v každom kroku bude zobrazený stav a akcia. Nasledujúce slúži len pre ilustráciu, pôvodné názvy tokenov a neterminálov gramatiky sú skrátené. Reálne prebieha analýza komplikovanejšie, podrobne skúmanie fungovania analyzátora ale nie je predmetom tejto práce. Znak | reprezentuje vrchol zásobníka.

Na obrázku 4.2 je znázornený približný postup zostavenia derivačného strom výrazu filtra počas analýzy. [13]. Varianta listu bez operátora doplní špeciálny operátor, ktorý sa neskôr na základe informácií o položke odvodí. Filter definuje rovnakú štruktúru pre reprezentáciu uzlov a listov, rozdiel je v tom, že: 1) list nesmie mať potomkov, 2) má

Zásobník | **Ostavajúce tokeny**	**Akcia:**	**Pravidlo** **Popis**
| prefix id hodnota and id lt hodnota <eof>	shift:	prefix
prefix | id hodnota and id lt hodnota <eof>	shift:	id
prefix id | hodnota and id lt hodnota <eof>	shift:	hodnota
prefix id hodnota | and id lt hodnota <eof>	reduce:	prefix id hodnota → VÝRAZ Volaj funkciu pre vytvorenie listu s implicitnou operáciou a skontroluj sémantiku
VÝRAZ | and id lt hodnota <eof>	shift:	and
VÝRAZ and | id lt hodnota <eof>	shift:	id
VÝRAZ and id | lt hodnota <eof>	shift:	lt
VÝRAZ and id lt | hodnota <eof>	shift:	hodnota
VÝRAZ and id OP hodnota | <eof>	reduce: akcia:	prefix id OP hodnota → VÝRAZ Volaj funkciu pre vytvorenie listu skontroluj sémantiku
VÝRAZ and VÝRAZ | <eof>	reduce: akcia:	VÝRAZ and VÝRAZ → VÝRAZ Volaj funkciu pre vytvorenie and uzlu a pripoj potomkov
VÝRAZ | <eof>	reduce: akcia:	VÝRAZ → FILTER Substitúcia neterminálu.
FILTER | <eof>	shift: akcia:	<eof> Príjmi reťazec a vráť strom filtra.

Obr. 4.2: Príklad postupu pri analýze filtra metódou zdola hore.

nastavené ID položky, 3) v sebe neobsahuje operátory AND OR a NOT. Táto štruktúra okrem potomkov obsahuje:

Identifikátor položky definovaný adaptérom, vrátený lookup funkciou po overení názvu poľa.

Dátový typ hodnoty z reťazca filtra v uzle.

Hodnota je zjednotením všetkých možných typov, položka má veľkosť vždy 8 bajtov. Typy s pevnou maximálnou dĺžkou nepresahujúcou 8 B sú uložené hodnotou, ostatné odkazom do dynamicky alokovanej pamäte.

Dĺžka bajtov hodnoty je nulová pokiaľ je hodnota uložená priamo, inak obsahuje počet alokovaných bajtov. Používa pri deštrukcií stromu a vyhodnocovaní niektorých typov.

Evaluátor

Je kus kódu, funkcia ktorá na základe enumeračných hodnôt pre typy a operátory, rozhoduje aká operácia s dátami v uzle a s dátami záznamu bude vykonaná. Táto časť filtra bude najčastejšie používaná, Pretože v každom liste stromu dochádza k jej volaniu, bude to spolu s data callbackom najčastejšie používaná funkcia filtra. Konkrétne v kóde sa nazýva `ff_oper_eval`. Skúmaniu výkonu vyhodnocovania sa budem venovať v experimentálnej časti. Po kompilácii sa očakáva, že rozhodovanie bude realizované vo forme skákacej tabuľky.

Evaluátor očakáva vždy ukazateľ na dáta záznamu aby sa predošlo zbytočnému kopírovaniu dát.

IPFIX mediátor – filtrovanie počas zberu

Je to flexibilný kolektor záznamov o tokoch [1]. Jeho modulárna architektúra mu umožňuje zároveň so zberom rôznym spôsobom transformovať záznamy pomocou mediačných modulov. Možnosť filtrovať záznamy je užitočná z viacerých ohľadov a preto v ňom už existuje modul pre profilovanie.

Profilovanie v IPFIXcole [10, sekcia 3.2] funguje veľmi podobne ako v nfsen [7]. V pozadí tohoto pracuje filtračný modul, ktorý ale podporuje iba IPFIX položky a jeho syntax je odlišná od nfdump. Pre zjednotenie jazyka s fdistdumpom v rámci distribuovaného kolektora [15] bolo rozhodnuté, že z dôvodu použitia jednej implementácie je potrebné zmeniť filtračné jadro za nové, flexibilnejšie. Jeho použitie zabezpečí, že existujúce nfdump filtračné programy budú funkčné v IPFIXcole aj v knižnici libnf, a teda aj v fdistdumpe.

IPFIXcol vo svojej konfigurácii uchováva zoznam platných položiek, ich mien, ID, dátových typov, dĺžok a ďalších. To môže využiť lookup funkcia ktorá pristúpi k záznamom pomocou `get_element_by_name` a `get_element_by_id` API funkcie a prekóduje identifikátory IPFIX dátových typov na interné typy filtra. Takto sa zabezpečí zahrnutie IPFIX položiek do platných mien polí filtra ale nerieši to otázku kompatibility s nfdump jazykom. Zoznam položiek nfdump, ich synonyma a k ním priradené typy, ktoré bude filter podporovať musí adaptér poskytnúť dodatočne. Párové položky musia byť v zozname nejakým spôsobom odlíšené a musia obsahovať odkazované reálne položky. To je riešené tak, že v zázname obsahuje relatívne odkazy do tabuľky na patričné položky. Vyhľadanie mena teda prebehne v dvoch fázach: najprv sa prehľadá zoznam nfdump a pokiaľ neprebehne úspešne, zavolá sa API a bude sa vyhľadávať v konfigurácii elementov IPFIXcolu. Získané údaje o mene a type položky poslúžia pre priradenie interného typu vo filtri. Prístup k dátam položiek v záznamoch IPFIX realizuje API funkcia `data_record_get_field`. Hodnoty získané z funkcie ale máju svoje špecifika: 1) celočíselné typy majú zadanú maximálnu dĺžku, čo znamená že číslo môže byť uložené aj na 3, 5, 7 B. Tento stav momentálne nemôže nastať (IPFIXcol toto nedovolí), ale stále je možné z API získať čísla s obvyklými dĺžkami. 2) Viacbajtové hodnoty sú uložene s veľkou endianitou (MSB uložené na nižších adresách v pamäti). 3) Usporiadanie informácii v rámci typu nie je kompatibilné so žiadnym interným typom.

Tieto problémy môžu byť riešené dvoma spôsobmi: adaptér zariadi prekódovanie a vráti filtru odkaz na nové dáta alebo filter bude mať variantu interného typu ktorý vykoná potrebné transformácie pred vyhodnotením. Endianita a variabilná dĺžka sa v tomto adaptéri rieši špeciálnymi typmi, pre MPLS zásobník ma jadro špeciálny typ a funkcie. Všetky dalšie komplikácie sa musia vyriešiť v adaptéri. Virtuálne položky (duration, pps ...) a MPLS zásobník, musí adaptér zostaviť z obsahu z položiek IPFIXu pred predaním do jadra.

Prí realizácii adaptéra vyvstal problém s vyhodnocovaním nfdump nesúcich IPv4 a v6 varianty. Pre filter „`src ip 192/8`”, sa data funkcia pokúsi získat položku sourceIPv4Address, čo je v tomto prípade správne, ale, nfdump filter nerozlišuje akého typu adresu vráti. K zhode medzi v4 a v6 adresami nemôže dojsť. Problem nastane pri kombinácii s výrazom `exists`, to je ale možné vyriešiť dotazom na konkrétny IPFIX ekvivalent.

Libnf – nový filter nfdump

Táto knižnica [14], okrem iného, pristupuje cez interné rozhranie nfdumpu k jeho prekladovému a filtračnému jadru. Je pripravená použiť nový modul filtra, keďže pôvodne sa filter vyvíjal práve pre knižnicu Libnf. Poskytuje vlastné funkcie pre prácu s nfdump záznamami, prístup k položkám sprostredkúva `lnf_rec_fget`, čo bude vyžívať adaptér. Dáta zo spomínanej funkcie nemajú variabilnú dĺžku, nie je treba riešiť podobný problém ako u predošlého adaptéru. Libnf obsahuje štruktúru s menami polí nfdump, ich internými identifikátormi, dátovými typmi a IPFIX ekvivalentnými názvami. Problém s párovými položkami sa rieši podobne ako u predošlého adaptéru. Informácie v nej sú sprístupnené volaním funkcie `lnf_fld_parse`. Libnf nemôže podporovať všetky IPFIX polia, pretože pracuje s formátom súborov nfdump a tento obsahuje menší počet položiek a aj z týchto nie všetky majú IPFIX ekvivalenty. Napríklad nfdump položka `inet` má ekvivalent `ipVersion = 4`, ale `sysid` alebo `ident` nemá. Interné dátové typy jadra v podstate vychádzajú z tých ktoré používa knižnica, takže zložité konverzie nie su potrebné, dôjde len k preznačení ID na ekvivalentné.

4.4 Funkcionalita

Filter sám o sebe je len sadou podporných funkcií a dátových typov s flexibilným vstupným jazykom. Pomocou rozhrania adaptéra je treba do filtra doprogramovať informácie o dátových políčkach a prístupu k ním a tým prepojiť filtračný jazyk na konkrétne dátové štruktúry. To sa hodí, pretože sa podstatná časť filtra nemusí znova implementovať kvôli rôznym zmenám použitia. Filter spolu s adaptérom je schopný vyhodnocovať filtračné výrazy nad záznamami danej aplikácie.

Napriek snahe o jednu implementáciu použitie adaptérov mierne komplikuje úplnú kompatibilitu výsledných filtrov. Je treba zabezpečiť, že libnf a IPFIXcol adaptéry pracujú s ekvivalentnými zoznamami položiek a majú spoločnú aspoň sadu polí nfdump a ich IPFIX synonymá, celý zoznam je v tabuľke B.1. Existujú špecifické položky ktoré su prístupne len jednom adaptéri a nie je možné to zmeniť, pretože dané dáta inde neexistujú. Ako príklad možno uviesť informácie o exportéri (len pre IPFIXcol).

Kapitola 5

Experimenty, testovanie, vyhodnotenie výsledkov

5.1 Testovanie, výzvy

Pre testovanie som využil knižnicu google-test. Predmetom testov bolo overenie funkčnosti častí jadra filtra. Test rozhrania filtra definuje vlastný adaptér schopný testovať na vysokej úrovni väčšinu funkcií ktoré rozhranie poskytuje. Testy konvertorov umožňujú odhaliť správnosť konverzie reťazcových hodnôt na požadované dátové typy. Navrhol som tiež sadu pre overenie podpory položiek, a konštrukcií jazyka nfdump v adaptéroch, avšak tieto zatiaľ neboli implementované. Fakt, že adaptéri obyčajne používajú rozhrania aplikácií ku ktorým prispôsobujú jadro komplikuje implementáciu.

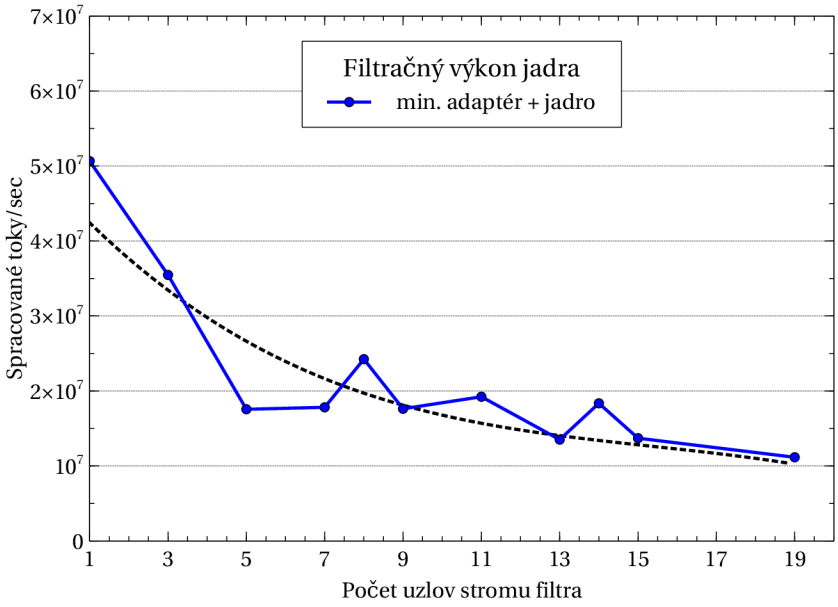
5.2 Výkonnostné testy

Pre zistene výkonu som navrhol niekoľko vzorových filtrov, nachádzajú sa v súbore tests/perf_test.in. Každý z týchto filtrov má priradený očakávaný počet uzlov v strome filtra po kompilácii. Táto sada bola použitá pre testovanie v nasledujúcich sekciách.

Pre každý filter bol zmeraný procesorový čas strávený vo funkcii pre vyhodnotenie záznamu v danej aplikácii. Výsledné počty záznamov za sekundu pre filtre s rovnakým počtom uzlov boli agregované priemerom do jednej hodnoty. Všetky testované programy boli skompilované prekladačom GCC 6.3.1, a testovanie bolo realizované na stroji s procesorom Intel® Core™ i7-5500U.

Filtračný výkon jadra

Pre testovanie výkonu jadra bol implementovaný minimálny adaptér s jednoduchou štruktúrou použitou ako formát dát. Pred zahájením testu sa záznamy naplnili pseudonáhodnými dátami, takže boli predom nahrané v pamäti. Namerané výsledky sú zobrazené na obrázku 5.1. Z grafu vyplýva, že časová náročnosť vyhodnocovania stúpa logaritmicky ako funkcia počtu uzlov filtra. Minimálne zaťažená eval funkcia teda pre filtre s počtom uzlov do 19, čo odpovedá asi 9 položkám, dokáže vyhodnotiť aj viac ako 10miliónov záznamov za sekundu.



Obr. 5.1: Test výkonu jadra

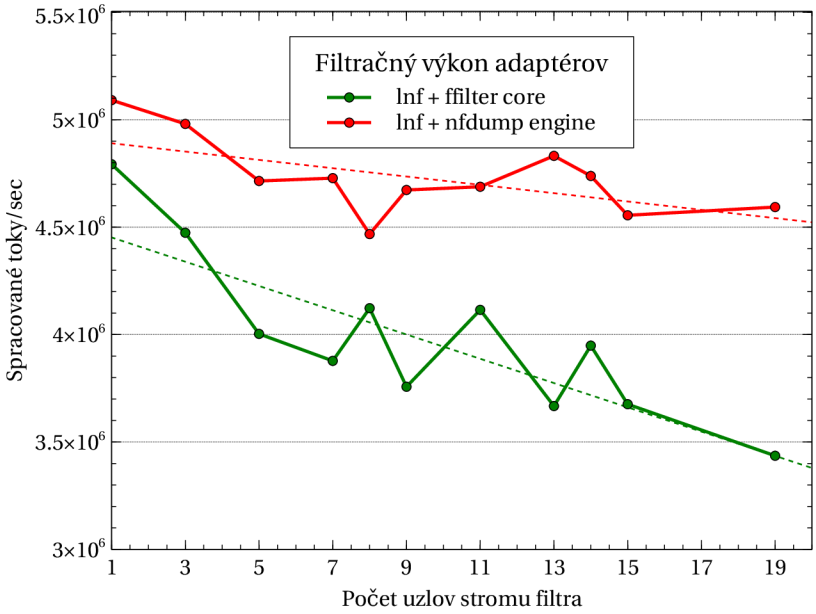
Porovnanie výkonu nfdump a filtra v knižnici libnf

Rozhranie knižnice libnf pomerne jednoducho umožnilo prístup k filtračnému jadru nfdump. Vďaka tomu bolo možné porovnať jeho výkon s výkonom nového filtra. Adaptér libnf pristupuje k záznamom cez rozhranie rovnomennej knižnice, to ale neodpovedá spôsobu akým so súborom pracuje čistý nfdump, avšak libnf API rovnako ovplyvňuje oba testované filtračné jadra a teda výsledné grafy vypovedajú o kombinovanom výkone libnf adaptéra s oboma filtrami. Pre účely testovania bol použitý súbor nfdump o veľkosti 500 MB s približne 20 miliónmi záznamov. Na obrázku 5.2 je zobrazený graf nameraného výkonu. Môžme usúdiť, že nové jadro zaostáva za nfdump jadrom približne o 18 % (ak výkon nfdumpu považujeme za 100 %) a jeho výkon s počtom uzlov degraduje 3x rýchlejšie.

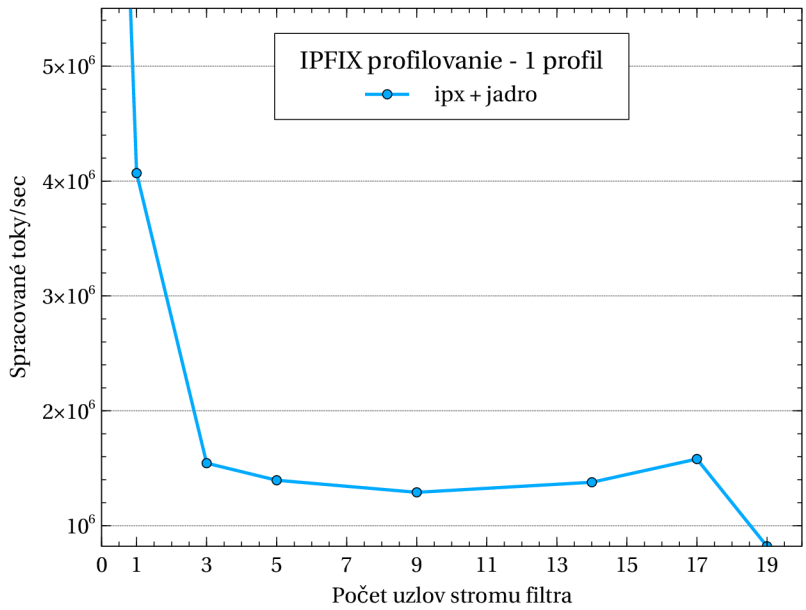
Test výkonu profilovania v IPFIXcol

Výkonnostný test adaptéra pre IPFIXcol prebehol odlišne od predošlých, za pomoci programu gprof som zmeral počty volaní funkcie ipx_filter_eval a jej spotrebu celkového času využitého programom. Zo zisteného času som na základe počtu záznamov vo spracovávanom súbore vypočítal filtračný výkon adaptéra. Graf nameranej výkonnosti je zobrazené na obrázku 5.3. Z grafu je možné vyčítať priemerný výkon približne 1,5 miliónu paketov za sekundu. Vzorka v bode x = 0 reprezentuje filter "any", tento nepristupuje k žiadnym dátam a preto je v nej výkon skreslený. Pozorované zníženie výkonnosti sa dá odôvodniť väčšou zložitosťou prístupu k položkám vo formáte IPFIX oproti formátu nfdump.

Pre účely testu boli použité ipfix vstupné a výstupné moduly so zapnutým profilovaním záznamov. Strom profilov činil jeden uzol s testovaným filtrom. Konfiguráciu programu a skript pre spúšťanie možno nájsť v prílohe.



Obr. 5.2: Test výkonu filtrovania v libnf



Obr. 5.3: Test výkonu adaptéra pre ipfixcol

Kapitola 6

Závěr

V práci je popísaná architektúra monitorovania v počítačových sieťach a načrtnutá problematika zachytávania zberu a ukladania získaných informácií. Pretože filtrovanie je základnou operáciou analýz v tomto kontexte a tiež priamo súvisí s praktickou časťou tejto práce je pre filtrovanie vyčlenená vlastná kapitola s popisom existujúcich algoritmov. Na základe týchto informácii a ďalších požiadavok ako flexibilita, kompatibilita a výkon som navrhol a popísal implementáciu modulu pre filtrovanie. Zmeral som jeho výkon porovnal s existujúcim nfdump filtrom.

Na základe výsledkov práce usudzujem, že som dokázal vytvoriť flexibilný, výkonný a štruktúrovane napísaný modul pre filtrovanie, ktorý je možno pomocou rozhrania prispôsobiť konkrétnym aplikáciam, bez toho aby bol silne viazaný na konkrétny formát dát. K nemu sú dostupne testy rozhrania pre účely ďalšieho vývoja a overenia funkčnosti.

Výhodou filtra oproti nfdumpu je jeho prispôsobiteľnosť vyšším aplikáciam a teda fakt, že nie je viazaný na konkrétny typ súboru, navyše jazyk ktorý používa pre zadávanie filtrov zahŕňa jazyk nfdumpu. Jadro filtra je výkonnostne na úrovní 78 % jadra nfdumpu a hoci je pomalší, stále vyhodnotí približne 4 000 000 Tokov/s. Jeho hlavnou vlastnosťou je ľahká rozšíriteľnosť, či už pridaním interného typu alebo použitím adaptérov, za cenu nižšieho výkonu na jednom vlákne. Spomínaná 3x rýchlejšia degradácia v závislosti na počte uzlov ako vykazuje nfdump súvisí s tým, že pre vyhodnocovanie sa používa derivačný strom generovaný pri preklade, kde došlo k jeho neoptimálnemu usporiadaniu.

V ďalšej práci na filtri by bolo dobé začať implementáciou vyváženia stromu výrazu bez zmeny jeho významu, keďže sa ukázalo že derivačný strom netvorí optimálnu štruktúru. Implementácia testu sady položiek nfdump pre adaptéry libnf a IPFIXcol zatiaľ nebola zrealizovaná. Z pohľadu užívateľa by iste prospelo vylepšenia chybových hlášok pri preklade.

Ďalšími možnými vylepšeniami by bolo: 1) umožniť filtru kompilovať viacero filtračných výrazov do seba a v podstate zmeniť filter na klasifikátor, to by využil program IPFIXcol. 2) Optimalizovať viac násobný prístup k dátam jednej položky, 3) optimalizovať uloženie stromu výrazu buď na úspornejší heap – strom v lineárnom poli, kde sú potomkovia vypočítaný z polohy uzla v danom poli, alebo na dátovú štruktúru RFC a zároveň previesť pôvodný strom výrazu na riadky v DNF forme. Vylepšenie štruktúry na RFC má obzvlášť zmysel, pokiaľ by malo dôjsť k prechodu z filtra na klasifikátor.

Filter poskytuje požadovanú funkcionalitu a je z veľkej časti hotový. Implementácia filtra je dostupná ako knižnica pre jazyk C na priloženom médiu. Novšie verzie sa budú objavovať na online repozitári github [16].

Literatúra

[1] CESNET: *IPFIXcol git repozitář*. [Online; navštíveno 20.6.2016].
URL https://github.com/CESNET/ipfixcol

[2] CESNET liberouter: *FlowMon Probe Handbook*. [Online; navštíveno 5.5.2017].
URL http://www.liberouter.org/package_releases/flowmon-1.3.0/flowmon-handbook.html

[3] Claise, B.; Trammell, B.; Aitken, P.: *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. [Online; navštíveno 12.3.2017].
URL https://tools.ietf.org/html/rfc7011

[4] Deepankar, M.; Karthikeyan, R.: *Network Routing. Algorithms, Protocols and Architecture*, kapitola 16 IP Packet Filtering and Classification. Elsevier Inc., 2007, ISBN 9780120885886.

[5] Gupta, P.; McKeown, N.: Packet classification on multiple fields. *Proceedings of ACM SIGCOMM*, 1999: s. 147–160.

[6] Haag, P.: *nfdump git repozitář*. [Online; navštíveno 20.4.2017].
URL https://github.com/phaag/nfdump

[7] Haag, P.: *NfSen readme*. [Online; navštíveno 12.4.2017].
URL http://nfsen.sourceforge.net/#mozTocId652064

[8] Hofstede, R.; Čeleda, P.; Trammell, B.; aj.: Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*, ročník 16, č. 14762503, Květen 2014: s. 2037–2064.

[9] IANA: *IP Flow Information Export (IPFIX) Entities*. [Online; navštíveno 20.2.2017].
URL http://www.iana.org/assignments/ipfix/ipfix.xhtml

[10] Kozubík, M.: *Profilování dat pomocí IPFIX mediátoru*. bakalářská práce, VUT v Brně, FIT, Brno, 2015.

[11] Levine, J. R.: *flex & bison*. O'Reilly Media, august 2009, ISBN 9780596155971.

[12] Matoušek, P.: *Síťové aplikace a jejich architektura*. VUTIUM, 2014, ISBN 9788021437661.

[13] Niemann, T.: *Yacc theory*. [Online; navštíveno 3.5.2017].
URL http://epaperpress.com/lexandyacc/thy.html

[14] Podermanski, T.: *Libnf git repozitář*. [Online; navštíveno 20.6.2016].
URL https://github.com/VUTBR/libnf

[15] Wrona, J.: *Optimalizace distribuovaného kolektoru síťových toků*. diplomová práce, VUT v Brně, FIT, Brno, 2016.

[16] Štoffa, I.; Podermanski, T.: *Ffilter git repozitář*. [Online; navštíveno 14.10.2016].
URL https://github.com/VUTBR/libnf-ffilter

Prílohy

Príloha A

Obsah priloženého pamäťového média

V koreňovom adresári CD sú umiestnené nasledujúce súbory:

extras/	Adresár obsahuje skript, konfiguráciu pre spúšťanie výkonnostných testov IPFIXcolu a vzorky záznamov v nfdump a ipfix formáte.
thesis/	Obsahuje zdrojové súbory textovej časti bakalárskej práce
ipfixcol/	Obsahuje verziu programu kolektora ipfix s adaptérom pre jadro. Adaptér sa nachádza na ceste base/src/utils/filter/ v súboroch filter_wrapper.c/h
libnf/	Obsahuje verziu knižnice libnf s adaptérom pre jadro. Adaptér sa nachádza v súboroch lnf_filter.c/h
libnf-ffilter/	Obsahuje zdrojové kódy modulu filtra, testy, výkonnostné testy a súbor README.MD s popisom prekladu
BP-Stoffa.pdf	Text teoretickej práce.
README.txt	Popis obsahu disku a detaily k integrácii filtra s ostatnými programami.

Príloha B

Množina kompatibilných nfdump položiek a ich IPFIX synonoym

Tabuľka B.1: Tabuľka kompatibilných nfdump položiek

nfdump meno	**IPFIX meno**	**typ**	**poznámka**
ip	–	ip	párová, nejednoznačná
src ip	sourceIPv4Address/ sourceIPv6Address		nejednoznačná
dst ip	destinationIPv4Address/ destinationIPv6Address		nejednoznačná
port	–	int 2B	párová
src port	sourceTransportPort		–
dst port	destinationTransportPort		–
if	–	int 4B	párová
in if	ingressInterface		–
out if	egressInterface		–
mac	–	mac	párová (4)
in mac	–		párová in(src/dst)
out mac	–		párová out(src/dst)
src mac	–		párová src(in/out)
dst mac	–		párová dst(in/out)
in src mac	sourceMacAddress		–
out dst mac	postDestinationMacAddress		–
in dst mac	destinationMacAddress		–
out src mac	postSourceMacAddress		–
as	–	uint 4B	párová
src as	bgpSourceAsNumber		–
dst as	bgpDestinationAsNumber		–
next as	bgpNextAdjacentAsNumber	uint 4B	–
prev as	bgpPrevAdjacentAsNumber	uint 4B	–
inet	ipVersion	uint 1B	konštanta
ipv4	ipVersion	uint 1B	konštanta

Pokračuje na ďalšej strane

Tabuľka B.1 – *Pokračovanie z predošlej strany*

nfdump meno	**IPFIX meno**	**typ**	**poznámka**
inet6	ipVersion	uint 1B	konštanta
ipv6	ipVersion	uint 1B	konštanta
proto	protocolIdentifier	uint 2B	–
icmp-type	icmpTypeIPv4	uint 1B	–
icmp-code	icmpCodeIPv4	uint 1B	–
engine-type	engineType	uint 1B	–
engine-code	engineId	uint 1B	–
flags	tcpControlBits	uint 2B	bitové porovnanie, kódovaný vstup
next ip	ipNextHopIPv4Address/ ipNextHopIPv6Address	ip	nejednoznačná
bgpnext ip	bgpNextHopIPv4Address/ bgpNextHopIPv6Address	ip	nejednoznačná
router ip	exporterIPv4Address/ exporterIPv6Address	ip	nejednoznačná
first	flowStartSysUpTime	uint 4B	–
last	flowEndSysUpTime	uint 4B	–
packets	packetDeltaCount	uint 4B	–
bytes	octetDeltaCount	uint 4B	–
flows	deltaFlowCount	uint 4B	–
tos	ipClassOfService	uint 1B	–
dst tos	postIpClassOfService	uint 1B	–
pps	–	uint 4B	virtuálne pole, packets/duration
duration	–	uint 4B	virtuálne pole, last-first
bps	–	uint 4B	virtuálne pole, bytes/duration
bpp	–	uint 4B	virtuálne pole, bytes/packets
mpls label1–10	–	mpls stack	vyber x–tu značku
mpls exp	–	mpls stack	vyber bity na vrchole
mpls eos	–	mpls stack	vypočítaj vrchol