VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

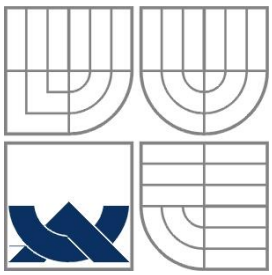
OPTIMALIZACE PROHLEDÁVÁNÍ NETFLOW DAT NÁSTROJEM NFDUMP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

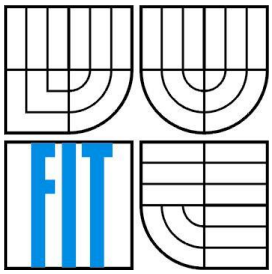
AUTOR PRÁCE
AUTHOR

MARTIN KUBOVIČ

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE PROHLEDÁVÁNÍ NETFLOW DAT NÁSTROJEM NFDUMP

OPTIMIZATION OF NETFLOW DATA SEARCH USING NFDUMP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN KUBOVIČ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV BARTOŠ

BRNO 2015

Abstrakt

Bakalářská práce se zabývá optimalizaci prohledávaní NetFlow dát nástrojem nfdump. Práce popisuje NetFlow protokol a nástroj nfdump a navrhuje řešení s využitím datové struktury Bloomův filtr. Cílem práce bylo optimalizovat ukládaní a zpracovaní dát tak, aby bylo možné prohledat obrovská množství sesbíraných dát a dostat výsledky v nejkratším čase. Výsledkem práce je optimalizovaný nástroj, který můžu správci síti použit při prohledáváni těchto dát a výrazně si tak zrychlit práci s monitorováním a analýzou sítě.

Abstract

This bachelor thesis deals with optimization of NetFlow data search using the nfdump tool. This thesis describes NetFlow protocol and tool nfdump and proposes the solution using data structure Bloom filter. The main goal was to optimize data storage and processing in order to be able to search the huge amounts of collected data and get results very quickly. The outcome of this thesis is the optimized tool that network administrators can use to search these data and significantly accelerate monitoring and analyzing network.

Klíčová slova

NetFlow, nfdump, Blomův filtr, optimalizace, monitorováni sítě

Keywords

NetFlow, nfdump, Bloom filter, optimization, network monitoring

Citace

Kubovič Martin: Optimalizace prohledávání NetFlow dat nástrojem nfdump, bakalářská práce, Brno, FIT VUT v Brně, 2015

Optimalizace prohledávání NetFlow dat nástrojem nfdump

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pána Ing. Václava Bartoša. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Kubovič
20. mája. 2015

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Václavovy Bartošovy za podporu a rady při vytvářeni této práce.

Obsah

1 Úvod

Počítačové siete tvoria v dnešnej dobe základ komunikácie medzi strojmi a často i medzi ľuďmi. Tieto siete sú monitorované sieťovými správcami a štatistiky zo sieťovej prevádzky sa ukladajú do súborov na pevný disk. Tieto štatistiky slúžia poskytovateľom internetových služieb na účtovanie cien za sieťové služby na základe prenesených dát. Sieťoví správci zase využívajú štatistiky na detekciu anomálií na sieťach, odhaľovanie slabých a silných miest siete, na plánovanie budúceho rozvoja alebo na sledovanie kto s kým komunikoval, akým protokolom a koľko dát preniesol. Sieťové toky na chrbticových sieťach však vytvárajú obrovské množstvá štatistík a ukladanie záznamov o týchto tokoch vytvára desiatky až stovky GB dát za deň, ktoré obsahujú milióny až miliardy záznamov. I napriek tomu že sú dnes počítače veľmi výkonné, spracovanie a prehľadanie takéhoto množstva dát môže nejaký čas trvať. Hlavne pri vyhľadávaní konkrétnych záznamov s požadovanou adresou, ktoré tvoria len malé percento z celkových dát sa zbytočne spracúvajú všetky záznamy, i tie nevyžadované čo spôsobuje nadbytočnú réžiu. Cieľom tejto práce preto bolo optimalizovať nástroj a upraviť súbory, do ktorých sa ukladaj tieto dáta tak, aby vyhľadanie a spracovanie záznamov bolo podstatne rýchlejšie s minimálnou nadbytočnou réžiou.

Práca by sa dala rozdeliť na dve časti, teoretickú a praktickú. V teoretickej časti kapitola 2 rozoberá poznatky, ktoré bolo nutné vedieť a pochopiť pre správne porozumenie práce. V prvej časti kapitoly je podrobne rozpísaný NetFlow protokol, využívaný na sieti pre zbieranie a ukladanie štatistík o sieťovej prevádzke. Druhá časť kapitoly rozpisuje nástroj nfdump, ktorý je základom pre túto prácu. Práca sa zaoberá práve optimalizáciou tohto nástroja. Najprv je vysvetlený princíp nfdumpu a následne je popísaná priamo jeho činnosť v implementácií. V záverečnej časti kapitoly je vysvetlený Bloomov filter, dátová štruktúra využitá pri optimalizácií.

V praktickej časti kapitola 3 podáva návrh na riešenie, akým spôsobom sa optimalizácia dosiahne. V nadväzujúcej kapitole 4 je už podrobne rozpísaná implementácia, ktorá z tohto návrhu vychádza. Celý princíp funkčnosti programu je vždy rozpísaný z dvoch pohľadov. Najprv z pohľadu vytvorenia nového, upraveného súboru a následne z pohľadu spracovania tohto súboru upraveným nástrojom s využitím optimalizácie.

Kapitola 5 sa zaoberá testovaním a výsledkami testov. Základom testov bolo porovnať pôvodnú verziu programu a súborov s upravenými a zistiť aký dopad mala modifikácia na výslednú veľkosť a spracovanie dát. Výsledky a prínosy práce sú zhrnuté v záverečnej kapitole 5.1.

2 Teoretický rozbor

Táto kapitola rozoberá základne teoretické znalosti, ktoré sú základom pre následný návrh a implementáciu optimalizácie nfdumpu. V kapitole je podrobne rozpísaný NetFlow protokol, ktorý popisuje formát dát. Následne je detailne popísaný nástroj nfdump, ktorý je základom pre túto prácu, preto je nutné poznať jeho formát a princíp. Na konci kapitoly je rozobratý Bloomov filter, dátová štruktúra využitá pri optimalizácií.

2.1 NetFlow

NetFlow je sieťový protokol vyvinutý spoločnosťou Cisco Systems slúžiaci na monitorovanie toku sieťovej prevádzky a zber dát. NetFlow architektúra sa skladá z dvoch primárnych častí: NetFlow exportér a NetFlow kolektor (1).

NetFlow exportér, obvykle cisco router alebo samostatné zariadenie, zbiera dáta zo sieťovej prevádzky a odosiela ich na kolektor. NetFlow kolektor je vysokokapacitné zariadenie ktoré prijíma dáta z exportéra, spracováva ich a ukladá na disk.

Základom NetFlow technológie je IP tok, z ktorého sa generujú štatistiky. Tok je definovaný ako sekvencia paketov so zhodnou zdrojovou/cieľovou IP adresou, zhodným zdrojovým/cieľovým portom a zhodným transportným protokolom. Pre každý tok sa zaznamenáva doba jeho vzniku, dĺžka trvania, počet prenesených paketov a bajtov a ďalšie údaje. Tok sa považuje za ukončený jedným z týchto spôsobov:

- U TCP je to detekované ukončením spojenia podľa FIN, RST bitu
- U UDP tato možnosť nie je preto sa tok považuje za skončený ak sa nejaký presne definovaný čas neobjaví žiaden paket z daného toku.
- Pre dlho pretrvávajúce toky sa tok považuje za ukončený po nejakom definovanom čase
- Zaplnenie Flow pamäte na exportéri môže spôsobiť predčasne ukončenie monitorovania toku

2.1.1 Popis protokolu

Protokol ma mnoho rôznych verzií, najpoužívanejšou sa stala verzia 5 a v súčasnosti sa najviac využíva verzia 9, ktorá ma upravený formát a je viac dynamická a flexibilná vďaka novému prístupu spracovania dát. Medzi novinky vo verzií 9 patrí podpora IPv6 a prístup k dátam prostredníctvom šablón.

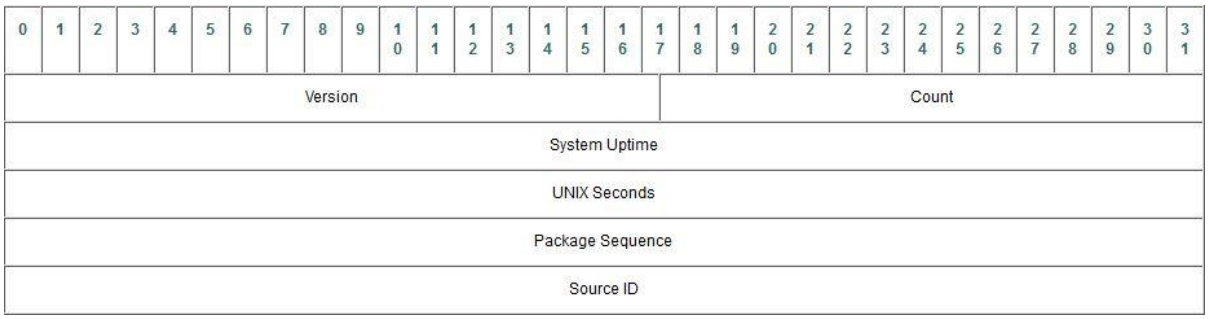
Záznamy sa štandardne exportujú pomocou User datagram protokolu (UDP) na porte 2055, prípadne 9555, 9995 alebo inom. Kvôli efektivite exportér po exportovaní záznam hneď vymaže a v prípade že sa paket po ceste stratí a nedorazí na kolektor, nie je možné ho obnoviť. To sa môže kvôli vlastnostiam UDP protokolu ľahko stať preto sa v súčasnosti často využíva Stream Control Transport Protokol (SCTP), ktorý na rozdiel od UDP zaručuje spoľahlivosť. Podrobný popis protokolu sa nachádza v dokumente (2).

2.1.2 Popis formátu

NetFlow formát verzie 9 využíva šablóny kvôli viac flexibilnejšiemu a rozšíriteľnejšiemu spôsobu predávania dát. Šablóny definujú význam jednotlivých polí v dátovej štruktúre. To umožňuje pridávať nové polia bez potreby zmeny štruktúry formátu a pridáva možnosť exportovať na kolektor len vyžadované polia.

Exportovaný paket sa skladá z hlavičky a jedného alebo viacerých Flowsetov. Flowset môže byť šablóna, dáta alebo šablona možnosti. Obrázok 2.1 zobrazuje príklad formátu paketu, avšak počet a poradie FlowSetov môže byť rôzny.

Obrázok 2.1 Formát Exportovaného paketu



Obrázok 2.2Hlavička exportovaného paketu (3)

2.1.2.1 Packet Header (Hlavička NetFlow paketu)

Podobne ako iné protokoly, i NetFlow ma svoju hlavičku, ktorá rozlišuje pakety, udáva ich poradie či zdroj paketu (3).

Popis položiek hlavičky:

Version – verzia exportovaného formátu, v našom prípade hodnota 9

Count – počet záznamov v exportovanom pakete

System Uptime – čas v milisekundách od prvého nabootovania zariadenia

UNIX Seconds – počet sekúnd od 0000 UTC 1970

Sequence number – inkrementálne počítadlo všetkých paketov, ktoré dané zariadenie odoslalo.

Využíva sa na detekciu chýbajúceho paketu.

Source ID – 32 bitová hodnota, ktorá jedinečne identifikuje zariadenie z ktorého bol záznam exportovaný.

2.1.2.2 Template FlowSet (Šablóny)

Jedným zo základných elementov NetFlow v9 formátu sú šablony. Šablóny definujú význam jednotlivých polí v dátových záznamoch čím zlepšujú flexibilitu záznamov, pretože kolektor nemusí poznať pevnú štruktúru a umožňuje to teda pridávanie vlastných polí. Jeden Flowset môže obsahovať viacero šablón s rôznymi identifikátormi. Formát šablóny popisuje Obrázok 2.3

Popis položiek šablóny:

FlowSet ID – identifikátor FlowSetu, pre šablóny je vyhradená hodnota 0

Length – Celková veľkosť FlowSetu

Template ID – každá šablóna ma unikátne ID, slúži na namapovanie šablóny na konkrétny set dát. Pre dáta sa využívajú hodnoty 256-65535, hodnoty 0-255 sú rezervované pre špeciálne typy FlowSetov.

Field Count – počet polí pre tento záznam šablóny

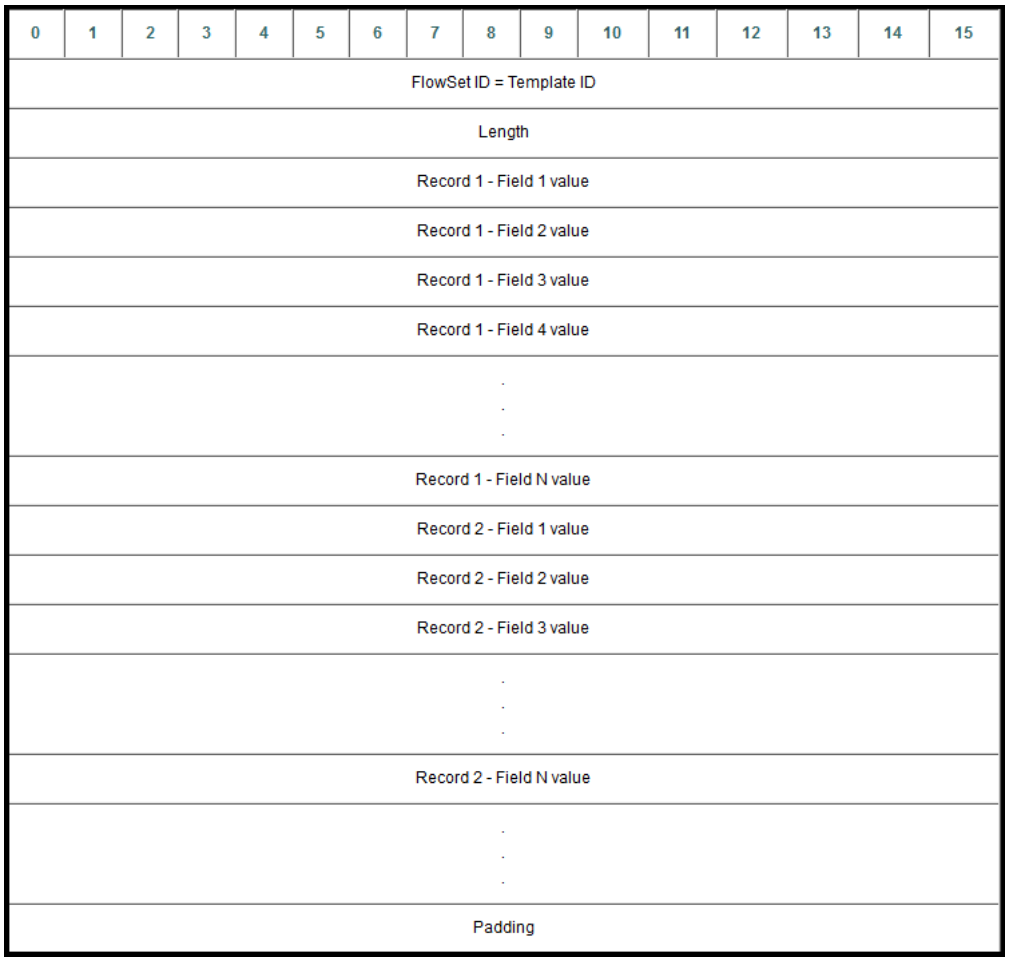
Field Type – numerická hodnota, ktorá reprezentuje typ poľa

Field length – veľkosť príslušného typového poľa v bajtoch

Obrázok 2.3Formát šablóny NetFlow záznamu (4)

2.1.2.3 Data FlowSet (Dáta)

Data FlowSet obsahuje samotné dáta exportované z exportéra, ktoré kolektor interpretuje podľa prislúchajúcej šablóny.



Obrázok 2.4Formát Data FlowSetu (4)

Popis Data Flowsetu:

FlowSet ID – u dát je FlowSet totožný s Template ID prislúchajúcej šablóny a podľa toho kolektor vie ktorú ma použiť na interpretáciu.

Length – Celková veľkosť FlowSetu v bajtoch

Record X - Field N value – zvyšok FlowSetu tvoria jednotlivé dátové záznamy, každý obsahuje množinu hodnôt. Dĺžka a typ jednotlivých polí bola definovaná v šablóna vďaka čomu môže kolektor záznamy od seba oddeliť.

Padding – výplň kvôli zarovnaniu

2.1.2.4 Options Template FlowSet a Options Data FlowSet

Záznamy tohto typu dodávajú informácie o konfigurácií NetFlow procesu alebo špecifické dáta, nie však informácie o konkrétnych IP tokoch. Ich princíp a formát je podobný ako u obyčajných dát.

2.2 NFDUMP

Nfdump je skupina sieťových nástrojov na zber a spracovávanie NetFlow dát. Vytvorená bola Petrom Haagom a je voľne šírená pod BSD licenciou. Jedná sa o konzolové aplikácie napísané v jazyku C ovládane príkazmi a parametrami programu. Bližšie informácie o nástroji možno nájsť na domovskej stránke projektu (5).

Súčasťou projektu je i grafická nadstavba NfSen, ktorá umožňuje toky zozbierané a spracované nfdumpom zobrazovať v grafoch a obrázkoch a to cez užívateľský prívetivé webové rozhranie.

Nfdump je napísaný na nízkej úrovni a vďaka rôznym optimalizáciám pracuje veľmi rýchlo a spoľahlivo. Súčasná verzia projektu je 1.6.13, ktorá transparentne podporuje NetFlow v5, v7 a v9. Zdrojové súbory si možno voľne stiahnuť na tejto adrese (6).

NFDUMP projekt obsahuje tieto nástroje:

- **nfcapd – NetFlow capture Daemon:**

  Zbiera NetFlow dáta zo siete a ukladá ich na disk v binárnom formáte. Automaticky vymieňa súbory každých *n* minút (obvykle 5 min).

- **nfdump – NetFlow dump:**

  Číta dáta zo súborov vytvorených nástrojom nfcapd a zobrazuje ich v prehľadnom, ľudsky čitateľnom formáte. Taktiež umožňuje vytvárať rôzne štatistiky, filtrovanie, zlučovanie atď.

- **nfprofile – NetFlow profiler:**

  Číta dáta zo súboru vytvoreného nfcapdom a filtruje ich podľa špecifických filtrov. Prefiltrovane dáta uloží na neskoršie použitie.

- **nfreplay – NetFlow replay:**

  Preposiela NetFlow dáta zozbierane nfcapdom cez sieť na ďalšie kolektory.

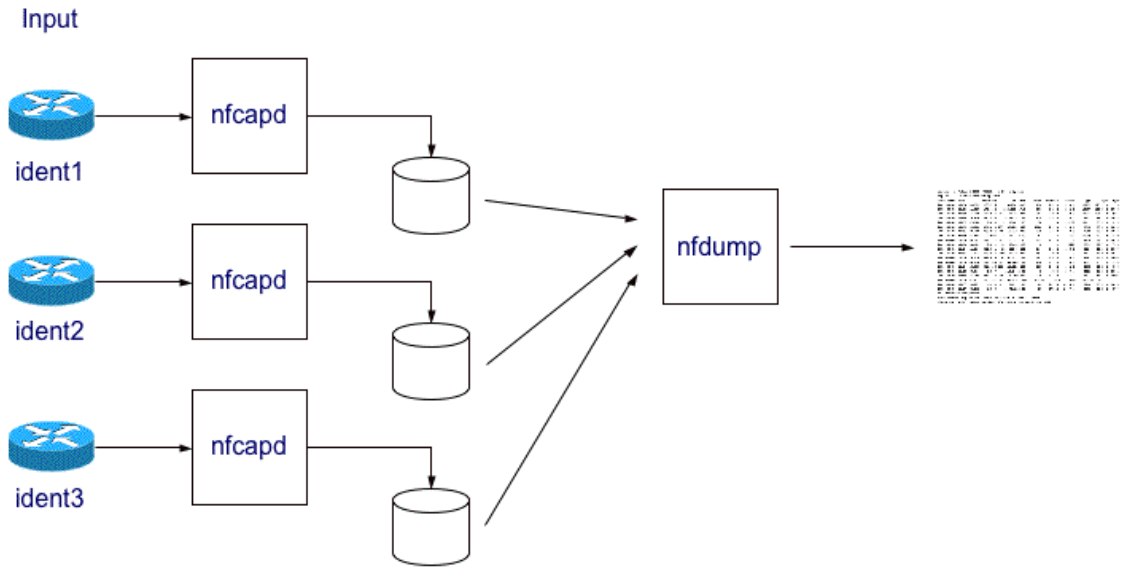
- **nfclean.pl – clean up old data:**

  Jednoduchý skript ktorý vymazáva staré dáta.

- **ft2nfdump:**

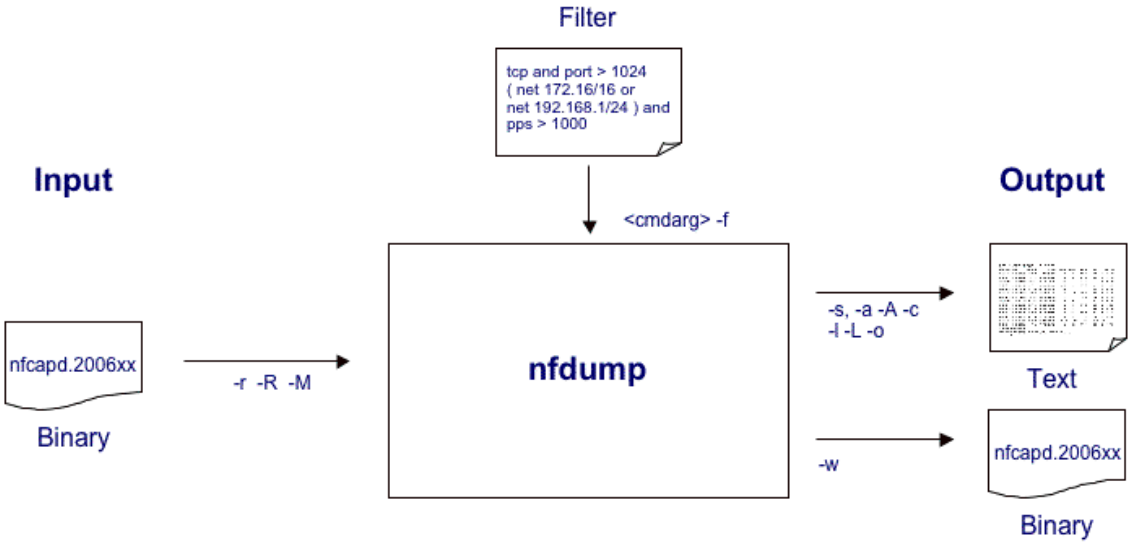
  Konvertuje dáta z iných zdrojov na formát kompatibilný s nfdumpom.

2.2.1 Princíp funkčnosti



Obrázok 2.5 Princíp činnosti nfdumpu (5)

Ako zobrazuje Obrázok 2.5, Nfcapd zbiera dáta zo sieťových zariadení a ukladá na disk. Všetky dáta sa najprv uložia na disk, až potom sa analyzujú, čo separuje proces ukladania a analýzy dát. Nfcapd ukladá súbory s názvom v tvare nfcapd.YYYYMMddhhmm, takže názov obsahuje časovú známku snímania, pričom každých obvykle 5 min sa vytvorí nový súbor. História dát sa ukladá po veľmi dlhú dobu, záleží to len na kapacite disku.



Obrázok 2.6 Spracovávanie dát nfdumpom (5)

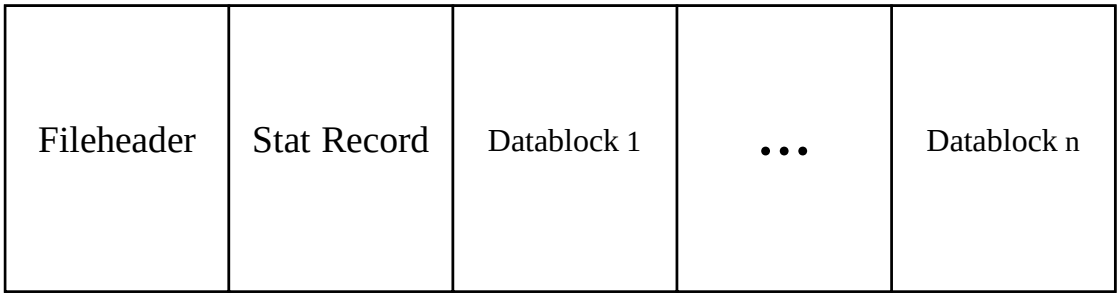
Nfdump spracováva binárne dáta vytvorené nástrojom Nfcapd alebo nfdump pri predošlom spracovaní a filtruje dáta podľa rôznych parametrov s ktorými môže byť spustený a výstup ukladá a zobrazuje v textovom alebo opäť binárnom formáte. Princíp je znázornený na Obrázok 2.6.

Nfdump ma veľmi mocný a rýchly filter, ktorý umožňuje prefiltrovať toky podľa požadovaných kritérií. Filter ma jednoduchú syntax a môže byť zadaný v parametri príkazového

riadku alebo v špeciálnom textovom súbore. Okrem filtrovania dokáže nfdump toky agregovať a vytvárať rôzne štatistiky sieťových tokov.

2.2.2 Detailný popis nfdump súboru

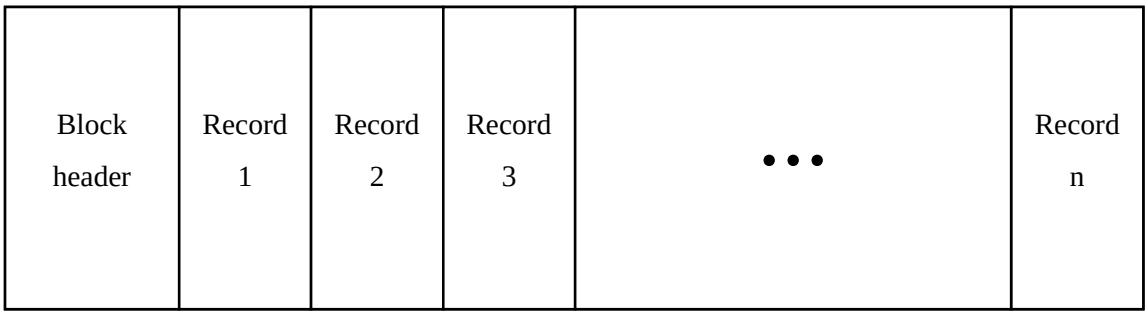
Nfcap ukladá zozbierane dáta do binárnych súborov s časovou známkou. Každý súbor je rozdelený do blokov rôznych typov. Principiálne rozdelenie je znázornené na obrázku Obrázok 2.7. Každý súbor teda začína hlavičkovým blokom, ktorý vždy na začiatku obsahuje 16 bitové magické číslo 0xA50C, ktoré identifikuje súbor patriaci nfdumpu a zaistí že sa súbor prečíta v správnom endiane. Hlavička ešte obsahuje verziu rozloženia blokov, príznaky, počet blokov v súbore a reťazcový identifikátor.



Obrázok 2.7 Rozloženie blokov v nfdump súbore

Bezprostredne po hlavičke vždy nasleduje blok so štatistikami, ktorý obsahuje štatistické informácie o všetkých záznamoch v súbore. Patria medzi to informácie o počte tokov na rôznych transportných protokoloch, veľkosti v bajtoch či počte paketov a rôzne časové údaje.

Zvyšok súboru tvoria dátové bloky, ktoré môžu byť rôznych typov a obsahujú samotné dáta a metadáta. Každý dátový blok začína spoločnou hlavičkou, ktorá špecifikuje veľkosť, typ, a počet záznamov v bloku. Od verzie 1.6.x používa nfdump nový typ dátových blokov, známych ako blok typu 2 (Block type 2), ktorý je však plne kompatibilný s predošlými verziami. Pôvodne bloky sa preto prevádzajú na novší typ. Tento blok sa skladá zo záznamov, ktoré sú rôzneho typu, spoločnú majú opäť len 32 bitovú hlavičku, ktorá obsahuje typ záznamu a jeho veľkosť vrátane tejto hlavičky. To umožňuje flexibilnejšie ukladanie dát a umožňuje tretím stranám pridávať si vlastné typy záznamov s rôznym dátami. Jednotlivé záznamy sú ukladané bezprostredne za sebou. Rozloženie dátového bloku je znázornené na obrázku Obrázok 2.8.



Obrázok 2.8 Rozloženie dátového bloku

Základným a najvyužívanejším typom je bežný typ záznamu (Common Record Type), ktorý obsahuje samotné záznamy o dátových tokoch zozbierané kolektormi. Záznam je zložený z tzv. rozšírení (extensions). Každý záznam tohto typu obsahuje minimálne spoločný základ, tzv. rozšírenie

0 (Extension 0) a 3 ďalšie rozšírenia, ktoré sú povinné. Nulté rozšírenie obsahuje vždy rovnaké a rovnako veľké položky, ďalšie rozšírenia sa už môžu líšiť podľa hodnôt bitov v poli príznakov, ktoré sa nachádza v rozšírení 0. Je to hlavne kvôli podpore jak IPv4 tak IPv6 adries.

- Rozšírenie 0 je spoločné pre všetky záznamy tohto typu a obsahuje všeobecné údaje o zázname ako napríklad transportný protokol, porty, stav a podobne. Zaujímavejšími položkami v tomto rozšírení je pole príznakov, ktoré určuje veľkosť alebo typ nasledujúcich rozšírení a odkaz na mapu rozšírení, ktorá podrobne popisuje použite rozšírenia.
- Rozšírenie 1 obsahuje zdrojovú a cieľovú IP adresu. Je to buď 64 bitová alebo 256 bitová štruktúra obsahujúca 2 IP adresy. Verzia IP protokolu a teda veľkosť položky sa odlišuje podľa nultého bitu v poli príznakov.
- Rozšírenie 2 je počítadlo paketov. Je to 32 alebo 64 bitové celé číslo, ktoré vyjadruje počet prenesených paketov v rámci tohto toku. Veľkosť položky je určená prvým bitom v poli príznakov.
- Rozšírenie 3 je počítadlo bajtov. Je to znovu 32 alebo 64 bitové celé číslo, ktoré vyjadruje počet prenesených bajtov v toku. Veľkosť určená druhým bitom v poli príznakov.

Medzi voliteľné rozšírenia patrí napríklad vstupné/výstupne rozhranie, číslo autonómneho systému, IP ďalšieho skoku, číslo VLAN, rôzne počítadla a podobne.

Aby bolo presne určené aké rozšírenia sú v zázname použité, každý záznam obsahuje odkaz na mapu rozšírení, ktorá to definuje(obdoba šablóny u NetFlow). Mapa rozšírení je samostatný typ záznamu. Je to flexibilnejšie a efektívnejšie riešenie ako pôvodne riešenie cez individuálne príznaky. V súčasnej verzii môže by až 65536 rôznych máp rozšírení. Zloženie mapy je jednoduché, po spoločnej hlavičke, ktorú majú všetky záznamy, nasleduje map ID – jednoznačný identifikátor mapy, na ktorý sa naviaže bežný záznam, a veľkosť mapy. Za tým nasledujú 16 bitové čísla, ktoré reprezentujú identifikátory rozšírení v zázname. Poradie týchto identifikátorov teda definuje poradie rozšírení v zázname bežného typu.

2.2.3 Vytvorenie nfdump súboru

Obvykle binárne súbory so záznamami o tokoch vytvára nástroj nfcapd zachytávaním dát zo siete a ukladaním vo vyššie uvedenom formáte. Ako je znázornene na obrázku Obrázok 2.6, nfdump ma možnosť vziať tento binárny súbor a prepísať ho do nového binárneho súboru, pričom môže uplatniť rôzne filtrovacie kritéria a zapísať len filtrované záznamy. nfdump teda v tomto prípade výsledky nevypíše v textovom formáte na štandardný výstup alebo do súboru, ale vytvorí nový binárny súbor s rovnakým formátom ako mal pôvodný. Princíp vytvárania takého súboru pritom nie je zložitý.

Na začiatku si nfdump vytvorí nový prázdny súbor a alokuje potrebne štruktúry a buffre do ktorých sa budú ukladať dáta pripravené na zápis. Pre každý súbor je vytvorená štruktúra, ktorá obsahuje ukazatele na tieto štruktúry a buffer. Následne zapíše do súboru prázdnu hlavičku a prázdny blok so štatistikami, ktoré majú nulové hodnoty a presne danú veľkosť. Jediná nenulová hodnota je magické číslo, ktoré sa nachádza na začiatku každého súboru. Tým sa rezervuje na začiatku súboru miesto pre tieto dva dôležite bloky, ktoré sa aktualizujú až po spracovaní celého vstupného súboru. Zatiaľ sa držia alokované v pamäti, kde sa postupne dopĺňajú a aktualizujú.

Pre dátový blok je alokovaný výstupný buffer, do ktorého sa za seba skladajú jednotlivé záznamy. Pri prechádzaní vstupného súboru sa každý záznam prefiltruje, spracuje a ak vyhovuje tak sa pridá do tohto buffru pre zápis. Na začiatok záznamu ukazuje pomocný ukazateľ. Ten sa vždy po prechode na ďalší záznam posunie práve o veľkosť toho záznamu.

Buffer ma obmedzenú veľkosť a vždy pred vložením ďalšieho záznamu sa skontroluje zaplnenie buffru. Veľkosť buffru pre dátový blok je 1 MB, je to kompromis medzi veľkosťou a časom potrebným pre zápis do súboru. Skutočná veľkosť alokovaného buffru je však 5 násobne väčšia, pre dáta iné ako záznamy tokov napríklad histogramy, ktoré môžu byť väčšie a majú potenciálne viac času na zápis do súboru.

Až sa buffer naplní, dátový blok vrátane hlavičky sa zapíše do súboru. Pred samotným zápisom sa môžu dáta ešte komprimovať. Po zápise bloku sa nastaví ukazateľ v buffri na začiatok čim sa buffer vyprázdni, hlavička bloku sa vynuluje a záznamy sa zapisujú do nového bloku.

Po tom čo sa spracuje celý vstupný súbor, resp. všetky vstupne súbory, posledný blok sa zapíše do výstupného súboru a aktualizujú sa štatistiky. Na konci sa vráti na začiatok výstupného súboru a prepíše v ňom zapísanú hlavičku a štatistiky novými, aktuálnymi hodnotami. Tým sa uzavrie celý súbor, ktorý sa následne môže opäť prečítať nfdumpom.

2.2.4 Spracovanie nfdump súboru

Nfdump spracuje vstupný binárny súbor a výsledkom môže byť opäť binárny súbor alebo výstup v textovom formáte, zobrazujúci jednotlivé záznamy o tokoch. Cez parametre môže byť požadovaný výstup rôzne ovplyvňovaný, môže byť filtrovaný, agregovaný, alebo sa môžu zobraziť len štatistiky o záznamoch. Záznamy tiež možno zobraziť v rôznych formátoch.

Principiálne rozloženie nfdump súboru je popísané v kapitole 2.2.2. Z tohto rozloženia vychádza prechod súborom a spracovanie jednotlivých častí tak ako idu za sebou

Na začiatku pred otvorením súboru si program alokuje potrebné dátové štruktúry a buffer do ktorých bude ukladať načítane dáta. Principiálne sa každý súbor skladá z troch základných časti: hlavička, štatistiky a dátové bloky. Pre každú časť je teda pripravená osobitná štruktúra, ktorá tieto informácie uchová a je možné k ním pri spracovaní súboru pristupovať podľa potreby. Ukazatele na všetky tieto dôležite štruktúry a iné informácie ako napr. popisovač súboru sú uložené v pomocnej štruktúre, ktorá drží dáta pohromade a predáva sa ako parameter do funkcií, ktoré s tým pracujú.

Ako prvú si program načíta zo súboru hlavičku, ktorá ma byť pre všetky nfdump súbory rovnaká. Veľkosť hlavičky je pevne daná počtom a veľkosťou povinných položiek hlavičky. Správnosť súboru sa overí podľa prvých 16 bitov, tzv. magické číslo ktoré musí byť 0xA50C. V hlavičke je ešte obsiahnutá verzia rozloženia, podľa ktorej sa vie ako sú v súbore radené bloky a príznaky pre daný súbor, kde je obsiahnutá napr. informácia o tom či je súbor komprimovaný alebo či sú dáta anonymizované. Údaj z hlavičky o počte blokov sa použije neskôr pri prechádzaní blokov.

Bezprostredne po hlavičke sa načíta do pripravenej pamäte i blok obsahujúci štatistiky, ktorý ma opäť presne danú veľkosť. Tento blok je prístupný počas celého spracovávania a slúži len pre informatívne účely, pri výpisoch kde sú štatistiky požadované. Tieto údaje sa počas čítania neprepisujú.

Za štatistikami už nasledujú jednotlivé dátové bloky. Obsah jednotlivých blokov ani ich rozloženie nie je presne špecifikované, všetky však majú rovnakú, pevne danú hlavičku. Pri načítaní bloku sa preto najprv načíta zo súboru 12 bajtová hlavička. V nej je obsiahnutá informácia o veľkosti celého bloku. Podľa tohto údaju sa do pripraveného buffru následne načíta zvyšok dát prisluhujúcich bloku. V prípade že je blok komprimovaný dochádza ešte k dekomprimácií. Takto načítaný blok je teda uložený v buffri dokiaľ sa celý nespracuje a nenačíta ďalší.

Bežný dátový blok typu 2 sa skladá z hlavičky a jednotlivých záznamov, ktoré sú naskladané za sebou v buffri. Hoci záznamy môžu byť rôzneho typu, všetky majú opäť spoločnú hlavičku, ktorá ma 2 položky, veľkosť a typ. Podľa veľkosti sa určí koľko miesta zaberá jeden záznam. Na začiatok aktuálne spracovaného záznamu ukazuje pomocný ukazateľ. Podľa typu sa určí, ako sa záznam spracuje.

Pre bežný typ záznamu sa najprv vyberie mapa rozšírení. Následne sa záznam prefiltruje časovým filtrom. Ten z tokov vyberá len záznamy z určitého časového okna. Ak nie je žiaden zadaný berú sa všetky záznamy. Hneď za tým sa záznam prefiltruje cez užívateľov textový filter a vyber sa len záznamy, ktoré vyhovujú kritériám. Tie sa buď vypíšu v textovom formáte alebo sa pridajú do výstupného buffru. Na koniec sa ešte aktualizujú štatistiky pre daný výber. Ukazateľ sa po spracovaní záznamu posunie o veľkosť záznamu, čím sa dostane na začiatok ďalšieho záznamu. Takto sa spracujú všetky záznamy z bloku, počet záznamov je zapísaný v hlavičke bloku.

Pre iné typy záznamov ako je napr. mapa rozšírení je spracovanie triviálnejšie. Záznam sa skontroluje, vyberú sa z neho relevantné informácie, ktoré sa uložia do pomocných premenných a ak sa vytvára nový súbor tak sa záznam nezmenený pridá do výstupného buffru k ostatným záznamom

Po spracovaní všetkých záznamov v bloku sa do buffru načíta ďalší blok a ten sa opäť po jednotlivých záznamoch spracuje. Tak to pokračuje až kým sa nespracuj všetky bloky, následne sa potom načíta ďalší súbor alebo program skončí.

2.3 Bloomov filter

Bloomov filter je pravdepodobnostná dátová štruktúra pomenovaná podľa svojho objaviteľa Burtona Howarda Blooma z roku 1970, ktorá sa používa na overenie príslušnosti prvku v množine. Pravdepodobnostná v tomto prípade znamená, že pri overovaní môžu nastať chyby. Pri tejto chybe sa o prvku, ktorý do množiny nepatrí, môžeme dozvedieť, že tam patrí, nikdy však naopak. To znamená že Bloomov filter nám vie so 100% určitosťou povedať že prvok do množiny nepatrí. Pravdepodobnosť chyby rastie s rastúcim počtom prvkov v množine (7).

V praxi sa Bloomov filter využíva na ušetrenie časovo náročných operácii, tým že dopredu zistí či ma význam operáciu vykonať a tým sa značne urýchľuje celý systém. Napríklad databázový systém BigTable (8) od spoločnosti Google využíva Bloomov filter na redukovanie počtu vyhľadávaní na disku. Najprv si cez filter overí či sa daný riadok alebo stĺpec na disku nachádza a až tak k nemu pristúpi. To výrazne zvyšuje výkon systému tým že sa ušetria zbytočne diskové operácie (ak prvok v databáze neexistuje) a je stále zaručená 100% spoľahlivosť systému. Medzi ďalšie systémy, ktoré využívajú Bloomov filter patrí napríklad Apache Cassandra (9), Squid webový proxy server (10) a mnohé iné.

2.3.1 Popis algoritmu

Prázdny Bloomov filter je bitové pole dĺžky *m* bitov, v ktorom sú všetky bity nastavene na nulu. Pre použitie Bloomovho filtru je potrebné mať definovaných *k* hašovacích funkcií, pričom každá z nich priradí každému prvku z množiny jednu z *m* pozícií v poli.

Nad Bloomovým filtrom sú definované 2 základne operácie: Operácia vloženia prvku do množiny a operácia dotazu, ktorá nám s určitou pravdepodobnosťou vracia odpoveď na otázku či sa nachádza daný prvok v množine.

Pre vloženie prvku do množiny, vypočítame pre daný prvok hash každou z *k* hašovacích funkcií. Získame tým *k* rôznych hodnôt. Na týchto pozíciách v poli nastavíme hodnotu bitu na 1.

Pri dotaze na daný prvok postupujeme podobne ako pri vkladaní. Pre prvok opäť vypočítame hodnoty tými istými hašovacími funkciami a otestujeme hodnoty na výsledných pozíciách v poli. Ak je na niektorej pozícií hodnota 0, vieme s určitosťou povedať že sa prvok v množine nenachádza. Ak sú všetky hodnoty nastavené na 1 tak je pravdepodobné že prvok je v množine zastúpený.

Jednou z nevýhod Bloomovho filtru je, že nie je možne odstraňovať prvky z množiny. Keďže výsledky hašovacích funkcií pre rôzne prvky sa vo filtri prekrývajú, tak pri vymazaní prvku

a vynulovaní príslušných bitov v poli, by sa mohli nechtiac ovplyvniť i záznamy iných prvkov čo by viedlo k nesprávnym výsledkom dotazov nad filtrom čím by sa porušila jeho základná vlastnosť.

2.3.2 Časová a priestorová zložitosť

Bloomov filter o veľkosti *m* bitov s *k* hašovacími funkciami má konštantnú časovú zložitosť *O(k)* pre vkladanie i testovanie príslušnosti prvku v poli. Nezávisí to od veľkosti Bloomovho filtru ani od počtu prvkov v množine. To robí Bloomov filter veľmi rýchlou a efektívnou dátovou štruktúrou.

Priestorová zložitosť je náročnejšia na výpočet a závisí od požadovanej pravdepodobnosti chybovosti. Čím nižšiu chybovosť požadujeme tým väčšie pole pre Bloomov filter potrebujeme aby sa záznamy jednotlivých prvkov čo najmenej prekrývali, takže to závisí aj od počtu vložených prvkov.

2.3.3 Optimálna veľkosť filtru a počet hašovacích funkcií

Výhodnou vlastnosťou Bloomovho filtra je že si môžeme prispôsobiť pravdepodobnosť chybovosti. Pravdepodobnosť chyby je približne $\left(1 - e^{-kn/m}\right)^k$ (7), kde *k* je počet hašovacích funkcií, *m* je počet bitov v Bloomovom filtri, a *n* je počet prvkov v množine. Takže stačí približne odhadnúť počet vkladaných prvkoch do množiny a vyskúšať rôzne kombinácie *m* a *k* na dosiahnutie požadovaných vlastností.

To vedie k problému optimálneho počtu hašovacích funkcií. Čím viac funkcií sa použije, tým pomalší bude Bloomov filter a rýchlejšie sa naplní bitové pole. Príliš malo funkcii však spôsobí viac chybových odpovedí. Pre výpočet počtu hašovacích funkcií je preto základom približne odhadnúť počet vkladaných prvkov a podľa toho si zvoliť veľkosť Bloomovho filtru. Počet funkcií sa potom vypočíta podľa vzorca $k = \frac{m}{n} \ln 2$ (7). Nakoniec si spočítame pravdepodobnosť chyby a ak je nevyhovujúca zmeníme *m* a počítame znovu až pokiaľ nám nebude pravdepodobnosť vyhovovať. Na odhad veľkosti výsledného filtru podľa zadaných parametrov poslúži jednoduchý kalkulátor, ktorý možno nájsť na tejto adrese (11).

3 Návrh optimalizácie

V tejto kapitole bude podrobne rozpísaný návrh optimalizácie spomínaného programu modifikáciou zdrojových súborov programu. Cieľom bolo optimalizovať program tak, aby prehľadávanie obrovského množstva dát bolo čo najrýchlejšie. Návrh stručne popisuje akým spôsobom by mala byť optimalizácia realizovaná

3.1 Návrh riešenia

Spracovávanie súborov nfdumpom je vďaka optimalizovanému a mocnému filtru veľmi rýchle. I napriek tomu môže byť prehľadávanie veľkého objemu dát (rádovo desiatky až stovky GB), v niekoľkých súboroch, ktoré obsahujú státisíce až milióny záznamov, veľmi zdĺhavé pretože nfdump musí prejsť všetky záznamy a porovnať na zhodu s filtrom. Pomalé je to hlavne z dôvodu že si program musí načítať z disku všetky bloky a záznamy, ktoré sú v súbore obsiahnuté a až tak ich môže spracovať. Diskové operácie sú pri tom v porovnaní s procesorom alebo internou pamäťou veľmi pomalé.

Myšlienka optimalizácie je preskočiť bloky alebo súbory, v ktorých sa filtrovaný záznam nenachádza a vynechať tak zbytočné spracovávanie záznamov a načítavanie dát z disku. Aby to bolo možné dosiahnuť, je potrebné upraviť súbor s dátami tak, aby obsahoval informácie o záznamoch na začiatku a bolo teda možne čo najrýchlejšie rozhodnúť o prítomnosti záznamu.

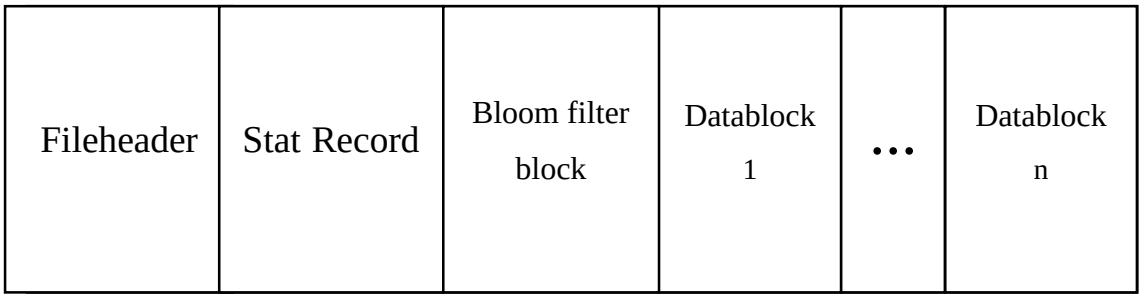
Na dosiahnutie týchto cieľov bola využitá dátová štruktúra Bloomov filter, ktorá je vďaka svojej rýchlosti a spoľahlivosti najvhodnejším kandidátom. V kapitole 2.2.3 boli vysvetlené princípy a použitie Bloomovho filtru, ktorý sa používa na overenie prítomnosti prvku v množine. Principiálne je to binárna štruktúra, v ktorej je poznačený výskyt záznamu v rámci nejakého rozsahu. Zavedením Bloomovho filtru sa teda prítomnosť záznamu pred samotným spracovaním overí voči filtru, ktorý bude umiestnený niekde na začiatku a v prípade nezhody sa celá časť preskočí. Bloomov filter sa teda vloží na začiatok každého súboru a do neho sa poznačí každý záznam, ktorý sa v súbore nachádza. Rozloženie takéhoto súboru je znázornené na obrázku Obrázok 3.1 Z toho vyplýva že vytváranie takéhoto súboru môže byť o niečo pomalšie než u bežných súborov. Podstatne sa však môže zrýchliť čítanie a spracovávanie upravených nfdump súborov, pri spracovaní súboru sa totiž pred samotným spracovávaním dát skontroluje filter a podľa výsledku je možne vynechať zvyšok spracovávania. Tým sa môžu preskočiť celé súbory, v ktorých sa nenachádzajú filtrované záznamy čím sa zrýchli výkon systému. Z vlastností Bloomovho filtra nám vyplýva že test prítomnosti sa môže zmýliť, čo povedie k tomu, že sa pristúpi k spracúvaniu súboru i keď sa tam záznam nenachádza a až pri spracovaní sa odfiltruje, k tomu však dochádza v minimálnom počte prípadov, takže prípadný relatívne malý čas na nadbytočne spracovanie je zanedbateľný. Vďaka pozitívnym vlastnostiam sa však nemôže stať aby sa test zmýlil tak, že preskočí súbor, v ktorom by sa záznam nachádzal. To nám zaistí 100% spoľahlivosť systému.

I samotný súbor obsahuje veľké počty záznamov, ktoré sú zaradené do jednotlivých dátových blokov, pričom nie v každom musí byť záznam ktorý požadujeme. Tento fakt sa dá využiť k ďalšej optimalizácií. Na začiatok každého bloku dát sa vložil menší Bloomov filter, ktorý obsahuje informáciu o prítomnosti požadovaného záznamu v rámci tohto bloku. Pred spracovaním dát je teda možne opäť skontrolovať prítomnosť voči tomuto filtru a prípadne sa môže celý blok preskočiť a opäť ušetriť výpočtový čas. Ako bude vyzerať takýto blok možno vidieť na obrázku Obrázok 3.2.

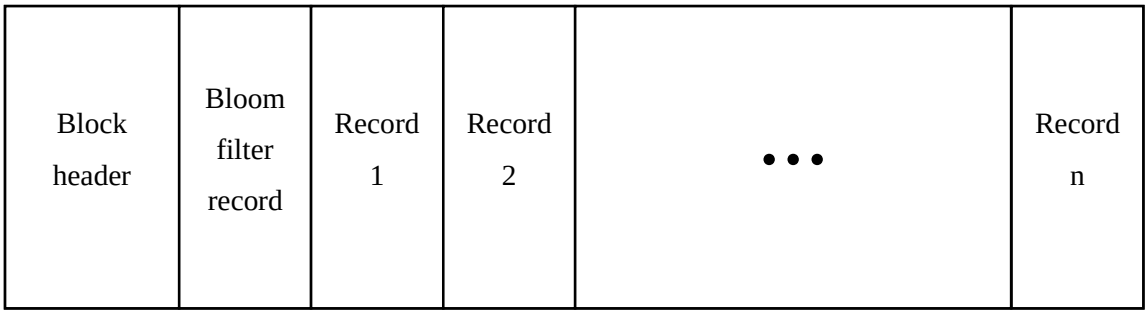
Pre filtrovanie záznamov sa môžu v nfdumpe použiť rôzne kritéria ako verzia sieťového protokolu, použitý transportný protokol, čísla portov a ďalšie. Medzi najpoužívanejšie a najčastejšie však patri filtrovanie na základe zdrojovej a cieľovej IP adresy toku. Práve pre tieto kritéria bolo teda

najvhodnejšie prispôsobiť optimalizáciu. Každý Bloomov filter sa preto skladá z dvoch častí, jedná pre zdrojovú a druhá pre cieľovú IP adresu.

Pri vytváraní upraveného súboru s vloženým Bloomovým filtrom sa spracuje celý súbor, postupne sa prejdú všetky záznamy a pre každý sa poznačí výskyt adresy do Bloomoveho filtru v bloku a hneď i v rámci súboru. Pre každý záznam sa teda vytvoria 4 značenia. Pri čítaní upraveného súboru sa najprv vstupný filter porovná s Bloomovým filtrom na začiatku súboru a následne v jednotlivých blokoch. Preskočia sa tie, v ktorých sa nenájde zhoda.



Obrázok 3.1 Rozloženie v upravenom súbore



Obrázok 3.2 Rozloženie bloku s Bloomovým filtrom

4 Implementácia

Táto kapitola popisuje implementáciu optimalizácie vložením Bloomoveho filtra na začiatok súboru a každého bloku, tak ako to bolo popísane v predošlej kapitole. Kapitola zvlášť rozoberá samotnú implementáciu Bloomovho filtru pre použitie na optimalizáciu. Následne je rozobraté prepísanie pôvodného nfdump súboru do nového s vložením Bloomovým filtrom, ktorý ma zrýchliť následne spracovávanie upraveného súboru. Na konci kapitoly je popísané akým spôsobom upravený nfdump spracuje upravený súbor, a ako si s ním poradí pôvodný nfdump.

4.1 Bloomov filter

Bloomov filter je bitové pole určitej veľkosti, do ktorého sa *k* hašovacími funkciami značí prítomnosť prvku v množine. Preto je Bloomov filter reprezentovaný štruktúrou, ktorá okrem samotného poľa ukladá i informáciu o počte bitov, veľkosti celého filtru v bajtoch a počet hašovacích funkcií, aby bolo možne rovnakým počtom záznam i overiť.

Pri vytváraní filtru je dôležite vedieť veľkosť a počet hašovacích funkcií. Výpočet týchto parametrov bol rozobratý v kapitole 2.3.3. Z toho vyplýva že sa dajú parametre vypočítať zo znalosti približného počtu záznamov v množine. Problém bol že počet záznamov sa nedá odhadnúť bez toho aby sme podrobne prešli celý súbor a záznamy spočítali. Navyše nás zajímajú len unikátne adresy, ktorých môže byť úplný iný počet než počet záznamov. Preto bolo najvhodnejším riešením stanoviť pevnú veľkosť filtra s tým, že si ju užívateľ môže cez parameter upraviť a pokúsiť sa sám počet adries odhadnúť.

Ako kľúč do Bloomovho filtra slúži 128 bitová IP adresa. Je to kvôli podpore IPv6, ktorá je 128 bitová. Pretože v tejto implementácií ma kľúč presne danú veľkosť, tak i IPv4 adresy, ktoré sú iba 32 bitové musia byť hašované v tejto veľkosti. Preto sa IPV4 adresa vloží na spodne bajty kľúča a zvyšok sa vyplní nulami a tým sa roztiahne. Pre tento kľúč sa vypočíta niekoľkou hašov a pre každý sa nájde v poli odpovedajúci bit. Tento bit sa nastaví na 1 pri vložení nového záznamu alebo sa zistí jeho hodnota pri testovaní príslušnosti.

Obvykle sa na hašovanie záznamov používa niekoľko rôznych nezávislých hašovacích funkcií. Pre veľký počet záznamov, kde je potrebný veľký počet funkcií by to mohlo byť komplikovane. Alternatívne sa môže miesto *k* rôznych hašovacích funkcií použiť jedna s rôznymi počiatočnými hodnotami, tzv. „salt“. Výsledok je adekvátny, počet je ľahko rozšíriteľný pridaním nejakých počiatočných hodnôt a nie je potrebne vymýšľať rôzne hašovacie funkcie.

4.2 Modifikácia nfdump súboru vložením Bloomovho filtru

Kapitola 2.2.3 podrobne rozpísala ako prebieha vytváranie bežného binárneho nfdump súboru. Aby však bolo možné uplatniť optimalizáciu rozoberanú v tejto práci, je potrebné do novo vytvoreného súboru vložiť na správne miesta Bloomové filtre, ktoré poslúžia k urýchleniu spracovávania.

Bloomov filter sa vloží do binárneho súboru pri prepisovaní originálneho súboru upraveným nfdumpom pri použití parametru -P. Ako bolo spomenuté v návrhu v kapitole 3.1, Bloomové filtre

budú 2: Globálny Bloomov filter pre celý súbor a lokálny pre jednotlivé bloky. Začiatok vytváranie súboru je rovnaký ako pri bežnom súbore, alokujú sa štruktúry, pripraví buffer a na začiatok súboru sa vloží prázdna hlavička a blok so štatistikami. V upravenej verzii ma nasledovať globálny Bloomov filter hneď za štatistikami aby pri spracovaní sa mohol hneď na začiatku porovnať na zhodu.

Aby sa zachovala spätná kompatibilita s pôvodnou verziou a bolo možné upravené súbory čítať aj originálnou verziou nfdumpu, globálny Bloomov filter je maskovaný ako obyčajný dátový blok. Má teda rovnakú hlavičku ako všetky dátové bloky, ktorá obsahuje okrem iného aj veľkosť tohto bloku a teda Bloomovho filtru. V tele však neobsahuje záznamy o toku ale bitové pole, ktoré reprezentuje Bloomov filter. Keďže tento blok musí byť na začiatku a bude sa počas spracovania súboru aktualizovať, bol využitý rovnaký mechanizmus ako pre hlavičku a teda prvý dátový blok so zatiaľ prázdnym Bloomovým filtrom sa zapíše do súboru hneď za blok so štatistikami. Jeho kopia ostáva v pamäti a zapíše sa do súboru na konci.

Za týmto blokom nasleduje už bežný dátový blok so záznamami. Blok začína hlavičkou a jeho telo je udržiavané vo výstupnom buffri. Na začiatku každého bloku ma byť lokálny Bloomov filter pre tento blok. Tento lokálny filter je maskovaný ako bežný záznam, s bežnou hlavičkou, ktorá obsahuje id a veľkosť. Prázdny lokálny Bloomov filter sa teda vloží ako prvý záznam do bloku. Keďže záznam zostáva v buffri až do zápisu bloku do súboru, ukazateľ na tento Bloomov filter sa uchová a vďaka tomu je možné tento záznam upravovať.

Pri spracovávaní vstupného súboru sa záznamy spracovávajú, filtrujú a postupne ukladajú do tohto výstupného buffru za seba. Pre každý vložený záznam do buffru sa do lokálneho i globálneho Bloomovho filtru poznačí prítomnosť záznamu. Po naplnení buffru sa blok zapíše do súboru, buffer sa vyprázdni a opäť sa vloží ako prvý záznam prázdny Bloomov filter.

Na konci až sa zapíše posledný blok, vráti sa na začiatok súboru, zapíše sa aktualizovaná hlavička, blok so štatistikami a hneď za tým sa aktualizuje globálny Bloomov filter, ktorý tam ma rezervované miesto. Tým je nový binárny súbor obsahujúci Bloomové filtre pripravený a dokončený.

4.3 Spracovanie nfdump súboru s Bloomovým filtrom

V kapitole 2.2.4 bolo rozpísané ako prebieha prečítanie nfdump súboru nfdumpom. V kapitole 4.2 bolo zase spomenuté ako sa Bloomov filter vytvorí a vloží do upraveného súboru. Z toho nám teda vyplýva že Bloomov filter je obyčajný dátový blok, umiestnený na začiatku súboru za blokom so štatistikami. Jeho načítanie teda prebieha ako načítanie bežného dátového bloku do pomocného buffru. Pri spracovaní tohto bloku pôvodný nfdump rozpozná blok ako neznámy blok typu 100 a keďže nevie ako s ním naložiť, preskočí ho a ďalšie bloky spracováva bežným spôsobom. Tým je zachovaná spätná kompatibilita s pôvodnou verziou a teda i upravené súbory je možne prečítať i originálnym nfdumpom. Ak však na tento blok na začiatku narazí modifikovaná verzia nfdumpu nepreskočí ho, ale miesto toho vykoná test. Vstupom pre tento test je vstupný textový filter zadaný užívateľom, ktorý obsahuje IP adresy a iné kritéria, podľa ktorých chceme záznamy filtrovať. Tento filter je spracovaný parserom, ktorý textový filter rozoberie a uloží do dátovej štruktúry červeno-čierny strom. Uzly v tomto strome sú jednotlivé kritéria. Pôvodný nfdump tento strom využíva na filtrovanie záznamov, ktoré však filtruje až počas spracovávanie jednotlivých záznamov. Upravený nfdump teda tiež využíva tieto stromy, vytiahne si však z nich iba zdrojové a cieľové IP adresy, a ostatné uzly preskočí. Každú adresu použije ako kľúč do príslušného Bloomovho filtra a tým otestuje, či sa v súbore nachádzajú adresy, ktoré užívateľ požaduje. Ak je zhoda negatívna, program preskočí

následne spracovanie súboru a načíta rovno ďalší súbor. Pri pozitívnej zhode spracovávanie pokračuje ďalej. Keďže tento blok neobsahuje žiadne záznamy, celý sa preskočí a načíta sa ďalší blok.

Pri spracovávaní bežného bloku by mal v upravenom súbore byť prvý záznam lokálny Bloomov filter, ktorý obsahuje informácie o príslušnosti adresy v bloku. Pôvodný nfdump opäť tento záznam preskočí pretože nerozpozná jeho typ, a pokračuje ďalším záznamom. V prípade upraveného nfdumpu sa znovu vykonajú testy ale tentokrát nad týmto filtrom. Ak sa adresy v bloku nachádzajú pokračuje spracovanie ďalším záznamom, v opačnom prípade sa preskočí celý blok, načíta ďalší v ktorom bude na začiatku znovu lokálny Bloomov Filter. Zvyšok pokračuje tak ako v pôvodnej verzii, jedinou zmenou sú teda testy Bloomovho filtra, ktoré môžu prekročením ušetriť výpočtový čas.

5 Testovanie a výsledky

Táto kapitola sa zaoberá testovaním a zhodnotením výsledkov optimalizácie. Bola vykonaná séria rôznych testov, na rôznych veľkostiach a počte vstupných súborov. Každý test bol vykonaný najprv s pôvodným nástrojom a následne s upraveným, pričom hlavným cieľom bolo sledovanie rozdielov medzi týmito dvoma verziami.

Každá séria testov sa da rozdeliť na 2 časti. Najprv sa testovalo vytvorenie nového binárneho súboru nfdumpom zo vstupných dát. Vždy sa vytvoril jeden súbor bez Bloomovho filtru a jeden s ním. Zaujímavé pri tom bolo sledovať veľkosť nového súboru s filtrom oproti originálu aby bolo vidieť koľko bajtov pridá Bloomov filter. Druhá časť sa zameriavala na čítanie a spracovanie súborov, opäť najprv sa spracovali pôvodne súbory a následne sa vyhľadávalo v súboroch s Bloomovým filtrom, pričom sa sledoval čas spracovania, ktorý sa porovnal. Podľa neho sa zistila efektivita optimalizácie. Sledovali sa dve typy času. Jednak to bol celkový čas behu programu, ktorý ukazoval ako dlho program bežal. Druhý čas je čas procesu strávený na CPU, ktorý demonštroval činnosť programu bez diskových operácií. Čas spracovanie samozrejme nie je vždy rovnaký pretože kvôli rutinám operačného systému a procesoru, môže i spustenie rovnakého procesu trvať rôznu dobu. Preto sú minimálne odchýlky v tomto prípade zanedbateľné.

Testovanie prebiehalo na notebooku s Procesorom Intel® Core™ i5-2450M CPU @ 2.50GHz × 4 pod operačným systémom Ubuntu 12.04 LTS. Ako testovacie dáta poslúžili nfdump súbory zo serveru collector-nfsen.liberouter.org, ktoré boli zozbierane dňa 1.2.2014 z linky s priepustnosťou 10GB/s, ktorá prepája akademickú sieť CESNET s ďalšími sieťami. Adresy boli v záznamoch anonymizované. Spolu bolo k dispozícií 288 súborov s celkovou veľkosťou 35,9 GB. V Súboroch sa celkovo nachádzalo 626294336 tokov. Testovacie súbory boli uložené na externom pevnom disku Seagate, čo spôsobilo pomalšie spracovávanie.

5.1 Testovanie bez vstupného filtru

V prvej časti sa testovanie zameralo na spracovanie dát bez vstupného užívateľského filtru a teda všetkých dát v súboroch. Test sa zameriava na spracovanie veľkého množstva dát, pričom v tejto časti sa ešte neuplatní pripravená optimalizácia. Cieľom bolo sledovať nadbytočnú veľkosť spôsobenú vložením Bloomovho filtru do súboru a čas spracovania pridaný kvôli vkladaniu ďalších záznamov.

5.1.1 Vytváranie súboru

V tomto teste bol z pôvodných súborov vytvorený nový binárny súbor. Celá vzorka dát sa postupne najprv spracovala pôvodnou verziou nfdumpu od autora a následne upravenou verziou. Cieľom bolo sledovať zmeny veľkosti súboru a čas potrebný na pridanie Bloomovho filtru do pôvodných súborov. Výsledný súbor v tomto prípade obsahuje všetky záznamy zo vstupných súborov.

Počet súborov	Veľkosť súborov [MB]	Pôvodny nfdump			Upravený nfdump s BF		
		Veľkosť [MB]	čas		Veľkosť [MB]	čas	
			CPU [s]	Real [m:s]		CPU [s]	Real [m:s]
1	136	136,1	0,26	0:06	136,9	2,72	0:05
10	1255	1255	3,07	0:55	1230	22,25	1:07
50	4749	4749	14,02	5:14	4766	90,02	5:14
100	7635	7635	21,76	8:37	7662	145,58	8:48
288	34226	34226	96,69	39:58	34346	643,14	40:09

Tabuľka 5.1: Výsledky testov prvej série

5.1.2 Spracovanie súboru

Druhý test sa zameral na spracovanie súborov. V tejto fáze bolo cieľom zistiť rýchlosť spracovania dát v súbore bez Bloomovho filtru v porovnaní so súbormi s ním. Kedže sa v tejto časti filtrovanie nepoužíva a musia sa spracovať všetky záznamy, optimalizácia sa neuplatní, ba naopak bloky a záznamy, ktoré sú v tejto časti navyše spôsobia dlhšie spracovávanie. Sledovaný bol teda čas spracovania.

Počet súborov	Veľkosť súborov [MB]	Pôvodny nfdump		Upravený nfdump	
		CPU [s]	Real [m:s]	CPU [s]	Real [m:s]
1	136	11,36	0:16	11,23	0:16
10	1255	104,41	2:22	99,16	2:25
50	4749	397,68	9:18	399,92	9:49
100	7635	627,99	14:48	629,95	15:10
288	34226	2952,94	68:57	2950,32	69:35

Tabuľka 5.2: Výsledky testov druhej série

5.2 Testovanie s jednoduchým vstupným filtrom

V druhej sérii testov bol použitý jednoduchý vstupný filter obsahujúci jednu IP adresu. Ako vzorka bola vybratá jedna adresa z náhodného súboru. Výsledkom teda boli filtrované záznamy obsahujúce danú adresu. V tomto prípade sa už uplatní optimalizácia a cieľom bolo zistiť k akému zrýchleniu pri spracovaní dôjde. Vybraná bola zdrojová adresa 177.148.84.79, ktorá sa nachádza iba v 3549 tokoch. Je zrejme že čim menej sa adresa vyskytuje v záznamoch, tým bude optimalizácia efektívnejšia, pretože sa bude preskakovať viac blokov a súborov. Keďže záznamy s touto adresou sa nachádzajú viac na začiatku v prvých súboroch, zmeny budú výraznejšie pri prechádzaní viacerých súborov.

5.2.1 Vytváranie súboru s filtrovanými záznamami

Podobne ako v prvej sérii, aj tu začíname s vytváraním súboru, tento krát v ňom však nebudú všetky záznamy ale iba filtrované. Výsledný súbor bude teda obsahovať iba záznamy s adresou odpovedajúcou vstupnému filtru. Čas vytvorenia bol v tomto prípade irelevantný.

Počet súborov	Veľkosť súborov [MB]	Výsledna veľkosť [MB]	Výsledna veľkosť s BF [MB]
1	136	0,15	0,36
10	1255	0,031	0,378
50	4749	0,152	0,503
100	7635	0,152	0,503
288	34226	0,191	0,542

Tabuľka 5.3 Vysledky testov s jednoduchým filtrom

5.2.2 Spracovanie súboru s filtrovaním

V tomto teste sa konečne plne uplatní optimalizácia, ktorú bola snaha dosiahnuť. Zatiaľ čo spracovanie pôvodných súborov prebehne pôvodným spôsobom a prehľadajú sa všetky dáta, pri spracovaní modifikovaných súborov sa testuje Bloomov filter a preskakujú sa súbory, v ktorých sa adresa nenachádza, čo by malo značne urýchliť spracovanie. Cieľom teda bolo sledovať aké veľké zrýchlenie oproti originálu optimalizácia priniesla.

Počet súborov	Veľkosť súborov [MB]	Pôvodny nfdump		Upravený nfdump	
		CPU [s]	Real [m:s]	CPU [s]	Real [m:s]
1	136	0,07	0:01	0,10	0:02
10	1255	1,06	0:26	0,9	0:24
50	4749	4,61	2:09	3,01	1:25
100	7635	5,94	3:18	1,47	1:17
288	34226	29,31	17:17	2,83	2:36

Tabuľka 5.4 Výsledky testov filtrovania

5.3 Testovanie s neprítomnou IP adresou

V tretej sérií testov sa ako do vstupného filtra použila adresa, ktorá sa nenachádza v žiadnom zo zaznamenaných tokov. V tomto prípade nebude výsledok obsahovať žiadne záznamy a zatiaľ čo

originálny nfdump i napriek tomu načíta a spracuje všetky záznamy, upravená verzia po teste v Bloomovom filtri preskočí celé spracovanie i načítavanie z disku a tým výrazne urýchli celý proces.

5.3.1 Vytváranie súboru bez záznamov

Vytvorenie súboru v tomto prípade viedlo k vytvoreniu súboru, v ktorom sa nenachádzajú žiadne záznamy, len bloky obsahujúce hlavičku, štatistiky, a záznamy iných typov. V upravenej verzii je navyše Bloomov filter. Cieľom bolo sledovať nadbytočnú veľkosť spôsobenú filtrom.

Počet súborov	Veľkosť súborov [MB]	Výsledna veľkosť bez BF [MB]	Výsledna veľkosť s BF [MB]
1	136	0,001	0,360
10	1255	0,001	0,360
50	4749	0,001	0,360
100	7635	0,001	0,360
288	34226	0,001	0,360

Tabuľka 5.5 Výsledky testov bez IP záznamov

5.3.2 Spracovanie súboru s prázdnym výstupom

V tomto teste sa prechádzajú súbory ale neprodukujú žiadne výstupy pretože nebola nájdená žiadna zhoda. V tomto prípade sa teda dáta prechádzajú a nahrávaj z disku zbytočne, ale optimalizácia sa uplatni v plnom rozsahu, pretože sa budú preskakovať všetky súbory a z disku sa nahrajú len prvé tri bloky pre každý súbor.

Počet súborov	Veľkosť súborov [MB]	Pôvodny nfdump		Upravený nfdump	
		CPU [s]	Real [m:s]	CPU [s]	Real [m:s]
1	136	0,11	0:01	0,01	0:00
10	1255	1,20	0:25	0,01	0:00
50	4749	3,85	2:07	0,01	0:01
100	7635	6,65	3:33	0,01	0:01
288	34226	29,38	16:42	0,01	0:04

Tabuľka 5.6 Výsledky testov spracovania bez IP záznamov

5.4 Testovanie na komprimovaných dátach

V tejto kapitole bolo testovanie vyskúšané na komprimovaných dátach. Všetky vstupne súbory boli najprv nfdumpom skomprimované a následne boli vykonané testy podobné sérii jednoduchým filtrom na rovnakú IP adresu. Cieľom bolo zistiť aký vplyv na rýchlosť a veľkosť ma kompresia dát.

5.4.1 Vytváranie komprimovaného súboru

Vstupom sú komprimované dáta a výstupom opäť komprimovaný súbor. Test mal za úlohy zistiť aký rozdiel vo veľkosti zohrá Bloomov filter pri komprimovaných dátach a aké spomalenie spôsobí dekomprimovanie záznamov pred spracovaním.

Počet súborov	Veľkosť súborov [MB]	Výsledna veľkosť [MB]	Výsledna veľkosť s BF [MB]
1	65,7	0,007	0,351
10	598	0,012	0,355
50	2230	0,050	0,394
100	3569	0,050	0,394
288	16564	0,062	0,406

Tabuľka 5.7 Výsledky testov s komprimovanými dátami

5.4.2 Spracovanie komprimovaného súboru

V tomto teste je prehľadávanie komprimovaných záznamov. Keďže test príslušnosti v Bloomovom filtri prebieha na globálnej úrovni pred dekomprimáciou, zrýchlenie môže byť ešte výraznejšie.

Počet súborov	Veľkosť súborov [MB]	Pôvodny nfdump		Upravený nfdump	
		CPU [s]	Real [m:s]	CPU [s]	Real [m:s]
1	65,7	0,54	0:01	0,78	0:01
10	598	4,13	0:10	4,90	0:13
50	2230	16,24	0:45	14,32	0:44
100	3569	22,06	1:27	10,14	0:41
288	16564	118,80	7:58	23,76	1:35

Tabuľka 5.8 Výsledky testov s komprimovanými dátami

5.5 Výsledky testov

Z výsledkov testov nám vyplynulo, že upravené bloky s Bloomovým filtrom majú len nepatrne väčšiu veľkosť, takže samotný Bloomov filter zaberá zanedbateľne miesto. Rozdiel je najväčší pri súboroch s minimálnym počtom záznamov alebo bez nich, ale to sú ojedinele prípady, ktoré sa v bežnej praxi veľmi nevyskytujú.

Z časového hľadiska testy preukázali že celkový čas na vytvorenie modifikovaného súboru je porovnateľný s časom vytvorenia pôvodného súboru, rozdielom bol len čas strávený na procesore spracovávaním. Keďže pri vytváraní súboru s Bloomovým filtrom sa musel každý záznam hašovať niekoľkými funkciami, bolo spracovanie takéhoto záznamu o niečo náročnejšie. Na druhú stranu však testy preukázali, že vyhľadávanie záznamov s konkrétnou IP adresou je v upravených súboroch s Bloomovým filtrom podstatne rýchlejšie.

Väčšinu celkového času spracovania v tomto prípade tvoria diskové operácie, ktoré zakrývajú čas strávený na CPU samotným spracovaním. Je to hlavne z dôvodu operácie nad externým diskom sú ešte pomalšie ako nad interným, preto to zaberá viac času. To ale v našom prípade znamená ešte väčšiu výhodu pre našu optimalizáciu. Preskočenie súboru, v ktorom sa nenachádzajú požadované záznamy totiž ušetrí veľký počet diskových operácií tým že sa nemusia načítavať nepotrebné bloky. I vďaka tomu bolo zrýchlenie tak výrazne. To platí hlavne pre globálny filter, lokálny diskovú operáciu neušetrí pretože v dobe kontroly je už celý blok načítaný v pamäti. Ušetrí však čas na CPU, tým že sa nemusia záznamy spracovávať.

Komprimácia dát nemala pri testoch veľký vplyv, zrýchlenie s prejavilo i v tomto prípade. Zaujímavé bolo že skomprimované súbory sa spracovali ešte rýchlejšie ako pôvodne.

Okrem týchto testov bolo vykonaných množstvo ďalších skúšobných testov, ktoré sa sem nevošli a všetky jasne preukázali že optimalizácia sa vyplatila a bolo dosiahnuté značne zrýchlenie.

6 Záver

Cieľom práce bolo optimalizovať prehľadávanie NetFlow záznamov nástrojom nfdump. Hlavnou požiadavkou bolo zrýchliť vyhľadanie záznamov s konkrétnou zdrojovou či cieľovou IP adresou.

V prvej teoretickej časti v kapitole 2 som podrobne rozobral a predstavil NetFlow protokol, nástroj nfdump a formát binárneho súboru, ktorý sa mal upraviť tak aby sa dosiahlo optimalizácie. Dôležite pri tom bolo pochopiť ako to celé funguje a ako je možne upraviť originálnu verziu tak, aby sa zachovala spätná kompatibilita a aby sa nijak výrazne nezmenilo správanie systému. Následne bola ešte popísaná dátová štruktúra, Bloomov filter, ktorý bol pre optimalizáciu základom. Bol popísaný jeho princíp, výhody, nevýhody a problémy spojené s jeho vytváraním a použitím.

V druhej časti práce som popísal v kapitole 3 navrhnuté riešenie optimalizácie a zhodnotil jeho výhody i nevýhody. V nasledujúcej kapitole 4 som rozpísal ako bolo navrhnuté riešenie implementované a ako to celé funguje pri práci s modifikovanými verziami. Nakoniec som na výslednom programe spravil sériu testov, zameranú na rôzne aspekty a dôsledky úprav. Výsledky boli zhodnotené a porovnané a z výsledkov vyplynulo že boli dosiahnuté požadované ciele a optimalizácia sa vyplatila. S minimálnou réžiou pri vytvorení upravených dát a so zanedbateľnou nadbytočnou veľkosťou v súboroch, ktorú zaberá vložený Bloomov filter bolo dosiahnuté výrazne zrýchlenie pri prehľadávaní veľkého množstva súborov s požiadavkou na len pár vybraných záznamov nachádzajúcich sa niekde v tom množstve dát.

Medzi možne rozšírenia do budúcna patrí implementácia tejto optimalizácie priamo do nástroju Nfcapd, tak aby sa zozbierane dáta z exportérov ukladali priamo v modifikovanej verzii a nebolo nutne im dodatočne túto úpravu dodávať. Ďalšou úpravou by mohlo byť pridanie optimalizovaného filtrovania i na základe iných kritérií, napríklad podľa zdrojových a cieľových portov, verzie protokolov alebo masky siete. K dosiahnutiu týchto cieľov by sa mohlo postupovať podobným spôsobom ako v tejto práci, len by sa pridali ďalšie Bloomové filtre pre tieto špecifické kritéria.

Literatura

(1) Wikipedie: Otevřená encyklopedie: NetFlow [online]. c2014 [cit. 2015-05-17]. Dostupný z URL: <http://cs.wikipedia.org/w/index.php?title=NetFlow&oldid=11145871>

(2) Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC3954, October 2004. [online]. [cit. 2015-05-17]. Dostupné na URL: https://www.ietf.org/rfc/rfc3954.txt

(3) Robertson, S.: NetFlow V9 Overview: What's in the NetFlow Packet Header? [online]. 2013-01-23 [cit. 2015-05-17] Dostupné na URL: https://www.plixer.com/blog/netflow/netflow-v9-overview-whats-in-the-netflow-packet-header/

(4) Cisco: NetFlow Export Datagram Formats [online] [cit. 2015-05-17]. Dostupné na URL: http://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/5-0/user/guide/user/format.html

(5) Haag, P.: Nfdump [online]. 2014-12-01 [cit. 2015-05-17]. Dostupné na URL: http://nfdump.sourceforge.net/

(6) Haag, P.: NFDUMP - NetFlow processing tools [online]. 2014-12-01 [cit. 2015-05-17]. Dostupné na URL: http://sourceforge.net/projects/nfdump/?source=typ_redirect

(7) Wikipedie: Otevřená encyklopedie: Bloomův filtr [online]. c2014 [cit. 2015-05-17]. Dostupný z URL: <http://cs.wikipedia.org/w/index.php?title=Bloom%C5%AFv_filtr&oldid=11770765>

(8) Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C; Wallach, Deborah A; Burrows, Michael 'Mike'; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E (2006), "Bigtable: A Distributed Storage System for Structured Data", *Research* (PDF), Google

(9) Hewitt, Eben (December 15, 2010). *Cassandra: The Definitive Guide* (1st ed.). O'Reilly Media. p. 300. ISBN 978-1-4493-9041-9

(10) Wessels, Duane (2004). *Squid: The Definitive Guide*. O'Reilly Media. ISBN 978-0-596-00162-9.

(11) Hurst, T.: Bloom filter calculator [online]. [cit. 2015-05-17]. Dostupné na URL: http://hur.st/bloomfilter

(12) Mill, B.: Bloom Filters by Example [online]. [cit. 2015-05-17]. Dostupné na URL: http://billmill.org/bloomfilter-tutorial/

Príloha A - Obsah CD

- Technická správa vo formáte .docx
- Technická správa vo formáte .pdf
- Zdrojové súbory upravenej verzie nfdumpu
- Vzorka testovacích dát