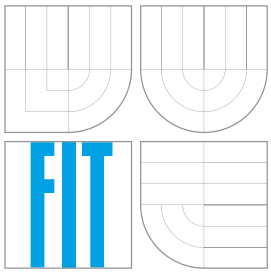
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ANALÝZA SÍŤOVÉHO PROVOZU POMOCÍ SKETCHÍ

NETWORK TRAFFIC ANALYSIS BASED ON SKETCHES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ALEŠ DŘEVO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV BARTOŠ

BRNO 2014

Abstrakt

Cílem této bakalářské práce je vytvořit program na analýzu síťového provozu a detekci anomálií v provozu počítačové sítě. K tomu je využívána technika zvaná Heavy-Changes Detection spadající do kategorie proudových algoritmů. Pro práci s daty jsou použity speciální struktury zvané sketche, které dokáží uchovávat velké množství dat s nízkou paměťovou náročností. K získání potřebných dat ze sítě jsou využívány programy běžící pod systémem Nemea, pro který je celý tento projekt vytvářen.

Abstract

Aim of this thesis is to create a program for network traffic analysis and for detection of anomallies in the traffic. The Heavy-Changes Detection technique which falls within the Data stream algorithm category is used to do so. Special structures called sketches are used for data processing. These structures are capable of maintaining large amounts of data with low memory consumption. Programs from Nemea system for which this project is created are used for gathering necessary network data.

Klíčová slova

anomálie, detekce, siťový provoz, Nemea, sketch, EWMA, analýza

Keywords

anomaly, detection, network traffic, Nemea, sketch, EWMA, analysis

Citace

Aleš Dřevo: Analýza síťového provozu pomocí sketchí, bakalářská práce, Brno, FIT VUT v Brně, 2014

Analýza síťového provozu pomocí sketchí

Prohlášení

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

.....
Aleš Dřevo
20. května 2014

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce Ing. Václavovi Bartošovi za jeho užitečné rady a čas věnovaný při konzultacích.

Obsah

Kapitola 1

Úvod

Počítačová síť a rozvoj internetu dnes zasahuje do širokého spektra všemožných oblastí, internet už není pouze součástí počítačů a notebooků jako takových. Připojení k internetu je v dnešní době běžnou součástí jak osobních počítačů, mobilních telefonů, tabletů, televizorů, tak mnoha dalších zařízení. S rozvojem internetu se bohužel také zvyšuje množství útoků na počítačové sítě. Z tohoto důvodu se v poslední době klade čím dál tím větší důraz na vývoj algoritmů a programů na detekci těchto útoků a síťových anomálií. Každý provozovatel počítačové sítě určitě uvítá možnost znát, co se děje na jeho síti, ať už se jedná o to, jaká je její vytíženost v průběhu jednotlivých dnů nebo hodin, jaká data jí protékají nebo to, zda došlo a dochází k podezřelým anomáliím a pokusům o útoky. Díky těmto informacím může provozovatel takto monitorované sítě včas podniknout patřičné kroky ke zlepšení bezpečnosti nebo stability.

Tento projekt se zabývá analýzou síťového provozu a detekcí anomálií využitím struktury nazývané $k - ary\ sketch$ podle metod popsaných v článku [8]. Analýza síťového provozu se věnuje sledování jednotlivých síťových toků (flow záznamů) v určitém bodě sítě, zaznamenávání získaných dat a vyhodnocování potřebných informací. Tato bakalářská práce je zasazena do systému pro on-line síťovou analýzu NEMEA, díky které je celá práce soustředěna přímo na vývoj algoritmu pro detekci síťových anomálií. Získávání dat a potřebných informací z nich řeší jiné části systému Nemea. Pojem síťová anomálie nemusí hned znamenat přímo síťový útok. Může se jednat o výskyt chyby způsobené například poruchou hardwaru, výpadkem některého zařízení a podobně. Za anomálii je považováno takové chování průběhu dat na síti, které se významně liší od standardního chování.

Detekcí anomálií se zabývá mnoho známých algoritmů založených na několika různých metodách, kterým bude stručně věnována jedna kapitola tohoto textu. Tato bakalářská práce se zabývá detekcí anomálií pomocí *sketchí*. Ta spadá do kategorie proudových algoritmů, konkrétněji jde o metodu zvanou $Heavy - Change\ Detection$, která sleduje nadměrné odchylky v normálním provozu.

Stručné seznámení s analýzou dat je uvedeno v kapitole 2. Známé techniky používané pro detekci anomálií jsou popsány v kapitole 3. Jak již bylo zmíněno, celý projekt je vytvořen jako modul pro systém NEMEA, jehož detailnějšímu popisu se věnuje kapitola 4. Přesnější popis, co je to sketch, jak funguje a informace o tom jak je tato struktura využívána při detekci anomálií, uvádí kapitola 5. Detailní informace o implementaci celého projektu, způsobu použití a výsledné hodnoty testování této metody na vzorku dat jsou popsány v kapitolách 6 a 7.

Kapitola 2

Analýza síťového provozu

Analýza síťového provozu je proces, při kterém se zkoumají a vyhodnocují data získaná z provozu na určitém zařízení v síti. Tato zařízení se v *NetFlow* nazývají Exportéry. Mohou jimi být routery nebo přímo specializované sondy, které zaznamenávají data nebo je přeposílají na jiné zařízení, které je ukládá nebo dále zpracovává. Síťový tok je definován jako posloupnost paketů mající společnou vlastnost a procházející bodem pozorování za určitý časový interval. Podle NetFlow můžeme za síťový tok považovat posloupnost paketů mající stejnou:

- zdrojovou a cílovou IP adresu,
- zdrojový a cílový port,
- číslo logického rozhraní,
- typ protokolu a hodnotu ToS.

Zpracovávání a vyhodnocování těchto dat, tedy síťových toků, může probíhat jak offline, pro každý příchozí tok, tak online. Bohužel při dnešních rychlostech internetu a množství dat, které prochází jedním bodem, je velmi obtížné a náročné na výkon provádět analýzu přímo online. Pro okamžitou online analýzu existují speciální nástroje, které zachytávají a většinou i rovnou graficky znázorňují procházející data. Jedná se například o známý program NfSen.

Pro offline analýzu je tedy možné data nejprve ukládat a později takto uložená data zpracovávat. Výhodou toho, že pracujeme s uloženými daty, je to, že můžeme pracovat se stejným vzorkem dat vícekrát. Existují metody, které pracují s celými pakety a jejich obsahem, ty jsou většinou dost náročné na paměť a není možné uchovávat příliš velké množství dat. Na druhé straně existují metody, které nepotřebují znát podrobný obsah každého paketu, ale zajímají se pouze o některé informace o daném toku, například počet bytů nebo paketů v něm. I když tyto algoritmy neznají přesný obsah paketů a může dojít ke ztrátě některých důležitých údajů, jsou tyto metody schopné provádět analýzu velmi spolehlivě. Tento způsob také umožňuje vytvářet dlouhodobé statistiky o provozu na síti.

Kapitola 3

Detekce anomálií

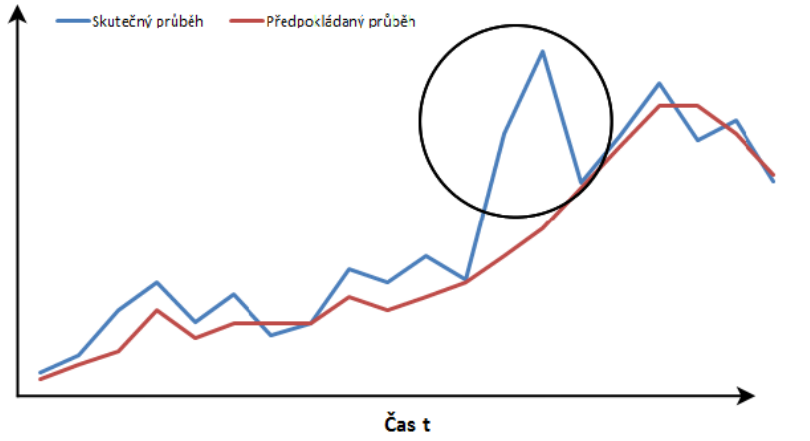
S rozrůstající se popularitou internetu roste i množství útoků na počítačové sítě. Z těchto důvodů se odvětví zabývající se detekcí anomálií stalo velmi důležitou součástí dnešního světa internetu. Detekce anomálií, jak již název napovídá, se věnuje odhalování podezřelých jevů v síťových datech. Tyto jevy se nazývají anomálie. Úplně přesnou definici anomálie nelze jednoduše určit. Obecně lze říct, že anomállii můžeme definovat jako vzorek dat, který neodpovídá normálnímu chování [3]. Příklad výskytu anomálie v časovém úseku a předpokladané chování na síti je znázorněno na Obrázku 3.1, kde svislá osa může reprezentovat počet přenešených toků, paketů, bytů atd. Detekci anomálií lze rozdělit do dvou základních kategorií, kde každá obsahuje metody založené na různých technikách, které mají stejný základ. Jedná se o:

- detekci založenou na modelu,
- bezmodelovou detekci.

Detekce založená na modelu předpodpokládá, že je předem známé chování různých anomálií, toto specifické a předem definované chování se snažíme najít v analyzovaných datech. Tento způsob detekce má své výhody i nevýhody. Mezi výhody bezesporu patří především to, že metody založené na tomto způsobu dokážou poměrně rychle a s vysokou přesností určit a detekovat předem známou anomálii s nízkým počtem falešných poplachů *false positive* [1]. Jejich největší nevýhodou je to, že je nutné předem znát chování hledaného útoku či anomálie, tedy model chování. Pokud vznikne nový typ útoku, který nebude program znát, nebude ho považovat za podezřelý, dokud se nezaznamená jeho model chování do databáze programu. Tyto systémy jsou tedy schopné velmi rychle a především přesně detekovat již známé útoky, bohužel u těch nově vzniklých je tomu naopak, a dokud není definován jejich model chování, prochází bez povšimnutí.

Detekce bez známého modelu se zaměřuje na problém hledání vzorku dat, který neodpovídá obvyklému chování sítě. Veškeré vzorky dat vyčnívající z tohoto normálního chování jsou označeny za anomálie. Každá z těchto dvou metod je založena na úplně odlišeném způsobu, každá má tedy jiné výhody i nevýhody a je použitelná v různých situacích. Mezi výhody této metody můžeme zařadit především schopnost detekovat anomálie bez jejich předešlé znalosti. To algoritmům založeným na těchto metodách umožňuje detekovat široké spektrum anomálií, a především i ty nově vzniklé. Oproti tomu, za hlavní nevýhodu považujeme vysoké množství falešných poplachů (tedy false positive), protože je těžké definovat, co je to normální chování a vhodně zvolit práh

[1] Případ, kdy je vzorek dat považován za anomálii, ale ve skutečnosti se jedná o normální průběh [2].



Obrázek 3.1: Modře je naznačen skutečný tok dat, červeně je naznačeno předpokládané chování. V oblasti označené kolečkem je vidět jak, se skutečný průběh chování značně liší od předpokládaného.

detekce. Pokud se však povede tyto hodnoty zvolit správně, jsou tyto algoritmy velmi efektivní a dnes často používané. Bezmodelově založené metody se dělí do dalších různých kategorií, které jsou popsány v následujících kapitolách. Některé z nich jsou více určeny na konrétní druh anomálie, například skenování portů, ostatní jsou zaměřeny spíše více obecně a detekují různé spektrum anomálií.

U většiny algoritmů nastává několik základních problémů, které je potřeba řešit, a každý z nich se k těmto problémům staví jinak, jedná se především o:

- definovat model normálního chování sítě,
- definovat hranici, kdy už se jedná o anomálii a kdy o běžné chování, rozdíl mezi normálním a abnormálním chováním může být velice malý a není tedy lehké definovat tuto hranici tak, aby nedocházelo k přehlédnutí anomálie nebo hlášení normálního chování jako anomálii,
- někteří útočníci jsou schopni vytvořit útok tak, aby dokázal zjistit jaké je chování sítě, a útok přizpůsobit tomuto chování tak, aby se choval "přirozeně".

Pro bezmodelově založenou detekci existuje široké spektrum různých přístupů, můžeme je rozdělit například takto:

- statistický přístup,
- proudové algoritmy,
- data mining,
- strojové učení.

Další sekce této kapitoly se budou zabývat právě každým z těchto přístupů, každý z nich bude stručně popsán a u některých budou zmíněny i konkrétní metody. Tato bakalářská práce se zabývá detekcí pomocí sketchí, která spadá pod kategorie proudových algoritmů, konkrétněji Heavy changes.

3.1 Statistický přístup

Tento přístup nepřetržitě monitoruje dění na síti a zaznamenává aktivity uživatelů, ze kterých vytváří profil chování. Jako data, které tento přístup zpracovává a vytváří z nich profil chování, mohou sloužit například informace o využití procesoru, přístup k datům nebo informace přímo ze sítě, jako je počet paketů a mnoho dalších. U tohoto přístupu jsou vytvářeny dva základní profily, a to profil normálního chování a současný profil. Jak přichází data, o které se model zajímá, dochází k aktualizaci profilu a průběžnému propočítávání rozdílu mezi současným a normálním profilem chováním. Pokud dochází k odchylkám, systém detekce anomálií je postupně připočítává a pokud překročí určitou zvolenou hranici, hlásí podezření.

Mezi výhody tohoto přístupu patří, jak již bylo zmíněno, že pro nalezení anomálií není potřeba znát chování existujících útoků a dokáže detekovat nově vzniklé a neznámé útoky. Další výhodou tohoto přístupu je to, že dokáže detekovat útoky, které jsou rozloženy do delšího časového intervalu, jako třeba DoS útoky a skenování portů. Jako nevýhodu je nutné zmínit to, že může být často složité vhodně zvolit hranici tak, aby nedocházelo k častým falešným poplachům (false positive). Další nevýhodou, kterou tento přístup v některých případech trpí, je nemožnost stanovit normální chování. Některé sítě mají velmi specifický průběh a je složité stanovit, jak by měl průběh na síti vypadat. Algoritmy založené na tomto přístupu jsou například Wavelet Analysis, Change-Point Detection a Kalmanův filtr.

3.2 Proudové algoritmy

Tento přístup vznikl v reakci na nevýhody dalšího z přístupů pro detekci anomálií zvaného *sampling* neboli vzorkování dat. Algoritmy založené na vzorkování dat vznikly z toho důvodu, že je velmi obtížné při velkém provozu na síti udržovat informace o všech tocích. Jejich princip spočívá v tom, že prozkoumává jen některé toky. Tento přístup má však jednu velkou nevýhodu, a to, že při vynechání některých toků můžeme přijít o důležité informace o anomáliích.

Za účelem překonání tohoto problému vznikly takzvané proudové algoritmy (*Data Stream mining* [2]). Jejich princip spočívá v tom, že zaznamenávají jen některé informace z každého toku, narozdíl od metod využívajících vzorkování, které uchovávají všechny informace, ale pouze z některých toků [12].

Proudové algoritmy se dále dělí do dalších dvou kategorií, a to Heavy hitters a Heavy changes, do té druhé zároveň spadá detekce pomocí k-ary sketchí, které se věnuje tato bakalářská práce.

3.2.1 Heavy hitters

Tato metoda se zaměřuje na vyhledávání toků, které představují značně velký podíl kapacity na lince nebo vysoký podíl aktivního síťového provozu.

3.2.2 Heavy changes

U této metody se zaměřujeme spíše na vyhledávání toků, jejichž chování specifikují náhlé a razantní změny oproti normálnímu chování v určitém časovém intervalu [12]. Pro uchování hodnot (platí i pro Heavy hitters) se používají struktury, které shlukují informace tak, aby byly co nejméně náročné na paměť. Především se jedná o takzvané *sketche*.

3.3 Strojové učení

Mechanismy strojového učení se používají v mnoha odvětvích, nejen u detekce anomálií na síti. Strojové učení můžeme definovat jako schopnost programu učit se a přizpůsobovat svoje chování pro určitý typ problému [10]. Tyto systémy mají určitý vzorek dat, ze kterého se učí a studují chování, a zjištěné poznatky aplikují na nově získaná data [5]. Používají se například i u zjišťování zneužití kreditních karet. Tyto systémy se zaměřují více na to, aby zlepšovaly svůj výkon a učily se ze získaných prostředků, a tím zlepšily svoji efektivitu.

Metoda strojového učení je využívána i v tomto projektu. Díky ní jsme schopni stanovit profil normálního chování provozu na síti a detekovat odchylky právě od takto stanoveného chování.

3.4 Data mining

Data mining, neboli dolování dat, někdy nazývané také jako *knowladge discovery*. Je to proces získávání užitečných informací z velkého množství, jinak nepřehledných, dat. Definic, co to data mining znamená, je mnoho. Jednou z těch základních a známějších je "Data mining je netriviální proces zjišťování platných, neznámých, potenciálně užitečných a snadno pochopitelných závislostí v datech"[6]. Jednoduše řečeno, dolování dat se zabývá zpracováním vstupních informací a získává z nich určitý vzor nebo odchylku, která není obyčejným pohledem a zjištěním patrná. Tato metoda nám umožňuje více se soustředit na samotnou detekci anomálií. Do tohoto odvětví spadá například detekce anomálií pomocí shlukové analýzy (Clustering).

Kapitola 4

NEMEA

Projekt NEMEA, neboli Network Measurement and Analysis, je *framework* pro on-line analýzu síťových dat [1]. Hlavní myšlenkou tohoto projektu je usnadnit práci při vytváření různých algoritmů. Je zaměřen na to, aby programátor vytvářející program, který bude fungovat jako takzvaný *modul* pro systém Nemea, mohl řešit pouze věci týkající se přímo implementovaného algoritmu. Ostatní věci, kterými by se musel v jiném případě zabývat, jako například způsob čtení dat ze sítě a získávání z nich potřebných informací, řeší ostatní *moduly* tohoto systému. Z nich programátor pomocí vestavěných funkcí a podpory knihovny Libtrap snadno získá pro něj potřebné a užitečné informace a nemusí řešit, jak jsou tato data získávána.

Architektura systému Nemea je tedy tvořena základními bloky, kterými jsou právě zmiňované moduly. Jednotlivé moduly jsou vzájemně propojeny pomocí tzv. vstupních a výstupních interface, přičemž každý model může mít 0-N jak vstupních, tak i výstupních interface. Jednou z hlavních výhod tohoto projektu je to, že implementace jednotlivých modulů je prováděná nezávisle jako samostatně běžící program a díky tomu může být vytvářena libovolným způsobem a teoreticky v libovolném jazyce. Jedinou podmínkou je podpora knihovny Libtrap pro daný programovací jazyk.

Jak již bylo v předchozích odstavcích zmíněno, systém je implementován nad knihovnou Libtrap, která umožňuje komunikaci mezi moduly pomocí tzv. interface. Všechny záznamy posílané po jednom interface musí být ve stejném formátu, to znamená, že musí obsahovat stejné typy dat, ale různé interface mohou posílat data v různých formátech. Formát těchto záznamů je specifikován vždy při startu modulu. Protokol, který specifikuje jak tento formát definovat, jak vytvářet a používat záznamy, se nazývá UniRec. Jedná se strukturu psanou v jazyce C, která obsahuje různé položky, jak statické, tak dynamické, jako je například zdrojová IP adresa, zdrojový port a další. Pro usnadnění práce a dosažení potřebné univerzálnosti je tato struktura využívána u každého modulu jak pro výstupní, tak pro vstupní formát dat. Podrobnější popis a vysvětlení, co to je a jak funguje formát UniRec, se nachází v kapitole 4.3.

Nemea je framework pro on-line analýzu, umožňuje tedy práci s daty jak v reálném čase, tedy přímo on-line zpracovávání právě procházejících dat přímo na síti, tak i jejich ukládání do souborů a zpracování tzv. store-and-post-ex, na které jsou zaměřeny některé moduly zpracovávající data z těchto souborů a dokáží později přesně simulovat průběh takto uložených dat. Další výhodou ukládání dat je možnost zpracovávat stejný úsek (vzorek) dat několikrát. To je vhodné například pro testování funkčnosti algoritmů. V následujících kapitolách budou podrobněji rozepsány jednotlivé hlavní části systému Nemea.

4.1 Moduly

Modul je nejzákladnější část systému Nemea. Jedná se o samostatně spustitelný program, který může běžet jako samostatný proces nebo zvlášť na úplně jiném pracovním stroji. Modul je tedy program. Jednotlivé moduly řeší pouze určitou problematiku, za jejímž účelem jsou vytvářeny, ostatní věci řeší jiné moduly, se kterými může komunikovat. Jednou z hlavních výhod takto vytvářených programů je jejich nezávislost a univerzálnost. Každý modul může být napsán libovolným způsobem a teoreticky i v jakémkoliv jazyce, pokud je pro tento jazyk vytvořena podpora knihovny Libtrap. Jednotlivé moduly mezi sebou komunikují pomocí takzvaných interface. Ty mohou být jak vstupní, tak i výstupní, a každý modul jich může obsahovat libovolný počet. Díky tomu, že jednotlivé moduly jsou tvořeny jako samostatně běžící procesy, mohou být do systému přidávány a odebírány (tedy spouštěny a ukončovány) dynamicky. To znamená, že každý z nich může být kdykoliv spuštěn nebo naopak ukončen a nezpůsobí tím pád ostatních modulů nebo celého systému Nemea. Pouze se ukončí komunikace mezi příslušnými moduly a běžící modul čeká na opětovné připojení jiného (nebo znovu stejného) modulu na jeho interface a pokračuje s komunikací v místě, kde byla předtím ukončena. Modul je možné nastavit i tak, že nečeká na ostatní a data posílá bez ohledu na to, jestli jsou na druhé straně přijímána a zpracovávána.

Každý modul vždy při startu vytvoří definovanou šablonu TRAP a pokusí se připojit na specifikované výstupní rozhraní jiného modulu, pokud se toto připojení nezdaří, dochází ve smyčce k opakování pokusu do té doby, než se připojení uskuteční. Pak se spustí běh samotného programu a probíhá komunikace mezi těmito moduly. Při přerušení tohoto spojení se modul, který zůstal běžet, pokouší znovu opětovně připojit na příslušné interface, aby mohl pokračovat ve své činnosti.

Nemea ve své základní podobě obsahuje několik základních modulů. Ty jsou většinou zaměřeny na základní zpracování dat ze sítě a přeposílání takto zpracovaných dat dál do jiných modulů. Na jejich funkčnosti je tedy založena většina ostatních modulů. Některé nejzákladnější budou popsány v následujících podkapitolách.

4.1.1 Nfdump Reader

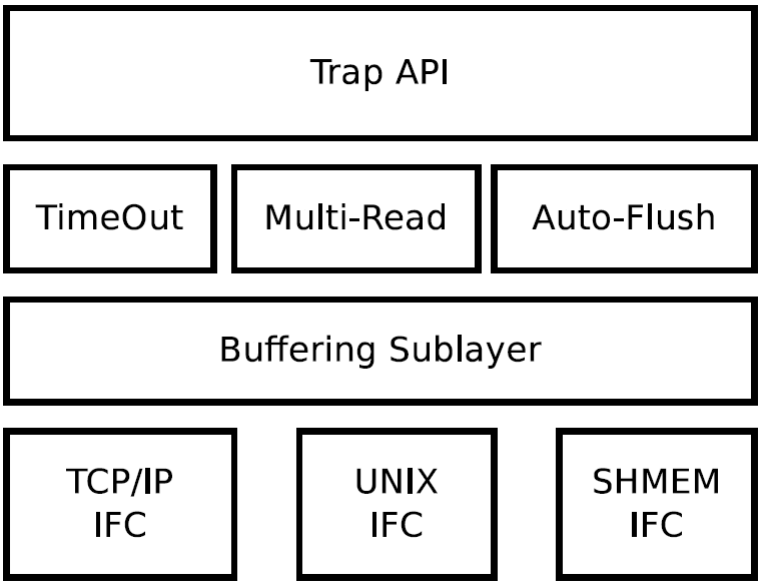
Tento modul slouží jako jeden z hlavních zdrojů dat v systému Nemea. Modul čte flow záznamy ze souborů typu Nfdump. Po načtení jsou zpracovány a převedeny do formátu UniRec a odeslány na výstup modulu. Nfdump Reader nabízí několik různých módů odesílání dat, jako například odesílání dat hned za sebou, jak jen to bude možné, nebo simulovat skutečný tok těchto dat podle jejich časových značek tak, jak přicházely.

4.1.2 Flowcounter

Jednoduchý modul, který pouze čte data na výstupu a počítá množství přijatých paketů, bytů nebo počty přijatých flow záznamů. Tento modul je používán pro testování a často jako výukový modul pro vytváření dalších modulů.

4.2 Libtrap

Libtrap je komunikační vrstva Nemei, která umožňuje sdílenou komunikaci mezi moduly. Ty komunikují po již zmiňovaných interface. Tato komunikace je možná díky tomu, že



Obrázek 4.1: Grafické znázornění architektury knihovny Libtrap, obrázek je převzatý z [1]

interface jsou implementovány jako sdílené objekty. Základní koncept spočívá v tom, že zdrojový modul odešle data na výstupní interface, kde cílový modul ve smyčce očekává data na jeho vstupním interface. Na Obrázku 4.1 je zobrazena architektura knihovny Libtrap, která je rozdělena do čtyř vrstev.

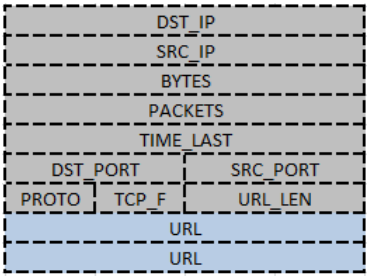
Vrstva obsahující bloky TimeOut, Multi-Read, Auto-Flush se stará o nastavení interface. Umožňuje nám zvolit si, jak bude příslušný interface fungovat. Díky funkci TimeOut můžeme například nastavit dobu, po kterou budou data čekat na příjetí nebo odeslání (tedy blokující nebo neblokující režim). Funkce Multi-Read umožňuje modulům, které podporují více vstupních interface, paralelní režim čtení a funkce Auto-Flush podporuje omezení stárnutí dat na výstupu.

Další vrstva nazvaná Buffering sublayer slučuje zprávy odesílané po interface tak, aby se snížila časová náročnost při odesílání dat, které nejsou odesílány jednotlivě, ale dávkově.

Poslední a nejnižší vrstvou je opět vrstva složená ze tří bloků, které se starají přímo o způsob komunikace, určují jaký typ socketu bude použit. Komunikace mezi moduly může probíhat třemi způsoby. To může být pomocí TCP socketů, UNIX socketů nebo s využitím sdílené paměti. Každý zajišťuje komunikaci mezi moduly založenou na jiném principu. UNIX sockety jsou určeny pro lokální komunikaci, zatímco TCP pro vzdálenou komunikaci mezi moduly běžícími na různých zařízeních. Oba tyto způsoby podporují posílání dat do více cílových modulům, zatímco SHMEM, který je určen pro komunikaci s využitím sdílené paměti, podporuje komunikaci pouze jedna ku jedné.

4.3 UniRec

UniRec neboli Unified Record je speciální datová struktura určující formát dat posílaných po interface, pomocí které mezi sebou jednotlivé moduly mohou jednoduše komunikovat. Jedná se o strukturu psanou v jazyce C, jejíž položky jsou určeny za běhu programu. Struktura Unirec je složena ze statické části obsahující položky, které mají předem



Obrázek 4.2: Ukázka struktury UniRec

definovanou velikost, a z části obsahující položky mající dynamickou velikost. Každé z těchto polí má předem definovaný název a typ. Při spuštění programu je nutné definovat, které položky budou využity. Takto definovanému formátu struktury Unirec se říká šablona. Každé dva moduly komunikující mezi sebou musí mít předem definovaný stejný formát šablony. Příklad toho, jak může šablona vypadat, je znázorněn na Obrázku 4.2.

Komunikace tímto způsobem má mnoho výhod. Mezi ty nejdůležitější patří, že data posílaná mezi moduly mají stejný tvar a je tedy jednoduché pracovat s nimi. Systém Nemea také obsahuje funkce na podporu práce se šablonami. Mezi nejzákladnější patří funkce `ur_get`, která slouží pro získávání jednotlivých položek a funkce `ur_set` pro zápis položek do ní.

Některé položky struktury Unirec jsou implementovány jako speciální datový typ. Prvním z nich je datový typ `i_paddr_t`, který je určen pro práci IP adresami. Ten dokáže pracovat jak s formátem IPv4, tak IPv6. Veškeré IP adresy jsou posílány právě v tomto formátu. V systému Nemea je implementována knihovna `ipaddr.h`, která obsahuje funkce pro práci s IP adresami právě v tomto formátu. Jsou to například funkce pro převod IP adresy na řetězec, řetězce na IP adresu a mnoho dalších.

Dalším datovým typem je struktura `u_rtime_t` pro práci s časem a časovými značkami. Tato struktura ukládá čas jako počet sekund od 1.1.1970 a veškeré časové informace, jako například začátek flow záznamu, jsou implementovány touto strukturou. Nemea opět obsahuje knihovnu pro práci s touto strukturou, která se nazývá `ur_time.h`. Ta také obsahuje několik funkcí pro práci s touto strukturou, které umožnují například získání počtu sekund nebo milisekund z této struktury.

Kapitola 5

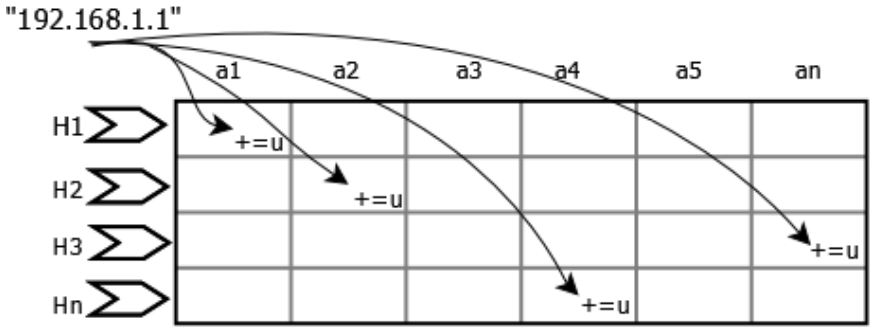
Detekce s využitím sketchí

Princip detekce anomálií nazvaný Sketch-based change detection [8] spočívá v tom, že se porovnává průběh dat v každém časovém intervalu I a hledají se nadměrné odchylky (heavy changes) oproti běžnému chování sítě. V ideálním případě by program na výpočet anomálií na síti touto technikou probíhal asi takto:

Průběh dat na síti bychom rozložili do stejných pravidelně se opakujících intervalů I_1, I_2, I_2 ... , kde každý interval I_t obsahuje za sebou jdoucí prvky α_1, α_2, ... , kde každý prvek $\alpha\ _i(a_i,\ u_i)$ obsahuje klíč a_i a hodnotu přírůstku (*update*) u_i. Je několik možností, jak vhodně zvolit jak klíč a, tak i hodnotu přírůstku u. Pro klíč a se může jednat například o hodnoty zdrojové IP adresy, cílové IP adresy nebo různé kombinace více hodnot, jako třeba kombinace cílové IP adresy a cílového portu nebo zdrojové a cílové IP adresy. Těchto kombinací je mnoho a je na potřebě každého, jak si tento klíč zvolí. Podobně je tomu i u hodnoty přírůstku u, pro který může být použita například hodnota počtu paketů v toku, počtu bytů v toku a další. Volba kombinace klíče a a přírůstku u záleží především na tom, jaké anomálie chceme detekovat. Tento projekt pracuje s hodnotou cílové IP adresy jako klíč a a počtem bytů v toku jako hodnotou přírůstku u.

Pro každý takto vzniklý interval I, bychom provedli výpočet takzvané pozorované (*observed*) hodnoty. Tu získáme tak, že pro každý příchozí klíč (například cílová IP) a_i vypočítáme součet všech hodnot přírůstku u_i v celém tomto intervalu. Takto bychom vytvořili součet všech hodnot přírůstků pro každý příchozí klíč a_i v každém intervalu I v čase t. Dále je třeba vypočítat tzv. *forecast* hodnotu v daném čase t. Pro její výpočet jsou aplikovány speciální algoritmy, které budou zmíněny dále v textu. Z takto získaných hodnot (observed a forecast) vypočítáme hodnotu forecast chyby (*forecast error*), která určuje odchylku od normálního průběhu dat. Tu bychom získali jako rozdíl hodnot observed a forecast pro každý klíč a_i. Vždy, pokud by takto získaná hodnota u některého klíče překročila určitou definovanou mez, takzvaný *threshold*, byl by tento klíč v čase t označen jako podezřelý (jeho hodnota prodělala v daném intervalu neobvykle velkou změnu, Heavy Change).

Bohužel při dnešních rychlostech internetu a při množství hodnot, kterých mohou nabývat klíče, například IPv4 adresa může teoreticky nabývat až 2^{64} (při kombinaci zdrojová-cílová IP adresa) rozdílných hodnot, by takto zaznamenávat hodnoty každého klíče zvlášť bylo velice náročné jak na paměť, tak i na výpočetní výkon. Proto vznikají algoritmy usnadňující práci s takovýmto množstvím dat. Jedním z nich je právě detekce pomocí sketchí. Ta využívá několik svých modelů pro výpočet každé z hodnot popsaných výše. Jedná se o *Sketch observed model, Forecasting model, Change detection model (Sketch error model).*



Obrázek 5.1: Příklad toho, jak může vypadat sketch tabulka a naznačení způsobu uložení informace o příchozím klíči.

Sketch observed uchovává informace o všech hodnotách získaných během každého intervalu I seskupené pomocí hash funkcí do určitého počtu prvku K. Forescasting model je vytvářen z observed modulu po výpočtu pomocí specializovaných algoritmů. Po získání forecasting modulu můžeme vypočítat hodnotu forecast error sketche jako rozdíl mezi S_o a S_f, $S_e = S_o$ - S_f. Díky ukládání dat do sketchí můžeme provádět detekci anomálií s nízkou časovou i paměťovou náročností. Podrobnější popis sketche a detekce pomocí ní bude probrána v následujících odstavcích.

5.1 Sketch

Sketch je datová struktura, která díky svým vlastnostem umožňuje efektivně uchovávat velké množství přicházejících dat s poměrně malou náročností na paměť a výpočetní výkon. Dále je také možné nad ní efektivně provádět potřebné operace k vyhodnocení získaných dat. Sketch struktura je založena na bázi náhodné projekce [9]. Slučuje tedy hodnoty podle získaných indexů do určitého počtu prvků v každém řádku sketch struktury. Hodnota indexu pro každý řádek se získá tak, že určitý klíč (např. IP adresu) v každém řádku sketch struktury zahashujeme podle různých hash funkcí. Po zahashování získáme index, který je vlastně pozice v řádku sketche, na kterou je přičtena hodnota přírůstku. Příklad toho, jak sketch může vypadat, je znázorněn na Obrázku 5.1. Sketch je tedy v podstatě tabulka o rozměrech HxK: $S[i][j] (i \in [H], j \in [K])$.

V tomto projektu je využívána *k-ary sketch*, která vychází z *Count-Min sketche* [4]. K-ary sketch podporuje stejné základní operace pro práci s celou sketch strukturou, jako je tomu u Count-Min sketche, pouze jsou tyto funkce upraveny tak, aby byly efektivnější pro práci, kvůli které byly vytvořeny. Jedná se následující čtyři operace:

- Update
- Estimate
- EstimateF2
- Combine

5.1.1 Update

Základní operace nad sketch strukturou určená k aktualizaci dat v ní. Je prováděna nad každým příchozím prvkem α. Každý příchozí klíč a_i je zahashován každou z hash funkcí H na určitý index K ve sketchi S, kam je přičtena hodnota přírůstku u_i příslušného prvku α. Formální popis této operace je následující:

Update(S, a, u):

$$\text{For } \forall_i \in [H],\ S[i][h_i(a)] += u$$

5.1.2 Estimate

Tato operace slouží k výpočtu odhadu chyby od normálního chování dat, tedy odchylky od průměrné hodnoty ve sketchi. Pro každý klíč v intervalu I je vypočítán odhad chyby Estimate, a pokud překročí hodnotu zvoleného Thresholdu, je tok s daným klíčem považován za podezřelý. Jelikož je každý klíč obsažen v hashi několikrát, podle počtu hash funkcí, zvolí se jako výsledná hodnota medián všech hodnot získaných následujícím způsobem:

Estimate(S, a):

$$\text{medián}(v_a^{h_i})$$

$$v_a^{h_i} = \frac{T[i][h_i a] - sum(S)/K}{1 - 1/K}$$

kde sum(S) je defnována jako součet hodnot ve sketchi. Tu získáme jako součet hodnot v jednom řádku:

$$sum(S) = \sum_{j \in [K]} S[0][j]$$

5.1.3 EstimateF2

Slouží k výpočtu odhadu druhého momentu hodnot ve sketchi. Dále po menších úpravách je tato hodnota použita pro výpočet thresholdu určující mez, kdy se jedná o podezřelý tok a kdy ještě vyhovuje normálnímu chování. Opět je těchto hodnot více než jedna, a proto je vybrán medián ze všech hodnot pro každý řádek sketche.

EstimateF2(S, a, u):

$$medinF_2^{h^i}$$

$$F_2^{h^i} = \frac{K}{K - 1}$$

5.1.4 Combine

Je určena pro zkombinování dvou sketchí. Každé z nich je přiřazena určitá konstanta reprezentující váhu dané sketche. Tou je vynásoben každý její prvek a sečten s takto získanou hodnotou druhé sketche. Tato operace je využívána u výpočtu forecast modelu.

Combine(c_1, S_1, c_2, S_2):

$$S[i][j] = c_1 * S_1[i][j] + c_2 * S_2[i][j]$$

5.2 Činnost

Princip činnosti detekce anomálií s využitím sketchí probíhá následovně. Nejprve si příchozí data rozdělíme do stejných časových intervalů I_1, I_2 ... I_n a v každém tomto intervalu provádíme postupně následující kroky. Pro každý příchozí záznam α v intervalu I_t v čase t provedeme operaci *update*, která postupně aktualizuje hodnoty sketche a na konci intervalu I_t takto vznikne observed Sketch. Observed sketch tedy uchovává informace o příchozích datech pro každý interval I a díky ní získáme hodnoty dalších sketchí nutných pro běh programu.

V dalším kroku, vždy na konci každého intervalu I, vypočítáme hodnotu forecast sketche. Ta pracuje s hodnotami starších observed sketchí získaných v čase menším než t. Pro výpočet forecast sketche existuje několik možných metod založených na odlišných způsobech, kde je vždy různým způsobem přiřazována váha starším sketchím a z nich získávána hodnota forecast sketche. V článku [8] jsou požity metody EWMA a ARIMA. Při testování obou těchto metod bylo zjištěno, že rozdíly výsledků mezi nimi jsou minimální a rozdíl ve složitosti implementace každé z nich je markantní, rozhodl jsem se tedy implementovat jen metodu EWMA. Ta, přestože je výrazně jednodušší, dokáže vracet téměř shodné výsledky jako metoda ARIMA.

Na konci každého intervalu I, s využitím EWMA metody, získáme forecast sketch. Jako poslední zbývá vytvoření error sketche S_e. Ta je definovaná jako rozdíl mezi forecast a observed sketchí. Formálně jako $S_e = S_o - S_f$. Hodnoty v error sketchi nám tedy ukazují rozdíl v chování sítě v čase t oproti normálnímů chování sítě. Nyní tedy máme všechny modely sketchí, které potřebujeme pro samotný výpočet a pro určení výskytu anomálií v daném intervalu I_t.

Nejdříve musíme určit hodnotu thresholdu, tedy mez mezi anomálií a běžným tokem. K jejímu výpočtu je určena zmiňovaná funkce $EstimateF2$. Výsledek této funkce musíme ještě upravit následujícím způsobem: T*$\sqrt{EstimateF2}$, kde T je parametr zadaný přímo uživatelem a v podstatě určuje míru tolerance chyb. Čím větší je, tím je větší bude hodnota thresholdu, tím pádem i vyšší propustnost toků, které nebudou hlášeny jako anomálie.

Dále musíme zjistit hodnotu chyby, kterou vrací funkce $Estimate$, pro každý klíč a_i v intervalu I_t. Každou vypočítanou odchylku porovnáme s již dříve získanou hodnotou thresholdu. Pokud se stane to, že hodnota odchylky překročí stanovenou mez, je tento klíč a označen jako podezřelý a je hlášen výskyt anomálie. Tento postup aplikujeme během každého intervalu I.

Zůstává otázka, jak zjistit, které klíče se vyskytly v intervalu I_t. Nabízí se několik možných řešení. Tím základním, a asi nejjednodušším, je to, že si v každém intervalu postupně uložíme všechny klíče, které se v něm objevily. Pak znovu celý tento seznam

postupně projdeme a vytváříme odhad chyby pro každý z nich. Toto řešení je sice nejjednodušší, ale poměrně náročné na paměť především ve větších sítích, kde se vyskytuje hodně síťových toků.

Další možností je dvouprůchodový režim. Jeho princip spočívá v tom, že každý interval I_t projdeme dvakrát. V prvním průchodu nejprve vytvoříme forrecast error sketch a vypočítáme hodnotu thresholdu. V druhém opět procházíme stejný vzorek dat a počítáme odhad chyby pro každý příchozí klíč. Tento přístup má jedno velké a důležité omezení a to, že při klasické online analýze dat většinou není možné procházet jeden stejný vzorek dat vícekrát. Z toho vyplývá, že tento způsob je většinou použitelný pouze v off-line režimu, kde jsou data uložena a je možné je procházet stále dokola.

Třetí možností je používat klíče přicházející až v dalším intervalu I_{t+1}. Nejdřív vytvoříme $S_e(t)$ v čase t stejným způsobem. Pro vyhledávání podezřelých klíčů a výpočet odhadu chyby počkáme až do dalšího intervalu v čase $t + 1$, kde pracujeme s klíči, které přichází v tomto intervalu. Tato alternativa zahrnuje možnost vzniku určité chyby, která může vzniknout při vynecháních některých klíčů, které se v intervalu I_t objeví, ale v následujících už nikoliv. Tuto chybu většinou můžeme akceptovat, protože průběh většiny anomálií zasahuje do více než jednoho intervalu.

Poslední možností je použití takzvané reversible sketche, která je zaměřena právě na efektivní řešení tohoto problému. Jak funguje reversible sketch je popsáno v článku [11]. V této bakalářské práci je použita první zmiňovaná varianta, tedy ukládání všech příchozích klíčů.

5.2.1 EWMA

Exponentially Weighted Moving Avarage (EWMA) je způsob výpočtu plovoucího průměru, který dává menší a menší váhu datům, která jsou starší a starší [7]. U sketchí je určena k výpočtu forecast modelu. Jeho hodnotu získáme jako vážený průměr starší forecast sketche a observed sketche. Tedy pro výpočet forecast hodnoty v čase t použijeme hodnoty sketchí v čase $t - 1$. To, jakou hodnotu mají novější data oproti starším, určuje takzvaná *smoothing constant* α. Ta může nabývat hodnot od 0 do 1. Definice výpočtu S_f s využitím EWMA metody je následující:

$$S_f(t) = \begin{cases} \alpha * S_o(t-1) + (1-\alpha) * S_f(t-1), & t > 2 \\ S_o(1), & t = 2 \end{cases}$$

Zůstává otázka, jak vhodně zvolit velikost konstanty α. Za nejvhodnější hodnotu α zvolíme tu hodnotu, která minimalizuje chybovost, tedy hodnotu forrecast error. Tu můžeme zjistit experimentálně na základě určitého vzorku dat, kdy stále testuje jeden stejný vzorek. Nejdřív tedy zvolíme hodnotu α postupně rovno $\alpha = 0{,}1$, $\alpha = 0{,}2$, …, $\alpha = 1{,}0$. Získané výsledky porovnáme a zjistíme, kdy je hodnota součtu druhých mocnin error sketche nejmenší. Tedy, kdy je nejmenší hodnota EstimateF2. Tímto získáme přibližný bod a můžeme stejným způsobem zjistit přesněji, kdy je konstanta α nejvhodnější tím, že zkusíme okolí tohoto bodu v rozsahu +0,1 až -0,1. Takto získáme nejvhodnější hodnotu α. Tato hodnota by měla být přepočítána vždy při spuštění na síti, protože charakteristika každé sítě je jiná, a tím pádem může být nejlepší hodnota smoothing konstanty na různých sítích odlišná.

Kapitola 6

Implementace

Implementace programu je pojata jako konzolová aplikace vytvářená v jazyce C pod operačním systémem GNU/Linux. Celá aplikace je vytvořena jako modul pro framework Nemea a pro její běh je nutný běh jiných modulů tohoto systému. Vstupem programu jsou data získávaná ze vstupního interface. Výstupem jsou data odesílaná na výstupní interface. Program byl vytvořen za účelem zjištění, zda je možné implementovat algoritmus pro detekci anomálií pomocí sketchí při dnešních rychlostech internetu. Program nebyl testován přímo v online provozu, ale pouze na uložených testovacích datech. Program je rozdělen do dvou zdrojových souborů, kde jeden obsahuje samotný program a druhý strukturu sketch a funkce implementované nad touto strukturou.

6.1 Vstupní data

Pro vstup i výstup programu jsou využívány interface. Očekávaná vstupní data jsou tedy získávána ze výstupního interface jiného modulu. Šablona pro vstupní interface je definovaná skupinou položek nazvanou `<COLLETCOTR_FLOW>`. Šablona v tomto formátu je získána z výstupního interface modulu NfReader, implementovaného v systému Nemea.

Podle toho, jakým způsobem jsou získaná data zpracovávána, dělíme činnost programu do dvou režimů, a to *Real Time* režim a režim pracující s časovými razítky (*Time Stamp*).

Režim pracující s časovými razítky používá při práci s časem místo skutečného času časová razítka příchozích záznamů. To umožňuje, že data mohou být do modulu posílána libovolnou rychlostí. Při Real Time režimu se pracuje s reálným časem a je tedy nutné, aby flow záznamy přicházely takovou rychlostí, jakou jsou ve skutečnosti sondami vytvářeny. Je tedy vhodný pro práci s online daty, případně je nutné modul NfReader spustit s přepínačem *-R*, který zajistí, že flow záznamy budou odesílány s patřičnými časovými prodlevami podle uložených časových značek.

6.2 Výstupní data

Výstup modulu je opět řešen pomocí interface. Na výstupní rozhraní jsou v každém časovém okně odesílány ty IP adresy, které byly označeny za podezřelé, a dále to, v jakém časovém okně byly detekovány a velikost naměřené odchylky anomálie od předpokládaného chování. Šablona pro výstupní interface je definována textovým řetězcem ve tvaru `"TIMESLOT,SRC_IP,EVENT_SCALE"`

6.3 Spouštění programu

Pro vytvoření spustitelného souboru je nutné program přeložit použitím `make`. Po přeložení se vytvoří spustitelný program `sketch_module`. Ten je možné spustit s několika povinnými přepínači, několika volitelnými přepínači a přepínači, které jsou součástí každého modulu vytvářeného pod systémem Nemea.

Přepínač a

Povinný přepínač *-a* určuje hodnotu konstanty α používáné pro vytváření forecasting modelu s použitím metody EWMA. Hodnota tohoto přepínače musí být číslo v rozsahu 0-1. Při chybném zadání hodnoty je program okamžitě ukončen s návratovým kódem `-1`.

Přepínač t

Určuje velikost konstanty T zmíněné v kapitole 5.2 určující velikost hranice. Jedná se o číselnou konstantu typu double, jejíž hodnota může být teoreticky libovolná, doporučená hodnota je v rozsahu mezi 0,2-0,25. Pokud je program spuštěn bez tohoto přepínače, je jeho hodnota implicitně nastavena na 0,23.

Přepínač T

Takzvaný *testing-only* mód programu. Při spuštění programu s aktivním přepínačem *-T* program nedetekuje podezřelé toky, ale pouze sčítá hodnoty druhých mocnin jednotlivých polí sketche. Tedy, sčítá hodnoty získané po výpočtu hodnoty EstimateF2 v každém časovém intervalu. Na konci je na standartní výstup vypsána takto získaná hodnota. Tento režim slouží k určení nejlepší hodnoty α , tedy i hodnoty přepínače *-a*. Pro její zjištění je nutné program spustit opakovaně nad stejnými daty s různými hodnotami α a vybrat takovou α, při které je vypsána nejmenší hodnota.

Přepínač R

Slouží k přepínání mezi takzvanými Real Time a Time Stamp režimy. Pokud je program spuštěn s aktivním přepínačem *-R*, běží v Real Time režimu. Pokud je spuštěn bez něj, běží podle časových značek.

6.4 Sketch struktura

Implementace struktury sketch a dalších věcí s ní souvisejících se nachází v souborech `sketch_structure.c` s hlavičkovým souborem `sketch_structure.h`. Ty obsahují samotnou strukturu sketch pojmenovanou jako `sketch_table` a funkce implementované nad touto strukturou. Jedná se o základní funkce popsané v kapitole 5 a několik dalších podpůrných funkcí pro práci se sketchí.

Sketch struktura je definovaná jako:

```
typedef struct{
sketch_item **data;
unsigned int hashes_c;
unsigned int items_c;
}sketch_table
```

kde prvky `hashes_c` a `items_c` udávají počet hash funkcí a počet indexů každého řádku sketche. Tedy počet řádků a sloupců tabulky. Prvek `**data` je dvourozměrná tabulka prvků `sketch_item`, které obsahují přímo hodnoty jednotlivých buněk sketch struktury. Každý prvek sketch tabulky `sketch_item` je definován jako:

```
typedef struct{
double update;
unsigned int key;
}sketch_item
```

Proměnná `update` je využívána k uchování hodnoty v každé buňce tabulky. Proměnná `key` byla využívána k testovacím účelům při vyvíjení programu. Z důvodu možnosti dalšího rozvoje a úprav programu byla tato proměnná zachována.

Nad strukturou `sketch_table` bylo implementováno několik podpůrných funkcí pro práci s ní. Při každém vytvoření sketch tabulky je nutné nejprve inicializovat tabulku a alokovat potřebné místo v paměti. To zajišťuje následující funkce, která podle zadaných hodnot H a K naalokuje potřebnou paměť pro tabulku o rozměrech $H \times K$

```
int sketch_init(sketch_table* table , int H, int K)
```

Implementaci operací popsaných v kapitole 5, jako jsou *Update*, *Estimate*, *EstimateF2*, *Combine*, zajištují funkce popsané v následujících odstavcích. První z těchto funkcí je

```
int sketch_insert(sketch_table *sketch, char *key, unsigned int value)
```

reprezentující operaci *Update*, která přičte hodnotu `value` na příslušný index v sketchi podle klíče `key`. Další z nich je

```
double sketch_estimate(sketch_table *sketch,char *key)
```

která implementuje operaci *Estimate* tak, že klíč `key` zahashuje jednou z hash funkcí H. Tímto je získán index, podle kterého je vypočítána odchylka od průměru v daném řádku pro buňku nacházející se právě na tomto indexu. Tato hodnota je uložena do pole a postup se opakuje pro další hash funkci. Po získání těchto hodnot pro každý řádek funkce vrací medián ze získaných hodnot. Na podobném principu je založena i funkce

```
double sketch_estimateF2(sketch_table *sketch)
```

Ta vypočítá a uloží hodnoty pro každý řádek tabulky podle příslušného vzorce a z těchto hodnot vybere střední hodnotu, tedy jejich medián. Poslední z funkcí, které implementují základní definované operace nad sketchí, je

```
sketch_table sketch_combine(double constant1, double constant2,
 sketch_table *S1, sketch_table *S2)
```

Reprezentuje operaci *Combine* a její princip spočívá v tom, že podle koeficientů vynásobí a sečte každou buňku zadaných sketch tabulek. Její návratovou hodnotou je přímo tímto způsobem vytvořená sketch.

Mimo již zmíněné funkce soubor `sketch_structure.c` obsahuje ještě několik dalších funkcí, ty jsou většinou implementovány pro usnadnění práce se sketch tabulkou, jako je například `sketch_sum` pro vypočet sumy hodnot na jednom řádku, `sketch_clear` pro vynulování všech hodnot ve sketchi.

6.5 Sketch modul

Implementace samotného modulu se nachází v souboru `sketch_module.c`. Ten se při spuštění snaží jako první inicializovat a připojit k patřičnému interface na výstupu jiného modulu běžícího v systému Nemea. Modul čeká, dokud nedojde ke korektnímu připojení a získání prvního záznamu se vzorkem dat. Další činnost modulu je rozdělena do dvou částí. Každá z nich je implementována jako samostatné vlákno běžící v nekonečné smyčce. Pokud se na vstupu objeví značka konce dat (při získávání dat ze souboru), je program korektně ukončen, jinak stále běží. Každé z těchto vláken vykonává specifickou činnost. První z nich čte data získaná na vstupu, získává z nich potřebné informace a zpracovává je do sketch tabulky. Druhé vlákno zpracovává sketch tabulku získanou z prvního vlákna a zjišťuje v ní výskyty anomálií. Pokud je detekována anomálie, toto vlákno odešle informace o výskytu anomálie na výstupní interface. V následujících dvou podkapitolách bude popsána činnost každého z jednotlivých vláken zvlášť.

6.5.1 Zpracování vstupu

První vlákno, běžící v nekonečné smyčce, se zabývá získáváním potřebných dat z příchozích záznamů a vyplňováním získaných hodnot do příslušné sketch tabulky. Prvním krokem prováděným tímto vláknem je získání dat. O to se stará funkce `trap_get_data` implementovaná v knihovně `trap.h`. Tím se načte příchozí záznam ze vstupního interface, který je ve formátu struktury UniRec, a proto musíme získat potřebné informace z této struktury. K tomu slouží funkce `ur_get`, která vrací vyžadovaný záznam v této struktuře. Poté, co jsou takto získána všechna potřebná data (IP adresa a počet bytů), jsou vložena do sketch struktury s využitím funkce `sketch_insert` a zároveň je do pomocného souboru uložena informace o výskytu dané IP adresy. Struktura, do které jsou získané informace vkládány, slouží jen jako lokální sketch.

Druhá část tohoto vlákna testuje, zda se přicházející data vyskytují stále ve stejném časovém intervalu *I*. Pokud je překročen limit a data přeskočí do dalšího intervalu *I*, je zmiňovaná lokální sketch nakopírována do struktury, která je přístupná z obou vláken. Poté je indikována přístupnost této sketche druhému vláknu. K indikaci přístupnosti je využívána globální proměnná `data_ready`. Jakmile druhé vlákno zjistí, že její hodnota byla nastavena na 1 (připraveno), začne zpracovávat získanou sketch tabulku.

6.5.2 Detekce

Druhé vlákno tedy čeká, dokud první vlákno neuvolní patřičná data. Pokud pomocí proměnné `data_ready` zjistí, že data v tabulce sketch jsou připravena, spustí se činnost druhého vlákna, která je určena na detekci podezřelých klíčů. Jeho princip je implementován podle postupu zmíněného v kapitole 5.

Nejprve tedy toto vlákno vypočítá hodnotu zmiňované forecasting sketche s využitím funkce `EWMA_module`. Po získání této sketche je vytvořena sketch error jako rozdíl mezi forecasting a observed sketchí. V dalších pár krocích je z této sketche vypočítána konečná hodnota thresholdu. Pokud je program spuštěn s parametrem *-T*, tak v těchto místech je cyklus ukončen a čeká se opět, dokud nejsou data ve sketchi připravena. V opačném případě se další kroky tohoto vlákna zabývají zjišťováním, zda daný klíč vyhovuje vypočítané hranici nebo je považován za podezřelý.

Postupně je ze zmiňovaného pomocného souboru načítána jedna IP adresa po druhé. Pro každou IP adresu sloužící jako klíč je vypočítána hodnota odchylky použitím

zmiňované funkce `sketch_estimate`. Pokud je tato hodnota pro daný klíč větší než hodnota meze, je označen za podezřelý. Aby nedocházelo k opakovanému nahlašování stejného klíče v jednom intervalu, je implementován mechanismus, který ukládá již nahlášené klíče a pokud se klíč opakuje, není znovu hlášen.

Pro každý podezřelý klíč je naplněna šablona UniRec, do které jsou uloženy patřičné informace o anomálii a odeslány na výstupní interface.

Kapitola 7

Testování

Testování modulu probíhalo na datech z reálného provozu na síti. Jedná se o data získaná ze sítě VUT uložených ze dne 13-02-2012 v časech mezi 7:00-13:00. Informace z těchto souborů jsou zpracovávány modulem NfReader a z něj odesílány na vstup tohoto projektu. Množství dat přenášených v testovacím vzorku v průběhu celého časového intervalu je zobrazeno na Obrázku 7.1.

Veškeré testy byly prováděny nad tímto vzorkem dat, modul nikdy nebyl nasazen přímo v on-line provozu, ale detekce on-line je podporována a je možné ji i simulovat, a to v kombinaci s přepínačem *-R* při spouštění NfReaderu. Z důvodů složitého ověřování funkčnosti byly vytvořeny pouze základní testy tohoto projektu, a to, abychom zjistili, zda je modul schopen obstát v charakteristice dnešních počítačových sítí.

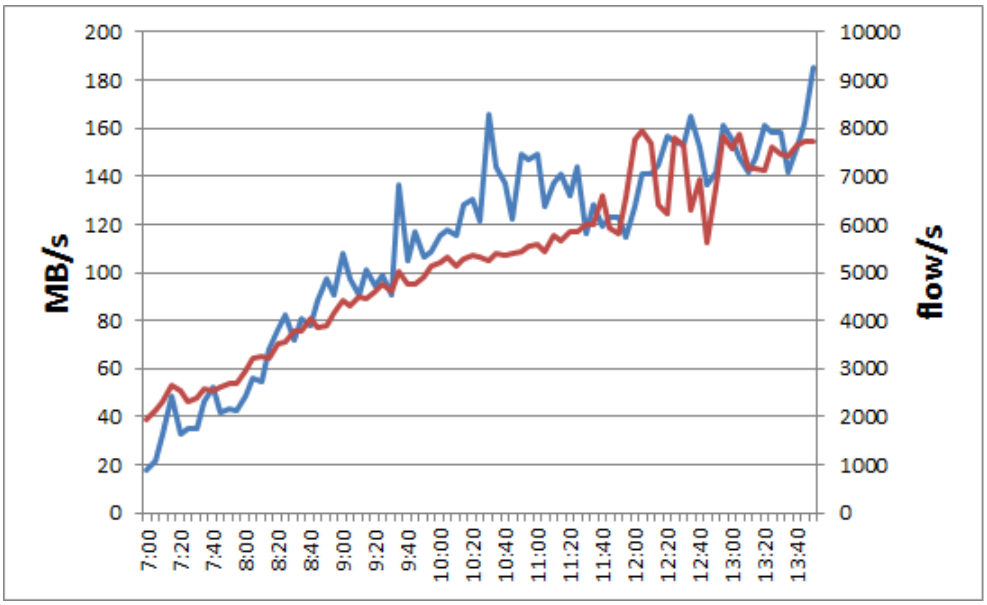
V následujících kapitolách je provedeno několik testů, které prověří, jak se mění funkčnost celého modulu v závislosti na změně některých parametrů, jak vypadají data uložená ve sektchi a zda detekované klíče odpovídají pravděpodobným anomáliím v provozu.

7.1 Vliv rozměrů sketch struktury

Celý projekt byl podroben několika základním testům, jeden z nich měl ukázat, jaký vliv mají rozměry sketch struktury na její funkcionalitu, tedy především množství hlášených poplachů při různě zvoleném množství jak hash funkcí, tak i počtu prvků řádku. Testování probíhalo při rozměrech sketchí o velikostech $H \times K = 3{\times}32000$, $3{\times}16000$, $3{\times}8192$, $3{\times}4096$, $5{\times}32000$, $5{\times}16000$, $5{\times}8192$ a $5{\times}4096$..

Z výsledků získaných testováním bylo zjištěno, že rozměry tabulky téměř neovlivňují správnou funkčnost programu při výskytu anomálie. Při zvolení počtu hash funkcí $H = 5$ byly změny při různých hodnotách počtu prvků zanedbatelné. Ve většině případů byl detekován stejný počet anomálií jak při volbě parametru K=32000, tak i K=4096. Pokud docházelo k různým počtu hlášených poplachů, tak rozdíl mezi nimi byl maximálně v jedné anomálii.

Při zvolení hodnoty $H = 3$ již dochází k viditelnějším změnám při různých hodnotách K. Viditelné odchylky nastávají při volbě parametru K=8192 a nižší, kde oproti hodnotám K=16000 a K=32000 dochází k rozdílným počtům hlášených anomálií. Tento rozdíl se ve většině případů pohyboval v rozmezí 2-4 poplachů, ale v některých případech se výsledné hodnoty lišily o 10 a více hlášených anomálií (především při volbě K=4096). Z tohoto důvodu spouštění programu s hodnotou $K < 16000$ není doporučeno.



Obrázek 7.1: Množství přenášených dat na VUT síti 13.2.2012 v průběhu mezi 07-13 hod. Červená osa zobrazuje počet toků za vteřinu, modrá osa počet MB/s.

	H = 5	H = 3
K = 4096	2,385	2,397
K = 8192	2,397	2,433
K = 16000	2,421	2,65
K = 32000	2,53	3,481

Obrázek 7.2: Tabulka znázorňující průměrný počet hlášených anomálií v jednom pětiminutovém intervalu *I*.

V předchozích odstavcích byl popsán vliv parametru K a nyní se budeme zabývat parametrem H a tím jak ovlivnuje činnost programu. V tomto bodě zkoumání byly porovnány výsledky při stejných hodnotách počtu prvků v řádu a rozdílných počtech hash funkcí. Ze zkoumaných hodnot bylo zjištěno, že počet hash funkcí v podstatě neovlivňuje počty hlášených anomálií. Průměrné hodnoty výskytů anomálií v jenotlivých intervalech jsou znázorněny v tabulce 7.2.

Z naměřených hodnot tedy můžeme určit, že nejlepších výsledků je dosahováno při spuštění s počtem prvků v řádku sketche K=32000 při kterých bylo dosahováno nejlepších výsledků. Počet hash funkcí funkcionalitu programu téměř neovlivní a z důvodů menší časové a paměťové náročnosti jsme jako vhodnější zvolili počet hash funkcí $H = 3$.

7.2 Časová naročnost

Dalším cílem testování bylo zjistit, jestli projekt bude stíhat zpracovávat příchozí data za rozumný čas tak, aby byl použitelný v dnešním běžném provozu sítě. Přesněji tedy, cílem tohoto testováni bylo zjistit, jak velký vliv má velikost sketchí na rychlost celého projektu a jak rychle dokáže zpracovávat získaná data.

Toto testování probíhalo tím způsobem, že celý projekt byl spouštěn v Time Stamp

	H = 5	H = 3
K = 4096	7,602	6,614
K = 8192	7,421	6,265
K = 16000	7,433	6,457
K = 32000	7,8	6,783

Obrázek 7.3: Tabulka znázorňující průměrný čas v sekundách potřebný na zpracování jednoho intervalu *I*.

režimu a po každém zpracování pětiminutového intervalu I byl vypsán čas uplynulý od konce předchozího intervalu I s rozdílnými rozměry sketchí. Hodnoty rozměrů tabulky byly zvoleny stejně jako v předchozím testování. Z těchto měření bylo vypozorováno, že rozměry tabulky mají na rychlost výpočtu minimální vliv. Rozdíl zpracování při změně počtu prvků v řádcích tabulky je zanedbatelný, jediný menší rozdíl nastal při navýšení počtu prvků hash funkcí ze 3 na 5. Při této změně se průměrná rychlost zpracování jednoho pětiminutového intervalu I navýšila přibližně o jednu vteřinu. Tabulka 7.3 ukazuje průměrné hodnoty zpracování jednotlivých intervalů.

Z měření bylo dále zjištěno, že je program schopen zpracovat průměrně cca 180 000 toků za jednu vteřinu. Tato rychlost zpracování je dostatečná k tomu, aby byl program schopen zpracovávat data ze všech měřících sond umístěných v síti *CESNET*

7.3 Velikost thresholdu

Velikost thresholdu je redukována pomocí hodnoty t, která je zadávána při spuštění programu uživatelem. Tento test byl zaměřen právě na to, jak se program bude chovat při různých velikost této hodnoty. Bylo vytvořeno několik testů pro hodnoty thresholdu v rozsahu 0,15 - 0,30, z těchto informací byly zjištěny následující informace.

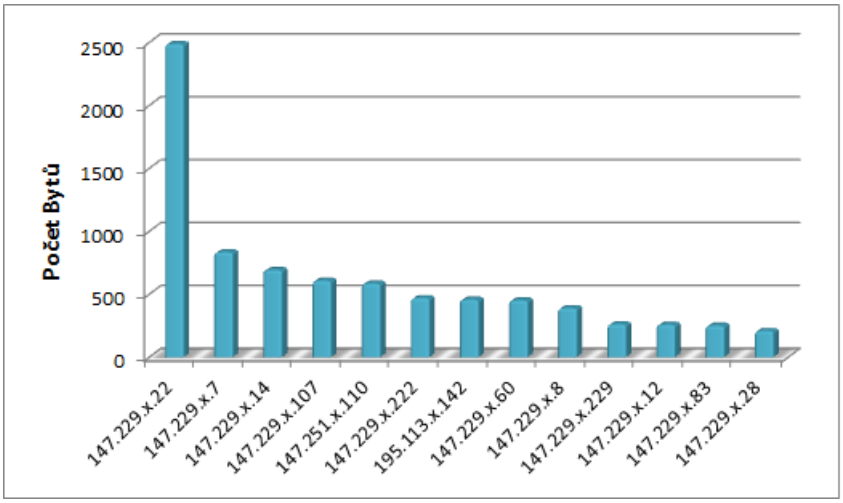
U hodnot v rozsahu 0,15 - 0,20 dochází k častějším výskytům hlášení anomálií. Většinou se jedná o rozdíl 1-2 hlášených anomálií při změně parametru t o hodnotu 0,02 v tomto rozsahu.

Při zadání hodnoty větší jak 0.20 dochází v některých místech k většímu poklesu hlášených anomálií oproti počtu hlášených anomálií s parametrem t menším než 0,20.

Ze zkoumaných dat bylo vyhodnoceno, že ideální rozsah tohoto parametru je mezi 0,20 - 0,25, v případě hodnoty 0,2 (a i nižší) dochází k častějším případům, kdy je hlášena anomálie, která odpovídá spíše normálnímu chování, a naopak při hodnotě 0,25(a vyšší) dochází k případům, kdy je anomálie považována za normální chování.

7.4 Porovnání výsledků

Je velice těžké najít způsob, kterým dokážeme zjistit, zda anomálie, která je hlášena vytvořeným programem, je opravdu anomálií, mj. proto, že neexistuje žádná exaktní definice anomálie. Řešení tohoto problému jsem se pokusil alespoň přiblížit tím, že jsem si použitím programu *nfdump* ze souborů s uloženými daty nechal vypsat 1000 toků s největším objemem přenesených dat a sloučil toky patřící k jednotlivým klíčům, tedy cílovým IP adresám. Z těchto hodnot jsem si vytvořil grafy pro určité intervaly I, a pokud se v grafu vyskytoval klíč, který se oproti ostatním vykazoval výrazně větším objemem přenášených bytů, považoval jsem jej za možnou anomálii a porovnal, zda detektor tento



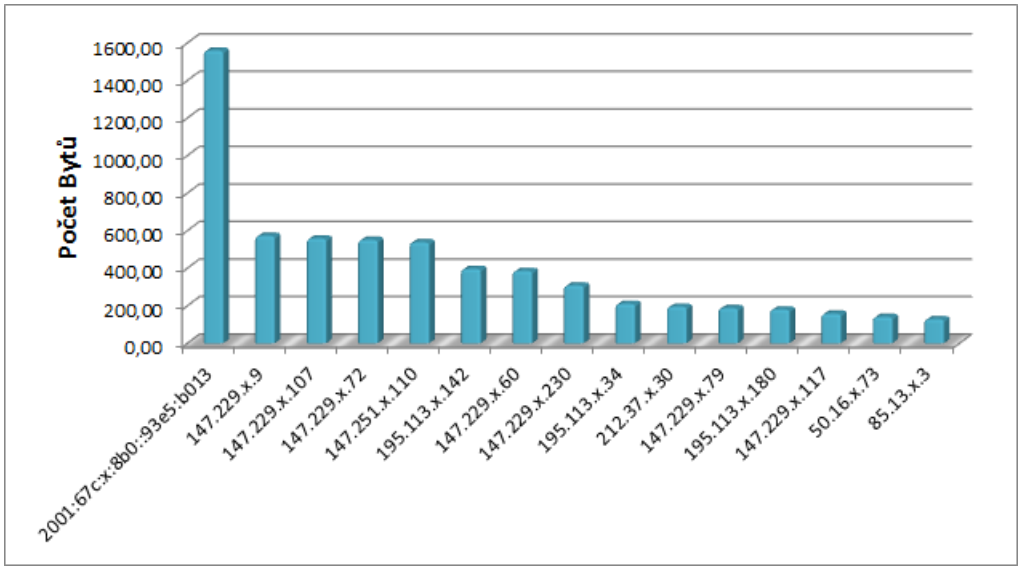
Obrázek 7.4: IP adresy s největším počtem přenesených bytů v čase 08:30

klíč hlásí jako podezřelý. Při testování jsem bral v úvahu i spuštění projektu s různou hodnotou thresholdu. Některé zajímavější výsledky jsou popsanány v následujících odstavcích.

První příklad se nachází v čase 8:30. Počet přenesených bytů pro prvních několik cílových IP adres je znázorněn na Obrázku 7.4. Při tomto rozložení dat je vidět, že počet přenesených bytů u cílové IP adresy 147.229.x.22 má oproti ostatním mnohem větší hodnotu. Můžeme tedy předpokládat, že se jedná o anomálii. Při spuštění programu s parametrem t = 0,18 jsou jako anomálie ohlášeny 3 IP adresy, a to 147.229.x.22, 147.229.x.14 a 147.229.x.7. Pravděpodobně dvě z nich jsou hlášeny jako anomálie, přestože zapadají do normálního chování sítě. Při spuštění s t = 0,2 jsou již vypsány pouze 2 IP adresy, a to 147.229.x.22 a 147.229.x.14. V tomto případě je tedy pravděpodobně hlášena jedna false postive anomálie. Se zadáním hodnoty t = 0,23 je vypsána pouze IP adresa 147.229.x.22, která je právě za anomálii považována. A nakonec při hodnotě t = 0,25 není vypsána žádná podezřelá adresa, zde tedy dochází k ignorování pravděpodobné anomálie.

Druhý případ se nachází v čase 8:00. IP adresy s největším počtem přenesených bytů a jejich hodnoty jsou zobrazeny na Obrázku 7.5. V tomto případě při spuštění s hodnotou t=0,18 je jako anomálie ohlášeno 6 IP adres. Jsou to 2001:67c:x:8b0::93e5:b013, 147.229.x.72, 147.251.x.110, 147.229.x.9, 147.229.x.107 a 195.113.x.142. Z těchto adres, jak je vidět na obrázku 7.5, se dá pravděpodobně považovat za anomálii pouze IP adresa 2001:67c:x:8b0::93e5:b013. U ostatních se, předpokládám, jedná o chybné hlášení anomálie. Při spuštění s parametrem t = 0,23 (a i vyšším) dochází k vypsání pouze právě IP adresy 2001:67c:x:8b0::93e5:b013, což se dá považovat za správné chování.

K podobným situacím dochází i v dalších případech v jiných čase. Všechny hodnoty zobrazené v grafech nejsou úplně přesné z toho důvodu, že programem nfdump bylo vypsáno pouze prvních 1000 nejaktivnějších toků. I přesto můžeme výsledky programu považovat za dobré. Z naměřených výsledků je vidět, že pravděpodobně nejpřesnější výstupní hodnoty program hlásí při spuštění s hodnotou parametru t=0,23.



Obrázek 7.5: IP adresy s největším počtem přenesených bytů v čase 08:00

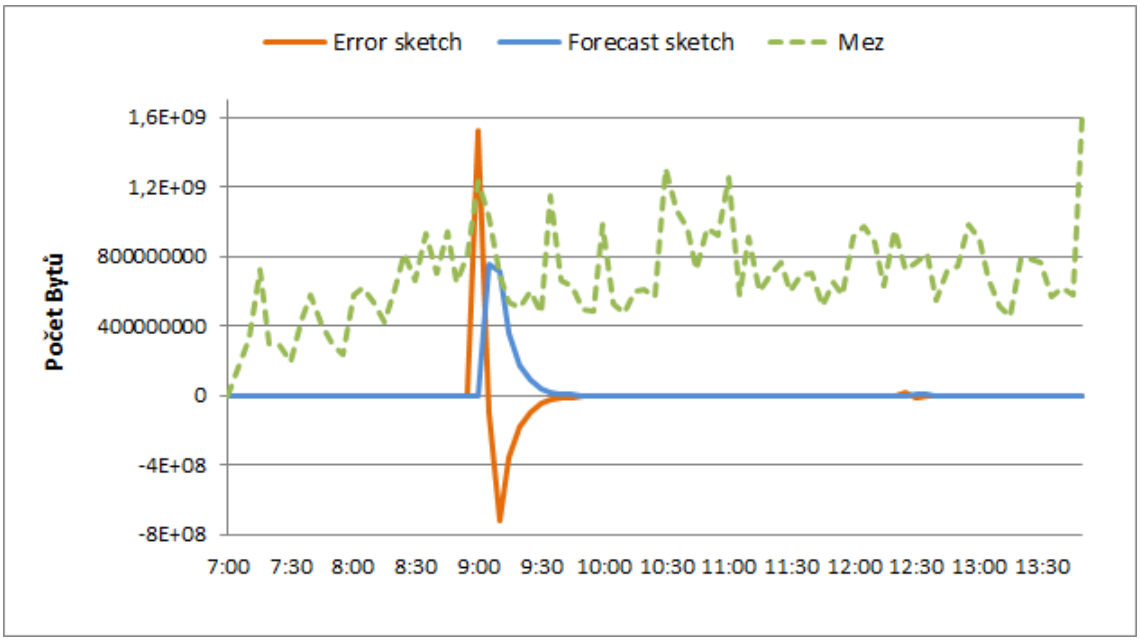
7.5 Ukázky hodnot ve sketchích

Tato kapitola je věnována ukázkám hodnot v jednotlivých sketchích v průběhu času. Tomu, jak ve sketchi vypadá výskyt anomálie a jak to ovlivní jednotlivé druhy sketchí.

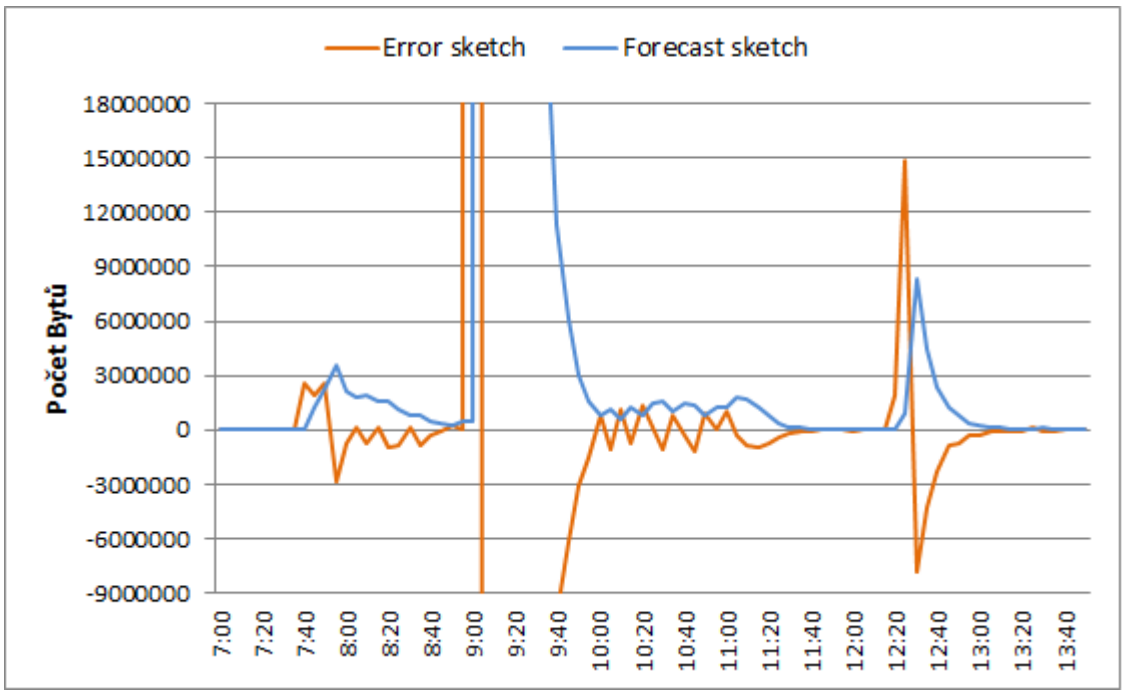
Na Obrázku 7.6 jsou znázorněny hodnoty jednoho políčka ve forecast sketchi a error sketchi, které bylo vybráno tak, aby se v něm objevila anomálie. V čase kolem 09:00 můžeme vidět abnormální nárůst hodnot v obou sketchích. Hodnota error sketche překročí stanovenou mez a je hlášena anomálie. V několika následujících intervalech jdou hodnoty error sketche do vysokých záporných čísel. To je způsobeno tím, že hodnota observed sketche je v dalším intervalu po odchycení anomálie opět v běžných hodnotách, zatímco hodnota forecast sketche, která je vyhlazeným průběhem hodnot, se v několika dalších intervalech stabilizuje ($S_e = S_o$ - S_f).

Na Obrázku 7.7 je přiblížen průběh sketchí z Obrázku 7.6. Můžeme zde vidět, že ke stabilizování forecast sketche po výskytu anomálie dojde až po hodině od výskytu anomálie. V čase kolem 12:00 se objeví nárůst, který však ve větším měřítku zdaleka nedosahuje takových hodnot, aby byl považován za anomálii.

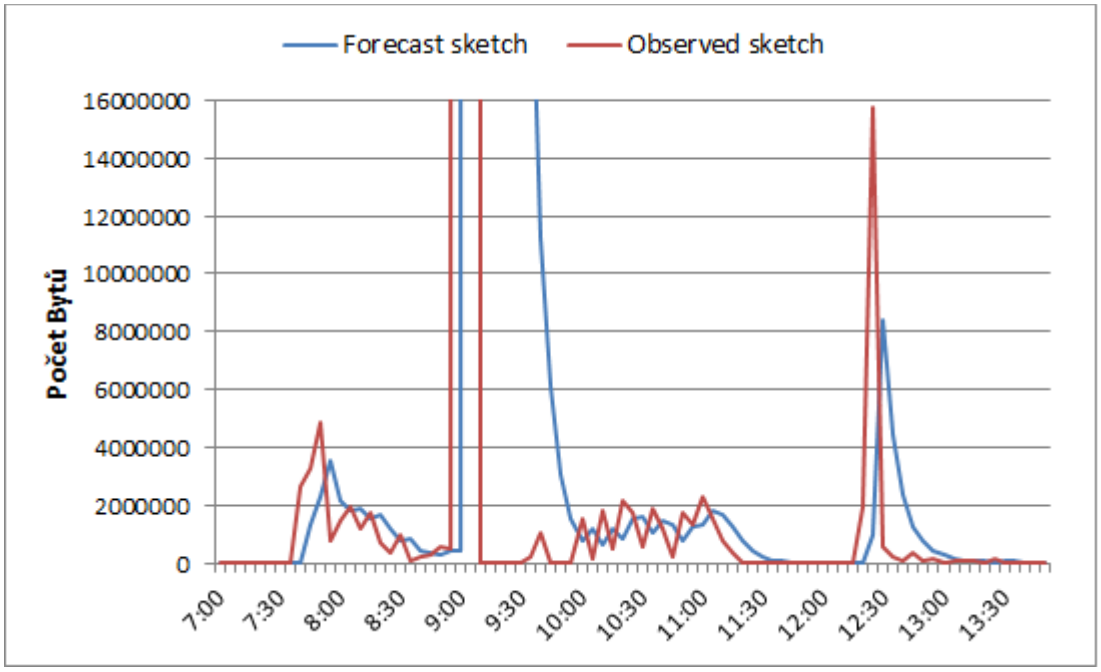
Obrázek 7.8 naznačuje práci algoritmu EWMA. Je zde vidět skutečný průběh (tedy observed sketch) a jak z něj vzniká vyhlazená hodnota (forecast sketch). Můžeme vidět, že po výskytu anomálie jdou skutečné hodnoty zpět k nule, ale forecast sketch, která zohledňuje i předchozí hodnoty, nabývá běžných hodnot až po uplynutí delšího časového úseku.



Obrázek 7.6: Graf znázorňující hodnoty ve forecast sketchi a error sketchi a výskyt anomálie.



Obrázek 7.7: Přiblížený průběh ve forecast a error sketchích.



Obrázek 7.8: Práce algoritmu EWMA při vytváření forecast hodnoty z observed hodnoty.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo implementovat program pro detekci síťových anomálií s využitím sketchí podle článku [8] a otestovat metodu při dnešních rychlostech sítě. Celý projekt byl od začátku implementován jako modul pro systém Nemea, což bylo velmi přínosné především pro rychlost celého algoritmu, který se téměř nemusel zabývat analýzou síťových dat. K tomuto účelu byl použit modul NfReader.

Ze získaných výsledků bylo zjištěno, že program při správně nastavených parametrech dokáže poměrně spolehlivě a v dobrém čase detekovat anomálie. Implementovaný modul je dostatečně rychlý, aby dokázal zpracovávat v reálném čase data ze všech monitorovaných sond umístěných v síti Cesnet (deset 10Gb/s linek). Jako nejvhodnější nastavení sketche byly po experimentálním testování zjištěny hodnoty *H=3, K=32000, t=0.23*. S tímto nastavením dokáže sketch vracet nejspolehlivější výsledky při detekci anomálií. Možností rozšíření projektu je implementace algoritmu *reversible sketche* podle článku [11], která dokáže vyřešit problém s pamatováním si, které adresy se objevily v jednotlivých intervalech.

Literatura

[1] Bartoš, V.; Žádník, M.; Čejka., T.: *Nemea: Framework for stream-wise analysis of network traffic*. CESNET, Praha, Czech Republic, 12 2013.

[2] Callegari, C.; Cyprus, N.: Statistical approaches for network anomaly detection. Prezentace na konferenci ICIMP. 2009.

[3] Chandola, V.; Banerjee, A.; Kumar, V.: Anomaly Detection: A Survey. *ACM Computing Surveys*, ročník 41, č. 3, Červenec 2009: s. 15:1–15:58, ISSN 0360-0300.

[4] Cormode, G.; Muthukrishnan, M.: Approximating Data with the Count-Min Sketch. *IEEE Software*, ročník 29, č. 1, Leden 2012: s. 64–69, ISSN 0740-7459.

[5] DeOrio, A.; Li, Q.; Burgess, M.; aj.: Machine learning-based anomaly detection for post-silicon bug diagnosis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, EDA Consortium, 2013, s. 491–496.

[6] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.: From data mining to knowledge discovery in databases. *AI magazine*, ročník 17, č. 3, 1996: str. 37.

[7] Hunter, J. S.: The exponentially weighted moving average. *Jurnal of Quality Technology.*, ročník 18, č. 4, 1986: s. 203–210.

[8] Krishnamurthy, B.; Sen, S.; Zhang, Y.; aj.: Sketch-based Change Detection: Methods, Evaluation, and Applications. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, New York, NY, USA: ACM, 2003, ISBN 1-58113-773-7, s. 234–247.

[9] Muthukrishnan, S.: Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, ročník 1, č. 2, Srpen 2005: s. 117–236, ISSN 1551-305X.

[10] Patcha, A.; Park, J.-M.: An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends. *Computer Networks*, ročník 51, č. 12, Srpen 2007: s. 3448–3470, ISSN 1389-1286.

[11] Schweller, R.; Gupta, A.; Parsons, E.; aj.: Reversible Sketches for Efficient and Accurate Change Detection over Network Data Streams. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, New York, NY, USA: ACM, 2004, ISBN 1-58113-821-0, s. 207–212.

[12] Thottan, M.; Liu, G.; Ji, C.: Anomaly detection approaches for communication networks. In *Algorithms for Next Generation Networks*, Springer, 2010, s. 239–261.

Příloha A

Obsah CD

- Instalační balík systému Nemea
- Soubory se zdrojovými kódy v adresáři `sketch`
- Návod na použití v souboru `readme.txt`
- Bakalářská práce v elektronické podobě