VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

FILTRACE A PROFILOVÁNÍ IP TOKŮ

IP FLOW FILTRATION AND PROFILING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL SEDLÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN KUČERA**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Sedlák Michal**
Program: Informační technologie
Název: **Filtrace a profilování IP toků**
IP Flow Filtration and Profiling
Kategorie: Počítačové sítě

Zadání:

1. Nastudujte si protokol IPFIX a modulární kolektor IPFIXcol2.
2. Seznamte se s problematikou filtrování a jeho použití pro IP toky.
3. Navrhněte moduly pro filtraci a profilování záznamu o IP tocích pro kolektor.
4. Implementujte moduly a zaměřte se na obecnou podporu IPFIX položek v rámci jazyka filtru.
5. Vytvořenou implementaci otestujte a proveďte výkonnostní testy.
6. Diskutujte dosažené výsledky a možná rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kučera Jan, Ing.**
Konzultant: Huták Lukáš, CESNET
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 25. října 2019

Abstrakt

Tato práce se zabývá filtrováním a profilováním IP toků, primárně pak dat systému IPFIX. V rámci práce je navrhnuta a implementována obecná filtrační komponenta, která má za cíl být dostatečně efektivní a flexibilní pro použití i v jiných projektech týkajících se IP toků. Tato komponenta je poté uzpůsobena pro práci s daty ve formátu protokolu IPFIX a integrována do existujícího modulárního kolektoru IPFIXcol2 v podobě pluginů umožňujících filtrování průchozích IPFIX dat a jejich třídení do profilů.

Abstract

This thesis addresses the problem of filtering and profiling IP flows, primarily data of IPFIX systems. Within the work, a general filtering component is designed and implemented, which aims to be sufficiently efficient and flexible for use in other projects related to IP flows. This component is then adapted to work with data in the IPFIX protocol format and integrated into the existing modular collector IPFIXcol2 in the form of plugins adding the support for filtering of passing IPFIX data and their sorting into profiles.

Klíčová slova

monitorování sítí, filtrování, formální jazyky, profilování, IP toky, IPFIX, kolektor

Keywords

network monitoring, filtering, formal languages, profiling, IP flows, IPFIX, collector

Citace

SEDLÁK, Michal. *Filtrace a profilování IP toků*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

Filtrace a profilování IP toků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Kučery. Další informace mi poskytl konzultant ze společnosti CESNET, Ing. Lukáš Huták. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Sedlák
28. května 2020

Poděkování

Chtěl bych poděkovat Ing. Lukáši Hutákovi za jeho podporu a mnoho cenných rad.

Obsah

Kapitola 1

Úvod

Počet zařízení připojených k počítačovým sítím a potažmo i internetu se neustále zvyšuje. Kybernetických útoků přibývá. Každé z těchto zařízení může být potenciální bezpečnostní hrozbou pro celou síť. Správnou konfigurací sítě a zařízení lze bezpečnostní riziko v jistých ohledech výrazně snížit, ovšem dodržování těchto bezpečnostních praktik je v řadě případů spíše výjimkou než pravidlem. Na samotné zabezpečení sítě a zařízení na ni přítomných se tedy spoléhat nemůžeme. Můžeme ovšem síť sledovat, všímat si podezřelého chování a tímto včas detekovat pokusy o útoky a zamezit jim, případně omezit škody u již úspěšných útoků.

Nemůžeme ovšem zaznamenávat veškerá data procházející sítí, a to jak z důvodů technických, tak z důvodů etických a legálních. Jen paměťové nároky pro uchovávání všech dat v podobě jaké prošly sítí by byly neudržitelné. Navíc stále větší množství protokolů přenáší data v šifrované podobě. Při monitorování sítí tedy typicky zaznamenáváme pouze určité vybrané informace.

Pro tyto účely se v dnešní době nejčastěji využívají systémy založené na protokolech NetFlow a IPFIX. Na různých částech monitorované sítě jsou umístěna zařízení (tzv. exportéry) sbírající informace o komunikaci mezi zařízeními, které pak odesílá do centrálního místa (tzv. kolektor) pro další zpracování. Funkci exportéru může zastávat přímo směrovač (router) nebo speciální zařízení na to určené tzv. sonda (probe).

Mezi informace, které takto zaznamenáváme, typicky patří např. zdrojová a cílová IP adresa a port, počet přenesených bajtů, protokol, čas začátku a konce spojení, a jiné pro protokol specifické informace. Přestože se těchto informací může zdát relativně málo oproti množství skutečně přenesených dat, vzhledem k počtu uskutečňovaných spojení je to stále obrovské množství dat. Analýzou i těchto zdánlivě základních informací toho ovšem o síti můžeme mnoho zjistit a mnoho útoků odhalit již ve stádiu pokusu.

Velice důležitou součásti zpracování a analýzy těchto dat je jejich filtrování. Kriterií, na základě kterých můžeme chtít tato data filtrovat je celá řada. Může se jednat o něco jednoduchého, jako je porovnání IP adresy záznamu s jinou zadanou IP adresou pro vybrání konkrétního zařízení nebo porovnání portu pro vybrání konkrétní služby, nebo také o něco složitějšího, jako je test zda se IP adresa nachází v blacklistu sestávajícího z tisíců IP adresa. Typicky také budeme chtít filtrovat na základě vícero kritérií zároveň.

Možných kritérií a případů užití je mnoho a všechny je předem předpovídat nelze, ideálně by tedy mělo být výsledné řešení také snadno rozšiřitelné, a vzhledem k velkému množství dat musí být zároveň dostatečně efektivní.

Obdobnou problematikou jako filtrování je profilování. Zatímco u filtrování vybíráme pouze ty datové záznamy, které vyhovují stanoveným kritériím, u profilování se jedná spíše o třídění všech datových záznamů do různých kategorií na základě zadaných kritérií. Hlavní

myšlenka je ovšem prakticky stejná, a i na filtrování lze pohlížet jako na formu profilování nebo-li třídění záznamů do kategorií, jen s tím rozdílem že v tomto případě pouze do dvou kategorií: vyhovující a nevyhovující. Ideálně by se tedy výsledné řešení pro problematiku filtrování mělo dát z velké části použít i pro problematiku profilování. Cílem této práce je takové řešení navrhnout a implementovat.

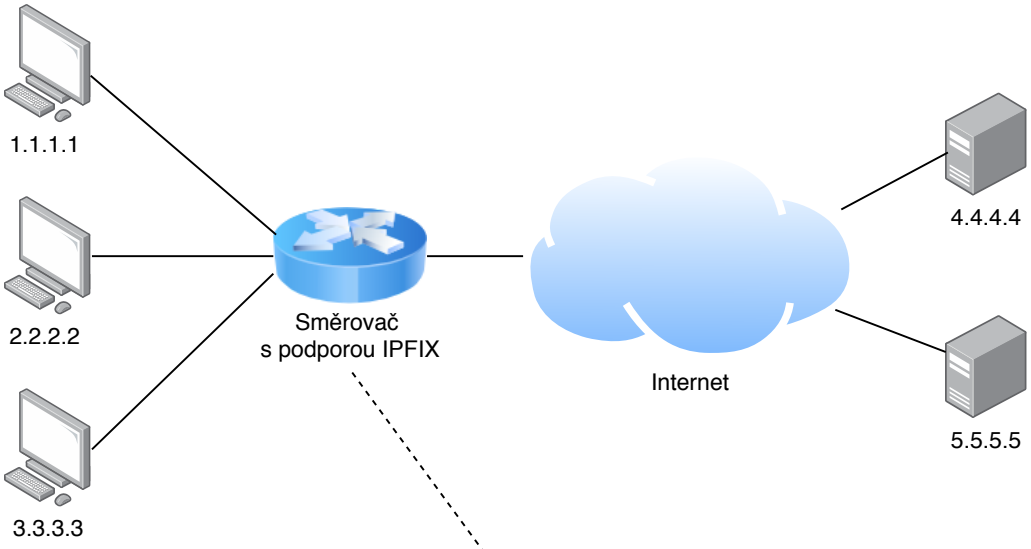
Kapitola 2

Monitorování síťových toků

Systém IPFIX je v dnešní době jeden z nejrozšířenějších přístupů monitorování sítě. Byl vytvořen organizací IETF jako otevřená alternativa systému NetFlow[3] od společnosti Cisco, ze kterého také vychází a je prakticky jeho nástupcem. Na provoz na sítí pohlíží jako na toky procházející prvky sítě. Na vybraných částech sítě můžeme umístit prvky, které o jednotlivých tocích, které jimi procházejí, zaznamenávají různé informace. Tyto prvky nazýváme exportéry. Sesbírané informace jsou poté odesílány do centrálního bodu nazývaného kolektor, kde jsou shromažďovány a případně analyzovány. Způsob přenosu těchto informací a jejich formát definuje stejnojmenný protokol IPFIX. Popis protokolu IPFIX v této sekci čerpá z RFC 7011[6] a RFC 7012[5] specifikující tento protokol a dále z RFC 6313[4] rozšiřující tento protokol o strukturované datové typy.

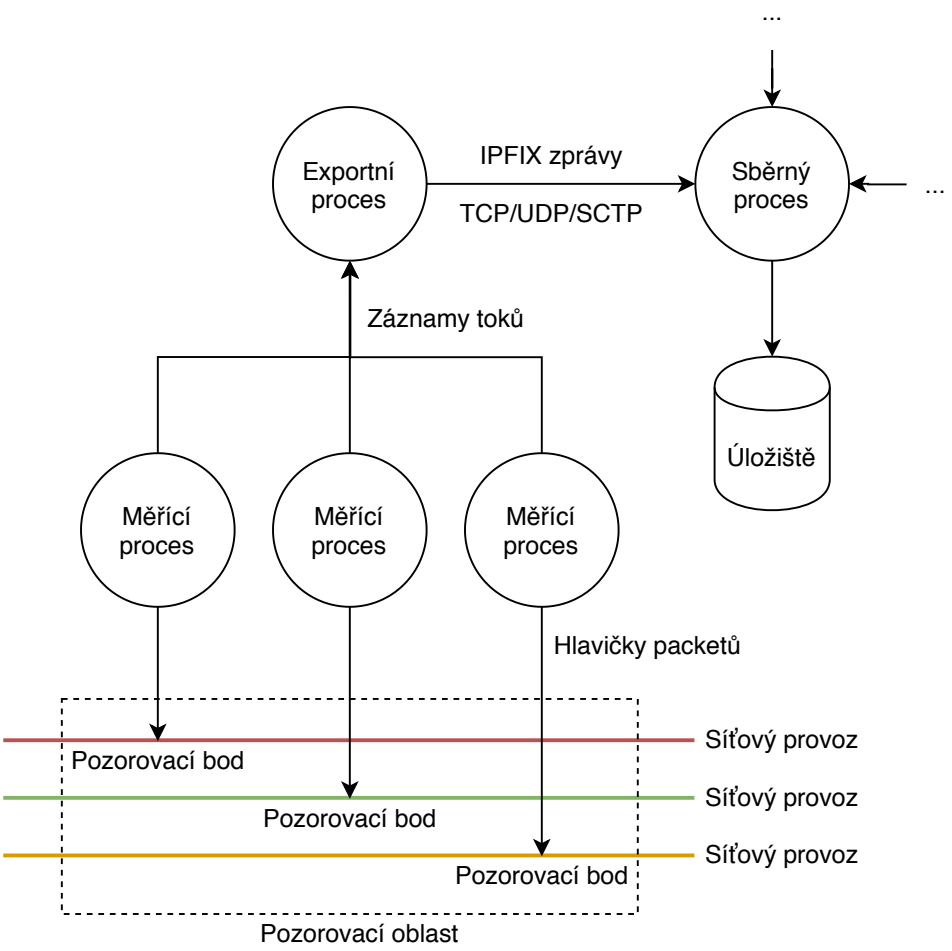
2.1 IP toky

Provoz na síti na úrovni IP vrstvy sestává z paketů plynoucích ze zdroje do cíle. Je zřejmé, že při komunikaci dvou zařízení bude přeneseno větší množství paketů, které spolu všechny souvisí, a uchovávat zvlášť informace o každém paketu je nepraktické. Proto se pro účely měření pakety agregují do tzv. IP toků na základě jejich charakteristických vlastností. Touto charakteristickou vlastností jsou obvykle hodnoty klíčových položek, u IP toků tedy typicky zdrojová a cílová IP adresa, zdrojový a cílový port a transportního protokol. Mimo klíčové položky jsou tu dále informace, které u toku měříme, jako např. počet paketů spadajících do daného toku, počet přenesených bajtů, které TCP příznaky byly v rámci spojení použity, čas vzniknutí a zaniknutí toku a jiné. Záznam o toku vzniká příchodem prvního paketu s unikátními hodnotami těchto položek na měřící prvek, který si vede tabulku aktivních toků. Následující pakety spadající hodnotami svých klíčových položek pod tento tok budou do záznamu tohoto toku zahrnuty. Tok zaniká po tom co po nějakou dobu nepřišel žádný paket spadající do daného toku nebo v případě TCP spojení jejím ukončením, případně také může nastat i v důsledku časového či paměťového limitu pro jeden tok v rámci monitorovacího zařízení. Tok po jeho zaniknutí již není dále aktivn, a při přijetí paketu se stejnými hodnotami klíčových položek se vytváří zcela nový záznam o toku v tabulce aktivních toků nezávislý na předešlém toku. Znázornění tohoto procesu můžeme vidět na obrázku 2.1.

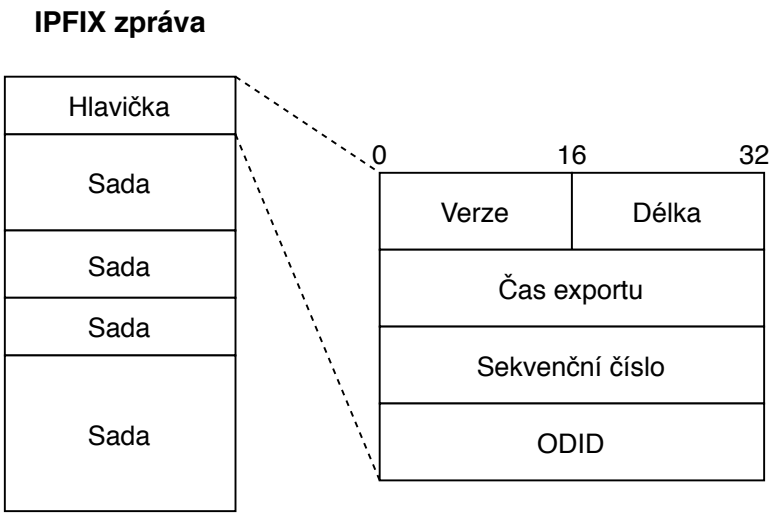


Zdrojová IP	Cílová IP	Protokol	Zdrojový port	Cílový port	Vstupní rozhraní	Výstupní rozhraní	Počet packetů
1.1.1.1	4.4.4.4	TCP	22091	443	1	4	2005
2.2.2.2	4.4.4.4	TCP	22078	80	2	4	1332
3.3.3.3	4.4.4.4	TCP	22132	22	3	4	20112
1.1.1.1	5.5.5.5	UDP	22097	554	1	4	31223

Obrázek 2.1: Měření IP toků v síti



Obrázek 2.2: Znázornění architektury systému IPFIX



Obrázek 2.3: Struktura IPFIX zprávy a její hlavičky

2.2 Architektura IPFIX

Monitorování začíná na tzv. pozorovacích bodech (Obeservation Point), což je místo kde můžeme pozorovat průchod paketů; např. port směrovače. Dále je zde tzv. měřící proces (Metering Process), který dostává na vstupu data z paketů z pozorovacích bodů, obvykle jejich hlavičky, a na základě informací z nich vytváří záznamy o tocích. Záznamy o tocích jsou poté po jejich expiraci předány tzv. exportnímu procesu (Export Process), který z nich vytváří IPFIX zprávy a odesílá je tzv. sběrným procesům (Collecting Process). Sběrný proces typicky běží na předem určeném zařízení na síti zvaném kolektor (Collector). Zařízení, na kterých běží exportní proces, nazýváme exportér (Exporter). Exportéry odesílají data na jeden nebo více kolektorů na síti pomocí TCP, UDP nebo SCTP protokolu. Pro identifikaci zdroje dat se používá tzv. ODID (Obeservation Domain ID), neboli identifikátor pozorovací oblasti. Pozorovací oblast je skupina jednoho nebo více pozorovacích bodů. Data jsou poté kolektorem typicky roztříděna a zapsána do úložiště pro pozdější zpracování. Ilustraci architektury popsané v této sekci můžeme vidět na obrázku 2.2.

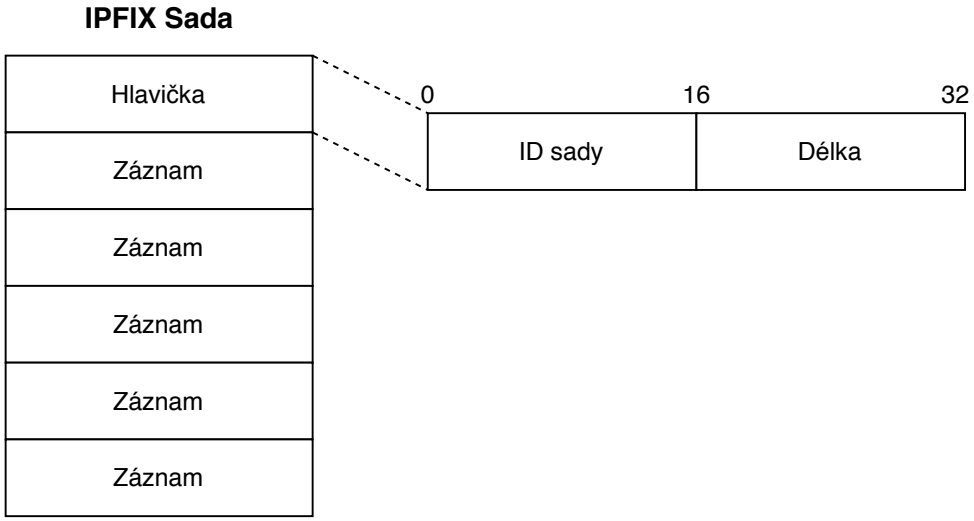
2.3 Protokol IPFIX

Protokol IPFIX využívá pro přenos dat z exportéru na kolektor jeden z transportních protokolů TCP, UDP, nebo SCTP. Jedná se o tzv. ”push protokol”, což znamená, že jsou data periodicky odesílána od odesílatele (exportéru) k příjemci (kolektoru) bez jakékoliv interakce ze strany příjemce. Data jsou přenášena v podobě IPFIX zpráv, jejichž strukturu si popíšeme dále. U všech hodnot v této sekci se uvažuje formát big endian (network byte order) a bez-znaménková varianta číselného typu, pokud není řečeno jinak.

IPFIX zpráva

Znázornění IPFIX zprávy můžeme vidět na obrázku 2.3. IPFIX zpráva se skládá z hlavičky a žádné nebo více sad obsahujících data různého charakteru v závislosti na typu sady. Hlavička obsahuje následující položky:

1. **Verze** – Verze protokolu této zprávy. Pro IPFIX je to hodnota 10 jako následník NetFlow verze 9.
2. **Délka** – Celková délka zprávy v bajtech včetně hlavičky.
3. **Čas exportu** – Čas odeslání zprávy z exportéru, vyjádřen v podobě UNIX časové značky (počet uplynulých sekund od 1.1.1970 0:00 UTC).
4. **Sekvenční číslo** – Celkový počet odeslaných datových záznamů v rámci jednoho ODID a spojení.
5. **Observation Domain ID (ODID)** – Číselný identifikátor oblasti, kde byla data naměřena. V ramci jednoho exportéru může být více ODID, ale různé exportéry by neměly používat stejné ODID.



Obrázek 2.4: Struktura IPFIX sady a její hlavičky

Za hlavičkou následující sady mohou být tří různých typů: datové (Data Set), šablonové (Templates Set) a speciální (Options Template Set). Každá sada začíná hlavičkou, která se skládá z číselného identifikátoru sady (ID) a její celkové délky v bajtech (včetně hlavičky).

Šablonové sady

Úkolem šablonový sad je popsat, jak vypadá struktura datových záznamů. Aby tento popis nemusel být přenášen zbytečně často a opakovaně, je od hodnot datových záznamů přenášen zvlášť. Šablonové sady jsou pak přenášeny typicky pouze v určitých časových intervalech a značně se tak snižuje velikost přenesených dat.

Šablonové sady mají ID nastaveno na hodnotu 2. Jejich tělo obsahuje posloupnost šablonových záznamů definující strukturu záznamů v datových sadách. Šablonový záznam se skládá z hlavičky obsahující ID šablony a počet datových políček v šabloně, poté následují definice jednotlivých políček. Definice políčka obsahuje pouze číselný identifikátor (Information Element ID) a jeho délku. Význam políčka zjistíme pomocí jeho ID, které odkazuje do tabulky informačních elementů, která bude popsána dále.

Datové sady

Úkolem datových sad je přenášet datové záznamy. Jeden datový záznam představuje zaznamenané informace o jednom IP toku.

U datových sad je hodnota ID větší nebo rovna 256. Tělo datové sady je posloupností datových záznamů, jejíchž strukturu specifikuje šablona se stejným ID jako je ID této sady. Datové záznamy již sestávají jen ze samotných hodnot políček, jejichž význam zjistíme z definice políčka v odpovídající šabloně.

Speciální šablonové sady

Speciální šablonové sady (Options Templates Set) mají ID nastaveno na hodnotu 3. Obsahují šablony definující strukturu datových záznamů podobně jako obyčejné šablonové sady. Tyto datové záznamy na rozdíl od těch obyčejných nepřenášejí informace o jednotlivých tocích, nýbrž různá metadata týkající se měřícího a exportního procesu. Záznam se skládá z posloupnosti tzv. Scope políček a posloupnosti Option políček. Scope políčka definují kontext, ke kterému se data vztahují. Option políčka jsou ony data. Např. můžeme mít Scope definován políčky ODID (ID pozorovací oblasti) a meteringProcessId (ID měřícího procesu), a Option políčka exportedMessageTotalCount (celkový počet exportovaných IPFIX zpráv), exportedFlowRecordTotalCount (celkový počet exportovaných záznamů o tocích), exportedOctetTotalCount (celkový počet exportovaných bajtů). V praxi z toho pak např. zjistíme kolik zpráv, toků a bajtů přišlo z jednotlivých rozhraní na routeru. Hlavička šablony obsahuje ID šablony a celkový počet políček, poté následuje počet Scope políček a počet Option políček a nakonec posloupnost definic Scope políček a posloupnost definic Option políček. Políčka v šabloně jsou definována stejně jako u obyčejných šablon pomocí ID, a případně PEN, odkazujícího do tabulky informačních elementů.

Tabulka informačních elementů

Tabulka informačních elementů obsahuje definice významů různých IPFIX položek. Na tyto definice se pak odkazují pomocí ID informačního elementu šablony. Základní tabulku, která se v protokolu IPFIX používá jako výchozí, je tabulka spravovaná organizací IANA. Tabulek může být ovšem více, což dovoluje různým subjektům rozšiřovat tento systém o další položky dle jejich potřeb nezávisle na sobě. V tom případě se kromě ID informačního elementu uvádí i hodnota PEN (Private Enterprise Number), která vybírá tabulku, ve které se má definice položky hledat. Seznam těchto dalších tabulek také spravuje organizace IANA.

Obsah těchto tabulek se protokolem IPFIX nijak nepřenáší, a tedy se předpokládá, že už je prvkům IPFIX systému předem znám. Nepředpokládá se, že by se tyto tabulky měnily nějak často, a proto by byl jejich neustálý přenos IPFIX protokolem zbytečným vytěžováním sítě. Definice položky v této tabulce obsahuje jméno položky, její datový typ, sémantický význam (např. počítadlo, identifikátor, příznaky), popis, jednotku (počet bajtů, paketů, milisekund, ...), slovní popis, případně číselný rozsah a další dodatečné informace.

Datové typy

Každé z políček datových záznamů je jednoho z následujících datových typů:

- **Celočíselné typy** – bez-znaménková (unsigned8, unsigned16, unsigned32, unsigned64) a znaménková (signed8, signed16, signed32, signed64). Jsou uložena v tzv. network

byte order (big endian, nejvýznamnější bajt je na nejnižší adrese), znaménkové varianty jsou reprezentovány pomoci dvojkového doplňku (two's complement).

- **Adresové typy** – macAddress, ipv4Address, ipv6Address. Jsou zakódovány jako celočíselné datové typy odpovídající velikosti v network byte order.
- **Typy s plovoucí řádovou čárkou** – float32 a float64. Jsou zakódovány do binární podoby podle standardu IEEE 754 pro odpovídající počet bitů a uložena v network byte order.
- **Booleovská hodnota** – boolean, reprezentována jako jeden bajt s hodnotou 1 pro true a 2 pro false. Ostatní hodnoty jsou nedefinované.
- **Řetězec, pole bajtů** – string, octetArray. Text řetězce je zakódován ve formátu UTF-8 a uložen jako posloupnost bajtů. Pole bajtů je posloupnost bajtů s blíže nespecifikovaným významem, jejich význam určuje až definice políčka v tabulce informačních elementů. Tento datový typ bude ve většině případů proměnlivé délky.
- **Datum a čas** – dateTimeSeconds, dateTimeMilliseconds, dateTimeMicroseconds, dateTimeNanoseconds. Typ dateTimeSeconds je uložen jako 32-bitová celočíselná hodnota bez znaménka, čas je reprezentován jako uplynulý počet sekund od 1.1.1970 0:00 UTC (Unixová epocha). Typ dateTimeMilliseconds je obdobný, jen se jedná o 64-bitovou hodnotu, která vyjadřuje počet milisekund namísto sekund. Typy dateTimeMicroseconds a dateTimeNanoseconds jsou uložena jako 64-bitová hodnota, která je rozdělena na dvě 32-bitové bez-znaménkové hodnoty v network byte order. První z těchto hodnot udává počet sekund uplynulých od 1.1.1900 0:00 UTC (NTP epocha), druhá hodnota je počet $\frac{1}{2^{32}}$ sekund (zhruba 233 pikosekund) od NTP epochy. Oba tyto typy jsou uloženy ve stejné podobě, liší se jen jejich přesnost.

Proměnlivá velikost u datových typů

Použití proměnlivé velikosti u datových typů je indikováno nastavením hodnoty velikosti u definice políčka v šabloně na hodnotu 65535 (maximální možnou na 16 bitech). Poté je velikost uložena přímo u hodnoty a to jedním z následujících způsobů:

- pokud je velikost hodnoty menší než 255, je uložena v prvním bajtu
- pokud je velikost hodnoty >= 255, má první bajt hodnotu 255, a velikost je uložena v následujících dvou bajtech jako 16-bitové číslo

Velikostí hodnoty se myslí počet bajtů, na kterých je hodnota uložena, nepočítaje tyto první 1 nebo 3 bajty. Proměnlivou velikost může teoreticky používat jakýkoliv datový typ, ale prakticky dává smysl pouze u typů jako string nebo octetArray.

Zkracování datových typů

Standard IPFIX podporuje přenášet hodnoty určitých datových typů na menším počtu bajtů, než by celý rozsah daného datového typu vyžadoval. Provádí se to u položek, u kterých je předem známo že jejich hodnota bude vždy v určitých mezích, a nikdy by celý rozsah datového typu nevyužila. Tímto můžeme snížit síťový provoz mezi exportérem a kolektorem.

Zkracování lze provést na datových typech unsigned64, signed64, unsigned32, signed32, unsigned16, signed16 a float64. Celočíselné datové typy mohou být zkráceny na jakýkoliv

počet bajtů menší než jejich skutečná velikost. Typ plovoucí čárky float64 může být zkrácen na jeho 4-bajtovou variantu float32.

Toto zkracování nemůže být aplikováno na datových typech, u kterých je předpokládána určitá pevná délka (např. ipv4Address, 4 bajty) nebo vnitřní struktura (např. dateTimeMicroseconds, skládá se ze dvou částí). Zkrácení datového typu se označuje v definici políčka v šabloně udáním daného počtu bajtů, na který je typ zkrácen. Velikost datového typu v tabulce definic informačních elementů je vždy největší možný.

Rozšířené datové typy

Dokument RFC 6313[4] rozšiřuje protokol IPFIX o podporu strukturovaných datových typů. Jejich strukturou a kódováním se zabývá tato sekce.

Jednoduchý seznam – basicList

Abstraktní datový typ basicList je seznam nula nebo více prvků stejného datového typu. Datový typ musí být jeden ze základních typů definovaných standardem IPFIX. Tento datový typ je zakódován následovně:

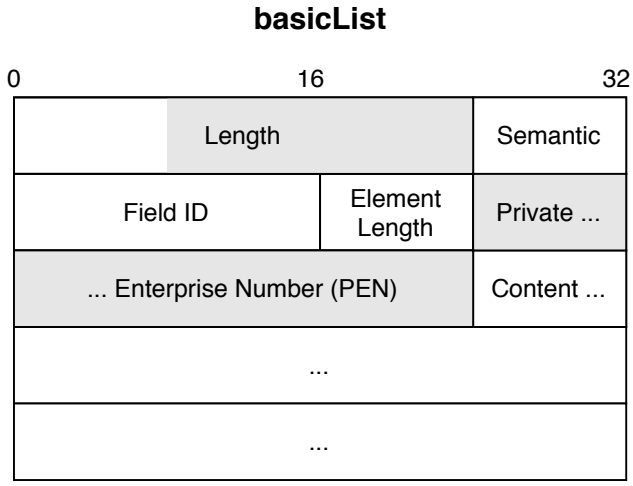
1. Length – 1 až 3 bajty – délka celé této struktury v bajtech včetně hlavičky, pokud je délka méně než 255 je uložena jako jeden bajt, v opačném případě je hodnota prvního bajtu 255 a délka je uložena na následujících dvou bajtech
2. Semantic – 1 bajt – sémantický význam tohoto seznamu pro potřeby kolektoru, má pouze informativní význam a na dekódování této struktury nemá žádný vliv
3. Field ID – 2 bajty – odkazuje do tabulky informačních elementů na definici typu prvků tohoto seznamu
4. Element Length – 1 bajt – velikost jednoho prvku v bajtech, případně 255 pokud jde o prvky proměnlivé délky viz. 2.3
5. Private Enterprise Number – 0 nebo 4 bajty – v případě, že je nejvíce významný bit Field ID nastaven na 0 (nevyužívá se výchozí tabulka informačních elementů) vybírá tabulku informačních elementů
6. Content – obsah samotný, posloupnost po sobě jdoucích prvků seznamu

Šablonový seznam – subTemplateList

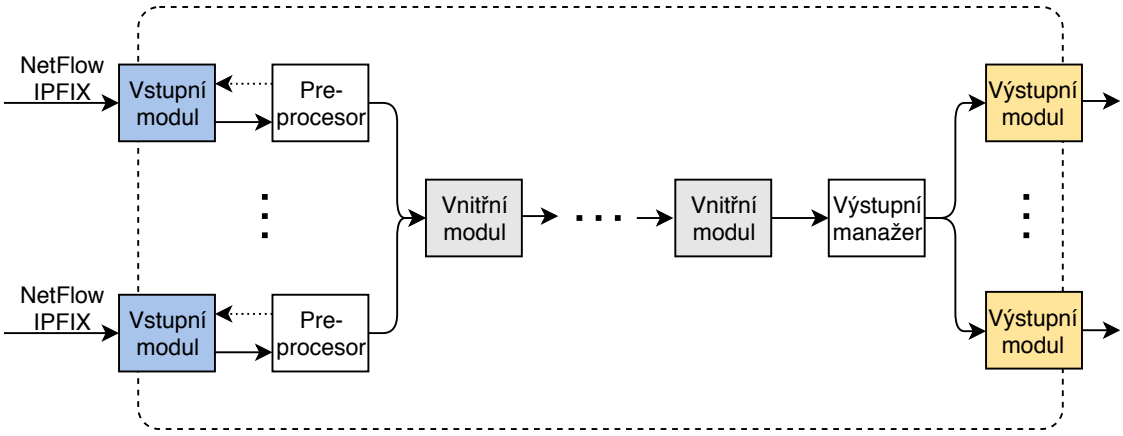
Abstraktní datový typ subTemplateList je seznam nula nebo více prvků stejného strukturovaného datového typu. Strukturovaný datový typ prvků tohoto seznamu je popsán šablonou. Hlavička seznamu je stejného formátu jako u basicList, jediným rozdílem je záměna Field ID odkazující na definici informačního elementu za Template ID odkazující na definici šablony.

Mnohošablonový seznam – subTemplateMultiList

Abstraktní datový typ subTemplateMultiList je seznam nula nebo více prvků strukturovaných datových typů. Na rozdíl od subTemplateList nemusí být všechny prvky seznamu



Obrázek 2.5: Znázornění struktury abstraktního datového typu basicList



Obrázek 2.6: Systém pluginů IPFIXcol2[8]

jednotného typu, ale každý prvek může být jiného typu. Strukturu tohoto datového typu si zjednodušeně můžeme představit jako několik subTemplateList v řadě za sebou, kde každý z těchto podseznamů obsahuje prvky jednoho datového typu.

2.4 Kolektor IPFIXcol2

IPFIXcol2 je druhá generace NetFlow/IPFIX kolektoru vyvíjena sdružením CESNET, který vznikl jako diplomová práce Lukáše Hutáka[8]. Kolektor byl navrhnut s důrazem na flexibilitu, modularitu a vysoký výkon.

Systém pluginů

Pluginy neboli zásuvné moduly jsou nedílnou součástí kolektoru IPFIXcol2. Samotné jádro kolektoru je v podstatě pouhý správce pluginů, který zajišťuje jejich chod a vzájemnou komunikaci. Veškerá funkcionalita týkající se příjmu, zpracování a ukládání IPFIX dat je řešena právě v nich. IPFIXcol2 má tři typy pluginů: vstupní (input), průchozí (intermediate, také označované jako vnitřní), a výstupní (output). Data vznikají ve vstupních pluginech,

následně procházejí průchozími (vnitřními) pluginy, a nakonec jsou předána výstupním pluginům. Tento proces je znázorněn na obrázku 2.6.

Data a ostatní informace jsou mezi pluginy předávány pomocí tzv. kolektorových zpráv (`ipx_msg`). Momentálně pluginy zpracovávají zprávy jednoho ze dvou typů, a to IPFIX zprávy (`IPX_MSG_IPFIX`) a zprávy sezení (`IPX_MSG_SESSION`). Kolektorové zprávy typu IPFIX, jak je již patrné z názvu, zaobalují data IPFIX zpráv přicházejících z exportérů. Zprávy sezení oznamují otevření či naopak uzavření spojení s exportérem. Zprávy typu IPFIX vznikají právě ve vstupních pluginech, které zajišťují příjem dat z exportérů. Data jsou poté zpracována, zaobalena do kolektorových zpráv a předána do další vrstvy pluginů pomocí volání funkce `ipx_ctx_msg_pass`. Data v podobě kolektorových zpráv tedy následně přicházejí do pluginů průchozího typu (intermediate), ty je typicky nějakým způsobem upraví a předají opět zaobalené do podoby kolektorové zprávy další vrstvě, případně je mohou úplně zahodit. Nakonec data končí ve výstupních pluginech, kde je typicky zajištěno jejich uložení, případně nějaký výpis či počítaní statistik nebo i přeposlání.

Implementace pluginů

Pluginy pro kolektor IPFIXcol2 se vytvářejí v podobě sdílených knihoven, a jsou po umístění do speciální složky načteny při jeho startu. Knihovna musí obsahovat speciální strukturu pro definici modulu, a několik speciálních funkcí, pomocí niž probíhá komunikace kolektoru s modulem. Kolektor je implementován v programovacím jazyce C a tedy i jeho moduly musí implementovat rozhraní jazyka C, rovněž poskytuje API funkce pro práci s vnitřními strukturami kolektoru.

Každý plugin musí obsahovat tyto klíčové prvky:

- Globalní proměnnou `ipx_plugin_info` typu `struct ipx_plugin_info`
- Definici funkce `int ipx_plugin_init(ipx_ctx *ctx, const char *params)`
- Definici funkce `void ipx_plugin_destroy(ipx_ctx *ctx, void *data)`
- *Pro vstupní pluginy:*
  Definici funkce `int ipx_plugin_get(ipx_ctx_t *ctx, void *data)`
- *Pro průchozí a výstupní pluginy:*
  Definici funkce
  `int ipx_plugin_process(ipx_ctx_t *ctx, void *data, ipx_msg_t *msg)`

Globální proměnná `ipx_plugin_info` obsahuje informace o pluginu jako jeho název, popis, typ (vstupní, průchozí, výstupní), verzi, minimální verzi API kolektoru, a případné další příznaky. Může vypadat např. následovně:

```
IPX_API struct ipx_plugin_info ipx_plugin_info = {
    .type = IPX_PT_OUTPUT,
    .name = "dummy",
    .dsc = "Example output plugin.",
    .flags = 0,
    .version = "2.0.0",
    .ipx_min = "2.0.0"
};
```

Položka `type` může nabývat jednu z hodnot `IPX_PT_INPUT`, `IPX_PT_INTERMEDIATE` nebo `IPX_PT_OUTPUT`. Jméno pluginu je zároveň jeho unikátním identifikátorem, pomocí kterého se na daný plugin odkazuje i v konfiguraci kolektoru. Popis pluginu, reprezentovaný položkou `dsc`, by měl být jednoduchý slovní popis pluginu, který slouží pouze pro informativní účely uživatele. Položka `ipx_min` udává minimální verzi kolektoru IPFIXcol2 nutnou pro běh pluginu. Položka `flags` je v době psaní tohoto textu rezervovanou položkou bez žádného speciálního významu.

Funkce `ipx_plugin_init` je volána jádrem kolektoru při startu instance pluginu. Tato funkce typicky obstarává zpracování konfiguračního souboru pluginu, tvorbu různých datových struktur nutných pro chod pluginu, případnou registraci rozšíření a další konfiguraci chování kolektoru pro plugin. Je důležité podotknout, že instancí pluginu se typicky spouští několik důsledkem vícevláknového zpracování kolektorem, a proto v naprosté většině případů není žádoucí využívat globální proměnné pro uchovávání stavových dat pluginu. API kolektoru poskytuje funkci `void ipx_ctx_private_set(ipx_ctx_t *ctx, void *data)`, pomocí niž lze v rámci kontextu právě běžící instance pluginu uložit data, která jsou poté jádrem kolektoru předávána do ostatních funkcí rozhraní pluginu ve speciálním parametru výše označeném jako `void *data`.

Při ukončení běhu instance pluginu je volána funkce `ipx_plugin_destroy`. Funkce by měla dokončit chod pluginu a uvolnit všechny používané zdroje.

Nyní se dostáváme ke dvojici klíčových funkcí `ipx_plugin_get` a `ipx_plugin_process`, které budou tvořit samotnou funkcionalitu pluginu. Funkce `ipx_plugin_get` musí být implementována všemi vstupními pluginy, obdobně funkci `ipx_plugin_process` musí implementovat všechny průchozí a výstupní pluginy. Funkce `ipx_plugin_get` je volána jádrem kolektoru pro získání dat v podobě IPFIX (případně NetFlow) zpráv ze vstupního pluginu. Pro představu se může jednat např. o vstupní UDP plugin, který data přijme ze socketu. Funkce `ipx_plugin_process` je volána jádrem kolektoru pro zpracování dat v podobě IPFIX zprávy průchozím nebo výstupním pluginem.

Kapitola 3

Návrh a implementace filtrování

Jak již bylo podrobněji popsáno v předchozích kapitolách, z exportéru na kolektor se posílají informace o tocích v podobě IPFIX zpráv. Tyto IPFIX zprávy obsahují datové sady, což jsou skupiny datových záznamů stejného formátu, a každý datový záznam pak obsahuje políčka se samotnými daty. Filtrováním vybíráme pouze takové záznamy, které vyhovují určitým kritériím. Úkolem filtru je testovat tyto datové záznamy vůči zadanému filtračnímu výrazu, a zjistit, zda datový záznam filtračnímu výrazu vyhovuje nebo nevyhovuje. Příkladem jednoduchého filtračního výrazu může být např. `ip 127.0.0.1 and port 80`. Vidíme, že výraz se skládá ze dvou podvýrazů spojených logickou operací `and`. Každý podvýraz pak obsahuje název datového políčka, na které se odkazuje, a hodnotu vůči které se testuje. Nemusí se jednat pouze o porovnání na přesnou shodu nebo neshodu, a může jít i o komplexnější výrazy. Co vše tento filtr podporuje bude shrnuto v následující kapitole.

3.1 Funkce filtru

Základní filtrační operace

Tím hlavním, co musí filtr podporovat je porovnávání políček vůči zadané hodnotě, tedy výrazy jako např. `"ip == 127.0.0.1"` nebo `"bytes < 1024"`. Filtr podporuje porovnávací operátory `==`, `!=`, `<`, `>`, `<=` a `>=`. Specialitou je pak tzv. "implicitní operátor", který dovoluje zapsat výrazy jako např. `"ip 127.0.0.1"`.

Další neméně důležitou funkcí je možnost tyto porovnání jednotlivých položek spojovat dohromady do větších výrazů. K tomu slouží átory `and`, `or` a `not`. Tímto získáváme možnost vytvářet výrazy jako např. `"ip 127.0.0.1 and port 80"`.

Tyto dvě základní funkcionality nám již dovolují sestavovat plnohodnotné filtrační výrazy. Tato omezená množina operací ovšem nemusí být vždy dostačující nebo alespoň nepříliš uživatelsky přívětivá. K sestavování složitějších výrazů by se nám často hodily i další funkce, proto tento filtr zavádí několik dalších funkcí.

Aritmetické a bitové operace

Někdy můžeme chtít jako součást výrazu provést nějaký výpočet. Toto často lze řešit výpočtem hodnoty mimo filtrační výraz a jejím dosazením, pohodlnější je však mít matematickou operaci přímo součástí výrazu. Toto navíc umožňuje zahrnout do matematické operace i samotnou hodnotu políčka, která se s každým vyhodnocením mění, nedovoluje tedy nějaké předpočítání venku. Mimo matematické operace můžeme chtít v rámci výrazu

použít i různé bitové operace, jelikož hodnota políčka není vždy pouze číslo, ale může to být třeba bitová maska. Jsou tedy přidány aritmetické operace sčítání `"+"`, odčítání `"-"`, násobení `"*"`, dělení `"/"`, modulo `"%"` a bitové operace součin `"&"`, součet `"|"`, exkluzivní disjunkce (XOR) `"^"`, negace `"~"`.

Číselné jednotky

Pro jednodušší práci s hodnotami velikostí a času byly zavedeny jednotky u číselných hodnot ve formě přípon. Jsou to přípony kilo `"k"`, mega `"M"`, giga `"G"`, tera `"T"` pro velikosti, a přípony nanosekunda `"ns"`, mikrosekunda `"us"`, milisekunda `"ms"`, sekunda `"s"`, minuta `"m"`, hodina `"h"`, den `"d"` pro čas. Použití ve výrazu pak vypadá jako např. `"bytes > 1024k"` nebo `endTime - startTime > 30s`. Pro hodnoty datumu a času je rovněž podporován zápis pomocí ISO časové značky[12] včetně jejich zkrácených forem, tedy např. `"startTime > 2020-01-30T00:00:00Z"` nebo jen zkáceně `"startTime > 2020-01-30Z"`.

Seznamy

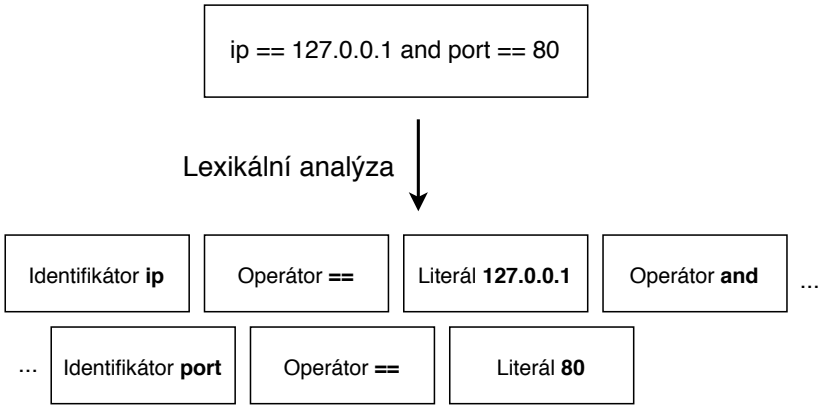
Další významnou vlastností je podpora listů neboli seznamů. Ve výrazu filtru lze zapsat seznamy položek stejného datového typu, poté existuje operátor `"in"`, který otestuje zda se položka v seznamu nachází či nikoliv. U výrazů se často setkáváme s případem, že chceme jedno políčko porovnat vůči několika různým hodnotám, a vznikají pak výrazy jako např. `"port == 21 or port == 22 or port == 80 or port == 443"`, které jsou zbytečně dlouhé, a neustále se opakuje jedna a ta samá věc. Seznamy nám umožňují napsat tento výraz mnohem stručnější a přehlednější formou `"port in [22, 23, 80, 443]"`.

Konstantní identifikátory

Seznamy v podobě literálu ve výrazu filtru nám sice ulehčí práci, ale toho samého by se dalo docílit rozepsáním na několik porovnání spojených operátorem `"or"`. Někdy také můžeme chtít testovat položku vůči mnohem větším seznamů jako je třeba seznam zablokovaných IP adres, který může obsahovat i tisíce prvků. Je zřejmé, že dodávat takový seznam do výrazu filtru v podobě literálu by vedlo k obrovskému výrazu, což je nepraktické jak vzhledem k použitelnosti, tak vzhledem k různým technickým omezením. Proto filtr podporuje tzv. konstanty, které fungují podobě jako identifikátory políček. Narozdíl od identifikátorů políček, jejichž hodnota se s každým záznamem mění, se tyto konstanty vyhodnocují již za překladu filtru, a umožňují do filtru dodat neměnná data i jiným způsobem než jen v podobě literálů zasazených do filtru výrazů v textové podobě.

Předpony identifikátorů

U názvů identifikátorů často nastává situace, že máme dvě téměř totožné názvy které se liší jen tím, že např. jeden z identifikátorů je zdrojový a druhý cílový. Děje se tak u zdrojových a cílových ip adres, zdrojových a cílových portů, vstupních a výstupních rozhraní, atd.. Pro lepší zápis takových identifikátorů mohou být identifikátory i dvojslovné, s tím že prvním slovem je jedno ze speciálních slov `in`, `out`, `src`, `dst`, `ingress`, `egress`. Můžeme tak zapisovat výrazy jako např. `"src port 80"`.



Obrázek 3.1: Proces lexikální analýzy

Vícenásobné hodnoty u identifikátorů

Mezi jednu ze zajímavějších funkcionalit patří ta, že jedna proměnná v rámci výrazu může nabývat několika hodnot zároveň. Filtrovací výraz je pak vyhodnocen pro všechny varianty. Tato vlastnost se může zdát poněkud zvláštní, a dost ovlivnila i výslednou implementaci, pro účely využití filtru je ovšem dosti klíčovou. Dovoluje nám např. zapsat výraz jako `"ip 127.0.0.1"`, kde je identifikátorem `"ip"` myšlena jak zdrojová tak cílová adresa, jak verze 4 tak verze 6. Jde tedy o 4 různá políčka, na které se tento jeden identifikátor mapuje.

Rozšiřování typů a operací

Poslední z tohoto výčtu položek je věc, která původně vůbec nebyla v plánu a dospělo se k ní až postupem času. Zároveň však nejvíce ovlivnila směr, kterým se celý návrh a vývoj filtru ubíral. Touto věcí je možnost libovolně do filtru přidávat nové typy a operace nad nimi pomocí aplikačního rozhraní filtru. V původním návrhu měl filtr pouze omezený výčet datových typů a operací nad nimi, které se nedaly nijak jednoduše rozšiřovat bez větších zásahů do samotné implementace filtru, a to na několika různých místech. Později se však ukázalo, že by tato obecná část filtru mohla najít využití i na různých jiných místech než je jen filtrace IPFIX dat v rámci kolektoru, a každé toto využití by mělo na datové typy a operaci nad nimi různé požadavky a priority. Proto se další klíčovou vlastností filtru stala jeho snadná rozšiřitelnost, čímž se stává mnohem více flexibilní. Toto nám umožňuje např. optimalizovat vyhledávání IP adres v seznamu IP adres transformací seznamu na trii[11] nebo přidání podpory regulárních výrazů bez zásahu do kódu samotného filtru.

3.2 Lexikální analýza

Lexikální analýza je proces převádějící textový řetězec, tedy v našem případě filtrovací výraz, na posloupnost tokenů. Tokeny jsou základní stavební kameny, ze kterých se v další fázi tvoří abstraktní syntaktický strom. Příkladem tokenu může být např. klíčové slovo `and`, číslo, IP adresa, či identifikátor. Komponenta provádějící lexikální analýzu se často také označuje jako *scanner*.

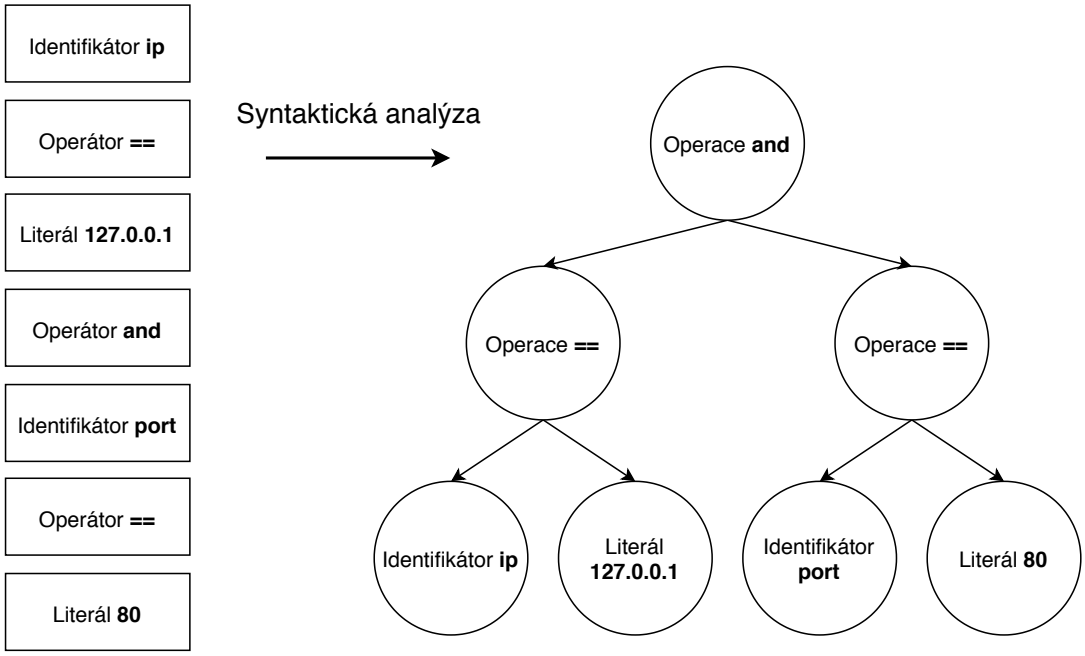
Pro zajištění lexikální analýzy bylo původně v plánu použít nástroje **flex**[1], který z popisu tokenů regulárními výrazy vygeneruje lexikální analyzátor v jazyce C. Hlavní problém tohoto přístupu byl v tom, že flex je poměrně starý nástroj (původní vydání v roce 1987), který nebyl navrhnut s ohledem na některé moderní konvence, a tak např. hojně využívá globální proměnné a preprocesorová makra pro definici chování[10]. V případě našeho filtru není použití globálních proměnných přípustné, primárně protože jde o součást knihovny a může se zpracovávat i několik instancí zároveň. Flex lze nakonfigurovat i tak, že globální proměnné nepoužívá, tato možnost byla ovšem do nástroje přidávána až později a je patrné že se s ní v původním návrhu nepočítalo. Určitě by se dalo chování flexu upravit tak, aby pro potřeby filtru vyhovoval, ovšem nakonec jsem usoudil, že toto řešení přináší více problémů než užitku, a rozhodl se lexikální analýzu implementovat přímo v jazyce C. Toto řešení přináší i další výhody, a to méně závislostí, a lepší chybové hlášky.

Ručně psaný lexikální analyzátor je funkčností inspirovaný nástrojem flex. Nástroj flex postupně zkouší vůči aktuální pozici v textu všechny zadané vzory či regulární výrazy, a výsledkem je takový token, jehož vzor či výraz od aktuální pozice odpovídá co největší části řetězce. Stejně tak ručně psaný analyzátor obsahuje několik funkcí, každá odpovídající určitému typu tokenu, které jsou postupně volány, a výsledkem je token té funkce, která zpracovala co největší množství textu.

Výsledný scanner generuje tokeny následujících typů a podob:

- **Literál** – hodnota zapsaná přímo ve výrazu filtru, má datový typ a hodnotu. Může nabývat datových typů int, float, string, bool, IP address nebo MAC address. Datové typy se většinou snaží svým formátem zachovávat konvence z programovacího jazyka C.
  - **int** – číslo zapsané v desítkové (např. 123), hexadecimální (s předponou 0x, např. 0x12AF), nebo binární (s předponou 0b, např. 0b1101100) podobě.
  - **float** – desetinné číslo. Umožňuje i zkrácený zápis s vynecháním celé nebo desetinné části (např. `1.` místo 1.0 nebo `.1` místo `0.1`) a zápis s exponentem (např. `1.1e5` místo `110000.0`)
  - **string** – řetězec uzavřený v dvojitých uvozovkách (např. `"text"`). Podporuje známé escape sekvence jako \n pro nový řádek, \t pro tabulátor, a také možnost napsat znak pomocí jeho číselné hodnoty v oktálním (např. `\042`) nebo hexadecimálním (např. `\xa2`) tvaru pomocí escapovacích sekvencí známých z jazyka C.
  - **bool** – jsou zapsány pomocí klíčových slov true nebo false.
  - **IP address** – IP adresa verze 4 nebo 6. IP adresy jsou zapsány v obvyklém tvaru (např. 127.0.0.1 nebo `0000:1111:2222:3333:4444:5555:6666:7777`), IP adresa verze 6 podporuje i zkrácený zápis (např. `::f`). IP adresy mohou být volitelně zapsány i s délkou prefixu (např. 127.0.0.1/24 nebo ::ff/64), který je pak při operacích s nimi (např. porovnání) patřičně zohledňován. Pokud není délka prefixu uvedena implicitně se bere maximální možná hodnota (32 u IPv4 adresy, 128 u IPv6).
  - **MAC address** – MAC adresa zapsána v obvyklém tvaru (např. `AA:BB:CC:DD:EE:FF`).

[1] The Fast Lexical Analyzer – scanner generator for lexing in C and C++ (dostupný na https://www.gnu.org/software/flex/)



Obrázek 3.2: Proces syntaktické analýzy

- **Identifikátor** – jednoslovné jméno odkazující se na hodnotu políčka nebo konstanty z venku (např. `ip`, `port`, `tcpFlags`).
- **Symbol** – rezervovaná klíčová slova či operátory, a jiné speciální znaky (např. `+`, `and`, `not`, `exists`, `>=`, `==`, `(`, `[`, ...)

Ukázku tokenů vzniklých zpracováním filtračního výrazu můžeme vidět na obrázku 3.1.

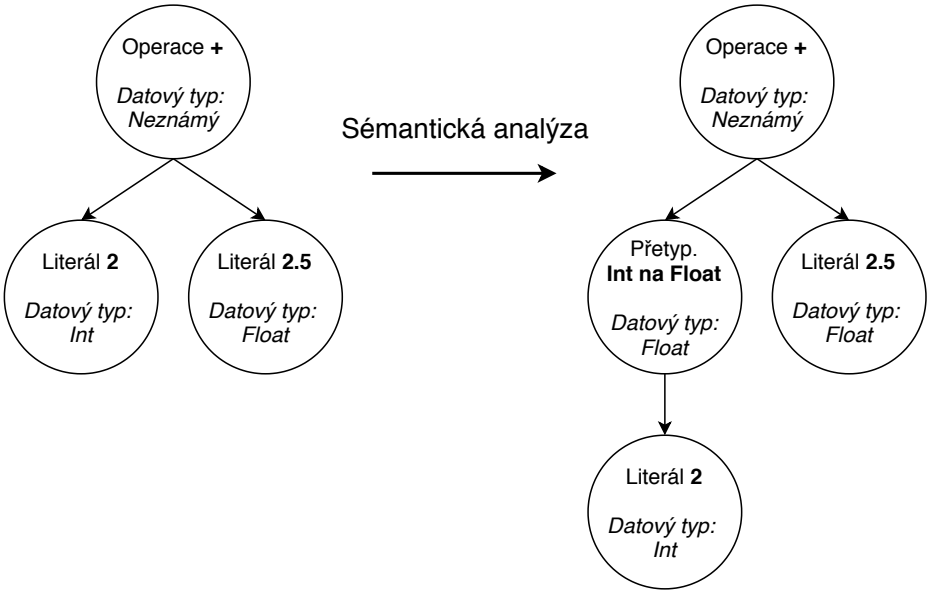
3.3 Syntaktická analýza

Syntaktická analýza neboli parsing je proces, při kterém se z výše zmíněných tokenů tvoří abstraktní syntaktický strom, a to na základě formální gramatiky jazyka. Ukázku tohoto vstupů a výstupů tohoto procesu můžeme vidět na obrázku 3.2.

Tak jako bylo u lexikální analýzy původně v plánu využít nástroj flex, zde bylo v plánu využít nástroje generujícího syntaktické analyzátory jménem **bison**[2]. Tyto dva nástroje se obvykle využívají současně a dosti úzce spolu spolupracují. Nástroj bison obdobně jako flex na základě kódu napsaném v doménově specifickém jazyce tohoto nástroje vygeneruje syntaktický analyzátor v jazyce C. V tomto jazyce zapíšeme gramatiku jazyka pomocí pravidel a akcí, respektive popíšeme jak transformovat tokeny do uzlů abstraktního syntaktického stromu, a jak tyto uzly skládat do stromů, z nichž nakonec vznikne finální strom[10]. Bison ovšem sdílí obdobné problémy jako bylo popisováno u nástroje flex, navíc jak bylo již zmíněno se tyto nástroje obvykle používají současně a dost úzce spolu souvisí, proto jsem i od tohoto nástroje upustil a syntaktickou analýzu napsal ručně.

Jazyk filtru je poměrně jednoduchý. Obsahuje prakticky jen binární infixové (např. 1 + 2) a unární prefixové (např. -1) operace nad identifikátory a literály, a žádné složitější konstrukce se zde nevyskytují. Parser je implementován metodou rekurzivního sestupu s

[2]GNU bison – general-purpose parser generator (dostupný na https://www.gnu.org/software/bison/)



Obrázek 3.3: Proces sémantické analýzy

úpravami pro zpracování operátorů. U klasické metody rekurzivního sestupu je problém správně zpracovávat výrazy s operátory s různou asociativitou a prioritou. Je to důsledkem toho, že u této metody nelze využít pravidla gramatiky obsahující levou rekurzi, což výrazy s levě asociativními operátory jsou. Pravidla s levou rekurzí by potom v implementaci vedly k nekonečnému zanoření. Toto lze určitou transformací pravidel vyřešit použitím pravé rekurze[1], potom ovšem výsledné stromy neodpovídají skutečné podobě výrazů a musely by se dále upravovat, což by také bylo řešením, přináší to však do procesu zpracování dodatečnou složitost a není to příliš elegantní řešení. Dalšími možnostmi by bylo použít úplně jiný způsob parsování, a to nějakou variantu parsování zdola nahoru, kde pravá derivace není problém, ty však nejsou dle mého názoru až tak flexibilní a intuitivní na implementaci jako právě metoda rekurzivního sestupu. Jako řešení jsem tedy zvolil obměnu metody rekurzivního sestupu využívající krom rekurze navic i iteraci a zavedl pro pravidla dodatečný parametr priority. Tato metoda vychází z postupu představeném ve vědeckém článku Keitha Clarka[7] a ukázek fungování tohoto postupu z webové stránky Eli Benderskyho [2].

3.4 Sémantická analýza

Sémantická analýza pracuje s abstraktním syntaktickým stromem sestrojeným během syntaktické analýzy, jejím úkolem je do uzlů stromu doplnit informace o typech. Jediné uzly, které mají typovou informaci na začátku tohoto procesu jsou literály, ostatní musí syntaktická analýza zjistit na základě potomků uzlu. Ukázku tohoto procesu můžeme vidět na obrázku 3.3.

Děje se tak na základě tabulky operací, která obsahuje seznam možných operací a datových typů operandů. Navíc obsahuje také operace přetypování a tzv. konstruktory a destruktory. Tabulka operací, s kterou tento proces pracuje, může být pomocí API filtru rozšiřována, a tím můžeme rozšiřovat filtr o nové typy a operace nad nimi, případně i měnit původní chování. Tato tabulka má podobu jednorozměrného pole záznamů o operacích, kde

Druh operace	Symbol	Arg 1	Arg 2	Výstup	Funkce
Binární	+	Float	Float	Float	AddFloat(Arg1, Arg2, Out)
Binární	-	Float	Float	Float	SubFloat(Arg1, Arg2, Out)
Binární	*	Float	Float	Float	MulFloat(Arg1, Arg2, Out)
Binární	/	Float	Float	Float	DivFloat(Arg1, Arg2, Out)
Unární	-	Float		Float	NegFloat(Arg, Out)
Přetypování		Float		Int	FloatToInt(Arg, Out)
Binární	+	Int	Int	Int	AddInt(Arg1, Arg2, Out)
Binární	-	Int	Int	Int	SubInt(Arg1, Arg2, Out)
Binární	*	Int	Int	Int	MulInt(Arg1, Arg2, Out)
Binární	/	Int	Int	Int	DivInt(Arg1, Arg2, Out)
Binární	%	Int	Int	Int	ModInt(Arg1, Arg2, Out)
Unární	-	Int		Int	NegInt(Arg, Out)
Přetypování		Int		Float	IntToFloat(Arg, Out)

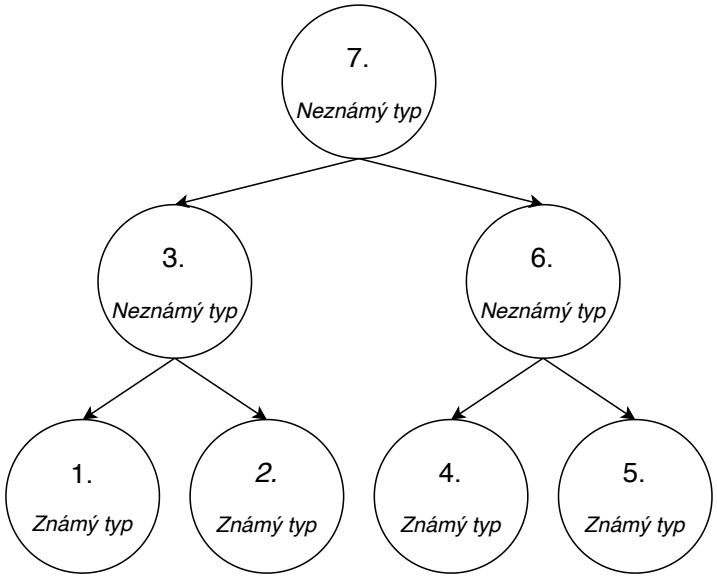
Tabulka 3.1: Ukázka tabulky operací

každý záznam obsahuje druh operace, symbol operátoru, datové typy operandů, výstupní datový typ, a ukazatel na funkci, která tuto operaci vykoná. Druhy operací mohou být:

- **Unární operace** – operace s jedním operandem, např. negace čísla.
- **Binární operace** – operace s dvěma operandy, např. součet čísel.
- **Přetypování** – speciální operace přetypování z jednoho typu na druhý.
- **Konstruktor** – speciální operace, která se chová jako přetypování s vyšší prioritou, které lze provádět jen při sestavování filtru. Bude podrobněji vysvětleno dále.
- **Destruktor** – speciální operace, která se volá při rušení hodnoty daného datového typu pro jejich správnou destrukci a uvolnění.

Část z operací používaných v implementaci filtru pro zpracování aritmetických operací nad číselnými typy můžeme vidět v tabulce 3.1.

Algoritmus prochází abstraktní syntaktický strom způsobem prohledávání do hloubky (viz obrázek 3.4), začíná tedy od listů stromu a postupuje směrem nahoru. Musí tedy platit, že při zjišťování typu pro daný uzel už mají všechny jeho potomci typ nastavený. Listovými uzly mohou být pouze literály nebo identifikátory. Literály již mají typ nastavený z lexikální analýzy, u identifikátorů se volá callbacková funkce pro zjištění datového typu a dalších vlastností, jako je jestli se jedná o konstantní identifikátor nebo proměnné políčko.



Obrázek 3.4: Pořadí průchodu uzlů abstraktního syntaktického stromu v sémantické analýze

Nelistovými uzly jsou operace, u kterých se zjišťuje typ následujícím způsobem:

1. Procházej tabulku operací shora dolů, a najdi operaci odpovídajícího druhu (např. binární) a symbolu (např. +).
2. Zkontroluj, zda typy operandů uzlu sedí s typy operandů operace
3. Pokud sedí, nastav datový typ uzlu na výsledný datový typ operace a konči.
4. Pokud nesedí, procházej tabulku operací a zjisti, zda existuje konstruktor ze stávajícího datového typu do požadovaného datového typu.
5. Pokud existuje, vytvoř pro tento operand uzel konstruktoru, nastav typ uzlu na výsledný datový typ operace a konči.
6. Pokud neexistuje, pokračuj v procházení tabulky operací viz krok 1.
7. Pokud nebyla nalezena vhodná operace, opakuj celý tento proces znova jen s tím rozdílem, že krom konstruktorů zohledňuj i přetypování.

Vidíme, že při hledání vhodné operace děláme dva průchody tabulkou operací. První průchod umožňuje operaci zvolit pouze pokud datové typy sedí, nebo jdou zkonstruovat. Při druhém průchodu můžeme operaci zvolit pokud typy sedí, jdou zkonstruovat, nebo jde požadovaný datový typ získat přetypováním. Toto se děje proto, že nechceme vybrat operaci, která vyžaduje přetypování operandů, když může existovat operace, která přetypování nevyžaduje. Konstruktory se naopak zohledňují už při prvním průchodu, proto byl i dříveji v textu použit výraz "prioritní přetypování". Záměrně totiž chceme, aby měli zkonstruované typy stejnou prioritu jako typy které konstrukci nepotřebují.

3.5 Generování vyhodnocovacího stromu

Celý proces filtrování je rozdělen na dvě hlavní části: sestavovací a vyhodnocovací. Úkolem části sestavování filtru je ze vstupního filtračního výrazu, a případných dalších dodatečných

Instrukce	Popis
`VALUE`	listový uzel reprezentující pouze hodnotu
`AND`	operace logického součinu se zkratovým vyhodnocením
`OR`	operace logického součtu se zkratovým vyhodnocením
`NOT`	neguje hodnotu operandu
`CAST_CALL`	volání funkce pro přetypování operandu
`UNARY_CALL`	volání funkce s jedním operandem
`BINARY_CALL`	volání funkce s dvěma operandy
`DATA_CALL`	volání callbackové funkce pro získání dat proměnné
`ANY`	operace pro vyhodnocení podstromu s proměnnou s vícero hodnotami
`EXISTS`	zjišťuje zda se políčko proměnné ve zpracovávaném záznamu vyskytuje

Tabulka 3.2: Tabulka instrukcí uzlů vyhodnocovacího stromu

dat získaných pomocí callbacků, sestavit datovou strukturu filtru. Část vyhodnocovací pak na základě této datové struktury a poskytnutých vstupních dat vyhodnotí, zda data odpovídají filtračnímu výrazu či nikoliv. Proč je celý tento proces takto rozdělen je asi zřejmé. Výpočetní náročnost sestavovací části není vůbec zanedbatelná, a opakovat ji znovu pro každý datový záznam, který by měl filtrem projít, by bylo obrovským zpomalením, a filtr by byl prakticky nepoužitelný. Cílem je tedy udělat toho co nejvíce během sestavovací části, která bude provedena jednou pro každý výraz, aby toho vyhodnocovací část, která se typicky provede mnohonásobně vícekrát pro jeden výraz, měla co nejméně na práci.

Procesy, které byly doposud popsány, se všechny zabývají částí sestavovací. Posledním z těchto procesů je proces generování a optimalizace, jehož výsledkem bude datová struktura, s níž už bude pracovat část vyhodnocovací. Zde bylo zvažováno několik variant.

Jedna z variant byla reprezentovat filtrační výraz jako posloupnost instrukcí pracujících se zásobníkem. Výhodou této varianty by byla její reprezentace v paměti jako jednorozměrné pole po sobě jdoucích instrukcí. Naopak záležitosti jako zkratové vyhodnocování logických výrazů, nebo znovu-vyhodnocování pouze určitých částí výrazu by se v této reprezentaci řešili poněkud obtížně, a tato přidaná komplexita by pravděpodobně nakonec překonala výhody, které tato reprezentace přináší.

Další možností, a dalo by se říct opačným extrémem, byla varianta vyhodnocovat přímo abstraktní syntaktický strom. Stromová struktura by mnohem zjednodušila proces vyhodnocování. Můžeme vyhodnocovat pouze ty uzly, které odpovídají určitým částem výrazu, které skutečně potřebujeme. Toto se velice hodí nejen pro zkratové vyhodnocování logických operací součtu a součinu, ale především pro implementaci vlastnosti proměnných mít vícero hodnot najednou. Nemusíme potom při změně hodnoty proměnné vyhodnocovat znovu celý strom, ale stačí vyhodnotit ty uzly, které byly změnou proměnné zasaženy, což je ve stromové struktuře snadno zjistitelné. Abstraktní syntaktický strom ovšem obsahuje mnoho dodatečných informací, které jsou ve fázi vyhodnocování již zbytečné. Také není příliš flexibilní. Jakékoliv zásahy do jeho struktury by se projevily i v procesu sémantické syntaktické analýzy, a mohly by být příčinou neočekávaných problémů.

Vznikla proto datová struktura, kterou budeme označovat jako vyhodnocovací strom. Tato struktura kombinuje obě tyto varianty. Jedná se o strom podobný abstraktnímu syntaktickému stromu, jehož uzly obsahují pouze instrukci udávající co má daný uzel vykonat, a případnou další hodnotu, kterou nutně potřebuje k jejímu vykonání.

Uzel vyhodnocovacího stromu nabývá jednu z instrukcí uvedenou v tabulce 3.2

Tento strom se tvoří z abstraktního syntaktického stromu a mapování jeho uzlů na uzly vyhodnocovací je typicky 1:1. Opět se využívá tabulky operací zmíněnou v části syntaktické analýzy, která krom informací o typech operací obsahuje i ukazatele funkcí, které dané operace vykonávají. V této fázi už má každý uzel abstraktního syntaktického stromu doplněný datový typ procesem syntaktické analýzy. Na základě datových typů uzlu, jeho operandů, a symbolu operace je v tabulce operací vyhledán odpovídající záznam o operaci, a ukazatel na funkci vykonávající tuto operace je dosazena do vytvořeného vyhodnocovacího uzlu.

Při vyhodnocování stromu se opět postupuje stylem průchodu do hloubky, s tím že některé instrukce mohou tento postup ovlivňovat. U logických operací součinu a součtu se po vyhodnocení jednoho podstromu zkoumá výsledná hodnota, a až na základě ní se rozhoduje zda se vůbec bude vyhodnocovat i druhý podstrom.

Dále je zde speciální instrukce ANY, která je potřebná pro vyhodnocení podstromů s proměnnou nabývající více hodnot. Operand této instrukce, kterým je podstrom, je vyhodnocován opakovaně pro různé varianty hodnoty proměnné, dokud není výsledkem podstromu pravda, nebo nejsou vyčerpány všechny hodnoty proměnné.

3.6 Optimalizace

Během sémantické analýzy jsou některé uzly označovány za konstantní. U uzlů reprezentující literál můžeme bez dalšího zkoumání prohlásit, že jsou konstantní, jelikož se jedná o hodnotu zapsanou přímo ve výrazu filtru. V případě identifikátorů se volá callback zjišťující datový typ a další příznaky proměnné, jednou z těchto příznaků je i konstantnost. Je-li tedy tento příznak nastaven, můžeme o daném uzlu identifikátoru rovněž prohlásit, že se jedná o konstantu. Dále zbývají nelistové uzly reprezentující různé operace. Zde platí jednoduché pravidlo. Jsou-li všechny operandy operace konstantní, pak nutně musí být i výsledek této operace konstantní. Součástí generování vyhodnocovacího stromu je pak i optimalizace částí stromu, které jsou takto označeny za konstantní. Tato část stromu je rovnou vyhodnocena, a nahrazena uzlem s předpočítanou hodnotou. Výsledkem je tedy stejný strom, jako by výsledek daného konstantní výrazu byl zapsán ve výrazu filtru jako literál.

3.7 Rozhraní pro práci s filtrem

Jak již bylo zmíněno, celý proces filtru je rozdělen na dvě hlavní fáze. Fázi sestavovací, kdy je z textového výrazu sestrojen objekt filtru, a fázi vyhodnocovací, kdy jsou nad objektem filtru testována data. Tyto dvě fáze reflektuje i rozhraní filtru, jehož hlavní část tvoří tyto dvě funkce:

- Funkce `Compile(Výraz, Konfigurace) -> Filtr` pro sestavení filtru s parametry výrazu filtru v textové podobě a konfiguračního objektu
- Funkce `Evaluate(Filtr, Data) -> Boolean` pro vyhodnocení sestaveného filtru nad daty

Vzhledem k tomu, že cílem bylo vytvořit obecný filtr, který bude použit i v rámci různých projektů, krom výrazu se při sestavování filtru dodává i tzv. konfigurační objekt. Tento konfigurační objekt tvoří další formu rozhraní filtru umožňující právě jeho specializaci a rozšiřitelnost, jeho hlavním prvkem jsou tzv. callbacky. Callbacky jsou uživatelem definované funkce předem známých signatur, které jsou předány filtru. Filtr poté v různých

fázích sestavování či vyhodnocování tyto funkce volá pro získávání dalších potřebných dat či informací. Konfigurační objekt filtru obsahuje tři callbacky, které musí uživatel definovat.

- `Lookup callback` – Callback pro získání datového typu a případných dalších příznaků o identifikátoru. Těmito příznaky mohou být např. jestli je proměnná konstantní, jestli se jedná o proměnné políčko, případně jestli toto políčko může nabývat více hodnot najednou.
- `Const callback` – Callback pro získání hodnoty konstantní proměnné.
- `Data callback` – Callback pro získání hodnoty proměnného políčka z právě vyhodnocovaných dat, které jsou callbacku předány ve formě parametru.

Další částí konfiguračního objektu je tabulka operací, která již byla popsána v části zabývající se sémantickou analýzou 3.4. Do této tabulky můžeme libovolně přidávat nové operace i datové typy, a tím rozšiřovat filtr o novou funkcionalitu. Speciálními typy operací jsou pak přetypování a tzv. konstruktory, které umožňují vytvářet hodnoty nových datových typů z hodnot datových typů stávajících. Přetypování funguje obdobně jako u běžných programovacích jazyků, pokud není pro operandy určitých datových typů nalezena vhodná operace, použije se přetypování pro převod jednoho datového typu na druhý tak, aby se již nějaká vhodná operace našla. Tyto konstruktory jsou téměř to stejné jako přetypování, jen s tím rozdílem, že přetypování se provádí jen když je to nutné, ovšem operace konstruktoru se provede i když to nutné není. Pro lepší pochopení si uveďme následující příklad:

- Filtr ve výchozí konfiguraci umí pracovat se seznamy IP adres, a definuje základní operace nad nimi. Jednou z těchto operací je operátor `"in"` pro zjištění, zda se IP adresa vyskytuje v seznamu IP adres.
- Pracuje se s velkým množstvím IP adres a záznamů, a tato operace ze základní sady operací není dostatečně výkonná.
- V tabulce operací se definuje operace `"in"` pro zjištění, zda se IP adresa nachází v trii IP adres.
- Rovněž se definuje konstruktor ze seznamu IP adres na trii IP adres.
- Při zpracování výrazu se nyní prioritně použije operace `"in"` pro trii IP adres, a ze seznamu IP adres se zkonstruuje trie IP adres.

3.8 IPFIX mezivrstva

Doposud byla popisována obecná podoba filtru, která však neobsahuje žádnou logiku pro zpracování dat ve formátu protokolu IPFIX. Pro rozšíření obecného filtru o tuto funkcionalitu vznikla tzv. IPFIX mezivrstva, která obecný filtr obaluje, a dodává mu konfigurační objekt s patřičnými callbacky pro práci s identifikátory a zpracování dat ve formátu protokolu IPFIX. Co tato mezivrstva zajišťuje můžeme rozčlenit na tři části, které si v této sekci popíšeme.

Mapování identifikátorů na IPFIX položky

První věcí, která musí být zajištěna, je mapování identifikátorů z výrazu filtru na odpovídající IPFIX položky. Děje se tak využitím tabulky informačních elementů, jejíž implementace již v rámci libfds existuje. Jelikož jsou názvy IPFIX položek občas příliš dlouhé a nepraktické pro potřeby filtračních výrazů tvořených uživatelem, byla stávající implementace tabulka informačních elementů rozšířena o podporu tzv. aliasů. Aliasy nám dovolují definovat nové, uživatelsky přívětivější názvy, pomocí kterých se můžeme na IPFIX položky odkazovat. Rovněž v rámci jednoho aliasu můžeme obsáhnout i několik najednou. Pro příklad mějme výraz pro filtraci na základě hodnoty zdrojové nebo cílové IP adresy `"sourceIPv4Address 127.0.0.1 or destinationIPv4Address 127.0.0.1"`, po definování aliasu `"ip"` odkazujícího na `"sourceIPv4Address"` a `"destinationIPv4Address"` ten samý filtrační výraz můžeme napsat jako `"ip 127.0.0.1"`.

Mapování hodnot pro IPFIX položky

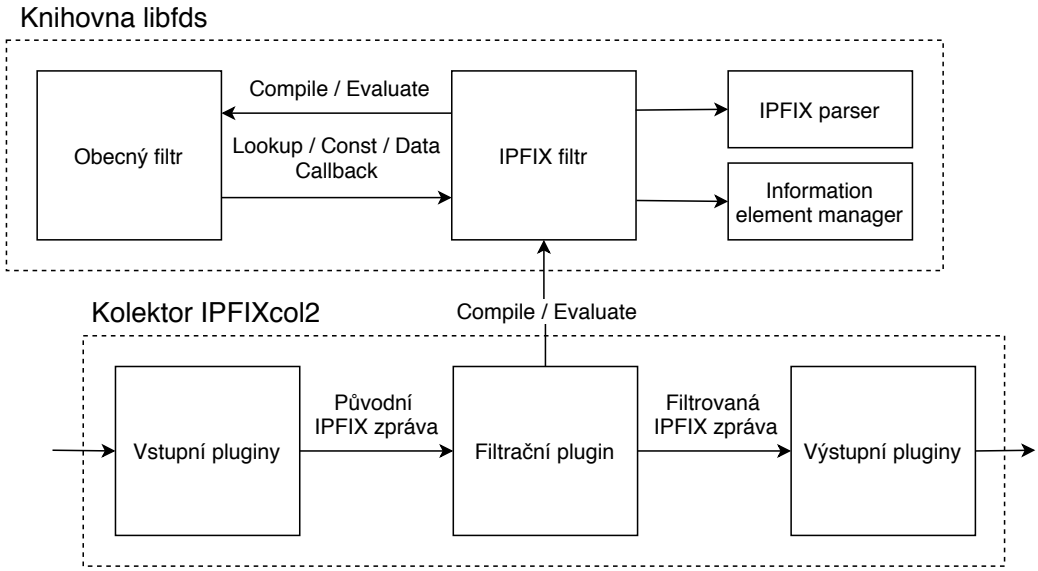
Obdobně jako aliasy pro odkazování se na IPFIX položky vznikly tzv. mappingy pro mapování hodnot k IPFIX položkám. Tyto mappingy nám dovolují vyjádřit hodnotu, vůči které IPFIX položku porovnáváme, pomocí identifikátoru. Zároveň se tento identifikátor vztahuje přímo k dané IPFIX položce s kterou je porovnáván, a tak může mít stejný identifikátor více významů v různých případech použití. Pro příklad, bez použití mappingů by výraz pro filtraci toků s transportním protokolem TCP a nastavenými příznaky SYN a FIN vypadal následovně: `"proto 6 and tcpFlags 1 and tcpFlags 2"`. Je zřejmé, že takový výraz není příliš uživatelsky přívětivý, jelikož obsahuje číselné hodnoty jejichž význam není zřejmý. S využitím mappingů stejný výraz můžeme napsat jako `"proto tcp and tcpFlags syn and tcpFlags fin"`, případně i jako `"proto tcp and tcpFlags syn|fin"`.

Extrakce hodnot IPFIX položek

Poslední záležitostí, kterou tato mezivrstva obstarává je extrakce hodnoty požadovaného políčka z IPFIX záznamu. Filtr při vyhodnocování dostává jako parametr data, o struktuře těchto dat, nebo jak s těmito daty pracovat ovšem nemá ani ponětí. Tuto funkci má na starost data callback, který dostává na vstupu ony data a číselný identifikátor položky, jejíž hodnotu filtr aktuálně potřebuje. Data callback zajistí extrakci dat této položky z datového záznamu, a její převod z datového typu formátu IPFIX do vhodného datového typu filtru. Pro extrakci položky je využíváno funkcí knihovny libfds, které byly již implementované pro potřeby kolektoru IPFIXcol2. Převody typů se ději především pro transformaci hodnot z komplexnější množiny číselných datových typů různých délek a endianit formátu IPFIX na jednodušší množinu číselných datových typů s jednou endianitou a délkou 64 bitů implementovanou v obecném filtru. Implementace všech těchto různých datových typů, které protokol IPFIX poskytuje, přímo v logice filtru se v minulosti ukázalo jako špatně udržitelné řešení.

3.9 Filtrační plugin

Nyní máme k dispozici filtrační komponentu, která dokáže vyhodnocovat a zpracovávat záznamy ve formátu protokolu IPFIX. Zbývá již jen tuto komponentu začlenit do systému kolektoru IPFIXcol2. Jak již bylo popsáno dříve, data chodí na kolektor v podobě IPFIX zpráv, které obsahují datové sady, což jsou skupiny datových záznamů. Tyto IPFIX zprávy



Obrázek 3.5: Výsledná architektura filtrační části

postupně procházejí systémem pluginů různých typů. Jedním z těchto typů jsou tzv. průchozí (intermediate) pluginy, kterými IPFIX zprávy pouze prochází při cestě ze vstupních do výstupních pluginů. Průchozí plugin může během tohoto procesu data i pozměňovat a různě s nimi manipulovat, nebo vytvořit IPFIX zprávu úplně novou. což je přesně to, co je pro implementaci filtrovacího pluginu zapotřebí.

Plugin prochází jednotlivé datové záznamy v datových sadách zprávy, a testuje je pomocí IPFIX filtru popsaného v předchozí části. Z datových záznamů, které filtrem úspěšně projdou, jsou poté skládány nové datové sady, které jsou ve finále zabaleny do nové IPFIX zprávy. Vzhledem k formátu protokolu IPFIX nestačí pouze vynechat nevyhovující datové záznamy z původní zprávy, ale musí se pozměňovat i hlavičky sad i celé zprávy. Původní zpráva, kterou plugin dostal je vstupu, je poté zahozena, a na výstupu pluginu odchází tato nová zpráva obsahující pouze datové záznamy vyhovující zadanému filtračnímu výrazu. Díky tomuto přístupu nemusí být nijak pozměňovány vstupní ani výstupní pluginy. Výstupní plugin dostane již přefiltrovanou IPFIX zprávu v obvyklém formátu, a tedy ani neví, že k nějakému filtrování v předchozích krocích došlo.

3.10 Výsledná architektura

Při návrhu filtru se myslelo na to, že filtr může být využíván nejen jako filtr pro IPFIX položky, ale jako filtr IP toků obecně. Proto je jeho architektura vymyšlena tak, aby jeho použití nebylo závislé na kolektoru IPFIXcol2, ani na protokolu a formátu dat IPFIX. Celý IPFIX filtr pro kolektor se tedy ve výsledku skládá ze tří oddělených částí.

- **Obecný filtr toků** – je součásti knihovny libfds, jedná se pouze o filtrační část která nic neví o kolektoru ani formátu IPFIX
- **IPFIX filtr** – rovněž součástí knihovny libfds, jde o mezivrstvu, která přizpůsobuje obecný filtr pro práci s daty IPFIX

- **Filtrační plugin** – plugin kolektoru IPFIXcol2 průchozího typu, využívá IPFIX filtr z knihovny libfds

Jak spolu tyto jednotlivé části spolupracují je znázorněno na obrázku 3.5.

Kapitola 4

Profilování

Při monitorování sítí typicky chodí na kolektor velké množství záznamů o tocích z mnoha různých zdrojů. Tyto data chceme často třídit do různých kategorií na základě hodnot jejich políček, která odpovídají různým vlastnostem těchto záznamů, jako např. jejich zdrojová síť nebo použitý protokol. Tento proces nazýváme profilování. Na rozdíl od filtrování, které se provádí typicky již na uložených datech tzv. offline, se profilování často odehrává tzv. online, tedy přímo na datech která přicházejí na kolektor ze sítě v reálném čase. Znázornění tohoto procesu lze vidět na obrázku 4.1.

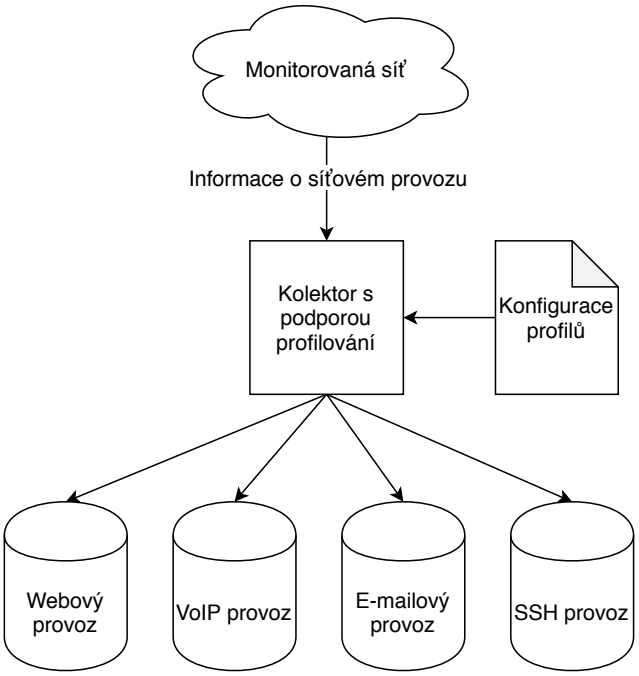
Typickým příkladem užití může být profilování na základě použitého aplikačního protokolu, z čehož pak můžeme např. zjišťovat, které služby představují největší podíl provozu na síti a v konfiguraci sítě pak tento poznatek zohlednit. Neobvyklý nárůst výskytu nějaké služby v síťovém provozu pak může signalizovat útok, např. velký počet neúspěšných přihlášení protokolem SSH zapříčiněný útočníkem snažící se uhádnout přístupové údaje. Tyto základní statistiky můžeme snadno počítat již během profilování, a výsledná roztříděná data pak můžeme snadno analyzovat pro zjištění podrobnějších informací. Zároveň pokud dostáváme data o provozu z více různých sítí můžeme chtít tyto data třídit také na základě jejich původu, tedy např. dle části jejich zdrojové IP adresy nebo identifikátoru exportního zařízení. V tomto případě už je patrné, že pro tyto účely nebude dostačující pouze nějaká jednodimenzionální množina profilů, ale půjde spíše o nějakou hierarchii profilů a jejich podprofilů.

4.1 Profilovací strom

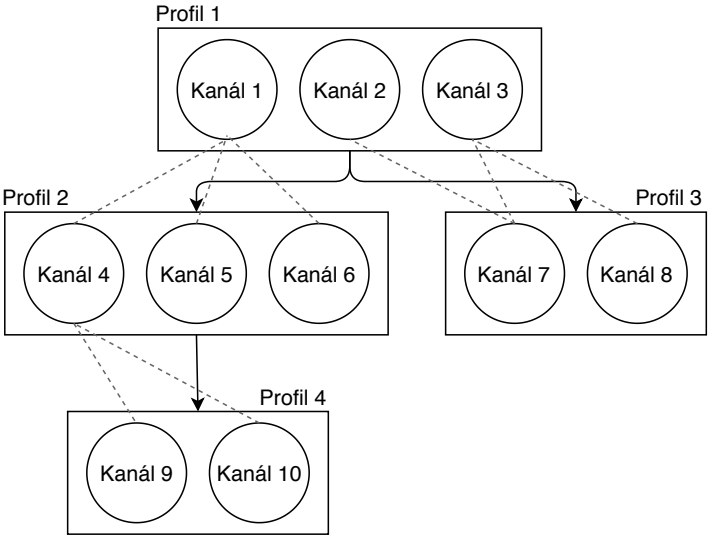
Pro reprezentaci hierarchie profilů a vazeb mezi nimi je použit tzv. profilovací strom. Podoba profilovacího stromu vychází z profilovacího stromu použitého v minulé verzi kolektoru IPFIXcol, především pro zachování jisté míry zpětné kompatibility. Tento profilovací strom vznikl v rámci bakalářské práce o profilování Michala Kozubíka[9] a vychází ze systému použitém v nástroji NfSen[1]. Struktura tohoto stromu je vyobrazena na obrázku 4.2.

Profilovací strom se skládá z profilů, které sdružují tzv. kanály. Každý profil zároveň může obsahovat (a většinou obsahuje) své potomky nazývané podprofily. Každý z těchto podprofilů je rovněž profilem a tudíž může také obsahovat své podprofily. Kanály obsahují filtrační výraz a své zdroje. Těmito zdroji mohou být jakékoliv kanály z otcovského profilu. Speciálním případem je kořenový profil nazývaný ”live”, který otcovský profil nemá a zdroji jeho kanálů jsou veškerá příchozí data.

[1] dostupný na http://nfsen.sourceforge.net/



Obrázek 4.1: Profilování toků kolektorem



Obrázek 4.2: Profilovací strom

Při vyhodnocování je pak daný datový záznam testován vůči filtračním výrazům v jednotlivých kanálech profilu. Pokud výrazu vyhovuje, pak záznam vyhovuje danému kanálu, dále pokud záznam vyhovuje alespoň jednomu z kanálů profilu, pak vyhovuje i danému profilu. Po vyhodnocení záznamu profilem je záznam vyhodnocován jeho podprofily stejným způsobem, jen s tím rozdílem, že se u jednotlivých kanálů také zkoumá, zda záznam vyhovuje alespoň jednomu z jeho zdrojových kanálů. V případě, že záznam nevyhovuje žádným ze zdrojových kanálů daného kanálu, nevyhovuje ani danému kanálu.

4.2 Implementace v kolektoru

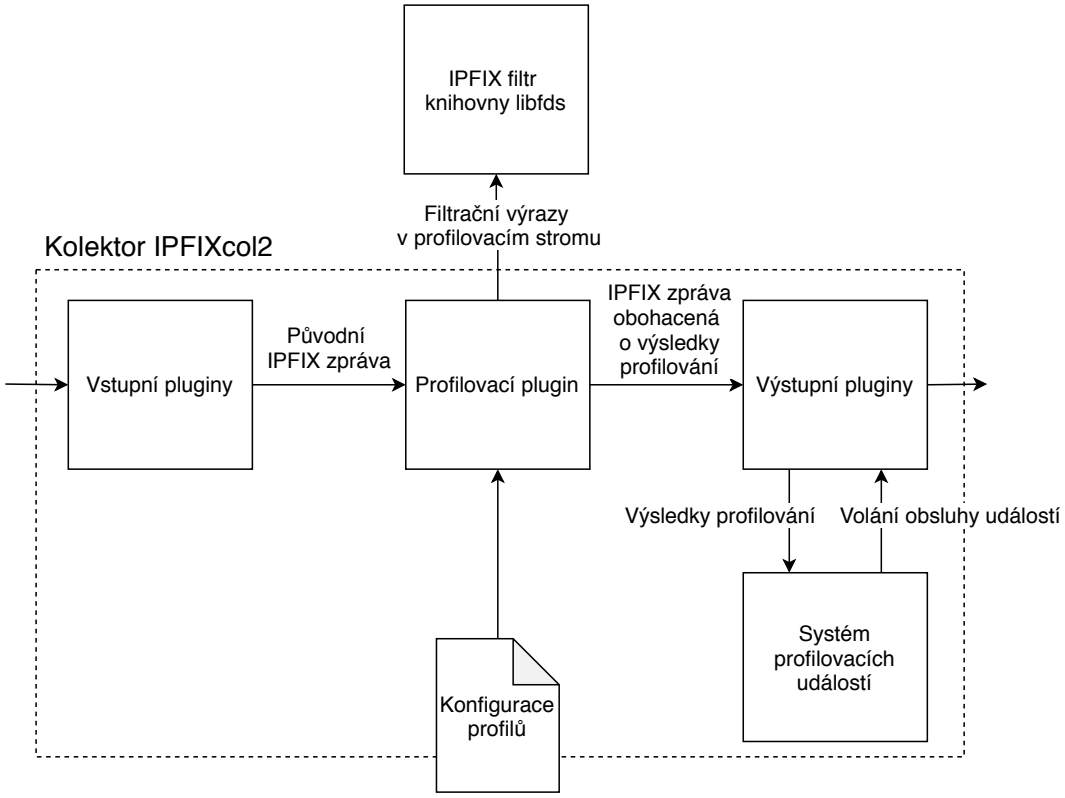
Při implementaci filtrování v rámci kolektoru bylo vcelku zřejmé, že se bude jednat o průchozí plugin, který jednoduše nevyhovující záznamy odfiltruje, vytvoří novou IPFIX zprávu bez těchto záznamů, která bude předána dále do výstupních pluginů, a výstupní pluginy se nebudou muset nikterak upravovat. V případě profilování už je situace poněkud složitější. Datové záznamy budou muset být roztřiďovány na základě profilovacího stromu do odpovídajících profilů a kanálů, a výstupní pluginy budou muset na základě tohoto rozdělení data ukládat např. do odpovídající adresářové struktury. Nyní už si nevystačíme s pouhou úpravou IPFIX zprávy pro výstupní pluginy, a tak bude nutný zásah do výstupních pluginů samotných. První možností by bylo implementovat podporu profilování plně ve výstupních pluginech. Výstupních pluginů je však několik, a další mohou přibývat, a muset opakovat celou tuto implementaci v každém výstupním pluginů zvlášť je přinejmenším nepraktické. Dalším problémem této varianty je výpočetní náročnost. Pokud by každý výstupní plugin prováděl profilování sám, pak v případě, kdy bude použito najednou více výstupních pluginů, bude každý záznam opakovaně vyhodnocován každým z těchto použitých výstupních pluginů. Ideálně bychom tedy chtěli, aby byl zásah nutný k implementaci podpory profilování v rámci výstupních pluginů co nejmenší, a aby byl každý záznam profilovacím stromem vyhodnocován pouze jednou. Řešením je rozdělit tento problém na více vzájemně spolupracujících částí, kdy každá z těchto částí bude implementována na tom nejvhodnějším místě.

Profilovací plugin

První z těchto částí bude plugin průchozího typu, který bude mít na starost konstrukci profilovacího stromu ze zadaného konfiguračního souboru a následné vyhodnocování datových záznamů vůči tomuto stromu. Tento plugin však již nijak neřeší žádné další zpracování záznamů s profilováním související, pouze ke každému záznamu přidává informaci o tom, do kterých kanálů a profilů daný záznam spadá. Tím, že je tato část oddělená, a je implementována jako průchozí plugin, který svou činnost výkoná ještě před tím než se datový záznam dostane k výstupním pluginům, jsme vyřešili problém se zbytečným opakovaným vyhodnocováním datových záznamů každým z výstupních pluginů.

Systém profilovacích událostí

Dalším problémem je nutnost implementovat podporu profilování u každého výstupního pluginu zvlášť. Tomuto problému se s ohledem na strukturu systému pluginů v kolektoru zcela nevyhneme, můžeme ji však co nejvíce zjednodušit implementací co největší části logiky nutné pro tuto část profilování v jádru kolektoru, a poskytnutí patřičného rozhraní pro využití této logiky výstupními pluginy. Tento problém byl řešen i v minulém kolektoru, a to



Obrázek 4.3: Znázornění struktury systému profilování

pomocí tzv. profilovacích událostí. Tento přístup se osvědčil, a tak byl tento přístup použit i v této implementaci. Profilovací události, jak již název napovídá, fungují na principu systému událostí. Události jsou rozděleny do dvou sad, a to mezi sadu událostí týkající se profilů a sadu událostí týkající se kanálů. Každá z těchto sad obsahuje čtyři druhy událostí: vytvoření (vznik), úprava, smazání (zánik), a shoda. Události vytvoření a smazání nastávají u každého profilu či kanálu jeho vznikem a zánikem. Tyto události nastanou zaručeně alespoň jednou, a to při začátku profilování kdy vznikají všechny profily a kanály, a pak při konci profilování kdy všechny profily a kanály zanikají. Během těchto událostí mohou chtít výstupní pluginy např. vytvořit odpovídající adresářovou strukturu pro ukládání datových záznamů odpovídající těmto profilům a kanálům. Dále tyto události mohou nastat při úpravě podoby profilovacího stromu za běhu. Profily a kanály zde mohou zanikat, mohou vznikat nové, a některé mohou být také upravovány což vyvolá událost úpravy namísto události vzniku a zániku. Poslední, avšak nejdůležitější, je událost shody. Tato událost je vyvolána když datový záznam vyhovuje profilu či kanálu z profilovacího stromu, a bude např. využívána k uložení daného záznamu do složky odpovídající danému profilu či kanálu. Tyto události jsou implementovány opět pomocí callbacků. Systému profilovacích událostí jsou poskytnuty funkce, které mají být zavolány, když dané události nastanou. Poté je při zpracování datového záznamu výstupním pluginem tomuto systému předán k vyhodnocení výsledek profilování tohoto záznamu a použitý profilovací strom. Systém událostí výsledek zpracuje a zavolá odpovídající callbacky událostí, kterým je mimo jiné poskytnut také kanál či profil, kterého se tato událost týká.

Výsledná struktura tohoto řešení je znázorněna na obrázku 4.3.

Kapitola 5

Výsledky a zhodnocení

5.1 Ověření správnosti implementace

Než začneme testovat výkonnost implementace, je nezbytné se ujistit, že tato implementace funguje správně.

První formou ověření správnosti implementace byla sada jednotkových testů, která je součástí implementace filtru. Jednotkové testy jsou implementovány využitím knihovny Google Test[1]. Každá z funkcí filtru je zkoušena skupinou filtračních výrazů využívající danou funkci za účelem pokrytí co největšího počtu řádků kódu. Jsou zkoumány jak výsledky validních výrazů, tak návratové kódy z části zabývající se zpracováním výrazu pro výrazy nevalidní.

Dále byly porovnávány výsledky filtrování provedené kolektorem v zapojení s filtračním pluginem a výsledky filtrování provedené nástrojem nfdump s odpovídajícím filtračním výrazem.

Poslední formou testování, a to především pro výrazy, které v programu nfdump zapsat nelze, bylo využitím výstupního pluginu IPFIXcol2 ve formátu JSON. Tento formát je na rozdíl od většiny ostatních možných výstupních formátu textový, a díky tomu se s ním jednoduše pracuje a je jednoduše čitelný i člověkem. Testovací soubory byly převedeny do formátu JSON nejdříve bez filtračního pluginu, tedy všechny jejich záznamy, a poté s filtračním pluginem, tedy jen záznamy odpovídající filtračnímu výrazu. Tyto výstupy byly poté kontrolovány s využitím Unixových nástrojů pro práci s textem. Díky čitelnosti tohoto formátu byla provedena i vizuální kontrola.

5.2 Testovací prostředí a konfigurace

Použitá hardwarová konfigurace a verze použitého softwaru je shrnuta v tabulce 5.1. Statistiky z datových sad použitých pro výkonnostní testy jsou uvedeny v tabulce 5.2. Obě datové sady představují data zachycená během cca 12 hodin na jednom z měřících bodů v síti CESNET. Data byla předem anonymizována, aby v nich nebyly obsaženy žádné citlivé informace. Datová sada A obsahuje pouze standardní IPFIX políčka definovaná výchozí tabulkou informačních elementů, datová sada B pak obsahuje navíc políčka s aplikačními daty

[1] dostupné na https://github.com/google/googletest

Stroj ”andre”	
Procesor	Intel Xeon(R) CPU E5-2609 @ 2.40GHz
Operační paměť	16 GiB
Rychlost čtení z disku	60 MB / s
Rychlost čtení z paměti	7690 MB / s
Stroj ”palava”	
Procesor	Intel Xeon(R) Silver 4114 CPU @ 2.20GHz
Operační paměť	96 GiB
Rychlost čtení z disku	893 MB / s
Rychlost čtení z paměti	7570 MB / s
Použitý software	
Operační systém	Scientific Linux 7
Verze jádra	Linux-3.10.0-1062.9.1.el7.x86$_6$4
Verze překladače	gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39)
Verze nfdump	nfdump 1.6.20 (release 1.el7)

Tabulka 5.1: Informace o testovacím prostředí

	Datová sada A	Datová sada B
Velikost fds souboru	17,94 GB	30,49 GB
Velikost nfdump souboru	17.68 GB	
Počet datových záznamů	307 917 528	328 504 547
Počet bajtů v monitorovaném provozu	15,8 TB	21,47 TB
Počet packetů v monitorovaném provozu	16 577 375 585	22 102 077 384

Tabulka 5.2: Informace o datových sadách

Test	Použitý filtrační výraz
Prázdný výraz	`true`
Porovnání IP adresy	`ip 2.2.2.2`
Porovnání IP adresy a portu	`ip 2.2.2.2 or port 4242`
Porovnání IP adresy, portu, a protokolu	`ip 2.2.2.2 or port 4242 or proto udp`
Porovnání DNS jména	`flowmon:dnsQname == "hello"`
Test DNS jména na podřetezec	`flowmon:dnsQname contains "hello"`
Porovnání DNS jména a test na podřetezce	`flowmon:dnsQname == "hello" or flowmon:dnsQname contains "bye"`

Tabulka 5.3: Filtrační výrazy použité v testech

definovaná rozšiřující tabulkou informačních elementů společnosti Flowmon [2]. U souborů není použita komprese.

Aby byly výsledky měření filtrovací části ovlivňovány co nejméně ostatními pluginy, byl IPFIXcol2 pro účely měření zapojen v konfiguraci se vstupem z FDS souboru, filtračním pluginem v průchozí části, a slepým výstupem. Obdobné zapojení se slepým výstupem bylo použito i při měření výkonnosti profilování.

Pro zasazení naměřených výsledků do kontextu byl pro porovnání zvolen nástroj nfdump[3]. Jedná se o jeden z komunitou vyvíjených nástrojů patřící do rodiny nástrojů pro sběr a dotazování se nad flow daty. Rozvíjel se již od počátků existence měření toků pomocí technologie NetFlow, ze které vychází. Toto sebou nese své výhody i nevýhody. Vzhledem k jeho stáří je široce známý, dobře optimalizovaný a časem prověřený. Naopak vzhledem k jeho původu u NetFlow má pouze omezenou podporu standardu IPFIX a např. nepodporuje políčka z privátních tabulek elementů, políčka dynamické délky a jiné novinky. Jako vstup požaduje soubor ve speciálním formátu nfdump, který typicky vzniká výstupem z dalších nástrojů této rodiny nebo sebe sama. Obdobně jako u IPFIXcol2 byl pro účely měření výstup nastaven na slepé zařízení `/dev/null`.

5.3 Výsledky měření

Všechny naměřené hodnoty jsou výsledkem průměrování výsledků 5 – 10 individuálních měření. Bylo také kontrolováno, že rozdíl mezi jednotlivými výsledky měření není nikterak velký např. důsledkem vytížení stroje jiným běžícím procesem. Naměřené hodnoty z jednotlivých měření se od sebe zpravidla lišily v řádu nízkých jednotek procent. Použité filtrační výrazy byly vždy konstruovány tak, aby nedocházelo ke zkratového vyhodnocení – vyhodnocuje se tedy vždy celý výraz, což znamená, že výsledky jsou nejhorší možný případ. Filtrační výrazy použité v jednotlivých testech jsou uvedeny v tabulce **??**.

Nejprve bylo provedeno měření výkonnosti kolektoru na výrazech s nejobvyklejšími IPFIX položkami na obou hardwarových konfiguracích, výsledky můžeme vidět v tabulkách 5.4 a 5.5. Toto měření bylo provedeno i nástrojem nfdump, jehož výsledky jsou k vidění v tabulkách 5.6 a 5.7.

Dále bylo provedeno měření s filtračními výrazy pracující s řetězcovými položkami z rozšiřující tabulky elementů, jak již bylo popsáno u souboru B. Výsledky tohoto měření

[2] dostupná na https://github.com/CESNET/libfds/blob/master/config/system/elements/flowmon.xml

[3] dostupný na https://github.com/phaag/nfdump

Test	Reálný čas	Strojový čas	Dat. záznamů / s
Bez filtračního pluginu	320,35 s	41,27 s	961 195
Prázdný výraz	323,91 s	46,82 s	950 621
Porovnání IP adresy	327,59 s	83,81 s	939 938
Porovnání IP adresy a portu	330,81 s	126,02 s	930 798
Porovnání IP adresy, portu a protokolu	325,01 s	144,90 s	947 419

Tabulka 5.4: Filtrování pomocí IPFIXcol2, stroj "andre", datová sada A

Test	Reálný čas	Strojový čas	Dat. záznamů / s
Bez filtračního pluginu	17,97 s	39,53 s	17 133 180
Prázdný výraz	18,23 s	44,19 s	16 886 998
Porovnání IP adresy	37,80 s	82,02 s	8 145 966
Porovnání IP adresy a portu	55,54 s	104,03 s	5 544 267
Porovnání IP adresy, portu a protokolu	67,63 s	115,39 s	4 552 635

Tabulka 5.5: Filtrování pomocí IPFIXcol2, stroj "palava", datová sada A

Test	Reálný čas	Strojový čas	Dat. záznamů / s
Porovnání IP adresy	289,47 s	20,29 s	1 063 734
Porovnání IP adresy a portu	293,53 s	25,23 s	1 049 015
Porovnání IP adresy, portu a protokolu	289,86 s	27,23 s	1 062 285

Tabulka 5.6: Filtrování pomocí nfdump, stroj "andre", datová sada A

Test	Reálný čas	Strojový čas	Dat. záznamů / s
Porovnání IP adresy	12,17 s	12,14 s	25 293 044
Porovnání IP adresy a portu	14,94 s	14,90 s	20 615 795
Porovnání IP adresy, portu a protokolu	16,50 s	16,45 s	18 666 193

Tabulka 5.7: Filtrování pomocí nfdump, stroj "palava", datová sada A

Test	Reálný čas	Strojový čas	Dat. záznamů / s
Bez filtračního pluginu	30,48 s	64,85 s	10 778 592
Prázdný výraz	30,30 s	71,75 s	10 840 839
Porovnání DNS jména	30,28 s	86,81 s	10 849 791
Test DNS jména na podřetězec	33,47 s	97,47 s	9 814 162
Porovnání DNS jména a test na podřetězec	52,66 s	127,61 s	6 238 810

Tabulka 5.8: Filtrování pomocí IPFIXcol2, stroj "palava", datová sada B

Počet IP adres v seznamu	Reálný čas	Strojový čas	Dat. záznamů / s
10 IP adres	67,89 s	114,75 s	4 535 822
100 IP adres	361,80 s	411,26 s	851 073
1000 IP adres	3073,16 s	3144,68 s	100 195

Tabulka 5.9: Porovnávání IP adresy vůči seznamu náhodných IP adres, stroj "palava", datová sada B

můžeme vidět v tabulce 5.8. Toto měření už s použitím nástroje nfdump být provedeno nemohlo, jelikož tento typ položek nepodporuje. Vzhledem k výsledkům z první části měření a jeho časové náročnosti už nebylo toto měření zopakováno ani na stroji "andre", jelikož bylo zřejmé, že výsledky by vypadaly obdobně jako u první části měření a použité filtrační výrazy by je nijak neovlivnily.

Pro demonstraci možnosti rozšiřovat filtr o vlastní datové typy bylo provedeno měření výrazu testující, zda se IP adresa nachází v seznamu IP adres. Toto měření bylo provedeno s instancí filtru, kdy je seznam IP adres běžným seznamem, a poté pro porovnání s instancí, kde je filtr rozšířen o vlastní datový typ IP trie a ze seznamu IP adres se již během překladu konstruuje trie, se kterou se pak při vyhodnocování filtru pracuje místo seznamu. V tabulce 5.9 můžeme vidět výsledky měření, kdy se se seznamem IP adres pracuje jako s běžným seznamem, a v tabulce 5.10 výsledky, kdy je seznam převeden na trii.

Výkonnost profilování byla testována v konfiguraci s profilovacím stromem vyobrazeným na obrázku 5.1. Filtrační výrazy použité v jednotlivých kanálech jsou uvedeny v tabulce 5.11. Výsledky z měření na obou strojích jsou pak uvedeny v tabulce 5.12.

5.4 Zhodnocení výsledků

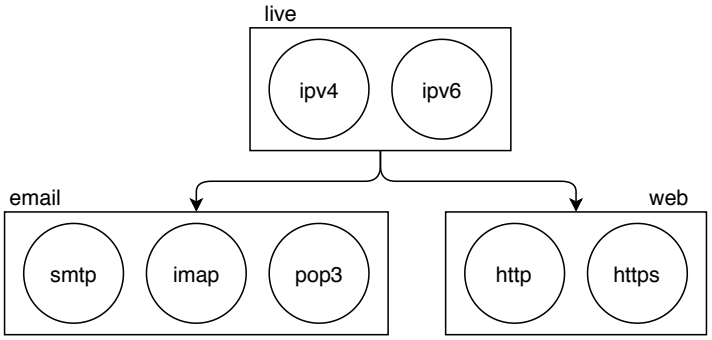
Z výsledků naměřených na stroji "andre" se ukázalo, že úzkým hrdlem celého procesu u této konfigurace není samotný proces filtrování, nýbrž čtení dat ze souboru umístěném na disku. V tomto případě nemohl pomoci ani proces cachování souborů zajišťovaný operačním

Počet IP adres v trii	Reálný čas	Strojový čas	Dat. záznamů / s
10 IP adres	55,18 s	29,69 s	5 953 056
100 IP adres	73,54 s	56,67 s	4 467 018
1000 IP adres	89,79 s	73,66 s	3 658 790
10000 IP adres	141,36 s	29,40 s	2 323 845
100000 IP adres	247,12 s	36,25 s	1 329 345

Tabulka 5.10: Porovnávání IP adresy vůči seznamu náhodných IP adres s optimalizací na trii, stroj "palava"

Název kanálu	Filtrační výraz
ipv4	`ipv4`
ipv6	`ipv6`
pop3	`port in [110, 995]`
imap	`port in [143, 993]`
smtp	`port in [25, 465]`
http	`port in [80, 8080]`
https	`port 443`

Tabulka 5.11: Filtrační výrazy použité v rámci kanálů profilovacího stromu



Obrázek 5.1: Konfigurace profilovacího stromu použitého pro měření

Stroj	Reálný čas	Strojový čas	Dat. záznamů / s
andre	677,67 s	312,32 s	484 755
palava	156,22 s	257,73 s	2 102 799

Tabulka 5.12: Výkonnost profilování, oba stroje, datová sada B

systémem, jelikož systém neměl dostatek dostupné operační paměti pro pojmutí celého souboru do mezipaměti, a tak i při vykonávání po sobě jdoucích testů pracujících se stejným souborem musel být opět celý soubor od začátku načítán z disku. Použitý filtrační výraz v tomto případě ovlivňuje pouze čas strávený na procesoru, skutečný čas je až na drobné odchylky konstantní. Při porovnání výsledků naměřených při použití kolektoru s výsledky naměřenými při použití nástroje nfdump můžeme i zde vidět, že použitý filtrační výraz naměřený reálný čas neovlivňuje, a rozdíl mezi těmito dvěma nástroji je dalo by se říci konstantní.

Na výsledcích naměřených na stroji "palava", kde je čtení z disku dostatečně rychlé, aby nepředstavovalo omezení pro proces filtrování, můžeme pozorovat rozdíly. Nástroj nfdump se na této konfiguraci ukazuje jako rychlejší, což se však dalo očekávat. Je nutné mít na paměti, že nfdump je nástroj specializovaný na čtení záznamů ze souboru a jejich následnou filtraci, agregaci, počítání statistik a podobné operace. Podporuje ovšem pouze omezené množství IPFIX položek (např. řetězce a operace nad nimi podporovány nejsou) a vyžaduje vstup ve speciálním formátu nfdump. IPFIXcol2 je primárně kolektor, a čtení dat ze souborů a jejich následná filtrace je pouze jednou z možných konfigurací tohoto nástroje umožněna jeho modulární architekturou a systémem pluginů, daní za to je však dodatečná režie a méně možností optimalizace. U výsledků IPFIXcol2 dále můžeme pozorovat značný nárůst ve spotřebovaném strojovém času oproti nástroji nfdump, což je zapříčiněno vícevláknovou architekturou kolektoru.

Vzhledem k tomu, že je k profilování toků používána stejná filtrační komponenta a profilovací strom je prakticky jen strom filtračních výrazů, je výkonnost profilování silně závislá na výkonu filtrační komponenty. Toto můžeme vidět i na výsledcích výkonu profilování, kde při srovnání hodnot naměřených na obou strojích vidíme obdobný charakter jako u hodnot naměřených při testech výkonu filtrování. Je nutné podotknout, že typický případ užití profilování bude spíše v režimu online, tedy např. profilování dat přicházejících ze sítě na kolektor. V tomto případě je tedy relevantnějším údajem počet datových zpracovaných datových záznamů za sekundu nežli čas zpracování celého souboru.

Během měření bylo prováděno i výkonnostní profilování, kde se ukázala slabá místa tohoto procesu. Největší zpomalení představoval úkon přistoupení k položce v IPFIX záznamu, což se ještě během testování povedlo výrazně zlepšit. Další část, která by se zajisté dala zlepšit a zabírá poměrně velkou část procesorového času je režie spojená s vyhodnocováním stromu filtračního výrazu. Části tohoto stromu jsou často vyhodnocovány i několikrát, např. identifikátor `ip` se mapuje na 4 různé IPFIX položky (zdrojová a cílová IPv4 a IPv6 adresa), identifikátor `port` na 2 různé IPFIX položky (zdrojový a cílový port), toto znamená v nejhorším případě nutnost některé části stromu vyhodnocovat i 6krát pro jeden datový záznam. Toto je taky důvod, proč zdánlivě složitější operace pracující s řetězci (viz tabulka 5.8) mají v naměřených výsledcích lepší časy, než některé zdánlivě jednoduché operace pracující pouze s IP adresou a portem. Do budoucna by se určitě dal úkon jednoho vyhodnocení stromu optimalizovat, a celkový počet nutných vyhodnocení snížit např. získáním více informací ze zadaného výrazu již za překladu nebo s pomocí analýzy šablonových sad procházejících kolektorem. Překvapivě se zase u některých jiných úkonů, u kterých se očekávalo, že budou způsobovat zpomalení, tato domněnka nepotvrdila. Jedná se např. o nutnost v rámci filtračního pluginu vzhledem k nynější architektuře kolektoru vytvářet nové IPFIX zprávy, překopírovávat data, a původní IPFIX zprávy uvolňovat namísto pouhé úpravy zpráv původních. Toto lze vidět na naměřených výsledcích srovnáním hodnot u testů bez filtračního pluginu a testů s prázdným filtračním výrazem.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat řešení pro filtrování IP toků, především pak IP toků protokolu IPFIX, a následně provést jeho integraci do modulárního kolektoru IPFIXcol2 za účelem jeho rozšíření o podporu filtrování a profilování záznamů o IP tocích. Tento záměr byl splněn, a v rámci knihovny libfds vznikl obecný filtr IP toků a jeho varianta specializovaná na práci s IPFIX položkami. Dále byl kolektor IPFIXcol2 rozšířen o filtrační a profilovací modul využívající tuto filtrační komponentu knihovny libfds.

V první části práce byl nejprve představen systém IPFIX, především pak jeho princip a terminologie, kterou tento systém využívá. Dále byl popsán stejnojmenný protokol IPFIX využívaný tímto systémem. Pro navržení vhodného řešení pro filtrování toků a pro to nezbytnou efektivní práci s daty reprezentující jednotlivé toky bylo obzvlášť důležité podrobně se seznámit s formátem dat a strukturami, který tento protokol využívá. Na závěr této části byl popsán kolektor IPFIXcol2 se zaměřením na jeho systém modulů.

Úvodem druhé části je popsána problematika filtrování toků. Jsou navrhnuty funkce a vlastnosti, které by mělo výsledné filtrační řešení obsahovat spolu s jejich odůvodněním a případy užití. Dále jsou popsány jednotlivé části implementace od zpracování textové formy filtračního výrazu až po jeho vyhodnocování v jeho finální podobě ve formě stromové struktury. Poté následuje přidání podpory IPFIX položek a začlenění finálního řešení do kolektoru IPFIXcol2 v podobě filtračního modulu.

Ve třetí části je filtrační komponenta, která byla vyvinuta v rámci předchozí části, využita pro rozšíření kolektoru IPFIXcol2 o podporu profilování toků. Nejprve je nastíněna problematika profilování toků, poté jsou popsány problémy spojené s profilováním vzhledem k aktuální architektuře kolektoru a je navrhnuto jejich řešení, a nakonec je toto řešení implementováno včetně profilovacího modulu.

Poslední část popisuje provedené funkční a výkonnostní testy. Úvodem je popsáno použité testovací prostředí a datové sady. Je provedeno měření několika testovacích případů na dvou různých konfiguracích. Pro porovnání je vybrán dobře známy nástroj nfdump a některé z podporovaných testovacích případů jsou změřeny i tímto nástrojem. Naměřené výsledky jsou následně zanalyzovány s ohledem na použitý filtrační výraz a hardwarovou konfiguraci stroje a jsou popsány možné problémy současného řešení a jejich případné řešení.

Do budoucna by bylo vhodné se zaměřit na optimalizaci některých procesů zmíněných v části věnující se testování. Dalším možným rozšířením je podpora některých dalších částí standardu IPFIX jako jsou např. strukturované datové typy. Možný je i další rozvoj jazyku filtru v závislosti na požadavcích uživatelů, v úvahu připadá např. podpora regulárních výrazů nebo uživatelsky definovaných funkcí.

Literatura

[1] Aho, A. *Compilers : principles, techniques, tools*. Boston: Pearson/Addison Wesley, 2007. ISBN 0321486811.

[2] Bendersky, E. *Parsing expressions by precedence climbing* [online]. Eli Bendersky's website, 2012. Dostupné z: https://eli.thegreenplace.net/2012/08/02/parsing-expressions-by-precedence-climbing.

[3] Claise, B. *Cisco Systems NetFlow Services Export Version 9* [Internet Requests for Comments]. RFC 3954. RFC Editor, October 2004. Dostupné z: http://www.rfc-editor.org/rfc/rfc3954.txt.

[4] Claise, B., Dhandapani, G., Aitken, P. a Yates, S. *Export of Structured Data in IP Flow Information Export (IPFIX)* [Internet Requests for Comments]. RFC 6313. RFC Editor, July 2011. Dostupné z: http://www.rfc-editor.org/rfc/rfc6313.txt.

[5] Claise, B. a Trammell, B. *Information Model for IP Flow Information Export (IPFIX)* [Internet Requests for Comments]. RFC 7012. RFC Editor, September 2013. Dostupné z: http://www.rfc-editor.org/rfc/rfc7012.txt.

[6] Claise, B., Trammell, B. a Aitken, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [Internet Requests for Comments]. STD 77. RFC Editor, September 2013. Dostupné z: http://www.rfc-editor.org/rfc/rfc7011.txt.

[7] Clarke, K. *The top-down parsing of expressions*. Dept. of Computer Science and Statistics, Queen Mary College, June 1986. Dostupné z: http://antlr.org/papers/Clarke-expr-parsing-1986.pdf.

[8] Huták, L. *Nová generace IPFIX kolektoru*. Brno, CZ, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/114769.

[9] Kozubík, M. *Profilování dat pomocí IPFIX mediátoru*. Brno, CZ, 2015. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/88503.

[10] Levine, J. *Flex & bison*. Sebastopol, Calif: O'Reilly Media, 2009. ISBN 0596155972.

[11] Matousek, P. *Sitove aplikace a jejich architektura*. Brno: VUTIUM, 2014. ISBN 978-80-214-3766-1.

[12] Wikipedia. *ISO 8601 — Wikipedia, The Free Encyclopedia* [http://en.wikipedia.org/w/index.php?title=ISO%208601&oldid=958726446]. 2020. [Online; accessed 02-May-2020].

Příloha A

Obsah CD

Na CD jsou v oddělených adresářích umístěny zdrojové soubory knihovny libfds a kolektoru IPFIXcol2. Dále jsou zde zdrojové soubory této práce a soubor README.txt.