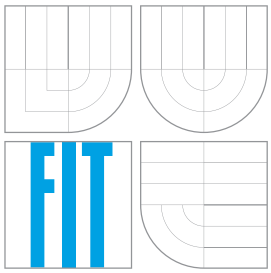
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# APLIKAČNÍ ROZHRANÍ PRO PRÁCI S NETFLOW DATY

NETFLOW DATA APPLICATION INTERFACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV ŠOLTÉS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ PODERMAŃSKI

BRNO 2013

# Abstrakt

Táto diplomová práce se zabývá návrhem a implementací aplikačního rozhraní pro práci s daty NetFlow. Obsahuje teoretický rozbor způsobu monitorování sítě za pomocí IP Flow, popis nástroje nfdump a jeho způsob ukládání dat NetFlow v9. Při návrhu aplikačního rozhraní je kladen důraz na efektivní manipulaci s daty.



# Abstract

This diploma thesis deals with design and implementation of NetFlow data manipulation tool. It contains analysis of IP Flow network monitoring, description of nfdump tool and format of Netflow v9 records saved by nfdump. The focus of this application interface lies in effective manipulation with NetFlow records.



# Klíčová slova

NetFlow, záznam NetFlow v9, IPFIX, nfdump, SWIG, XS, Cython, Ctypes

# Keywords

NetFlow, record NetFlow v9, IPFIX, nfdump, SWIG, XS, Cython, Ctypes


# Citace

Miroslav Šoltés: Aplikační rozhraní pro práci s netflow
daty, diplomová práce, Brno, FIT VUT v Brně, 2013

# Aplikační rozhraní pro práci s netflow daty

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Tomáše Podermańského. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Šoltés
22. května 2013

## Poděkování

Děkuji Ing. Tomáši Podermańskimu za vedení diplomové práce, cenné připomínky, zpřístupnení záznamu NetFlow a poskytnuté konzultace k dosaženým výsledkům.

# Obsah

# Kapitola 1

# Úvod

Informačné technológie prešli veľkým rozvojom vo svete. Prínos do mnohých oblastí uľahčilo prácu ľuďom. Medzi jeden z najzákladnejších prínosov bola komunikácia, čo je vďaka masívnej celosvetovej infraštruktúre, ktorú nazývame Internet. V dnešnej dobe je veľmi populárny a pozostáva z mnoho menších sietí, ktoré sú navzájom poprepájané. Internet je využívaný veľkým množstvom služieb, ktoré potrebujú na svoj beh komunikáciu. Ľudia tak nemajú veľký problém byť v kontakte so svojimi blízkymi pomocou e-mailov alebo taktiež komunikovať v reálnom čase na základe instant messagingu. Problémom nie je ani hlasová komunikácia spojená s videom, za ktorú ľudia nemusia platiť. Sociálne siete patria dnes k najvyužívanejším prostriedkom na Internete. Možnosť zdieľať fotografie, videá a vlastné myšlienky priťahuje čím ďalej tým viac ľudí. Informácie zo sveta sú k dispozícii hneď.
Internet nám poskytuje aj možnosti týkajúce sa vzdelávania napríklad pomocou encyklopédií alebo streamovaním videí zo škôl a iných inštitúcií. V dnešnej dobe je pre veľké spoločnosti nutné mať k dispozícii dáta zo siete, ktoré slúžia ako informácie o stave danej sieti. Informácie zahrňujú kto, kedy a kde spolu komunikoval. Dôvodom je rozsiahli počet útokov a bezpečnosť je tým pádom vyžadujúca a nutná. Uchovávanie týchto dát nám umožní detekovať útok a nájsť potencionálneho páchateľa. Analýzou môžeme získať vedomosť ohľadom slabých miest a vďaka tomu ich aj zabezpečiť. Zachytené dáta je možné využiť aj na účtovanie a fakturizáciu užívateľov v sieti poskytovateľa internetu. Firma Cisco Systems nám všetky tieto výhody monitorovania, bezpečnosti, sledovania a analýzy zabezpečila protokolom NetFlow. Tento protokol slúži na monitorovanie IP tokov na sieti.

## 1.1 Cieľ diplomovej práce

Hlavným cieľom diplomovej práce bude vytvoriť aplikačné rozhranie pre prácu s dátami NetFlow. Úlohou aplikačného rozhrania je uľahčiť prácu s danými dátami a vytvoriť nové možnosti ako s nimi manipulovať. Prínos tejto práce bude hlavne pre administrátorov počítačových sietí, ktoré využívajú na monitorovanie nástroj nfdump. Táto diplomová práca naviazuje na semestrálny projekt, v ktorom bol popísaný protokol NetFlow, jeho architektúru a aké mechanizmy sa podieľajú a slúžia na generovanie záznamov NetFlow. Taktiež bol popísaný nástroj nfdump, ktorý dokáže pracovať s NetFlow dátami verzie 9 a IPFIX. Časť semestrálneho projektu bola venovaná štruktúre binárnych súborov, ktoré ukladá nfdump. Bol popísaný návrh aplikačného rozhrania a možnosti techník, ktoré sú k dispozícii. Rozobraná bola aj implementácia rozhrania a ďalšie možnosti jeho rozšírenia, ktoré môžu vylepšiť prácu s rozhraním.

## 1.2 Členenie práce

Kapitola 2 bola venovaná popisu protokolu NetFlow a jeho použitiu. V kapitole 3 sa zaoberám nástrojom nfdump, jeho popisu a ďalšími službami, ktoré poskytuje. V tejto kapitole bola taktiež rozobraná štruktúrá binárnych súborov, ktoré ukladá nástroj nfdump. Štruktúra súborov je rozdelená na logické celky. V kapitole 4 venujem pozornosť sieti VUT, jej architektúry a fyzickej realizácie. Kapitola 5 popisuje návrh a možné techniky prepojenia nízkoúrovňového jazyka C s vysokoúrovňovými skriptovacími jazkymi. Kapitola 6 popisuje implementáciu aplikačného rozhrania, rozbor jeho funkcií a ako sa s ním pracuje. Následne sú uvedené aj jednoduché príklady. Kapitola 7 sa venuje možným rozšíreniam, ktoré by mohli dopomôcť k manipulácii s NetFlow dátami.

# Kapitola 2

# NetFlow

Zameranie tejto kapitoly sa sústredí na popis protokolu NetFlow. V popise bude zahrnutý stručný úvod do protokolu, architektúra protokolu, princíp a použitie. Za pomoci protokolu NetFlow sa dajú ukládať záznamy o prevádzke na sieti. V tejto práci sa bude s týmito dátami pracovať.

## 2.1 Protokol NetFlow

V roku 1990 firma Cisco systems vyvinula tento protokol pôvodne iba pre ich firemné zariadenia. NetFlow je uzavretý protokol, no jeho špecifikácia je plne k dispozícii. Spočiatku bola softwarová implementácia pre vylepšenie rýchleho prepínania (Cisco Fast Switching). Hlavnou myšlienkou bolo po prvom pakete z toku vytvoriť prepínací NetFlow záznam. Tento záznam bol používaný pre ďalšie pakety z rovnakého toku, ktoré dorazili neskôr až do jeho životnosti. Technológia bola výhodná najmä pre lokálne siete napríklad pri filtrovaní prevádzky pomocou ACL[1]. Prepínanie NetFlow (NetFlow switching) nebolo vhodné pre internetové smerovače na chrbticovej sieti, kde počet súbežných tokov bol viac dôležitejší ako v lokálnych sieťach. Ukázalo sa, že bol nevhodný aj do sietí kde prevádzka spôsobovala veľa tokov, ktoré netrvali dlho ako napríklad DNS[2] požiadavky. Čo sa týka prepínacej technológie, NetFlow bol nahradený expresným prepínaním (Cisco Express Forwarding). Technológie podobné prepínaniu NetFlow sú stále používané vo väčšine firewalloch a softwarovo založených IP smerovačoch[13].

V súčasnosti je najnovšou verziou verzia 9. Popis môžeme nájsť v RFC 3954[1]. Implementovať protokol je možné aj na iné zariadenia alebo operačné systémy (GNU/Linux, FreeBSD, OpenBSD) vďaka dostupnej špecifikácii. Protokol NetFlow je určený na monitorovanie sietí. Špecifikuje štruktúru zachytených záznamov, ktoré sú zozbierané po sieti a prenos daných záznamov z exportéru na kolektor. Je chápaný ako celý proces merania tokov. Medzi hlavné pojmy spojené s protokolom NetFlow patria:

- **Bod pozorovania** (Observation Point): Miesto v sieti, kde IP pakety môžu byť pozorované. Príkladom je množina rozhraní na sieťovom zariadení (smerovač, prepínač). Každý bod pozorovania je spojený s pojmom doména pozorovania.
- **Doména pozorovania** (Observation Domain): Množina pozorovacích domén, ktorá je najväčšia agregovateľná množina informácii tokov na sieťovom zariadení. Príkladom

[1] Access Control List
[2] Domain Name System

tohto pojmu je modul smerovača pozostávajúci s niekoľkými rozhraniami, ktoré sú zároveň bodom pozorovania.

- **IP tok** (IP Flow, Flow): Je definovaný ako množina IP paketov prechádzajúca bodom pozorovania v sieti počas určitého časového intervalu. Všetky pakety, ktoré patria do daného toku majú spoločnú množinu vlastností a sú spracované v bode pozorovania. Medzi kľúčové vlastnosti patrí zdrojová a cieľová IP adresa, zdrojový a cieľový port, číslo protokolu, rozhranie a ToS (Type Of Service). Medzi ďalšie vlastnosti, ktoré boli zaznamenané a IP tok obsahuje, môžeme zaradiť napríklad aj časové značky (doba vzniku toku a jeho ukončenia), počet paketov, počet bajtov a iné.
- **Záznam toku** (Flow Record): Poskytuje informácie o IP toku spracované bodom pozorovania.
- **Exportér** (Exporter): Zariadenie poskytujúce služby NetFlow. Monitoruje pakety, ktoré prechádzajú bodom pozorovania a vytvára toky zo zachytených paketov. Exportér je popísaný v kapitole 2.2.1.
- **NetFlow Kolektor** (NetFlow Collector): Prijíma záznamy z jedného alebo viacerých exportérov. Prijatý paket analyzuje a ukladá informácie. Tieto záznamy sa môžu, ale nie nutne agregovať pred uložením. Kolektor je podrobnejšie rozobraný v kapitole 2.2.2.
- **Exportovaný paket** (Export packet): Paket z exportéra, ktorý obsahuje záznam o exportéri a cieľovom kolektore.

Na obrázku 2.1 je znázornena identifikácia dátového toku.


Obrázek 2.1: Identifikácia dátového toku [4]

## 2.2 Architektúra

K zachytávaniu informácii je potrebné do siete nasadiť zariadenie, ktoré umožňuje zachytávať IP toky, spájať ich a poslať do nejakého úložiska, kde sa môžu dané uložené informácie ďalej analyzovať. K tomu nám slúži exportér (zber a zaslanie dát) a kolektor (zálohovanie a analýza). Vzťah medzi exportérom a kolektorom je $N : N$. To znamená, že jeden exportér môže zaslať informácie viacerým kolektorom a jeden kolektor môže prijmať dáta od viacerých exportérov.

### 2.2.1 Exportér

Hlavnou úlohou exportéru je sledovať a monitorovať premávku na sieti, následne vytvárať IP toky a zaslať ich na kolektor. Každý tok je označený časovými značkami (čas jeho vzniku a ukončenia), počtom paketov a počtom bajtov. Exportér môže zaznamenať aj ďalšie informácie zo siete, no záleží na verzii protokolu a podpore exportéru. Zaslanie toku na kolektor záleží od dvoch faktorov. Buď uplynul určitý čas zberu alebo bolo zozbieraných určitý počet tokov. Ak exportér odošle informácie, tak ich vymaže aby bola možnosť ukladať do pamäte nasledujúce toky, ktoré bude následne zaznamenávať[1].
Princíp exportéru a jeho fungovanie môžeme rozčleniť do niekoľkých bodov po poradí.

1. Prijatie paketu a extrakcia dát
2. Aktualizovanie alebo založenie záznamu v NetFlow cache
3. Expirácia
4. Export dát

Expirácia je čas, ktorý po vypršaní exportuje IP toky na kolektor. Za neaktívny tok sa považuje tok, v ktorom nie je zachytený žiaden paket v určitom bode pozorovania za určitý časový interval. Exportovanie toku môže zavisieť na základe z niektorých uvedených podmienok.

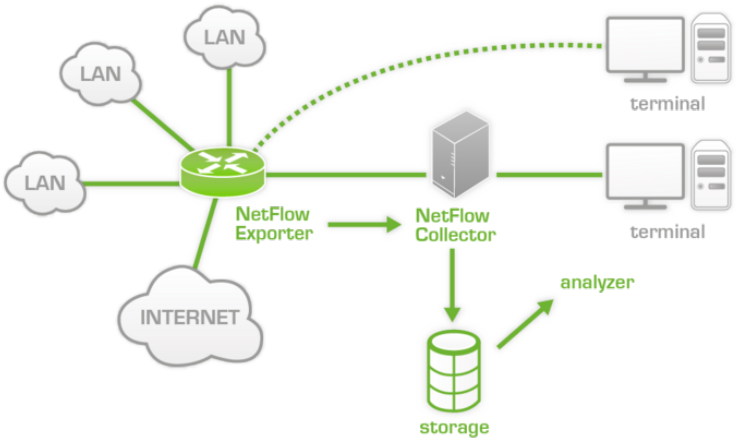
- Pri obmedzení exportéru na pamäť. V tomto prípade sú toky exportované permanentne z dôvodu uvolnenia danej pamäte.
- Export dát pravidelne po dlhotrvajúcom čase pre tok.
- Neaktívnosť toku v časovom intervale. Časový interval pre neaktívny tok by mal byť konfigurovateľný na strane exportéru s minimálnou hodnotou 0, čo značí okamžitú expiráciu.
- Detekovanie konca toku v exportéri, napríklad vďaka nastavenému bitu RST alebo FIN v TCP spojeni.

Z hľadiska menších nárokov na hardware je výhodne využiť na exportéri vzorkovanie (sampling). Vzorkovanie nám umožňuje výber iba niektorých paketov. Čo sa týka hardwarovej realizácie exportéru, tak môže byť na sieť zahrnutý v aktívnom prvku alebo ako sonda. Sonda je samostatné zariadenie, ktoré je pripojené k sledovanej linke pomocou rozbočovača. Medzi aktívne zariadenie patrí router alebo switch.

### 2.2.2 Kolektor

Kolektor je zariadenie pre ukladanie informácii o dátových tokoch na disk alebo do databáze z jedného alebo viacerých exportérov. Slúži nám ako úložisko pre IP toky. Keďže komunikácia medzi kolektorom a exportérom prebieha pomocou nespoľahlivého protokolu UDP, nemusia exportované toky z exportéru doraziť na kolektor. Takto dochádza k strate dát, ktoré nebudú môcť byť analyzované. Vzorkovanie (sampling) a filtrovanie paketov je možné aj na strane kolektoru, tak ako to bolo aj na strane exportéru. Uložené dáta a informácie o dátových tokov zo siete môžu byť zobrazené v grafickej reprezentácii pomocou grafov alebo tabuliek. To umožňuje administrátorom ľahšiu a priehľadnú kontrolu nad sieťou.

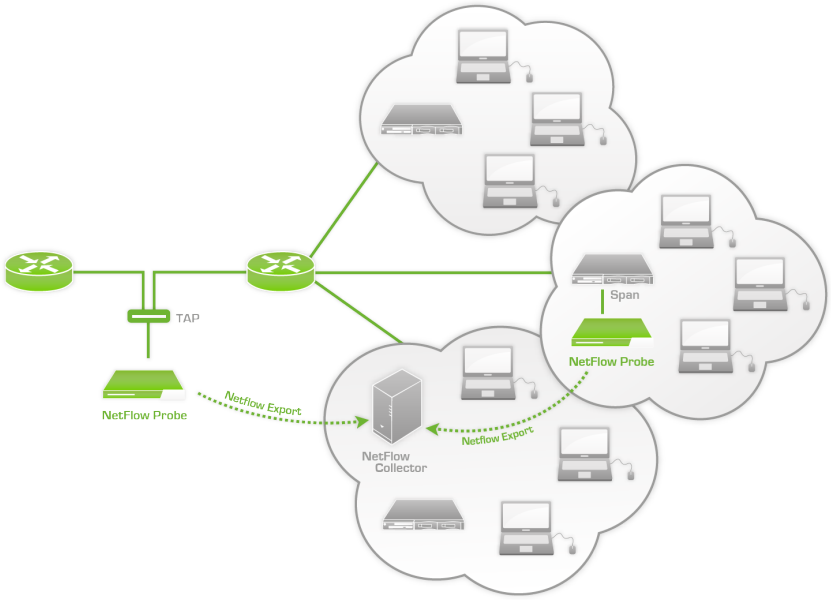
Na obrázku 2.2 je znázornený protokol NetFlow, kde export dát realizuje router.




Obrázek 2.2: Export dát z NetFlow exportéru [10]

Z hľadiska hardwarových nárokov a pomerne vysokej ceny exportéru, ktorý je realizovaný v aktívnom zariadení (router), boli navrhnuté a implementované do siete sondy. Sondy participujú ako samostatné zariadenia a slúžia výhradne na monitorovanie siete. Vo väčšine prípadov sondy majú finančne menšie nároky a preto nasadenie v malých a stredných sieťach je výhodou. Ak sú do sieťe nasadené sondy, tak sú potrebné zariadenia TAP alebo SPAN. Tieto zariadenia nám umožňujú kopírovať premávku na sieti na iné porty.

**SPAN** (Switch port analyzer) analyzuje sieťovú premávku prechádzajúcu cez porty a preposiela kópiu premávky do ďalšieho portu rozbočovača, ktorý bol pripojený na sondu alebo bezpečnostné zariadenie. SPAN odráža prijatú, odoslanú alebo obojakú premávku na jeden alebo viac zdrojových portov do cieľového portu na analýzu [2].
**TAP** je hardvérové zariadenie, ktoré poskytuje prístup k dátam, ktoré prechádzajú cez počítačovú sieť. Obsahuje najmenej tri porty A, B a port na monitorovanie. TAP vôbec neblokuje premávku a zároveň ju kopíruje do portu na monitorovanie. Tento port nám slúži na umožnenie zberu dát [12].
Na obrázku 2.3 vidíme realizáciu NetFlow protokolu za pomoci sond a zariadení TAP a SPAN, ktoré sú k tomu potrebné.




Obrázek 2.3: Export dát z NetFlow sónd [10]

## 2.3 Verzie

Protokol NetFlow prešiel počas svojho vývoja k viacerým zmenám. Tieto zmeny spočívajú v rozšírení dát, ktoré sa majú zachytávať. Medzi najviac používané patrí verzia 5 a do popredia sa tlačí taktiež verzia 9. Verzia 9 umožňuje monitorovať viacero údajov oproti verzii 5, napríklad VLAN, BGP, IPv6 a iné. V súčastnosti je vo vývoji verzia IPFIX[3], ktorá bude založená na najnovšej verzii 9. IPFIX je popísaný v kapitole 2.3.1.
Tabuľka 2.1 nám zobrazuje prehľad verzií protokolu NetFlow od jeho vzniku.

| Verzia | Popis |
|---|---|
| v1 | Najstaršia verzia protokolu |
| v2 - v4 | Tieto verzie neboli nikdy zverejnené |
| v5 | Najrozšírenejšia verzia, vyhradená pre toky s IPv4 adresou |
| v6 | Pridané informácie o zapuzdrení |
| v7 | Obsahuje informácie o smerovači |
| v8 | Agregácia tokov |
| v9 | Aktuálna verzia protokol s novým formátom, obsahuje nové položky a využíva šablóny pre záznamy |
| IPFIX | Nadstavba na verziu 9, IETF štandard |

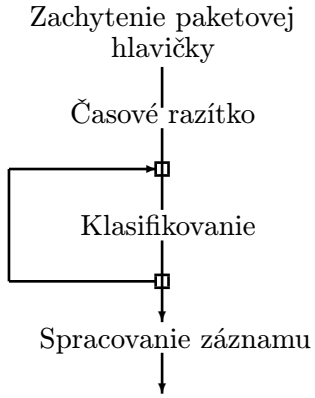
Tabulka 2.1: Prehľad verzií protokolu NetFlow

[3]IP Flow Information eXport

### 2.3.1 IPFIX

IETF štandardizovala protokol NetFlow a nazvala ho IPFIX (IP Flow Information eXport). Je postavený na verzii 9. Na prenos je možné využiť protokol SCTP pre zabezpečenú prevádzku na sieti alebo TCP a UDP. Taktiež sa môže použiť protokol IPsec[4] s autentifikáciou a šifrovaním alebo šifrovací protokol TLS[5]. Tento štandard umožňuje vzorkovať dáta z dôvodu menšieho zaťaženia zariadení[11]. Keďže IPFIX vychádza z protokolu NetFlow verzie 9, tak pojmy ako bod pozorovania, IP tok a záznam toku sú už známe. Medzi ďalšie pojmy spojené s protokolom IPFIX patria [3]:

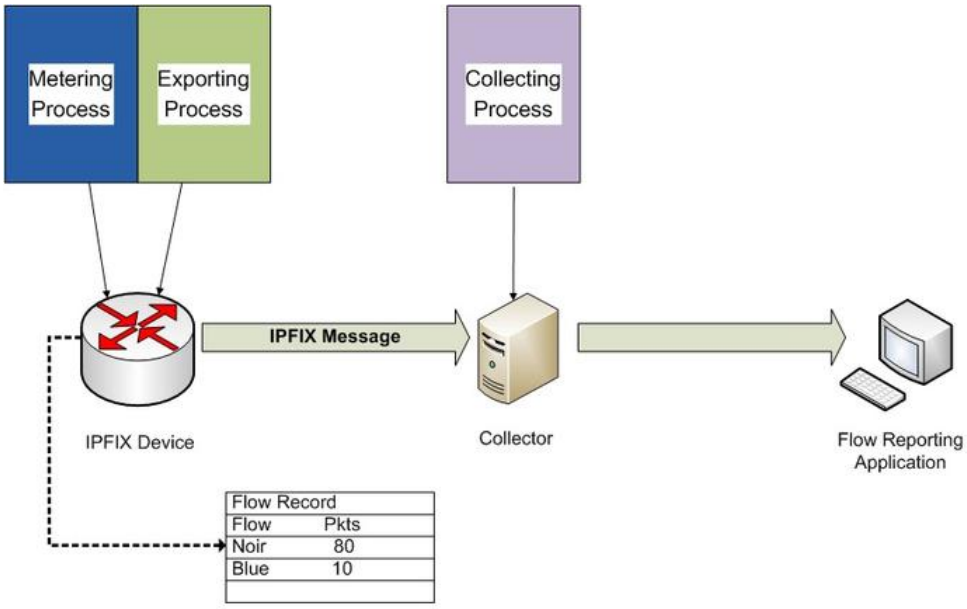
- **Proces merania** (Metering process): Generuje záznamy toku. Vstupom sú paketové hlavičky získané z bodu pozorovania a informácie paketu ako napríklad zvolené výstupné rozhranie. Tento proces pozostáva z funkcií (zachytenie paketovej hlavičky, časové značky, vzorkovanie, klasifikovanie, údržba záznamu). Údržba spočíva vo vytvorení nového záznamu, aktualizovanie existujúceho, výpočet štatistik, odvodzovanie vlastností, detekovanie expirácie toku, predávanie záznamu do exportovacieho procesu a mazanie záznamu.
  Funkcie vzorkovania a klasifikácie môžu byť aplikované viac ako jedenkrát s rôznymi parametrami. Na obrázku 2.4 je znázornená sekvencia, v ktorej sú aplikované funkcie. Vzorkovanie nie je znázornené a môže byť aplikované pred každou funkciou.
- **Proces exportovania** (Exportering process): Spočíva v zasielaní záznamov do jedného alebo viacerých procesov zhromažďovania. Záznam je generovaný jedným alebo viacerými procesmi merania.
- **Proces zhromažďovania** (Collecting process): Príjma záznamy z jedného alebo viacerých procesov exportovania. Môže ukladať prijaté záznamy alebo ich spracovať.




Obrázek 2.4: Funkcia procesu merania

Na obrázku 2.5 je znázornená architektúra protokolu IPFIX. Môžeme vidieť, že proces merania a exportovania je súčasťou exportéru. Kolektor má za úlohu ukladať dané dáta a preto je proces zhromažďovania jeho súčasťou.

[4] Internet Protocol Security
[5] Transport Layer Security




Obrázek 2.5: Architektúra protokolu IPFIX [6]

## 2.4 Použitie

Protokol NetFlow nám umožňuje monitorovanie na sieti. Znalosť premávky na sieti je veľmi výhodná pre administrátorov. Môže tak sledovať aplikácie, ktoré sa využívajú na danej sieti užívateľov a za pomoci analýzy zistiť časy, v ktorých bola najviac využívaná a zaťažená sieť. Tento protokol je veľmi výhodný pre prevádzkovateľov internetu, z dôvodu fakturizácie a účtovania pre užívateľov. Verzia 9 protokolu NetFlow bola navrhnutá s predpokladom rozmiestnenia kolektoru a exportéru v jednej privátnej sieti v tesnej blízkosti. Je možnosť použiť transport IP tokov aj cez verejné siete, no môže tak dôjsť k bezpečnostným rizikám, poprípade strate dát vďaka nespoľahlivému protokolu UDP slúžiaceho na prenos paketov medzi exportérom a kolektorom. Útočník môže pakety, ktoré boli exportované zachytiť, následne modifikovať alebo uložiť. Preto je tu dané riziko odchytenia a sfalšovania údajov. Z dôvodu efektívnosti implementácie protokolu, nebolo zavedené žiadne utajenie dát, integrita alebo autentifikačné požiadavky. Predpoklad bol kladený hlavne na nasadenie do privátnych sietí. Keďže na prenášané pakety sa nepoužíva šifrovanie, tak odposluch dát môže útočnikom poskytnúť informácie o aktívnych tokoch, o koncových zariadeniach, ktoré medzi sebou komunikujú a vzorky premávky. Tieto informácie môžu byť použité na špehovanie užívateľov a naplánovanie utajených útokov v budúcnosti. Informácie, ktoré si útočník môže odvodiť z odpočúvania paketov záleží na definícii toku. Ich silná iniciatíva sfalšovať záznamy sa väčšinou môže prejaviť napríklad v bankovníctve. Sfalšované môžu byť šablóny a tak zmiasť kolektor, ktorý nebude môcť dekódovať záznamy, ktoré používajú danú šablónu. Útok DoS (Denial of service) môže zničiť veľa zdrojov z kolektoru, ktorý nebude preto môcť zachytiť a dekódovať nejaké pakety NetFlow. Avšak známe metódy na ochranu serveru od DoS útoku, znižujú tento výskyt problému.

# Kapitola 3

# Nfdump

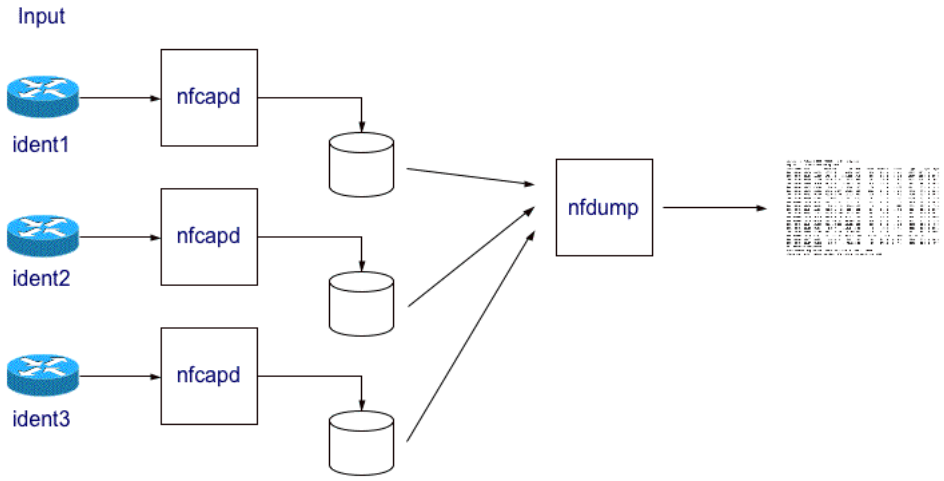
Zámer tejto kapitoly spočíva v úvodom popise programu nfdump a jeho nástrojov, z ktorých sa skladá. Následne bude načrtnutá štruktúra súboru nfdump. Tento nástroj je jadrom tejto práce.

## 3.1 Popis

Aplikácia nfdump (NetFlow dump) je nástroj, ktorý nám umožňuje manipuláciu so záznamami NetFlow. Nechýba ani podpora protokolu NetFlow verzie 9 a momentálne sa testuje podpora IPFIX. Užívateľ je schopný pomocou nfdump nástroja zobraziť a vytvoriť mnoho štatistík. Veľmi vhodné pre administrátorov na zobrazenie záťaži siete. Nfdump je podobný nástroju tcpdump. Hlavným cieľom je analýza dát z minulosti a nepretržité sledovanie premávky na sieti. Medzi hlavné nevýhody patrí to, že je limitovaný kapacitou disku vyhradené pre NetFlow dáta.

- **nfcapd** (NetFlow capture deamon): Ukladá zaznamenané dáta zo siete do súborov. Pre každý NetFlow prúd je potrebný jeden nfcapd proces pre spracovanie.
- **nfprofile** (NetFlow profiler): Spracováva NetFlow dáta zo súborov, ktoré boli uložené nfcapd. Filtruje NetFlow dáta pomocou špecifikovaných filtrov a ukladá filtrované dáta do súborov pre použitie v budúcnosti.
- **nfreplay** (NetFlow replay): Číta uložené dáta a posiela cez sieť inému užívateľovi.
- **nfclean.pl** (cleanup old data): Jednoduchý skript pre vymazanie starých dát.
- **ft2nfdump** (Read and convert flow-tools data): Konvertuje dáta do nfdump formátu, aby mohli byť spracované.
- **nfsen** (NetFlow sensor): Nástroj pre grafickú reprezentáciu NetFlow dát. Používa nfdump ako záložný nástroj. Jeho webové rozhranie je ľahko ovládateľné. Je implementovaný v skriptovacích jazykoch PHP a Perl. Môže byť rozšírený použitím rôznych rozšírení. Vytvára históriu, nastavuje upozornenia založené na rôznych podmienkach.
- **nfanon** (NetFlow anonymization): Používa sa na anonymizovanie všetkých IP adries v zázname NetFlow použitím kryptografie.

Existuje aj nadstavba nad nástrojom nfdump (nfsen), ktorý uložené dáta môže zobraziť v grafickej reprezentácii pomocou grafov. Pre ľahšiu predstavu fungovania nfdumpu je na obrázku 3.1 znázornený jeho princíp a architektúra.

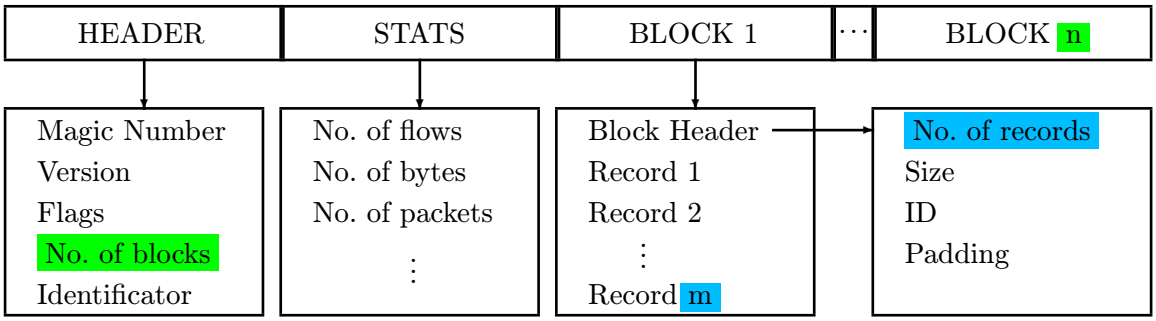



Obrázek 3.1: Princíp a architektúra nfdump [8]

Dáta, ktoré sú zaslané z exportéru na kolektor sa ukladajú pomocou nástroja nfcapd na disk v presne danom formáte. Pomocou nástroja nfdump môžeme tieto uložené dáta čítať a zobraziť v konzole. Keďže je to konzolová aplikácia, jedná sa o pomerne rýchly nástroj. Táto práca je zameraná na prácu s týmto nástrojom a jeho súbormi a preto je potrebné sa zoznámiť podrobnejšie s ich štruktúrou.

## 3.2 Štruktúra súboru

Nástroj nfdump ukladá súbory do binárnej formy v určitej štruktúre. V tejto podkapitole kladiem dôraz na štruktúru daných súborov. Na obrázku 3.2 je znázornena štruktúra súboru.




Obrázek 3.2: Štruktúra súboru nfdump

### 3.2.1 Hlavička súboru (File Header)

Na počiatku každého súboru je hlavička, ktorá určuje či sa jedná o nfdump súbor. Na začiatku hlavičky je magické číslo (Magic number). Jeho hodnota je 0xA50C. Slúži na

kontrolu poradia bajtov (endian) a identifikáciu toho, že sa jedná o nfdump súbor. Verzia (Version) určuje akú štruktúru má súbor. Flagy (Flags) poukazujú na komprimovaný alebo nekomprimovaný súbor. Počet blokov (Number of blocks) je číslo udávajúce koľko blokov sa nachádza v súbore. Každý blok obsahuje niekoľko záznamov. Posledný je identifikátor (Identificator) a slúži na identifikáciu odkiaľ je daný súbor. Dá sa vhodne nastaviť podľa administrátora.

### 3.2.2 Štatistiky záznamov (Stats of Records)

Hneď po hlavičke máme štatistiky všetkých záznamov. Slúžia ako informácia pre administrátorov o danom nfdump súbore. Štatistiky je možné rozdeliť do štatistík tokov, bajtov, paketov, štatistiky o poruchách a úplné štatistiky. Vo všetkých je zahrnutý počet TCP, UDP, ICMP a iných protokolov.

### 3.2.3 Bloky dát (Data Blocks)

Každý blok dát sa skladá z hlavičky, ktorá obsahuje počet záznamov v danom bloku, veľkosť bloku, identifikátor a zarovnanie. Veľkosť jedného bloku nesmie prekročiť viac ako 1 MB. V opačom prípade sa súbor považuje za poškodený. K dispozícii sú dva typy blokov.
**Blok typu 1** je určený predovšetkým pre NetFlow verzie 5 a 7 a obsahuje iba základne informácie o sieti. Záznam obsahuje napríklad položky ako časové značky, flagy a veľkosť záznamu, zdrojový a cieľový port, zdrojový a cieľový autonómny systém, ToS, protokol, TCP flagy, SNMP index vstupného a výstupného rozhrania.
**Blok typu 2** obsahuje nielen tie isté základné informácie zo siete ako nájdeme v bloku typu 1 ale aj rozšírenia pre NetFlow verzie 9. Verzia 9 je v nástroji nfdump implementovaná od verzie 1.6.x. Takto sa otvorili nové možnosti ukladania dát do súboru a uchovávať viac informácii zo siete. Štruktúra NetFlow verzie 9 je daná šablónami a preto umožňuje mnoho kombinácii. Blok typu 2 obsahuje niekoľko typov záznamov. Predovšetkým sa jedná o záznamy typu 1 a 2, ktoré sú rozpísané v ďalších kapitolách. Ostatné záznamy sú v štádiu vývoja. Sú to záznamy určujúce port histogram, bpp histogram, informácie o exportéri a iné.

#### Záznam typu 1 (Common record)

Záznam typu 1 popisuje NetFlow dátový záznam vrátane všetkých voliteľných rozšírení. Každý záznam vyžaduje minimálne prvé tri rozšírenia t.j. IP adresa, počet paketov a počet bajtov. Ostatné položky sú definované v mapách. Štruktúra máp a ich kooperácia so záznamami typu 1 je vysvetlená v ďalších častiach. Tabuľka 3.1 zobrazuje jednotlivé bity flagov. Podľa nultého, tretieho a štvrtého bitu sa zisťuje či záznam obsahuje IPv4 alebo IPv6 adresu komunikujúcich uzlov, adresu ďalšieho zariadenia a adresu ďalšieho zariadenia v BGP. Bity jedna a dva určujú veľkosť počítadla prijatých paketov a bajtov. Posledný bit nám dáva najavo, či sa jedná o vzorkovaný tok alebo nie. Ak záznam obsahuje zdrojovú aj cieľovú IP adresu verzie 4, tak nultý bit je nastavený na hodnotu 0. V opačnom prípade, ak sa jedná o IP adresu verzie 6, tak nultý bit je nastavený na hodnotu 1. Počet paketov je určený prvým bitom. Ak je hodnota prvého bitu rovná 0, tak veľkosť, ktorú zaberá toto pole je 4 bajty. Inak ak je nastavený na hodnotu 1, pole zaberá 8 bajtov. Tak ako aj pre počet paketov, tak aj pre počet bajtov, je nastavenie bitov a zaberanie miesta v súbore rovnaké. Vďaka tomu môžeme odvodiť akú veľkú časť zaberú aspoň niektoré položky v súbore.

| | 0 | 1 |
|---|---|---|
| bit 0 | IPv4 | IPv6 |
| bit 1 | 32 bit Packet counter | 64 bit Packet counter |
| bit 2 | 32 bit Byte counter | 64 bit Byte counter |
| bit 3 | IPv4 Next Hop | IPv6 Next Hop |
| bit 4 | IPv4 BGP Next Hop | IPv6 BGP Next Hop |
| bit 5 | | |
| bit 6 | | |
| bit 7 | unsampled | sampled |

Tabulka 3.1: Flagy

Tabuľka 3.2 znázorňuje základnú štruktúru. Prvý riadok v tabuľke nám určujú bajty. Na začiatku záznamu je typ. Keďže sa jedná o záznam typu 1, tak hodnota tohto poľa je nastavená na 1. Nasleduje veľkosť záznamu, flagy, tagy, identifikátor pre mapu rozšírení. Potom následujú časové značky, status, ktorý určuje či paket bol zahodený, prerušený alebo fragmentovaný. Následne TCP flagy, protokol, zdrojový ToS, zdrojový a cieľový port. Po tomto základe nasledujú ďalšie údaje, ktoré sú nadefinované v mape rozšírení. Pole ext.map nám určuje, ktorú šablónu použiť pre daný záznam. V tabuľke 3.2 je znázornena štruktúra záznamu typu 1.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | record type = 1 | | size | | flags | tag | ext.map | |
| 1 | msec first | | msec last | | first | | | |
| 2 | last | | | | fwd status | tcpflags | proto | src tos |
| 3 | srcport | | dstport/ICMP | | | | | |

Tabulka 3.2: Záznam typu 1

### Záznam typu 2 (Extension Record)

Záznam typu 2 je mapa rozšírení. Vytvára sa na základe toho, aké údaje boli prijaté z exportéra. S množstvom rozšírení a kombinácií rozšírení je viac účinná a flexibilná pri čítaní a dekódovaní záznamov. V poslednej verzii nástroja nfdump je podporovaných maximálne 65535 individuálnych máp, čo je považované za postačujúce. Pre každý prístupný rozšírený záznam sú identifikátory zaznamenané v mape rozšírení v poradí akom nasledujú. Všetky mapy obsahujú jednoznačný identifikátor, veľkosť rozšírení, koľko bajtov zaberajú v súbore a následne pole identifikátorov rozšírení, ktoré sú kľúčovým aspektom. V tabuľke 3.4 je znázornená štruktúra záznamu typu 2.
V najnovšej verzii nfdumpu (1.6.9) je možnosť 49 rozšírení. Z toho prvé tri má každý záznam implicitne a medzi ne patrí IP adresa, počet paketov a počet bajtov. Či sa jedná o IPv4 alebo IPv6 adresu zistíme z políčka flags na začiatku záznamu. V tabuľke 3.1 bolo znázornené ako sa s tým vysporiadáva nfdump. Ďalej nasledujú voliteľné rozšírenia definované v mapách.

Rozšírenia môžeme rozdeliť do troch skupín a to: základné, NSEL[1] a NEL[2] rozšírenia. Pred oboznámením sa s rozšireniami, najskôr rozoberiem Cisco zariadenie ASA[3], jeho popis a základné informácie.
**Cisco ASA** je bezpečnostné zariadenie, ktoré sa skladá z niekoľkých modulov s rôznymi funkciami. Poskytuje obranu proti hrozbám, ktorá zabraňuje útokom predtým, než sa rozšíria cez sieť. ASA môže byť vhodná nie len pre malé ale aj pre veľké siete vďaka jej flexibilite. Pre zhrnutie toto zariadenie je schopné slúžiť ako:

- firewall
- antivirus
- antispam
- prostriedok pre IDS/IPS[4]
- VPN zariadenie
- SSL zariadenie
- zariadenie pre kontrolu obsahu

Tieto vypísané služby nie sú ani zďaleka všetké a bezpečnosť sa môže vylepšiť pridaním modulov, ktoré poskytujú rôznorodé funkcie[7]. Cisco ASA podporuje služby NetFlow verzie 9. Ďalej poskytuje bezpečnostné logovacie mechanizmy, ktorých základom je práve verzia NetFlow.
**NSEL/NEL** je metóda sledovania IP tokov, za pomoci ktorej sa exportujú iba tie záznamy, ktoré indikujú významné udalosti v toku. Tieto toky prechádzajú cez niekoľko stavov. NSEL udalosti sa používajú na export dát o statuse tokov a sú spúšťané udalosťami, ktoré spôsobujú zmenu stavu. Udalosti majú niekoľko funkcií a to vytvorenie toku, rozobranie toku a zahodenie toku. Každý NSEL záznam obsahuje jednoznačný identifikátor udalosti a rozšírený jednoznačný identifikátor udalosti, ktorý popisuje udalosť toku. Implementácia NSEL pre ASA uskutočňuje nasledujúce hlavné funkcie:

- Udržovanie informácii o udalostiach vytvorenia toku, jeho rozloženia a zamietnutia. Na základe toho generuje potrebné NSEL datové záznamy.
- Definuje a exportuje šablony, ktoré popisujú postupnosť toku. Šablona popisuje formát dátového záznamu, ktorý je exportovaný cez NetFlow. Každá udalosť má niekoľko formátov pre záznamy alebo šablony k nim priradené.
- Zaznamenávanie NSEL kolektorov a doručovanie šablón a dátových záznamov k týmto nakonfigurovaným NSEL kolektorom pomocou NetFlow cez protokol UDP.
- Periodické posielanie informácií o šablónach kolektorom. Kolektory prijímajú definície šablón pred prijímaním dátových záznamov, ktoré využívajú tieto šablony.
- Odkladanie události pre vytvorenie tokov.

[1] NetFlow Secure Event Logging
[2] NetFlow Event Logging
[3] Adaptive Security Appliance
[4] Intrusion Detection System/Intrusion Prevention System

- Filtrovanie NSEL udalostí na základe premávky a type události cez Modular Policy Framework. Až po filtrovaní zasiela záznamy kolektorom. Premávka je kontrolovaná na základe toho, v ktorej triede je nakonfigurovaná. Ak je nájdená zhoda, tak ďalšie kontrolovanie nie je potrebné.

Do prvej skupiny rozšírení patria tieto základné:

- **4, 5** (Input and output interface): Vstupné a výstupné rozhranie
- **6, 7** (Autonomous system): Autonómny systém
- **8** (Type Of Service, Direction, Source and Destination Mask): ToS, smer, zdrojová a cieľová maska
- **9, 10** (Next hop IP address): IP adresa následníka IPv4 a IPv6
- **11, 12** (Next hop IP address in BGP): IP adresa následnika v BGP IPv4 a IPv6
- **13** (Source and destination VLAN): Zdrojová a cieľová VLAN
- **14, 15** (Out packets counter): Odchádzajúci počet paketov
- **16, 17** (Out byte counter): Odchádzajúci počet bajtov
- **18, 19** (Aggregated flows): Agregované toky
- **20, 21** (MAC address): MAC adresy
- **22** (MPLS labels): MPLS značky
- **23, 24** (Router IP address): IP adresa routra IPv4 a IPv6
- **25** (Engine type, engine ID / Source ID): Typ toku zariadenia a identifikačné číslo toku zariadenia iba ak sa jedná o verziu NetFlow 5 a pre verziu 9 je identifikačné číslo zdroja
- **26** (BGP next adjacent AS number): Číslo autonomného systému BGP ďalšieho suseda
- **27** (Time flow received in $ms$): Čas toku prijatý v $ms$
- **28 - 36** (Reserved extensions): Rezervované rozšírenia
- **45** (Latency): Latencia

Medzi rozšírenia NSEL patrí:

- **37** (Flow Start, Connection ID, ICMP type and code, Fw. event, Fw. xevent): Začiatok toku, identifikátor pripojenia, ICMP typ a kód, nasledujúce události.
- **38** (Xlate source and destination port): Xlate zdrojový a cieľový port
- **39, 40** (Xlate source and destination IPv4/IPv6 address): Xlate zdrojová a cieľová IPv4/IPv6 adresa
- **41** (Ingress and Egress ACL ID): Indetifikátor vstupného a výstupného ACL
- **42, 43** (Username): Užívateľské meno
- **44** (Reserved extension): Rezervované rozšírenie

Medzi NEL rozšírenia môžeme zaradiť:

- **46** (NAT event, Flags, Post source and destination port, Ingress vrf ID): Udalosť NAT, flagy, zdrojový a cieľový port
- **47, 48** (Inside and outside NAT): vstupné a výstupné NAT
- **49** (Reserved extension): Rezervované rozšírenie

Rezervované rozšírenia slúžia pre budúce účely. Veľkosti jednotlivých rozšírení sú rôzne. Niektoré položky by nemuseli zaberať toľko miesta na disku, no z implementačného hľadiska a pre ľahšiu manipuláciu s dátami je to výhodné. Vhodným príkladom sú MAC adresy kde veľkosť MAC adresy je 6 bajtov, no v súbore je táto položka uložená ako 8 bajtová.

| **2 Bajty** | **6 Bajtov** |
|---|---|
| 0 | MAC adresa |

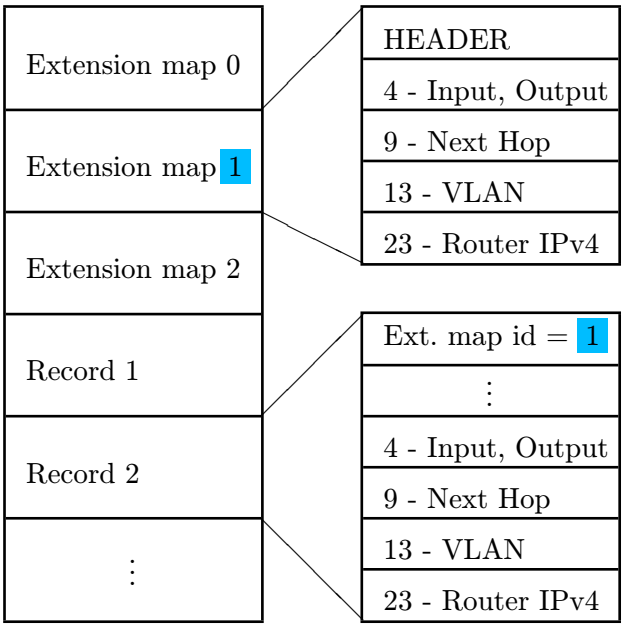
Tabulka 3.3: Uloženie MAC adresy v súbore

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | record type = 2 | | size | | map id | | extension size | |
| 0 | extension id 1 | | extension id 2 | | extension id 3 | | extension id 4 | |
| ⋮ | | | | | | | | |
| 0 | extension id n | | extension id n+1 | | extension id n+2 | | extension id n+3 | |
| ⋮ | | | | | | | | |
| 0 | 0 | | opt. 32 bit alignment | | | | | |

Tabulka 3.4: Záznam typu 2

Na obrázku 3.3 je znázornené ako môže vyzerať blok typu 2. Spočiatku sú nadefinované mapy a následne záznamy. Pomocou identifikačného čísla mapy, vieme zistiť aké ďalšie rozšírené položky obsahuje daný záznam. Mapy a obyčajné záznamy môžu byť poprehadzované. Ak záznam je určený mapou, ktorá je definovaná až za ním, tak nfdump vyhodí chybové hlásenie a záznam sa preskočí. Preskočenie záznamu je možné vďaka informácii o jeho veľkosti, ktorá sa nachádza v jeho hlavičke.

Z implementačného hľadiska nástroj nfdump podporuje iba tieto rozšírenia. Vo verzii NetFlow 9 je možné zachytiť aj ďalšie ako sú napríklad minimálne a maximálne TTL (Time to live) prichádzajúcich paketov v toku, skrátený a celý názov rozhrania odkiaľ daný tok je, alebo názov vzorkovača tokov. Vývojom sa možno tieto ďalšie položky doplnia, a tak bude protokol plne využitý týmto nástrojom.




Obrázek 3.3: Blok typu 2 a použitie máp

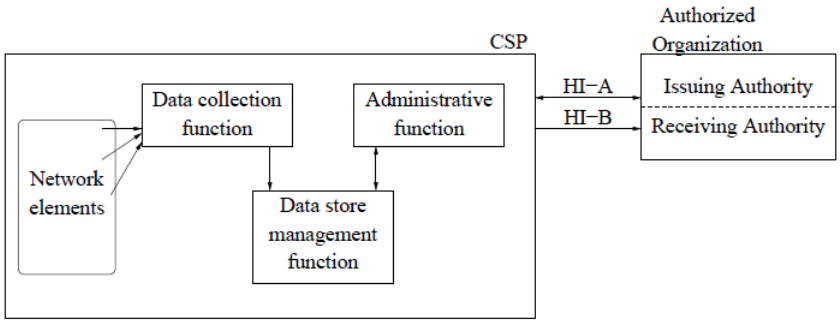
# Kapitola 4

# Sieť VUT

Táto kapitola popisuje architektúru siete VUT a jej monitorovacieho systému pre IPv4 a IPv6 adresy. Načrtnutý bude dizajn systému pre uchovávanie dát v IPv6 sieti a vysvetlené použité nástroje, ktoré medzi sebou spolupracujú. Kapitola neobsahuje všetke informácie o sieti VUT, preto podrobnejšie informácie je možné nájsť na [5].

## 4.1 Navrhnutá architektúra

Navrhnutá architektúra je postavená na štandarde ETSI, ktorý definovaný v dokumente TS 102 657. Architektúra pre zachytávanie dát v tejto kapitole je rozobraná iba povrchne. Na najvyššej vrstve ETSI definuje dve komunikačné entity CSP[1] a AO[2]. Medzi nimi sú dve komunikačné linky. Jedna v smere od AO k CSP, kde sa doručujú správy administratívy požiadavok/odpoveď. V opačnom smere sa prenášajú odchytené dáta. Dôraz bude kladený hlavne na entitu CSP, ktorá sa skladá z niekoľkých blokov. Na obrázku 4.1 je znázornený tento model, kde HI-A a HI-B značia komunikačné kanály. Do funkčných blokov CSP patrí (Administratívna funkcia (AF[3]), funkcia pre riadenie ukladania dát (DSMF[4]), funkcia zberu dát (DCF[5])).



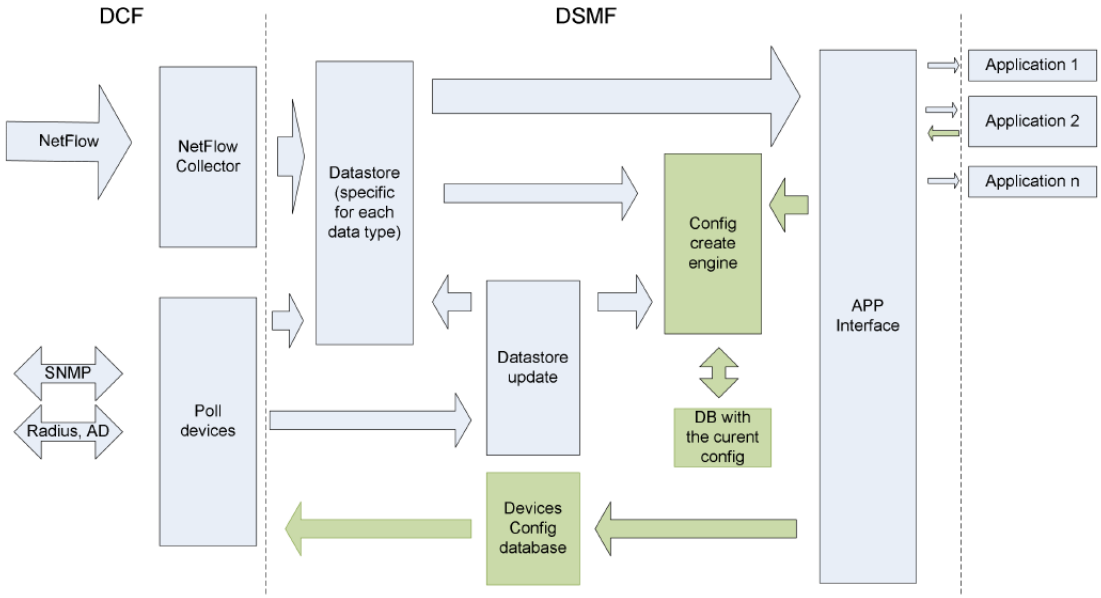

Obrázek 4.1: CSP a AO model [5]

Administratívna funkcia implementuje oba HI kanály a vnútorné rozhranie pre zhroma-

[1] Communication Service Provider
[2] Authorized Organization
[3] Administrative Function
[4] Data Storage Management Function
[5] Data Collection Function

ždenie zachytených dát. Úlohou je prijať a potvrdiť požiadavok pre zachytené dáta, transformovať a vydať požiadavky, oznámiť stav odchádzajúcich požiadavkov a nakoniec zaslať výsledok požiadavkov ako zachytené dáta cez kanál HI-B. Funkcia zberu dát zbiera dáta z rôznych vnútorných sieťových elementov a pripravuje dáta na ich uchovanie. To zahŕňa synchronnú a aj asynchronnú komunikáciu so sondami, prepínačmi, smerovačmi, servermi (DHCP, DNS, RADIUS) a databázou s užívateľmi. Tento proces indexuje a ukladá dáta, vykonáva požiadavky a riadi periódu zadržiavania.

V sieti VUT je navrhnutý monitorovací systém, ktorý spĺňa požiadavky aj na IPv6 sieť. To vyžaduje rôzne zdroje zachytených dát, ich spracovanie, uchovávanie a navrhnuté rozhranie. Mapovanie jednotlivých blokov štandardu ETSI je znázornené na obrázku 4.2. DCF zahŕňa tri zdroje dát rôzneho typu na vstupe do systému. Prvý zdroj ustanovuje vygenerované NetFlow data sondami a smerovačmi v sieti. NetFlow data sú zozbierané a uložené. Druhý zdroj dát je zo smerovačov a prepínačov pomocou protokolu SNMP. Číta dostupné premenné v každom zariadení založené na jeho MIB strome a ukladá tieto dáta do databázy. Tretím a zároveň posledným zdrojom dát sú správy udalostí a zázmami zo serverov (DHCP, RADIUS, ...). Tieto dáta sú rozobrané a extrahované informácie sú uložené taktiež do databázy. Funkcia DSMF je spojenie dát z databázy a uloženými NetFlow dátami. Konfiguračné rozhranie dovoľuje kontrolu nad DCF a DSMF procesmi. Dátové rozhranie spracováva požiadavky na uložené dáta generované rôznymi aplikáciami, ktoré patria do administratívnej funkcie. V sieti sú zostavené nástroje, ktoré implementujú celý monitorovací systém podľa navrhnutej architektúry.




Obrázek 4.2: Navrhnutá architektúra monitorovacieho systému [5]

### Data Collection Function (DCF)

ETSI rozdeľuje zozbierané dáta do nasledujúcich kategórií:

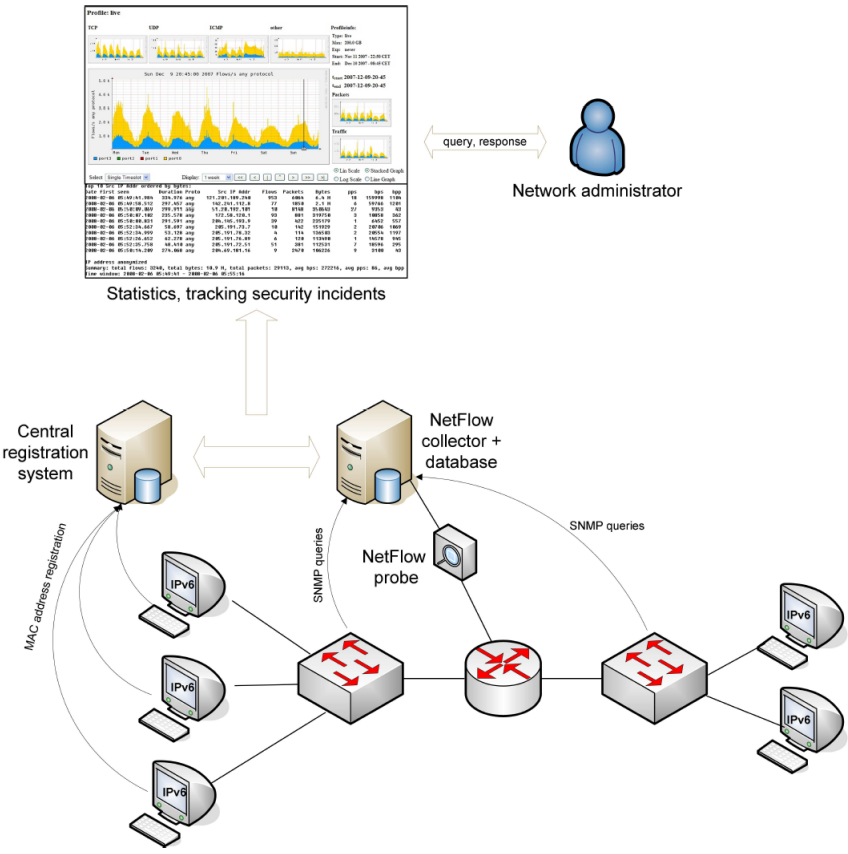
- Dáta odoberateľa: Informácie o zariadení odoberateľa napríklad z užívateľskej databázy (meno, adresa, identifikátor, ...)
- Použité dáta: Informácie o použití služieb napríklad z protokolov NetFlow, IPFIX, SIP proxy (toky záznamov, SIP záznamy, ...)
- Dáta zo zariadení: Informácie o zariadeniach koncových užívateľov napríklad z prepínačov, DHCP, RADIUS serverov (MAC adresa, operačný systém, ...)
- Sieťové dátové elementy: Informácie o komponentách v sieťovej infraštruktúre napríklad zo sieťových elementov SNMP (lokácia a identifikátor pristúpového bodu, štatistiky z rozhraní, ...)
- Ďalšie používané služby: Informácie použitých služieb napríklad z aplikačných serverov (SMTP, IMAP, ...)

DCF musí byť schopné zberať dáta z rôznych rozhraní a protokolov ako napríklad syslog, SNMP, NetFlow/IPFIX. Niektoré elementy pracujú asynchronne (vypršanie časového intervalu, koniec toku a iné) a niektoré synchronne (nutnosť aktívneho dotazovania). Výstupom DCF sú dáta pripravené na uchovanie. Táto časť monitorovacieho systému je komplexná z dôvodu použitia nástrojov a zariadení od rôznych výrobcov. Pre uchovávanie sieťových elementov sa využíva systém NAV (Network Administration Visualized). Je voľne distribuovateľný pod licenciou GNU GPLv2 a podporuje oba protokoly IPv4 a IPv6. Dokáže prijmať SNMP správy od rôznych výrobcov a ukladať záznamy a správy z RADIUS servera. Rozhodujúcu časť systému uchovávania dát tvorí asociácia užívateľova IP, MAC adresa a číslo portu prepínača na ktorý je užívateľ pripojený. To umožňuje zistiť komu patrí daná IP adresa, aká je jeho poloha a aké adresy užívateľ používa. SNMP sa používa k získaniu dát každých 15 minút z prepínača. Mapovanie medzi IPv6 adresou a korešpondujúcou MAC adresou je stiahnuté z pamäte (neighbor cache) smerovača. Číslo VLAN, port a ďalšie iné informácie poskytuje prepínač v svojej FDB (Forwarding Database) tabuľke. Zozbierané informácie z ARP (MAC - IP adresa), CAM (MAC - port) a Radius servera (užívateľske meno - MAC adresa, užívateľske meno - IP adresa) nám umožňujú zachovať vzťahy medzi IP, MAC adresou a číslom portu. Je to veľmi dôležité z hľadiska IPv6, ktorá sa pravidelne generuje a mení. Naviac k tomu sú časové značky, ktoré určujú interval ich validnosti. Je to dôležité s vysporiadaním sa požiadavkov zachádzajúcich do minulosti. Pretože NAV nezbiera informácie o provoze na siete bol na zhromažďovanie metadát použitý protokol NetFlow. Obrázok 4.3 znázorňuje centrálny monitorovací systém na sieti VUT.

Protokol NetFlow posiela záznamy ako IP toky zo sónd na kolektor. NetFlow záznam obsahuje identifikátor toku a štatistiky. Veľkosť uložených dát závisí na kompozícii a type provozu. V priemere na 100 Mbps linke sa nazbiera 300 MB dát za hodinu a na 1 Gbps je to 600 MB za hodinu. Dôležitá časť kolektoru je jeho miesto vyhradené na NetFlow dáta, rýchlosť čítania a zápisu.

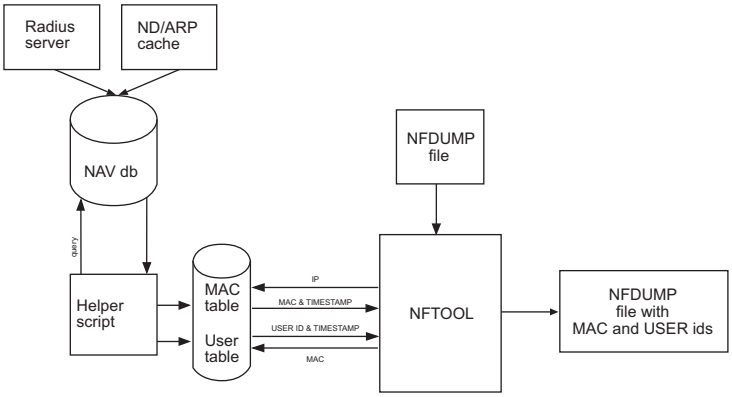
### Data Storage Management Function (DSMF)

DSMF zriaďuje procesy, ktoré pracujú s uloženými dátami a vytvára rozhranie pre administratívnu funkciu. Na zachytávanie NetFlow dát sa používa nástroj nfcapd a nfdump na spracovanie a pristupovanie k dátam. Cieľom tejto časti systému je rozšíriť dáta o MAC adresy a užívateľské mená. K dispozícii sú dva prístupy importovania mien užívateľov do nfdump súboru. Jedným je využitie políčok, ktoré sa nevyužívajú (napr. MPLS) a druhým je




Obrázek 4.3: Centrálny monitorovací systém pre IPv4 a IPv6 na sieti VUT [5]

doplniť nfdump o možnosť pracovať s týmito dátami a vytvoriť patch. Prvý prístup je kompatibilný s grafickým užívateľským prostredím nfsen, no druhý túto možnosť neposkytuje. Obrázok 4.4 zobrazuje činnosť tohoto systému.




Obrázek 4.4: Framework pre import mien užívateľov do NetFlow [5]

NAV databáza sa plní informáciami z radius servera a ND/ARP cache. Potom pomocný skript, ktorý je napísaný v programovacom jazyku perl sa spúšťa každú hodinu pomocou cronu aby vytvoril dve Berkeley databáze s tabuľami pre IP - MAC adresu a MAC - meno užívateľa. Tento skript sa dotazuje na NAV a takto napĺňa tieto tabuľky. K dispozícii sú aj

časové značky aby sa dalo určiť, ktorý užívateľ komunikoval práve v daný čas. Nástroj nftool sa spúšta taktiež každú hodinu až po pomocnom skripte aby mal plne k dispozícii naplnené tabuľky. Nftool je napísaný v programovacom jazyku C z dôvodu veľkého množstva dát a operácii nad nimi. Jeho vstupom je binárny súbor nfdump a na základe získaných informácii z NAV k príslušným IP adresam priraďuje MAC adresy a následne k danej MAC adrese priraďuje meno užívateľa.

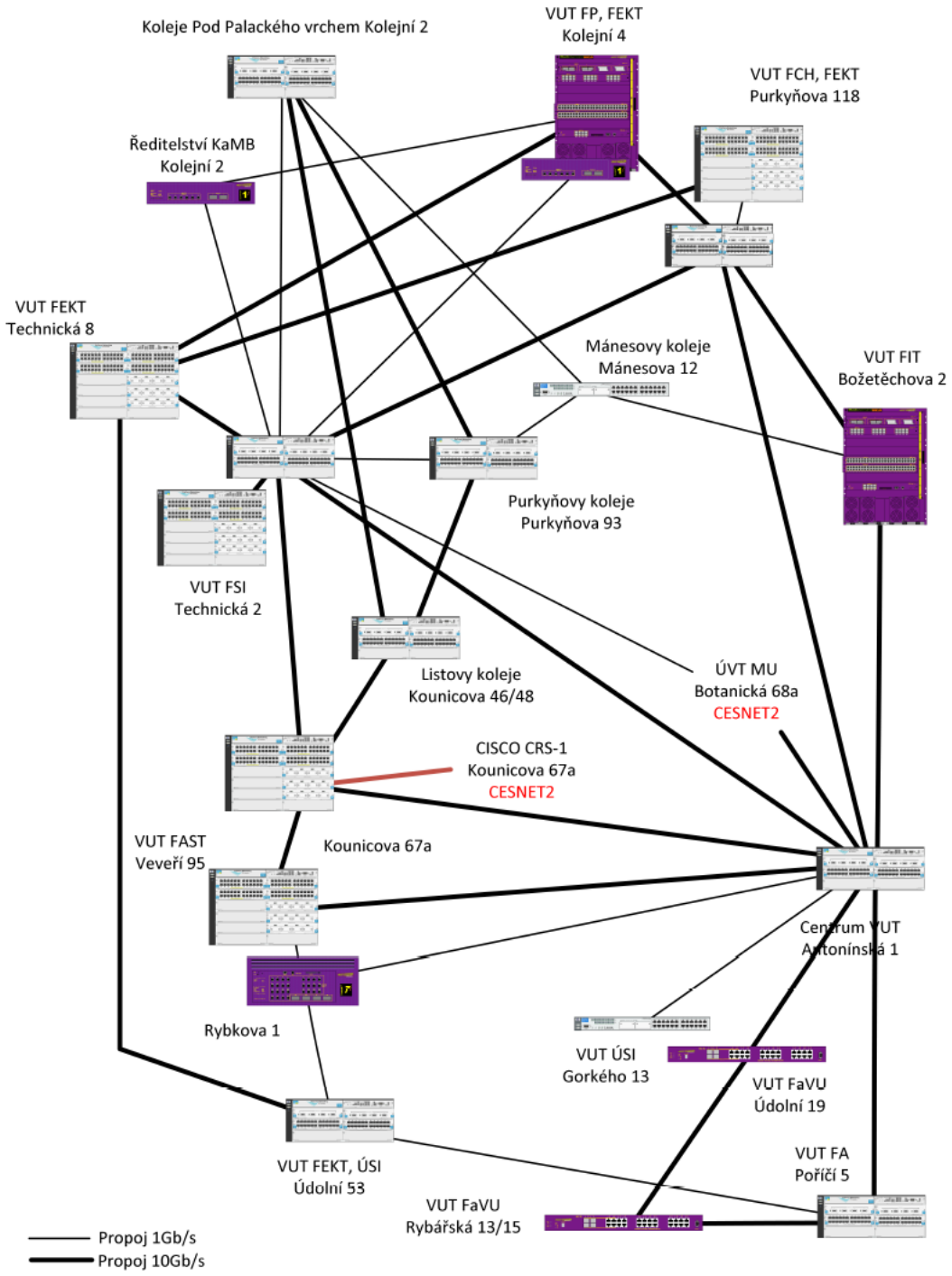
#### Administrative Function (AF)

Hlavnou úlohou administratívnej funkcie je implementovať rozhranie pre zadržané dáta. Administratívna funkcia je interpretovaná serverom a je napísaná v jazyku PHP. Dotazy smerujú na aplikáciu nfdump a na základe získaných informácií sa vo webovom rozhraní zobrazia korešpondujúce dáta.

## 4.2 Fyzická realizácia

V tejto podkapitole sú rozobrané problémy IPv4 a IPv6 monitorovania. Sieť VUT obsahuje 134 aktívnych smerovacích zariadení na hlavnej sieti a tisíce pripojených užívateľov (vo väčšine študentov). Kampusová sieť prepája niekoľko inštitúcii ako univerzitné fakulty, inštitúcie zamerané na výzkum, Českú akadémiu a vysoké školy na 20 miestach v rôznych častiach Brna. Každá lokácia je prepojená s najmenej dvoma optickými káblami z dvoch nezávislých smerov pre dosiahnutie maximálnej spoľahlivosti na sieti. Celková dĺžka optickej kabeláže dosahuje cez $100km$. Jadro celkovej architektúry je znázornené na obrázku 4.5.

Pre vnútorné smerovanie je použitý protokol OSPF a OSPFv3. Externé smerovanie zabezpečuje BGP a BGP+. Sieť VUT univerzitného kampusu spája viac ako 2500 zamestnancov a viac ako 23000 študentov. Najväčšie využitie je na študentských internátoch kde je viac ako 6000 študentov pripojených cez 100 Mb/s a 1Gbps linkami. Celý kampus plne využíva IPv6 konektivitu. Systém pre uchovávanie dát zo siete s informáciami o IP a MAC adresách pripojených užívateľov využíva ARP tabuľky FDB tabuľky zo všetkých smerovacích prepínačov, ktoré sú lokalizované v rôznych budovach VUT kampusu. Každý z týchto prepínačov slúži ako brána danej budovy. V sieti sa využíva systém NAV (Navigation database), ktorý každých 15 minút komunikuje s prepínačmi či nenastala nejaká zmena. Ak zmena nastala, tak je zaznamenaná do NAV databázy. Tento systém je na vyhradenom zariadení, ktorý poskytuje iba túto databázu. Základné NetFlow dáta sú získané z troch monitorovacích sónd, ktoré sú uložené v rôznych častiach siete kampusu. Dve z nich zachytávajú dáta v smeroch medzi kampusom a internetom. Tretia sonda je na študentských internátoch, kde je provoz najviac koncentrovaný. Tieto sondy sú importované rovno do prepínačov. To umožní spracovanie dát priamo na základnej doske prepínača. Dáta získané zo sónd sú exportované a ukladané na jeden NetFlow kolektor. Kolektor vyberá informácie z NAV zariadenia a spája ich s NetFlow dátami.




Obrázek 4.5: Topológia VUT siete [5]

# Kapitola 5

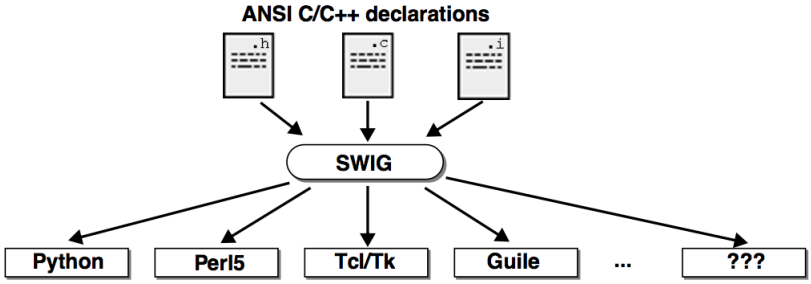
# Návrh

Táto kapitola je venovaná návrhu aplikačného rozhrania pre prácu s dátami NetFlow. V dnešnej dobe prístup vysokoúrovňového jazyka k funkciam nízkoúrovňového nie je žiadnou novinkou. Existuje mnoho technológii a prístupov. Nie však všetké majú rovnaké vlastnosti a preto popíšem len tie, ktoré sú najviac používané. Zameral som sa hlavne na skriptovacie jazyky Python a Perl. Spočiatku bližšie vysvetlím ich popis a pre lepšiu predstavu uvediem nejaké jednoduché príklady. Táto kapitola bude najviac venovaná technológii SWIG z toho dôvodu, že bola vybraná ako najlepšia voľba pre toto aplikačné rozhranie. Budem ju porovnávať hlavne s ďalšou technológiou XS.

## 5.1 SWIG

SWIG (Simplified Wrapper and Interface Generator) je softwarový vývojarský nástroj na vytvorenie rozhrania pre skriptovací jazyk do C a C++ programov. Bol vytvorený v roku 1995 a bol prvykrát použitý vedcami pre vybudovanie užívateľského rozhrania do simulačného kódu bežiaceho na superpočitači. V tomto prostredí potrebovali vedci pracovať s veľkým množstvom simulovaných dát, komplexným hardwareom a konštantne sa meniacom kóde. SWIG zjednodušuje vývoj veľkým automatickým integráciam úloh skriptovacích jazykov. SWIG bol hlavne vyvíjaný pre vedecké aplikácie ale v dnešnej dobe tvorí silný nástroj a je využívaný aj k všeobecnému použitiu v rôznych aplikáciách.
Stabilná verzia bola vydaná v neskorších 90-tych rokoch a prešla viacerými vylepšeniami a rozšíreniami. Oficiálna stabilná verzia bola vydaná s rozhodnutím zmeny licencie a v roku 2010 vyšla verzia 2.0.0. Vďaka novej licencii mohol kód, ktorý SWIG vygeneroval, byť distribuovaný pod licenčnými podmienkami užívateľových rozhodnutí a požiadaviek. V rovnakom čase bol SWIG pod licenciou GNUv3.
Medzi hlavné prerekvizity patrí znalosť programovacích jazykov C alebo C++ a nejaká znalosť skriptovacích jazykov ako napríklad Tcl, Python a Perl. Nie sú potrebné skúsenosti s prekladom C rozšírení do týchto jazykov (SWIG to vykoná automaticky). No je potrebná znalosť používania kompilátorov, linkerov a makefileov. Časom sa SWIG stal viac schopný zaobchádzať a pracovať s C++ (podpora rozšírených možností ako namespace, preťaženie operátorov a šablony). Nedá sa predpokladať a spoliehať, že SWIG poskytuje spätnú kompatibilitu hoci je o to snaha. Ak je potrebné využiť spätnú kompatibilitu, tak je možné použiť preprocesor symbol *SWIG_VERSION*, ktorý obnáša verziu aká sa má použiť.
V celku je SWIG kompilátor, ktorého vstupom sú C/C++ deklarácie. Zároveň vytvára spojenia potrebné na prístup k týmto deklaráciam z iného jazyka (Perl, Python, Tcl, Ruby,

Guile a Java). Na obrázku 5.1 je znázornený globálny princíp fungovania SWIGu.




Obrázek 5.1: Princíp SWIGu

SWIG nevyžaduje žiadne modifikácie existujúceho kódu a môže byť používaný na vytvorenie použiteľného rozhrania. Možné aplikácie SWIGu zahŕňajú:

- Budovanie interpretovaného rozhrania do existujucích programov napísaných v programovacom jazyku C
- Rapídne prototypovanie a vývoj aplikácií
- Interaktívne ladenie programov
- Preriadenie alebo refaktorácia staršieho softwaru do komponent skriptovacieho jazyka
- Tvorba grafického rozhrania
- Testovanie C knižníc a programov
- Budovanie vysokovýkonných modulov pre skriptovacie jazyky, napísané v programovacom jazyku C
- Viac pôžitku z programovania v jazyku C

SWIG je v dnešnej dobe používaný vo veľkých open source alebo komerčných projektoch. Ako už bolo spomenuté, primárnym cieľom SWIGu je zjednodušovanie úloh integrácie C/C++ s ostatnými programovacími jazykmi. Odpoveď na otázku prečo by so SWIGom mal niekto pracovať, je jednoduchá. Dôvodom je týchto pár bodov ohľadom programovania v C/C++:

- Výborná podpora písania programových knižníc
- Vysoká výkonnosť (spracovanie dát, grafika, ...)
- Systémové programovanie a systémová integrácia
- Veľká užívateľská komunita a softwarový základ

Samozrejme, všetko čo má svoje výhody, má aj svoje nevýhody a preto medzi hlavné problémy patria:

- Programovanie užívateľského rozhrania
- Testovanie nie je časovo úsporné (odstraňovanie chýb)

- Nie je jednoduché prekonfigurovanie a upravenie bez ďalšieho prekladu
- Modularizácia môže byť nespoľahlivá
- Znepokujúca ochrana (pretečenie zásobníka)

Na základe týchto obmedzení sa došlo k záveru, že je oveľa ľahšie používať rôzne programovacie jazyky na rôzne úlohy. Napríklad naprogramovanie grafického užívateľského rozhrania je značne jednoduchšie v skriptovacom jazyku ako napríklad Python alebo Tcl. Interaktívny interpreter, môže taktiež veľmi dobre slúžiť ako ladiaci a testovací nástroj. Ostatné jazyky ako napríklad Java, by mohla veľmi dobre zjednodušiť úlohy pre programovanie distribuovaných systémov. Ide vlastne o to, že rôzny programovací jazyk poskytuje rôzne silné a slabé stránky. Kombináciou jazykov dokopy je možné využiť najlepšie funkcie každého jazyka a značne zjednodušiť isté aspekty softwareového vývoja. Z pohľadu C/C++ veľa užívateľov používa SWIG, pretože chcú prelomiť tradičný monolitický programovací model v C, ktorý zvyčajne obsahuje kolekciu funkcií a premenných. Tie spolu tvoria nejakú funkčnú časť (funkciu `main()`), ktorá je štartovacím bodom. Kooperácia s vyšším programovacím jazykom dodáva modulárny dizajn, menej kódu, lepšiu flexibilitu a zvýšenú produktivitu.
SWIG sa snaží o zjednodušenú integráciu do vyššieho programovacieho jazyka. Poskytuje široké možnosti úprav, ktoré dovoľujú zmenu skoro všetkých aspektov spájania jazykov.
Hlavným cieľom projektu SWIG je spraviť proces spojenia čo najjednoduchšie. SWIG je schopný a podporuje väčšinu vlastností jazyka C/C++. Medzi niektoré hlavné patria:

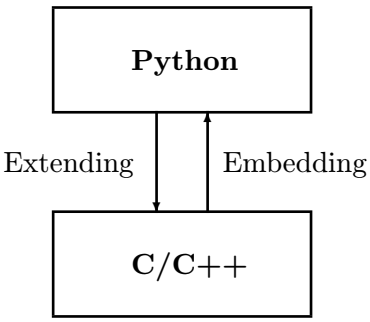
- Plné spracovanie preprocesorom C99
- Všetké ANSI C a C++ dátové typy
- Funkcie, premenné a konštanty
- Triedy
- Jednoduchá a viacnásobná dedičnosť
- Preťažené funkcie a metódy
- Preťažené operátory
- C++ šablóny
- Namespace
- Premenná dĺžka argumentov
- C++ ukazatele

Momentálne najväčšou vlastnosťou, ktorá nie je podporovaná sú vnorené triedy. Preto je predpoklad, že pri najbližšej verzii bude tento nedostatok odstránený. Dôležité je, že SWIG nie je zjednodušený C++ lexikálny nástroj ako niektoré podobné spájacie všeobecné nástroje. SWIG nielen, že parsuje C++ ale aj implementuje celkový typový systém a je schopný porozumieť sémantike jazyka C++. Generuje ich spojenia s plnou znalosťou tejto informácie.
SWIG vyžaduje minimálne alebo dokonca žiadne modifikácie existujúceho C/C++ kódu. Preto je jednoduché použitie už existujúcich balíčkov. Taktiež podporuje aby software bol

znovupoužiteľný a modulárny. Aby bol kód v C/C++ nezávislý na vysokoúrovňovom rozhraní, môže sa zmeniť dané rozhranie a použitie kódu znovu v inej aplikácii. Možná je taktiež podpora rôznych typov rozhrania závislého na aplikácii.

### 5.1.1 SWIG a Python

Ako vlastne komunikuje skriptovací jazyk s nízkoúrovňovým C? Skriptovacie jazyky boli stavané okolo parsera, ktorý vie ako vykonávať príkazy a skripty. V jeho vnútri je akýsi mechanizmus na vykonávanie príkazov a pristupovanie k premenným. Avšak rozšírením interpretera je zvyčajne možné pridať nové príkazy a premenné. K tomu mnoho jazykov definuje špeciálne API na pridávanie príkazov. Okrem toho, špeciálne rozhranie definuje ako tieto príkazy zakomponovať do interpreteru.
Ak sa pridá nový príkaz do skriptovacieho jazyka, je potrebné spraviť dve veci. Za prvé je potrebné napísať špeciálnu spájaciu funkciu, ktorá slúži ako spoj medzi interpreterom a C funkciou. Potom je potrebné dať interpreteru informáciu o spájaní ako sú detaily o názve funkcie, argumentov a iné. Tento proces bude ilustrovaný v nasledujúcich častiach, zameraný na skriptovací jazyk Python.
K dispozícii sú dve metódy integrácie C/C++ s Pythonom: Rozširovanie (Extending) a vkladanie(Embedding). Rozširovanie umožňuje Pythonu prístup do C/C++ a vkladanie prístup z C/C++ do interpretu Pythonu. Na obrázku 5.2 je znázornená interakcia týchto jazykov.




Obrázek 5.2: Interakcia jazykov

Koncetrácia bude hlavne na rozširovanie, pretože cieľom práce je aplikačné rozhranie v Pythone, ktoré komunikuje s C/C++ programom.
Na prístup do jazyka C/C++ sú potrebné spájacie funkcie. Tie slúžia ako vrstva medzi jazykmi. Je potrebné konvertovať argumenty funkcie z Pythonu do C a vrátiť výsledok vo forme, ktorej Python porozumie. Ako príklad si uvedieme nasledujúcu funkciu v C. Jedná sa o rekurzívnu funkciu, ktorá počíta faktoriál čísla zadaného na vstupe.

```
1  double Foo = 10.0;
2
3  int factorial(int n) {
4          if (n <= 1) return 1;
5          else return (n * factorial(n - 1));
6  }
```

Najskôr uvediem ako vyzerá celkové rozšírenie a následne to rozoberiem viac do detailov.

```
#include <Python.h>

PyObject *wrap_factorial(PyObject *self, PyObject *args) {
        int n, result;
        if (!PyArg_ParseTuple(args, "i:factorial", &n))
                return NULL;
        result = factorial(n);
        return Py_BuildValue("i", result);
}

static PyMethodDef exampleMethods[] = {
        { "factorial", wrap_factorial, 1 },
        { NULL, NULL }
};

void initexample() {
        PyObject *m;
        m = Py_InitModule("example", exampleMethods);
}
```

Spájacia funkcia do Pythonu, k predchádzajúcej funkcii je funkcia `wrap_factorial`. Vstupom pre túto spájaciu funkciu je zoznam argumentov, ktoré prechádzajú z Pythonu do C. Funkcia `PyArg_ParseTuple` prijíma na vstup tieto argumenty, nadefinuje sa ich formát (o aké typy sa jedná) a priradia sa hodnoty k nasledujúcim adresám premenným. Návratová hodnota spájacej funkcie je výsledok funkcie `Py_BuildValue`, ktorá robí opačnú úlohu ako funkcia `PyArg_ParseTuple`.
Každý modul rozšírenia potrebuje registrovať spojenia s Pythonom. Inicializačná funkcia je volaná kedykoľvek, keď je importovaný modul rozšírenia. Inicializačná funkcia registruje nové metódy s interpreterom Pythonu. V našom prípade môže vyzerať inicializačná funkcia ako funkcia `initexample`. Tabuľka metód, ktoré budú využívané, je definovaná medzi inicializačnou a spájacou funkciou.
Preklad Python rozšírenia je možný dvoma spôsobmi. Dynamicky (Dynamic Loading) a staticky (Static linking). Dynamicky znamená, že sa modul preloží do zdieľanej knižnice alebo do DLL. Potom v Pythone stačí už len nainicializovať modul za behu pomocou kľúčového slova import. Ak sa prekladá staticky, tak modul je preložený do jadra Pythonu a vznikne ako nový "built-in"modul. A tak ako aj pri dynamickom, tak aj pri statickom stačí pomocou import ho nainicializovať.

```
// example.i

%module example                 // Meno modulu
%{
        #include "headers.h"    // Hlavičky
%}

int factorial(int n);           // C deklarácie
double Foo;
#define IP 6
```

Preklad na rôznych operačných systémoch sa nemusí zhodovať. Na systéme Linux môže vyzerať nasledovne.

```
% swig -python example.i
% cc -c example.c example_wrap.c -I /usr/include/python2.7
% ld -shared example.o example_wrap.o -o _example.so
```

SWIG vyprodukuje súbor `example_wrap.c`, ktorý je kompilovaný do Python modulu. Meno modulu a zdieľanej knižnice by sa mal zhodovať. Jeho jednoduché použitie je znázornené v nasledujúcom príklade v Pythone. V prvom rade sa naimportuje modul, ktorý chceme používať a následne za pomoci modulu sa volajú potrebné funkcie.

```
1 $ python
2 >>> import example
3 >>> example.factorial(4)
4 24
5 >>> print example.cvar.Foo
6 10.0
7 >>> print example.IP
8 6
```

Úloha SWIGU spočívala v tom, že C funkcie (príkazy) sa stali funkciami Pythonu, globálne premenné sa stali atribútom špeciálneho objektu `cvar` v Pythone a konštanty sa stali premennými. V nasledujúcej tabuľke je zobrazené mapovanie dátových typov z C do Pythonu.

| **C** | **Python** |
|---|---|
| int<br>long<br>short | integer |
| float<br>double | float |
| void | None |
| long long<br>long double | nepodporované |

Tabulka 5.1: Mapovanie dátových typov C - Python

K dispozícii sú štyri direktívy pre vkladanie kódu.

- %{ ... %} vkladá kód do sekcie hlavičiek (Headers)
- %wrapper %{ ... %} vkladá kód do spájacej sekcie (Wrappers)
- %init %{ ... %} vkladá kód do inicializačnej sekcie (Initialization)
- %inline %{ ... %} vkladá kód do sekcie hlavičiek a spája ho

SWIG umožňuje zakomponovať do jedného súboru rozhrania (interface file) aj ďalšie a tak vytvoriť jeden kompletný celok. Slúži nám k tomu direktíva %include. Je možné tieto menšie celky zakomponovať aj do iných súborov preto je to flexibilné. Ďalšou možnosťou sú už vopred nadefinované knižnice SWIGu ktoré obsahujú definíciu rozhrania do C knižníc, funkcie na vytvorenie napríklad polí a iné (malloc.i, pointer.i, timers.i, ...). Ako príklad uvediem súbor `malloc.i`. Štruktúra súboru bola rozobraná v predchádzajúcich častiach tejto kapitoly.

```
1  // malloc.i
2
3  %module malloc          // Meno modulu
4  %{
5          #include <stdlib.h>     // Hlavičky
6  %}
7
8  void *malloc(unsigned int size);
9  void *calloc(unsigned int nobj, unsigned int size);
10 void *realloc(void *ptr, unsigned int size);
11 void free(void *ptr);
```

Medzi pokročilejšie funkcie SWIGu patria typové mapy (Typemaps), ktoré umožňujú zmeniť proces hocijakého dátového typu (zaobchádzanie so vstupnými a výstupnými hodnotami, konvertovanie Python objektov do ekvivalentých v C/C++ a mnoho ďalších).
V ďalšej sekcii rozoberiem technológiu XS, ktorá je asi najväčšou konkurenciou pre SWIG.

## 5.2 XS

XS popisuje formát rozhrania súboru, ktorý sa používa na vytvorenie na rozšírenie rozhrania medzi skriptovacím jazykom Perl a nízkoúrovňovým kódom jazyka C (alebo C knižníc), ktoré programátor chce využívať v jazyku Perl. Rozhranie XS je kombinované s knižnicami na vytvorenie novej knižnice, ktorá následne môže byť nalinkovaná nielen dynamicky, ale aj staticky do Perlu. Popis Perl rozhrania XS je napísané v XS jazyku, ktorý tvorí jeho jadro. **XSUB** vytvára základnú jednotku pre XS rozhranie. Po preklade prekladačom **xsubpp** každý XSUB (XS subroutine) činí definíciu funkcie v C, ktorá uskutočňuje spojenie medzi prechodmi volania v jazyku Perl a v C.
Spájací kód vyberá argumenty z Perl zásobnika, konvertuje tieto hodnoty Perlu do formátu očakávaného funkciou v C. Volanie tejto C funkcie prekladá návratovú hodnotu C funkcie naspäť do Perlu. Návratová hodnota môže byť konvenčná návratová hodnota v C alebo nejaký jej argument, ktorý môže slúžiť ako výstupný parameter. Tieto návratové hodnoty môžu byť vrátené naspäť do Perlu tiež vložením do Perl zásobnika alebo modifikovaním argumentov dodané zo strany Perlu. Doterajší popis bol iba základný. Perl dovoľuje viac flexibilné volanie konvencie ako C. XSUB môže v praxi toho robiť oveľa viac ako napríklad kontrolovanie validity vstupných parametrov, vyhodenie výnimiek ak návratová hodnota z funkcie C indikuje zlyhanie, volanie rôznych C funkcií založené na počte a typoch argumentov, vykonávanie objektovo orientovaného rozhrania a iné.
Možnosťou je taktiež napísanie spájacieho kódu priamo v jazyku C. Avšak, to môže byť zdĺhavé v prípade, že je potrebné napísať spojenie pre viacero C funkcií a programátor nie je oboznámený s Perl zásobnikom a podobnými prípadmi. Preto je XS dobrá voľba pre takéto situácie keď namiesto písania dĺheho spájacieho kódu v C sa napíše krátky popis, čo sa bude vykonávať ako spoj. Potom sa kompilátor XS (xsubpp) postará o zvyšok.
Jazyk XS povoľuje popísať mapovanie medzi tým, ako je používaná rutina v C a taktiež ako je používaná korešpondujúca rutina v jazyku Perl. Ďalšou možnosťou je vytvorenie Perl rutiny, ktorá je priamo preložená do kódu napísaného v programovacom jazyku C a nie je v žiadnom vzťahu s už existujúcou funkciou C. V prípade, keď C rozhranie sa zhoduje s Perl rozhraním, tak XSUB deklarácia je skoro identická deklarácii C funkcie. V iných prípadoch existuje ďalší nástroj zvaný *h2xs*, ktorý je schopný prekladať celkový hlavičkový C súbor do korešpondujúceho XS súboru, ktorý slúži ako spojenie funkcií a makier popísané v hlavičkovom súbore.
Kompilátor XS (xsubpp) vytvára potrebnú konštrukciu, aby mohla XSUB manipulovať

s premennými Perlu a vytvoriť spojenie potrebné aby Perl, mohol pristupovať k XSUB. Kompilátor používa typové mapy aby vedel určiť ako namapovať parametre C funkcie a výstupné premenné do Perl premenných, a naopak. Základná typová mapa sa vysporiada s mnohými všeobecnými typmi jazyka C. Doplnkové typové mapy môžu byť taktiež potrebné pre vysporiadanie sa so špeciálnymi štruktúrami a typmi pre knižnicu, ktorá bude nalinkovaná.
Súbor v XS formáte začína sekciou jazyka C, ktorá je obsiahnutá až po direktívu *MODULE*. Následne môžu byť definované ďalšie direktívy a definície XSUB. Jazyk použitý v tejto časti súboru sa označuje ako XS jazyk. Kompilátor (xsubpp) spozná a preskakuje POD (perlpod) v oboch sekciách jazykov, čo umožňuje súboru XS obsahovať vstavanú dokumentáciu.
Ako príklad uvediem jednoduchý výpis "Hello world.". Na počiatku vygenerujeme súbory, s ktorými budeme pracovať, pomocou nástroja *h2xs*.

```
1  $ h2xs -A -n test
2          Writing test/ppport.h
3          Writing test/lib/test.pm
4          Writing test/test.xs
5          Writing test/Makefile.PL
6          Writing test/README
7          Writing test/t/test.t
8          Writing test/Changes
9          Writing test/MANIFEST
```

*-A* slúži pre vynechanie všetkých samospúšťacích prostriedkov.
*-n* špecifikuje meno, ktoré bude použité pre rozšírený modul.

Po spustení tohto nástroja sa nám vytvoril priečinok s názvom "test"a niekoľko súborov.

Súbor **test.xs** obsahuje import potrebných knižníc, meno modulu a balíčka. Po týchto direktívach nasledujú funkcie, ktoré chceme volať pomocou jazyka C. Štruktúra súboru vyzerá nasledovne:

```
1  #include "EXTERN.h"
2  #include "perl.h"
3  #include "XSUB.h"
4
5  #include "ppport.h"
6
7
8  MODULE = test           PACKAGE = test
9
10 void
11 hello()
12 CODE:
13     printf("Hello, world!\n");
```

V tomto súbore bolo vygenerované všetko po 8. riadok. Pre príklad bol doplnený zvyšok súboru. To je funkcia `hello()` so žiadnou návratovou hodnotou a telo funkcie tvorí jednoduchý výpis.
Preklad spočíva v spustení Perl súboru `Makefile.PL`, ktorý vygeneruje **Makefile** a spustenie vygenerovaného Makefile súboru, poprípade inštalácia balíčku.

```
1  $ perl Makefile.PL
2  $ make
3  $ make install
```

Vytvoríme si jednoduchý Perl skript `hello.pl`, ktorý bude používať vytvorený balík *test* a zavolá funkciu `hello()` z daného balíka.

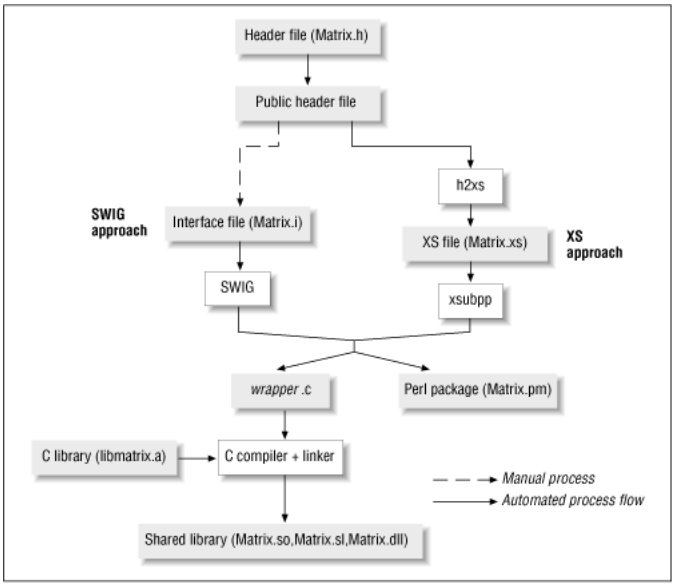
```
1  #!/usr/bin/perl
2  use ExtUtils::testlib;
3  use test;
4
5  test::hello();
```

Potom už len stačí nastaviť práva na spúštanie súboru a jeho spustenie.

```
1  $ $chmod +x hello.pl
2  $ ./hello.pl
3  Hello, world!
4  $
```

**MANIFEST** obsahuje názvy súborov, ktoré sú potrebné k spájaciemu kódu a ktoré medzi sebou interagujú.
Súbory **README** a **Changes** obsahujú informácie o tom ako program funguje, ako nainštalovať vytvorený Perl modul a aké zmeny nastali.
Do súboru **/t/test.t** môžeme nadefinovať testy modulu podľa vlastnej fantázie.

V nasledujúcej časti porovnám prístup technológii SWIG a XS. Na obrázku je znázornený postup prekladu.




Obrázek 5.3: Prístup SWIG a XS [9]

Čo sa týka popisu rozhrania, tak SWIG generuje spájací kód. V tomto prípade je to kód obsiahnutý v súboroch `Matrix.pm` a `Matrix_wrap.c`. Ak systém podporuje dynamické linkovanie a Perl je nastavený tak, aby ho bol schopný používať, tak ostáva už len konvertovať spajací kód a napísanú C knižnicu do dynamickej knižnice. Ak je problém na systéme s dynamickým linkovaním, tak je statickým linkovaním Perl knižníc (libperl.a na Unix systéme alebo perl.lib na Microsoft Windows) a časťami kódu spomenutom vyššie, vygenerovaný nový Perl spustiteľný súbor.

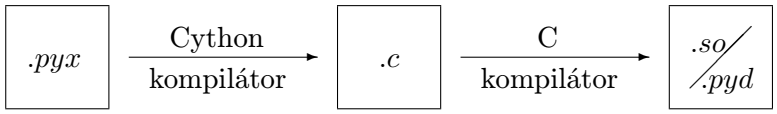
## 5.3 Cython

Cython je programovací jazyk, ktorého základ tvorí skriptovací jazyk Python s doplnenou syntaxou, ktorá umožňuje deklaráciu voliteľných statických typov. Je to nadmnožina Pythonu, ktorá dodáva vyššiu úroveň, objektovo orientované, funkcionálne a dynamické programovanie. Zdrojový kód je preložený do optimalizovaného C/C++ kódu a skompilovaný ako rozšírený modul pre Python. To umožňuje veľmi rýchle vykonávanie programu a pevnú integráciu s externými knižnicami jazyka C. Primárne vykonávacie prostredie Pythonu je známe ako CPython a je naprogramované v jayku C. Toto prostredie je nápomocné pri balení externých knižníc, ktorých rozhranie je cez C jazyk. Nie je preto triviálne napísať požadovaný spájací kód v C, hlavne pre programátorov, ktorý nemajú požadované znalosti. Je taktiež založený na Pyrexe. Jeho zdrojový kód prekladá kód Pythonu do ekvivalentného C kódu. Tento kód je vykonávaný bežiacim prostredím CPythonu ale rýchlosť je daná skompilovaným C kódom a so schopnosťou priamo volať knižnice jazyka C. Zároveň udržuje originálne rozhranie zdrojového kódu Pythonu, ktoré je umožňuje priamo využívať z Pythonovského kódu. Tieto dve hlavné charakteristiky umožňujú Cythonu dva prípady použitia:

- Rozšírenie CPython interpreteru s rýchlymi binárnymi modulmi
- Prepojenie medzi kódom v Pythone a externými C knižnicami

Zatiaľ čo Cython dokáže kompilovať normálny kód v Pythone, tak generovaný kód z C dodáva veľké vylepšenia rýchlosti od voliteľných deklarácií statických typov pre oba typy Pythonu a C. Deklarácia typov môže byť využitá z dvoch dôvodov. Prvým je presun časti kódu z dynamickej sémantiky Pythonu do statickej a rýchlej sémantiky jazyka C. Druhým dôvodom je umožnenie priameho manipulovania s typmi, ktoré sú definované v externých knižniciach.
Cython je implicitne v niektorých distribuciách Pythonu (Enthought Python Distribution, Pythonxy a Sage) a preto nie je potrebná jeho inštalácia. Vyžaduje kompilátor jazyka C a ten závisí od operačného systému (Linux, Mac OS X, Windows).
Cython kód musí byť narozdiel od Python kód skompilovaný a to sa realizuje v dvoch štádiách:

1. Súbor *.pyx* je skompilovaný Cythonom do súboru *.c* obsahujúci kód rozšíreného modulu Pythonu.
2. Následne je súbor *.c* skompilovaný prekladačom pre jazyk C do súboru *.so* (vo Windowse do *.pyd*), ktorý môže byť už importovaný do Pythonu.




Obrázek 5.4: Preklad Cython kódu

Pre ukážku ako môže vyzerať volanie C funkcií z Cython kódu je v nasledujúcom príklade. Pre jednoduchosť je vybrané volanie funkcie zo štandardnej knižnice jazyka C. Nepridávajú sa žiadne závislosti do kódu a obsahuje ďalšiu výhodu, že Cython už definuje

veľa podobných funkcií. Preto je v jednoduchosti ich iba importovať za pomoci `cimport` a následne ich použiť. V tomto príklade sa z reťazca `char*`, ktorý obsahuje číslo, parsuje jeho obsah a jeho návratová hodnota je typu `int`.

```
1  from libc.stdlib cimport atoi
2
3  cdef CstringToPyInt(char* cstring):
4      assert cstring is not NULL, "string is NULL"
5      return atoi(cstring)
```

Kompletný zoznam týchto štandardných cimport súborov je možné nájsť v Cython zdrojovom balíčku *Cython/Includes*. Preklad kódu je možný viacerými spôsobmi a to napríklad pomocou príkazového riadku, pomocou *distutils*, *pyximport*, *cython.inline*, pomocou Sage alebo direktív.

## 5.4 Ctypes

Ctypes je vzdialená knižnica pre skriptovací jazyk Python. Poskytuje kompatibilitu s dátovými typmi jazyka C a umožňuje používať funkcie z DLL alebo zdieľaných knižníc. Môže byť využívaný na zabalenie knižníc co čistého Pythonu. V porovnaní s Cythonom má výhodu v tom, že môže byť štandardná knižnica používaná priamo v Python kóde bez ďalších závislostí. Veľkou nevýhodou je jeho výkonnosť. Je to z dôvodu toho, že najskôr musia všetké operácie prejsť kód Pythonu. Cython ako skompilovateľný jazyk sa môže viacerým záťažiam vyhnúť presunom viacerých funkcií a dlhými cyklami do rýchlého kódu v jazyku C.

# Kapitola 6

# Implementácia

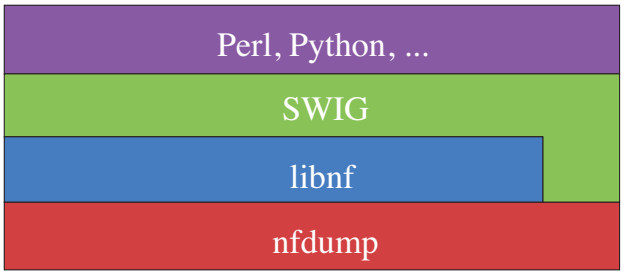
Táto kapitola sa venuje implementácii aplikačného rozhrania pre prácu s dátami NetFlow. V úvodnej časti popíšem aká technológia bola vybraná a prečo. Potom budem postupne rozoberať implementované funkcie a ich význam. Na konci tejto kapitoly dosiahnuté výsledky zhrniem a poukážem na výkonnosť tohto aplikačného rozhrania.

## 6.1 Výber technológie

Výber technológie spočíval so zameraním na jej flexibilitu. Ako bolo spomenuté v predchádzajúcej kapitole, tak SWIG umožňoval najviac výhod v oblasti použitia skriptovacích jazykov. Podporoval nielen najznámejšie jazyky Perl a Python ale aj mnoho ďalších na rozdiel od XS, ktorý podporoval iba jazyk Perl. Preto som pri tomto zistení sa zameral na SWIG aby nemusel administrátor byť obmedzený len na skriptovací jazyk Perl. Taktiež pri skúšaní spomenutých technológii sa mi najlepšie pracovalo hlavne so SWIGom.

## 6.2 Rozhranie

Aplikačné rozhranie je závislé na aplikácii nfdump. Vývoj spočíval v niekoľkých krokoch a spočiatku bol zameraný len na využitie funkcií nástroja nfdump. To z hľadiska manipulácie s dátami nestačilo. Aplikačné rozhranie má poskytovať plnú kontrolu nad týmito dátami. Funkcie nfdumpu túto skutočnosť nepodporovali a ak áno, tak v pozadí vykonávali aj iné činnosti, ktoré by spomaľovali manipuláciu s dátami. Preto bolo potrebné niektoré funkcie prepísať a doplniť aby funkcionalita bola jednak jednoduchšia, ale aj flexibilnejšia. Na obrázku 6.1 sú znázornené vrsty ako nad seba nadväzujú.




Obrázek 6.1: Nadväzovanie vrstiev aplikačného rozhrania

Úplne spodnú vrstvu tvorí nástroj nfdump a jeho funkcie. Druhú vrstvu tvoria obmenené funkcie nfdumpu, ktoré tvoria jadro celej diplomovej práce. Nasleduje technológia, v našom prípade je to SWIG, na prepojenie funkcií s vyšším programovacím jazykom. Ako je vidieť na obrázku, SWIG využíva aj funkcie priamo z nástroja nfdump. Poslednú vrstvu tvorí skriptovací jazyk, kde môže administrátor manipulovať s dátami, volaním funkcií z vygenerovanej knižnice.
Libnf sa skladá z niekoľkých častí. Snaha bola o logické rozdelenie a implementované funkcie, ktoré spolu súvisia, oddeliť do samostatných modulov. Skladá sa z týchto častí:

- **libnf.i** - súbor rozhrania, ktorý obsahuje definície funkcií a premenných. Tieto funkcie a premenné sa môžu využívať vo vyššom programovacom jazyku.
- **arguments.i** - doplnkový súbor rozhrania, v ktorom je nadefinovaná typová mapa pre typ *char***. Je to z dôvodu, že SWIG nepodporuje tento typ a preto ho bolo potrebné nadefinovať.
- **libnf.c** - jadro knižnice. Obsahuje funkcie na inicializáciu, na uvolnenie naalokovaných prostriedkov, na načítanie záznamu zo súboru alebo jeho zápis.
- **libnf.h** - hlavičkový súbor obsahujúci definície funkcií a premenných.
- **libnffile.c** - obsahuje funkcie na prácu so súbormi ako napríklad importovanie súborov, ktoré sa budú spracovávať. Ďalej sú v tomto module implementácie funkcií na vytvorenie súboru na zápis a spočítanie súborov na spracovanie.
- **libnfmapping.c** - v tomto module sú zahrnuté funkcie na mapovanie záznamov. To znamená naplniť vnútornú štruktúru, s ktorou sa bude pracovať v skriptovacom jazyku. Taktiež je potrebné mapovanie opačným smerom na zápis do súboru.
- **libnfexpansion.c** - doplnkové funkcie, ktoré nie sú nevyhnutné na beh programu. Obsahuje funkciu na výpis procesu spracovania.

Rozbor funkcií bude popísaný v nasledujúcej podkapitole.

### 6.2.1 Rozbor funkcií

V tejto podkapitole rozoberiem implementované funkcie, ich závislosti a využité algoritmi. Dôraz bol kladený na jednoduchú manipuláciu a tým čo najmenšie možné volania funkcií knižnice zo skriptovacieho jazyka. Knižnica je implementovaná v jazyku C pre rýchle spracovanie a taktiež kompatibilitu s nástrojom nfdump. Prehľad funkcií, ktoré je možné využiť pri práci s NetFlow dátami:

- **int Init(char p)** - Inicializačná funkcia na prípravu prostredkov, s ktorými sa bude manipulovať.
  *Vstup*: Indikátor, ktorý určuje či sa pri spracovaní súborov bude vypisovať proces jeho vykonávania. To zahrňuje v poradí aktuálny spracovávaný súbor, celkový počet spracovávaných súborov, aktuálne spracovávaný blok, celkový počet blokov zo všetkých súborov a koľko percent sa už spracovalo. Ak chceme tento proces vypisovať na konzolu je potrebné zadať hodnotu indikátora písmeno 'p', inak ostatné znaky budu ignorované. Na obrázku 6.2 je znázornená ukážka procesu spracovania súborov.
  *Výstup*: Hodnota makra `EXIT_SUCCESS` pri správnej inicializácii knižnice. V opačnom prípade je návratova hodnota `EXIT_FAILURE` makra jazyka C.


Obrázek 6.2: Ukážka procesu spracovania

- **int Free()** - Uvoľnenie alokovaných zdrojov.
  *Výstup*: Ako pri inicializačnej funkcii, tak aj v tejto sú návratové hodnoty `EXIT_SUCCESS` a `EXIT_FAILURE`.

- **int Import_files(char *Mdirs, char *rfile, char *Rfile)** - Funkcia, ktorá definuje súbory na spracovanie.
  *Vstup*: Na vstupe môžu vzniknuť kombinácie vstupných hodnôt.

  - `char *Mdirs` - Spracovanie viacerých adresárov. Napríklad:
    - */any/path/to/dir1:dir2:dir3* - Tento výraz bude expandovaný do adresárov, /any/path/to/dir1, /any/path/to/dir2 a /any/path/to/dir3. Spracovávané súbory sú špecifikované druhým alebo tretím parametrom tejto funkcie.
  - `char *rfile` - Spracovanie jedného súboru.
  - `char *Rfile` - Spracovanie sekvencie súborov v rovnakom adresári. Napríklad:
    - */any/dir* - Rekurzívne spracovanie všetkých súborov v adresári *dir*.
    - */dir/file* - Spracovanie všetkých súborov v adresári *dir* začínajúce súborom *file*.
    - */dir/file1:file2* - Spracovanie všetkých súborov v adresári *dir* začínajúce súborom *file1* a končiace súborom *file2*.

  *Výstup*: Rovnaký ako v predchádzajúcich funkciách. Povolené sú len tieto kombinácie vstupných hodnôt (spomenuté budú iba poradia operátorov a nie celé názvy): 2, 3, 1 a 2, 1 a 3. Iné kombinácie vedú k chybe.

- **libnf_record_t**[1] **Read_record()** - Funkcia na načítanie jedného záznamu z akuálne spracovávaného súboru. Preto je potrebné túto funkciu volať v cykle.
  *Výstup*: Výstupom je štruktúra, ktorá obsahuje všetky podporované rozšírenia aplikácie nfdump.

- **int Write_record(libnf_record_t record)** - Funkcia na zápis záznamu, ktorý je uložený následne do súboru.
  *Vstup*: Záznam, ktorý mohol byť upravený administrátorom, poprípade úplne nový záznam.
  *Výstup*: Hodnota makra `EXIT_SUCCESS` pri správnom vyhodnotení. V opačnom prípade je výstupom hodnota makra `EXIT_FAILURE`.

### 6.2.2 Práca s rozhraním

Ako už bolo spomenuté v predchádzajúcich častiach, tak cieľom aplikačného rozhrania je čo najjednoduchšie manipulovať s dátami NetFlow. Preto práca s rozhraním zahŕňa kroky, ktoré je nutné dodržať. V opačnom prípade bude dochádzať k chybám. Pozostáva z nasledujúcich krokov:

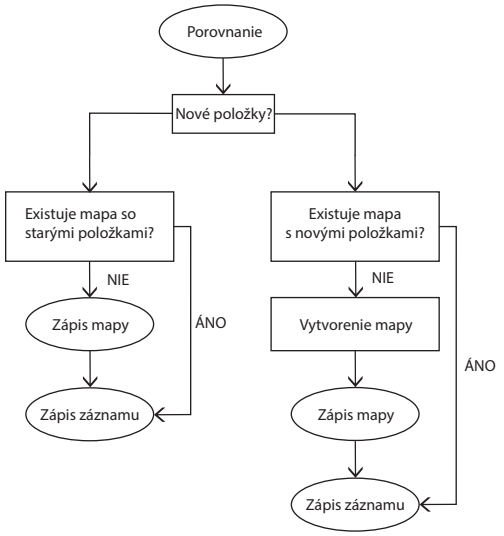
[1] obsah štruktúry je popísaný v prílohe B

1. Inicializácia rozhrania (alokovanie miesta v pamäti)
2. Importovanie súborov, ktoré chce užívateľ spracovať
3. Načítanie záznamu zo súboru
4. Spracovanie (obmena záznamov podľa požiadavok)
5. Zápis záznamu (ak sa jedná o posledný záznam tak nasleduje krok 6. V opačnom prípade sa opakuje 3. krok)
6. Uvolnenie naalokovaných zdrojov

Pre jednoduchšiu predstavu je na obrázku 6.3 načrtnutá kostra, pre prácu s rozhraním.


Obrázek 6.3: Kostra rozhrania

Z dôvodu štruktúry súborov aplikácie nfdump bolo potrebné sa vysporiadať so záznamami rozšírení. Tie obsahujú okrem iného aj identifikátor a zoznam použitých rozšírení v záznamoch, ktoré obsahujú samotné dáta. Proces zápisu máp a záznamov je znázornený na obrázku 6.4.




Obrázek 6.4: Spracovanie záznamov rozšírení

Knižnica libnf používa mapy zo súborov, aby sa nemuseli znova vytvárať. To umožňuje rýchlejšie spracovanie máp a dopĺňanie do ich zoznamu. Po porovnaní starého a nového záznamu sa zapamätajú nové rozšírenia, ktoré sa doplnili. Ak boli pridané nové položky, tak nasleduje porovnanie, či novo vytvorená mapa už existuje. Ak neexistuje, tak sa zapíše mapa a následne spracovávaný záznam. V opačnom prípade sa zapíše iba záznam. V prípade keď záznam bol nezmenený sa porovnáva stará mapa, či už je obsiahnutá v súbore. Ak nie sa zapíše, v opačnom prípade sa zapíše iba spracovávaný záznam. Pre jednoduchšiu prácu sú IP a MAC adresy prevedené na reťazce aby boli v čitateľnej forme. Či už užívateľ zadá IPv4 alebo IPv6 adresu, tak o to sa postará libnf a priradí správne rozšírenie do mapy. V nasledujúcich podkapitolách uvediem zopár príkladov použitia aplikačného rozhrania. Tieto príklady sú len pre inšpiráciu a preto dáta sú vymyslené. Ešte pred príkladmi poukážem na čas, ktorý bol potrebný na pridanie alebo prepísanie niektorých položiek. Test prebiehal na dvoch rôznych strojoch s nasledujúcimi parametrami.

| | PC 1 | PC 2 |
|---|---|---|
| Procesor | 2 GHz Intel core i7 | 2.67 GHz Intel Xeon X5650 |
| Počet jadier | 2 | 12 |
| RAM | 8BG 1600 MHz DDR3 | 64GB 1333 MHz DDR3 |
| OS | MAC OS X 10.8.3 64bit | GNU/LINUX 64bit |
| Jadro OS | Darwin Kernel Version 12.3.0 | Linux 2.6.32-131.21.1.el6.x86_64 |

Tabulka 6.1: Parametre testovacích zariadení

V nasledujúcej tabuľke sú hodnoty časového trvania príkladov, ktoré budú ukázané v ďalších podkapitolách. Do úvahy bol braný súbor veľkosti 73.5 MB s 154 blokmi. V 2. príklade boli v adresári dva rovnaké súbori. Podľa nameraných hodnôt môžeme vyvodiť záver, že výkonnosť veľmi neovplyvňuje rýchlosť pridávania položiek.

| | Príklad 1 | Príklad 2 | Príklad 3 |
|---|---|---|---|
| PC 1 | 33.4s | 1min 6s | 34.8s |
| PC 2 | 32.1s | 1min 5s | 31.3s |

Tabulka 6.2: Čas trvania príkladov

### 6.2.3 Príklad č.1

V tomto príklade sa priradí k zdrojovým a cieľovým IP adresám určitá MAC adresa. Spracovávať sa bude len samostatný súbor.

```
1  #!/usr/bin/python
2
3  import os
4  import sys
5  import struct
6  import libnf
7
8  ret = 0
9
10 libnf.Init("p")          # 'p' indicates to run progressbar
11
12 # import only single file (second argument)
13 ret = libnf.Import_files(None, "nfcapd.example", None)
14 if (ret):
15         print >> sys.stderr, 'Python: Error in importing files.'
16
17 record = libnf.libnf_record_t() # intialization of libnf_record
18
19 while record.end == 0:
20         record = libnf.Read_record()    # read one record from file
21                                         # handling with records
22
23         if (record.srcaddr == "192.168.22.1"):
24                 record.in_src_mac = "AA:BB:CC:00:11:22"
25         if (record.dstaddr == "192.168.33.2"):
26                 record.in_src_mac = "AA:BB:CC:55:66:77"
27
28         ret = libnf.Write_record(record)        # write actual record into file
29         if (ret):
30                 print >> sys.stderr, 'Python: Error while writing record into file'
31
32 libnf.Free()    # free libnf
33
34 sys.exit(0)
```

### 6.2.4 Príklad č.2

Priradenie IPv6 adresy smerovača, z ktorého bol zaslaný tok do súborov v adresári.

```
1  #!/usr/bin/python
2
3  import os
4  import sys
5  import struct
6  import libnf
7
8  ret = 0
9
10 libnf.Init("p")          # 'p' indicates to run progressbar
11
12 # import whole directory (second argument)
13 ret = libnf.Import_files(None, None, "files/")
14 if (ret):
15         print >> sys.stderr, 'Python: Error in importing files.'
16
17 record = libnf.libnf_record_t() # intialization of libnf_record
18
19 while record.end == 0:
```

```
20          record = libnf.Read_record()    # read one record from file
21                                          # handling with records
22
23          if (record.srcaddr == ”192.168.22.1”):
24                  record.ip_router = ”2001:db8:85a3:8d3:1319:8a2e:370:11”
25
26          ret = libnf.Write_record(record)        # write actual record into file
27          if (ret):
28                  print >> sys.stderr, 'Python: Error while writing record into file'
29
30  libnf.Free()    # free libnf
31
32  sys.exit(0)
```

### 6.2.5 Príklad č.3

Priradenie IPv6 adresy smerovača, z ktorého bol zaslaný tok do súborov v adresári.

```
1   #!/usr/bin/python
2
3   import os
4   import sys
5   import struct
6   import libnf
7
8   ret = 0
9
10  libnf.Init(”p”)         # 'p' indicates to run progressbar
11
12  # import whole directory (second argument)
13  ret = libnf.Import_files(None, ”nfdump.example”, None)
14  if (ret):
15          print >> sys.stderr, 'Python: Error in importing files.'
16
17  record = libnf.libnf_record_t() # intialization of libnf_record
18
19  while record.end == 0:
20          record = libnf.Read_record()    # read one record from file
21                                          # handling with records
22
23          if (record.srcaddr == ”192.168.5.4” and record.dstaddr == ”192.168.10.5”):
24                  record.srcaddr = ”2001:db8:85a3:8d3:1319:8a2e:370:432”
25                  record.in_src_mac = ”aa:11:55:23:66:00”
26
27                  record.dstaddr = ”2001:db8:85a3:8d3:1319:8a2e:370:333”
28                  record.out_dst_mac = ”aa:63:77:06:32:49”
29
30                  record.ip_router = ”192.168.1.1”
31                  record.srcas = 11
32                  record.dstas = 22
33
34          ret = libnf.Write_record(record)        # write actual record into file
35          if (ret):
36                  print >> sys.stderr, 'Python: Error while writing record into file'
37
38  libnf.Free()    # free libnf
39
40  sys.exit(0)
```

# Kapitola 7

# Možné rozšírenia

Aplikačné rozhranie poskytuje kompletnú zmenu NetFlow dát zachytených nástrojom nfdump. Preto by sa dalo povedať, že jeho funkčnosť je kompletná. Medzi ďalšie možné rozšírenia, ktoré by administrátor mohol využiť môžu patriť napríklad tieto:

- *Grafické rozhranie* - Asi každý administrátor, ktorý by nemal skúsenosti so skriptovacími jazykmi, by ocenil grafické rozhranie kde by si mohol zadať čo by chcel vylepšiť a v akých súboroch. Problémom by mohol nastať ak by grafické rozhranie neobsahovalo všetké možnosti manipulácie s dátami s akými je schopný vytvoriť napríklad vo flexibilnom skriptovacom jazyku.
- *Vlastné funkcie* - Implementácia vlastných funkcií by mohlo dopomôcť k zrýchleniu programu. Tieto funkcie by boli zamerané len na určité vlastnosti, ktoré administrátor chcel použiť keďže niektoré funkcie naimplementovné v aplikácii nfdump majú zbytočné funkcie naviac, ktoré sa nevyužívajú.
- *Implementácia ďalších funkcií nfdumpu* - Nástroj nfdump poskytuje mnoho ďalších funkcií ku ktorým patrí napríklad agregácia dát. Administrátor by potom nemusel znova spúšťať nástroj nfdump na novo vytvorený súbor.
- *Vytvorenie štatistík* - Ukončenie aplikácie by mohlo umožňovať vytvorenie štatistík o informáciach, koľko záznamov bolo pozmenených, koľko ich bolo vytvorených, odobraných alebo nezmenených.
- *Importovanie s Berkeley databázou* - Rozhranie by mohlo obsahovať funkciu, ktorej vstupom by bola Berkeley databáza, ktorá by obsahovala tabuľku s IP, MAC adresami a časovými značkami od kedy do kedy bola priradená daná MAC adresa k IP adrese. Poprípade s tabuľkou MAC, užívateľské meno a časové značky od kedy do kedy bolo priradené užívateľské meno k danej MAC adrese. Podobný nástroj je už implementovaný a je zameraný len na túto časť. Viac o tejto aplikácii je možné nájsť na [14].

# Kapitola 8

# Záver

V tejto diplomovej práci bolo potrebné spočiatku naštudovať teóriu k problematike týkajúcu sa protokolu NetFlow. Bol naštudovaný jeho popis, využitie a architektúra. Architektúra je zložená z exportéru a kolektoru. Ďalej som sa zoznámil s nástrojom nfdump a jeho funkciami. Vďaka jeho zdrojovým kódom som zistil štruktúru binárnych súborov, ku ktorej mi pomohol hlavne hlavičkový súbor *nffile.h*. Pomocou nástroja nfdump som sa uisťoval, či súbory čítam správne. Pomohli mi k tomu výpisy základných údajov, výpis máp rozšírení a overenie platnosti súborov. Ďalšou úlohou bolo pre mňa naštudovať ako volať funkcie jazyka C vo vyššom programovacom jazyku. Zameral som sa hlavne na metódy pomocou XS a SWIG. Po porovnaní som sa na jednu zameral. Ako už bolo spomenuté, tak technológia SWIG mi pripadala jednoduchšia na prácu a vytvorenie rozhrania. Za vysokoúrovňový jazyk som zvolil Python a preto som aj rozhranie optimalizoval na jeho použitie.
Po zoznámení sa s funkciami nfdumpu a po výbere technológie, ktorá spája nízkoúrovňový jazyk s vysokoúrovňovým, som začal navrhovať aplikačné rozhranie. Hlavným zámerom bolo čo najednoduchšie manipulovať s dátami.
Následne po vytvorení rozhrania som vytvoril zopár príkladov a demonštroval jeho funkčnosť. Testovaním sa ukázalo, že doplňovanie ďalších položiek do binárnych súborov nie je až tak časovo odlišné. Aplikačné rozhranie som na záver otestoval na sieti VUT.

# Literatura


[1] B. Claise, E.: Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.
URL http://tools.ietf.org/html/rfc3954

[2] Cisco: Configuring SPAN.
http://www.cisco.com/en/US/docs/switches/lan/catalyst2940/software/release/12.1_19_ea1/configuration/guide/swspan.html, [cit. 2013-04-26].

[3] Claise, B.: Requirements for IP Flow Information Export (IPFIX). RFC 3917, October 2004.
URL http://www.ietf.org/rfc/rfc3917.txt

[4] Flowmon, N.: Vaša sieť pod kontrolou. http://www.nextcom.sk/flowmon_monitorovanie_siete_uchovavanie_dat.xhtml, [cit. 2013-04-26].

[5] Grégr, M.; Podermański, T.; Šoltés, M.; aj.: Design of Data Retention System in IPv6 network. Technická zpráva, 2011.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9840

[6] LLC, T.: Transmission System. http://data-acquisition-from-mobiles.wikispaces.com/4.+Transmission+System, 2013 [cit. 2013-04-26].

[7] P., R.: What is the Cisco ASA? http://www.cxtec.com/blog/what-is-cisco-asa-security-appliance/, 2013 [cit. 2013-04-30].

[8] Sourceforge: NFDUMP. http://nfdump.sourceforge.net, 2013 [cit. 2013-04-27].

[9] Srinivasan, S.: *Advanced Perl Programming*. O'Reilly Media, 1997, iSBN 1-56592-220-4.

[10] Tecnológicas, A. S.: NetFlow.
http://www.openideas.info/wiki/index.php?title=Pandora:Documentation_en:Netflow, [cit. 2013-04-26].

[11] Welcher, P. J.: NetFlow and IPFIX. http://www.netcraftsmen.net/resources/archived-articles/459-netflow-and-ipfix.html, 2005 [cit. 2013-04-26].

[12] Wikipedia: Network tap. http://en.wikipedia.org/wiki/Network_tap, 2013-04-02 [cit. 2013-04-26].

[13] Wikipedia: NetFlow. http://en.wikipedia.org/wiki/NetFlow, 2013-04-26 [cit. 2013-04-26].

[14] Šoltés, M.: *Nástroj pro práci s daty NetFlow*. Bakalářska práce, FIT VUT v Brně, 2011.
URL http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=11256&y=2010

# Příloha A

# Obsah CD

CD obsahuje nasledujúce adresáre:

- **nflib** – zdrojové súbory knižnice libnf.
- **libnf/nfdump** – zdrojové súboru nástroja nfdump verzie 1.6.9
- **tex** – LaTeXzdrojové súbory tohto dokumentu.

# Příloha B

# Obsah štruktúry libnf_record

```
1  typedef struct libnf_record_s {
2          int              end;
3          uint16_t         type;
4          uint16_t         size;
5          uint8_t          flags;
6          uint8_t          exporter_sysid;
7          uint16_t         ext_map;
8          uint16_t         msec_first;
9          uint16_t         msec_last;
10         uint32_t         first;
11         uint32_t         last;
12         uint8_t          fwd_status;
13         uint8_t          tcp_flags;
14         uint8_t          prot;
15         uint8_t          tos;
16         uint16_t         srcport;
17         uint16_t         dstport;
18         char             srcaddr[40];
19         char             dstaddr[40];
20         uint32_t         input;
21         uint32_t         output;
22         uint32_t         srcas;
23         uint32_t         dstas;
24         uint8_t          dst_tos;
25         uint8_t          dir;
26         uint8_t          src_mask;
27         uint8_t          dst_mask;
28         char             ip_nexthop[40];
29         char             bgp_nexthop[40];
30         uint16_t         src_vlan;
31         uint16_t         dst_vlan;
32         uint64_t         out_pkts;
33         uint64_t         out_bytes;
34         uint64_t         aggr_flows;
35         char             in_src_mac[20];
36         char             out_dst_mac[20];
37         char             in_dst_mac[20];
38         char             out_src_mac[20];
39         uint32_t         mpls_label[10];
40         char             ip_router[40];
41         uint8_t          engine_type;
42         uint8_t          engine_id;
43         uint32_t         fill2;
44         uint32_t         bgpNextAdjacentAS;
45         uint32_t         bgpPrevAdjacentAS;
46         uint32_t         conn_id;
47         uint8_t          fw_event;
48         uint16_t         fw_xevent;
49         uint64_t         flow_start;
```

```
50          uint16_t        xlate_src_port;
51          uint16_t        xlate_dst_port;
52          uint32_t        xlate_flags;
53          char            xlate_src_ip[40];
54          char            xlate_dst_ip[40];
55          uint32_t        ingress_acl_id[3];
56          uint32_t        egress_acl_id[3];
57          uint8_t         nat_event;
58          uint8_t         nat_flags;
59          uint16_t        nat_fill;
60          uint16_t        post_src_port;
61          uint16_t        post_dst_port;
62          uint32_t        ingress_vrfid;
63          char            nat_inside[40];
64          char            nat_outside[40];
65          char            username[72];
66          uint64_t        client_nw_delay_usec;
67          uint64_t        server_nw_delay_usec;
68          uint64_t        appl_latency_usec;
69          uint64_t        received;
70  } libnf_record_t;
```