

Technická univerzita v Košiciach  
Fakulta elektrotechniky a informatiky

Aplikačný rámec pre sprostredkovanie  
IPFIX správ v nástroji SLAmeter

Diplomová práca

2013

Bc. Rastislav Kudla

**Technická univerzita v Košiciach**  
**Fakulta elektrotechniky a informatiky**

# **Aplikačný rámec pre sprostredkovanie IPFIX správ v nástroji SLAmeter**

**Diplomová práca**

Študijný program: Informatika  
Študijný odbor: Informatika  
Školiace pracovisko: Katedra počítačov a informatiky (KPI)  
Školiteľ: Ing. Peter Fecilák, PhD.  
Konzultant: Ing. Adrián Pekár

**Košice 2013**

**Bc. Rastislav Kudla**

## **Abstrakt v SJ**

!!! TODO !!!

## **Kľúčové slová**

IPFIX, Sprostredkovanie, Mediátor, Kolektor, Exportér, SLAmeter, BasicMeter, MONICA

## **Abstrakt v AJ**

!!! TODO !!!

## **Kľúčové slová v AJ**

IPFIX, Mediation, Mediator, Collector, Exporter, SLAmeter, BasicMeter, MONICA

## ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **9.2.1 Informatika**

Študijný program: **Informatika**

Názov práce:

**Aplikačný rámec pre sprostredkovanie IPFIX správ v nástroji SLAmeter**  
IPFIX Mediation Framework of the SLAmeter Tool

Študent (tituly, meno, priezvisko): **Bc. Rastislav Kudla**

Školiteľ (tituly, meno, priezvisko): **Ing. Peter Fecil'ak, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

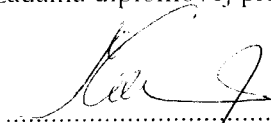
Konzultant práce (tituly, meno, priezvisko): **Ing. Adrián Pekár**

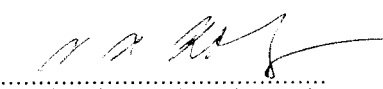
Pracovisko konzultanta: **Katedra počítačov a informatiky**

Pokyny na vypracovanie diplomovej práce:

1. Analyzovať problematiku sprostredkovania IPFIX správ v počítačových sieťach.
2. Navrhnuť architektúru aplikačného rámca pre sprostredkovanie IPFIX správ.
3. Implementovať navrhnuté riešenie v nástroji SLAmeter.
4. Overiť funkčnosť implementovaného riešenia.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský  
Termín pre odovzdanie práce: 26.04.2013  
Dátum zadania diplomovej práce: 31.10.2012

  
prof. Ing. Ján Kollár, CSc.  
vedúci garantujúceho pracoviska

  
prof. Ing. Liberios Vokorokos, PhD.  
dekan fakulty

### **Čestné vyhlásenie**

Vyhlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice 29. 4. 2013

.....

*Vlastnoručný podpis*

## **Podakovanie**

Na tomto mieste sa chcem poďakovať vedúcemu diplomovej práce Ing. Petrovi Fecilakovi, PhD., konzultantovi Ing. Adriánovi Pekárovi ako aj členom výskumnej skupiny MONICA za ich ochotu, cenné rady, pripomienky a odbornú pomoc pri riešení diplomovej práce.

Obzvlášť veľká vďaka patrí mojej rodine a najbližším za podporu a pomoc počas celého štúdia na vysokej škole.

# Predhovor

Trendom posledných rokov v počítačových sieťach sú tzv. konvergované siete, ktoré spájajú prenos dát, hlasu a video komunikácie do jednej spoločnej infraštruktúry. Avšak so zvyšujúcimi sa možnosťami rastú aj nároky a dôraz na kvalitu. Aby sa kvalita sietí a poskytovaných služieb mohla zvyšovať, resp. porovnávať, je potrebné siete merať.

Meracia architektúra SLAmeter vyvíjaná výskumnou skupinou MONICA v Laboratóriu počítačových sietí na Technickej univerzite v Košiciach predstavuje nástroj na pasívne meranie a analýzu parametrov prevádzky počítačových sietí na báze tokov za účelom určovania ich triedy kvality.

Meranie na báze tokov je populárnou a osvedčenou metódou rôznych monitorovacích infraštruktúr. Avšak zdieľanie informácií o tokoch medzi monitorovacími aplikáciami s rôznymi požiadavkami vyvoláva problémy týkajúce sa škálovateľnosti meracieho systému, flexibility monitorovania tokov, ale aj spoľahlivosti exportovania a kapacity meracieho systému. Tieto problémy môže vyriešiť sprostredkovanie správ - *IP Flow Information Export (IPFIX) Mediation*.

Hlavnou úlohou tejto práce je navrhnúť aplikačný rámec pre sprostredkovanie správ medzi exportérom a kolektorom v nástroji SLAmeter. K výberu tejto témy ma viedla možnosť skĺbenia mojich niekoľkých záľub, a to záujmu o počítačové siete spolu so záujmom o programovanie a vyvíjanie nových aplikácií. Navyše už niekoľko rokov som členom výskumnej skupiny MONICA.

# Obsah

Úvod	1
<b>1 Formulácia úlohy</b>	<b>3</b>
<b>2 Analýza</b>	<b>4</b>
2.1 Analýza protokolu IPFIX . . . . .	4
2.1.1 Terminológia . . . . .	4
2.1.2 Formát IPFIX správ . . . . .	7
2.1.2.1 Formát hlavičky správy . . . . .	7
2.1.2.2 Formát sady . . . . .	8
2.1.2.3 Špecifikátory poľa . . . . .	9
2.1.2.4 Formát záznamov šablóny . . . . .	10
2.1.2.5 Formát záznamov šablóny možností . . . . .	11
2.1.2.6 Formát dátových záznamov . . . . .	11
2.2 Analýza sprostredkovania správ v IPFIX . . . . .	12
2.2.1 Terminológia . . . . .	12
2.2.2 Analýza nevýhod meracej architektúry bez Mediátora . . . . .	13
2.2.2.1 Vyrovnanie sa s rastom sieťovej prevádzky . . . . .	14
2.2.2.2 Vyrovnanie sa s viacúčelovým meraním . . . . .	14
2.2.2.3 Vyrovnanie sa s heterogénnym prostredím . . . . .	15
2.2.2.4 Zhrnutie problémov . . . . .	15
2.2.3 Vybrané príklady použitia sprostredkovania správ . . . . .	16
2.2.3.1 Distribuovaná zhromažďovacia infraštruktúra . . . . .	16
2.2.3.2 Anonymizácia dátových záznamov . . . . .	16
2.2.3.3 Selektívne spracovanie dátových záznamov . . . . .	17
2.2.3.4 Interoperabilita medzi protokolmi starších verzií a IPFIX . . . . .	17
2.2.3.5 Prispôsobovanie granularity tokov . . . . .	18



2.2.3.6	Spájanie času . . . . .	18
2.2.3.7	Spájanie priestoru . . . . .	19
2.2.4	Vybrané implementačno-špecifické problémy IPFIX Mediátora . . . . .	19
2.2.4.1	Strata informácie o pôvodnom exportéri . . . . .	19
2.2.4.2	Strata informácie o čase exportu . . . . .	20
2.2.4.3	Interpretácia sprostredkovaných správ . . . . .	21
2.3	Analýza aplikačného rámca pre IPFIX Mediátor . . . . .	21
2.3.1	Referenčný model sprostredkovania správ v IPFIX . . . . .	21
2.3.2	Komponenty sprostredkovania správ v IPFIX . . . . .	22
2.3.2.1	Zhromažďovací proces . . . . .	23
2.3.2.2	Exportovací proces . . . . .	23
2.3.2.3	Sprostredkovateľské procesy . . . . .	24
<b>3</b>	<b>Projekty výskumnej skupiny MONICA . . . . .</b>	<b>26</b>
3.1	Meracia platforma BasicMeter . . . . .	26
3.2	Merací nástroj SLAmeter . . . . .	27
<b>4</b>	<b>Návrh a implementácia aplikačného rámca pre IPFIX Mediátor . . . . .</b>	<b>29</b>
4.1	Požiadavky na rámec pre IPFIX Mediátor . . . . .	29
4.2	Hlavná trieda Mediátora . . . . .	31
4.3	Zhromažďovací proces . . . . .	32
4.3.1	1. fáza zhromažďovacieho procesu . . . . .	32
4.3.2	2. fáza zhromažďovacieho procesu . . . . .	33
4.4	Rozhranie a podpora pre sprostredkovateľské procesy – moduly . . . . .	35
4.4.1	Java ClassLoader a dynamické načítavanie tried . . . . .	35
4.4.2	Abstraktná trieda AIntermediateProcess . . . . .	36
4.4.2.1	Viacvláknovosť . . . . .	36
4.4.2.2	Jediná inštancia modulov . . . . .	36
4.4.2.3	Dekódovanie dátových záznamov . . . . .	40
4.4.2.4	Zakódovanie dátových záznamov . . . . .	41

4.4.2.5	Distribúcia dát medzi modulmi . . . . .	43
4.4.3	Príklad implementácie modulu – ExampleProcess . . . . .	44
4.4.4	Dynamické načítavanie sprostredkovateľských procesov . . . . .	44
4.5	Trieda FlowRecordDispatcher . . . . .	45
4.6	Exportovací proces . . . . .	48
<b>5</b>	<b>Experimentálne overenie funkčnosti riešenia</b>	<b>52</b>
5.1	Testovacia topológia . . . . .	52
5.2	Test konektivity . . . . .	53
5.3	Test správnej reprezentácie údajov . . . . .	55
5.4	Test Mediátora s anonymizačným modulom . . . . .	55
5.5	Záťažový test . . . . .	58
<b>6</b>	<b>Zhodnotenie riešenia</b>	<b>60</b>
	<b>Zoznam použitej literatúry</b>	<b>62</b>
	<b>Zoznam príloh</b>	<b>66</b>

## Zoznam obrázkov

2–1 Zjednodušená verzia IPFIX architektúry . . . . .	4
2–2 Príklad formátu IPFIX správy . . . . .	7
2–3 Formát hlavičky IPFIX správy . . . . .	8
2–4 Formát sady . . . . .	9
2–5 Formát hlavičky sady . . . . .	9
2–6 Formát špecifikátora poľa . . . . .	10
2–7 Formát hlavičky záznamu šablóny . . . . .	11
2–8 Formát hlavičky záznamu šablóny možností . . . . .	11
2–9 Príklad jednej z možných architektúr exportér - mediátor - kolektor .	13
2–10 Strata informácie o originálnom exportéri . . . . .	20
2–11 Referenčný model sprostredkovania správ v IPFIX . . . . .	22
2–12 Zjednodušený model komponentov IPFIX Mediátora . . . . .	23
3–1 Architektúra nástroja BasicMeter (Kudla, 2010) . . . . .	26
3–2 Príklad architektúry nástroja SLAmeter s využitím Mediátora . . . .	28
4–1 Schéma prvej fázy zhromažďovacieho procesu Mediátora . . . . .	33
4–2 Schéma druhej fázy zhromažďovacieho procesu Mediátora . . . . .	34
4–3 Schéma toku dát cez triedu FlowRecordDispatcher . . . . .	46
4–4 Schéma exportovacieho procesu . . . . .	49
5–1 Testovacia topológia . . . . .	53
5–2 Test 1: Konektivita medzi exportérom a Mediátorom – strana exportéra	54
5–3 Test 1: Konektivita medzi exportérom a Mediátorom – strana Mediátora	54
5–4 Test 2: Výpis obsahu databázy . . . . .	55
5–5 Test 2: Správy odosielané exportérom zachytené programom Wireshark	56
5–6 Test 3: Správy odosielané exportérom zachytené programom Wireshark	57
5–7 Test 3: Výpis obsahu databázy . . . . .	58

## Zoznam tabuliek

2–1 Základné skupiny informačných elementov podľa (Quittek, et al., 2008)	6
2–2 Prehľad identifikátorov, typov a záznamov sady . . . . .	9
2–3 Prehľad informačných elementov skupiny 2 . . . . .	24

## Zoznam symbolov a skratiek

a pod.	a podobne
ACP	Analyzer-Colector Protocol
atd.	a tak ďalej
BEEM	BasicMeter Exporting and Measuring process
BGP	Border Gateway Protocol
BMAalyzer	BasicMeter Analyzer
bmIDS	BasicMeter Intrusion Detection System
CNL	Computer Networks Laboratory
ECAM	Exporter-Collector-Analyzer Manager
FIFO	First-In-First-Out
Gb/s	Gigabit za sekundu, jednotka rýchlosti dátového presnosu
IANA	Internet Assigned Numbers Authority
ID	Identification (number)
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPFIX	IP Flow Information eXport
IPv4	Internet Protocol verzie 4
IPv6	Internet Protocol verzie 6

ISP	Internet Service Provider
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
JXColl	Java SXML Collector
LAN	Local Area Network
LTS	Long Term Support
MAC	Media Access Control
MB/s	Megabyte za sekundu, jednotka rýchlosti dátového presnosu
MONICA	Monitoring and Optimization of Network Infrastructures Communications and Applications
napr.	napríklad
OS	operačný systém
PEN	Private Enterprise Number
QOS	Quality of service
resp.	respektíve
RFC	Request for Comments
SCTP	Stream Control Transmission Protocol
SLA	Service-level agreement

SQL	Structured Query Language
TCP	Transmission Control Protocol
tzv.	takzvaný
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
XML	eXtensible Markup Language

## Slovník termínov

**Unix Timestamp** nazývaný tiež Unix time, alebo POSIX time je systém na vyjadrovanie hodnoty času. Je definovaný ako počet uplynutých sekúnd od polnoci 1. januára 1970 UTC.

**Singleton** je návrhový vzor, ktorý povoľuje vytvorenie iba jednej inštancie triedy.

**Parser** je program, ktorý analyzuje vstupné dáta tak, že ich rozdelí na jednotlivé významové časti, ktoré môžu byť následne spracované. (Vereščák, 2012)

**Paket** je forma, resp. blok binárnych dat prenášaných v počítačových sieťach.

**Hash mapa** alebo hash tabuľka je dátová štruktúra používaná na implementáciu asociatívnych polí. K hashovacím kľúčom priraduje zodpovedajúce hodnoty. V jazyku Java môže byť hodnotou akýkoľvek objekt a kľúčom každý objekt, ktorý správne implementuje funkcie *equals()* a *hashCode()*.

**IPv4 adresa** je 32 bitové označenie zariadenia v počítačovej sieti, ktoré na komunikáciu používa Internet Protocol verzie 4 (Information Sciences Inst., 1981).

**IPv6 adresa** je 128 bitové označenie zariadenia v počítačovej sieti, ktoré na komunikáciu používa Internet Protocol verzie 6 (Deering, Hinden, 1998).

**MAC adresa** je 48 bitový, unikátny identifikátor sieťového adaptéra komunikujúceho na fyzickej vrstve.

**Big-Endian** je spôsob usporiadania bajtov dátových typov v pamäti počítača. Najvýznamnejší bajt je uložený na najnižšej (prvej) pozícii. Používa sa pri komunikácii v počítačových sieťach, preto sa tiež nazýva „sieťové poradie bajtov“. Jeho presným opakom je Little-Endian.

**Little-Endian** je spôsob usporiadania bajtov dátových typov v pamäti počítača. Najmenej významný bajt je uložený na najnižšej (prvej) pozícii. Jeho presným opakom je Big-Endian.



**Buffer** je oblasť pamäte používaná na dočasné uchovanie dát pred ich presunom na iné miesto – vyrovnávacia pamäť.

**Socket** je koncovým bodom spojenia v TCP/IP sieťach. Je tvorený kombináciou portu a IP adresy. (Tom Sheldon, 2001)

## Úvod

Rozmach počítačových sietí prináša ľudstvu nové možnosti a služby, o akých sme kedysi ani nesnívali. Internet sa stal súčasťou nášho každodenného života, mnohí sú pripojení aj 24 hodín denne, nielen prostredníctvom počítačov, ale aj mobilných zariadení. Práve vďaka rozmachu mobilného Internetu si už ani nevieme predstaviť situáciu, že by sme neboli v dosahu Internetového obsahu. Týmto sú kladené obzvlášť vysoké nároky na kvalitu sietí a poskytovaných služieb. Aby poskytovatelia Internetu mohli zabezpečovať a zlepšovať kvalitu služieb (QoS), musia ju vedieť merať.

V Laboratóriu počítačových sietí (*CNL*) na Technickej univerzite v Košiciach vyvíja výskumná skupina MONICA pasívny merač parametrov sieťovej prevádzky na báze tokov, ktorý vyhodnocuje dodržiavanie zmluvy o úrovni poskytovanej služby (*SLA*). Jeho cieľom je spracovať určité parametre sieťovej prevádzky a na ich základe vypočítať triedu kvality. Táto aplikácia má slúžiť nielen poskytovateľom pripojenia k Internetu ako monitorovací nástroj. Triedy umožňujú aj ich klientom jednoducho porovnávať kvalitu poskytovaných služieb. Jadro nástroja SLAmeter sa skladá z komponentov, ktoré sú navrhnuté v súlade s protokolom IPFIX. (SLAmeter, 2013)

Komponentmi architektúry IPFIX (*IP Flow Information Export*) podľa RFC 5470 (Sadasivan, et al., 2009) sú exportéry a kolektory komunikujúce protokolom IPFIX. Vzhľadom k trvalému rastu IP prevádzky v heterogénnych sieťových prostrediach, tieto exportér-kolektor systémy môžu viesť k problémom škálovateľnosti. Ba čo viac, neposkytujú flexibilitu potrebnú pre široký rad meracích aplikácií.

Pre naplnenie požiadaviek aplikácií s obmedzenými systémovými zdrojmi, IPFIX architektúra zavádza sprostredkovateľskú entitu medzi exportéry a kolektory. Z pohľadu manipulácie s dátami môžu sprostredkovateľské moduly tejto entity poskytovať agregáciu, koreláciu, filtrovanie, selekciu, anonymizáciu a iné úpravy záznamov o tokoch za účelom šetrenia výpočtových zdrojov meracieho systému a vykonávania predspracovania úloh pre kolektor. Z hľadiska interoperability nástrojov rôznych

vývojárov, môžu poskytovať konverziu z iných protokolov na IPFIX, respektíve zvyšovať spoľahlivosť exportov napríklad prevodom z nespoľahlivého, bezspojoovo orientovaného protokolu UDP na spoľahlivý SCTP. (Kobayashi et al., 2011)

Táto práca analyzuje nastolený problém sprostredkovania správ a sprostredkovateľských entít nazývaný *IP Flow Information Export (IPFIX) Mediation Problem*. Jej hlavným cieľom je navrhnúť a implementovať aplikačný rámec pre sprostredkovateľskú entitu zvanú IPFIX Mediátor pre účely nástroja SLAmeter. Rámec musí klásť veľký dôraz na jeho modularitu, aby bolo jednoduché a pohodlné implementovať nové sprostredkovateľské moduly a tým zvyšovať možnosti monitorovania aplikáciou SLAmeter.

Štruktúra práce je nasledovná. V úvodnej kapitole je formulovaná definícia úlohy. Druhá kapitola sa v krátkosti venuje protokolu IPFIX. Popisuje len formát IPFIX správy, jej hlavičku, typy sád a ich komponenty. Ostatné rysy protokolu si môže čitateľ vyhľadať v príslušných dokumentoch RFC. Nasledujúca podrobne analyzuje problém sprostredkovania správ v IPFIX. Úvodom definuje použitú terminológiu. Analyzuje nevýhody exportér-kolektor architektúr bez Mediátora, príklady použitia sprostredkovania, ale aj jeho implementačno-špecifické problémy. Ďalšia sekcia sa zaoberá analýzou aplikačného rámca pre IPFIX Mediátor. Tretia kapitola stručne priblíži projekty výskumnej skupiny MONICA. Nasledujúca kapitola s poradovým číslom štyri, tvorí jadro práce. V nej sa čitateľ dočíta o návrhu a implementácii aplikačného rámca pre sprostredkovateľskú entitu. Jednotlivým triedam a metódam vyššieho významu je venované podrobné vysvetlenie. Predposledná kapitola niekoľkými testami experimentálne overuje implementované riešenie. Záver už len zhodnotí diplomovú prácu a jej dosiahnuté výsledky.

# 1 Formulácia úlohy

Cieľom tejto diplomovej práce je analyzovať, navrhnúť, implementovať a otestovať aplikačný rámec pre problém sprostredkovania správ v protokole IPFIX. Riešenie je zároveň potrebné integrovať s existujúcou architektúrou nástroja SLAmeter. Za týmto účelom je nutné vykonať nasledujúce kroky.

V prvom rade je potrebné analyzovať IPFIX z pohľadu protokolu ale aj architektúry. Najväčší dôraz sa kladie na analýzu správ, pretože úlohou Mediátora bude aj ich dekódovanie a následné zakódovanie podľa špecifikovaného formátu.

Druhým krokom je analýza problému sprostredkovania správ v IPFIX. Konkrétne ide o definovanie terminológie, analýzu výhod a príkladov použitia, ale aj priblíženie niektorých problémov spojených s implementáciou takéhoto nástroja.

Následne je potrebné čitateľovi aspoň stručne priblížiť projekty skupiny MONICA, medzi ktoré patrí BasicMeter a jeho nadstavba SLAmeter.

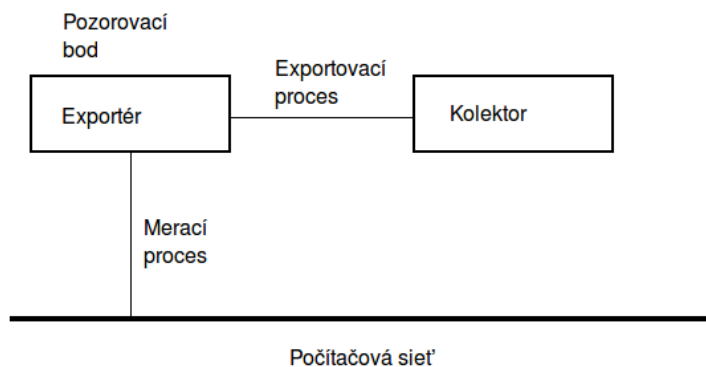
Štvrtým a najdôležitejším krokom je navrhnúť a implementovať samotný aplikačný rámec podľa definovaných požiadaviek. Riešenie musí byť experimentálne overené vhodnými testami.

Posledným, no nie menej dôležitým krokom je podľa pokynov vypracovať dokumentáciu vykonanej práce.

## 2 Analýza

### 2.1 Analýza protokolu IPFIX

Protokol IPFIX - *IP Flow Information Export* je IETF (*Internet Engineering Task Force*) štandard pre export informácií o sieťových tokoch zo smerovačov, meracích sond, alebo špecializovaných nástrojov. Aby bolo možné prenášať tieto informácie od exportovacieho procesu k zhromažďovaciemu procesu, je potrebný štandardizovaný spôsob komunikácie a taktiež jednotná reprezentácia odovzdávaných dát. Protokol je vyvíjaný rovnomennou pracovnou skupinou (IETF, 2013) od roku 2001 a vznikol ako priamy nasledovník proprietárneho protokolu Cisco Netflow Version 9 (Claise, 2004). To znamená, že IPFIX je založený na systéme výmeny informácií na základe šablón. To ho robí veľmi flexibilným, pretože je možné nakonfigurovať, ktoré vlastnosti tokov sa majú merať. Zjednodušená architektúra protokolu IPFIX je na Obrázku 2–1. (Claise et al., 2008; IPFIX protocol; Juvhaugen, 2007; Vereščák, 2012)



Obr. 2–1 Zjednodušená verzia IPFIX architektúry

#### 2.1.1 Terminológia

Podľa (Quittek et al., 2004) existuje veľa definícií termínu „tok“ používaných Internetovou komunitou. Pracovná skupina IPFIX používa nasledujúcu:

**Tok** je definovaný ako množina IP paketov prechádzajúcich pozorovacím bodom v sieti, počas určitého časového intervalu. Všetky pakety patriace príslušnému toku majú množinu spoločných vlastností. Opačne môžeme tvrdiť, že každý paket patrí toku, ak spĺňa všetky tieto spoločné vlastnosti.

Ďalšie termíny zavádza (Claise et al., 2008):

**Záznam o toku** obsahuje namerané informácie a charakteristické vlastnosti konkrétneho toku, ktorý bol pozorovaný pozorovacím bodom (napr. celkový počet prenesených bajtov, zdrojová IP adresa toku, a pod.).

**Pozorovací bod** je miestom v sieti, kde sú IP pakety pozorované. Medzi príklady patrí sieťová linka, na ktorej je zavedená meracia sonda, zdieľané médium (napr. Ethernet LAN), alebo fyzické, či logické rozhranie smerovača. Každý pozorovací bod je asociovaný s pozorovacou doménou.

**Pozorovacia doména** je množina pozorovacích bodov. Je identifikovaná číslom (Observation Domain ID), ktoré je unikátne v rámci exportovacieho procesu.

**Merací proces** vytvára záznamy o tokoch na základe hlavičiek paketov. Vykonáva rôzne funkcie ako napríklad odchytyvanie hlavičiek paketov, vytváranie časových známkov, vzorkovanie, klasifikovanie a údržba záznamov o tokoch.

**Exportovací proces** odosiela záznamy o tokoch zhromažďovacím procesom. Tieto záznamy sú generované jedným alebo viacerými meracími procesmi.

**Exportér** odosiela dáta o tokoch. Je to nástroj ktorý zastrešuje exportovacie procesy.

**Kolektor** je nástroj, ktorý prijíma dáta od exportéra. Pozostáva z jedného, alebo viacerých zhromažďovacích procesov.

**Zhromažďovací proces** prijíma záznamy o tokoch od jedného, alebo viacerých exportovacích procesov. Prijaté záznamy môže spracovávať, alebo uchovávať.

**Informačné elementy** sú protokolovo nezávislým popisom atribútov záznamov o tokoch. Informačný model IPFIX (Quittek, et al., 2008) obsahuje základnú množinu informačných elementov, vrátane ich popisu, významu, dátového typu a pod. Informačné elementy sú rozdelené do 12 skupín, pozri Tabuľku 2–1, na základe ich sémantiky a použitia. Každý element je asociovaný s dátovým typom, ktorý určuje jeho formát a spôsob kódovania. Na základe tohto modelu sú jednotlivé dátové záznamy kódované na strane exportéra a dekódované v kolektore. Informačný model povoľuje aj jeho rozširovanie. Organizácie môžu definovať vlastné informačné elementy, ktorým musia prideliť unikátny identifikátor. Tieto elementy navyše obsahujú identifikátor organizácie (*PEN*), ktorý musí byť registrovaný v IANA<sup>1</sup>.

**Tabuľka 2–1** Základné skupiny informačných elementov podľa (Quittek, et al., 2008)

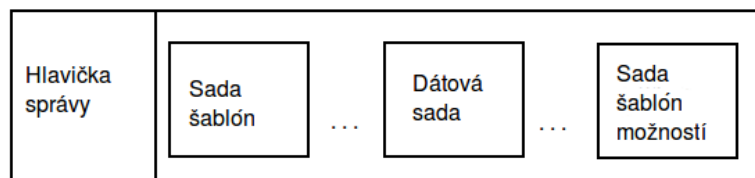
#	názov skupiny
1.	Identifikátory
2.	Konfigurátory meracieho a exportovacieho procesu
3.	Štatistické hodnoty meracieho a exportovacieho procesu
4.	Polia IP hlavičky
5.	Polia transportnej hlavičky
6.	Polia ostatných hlavičiek
7.	Odvodené vlastnosti paketov
8.	Min/Max vlastnosti tokov
9.	Časové známky tokov
10.	Počítadla vlastností tokov
11.	Rôzne vlastnosti tokov
12.	Výplň ( <i>padding</i> )

<sup>1</sup><http://www.iana.org/assignments/enterprise-numbers>

**Šablóna** špecifikuje formát odosielaných záznamov o tokoch pomocou zoznamu informačných elementov. Musí byť odoslaná kolektoru z exportovacieho procesu ešte pred odoslaním samotných dát. Neskôr sú šablóny periodicky preposielané, aby kolektor vedel v každom okamihu aký formát dát prijme. Šablóny musia byť dostupné administrátorom, preto sú definované v konfiguračnom súbore exportéra. (Juvhaugen, 2007)

### 2.1.2 Formát IPFIX správ

Formát správ je definovaný v Špecifikácii IPFIX Protokolu (Claise et al., 2008). Správa pozostáva z hlavičky, nasledovaná niekoľkými IPFIX sadami. Exportér musí zakódovať všetky časti správy v sieťovom poradí bajtov (Big-Endian). Na Obrázku 2–2 je jedna z možností formátu IPFIX správy. Za hlavičkou nasleduje sada šablón, pretože šablóny musia byť exportované ihneď po vytvorení. Za šablónami nasledujú dátové sady a sady šablón možností v akomkoľvek poradí.



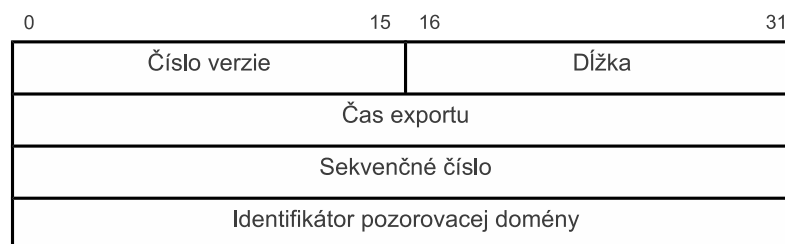
**Obr. 2–2** Príklad formátu IPFIX správy

**2.1.2.1 Formát hlavičky správy** Formát hlavičky správy je znázornený na Obrázku 2–3. Pozostáva z piatich poli:

- **Číslo verzie** záznamu toku v správe. Pre IPFIX je to hodnota 0x000a.
- **Dĺžka** predstavuje celkovú dĺžku IPFIX správy v oktetoch, vrátane hlavičky a sád.
- **Čas exportu** vo formáte UNIX timestamp.



- **Sekvenčné číslo** vyjadruje počet odoslaných záznamov dátových záznamov modulo  $2^{32}$  exportovacím procesom v tejto transportnej relácii. Túto hodnotu používa kolektor na odhalenie chýbajúcich správ resp. dátových šablón.
- **Identifikačné číslo pozorovacej domény**, ktorý je lokálne jedinečný pre exportovací proces.



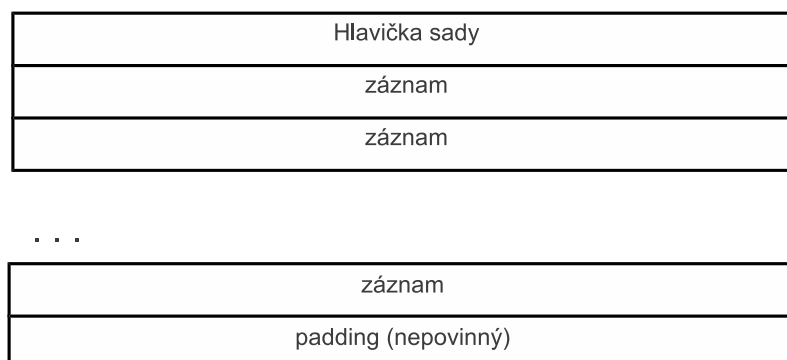
**Obr. 2 – 3** Formát hlavičky IPFIX správy

**2.1.2.2 Formát sady** V IPFIX terminológii je sada všeobecný pojem pre kolekciu záznamov s podobnou štruktúrou. IPFIX správa môže obsahovať tri rôzne druhy sád:

- Sada šablón,
- Dátová sada,
- Sada šablón možností.

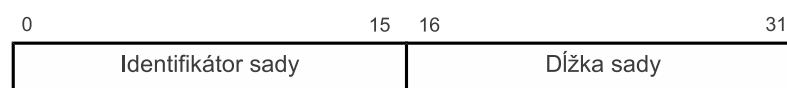
Každá zo sád sa skladá z hlavičky sady a jedného, alebo viacerých záznamov sady, Obrázok 2–4. Na úplnom konci môže byť vložená výplň (*padding*), no nie je to povinná súčasť sady. Exportovací proces ho pridáva iba v tom prípade, keď chce aby sada bola zarovnaná na dĺžku, ktorá je násobkom 4, alebo 8.

**Hlavička sady** je rovnaká pre všetky tri typy sád. Je zobrazená na Obrázku 2–5. *Identifikátor sady* určuje typ sady a teda aj typ všetkých záznamov obsiahnutých v sade, pozri Tabuľku 2–2. Sada istého druhu nemôže obsahovať záznamy iného typu. Hodnota 2 je rezervovaná pre sadu šablón. Sada šablón možností nadobúda



Obr. 2 – 4 Formát sady

hodnotu 3. Identifikátory 0 a 1 sa nepoužívajú z historických dôvodov (Claise, 2004) a hodnoty od 4 po 255 sú rezervované pre budúce použitie. Dátové sady sú označené hodnotami väčšími ako 255.



Obr. 2 – 5 Formát hlavičky sady

*Dĺžka sady* zahŕňa celkovú dĺžku všetkých sád, vrátane hlavičky sady a prípadne výplne. Na jej základe sa určuje koniec tejto sady a začiatok ďalšej, pretože sada môže obsahovať rôzny počet záznamov.

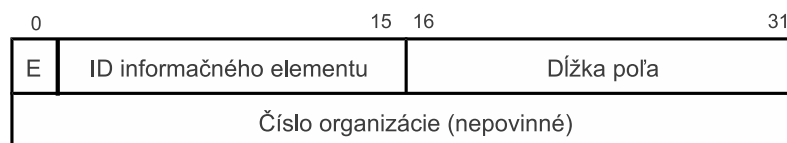
Tabuľka 2 – 2 Prehľad identifikátorov, typov a záznamov sady

Identifikátor sady	typ sady	typ záznamov
0 - 1	–	–
2	sada šablón	záznamy šablóny
3	sada šablón možností	záznamy šablóny možností
4 - 255	–	–
255 - 65535	dátová sada	dátové záznamy

**2.1.2.3 Špecifikátory poľa** Špecifikátor poľa je akousi obálkou nad informačným elementom, Obrázok 2 – 6, vďaka nemu vie zhromažďovací proces spracovávať

prijaté dáta.

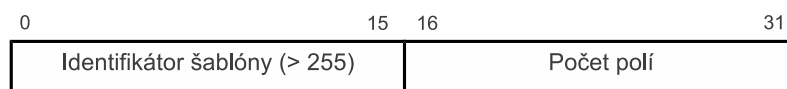
Prvý bit sa nazýva *Enterprise bit*. Ak je nastavený na 0, tak hovoríme o oficiálnom informačnom elemente charakterizovanom organizáciou IETF a registrovanom v IANA. V tomto prípade je *číslo organizácie (PEN)* nevyplnené. V opačnom prípade, keď je tento bit nastavený na 1, ide o organizáciu špecifikovaný informačný element a číslo organizácie musí byť zadané. Laboratórium počítačových sietí na Technickej univerzite v Košiciach má pridelené číslo organizácie 26235 (Private Enterprise Numbers, 2013). Dĺžka poľa vyjadruje na koľkých oktetoch je daný informačný element kódovaný. Zoznam informačných elementov aj s ich dĺžkami je dostupný v (Quittek, et al., 2008). Špeciálny prípad nastáva pri redukovanom kódovaní. Vtedy je dĺžka poľa menšia ako popisuje Informačný model.



**Obr. 2–6** Formát špecifikátora poľa

**2.1.2.4 Formát záznamov šablóny** Záznamy šablóny patria k nevyhnutným prvkom IPFIX správy. Na ich základe, a základe Informačného modelu IPFIX (Quittek, et al., 2008) vie zhromažďovací proces dekodovať dátové záznamy. Šablóny môžu obsahovať akúkoľvek kombináciu informačných elementov. Či už oficiálnych – navrhnutých spoločnosťou IANA, alebo vlastných – vytvorených organizáciami.

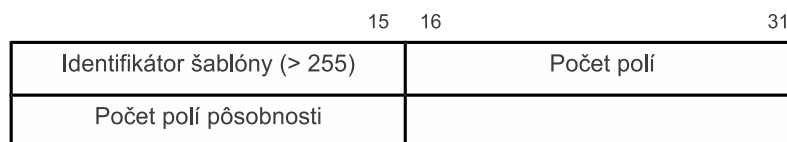
Záznam šablóny tvorí *hlavička*, pozri Obrázok 2–7, nasledovaná *špecifikátormi poľa*. Každá šablóna musí mať v rámci transportnej relácie a pozorovacej domény jedinečný *identifikátor*. Čísľuje sa od 255 do 65535, rovnako ako identifikátory dátových sád, pretože každá šablóna referuje na dátovú sadu, ktorej štruktúru popisuje. *Počet polí* sa týka počtu špecifikátorov poľa.



Obr. 2 – 7 Formát hlavičky záznamu šablóny

**2.1.2.5 Formát záznamov šablóny možností** Tieto záznamy dávajú exportéru možnosť poskytnúť kolektoru dodatočné informácie o kontexte posielených dát, ktoré by zo samotných záznamov tokov nevedel vyčítať. Príkladom týchto informácií sú kľúče tokov, alebo konfigurácia šablóny, vzorkovacie parametre a pod.

Formát záznamov (Obrázok 2 – 8) je podobný ako v prípade záznamov šablóny. Pozostáva z hlavičky záznamu a jedného, alebo viacerých špecifikátorov poľa. Formát špecifikátorov je rovnaký ako pri záznamoch šablóny. Hlavička navyše obsahuje *počet polí pôsobnosti*. Pôsobnosť charakterizuje kontext informácií. Šablóna povoľuje definovať viac polí pôsobnosti ako jedno. V tomto prípade je celková pôsobnosť daná kombináciou týchto polí. Počet polí pôsobnosti nemôže byť nulový, pričom *počet polí* je súčtom polí pôsobnosti a špecifikátorov poľa. (Quittek, et al., 2008)



Obr. 2 – 8 Formát hlavičky záznamu šablóny možností

**2.1.2.6 Formát dátových záznamov** Dátové záznamy sú posielené v dátových sadoch. Ich formát je veľmi jednoduchý. Pozostávajú len z hodnôt polí, nemajú ani vlastnú hlavičku. Sú kódované podľa popisu v Informačnom modeli (Quittek, et al., 2008). Identifikátor šablóny, ktorá popisuje tieto hodnoty je zakódovaný v hlavičke sady, v časti identifikátor sady. Inými slovami „identifikátor sady“ = „identifikátor šablóny“. Aby vedel kolektor tieto dáta dekodovať, musí poznať formát šablóny už pred prijatím prvého dátového záznamu.

## 2.2 Analýza sprostredkovania správ v IPFIX

Výhodou monitorovania sieťovej prevádzky na báze tokov je to, že je možné merať veľké množstvo sieťovej prevádzky v distribuovaných pozorovacích bodoch. Zatiaľ čo tento typ monitorovania môže byť použitý na rôzne účely a pre rozmanité aplikácie, je veľmi obtiažne aplikovať ho paralelne na viac aplikácii s veľmi rozdielnymi požiadavkami. Sieťoví administrátori musia nastaviť parametre meracích nástrojov tak, aby vyhoveli požiadavkám každej jednej monitorovacej aplikácii. Takéto konfigurácie často nie sú podporované meracími nástrojmi. Či už kvôli funkčným obmedzeniam, alebo kvôli pamäťovým a výpočtovým limitom, ktoré zamedzujú meraniu veľkých dátových tokov. Sprostredkovanie správ v IPFIX - *IP Flow Information Export (IPFIX) Mediation* vyplňa túto medzeru medzi obmedzenými možnosťami merania a požiadavkami na monitorovacie aplikácie zavedením sprostredkovateľského zariadenia nazývaného *IPFIX Mediátor* (Kobayashi, Claise, 2010).

### 2.2.1 Terminológia

Terminológia použitá v tejto kapitole je čiastočne definovaná v Kapitole 2 (Sekcia 2.1.1). Dodatočne zadefinujeme nasledujúce termíny:

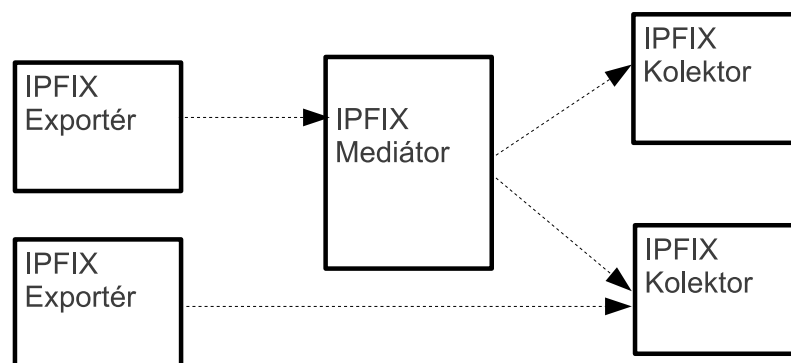
**Prúd záznamov** – *record stream* je sled dát nesúcich informácie o tokoch.

**Sprostredkovanie správ v IPFIX** – *IPFIX Mediation* je manipulácia a konverzia prúdu záznamov pomocou IPFIX protokolu.

**Sprostredkovateľský proces** – *Intermediate Process* prijíma prúd záznamov ako vstupnú veličinu od zhromažďovacieho procesu, meracieho procesu, čítačky IPFIX súborov, iného sprostredkovateľského zariadenia, alebo akéhokoľvek zdroja záznamov. Nad prijatými záznamami vykoná rôzne transformácie, na základe ich obsahu. Napokon zmenené záznamy posúva na svoj výstup buď smerom k exportovaciemu procesu, inému sprostredkovateľskému zariadeniu,

alebo zapisovaču IPFIX súborov za účelom vykonania sprostredkovania IPFIX správ.

**IPFIX Mediátor** je nástroj vykonávajúci sprostredkovanie správ tak, že prijíma prúd záznamov z rôznych dátových zdrojov, zastrešuje jeden alebo viac sprostredkovateľských procesov aby modifikoval obsah prúdu a nakoniec exportuje pozmenené dáta vo forme IPFIX správ pomocou exportovacieho procesu. V typickom prípade prijíma Mediátor prúd záznamov od zhromažďovacieho procesu. No rovnako môže prijímať údaje od iných zdrojov, ktoré nie sú zakódované pomocou IPFIX, napr. v prípade konverzie protokolu NetFlow verzie 9 (Claise, 2004) na IPFIX. Príklad jednej z možných architektúr, v ktorej je použitý Mediátor je na Obrázku 2–9 (Kobayashi, Claise, 2010).



**Obr. 2–9** Príklad jednej z možných architektúr exportér - mediátor - kolektor

### 2.2.2 Analýza nevýhod meracej architektúry bez Mediátora

Problematika sprostredkovania IPFIX správ je podrobne spracovaná v (Kobayashi, Claise, 2010). Hovorí o tom, že sieťoví administrátori často čelia problémom týkajúcim sa škálovateľnosti meracieho systému, flexibility monitorovania na základe tokov, alebo aj spoľahlivosti exportovania. Napriek tomu, že sa vyvinuli známe techniky ako *vzorkovanie a filtrovanie paketov*, *zoskupovanie dátových záznamov*, alebo *replikácia exportu*, tieto problémy nevymizli. Pozostávajú z prispôbovania niekto-

rých parametrov meracích nástrojov zdrojom meracieho systému zatiaľ čo musia naplniť patričné podmienky ako sú *presnosť nameraných dát*, *granularita toku*, či *spôľahlivosť exportu*. Tieto okolnosti závisia na dvoch faktoroch:

1. **Kapacita meracieho systému** - pozostáva zo šírky pásma spravovanej siete, kapacity úložiska a výkonu exportovacích a zhromažďovacích nástrojov
2. **Požiadavky aplikácie** - rôzne aplikácie vyžadujú rôznu zrnitosť záznamov o tokoch a presnosť dát.

**2.2.2.1 Vyrovnanie sa s rastom sieťovej prevádzky** Velké spoločnosti a poskytovatelia Internetového pripojenia (ISP) majú bežne vo svojej sieťovej infraštruktúre linky so šírkou pásma 10 Gb/s a ich celková sieťová prevádzka presahuje 100 Gb/s. Podľa (Cho et. al, 2006) sieťová prevádzka používateľov širokopásmového pripojenia k Internetu v blízkej budúcnosti sa bude každým rokom zvyšovať približne o 40%. Sieťoví administrátori monitorujúci IP prevádzku môžu udržiavať krok s týmto nárastom vďaka použitiu viacerých exportérov. Tento prístup však môže viesť k prekročeniu výpočtových a pamäťových možností jediného kolektora.

Tento problém zmierňujú redukčné techniky *vzorkovanie a filtrovanie paketov* popísané v (Zseby, et al., 2009) implementované v exportéroch. Podobne agregácia meraných dát. Tieto techniky však majú aj svoje nevýhody. Môžu viesť k stratám malých tokov, nemožnosti odhalenia drobných zmien a anomálií v prenášaných dátach. Filtrovanie spôsobuje, že len podmnožina dátových záznamov je exportovaná.

Vzhľadom k týmto nedostatkom sa vyžaduje aby sa veľké meracie infraštruktúry nespoliehali na spomínané redukčné techniky.

**2.2.2.2 Vyrovnanie sa s viacúčelovým meraním** Rôzne monitorovacie aplikácie majú rôzne požiadavky na meraciu infraštruktúru. Niektoré vyžadujú monitorovanie na úrovni tokov, iné informácie o individuálnych paketoch a ďalšie agre-

gované toky a podobne.

Ak by mal exportér naplniť tieto požiadavky, musel by paralelne vykonávať rôzne meracie operácie, čo je kvôli limitovaným výpočtovým zdrojom takmer nemožné. Preto je výhodnejšie použiť exportér s jednoduchším a výkonnejším nastavením a namerané dáta vhodne spracovávať na ďalšej úrovni meracej architektúry.

**2.2.2.3 Vyrovnanie sa s heterogénnym prostredím** Administrátori môžu používať IPFIX nástroje od rozmanitých výrobcov, s rozdielnymi verziami softvéru a s rôznymi typmi sieťových zariadení (smerovač, prepínač, meracia sonda) v jednej sieťovej doméne. V niektorých topológiách sú stále nasadené historické protokoly na export tokov. Dosiahnutie plnej interoperability týchto zariadení nie je možné.

Monitorovací systém sa vie vysporiadať s týmto problémom iba keď je prítomné sprostredkovanie IPFIX správ. Avšak obsiahnuť sprostredkovanie vo všetkých zhromažďovacích zariadeniach je náročné.

**2.2.2.4 Zhrnutie problémov** Vzhľadom k limitovaným zdrojom monitorovacieho systému, je dôležité použiť techniky redukcie sieťových dát čím nižšie v hierarchii systému, teda v exportéri. Avšak implementácia tohto návrhu je sťažená v heterogénnom prostredí exportovacích nástrojov. Na druhej strane, udržiavanie presnosti dát a granularity tokov tak, aby boli splnené požiadavky rôznych monitorovacích aplikácií vyžaduje škálovateľnú a flexibilnú zhromažďovaciu infraštruktúru.

Toto zhrnutie implikuje, že sprostredkovateľská entita (Mediátor), ktorá bude zabezpečovať nové funkcie a výpočty pred tým, ako sa IPFIX správy dostanú ku zhromažďovaciemu procesu je potrebná v typických exportér - kolektor infraštruktúrach.



### 2.2.3 Vybrané príklady použitia sprostredkovania správ

RFC 5982 (Kobayashi, Claise, 2010) uvádza viacero príkladov zaradenia IPFIX Mediátora do klasickej exportér - kolektor architektúry. Uvedme aspoň niektoré.

**2.2.3.1 Distribuovaná zhromažďovacia infraštruktúra** Zvyšovanie počtu IPFIX exportérov, rast objemu IP dát a rôzne požiadavky na operácie vykonávané nad dátovými záznamami v kolektore spôsobujú vysokú náročnosť na implementáciu všetkých meracích aplikácií v rámci jedného kolektora.

Za účelom navýšenia zhromažďovacej kapacity resp. objemu spracovania dátových záznamov musia byť nasadené distribuované kolektory čím bližšie ku exportérom. V tomto prípade sa kolektory stanú IPFIX Mediátormi, ktoré budú preposielať dátové záznamy podľa požiadaviek centralizovaným aplikáciám.

**2.2.3.2 Anonymizácia dátových záznamov** IPFIX exporty krížom cez administratívne domény, tak ako to bude popísané v prípade 4 v Kapitole 2 (Sekcia 2.2.3.7), môžu byť použité na monitorovanie sieťovej prevádzky na veľké vzdialenosti, napríklad pre analýzu dátových trendov v Internete. Pri takomto použití sa musia administrátori riadiť pravidlami pre ochranu súkromia a predísť monitorovaniu dôvernej sieťovej prevádzky cudzími osobami. Typicky anonymizačné techniky umožňujú poskytovanie sieťových dát iným osobám bez porušenia týchto zásad.

Všeobecne platí, že anonymizácia upraví sadu dát tak, aby chránila identitu ľudí alebo subjektov, ktorých sa súbor dát týka. Zároveň sa pokúša zachovať dáta tak, aby boli stále zmysluplné pre danú analýzu ale súčasne nemôžu byť stopovateľné naspäť ku konkrétnym sieťam, pracovným staniciam, alebo používateľom generujúcim tieto dáta. Napríklad, anonymizácia IP adresy je veľmi dôležitá pre zamedzenie identifikácie užívateľov, alebo smerovačov.

Jedným z možných prevedení v tomto prípade používa anonymizačnú funkciu v

exportéri. To však príliš zvyšuje zaťaženie exportéra. Flexibilnejšia implementácia využíva samostatný IPFIX Mediátor medzi exportérom a kolektorom.

**2.2.3.3 Selektívne spracovanie dátových záznamov** Trendom v moderných dátových sieťach je súčasný prenos dát, hlasu a video komunikácie jednou spoločnou infraštruktúrou. Takéto siete nazývame konvergovanými sieťami. Počítačové siete paralelne prenášajú dáta viacerých protokolov ako napríklad IPv4, IPv6, VPN, MPLS a pod. Dátové záznamy každého z týchto protokolov musia byť analyzované oddelene a z rôznych perspektív pre rôzne organizácie.

Jeden kolektor pokrývajúci všetky typy dátových záznamov sa môže stať úzkym hrdlom zhromažďovacej infraštruktúry. Preto je lepšie distribuovať dátové záznamy na základe ich typu viacerým kolektorom, čo má za následok rozloženie záťaže. Pod typom dátového záznamu máme na mysli napríklad typ sieťového protokolu, ktorým boli dáta prenášané.

Jedna z možných implementácií v tomto prípade používa replikáciu IPFIX správy v exportéri pre viac kolektorov. Každý kolektor potom dekoduje tie dátové záznamy, ktoré jeho aplikácia potrebuje. To však zvyšuje zaťaženie exportovacieho procesu a mrhanie šírkou pásma medzi exportérom a kolektorom.

Sofistikovanejšie prevedenie používa sprostredkovateľský proces v exportéri, ktorý určuje, ktorému kolektoru sa dáta pošlú, v závislosti na hodnotách určitých polí. Ak exportér nemá túto schopnosť, posiela dátové záznamy IPFIX Mediátoru, a ten ich distribuuje kolektorom.

**2.2.3.4 Interoperabilita medzi protokolmi starších verzií a IPFIX** Počas migrácie z protokolov starších verzií ako napríklad NetFlow (Claise, 2004) na IPFIX zvyknú nástroje týchto protokolov súčasne existovať v jednej sieti. Napríklad aj po zavedení IPFIX kolektora je nutné monitorovať sieť, hoci exportér je verzie NetFlow.

Jedna z možností je použiť IPFIX Mediátor, ktorý bude konvertovať starší protokol na IPFIX.

**2.2.3.5 Prispôsobovanie granularity tokov** Najzákladnejšia sada kľúčov toku je päťica: *protokol, zdrojová a cieľová IP adresa, číslo zdrojového a cieľového portu*. Menšie sady kľúčov, teda trojice, dvojice, alebo len jeden samotný prvok (napr. *maska siete, BGP čísla autonómnych systémov*, a pod.) vytvárajú viac agregované záznamy o tokoch. Toto je vhodné pri meraní na úrovni jadra sieťovej domény, alebo pri manipulovaní s výkonnosťou exportérov a kolektorov.

Najvhodnejšia implementácia predstavuje konfigurovateľný merací proces v exportéri. Administrátor špecifikuje požadovanú sadu kľúčov toku a tým pádom exportér generuje záznamy o toku želanej zrnitosti.

V opačnom prípade, kde merací proces nemá schopnosť nastavovať kľúče toku exportéra, IPFIX Mediátor môže agregovať dátové záznamy na základe definovaných kľúčov vo svojej konfigurácii.

**2.2.3.6 Spájanie času** Spájanie resp. kompozícia času je definovaná ako agregácia za sebou idúcich dátových záznamov s rovnakými kľúčmi toku. Tento proces vedie k rovnakému výstupu ako nastavenie dlhšieho aktívneho timeoutu (*active timeout*) v exportéri, no má jednu výhodu. Nové metriky ako napríklad výpočet priemerných, maximálnych, alebo minimálnych hodnôt zo záznamov o tokoch v kratších časových intervaloch umožňujú presnejšie výsledky a sledovanie aj menších zmien.

Jedna z možných implementácií je použitie sprostredkovateľského procesu umiestneného medzi meracím a exportovacím procesom exportéra. Druhou možnosťou je samostatný IPFIX Mediátor situovaný medzi exportérom a kolektorom. Táto možnosť prináša väčšiu flexibilitu a nezatažuje exportér ďalšími výpočtami.

**2.2.3.7 Spájanie priestoru** Spájanie priestoru je vlastne zoskupenie dátových záznamov v rámci množiny pozorovacích bodov jednej pozorovacej domény, združovanie záznamov viacerých exportérov, alebo jedného exportéra ale viacerých pozorovacích domén. Delí sa na tieto štyri typy:

**1. Spájanie priestoru v rámci jednej pozorovacej domény**

Príkladom je meranie dátového toku jedného logického rozhrania, ktoré vzniklo agregáciou liniek podľa 802.3ad (IEEE 802.3ad, 2000).

**2. Spájanie priestoru viacerých pozorovacích domén jedného exportéra**

Tak isto ako v predchádzajúcom prípade aj tu ide o agregáciu viacerých fyzických liniek.

**3. Spájanie priestoru niekoľkých exportérov**

Dátové záznamy namerané v rámci jednej administratívnej domény môžu byť zlučované.

**4. Spájanie priestoru administratívnych domén**

Dátové záznamy zaznamenané vo viacerých administratívnych doménach, ako napríklad v rôznych klientských sieťach, alebo v sieťach rozdielnych poskytovateľov Internetového pripojenia môžu byť tiež zlučované. Kolektor vie na základe IP adresy exportéra rozlíšiť, ktorej klientskej sieti exportér patrí a tak rozlišovať klientske dáta.

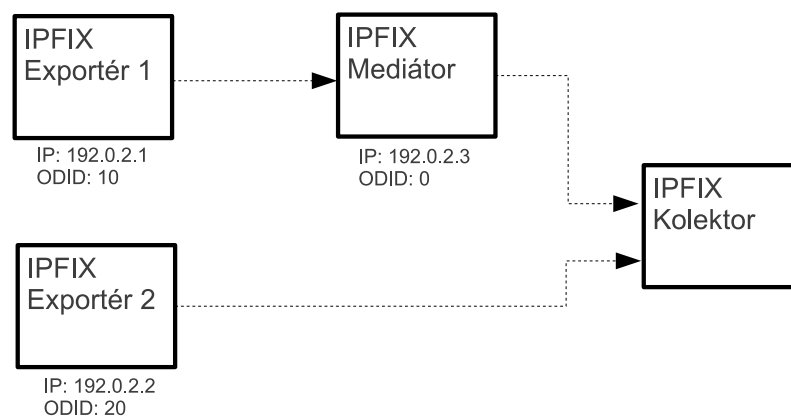
Implementácia pomocou sprostredkovateľského procesu umiestneného v exportéri rieši prípady 1 a 2. Separátny IPFIX Mediátor je riešením pre všetky štyri prípady.

## **2.2.4 Vybrané implementačno-špecifické problémy IPFIX Mediátora**

**2.2.4.1 Strata informácie o pôvodnom exportéri** Pri využívaní sprostredkovania správ v IPFIX dochádza k strate potrebných informácií. V prvom rade to je IP adresa exportéra, ktorá bola získavaná zo zdrojovej IP adresy transportnej relá-

cie, rovnako ako identifikačné číslo pozorovacej domény, ktoré je zahrnuté v hlavičke IPFIX správy. V niektorých prípadoch môže Mediátor zahodiť tieto informácie úmyselne. Vo všeobecnosti však platí, že kolektor musí rozpoznať pôvod nameraných dát, ako napríklad IP adresu exportéra, ID pozorovacej domény (Observation Domain ID – ODID), alebo dokonca ID pozorovacieho bodu. Ak Mediátor tieto informácie o exportéri neoznami kolektoru, tak ten nesprávne usúdi, že IP adresa Mediátora je adresa pôvodcu dát (exportéra).

V nasledujúcom Obrázku 2 – 10 kolektor vie rozoznať dve IP adresy - 192.0.2.3 (Mediátor) a 192.0.2.2 (Exportér 2). To nie je správne. Mediátor musí informovať Kolektor o IP adrese Exportéra 1.



**Obr. 2 – 10** Strata informácie o originálnom exportéri

**2.2.4.2 Strata informácie o čase exportu** Pole čas exportu *export time*, ktoré je zahrnuté v hlavičke správy predstavuje referenčnú časovú známku dátové záznamy. Niektoré informačné elementy popísané v (Quittek, et al., 2008) nesú časové známky delta *delta timestamps*, ktoré udávajú časový rozdiel voči hodnote v poli čas exportu. Ak dátový záznam zahŕňa nejaké pole s časovou známkou delta a Mediátor prepíše hodnotu času exportu, tak časová známka delta týmto stráca význam. Kolektor však túto situáciu nevie rozpoznať a tak pracuje so zlými hodnotami.

**2.2.4.3 Interpretácia sprostredkovaných správ** V niektorých prípadoch potrebuje kolektor vedieť, ktoré konkrétne operácie resp. funkcie vykonal Mediátor nad dátovými záznamami. Kolektor nedokáže rozlíšiť medzi spájaním času a spájaním priestoru, v prípade, že Mediátor neexportuje použitú funkciu. Niektoré parametre vzťahujúce sa k funkcii by tiež mali byť exportované.

V prípade spájania času, kolektor musí poznať aktívny časovač *active timeout* pôvodných záznamov o tokoch. Pri spájaní priestoru je potrebné poznať nad akou oblasťou bola vykonaná kompozícia dátových záznamov. (Kobayashi, Claise, 2010)

## 2.3 Analýza aplikačného rámca pre IPFIX Mediátor

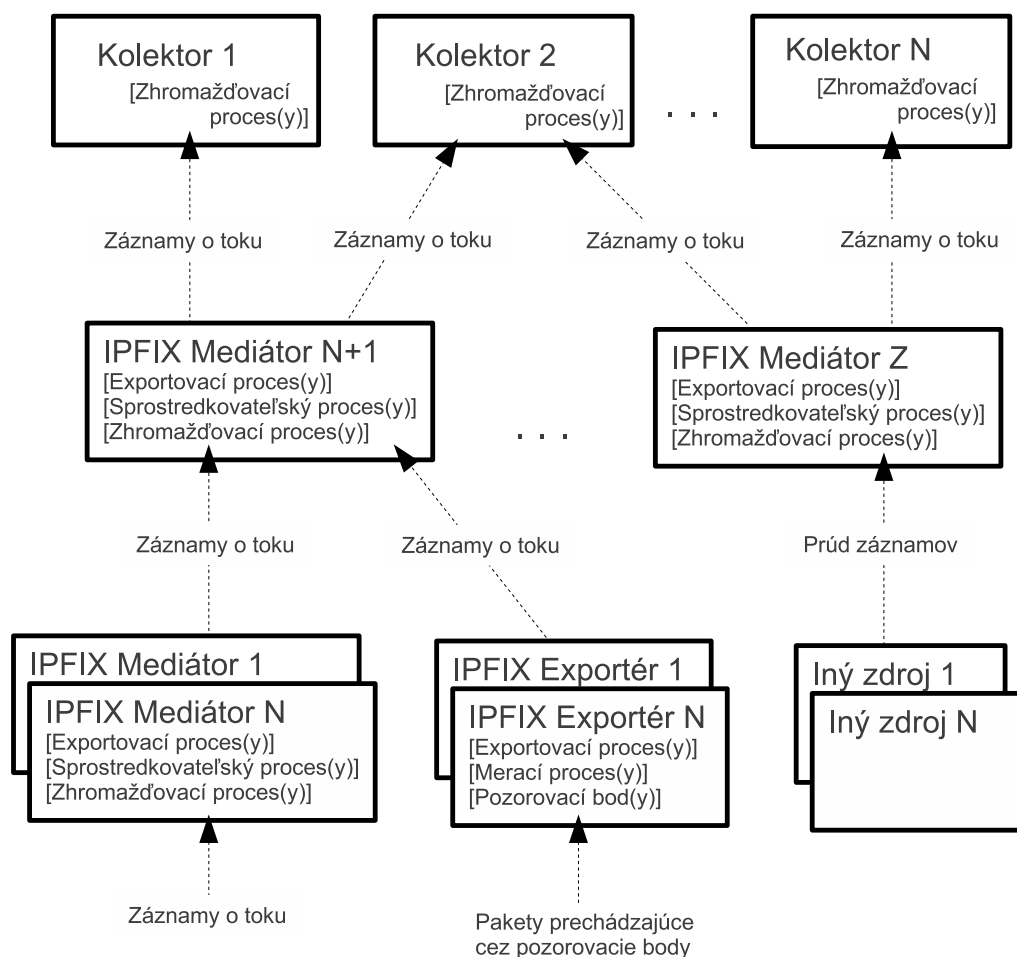
Analýze aplikačného rámca pre sprostredkovanie správ v IPFIX sa venuje RFC 6183 (Kobayashi et al., 2011). V nasledujúcich kapitolách si podrobnejšie priblížime referenčný model a vybrané komponenty, či procesy aplikačného rámca.

### 2.3.1 Referenčný model sprostredkovania správ v IPFIX

Obrázok 2–11 predstavuje referenčný model sprostredkovania správ v IPFIX ako rozšírenie referenčného modelu IPFIX, popísaného v *Architecture for IP Flow Information Export* (Sadasivan, et al., 2009). Táto schéma zobrazuje možné scenáre, ktoré môžu existovať v meracej architektúre. Funkčné komponenty v rámci každej entity sú ohraničené zátvorkami []. Mediátor môže prijímať záznamy o toku od iných mediátorov a exportérov a prúd záznamov z iných zdrojov. Za iné zdroje sa považujú nástroje iných protokolov, ako napríklad NetFlow exportéry (Claise, 2004). Spracované dáta vo forme záznamov o toku potom exportuje jednému alebo viacerým kolektorom a mediátorom.

Zjednodušený model komponentov IPFIX mediátora je zobrazený na Obrázku 2–12. Mediátor obsahuje jeden alebo viac sprostredkovateľských procesov, hierarchicky

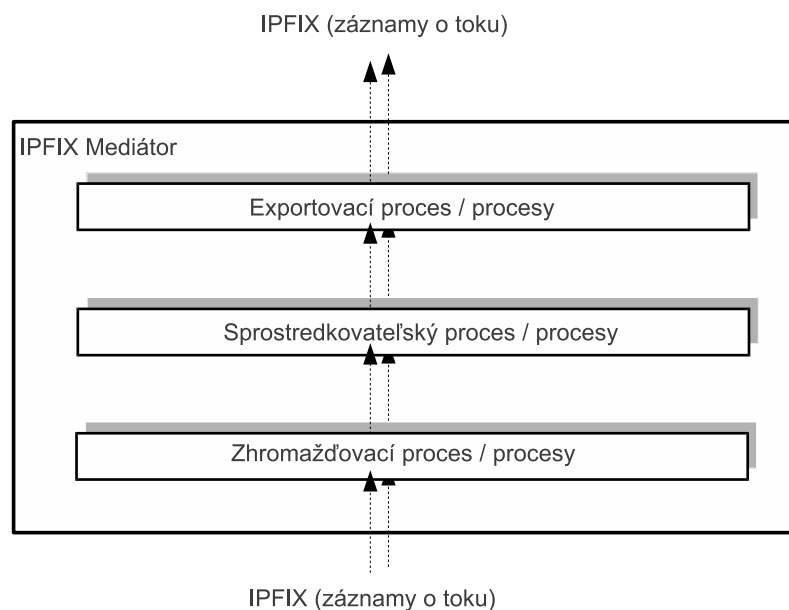
uložených medzi jedným alebo viacerými exportovacími a zhromažďovacími procesmi. Tento model sa týka najbežnejšieho prípadu, kedy mediátor prijíma dátové záznamy od exportéra, alebo iného mediátora.



Obr. 2 – 11 Referenčný model sprostredkovania správ v IPFIX

### 2.3.2 Komponenty sprostredkovania správ v IPFIX

V nasledujúcich častiach si bližšie priblížime jednotlivé komponenty IPFIX mediátora, ktoré sú znázornené na Obrázku 2 – 12.



Obr. 2 – 12 Zjednodušený model komponentov IPFIX Mediátora

**2.3.2.1 Zhromažďovací proces** Zhromažďovací proces v IPFIX Mediátore sa nelíši od zhromažďovacieho procesu popísaného v špecifikácii IPFIX protokolu (Claise et al., 2008). Jedinou funkciou navyše je odovzdanie sady dátových záznamov a riadiacich informácií jednému, alebo viacerým komponentom, tj. sprostredkovateľským procesom, alebo ďalším aplikáciám. To znamená, že zhromažďovací proces môže vytvárať kópie sady a prenášať ich buď sériovo, alebo paralelne. Medzi riadiace informácie patrí hlavička IPFIX správy, informácie o transportnej relácii, spolu s informáciami o meracom a exportovacom procese v exportéri, napr. vzorkovacie parametre.

**2.3.2.2 Exportovací proces** Exportovací proces IPFIX Mediátora sa vo svojej podstate tiež nelíši od toho, ktorý je popísaný v špecifikácii protokolu (Claise et al., 2008). Prídavné funkcie môžu byť nasledujúce:

- Prijímať spúšťač *trigger* od sprostredkovateľských procesov, ktorý vyvolá odoslanie správy kolektoru na odstránenie neplatnej šablóny (*Template Withdrawal*).



*val Message*).

- Z dôvodu uvedeného v Kapitole 2 (Sekcia 2.2.4.1), je potrebné preposielať informácie o pôvodcovi dát (exportérovi), napríklad ID pozorovacieho bodu a pozorovacej domény, IP adresa exportéra atď. Exportovací proces tieto dáta zakóduje do prídavných dátových záznamov, buď s využitím informačných elementov skupiny 2 (Tabuľka 2–3), alebo organizáciou špecifikovaných elementov.

**Tabuľka 2–3** Prehľad informačných elementov skupiny 2

ID	názov informačného elementu
130	exporterIPv4Address
131	exporterIPv6Address
217	exporterTransportPort
211	collectorIPv4Address
212	collectorIPv6Address
213	exportInterface
214	exportProtocolVersion
215	exportTransportProtocol
216	collectorTransportPort
173	flowKeyIndicator

**2.3.2.3 Sprostredkovateľské procesy** Sprostredkovateľské procesy sú kľúčovými funkčnými blokmi sprostredkovania správ v IPFIX. Rôzne procesy pokrývajú každý príklad použitia sprostredkovania správ z Kapitoly 2 (Sekcia 2.2.3). Mediátor musí byť schopný súčasne podporovať viac ako jeden sprostredkovateľský proces. Spolupráca viacerých procesov je konfigurovaná nasledujúcimi spôsobmi.

- **Paralelné spracovanie** - Prúd záznamov je spracovaný viacerými sprostredkovateľskými procesmi paralelne tak, aby boli splnené požiadavky koncových

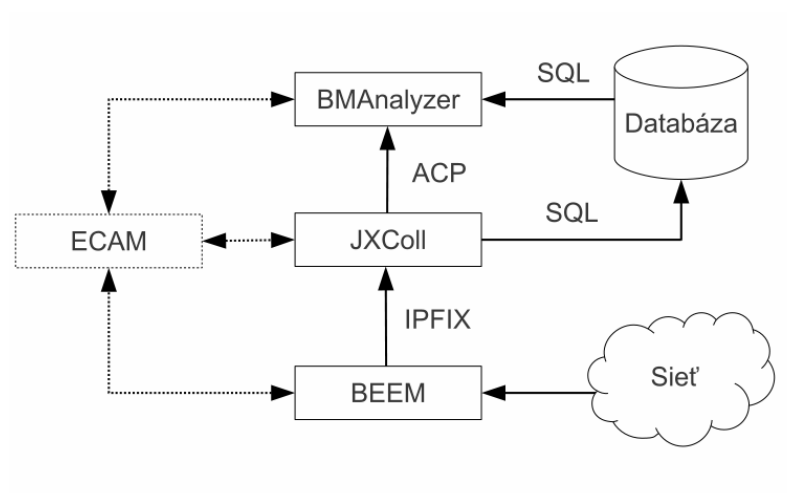
aplikácií. V tomto scenári, každý sprostredkovateľský proces dostáva kópiu celého prúdu záznamov ako vstup.

- **Sériové spracovanie** - Aby bolo zabezpečené flexibilné spracovanie prúdu záznamov, sprostredkovateľské procesy sú zapojené sériovo. V tomto prípade výstupný prúd záznamov jedného procesu je vstupným prúdom nasledujúceho.

### 3 Projekty výskumnej skupiny MONICA

#### 3.1 Meracia platforma BasicMeter

BasicMeter (MONICA, 2013) je jedným z projektov výskumnej skupiny MONICA, sídliacej v Laboratóriu počítačových sietí (CNL) na Technickej univerzite v Košiciach. Je to merací nástroj založený na protokole IPFIX. Slúži na pasívne meranie parametrov prevádzky počítačových sietí a ich následné vyhodnocovanie. Začiatky vývoja siahajú až do roku 2003. Jeho architektúra je znázornená na Obrázku 3–1.



**Obr. 3–1** Architektúra nástroja BasicMeter (Kudla, 2010)

Platforma pozostáva z nasledujúcich komponentov:

- **BEEM** – merací a exportovací proces – exportér,
- **JXColl** – zhromažďovací proces – kolektor,
- **BMAnalyzer** – aplikácia na vyhodnocovanie údajov,
- **ECAM** – riadiaci komponent nástroja,
- **bmIDS** – systém pre detekciu narušenia.

Najnižšou vrstvou architektúry je *BEEM*. Zabezpečuje všetky funkcie meracieho

a exportovacieho procesu definovaného v špecifikácii IPFIX. Namerané záznamy o tokoch posíla komponentu *JXColl* vo formáte IPFIX správ. Kolektor dekoduje prijaté správy od jedného alebo viacerých exportérov a ukladá ich do databázy kvôli neskoršej analýze. Za účelom analyzovania dát a ich grafického zobrazenia vo forme grafov v reálnom čase ich posíla komponentu *BMAalyzer* prostredníctvom protokolu ACP (Pekár, 2009). *bmIDS* tak isto prijíma dáta v reálnom čase, no jeho úlohou je analyzovať prebiehajúcu komunikáciu v uzli siete a odhaľovať prípadné útoky, resp. anomálie. *ECAM* umožňuje centrálné riadiť beh jednotlivých častí architektúry. Medzi jeho funkcie patrí vytvorenie a zmazanie inštancie exportérov a kolektorov, prípadne meniť ich konfiguráciu. (Kudla, 2010; Vereščák, 2012)

### 3.2 Merací nástroj SLAmeter

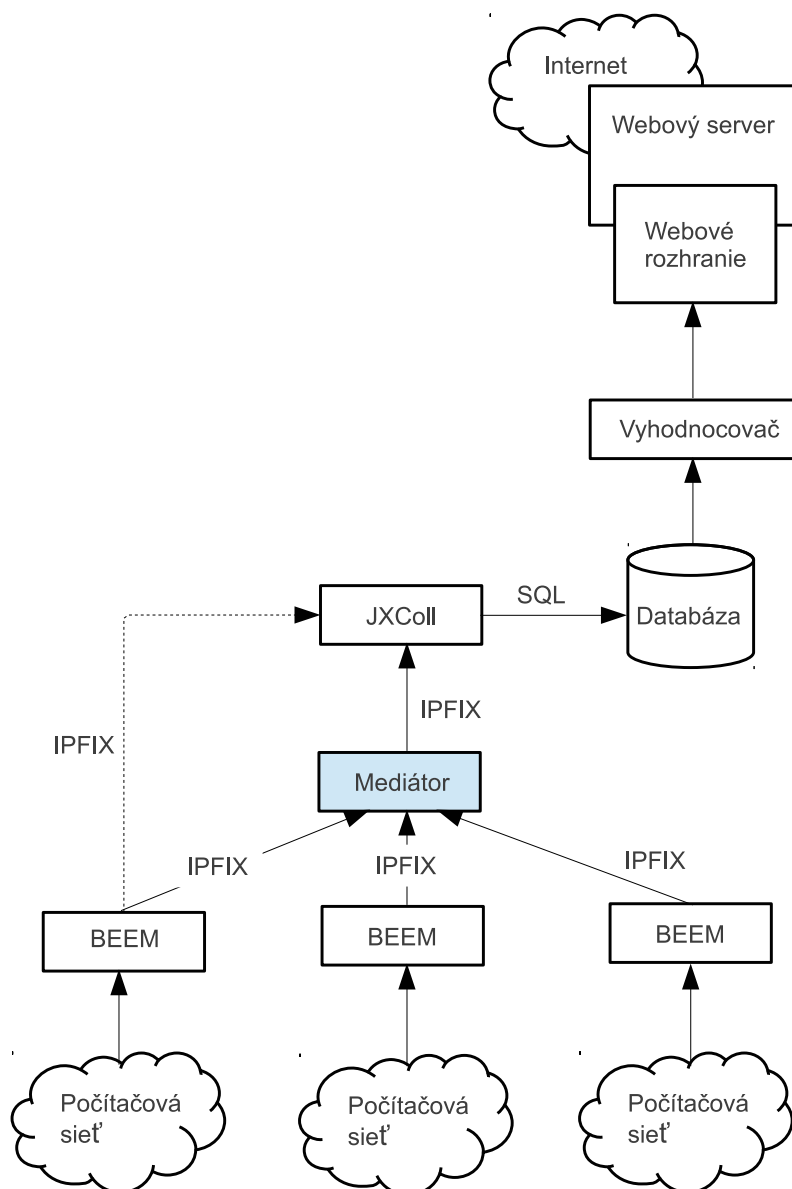
SLAmeter je merač parametrov sieťovej prevádzky, ktorý vyhodnocuje dodržiavanie zmluvy o úrovni poskytovanej služby (*SLA*). V tomto prípade sa pod poskytovanou službou rozumie prístup do siete Internet. Základ nástroja je postavený na komponentoch nástroja BasicMeter, a rovnako je projektom výskumnej skupiny MONICA.

Cieľom nástroja je spracovať vybrané parametre sieťovej prevádzky a vypočítať z nich akúsi triedu kvality. SLAmeter slúži každému, kto si chce skontrolovať kvalitu svojho pripojenia do Internetu. Triedy umožňujú jednoduchý spôsob porovnávania jednotlivých pripojení ponúkané poskytovateľmi, čo by malo mať dopad na konkurenčný boj a zvýšenie snahy o zlepšovanie kvality služieb. (SLAmeter, 2013)

Ako bolo spomenuté, SLAmeter je akousi nadstavbou na BasicMeter. Jeho architektúra pozostáva z exportérov, ktoré posielajú namerané záznamy o tokoch zhromažďovaču. Ten spracováva záznamy a ukladá ich do centrálnej databázy. Úlohou *Vyhodnocovača* je na základe požiadaviek od webového rozhrania spracovávať IPFIX záznamy a vytvárať tak štatistické a analytické údaje o charaktere meranej sieťovej prevádzky (Vyhodnocovač, 2013). *Webové rozhranie* je modulárna webová aplikácia

s pohľadmi pre zákazníka a poskytovateľa Internetových služieb.

Príklad architektúry nástroja SLAmeter so zapojením Mediátora je na Obrázku 3–2. BEEM môže exportovať IPFIX správy priamo kolektoru JXColl. No v prípade, že chce využiť sprostredkovateľské procesy na modifikáciu údajov, posiela správy Mediátoru a ten ich ďalej preposiela kolektoru.



**Obr. 3–2** Príklad architektúry nástroja SLAmeter s využitím Mediátora

## 4 Návrh a implementácia aplikačného rámca pre IPFIX Mediátor

Na základe analýzy aplikačného rámca pre sprostredkovanie správ v IPFIX, pozri Kapitulu 2 (Sekcia 2.3) a analýzy exportovacieho a zhromažďovacieho procesu v RFC 5101 (Claise et al., 2008) boli zhrnuté požiadavky a navrhnutá samotná architektúra IPFIX Mediátora. Jeho jednotlivým komponentom, ktoré sú vyššie znázornené na Obrázku 2–12 a požiadavkám sú venované nasledujúce kapitoly. Upozorňujeme, že termíny „sprostredkovateľský proces“, „sprostredkovateľský modul“, resp. „modul“ sú úplne totožné a zameniteľné.

### 4.1 Požiadavky na rámec pre IPFIX Mediátor

- **Modulárna implementácia** – už počas analýzy sprostredkovania správ v IPFIX bolo zrejmé, že aplikačný rámec musí byť modulárny. Musí mať podporu pre jednoduché a flexibilné pridávanie resp. odoberanie modulov. Opäť zdôrazňujeme, že predstaviteľom modulu je sprostredkovateľský proces.
- **Oddelenie logiky rámca od logiky sprostredkovateľských procesov** – najdôležitejšie je, aby budúci riešitelia sprostredkovateľských procesov nemuseli vôbec zasahovať do zdrojového kódu aplikačného rámca. Všetky potrebné metódy pre prácu so záznamami o tokoch im musí zabezpečiť rámec, ale rovnako im musí zakázať prístup k jeho interným metódam. Iba tak môže byť zachovaná jednotnosť prístupu. Nie je možné, aby každý sprostredkovateľský proces riešil napr. zakódovanie, alebo dekodovanie dátových záznamov po svojom. Preto je potrebné navrhnuť a implementovať rozhranie, prostredníctvom ktorého budú procesy komunikovať s aplikačným rámcom.
- **Dynamické načítavanie sprostredkovateľských procesov** – pridávanie a odoberanie sprostredkovateľských procesov musí byť riadené výlučne cez

konfiguračný súbor. Nesmie byť potrebný žiadny zásah do zdrojového kódu rámca.

- **Jednoduchá konfigurácia** – ako už bolo spomenuté v analýze – Kapitola 2 (Sekcia 2.3.2.3), sprostredkovateľské procesy spracúvajú prijaté záznamy o tokoch buď sériovo, alebo paralelne. Celková štruktúra odovzdávania dát v rámci mediátora od zhromažďovacieho procesu, sériovo a paralelne cez všetky procesy a napokon až k exportovaciemu procesu musí byť jednoznačne konfigurovateľná v textovom XML súbore a v čo najviac používateľsky priateľskom formáte.
- **Distribúcia dát medzi komponentmi** – aplikačný rámec musí zabezpečiť spôsoby prenosu dát medzi jednotlivými sprostredkovateľskými procesmi, zhromažďovacím a exportovacím procesom. Tieto spôsoby musia byť pre procesy jednotné a bez možnosti zmeny z vnútra sprostredkovateľského procesu.
- **Jediná inštancia modulov** – bolo navrhnuté, že každý sprostredkovateľský proces musí byť implementovaný podľa návrhového vzoru *Singleton*. Je to z toho dôvodu, že každý proces musí byť unikátny a jednoznačne rozpoznateľný v rámci celého programu na základe mena triedy procesu. Konfigurácia toku dát cez procesy spomínaná vyššie bude daná práve prostredníctvom názvov ich tried. Jedinečnosť procesov musí zabezpečiť aplikačný rámec.
- **Jednotné dekódovanie a zakódovanie dátových záznamov** – aplikačný rámec musí obsahovať metódy prístupné všetkým sprostredkovateľským procesom, ktoré budú dekódovať dátové záznamy na dáta a opačne na základe šablón.
- **Viacvláknovosť** – nielen zo samotnej povahy paralelných procesov, ale aj modularity vyplýva, že každý sprostredkovateľský proces bude vykonávaný v samostatnom vlákne, prípadne viacerých vláknach. Podobne zhromažďovací a exportovací proces budú rozdelené na viac vlákien.

- **Konformita s protokolom IPFIX** – výstupom Mediátora musia byť správy zakódované v konformite so špecifikáciou IPFIX protokolu. Kolektor spracováva správy prijaté od Mediátora rovnakým spôsobom, akoby ich prijal od exportéra. Navyše zhromažďovací a exportovací proces aplikačného rámca sa nesmú líšiť od charakteristík týchto procesov daných špecifikáciou IPFIX protokolu v RFC 5101 (Claise et al., 2008).
- **Komunikácia pomocou UDP** – v tejto fáze projektu bol zvolený ako komunikačný protokol UDP. Dôvodom bola rýchlosť, jednoduchosť a dobré skúsenosti s implementáciou UDP v JXColl. Napriek tomu, že UDP nie je spojovo orientovaný transportný protokol, v prípade nasadenia Mediátora v SLAmetri to vôbec nevadí. Mediátor bude nasadený fyzicky na tej istej lokálnej sieti ako exportér a kolektor.
- **Programovací jazyk Java** – pre implementáciu bol vybraný programovací jazyk Java. Hlavným dôvodom bol fakt, že zhromažďovací proces Mediátora a IPFIX kolektora sú veľmi podobné a JXColl je naprogramovaný v tomto jazyku. Ďalším faktom je relatívne jednoduchá tvorba modulárnych aplikácií, vďaka načítavaniu tried pomocou *Java ClassLoader*. V neposlednom rade zohrala svoju rolu aj vysoká podpora jazyka Java, či už sa jedná o kvalitu dokumentácie, veľké množstvo odborných fór, dostupnosť knižníc jazyka, ale aj fakt, že Java aplikácie sú spustiteľné na väčšine operačných systémov.

## 4.2 Hlavná trieda Mediátora

Úlohou hlavnej triedy Mediátora je postupne spustiť všetky svoje vlákna a procesy potrebné pre beh programu. Najprv sa prečítajú a spracujú argumenty príkazového riadku. Program vie rozpoznávať dva druhy argumentov. Jedným z nich je cesta ku konfiguračnému súboru. Ak nie je zadaná, používa sa východiskový konfiguračný súbor. Druhým argumentom môže byť zadaná možnosť `--logtofile`. Vtedy sú všetky



logovacie výstupy presmerované zo štandardného výstupu do súboru.

Potom ako program načíta všetky nastavenia z konfiguračného súboru, spustí svoje moduly – sprostredkovateľské procesy pomocou triedy **IPLoader**. Nasleduje spustenie vlákna, ktoré prijíma IPFIX pakety prostredníctvom protokolu UDP a vlákna, ktoré ich spracováva. Hovoríme o **UDPServer** a **UDPProcessor**. Nakoniec je spustené exportovacie vlákno – **UDPExporter**. Kedykoľvek keď nastane chyba je Mediátor korektne ukončený a to tak, že uvoľní všetku pamäť a zastaví bežiacie vlákna. Rovnako je Mediátor zastavený po stlačení kombinácie kláves **Ctrl-c**. Podrobne o každom spomenutom vlákne a procese bude povedané v nasledujúcich kapitolách.

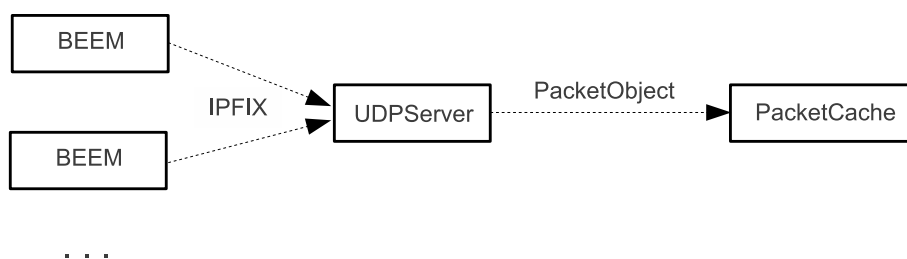
## 4.3 Zhromažďovací proces

Na základe analýzy a zhodnotenia požiadaviek na zhromažďovací proces bola navrhnutá jeho architektúra. Logická štruktúra procesu sa skladá z dvoch fáz, pričom každú fázu predstavuje jedno vlákno. Venujme sa teda jednotlivým fázam procesu.

### 4.3.1 1. fáza zhromažďovacieho procesu

Prvá fáza je znázornená na Obr. 4–1 a predstavuje najnižšiu vrstvu celého nástroja. Jej jadrom je UDP server, bežiaci v samostatnom vlákne. V jeho hlavnej metóde *run()* cyklicky vykonáva kód, dokiaľ nie je prerušený výnimkou *InterruptedException*. Tento kód odchyťáva údaje posielané protokolom UDP na úrovni bajtov a ukladá ich do vyrovnávacej pamäte. K tomuto je použitý objekt triedy jazyka Java – **ByteBuffer**. Zároveň sa do objektu triedy **InetSocketAddress** uloží IP adresa exportéra a zaznamená sa čas prijatia dát v milisekundách vo formáte časovej známky Unix-u (*Unix Timestamp*). Tieto tri premenné sú argumentom funkcie *write()*, ktorá z prijatých premenných vytvorí objekt typu **PacketObject**. Tento objekt je akousi abstraktnou reprezentáciou paketu, vo svojich členských premenných uchováva údaje z hlavičky IPFIX správy (sekvenčné číslo, čas exportu, ID pozoro-

vacej domény) spolu s obsahom správy, časom prijatia a adresy z ktorej bol prijatý. Prijatá IPFIX správa sa v tejto forme uloží do vyrovnávacej pamäte, ktorú predstavuje trieda `PacketCache`. Jej členská premenná *cache* je typu `ArrayBlockingQueue`, čo je vlastne Java implementácia *FIFO* frontu, ktorý je navyše synchronizovaný a optimalizovaný na vysoký výkon. Táto vyrovnávacia pamäť medzi prvou a druhou fázou zhromažďovacieho procesu je kritická vo vysoko rýchlostných sieťach.



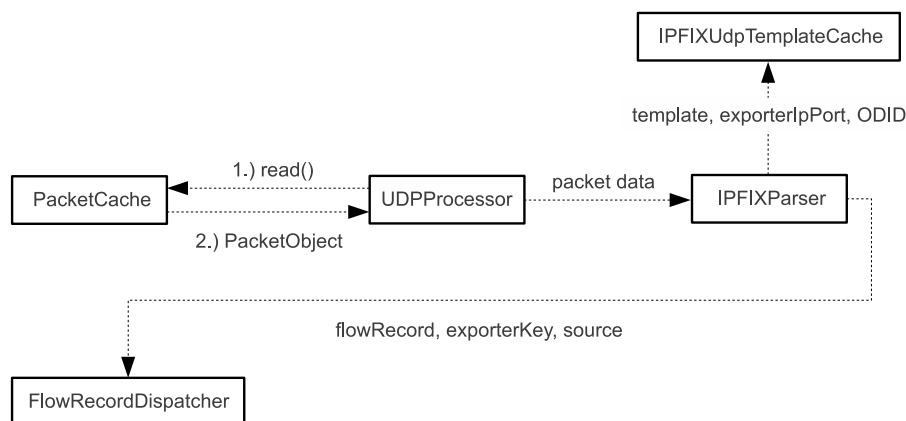
**Obr. 4–1** Schéma prvej fázy zhromažďovacieho procesu Mediátora

Návrh umožňuje do budúcnosti jednoduchú rozšíriteľnosť programu o triedy implementujúce serveri iných transportných protokolov, napr. TCP a SCTP. Tieto serveri budú rovnako ako `UDPServer` bežiacie v samostatných vláknach.

#### 4.3.2 2. fáza zhromažďovacieho procesu

Schému druhej fázy vidíme na Obr. 4–2. Hlavná metóda *run()* vlákna `UDPPProcessor` cyklicky vyberá dáta z `PacketCache` a odovzdáva ich *parseru*. Ten je reprezentovaný triedou `IPFIXParser`. Správy číta z vyrovnávacej pamäte dokiaľ nie je prerušený výnimkou *InterruptedException*. UDP procesor robí zároveň kontrolu, či prijaté dáta sú IPFIX paketom. V tomto prípade, ale aj v prípadoch keď prijaté dáta sú poškodené sa správa zahadzuje a program pokračuje spracovávaním ďalšej správy. Trieda `IPFIXParser` sa používa na spracovanie prijatých paketových binárnych dát, ktorého výsledkom je hotový objekt IPFIX správy, obsahujúci všetky jej komponenty, pozri Kapitolu 2 (Sekcia 2.1.2). Metódy tejto triedy najprv vyskladajú kompletnú hlavičku správy a potom sa pustia do parsovania IPFIX sád. Na základe identifi-

kátora sady spracujú a vytvoria objekty pre sadu šablón, sadu šablón možností a dátovú sadu. Sady následne naplnia príslušnými záznamami.



Obr. 4 – 2 Schéma druhej fázy zhromažďovacieho procesu Mediátora

Prichádzajúce záznamy šablón a záznamy šablón možnosti sú posielané špecializovanej triede `IPFIXUdpTemplateCache`, ktorá má na starosti správu prijatých šablón osobitne pre každý exportér. Ako je zrejmé zo špecifikácie exportovacieho procesu, šablóna musí byť odoslaná kolektoru okamžite ako je vytvorená a ešte pred odoslaním jej prislúchajúcich dátových záznamov. Potom je šablóna periodicky preposielaná. Trieda `IPFIXUdpTemplateCache` nové šablóny ukladá. Objekty šablón, ktoré už má uložené ale aktualizuje. Zároveň maže staré šablóny, ku ktorým nedostala aktualizáciu po dobu definovanú v konfiguračnom súbore.

Napokon sa každý dátový záznam v dátovej sade, s prislúchajúcou šablónou a hlavičkou IPFIX správy zabalí do objektu triedy `IPFIXFlowRecord`, ktorá je reprezentáciou záznamu o toku. Tento záznam spolu s reťazcom, ktorý určuje odkiaľ záznam vystupuje (*inputProcess*) sú posielané ako parametre triede `FlowRecordDispatcher`. V tomto prípade reťazec, ktorý určuje pôvodcu záznamu je „*exporter*“. Dispečer záznamov o tokoch – `FlowRecordDispatcher` rozdistribuuje prijaté záznamy o tokoch príslušným sprostredkovateľským procesom, alebo ich pošle na export. O tom podrobnejšie v samostatnej Kapitole 4 (Sekcia 4.5).

## 4.4 Rozhranie a podpora pre sprostredkovateľské procesy – moduly

Vyššie boli definované viaceré požiadavky na sprostredkovateľské procesy, ktoré musí zabezpečiť aplikačný rámec. Hovoríme o modulárnej implementácii, oddelení logiky rámca od logiky procesov, ich dynamické načítavanie, jediná inštancia procesov atď. Riešenie týchto požiadaviek bude vysvetlené v nasledujúcich kapitolách. Začnime ale najprv teoretickým základom dynamického načítavania tried.

### 4.4.1 Java ClassLoader a dynamické načítavanie tried

*Java ClassLoader* je súčasťou *Java Runtime Environment* (JRE) a jeho úlohou je dynamické načítavanie tried jazyka Java do *Java Virtual Machine* (JVM) na základe ich mena. Načítavanie tried je jeden zo základných a najsilnejších mechanizmov, ktorý poskytuje programovací jazyk Java. Vďaka nemu JRE nemusí vedieť nič o súboroch a súborovom systéme počas vykonávania Java programov. Navyše tieto súbory tried nie sú načítavané do pamäte naraz, ale podľa požiadaviek programu. (Mcmanis, 1996; Travis, 2001)

Načítavanie tried je organizované do stromovej štruktúry. Koreňom štruktúry je samozavádzací (*bootstrap*) *class loader*, ktorý je napísaný v natívnom kóde a nie je možné vytvárať jeho inštancie. Vytvára ho JVM a je zodpovedný za načítanie interných tried Java Development Kit (JDK) a *java.\** balíčkov obsiahnutých v JVM. Príkladom je `java.lang.String`. Potomkom samozavádzacieho loadera je *extension class loader*, ktorého primárnou zodpovednosťou je načítavať triedy z *extension* priečinkov. Toto je pohodlný spôsob rozšírenia JDK bez pridávania položiek do premenných prostredia – *CLASSPATH*. Rozšírením *extension classloader*-a je aplikačný, resp. *systémový class loader*. Jeho úlohou je načítanie tried z cesty danej premennou prostredia. Inštanciu systémového class loadera získavame metódou `ClassLoader.getSystemClassLoader()`. (TechJava, 2008; Antl, 2012)

Abstraktná trieda `ClassLoader` je umiestnená v balíčku `java.lang`. Vývojári môžu pridať vlastnú funkcionality do načítavania tried vytvorením vlastnej, ktorá bude rozšírením triedy `ClassLoader`. Typickou stratégiou je transformovanie mena triedy na meno súboru a následné prečítanie „súboru triedy“ (*class file*) zo súborového systému. (Christudas, 2005; Oracle, 2011)

Na dynamické načítavanie tried na základe ich binárneho mena sa používa metóda `loadClass(String name)`. Táto metóda bola použitá pri načítavaní modulov definovaných v konfiguračnom súbore a aj pri získavaní inštancií sprostredkovateľských procesov, ktoré boli potrebné pri riadení toku dát medzi procesmi. Podrobnejšie sa tomu venujeme v príslušných kapitolách.

#### 4.4.2 Abstraktná trieda `AIntermediateProcess`

Po analýze požiadaviek bolo jasné, že je potrebné pripraviť akési rozhranie pre sprostredkovateľské procesy, ktoré by oddeľovalo ich logiku od logiky aplikačného rámca. Navyše toto rozhranie musí definovať základné vlastnosti, ktoré sú rovnaké pre všetky procesy a implementovať metódy, ktoré majú byť procesom dostupné. Jednoznačnou odpoveďou na tieto otázky je abstraktná trieda, od ktorej budú všetky moduly dediť.

**4.4.2.1 Viacvláknosť** Každý modul musí byť vykonávaný v samostatnom vlákne. Preto trieda `AIntermediateProcess` dedí od triedy `Thread` a obsahuje abstraktnú metódu `run()`, čo je vlastne deklaráciou hlavnej metódy vlákien. Toto zabezpečí, že v každom module bude musieť byť jej konkrétna implementácia.

**4.4.2.2 Jediná inštancia modulov** Ďalšou požiadavkou bolo, že moduly musia byť implementované podľa návrhového vzoru Singleton. V prípade, že by existovalo viacero inštancií každého modulu, trieda `FlowRecordDispatcher` by nemohla

správne distribuovať záznamy o tokoch medzi procesmi. Vzorová implementácia návrhového vzoru singleton je nasledovná:

```
public class Singleton {  
    private static Singleton instance = null;  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        if (instance == null) {  
            instance = new Singleton();  
        }  
        return instance;  
    }  
}
```

Nedá sa spoľahnúť na to, že budúci vývojári modulov budú implementovať sprostredkovateľské procesy podľa tohoto návrhového vzoru, preto to musí zabezpečiť aplikačný rámec, presnejšie trieda od ktorej dedia – `AIntermediateProcess`. Bohužiaľ v jazyku Java nie je možné aby Singleton implementovala abstraktná trieda, hneď z viacerých dôvodov (vymenované len niektoré):

1. Singleton vyžaduje konštruktor s viditeľnosťou *private*. Toto sa vylučuje s možnosťou dedenia od abstraktnej triedy.
2. Členská premenná `instance` je typu *static*. Teda sa viaže k triede Singleton a nie k jej potomkom.
3. V prípade, že členská premenná `instance` má hodnotu *null*, je potrebné vytvoriť novú inštanciu potomka a nie rodičovskej triedy. Avšak inštanciu ktorého potomka?
4. Druhý prístup je deklarovať metódu `getInstance()` abstraktnou. Konkrétna im-

plementácia by tak bola zabezpečená triedami sprostredkovateľských procesov, premenná `instance` by bola správneho typu. Tento návrh je však nerealizovateľný. Metóda `getInstance()` rodičovskej triedy nemôže byť statická a zároveň abstraktná. V jazyku Java to nie je možné. *Static* metóda patrí triede, kde je definovaná. Pričom *abstract* naznačuje, že funkcionality bude definovaná až v potomkoch. Tu dochádza k logickému rozporu.

Preto bolo potrebné navrhnúť a implementovať iný spôsob, ktorý by zabezpečil jedinou inštanciu pre všetky moduly z abstraktnej rodičovskej triedy. Riešenie navrhol britský Java programátor *Niall Gallagher* na jednom z diskusných fór o probléme dedenia a návrhového vzoru Singleton (Gallagher, 2010). Jeho riešenie je hybridom viacerých prístupov, ktoré sa diskutujú na Internete, no vychádza z návrhového vzoru *Factory method*. Výsledkom je abstraktná trieda, slúžiaca ako továreň na podtriedy tým, že volá jej statická metóda `getInstance(Class clazz)`.

```
private static final Map singletonRegistry = new HashMap();

public static final synchronized
<T extends AIntermediateProcess> T getInstance(Class clazz) {
    T instance = (T) singletonRegistry.get(clazz);
    if (instance == null) {
        try {
            instance = (T) clazz.newInstance();
        } catch (InstantiationException | IllegalAccessException ex) {
            log.error(ex);
        }
        if (instance != null) {
            singletonRegistry.put(clazz, instance);
        } else {
            log.error(Could not register singleton);
        }
    }
}
```

```
    }  
}  
return instance;  
}
```

Ak sú splnené podmienky, že konkrétna trieda, napr. `SelectionProcess` je definovaná v rovnakom balíčku ako `AIntermediateProcess` a ich konštruktory nemajú explicitne nastavený prístup (predvoleným prístupom je „privátny v rámci balíčka“), tak jediným spôsobom ako získať inštanciu podtriedy mimo balíčka je cez konštrukciu:

```
SelectionProcess instance =  
AIntermediateProcess.getInstance(SelectionProcess.class);
```

Dalo by sa vyčítať, že vytváranie inšancií používa reflexiu, ktorá je pomalá. Avšak, keďže vytvárame Singleton-y, volanie `newInstance()` sa vykoná pre každý modul práve raz.

Aby bolo možné získavať inšancie modulov aj na základe mena triedy a nie len cez `class` objekty, bola vytvorená ďalšia metóda `getInstance(String processName)`. Parameter `processName` je práve meno procesu, napr `SelectionProcess`. Táto premenná sa prevedie na binárne meno procesu, podľa špecifikácie jazyka Java (Oracle, 2013), napr. `sk.tuke.cnl.Mediator.SelectionProcess`. Prostredníctvom systémového class loadera, tak ako to bolo vyššie spomínané, sa z binárneho mena sprostredkovateľského procesu získa jeho `class` objekt. Potom sa zavolá pôvodná metóda `getInstance(Class clazz)` a tá vráti inštanciu procesu.

```
String name = Default.PROCESSES_LOCATION + processName;  
Class clazz = ClassLoader.getSystemClassLoader().loadClass(name);  
instance = AIntermediateProcess.getInstance(clazz);
```



**4.4.2.3 Dekódovanie dátových záznamov** Bola vyslovená požiadavka, že aplikčný rámec musí obsahovať metódy pre dekódovanie a zakódovanie dátových záznamov na základe šablón. Nie je žiadúce, aby v budúcnosti každý vývojár sprostredkovateľských procesov riešil tieto úlohy po svojom.

Už v konštruktori triedy sa získa inštancia triedy `IPFIXElements`, ktorá poskytuje funkcie na jednoduché získanie informácií o podporovaných informačných elementoch. Trieda sparsuje XML súbor (*ipfixFields.xml*) a dáta uloží do hash mapy, ktorá používa mapovanie z ID informačného elementu na objekt typu `IpfixField`. Tento objekt obsahuje informácie o elementoch, ako napríklad meno, dátový typ, meno skupiny do ktorej patrí a identifikačné číslo. (Vereščák, 2012)

Na dekódovanie slúži metóda *decodeDataRecord()*, jej parametrami sú dátový záznam a príslušná šablóna. V prvej fáze je potrebné získať zakódovanú hodnotu z dátového záznamu, druhá fáza dekóduje bajty informačného elementu na hodnotu definovanú dátovým typom.

V cykle sa prechádzajú všetky špecifikátory poľa v šablóne. Na začiatku procesu sa pre každý špecifikátor určí číslo organizácie (ak je definované) a ID informačného elementu. Ak sa daný informačný element nenachádza v hash mape informačných elementov, tak je zaznamenaná chyba a pokračuje sa na spracovanie ďalšieho špecifikátora poľa. Potom sa získa meno, dátový typ a skupina informačného elementu. Posledne menované je skôr z informačných dôvodov, kvôli logovacím výpisom. Následne sa určí pozícia špecifikátora poľa v šablóne, pretože na rovnakej pozícii je uložená zakódovaná hodnota informačného elementu v dátovom zázname. Metóda teda získa tieto bajty, obalí ich do objektu triedy *ByteBuffer* a ten pošle spolu s pomenovaním dátového typu na dekódovanie triede `IPFIXDecoder`.

Na základe dátového typu je určená metóda, ktorá dekóduje bajty obsiahnuté v bufferi. Dekódovanie je implementované v súlade s RFC 5101 (Claise et al., 2008) a RFC 5102 (Quittek, et al., 2008) pre všetky dátové typy podporované protokolom

IPFIX. Vymenujme aspoň niektoré: znamienkové a bezznamienkové celé čísla na 8, 16, 32, 64 a 128 bitoch, čísla v pohyblivej rádovej čiarkke na 32 a 64 bitoch, dátumy, IPv4 a IPv6 adresy, MAC adresy a pod.

Dekódovaná hodnota je z dekodéra vrátená ako reťazec. Všetky hodnoty sa ukladajú do hash mapy, ktorá kvôli jednoduchému vyhľadávaniu prvkov asociuje názov informačného elementu na jeho hodnotu. Takáto dátová mapa so všetkými dekodovanými hodnotami z dátového záznamu je vrátená sprostredkovateľskému procesu, ktorý si ju vyžiadal.

**4.4.2.4 Zakódovanie dátových záznamov** Presným opakom predchádzajúcej metódy je metóda *encodeDataRecord()*. Jej parametrami sú dátová mapa (výsledok dekodovania), príslušná šablóna a počet polí v dátovom zázname, ktoré je potrebné zakódovať – *recordsCount*.

Na začiatku vykonávania metódy sa inicializuje pole objektov *ByteBuffer* o veľkosti *recordsCount*. Toto pole slúži ako pomocná premenná pre uchovávanie zakódovaných hodnôt. Zároveň sa vytvorí objekt nového dátového záznamu. Následne sa v cykle prechádzajú všetky hodnoty v dátovej mape. Pri každom prvku sa získa jeho identifikátor z inštancie triedy *IPFIXElements* na základe mena prvku. Vďaka tomu sa potom dá určiť číslo organizácie, dátový typ a pozícia špecifikátora poľa v šablóne.

Keď sú určené všetky potrebné hodnoty, tak dátový typ a hodnota informačného elementu sú poslané triede *IPFIXEncoder* na zakódovanie. Táto trieda bola navrhnutá a implementovaná analogicky k dekodéru. Aj tu sú pokryté všetky dátové typy, ktoré podporuje IPFIX protokol vrátane jedného navyše – bezznamienkového celého čísla na 128 bitoch – *unsigned128*. Tento dátový typ využívajú niektoré informačné elementy definované Laboratóriom počítačových sietí na Technickej univerzite v Košiciach. Podľa špecifikácie (Claise et al., 2008) musia byť zakódované informačné elementy posielané v sieťovom poradí bajtov, známom tiež ako *Big-Endian*.

Na základe dátového typu je určená funkcia, ktorá vykoná kódovanie. Tieto funkcie musia uskutočniť radu kontrol, či je vstupná hodnota v reťazci validná. Pri číselných typoch a dátumoch sa kontroluje správny rozsah a formát čísla, pri MAC adresách zase správny formát a podobne. V prípade, že vstupná kontrola nie je validná, je vyhodnená príslušná výnimka a kódovanie je úplne ukončené, dátový záznam sa neexportuje. Je neprípustné, aby Mediátor posielal kolektoru dátové záznamy s prázdnyimi hodnotami z dôvodu, že nebolo možné zakódovať hodnotu danú v zlom formáte.

Pri kódovaní je veľmi dôležitá rýchlosť a pamäťová nenáročnosť kódovacích funkcií. Jedna funkcia, napr. na zakódovanie znamienkového celého čísla na 32 bitoch môže byť zavolaná veľakrát v rámci kódovania jediného dátového záznamu. Tento počet závisí od dátových typov informačných elementov v dátovom zázname. Nemusíme zdôrazňovať aké množstvo IPFIX paketov a dátových záznamov môže Mediátor v čase prijímať. Preto bol kladený veľký dôraz na to, aby boli kódovacie funkcie čo najoptimálnejšie.

Ukážme si to na príklade. Úlohou je zakódovať znamienkové celé číslo  $-42$  na pole štyroch bajtov. Najjednoduchším a najpohodlnejším riešením je použiť triedu *ByteBuffer*, alokovať pole veľkosti 4 bajty, nastaviť poradie bajtov na *Big-Endian*, vložiť číslo  $-42$  a konvertovať na pole volaním `array()`. Druhou možnosťou je použiť triedu *BigInteger*, vložiť hodnotu  $-42$ , a konvertovať na pole bajtov. V tejto metóde sa nedá explicitne nastavovať poradie bajtov, pole je stále zoradené v *Big-Endian*, čo nám vyhovuje. Nepříjemnosťou je dodatočné orezanie na potrebný počet bajtov.

```
ByteBuffer buf = ByteBuffer.allocate(4).order(ByteOrder.BIG_ENDIAN);  
byte[] b1 = buf.putInt(-42).array();
```

```
byte[] b2 = BigInteger.valueOf(-42).toByteArray();
```

Obe tieto metódy zbytočne pridávajú réžiu, zložitosť a zvyšujú pamäťovú náročnosť

výpočtu tým, že používajú komplexné Java triedy. Preto boli pre všetky konverzie implementované metódy pomocou bitových posunov a bitových operátorov. Tieto metódy prevádzajú všetky číselné primitívne typy jazyka Java (byte, short, int, long, float a double) na pole bajtov na najnižšej možnej úrovni, čo zabezpečuje vysoký výpočtový výkon. Ukážme si túto konverziu na 4-bajtovom čísle  $-42$ .

```
public static byte[] intToByteArray(int x) {  
    return new byte[]{  
        (byte) ((x >> 24) & 0xFF),  
        (byte) ((x >> 16) & 0xFF),  
        (byte) ((x >> 8) & 0xFF),  
        (byte) (x & 0xFF)  
    };  
}
```

```
byte[] b3 = intToByteArray(-42);
```

Hodnota zakódovaná na pole bajtov sa uloží do pomocného poľa spomínaného vyššie, na rovnakú pozíciu ako je pozícia špecifikátor poľa v šablóne. Keď sa prejdú všetky hodnoty pripravené na zakódovanie, pomocné pole je vyskladané v správnom poradí a teda ho môžeme uložiť do objektu dátového záznamu. Hotový dátový záznam je vrátený sprostredkovateľského procesu, ktorý si ho vyžiadal.

**4.4.2.5 Distribúcia dát medzi modulmi** Trieda `AIntermediateProcess` poskytuje rozhranie pre distribúciu záznamov o tokoch medzi jednotlivými sprostredkovateľskými procesmi vďaka metóde `dispatchFlowRecord()`. Jej jedinou úlohou je zavolať rovnomennú metódu distribútora záznamov - triedy `FlowRecordDispatcher`. O tom podrobne v samostatnej Kapitole 4 (Sekcia 4.5).

#### 4.4.3 Príklad implementácie modulu – `ExampleProcess`

Pre budúcich riešiteľov bol propražený jednoduchý príklad implementácie sprostredkovateľského procesu. Predstavuje ho trieda `ExampleProcess`, ktorej úlohou je veľmi jednoduchá anonymizácia zdrojovej a cieľovej IP adresy zmenením čísla posledného oktetu na nulu.

Trieda demonštruje všetky pravidlá programovania sprostredkovateľských procesov. V prvom rade dedí od abstraktnej triedy `AIntermediateProcess`. Taktiež má konštruktor bez explicitne definovaného prístupu, v ktorom volá rodičovský konštruktor so svojím menom ako parametrom. Toto síce nie je povinné, ale zaistí to, že vlákno bude pomenované, teda v príslušných logovacích výpisoch bude jeho meno. V opačnom prípade dostane vlákno automaticky vygenerované meno „Thread-*n*“, kde *n* je celé číslo. Poslednou podmienkou je implementovať hlavnú, štartovaciu metódu vlákien – `run()`.

Trieda `ExampleProcess` zároveň predvádza použitie metód, ktoré poskytuje jej rodičovská trieda. V cykle čaká na záznamy o tokoch vo svojom vstupnom bufferi (*inputBuffer*) a postupne ich odtiaľ číta a odstraňuje. Nazvime ich *vstupné záznamy*. Vstupnú vyrovnávaciu pamäť jej naplňa trieda `FlowRecordDispatcher`. Po prečítaní vstupného záznamu vytvorí a inicializuje *výstupný záznam*. Následne prechádza všetky dátové záznamy vstupného záznamu, dekóduje ich, anonymizuje zdrojovú a cieľovú IP adresu a naspať zakóduje. Ak všetko prebehlo bez problémov, tak dátový záznam priradí výstupnému záznamu. Napokon výstupný záznam o toku posunie distribútorovi záznamov, ktorý ho buď prepošle nasledujúcemu sprostredkovateľskému procesu, alebo pripraví na export.

#### 4.4.4 Dynamické načítavanie sprostredkovateľských procesov

Bola definovaná požiadavka, že sprostredkovateľské procesy musia byť načítavané dynamicky, bez nutnosti zásahu do zdrojového kódu aplikačného rámca.

Administrátori definujú zoznam procesov v XML konfiguračnom súbore, v elemente `<processes>`. Táto položka sa skladá z ďalších položiek typu `<process>` s atribútom obsahujúcim meno sprostredkovateľského procesu. Každý proces, ktorý má byť načítaný, musí mať samostatnú položku. Príklad takejto konfigurácie uvidíme v nasledujúcej Sekcii 4.5.

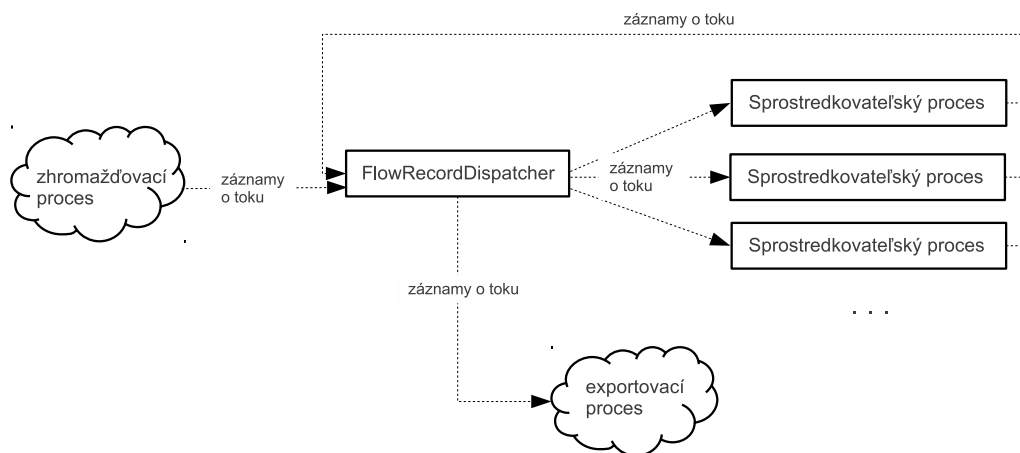
Parser konfiguračného súboru spracuje tieto položky a vytvorí zoznam modulov, ktoré sa majú načítať. Dynamické načítavanie tried podľa ich mena bolo analyzované v Kapitole 4 (Sekcia 4.4.1). Načítavanie modulov má na starosti trieda `IPLoader`. Jej hlavná metóda `loadProcesses()` cyklicky prechádza zoznam reťazcov obsahujúcich názvy modulov. Meno každého modulu prevedie na binárny názov a vďaka *systemovému class loader-u*, získa jeho *class* objekt. Podmienkou však je, že hlavná trieda modulu musí byť v rovnakom balíčku ako trieda `AIntermediateProcess`. Tá potom pomocou metódy `getInstance()`, podrobne rozpísanej vyššie, získa jedinečnú inštanciu sprostredkovateľského procesu. Keďže každý proces je samostatným vláknom, teda dedí od triedy `Thread`, už ho len ostáva spustiť pomocou metódy `start()`. Toto zabezpečí reflexia, ktorá získa metódu a následne ju vyvolá (*invoke*).

```
Object instance = AIntermediateProcess.getInstance(clazz);  
Method start = clazz.getMethod("start");  
start.invoke(instance);
```

Ak pri načítavaní modulov nenastane žiadna chyba, pokračuje sa v ďalšom vykonávaní programu. V opačnom prípade, hoci ak len jeden proces nebol úspešne načítaný, je program Mediátor ukončený.

## 4.5 Trieda `FlowRecordDispatcher`

Úlohou tejto triedy je riadiť tok dát medzi komponentami IPFIX Mediátora na základe nastavenia v konfiguračnom súbore. Ukážme si príklad takejto konfigurácie:



**Obr. 4–3** Schéma toku dát cez triedu FlowRecordDispatcher

```

<processes>
  <process name="SelectionProcess">
    <input>exporter</input>
  </process>
  <process name="AggregationProcess">
    <input>exporter</input>
  </process>
  <process name="AnonymizationProcess">
    <input>AggregationProcess</input>
  </process>
</processes>

```

Majme 3 sprostredkovateľské procesy:

- SelectionProcess,
- AggregationProcess a
- AnonymizationProcess.

Formát zápisu toku dát medzi procesmi je rovnaký ako v príklade. Vstupnými údajmi pre *SelectionProcess* a *AggregationProcess* sú dáta priamo prijaté od exportéra, teda

sú vlastne výstupom zhromažďovacieho procesu. Pričom vstupnými údajmi pre *AnonymizationProcess* je výstup z *AggregationProcess*. Tie procesy, ktorých dáta nevstupujú do žiadneho iného sprostredkovateľského procesu sú logicky „koncovými“ procesmi, ich výstup je posunutý exportovaciemu procesu a poslaný kolektoru. Kým pri *SelectionProcess* a *AggregationProcess* hovoríme o paralelnom spracovaní, *AnonymizationProcess* spracováva záznamy sériovo.

Úloha triedy *FlowRecordDispatcher* začína keď prijme prvé dáta od zhromažďovacieho procesu. Dáta prijíma cez nasledujúce dva parametre: záznam o toku – *IPFIXFlowRecord* a reťazec určujúci odkiaľ tento záznam vystupuje – *inputProcess*. Následne získa zoznam prijímateľov tohto záznamu o toku, teda tie sprostredkovateľské procesy, ktorých položkou *<input>* v konfiguračnom súbore je reťazec *inputProcess*, v tomto prípade „exporter“. Pri konfigurácii ako je daná v príklade by zoznam obsahoval dva reťazce: „SelectionProcess“ a „AggregationProcess“. Teraz potrebujeme získať inštancie týchto tried a ich vstupným vyrovnávacím pamätiam poslať prijatý záznam o toku. *FlowRecordDispatcher* to vykoná v cykle. Metódou *getInstance(String processName)* zavolanou nad abstraktnou triedou *AIntermediateProcess* získa jedinú inštanciu sprostredkovateľského procesu. Tato metóda nie je triviálna, podrobne sme sa jej venovali v Sekcii 4.4.2.2 – *Jediná inštancia modulov*. Každý takto získanej inštancii sprostredkovateľského procesu zapíše do vstupnej pamäte – *inputBuffer* záznam o toku.

Vstupnú vyrovnávaciu pamäť implementuje trieda *IPInputBuffer*, ktorá je analógiou k triede *PacketCache*. Samotnú pamäť rovnako predstavuje synchronizovaný a vysoko výkonný *FIFO* front, implementovaný pomocou *ArrayBlockingQueue*, do ktorého sú zapisované objekty typu *IPFIXFlowRecord*.

Sú tri základne metódy s rôznymi parametrami, ktoré zapisujú do frontu. Metóda *offer()* vloží objekt na koniec frontu, iba v prípade, že je to možné okamžite bez prevýšenia kapacity frontu a vráti *true* ak bol zápis úspešný, *false* v opačnom prípade. Tato metóda sa viac preferuje ako *add()*, ktorá pri neúspešnom zápise vyhodí



výnimku. Treťou je metóda *put()*. Jej špecialitou je to, že v prípade, že je front plný, čaká na uvoľnenie miesta. Toto však spôsobí zablokovanie celého vlákna (Oracle, 2011).

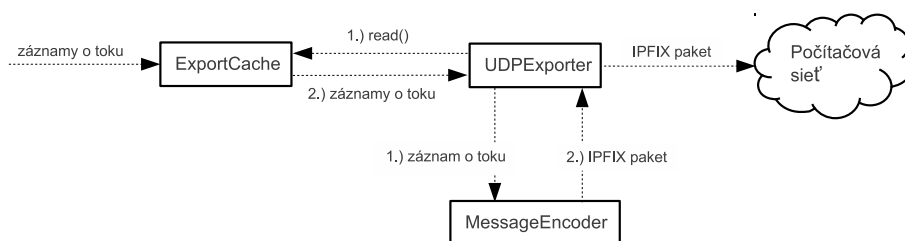
V zhromažďovacom procese, v Kapitole 4 (Sekcia 4.3), sa do frontu zapisuje metódou *put()*. V prípade, že sa *PacketCache* naplní, je na mieste zablokovať *UDPServer* a počkať a uvoľnenie miesta vo vyrovnávacej pamäti. Avšak na tomto mieste by to nebolo vhodné. Stačilo by, že by jeden sprostredkovateľský proces nestíhal spracovávať svoje záznamy o tokoch, zablokovalo by to dispečera tokov a tým cele vlákno predstavujúce druhú fázu zhromažďovacieho procesu. Kvôli jednému procesu by žiaden proces nedostával nové záznamy. Preto zápis do vstupnej pamäte sprostredkovateľských procesov zabezpečuje metóda *offer()*.

Ak metóda na získanie zoznamu prijímateľov vráti prázdny zoznam, vyplýva, že záznam o toku sa nemá presmerovať ďalšiemu sprostredkovateľskému procesu, ale je už určený na export. Preto záznam o toku je zapísaný do vyrovnávacej pamäte pre export. Výsledkom je, že dáta boli presmerované správnym prijímateľom a boli splnené príslušné požiadavky na rámec pre IPFIX Mediátor.

## 4.6 Exportovací proces

Poslednou fázou Mediátora je exportovací proces. Jeho schéma je zobrazená na Obrázku 4–4. Ako bolo spomenuté v predchádzajúcej kapitole, záznamy o tokoch, ktoré sú výstupom „koncových“ sprostredkovateľských procesov sú prostredníctvom dispečera tokov pripravené na export zapísaním do exportovacej pamäte.

Exportovaciú pamäť predstavuje *ExportCache*. Tá je rovnako ako *PacketCache* a *IPInputBuffer* synchronizovaný *FIFO* front, do ktorého sa zapisujú objekty typu *IPFIXFlowRecord*. Ak sa *ExportCache* naplní, sprostredkovateľské procesy nemajú byť blokové, ale majú pokračovať vo svojej práci. Preto sa do cache zapisuje metódou *offer()*.



Obr. 4–4 Schéma exportovacieho procesu

Jadrom exportovacieho procesu je trieda **UDPExporter**, predstavujúca samostatné vlákno. V konštruktori vytvára UDP socket na prijímanie a posielanie paketov a naviaže ho na akýkoľvek voľný port. K tomu slúži volanie bezparametrického konštruktora triedy **DatagramSocket**. V jeho hlavnej metóde *run()* vykonáva cyklus dokiaľ nie je prerušený. V cykle číta a vyberá záznamy o tokoch z vyrovnávacej pamäte pre export. Záznamy posielajú triede **MessageEncoder**, ktorá z neho poskladá IPFIX paket.

**MessageEncoder** vo svojich metódach postupne tvorí obsah IPFIX správy podľa formátu, ktorý bol analyzovaný v Kapitole 2 (Sekcia 2.1.2). Najprv vypočíta sekvenčné číslo, ktoré je obsahom hlavičky každej správy. Toto číslo zodpovedá celkovému počtu doteraz odoslaných dátových záznamov modulo  $2^{32}$ . Ich celkový počet si priebežne zvyšuje vo svojej členskej premennej. Potom postupne kóduje sady šablón, dátové sady, určí, resp. vypočíta všetky položky hlavičky správy a napokon všetko pospája do výslednej správy.

Trieda rozhoduje, či v posielanej IPFIX správe má byť zahrnutá aj šablóna príslušajúca dátovým záznamom obsiahnutým v zázname o toku z ktorého vytvára správu. Každá šablóna vo svojej členskej premennej uchováva čas, kedy bola posledne exportovaná. Ak rozdiel medzi prítomnosťou a časom posledného exportu je väčší ako je hodnota `<ipfixTemplateTimeout>` definovaná v konfiguračnom súbore, tak šablóna musí byť pripojená. Rovnako je šablóna pripojená okamžite po jej aktualizácii exportérom.

Metóda na zakódovanie šablóny používa objekt triedy `ByteArrayOutputStream`. Táto trieda implementuje prúd údajov, v ktorom sú dáta zapisované do poľa bajtov. Do tohto prúdu postupne kóduje údaje v takom poradí ako definuje formát IPFIX správy. Najprv do prúdu zakóduje číslo šablóny, za ním počet špecifikátorov poľa a potom samotné špecifikátory. Formát špecifikátorov je nasledovný. Ako prvé sa kóduje číslo informačného elementu na dvoch bajtoch. V prípade, že ide o element definovaný organizáciou, tak najvyšší bit sa nastaví na 1. Nasleduje dĺžka informačného elementu a v prípade potreby číslo organizácie definované spoločnosťou IANA. Keď je prúd záznamu šablóny hotový, tak sa pred neho zaradia zakódované údaje sady šablóny. Tie pozostávajú z čísla sady, čo je v prípade sady šablóny číslo 2 a celkovej dĺžky záznamu šablóny vrátane veľkosti hlavičky sady. Touto operáciou vznikne prúd bajtov sady šablóny.

Po zakódovaní šablóny sa postupne kódujú všetky dátové záznamy obsiahnuté v zázname toku. Jednotlivé hodnoty polí už sú správne zakódované podľa dátového typu, toto majú na zodpovednosti sprostredkovateľské procesy. Takže v tejto fáze stačí cyklicky prejsť všetky dátové záznamy a ich zakódované polia zapísať do prúdu bajtov. Napokon je potrebné pred tento prúd zaradiť zakódované údaje dátovej sady. Konkrétne ide o číslo prislúchajúcej šablóny a súčet dĺžok všetkých dátových záznamov vrátane veľkosti hlavičky sady. Takto je vytvorený prúd bajtov dátovej sady.

Ako posledný sa zostaví prúd bajtov hlavičky IPFIX správy. Dĺžka správy je určená súčtom veľkosti hlavičky správy s veľkosťou sady šablóny a dátovej sady. Ako prvé sa do prúdu bajtov hlavičky zakóduje číslo verzie, teda `0x000a` a dĺžka správy. Nasleduje čas exportu, vypočítané sekvenčné číslo a ID pozorovacej domény, ktoré je nastavené administrátorom v konfiguračnom súbore. Toto číslo však definuje pozorováciu domény v ktorej sídli Mediátor, nie exportér. O určovaní času exportu podrobnejšie neskôr.

V analýze boli uvedené implementačno-špecifické problémy Mediátora, Kapitola 2

(Sekcia 2.2.4). Prvým bola strata informácii o pôvodnom exportéri. Ako bolo spomenuté neskôr, v analýze exportovacieho procesu v Kapitole 2 (Sekcia 2.3.2.2), spôsobom ako posielat tieto informácie je zakódovať ich do informačných elementov skupiny 2. Na požiadavku Mediátora boli všetky informačné elementy z Tabuľky 2–3 implementované na strane exportéra. ID pozorovacieho bodu a pozorovacej domény v ktorej sídli exportér sa kolektor dozvie z príslušných informačných elementov, ktoré taktiež exportér podporuje.

Druhým problémom bola strata informácie o čase exportu. RFC 6183 (Kobayashi et al., 2011) popisuje dva spôsoby určenia času exportu:

1. Zachovávať hodnotu z hlavičky prichádzajúcich IPFIX správ.
2. Nastaviť aktuálnu hodnotu času keď IPFIX správa opúšťa Mediátor.

V prípade, že exportér posielal akýkoľvek „delta“ informačný element, napr. *flowStartDeltaMicroseconds*, tak musí byť použitý prvý spôsob určenia času exportu, teda zachovanie pôvodného času. Keďže „delta“ hodnoty sa viažu na tento čas, pri jeho prepísaní by stratili význam. Aby Mediátor vyhovoval obom prípadom, navrhli sme, že spôsob určovania času exportu definuje administrátor v konfiguračnom súbore, v elemente `<exportTime>`. Ak zadá reťazec `KEEP`, tak trieda `MessageEncoder` zakóduje do prúdu bajtov hlavičky pôvodnú hodnotu. V prípade reťazca `RENEW` sa zakóduje aktuálna hodnota času.

Poslednou úlohou triedy `MessageEncoder` je pospájať jednotlivé prúdy bajtov do výsledného prúdu IPFIX správy. Prvým je prúd bajtov hlavičky správy. Ak sa exportuje aj šablóna, tak nasleduje prúd sady šablóny a posledným je prúd dátovej sady. Trieda `UDPExporter` teraz z prúdu IPFIX správy vytvorí UDP paket. Na to slúži trieda `DatagramPacket`, pričom jej parametrami sú dáta, dĺžka správy, IP adresa a UDP port. Posledné dve menované sú zadané administrátorom v konfiguračnom súbore. Metódou `send()` zavolanou nad socketom je správa odoslaná.

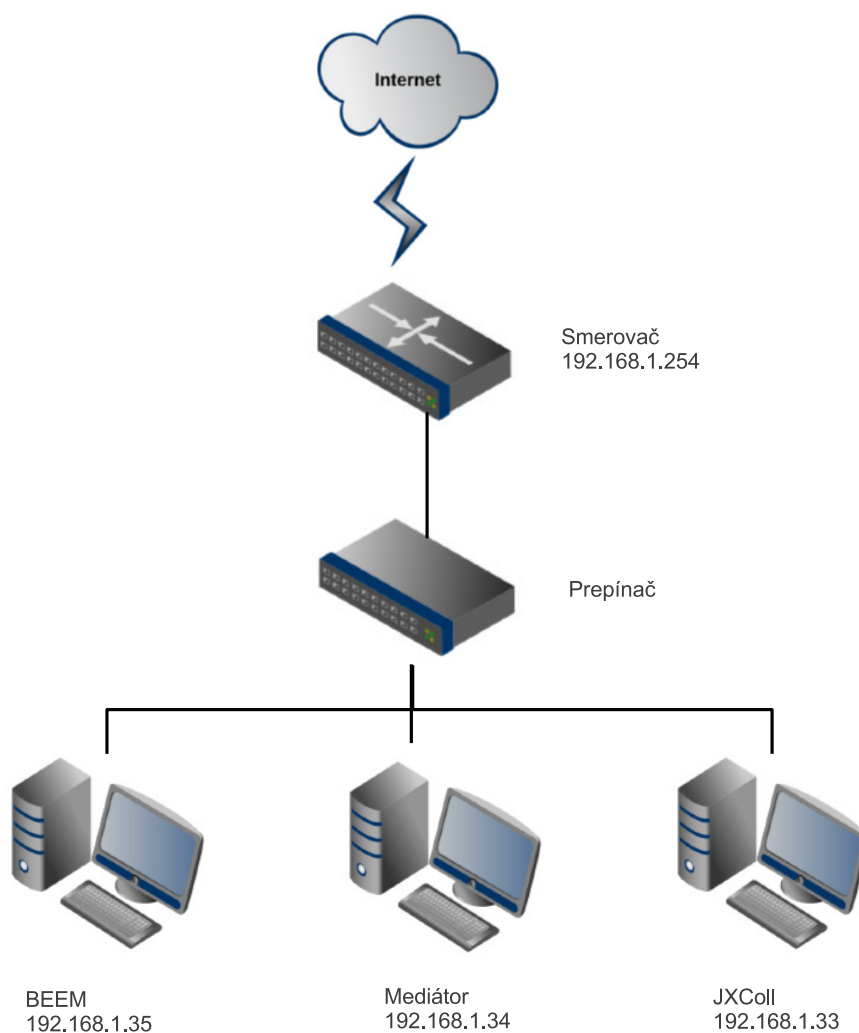
## 5 Experimentálne overenie funkčnosti riešenia

Táto kapitola je venovaná overeniu správnosti implementácie. Boli vykonané nasledujúce štyri experimenty, ktoré mali potvrdiť, alebo vyvrátiť správnu funkčnosť riešenia.

1. **Prvý test** overil konektivitu a prenos údajov medzi exportérom a mediátorom a následne medzi mediátorom a kolektorom.
2. **Druhý test** porovnal hodnoty údajov exportovaných BEEM-om s hodnotami uloženými v databáze po prechode cez Mediátor bez sprostredkovateľského procesu.
3. **Tretí test** demonštroval správnosť anonymizácie údajov v podaní sprostredkovateľského procesu `ExampleProcess`, spomínaného vyššie v návrhu a implementácii.
4. **Štvrtý test** bol tzv. záťažovým testom, overoval stabilitu Mediátora pri dlhodobom behu a pri spracovávaní prevádzky s vysokým počtom tokov.

### 5.1 Testovacia topológia

Pri prvých troch testoch bol použitý virtualizačný nástroj VirtualBox, v ktorom boli vytvorené tri virtuálne počítače, zapojené do topológie, ktorú možno vidieť na Obrázku 5 – 1. Na všetkých troch počítačoch sa používal operačný systém *Ubuntu 12.04 LTS* v desktopovej verzii. Na prvom počítači bol nainštalovaný exportér BEEM (verzia 1.1-6 s podporou pre IPFIX Mediátor), ktorý exportoval správy druhému počítaču na UDP port vyhradený pre IPFIX komunikáciu - 4739. Tam bežal Mediátor verzie 1.0. Ten zase preposielal správy na tretí počítač (taktiež port 4739), kde bol spustený JXColl (verzia 3.9) s funkčnou PostgreSQL databázou. Počítače boli zapojené v jednej lokálnej sieti a všetky mali prístup na Internet.



Obr. 5 – 1 Testovacia topológia

## 5.2 Test konektivity

Prvý experiment overoval základnú konektivitu medzi exportérom a kolektorom v prípade, že je medzi ne zaradený mediátor. Vo svojej východiskovej konfigurácii vypisuje BEEM logovacie správy na štandardný výstup. Tie obsahujú základné informácie o každom exportovanom zázname o toku ako to môžeme vidieť na Obrázku 5–2. Mediátor mal nastavenú úroveň logovacích výstupov na *DEBUG*, čo zahŕňa nielen chybové hlášky, ale aj všetky informačné a ladiace oznamy.

Konektivita medzi exportérom a mediátorom bola potvrdená, na Obrázku 5–3 je zachytený výstup Mediátora po prijatí správy od exportéra. Výpis obsahuje IP adresu a port z ktorého prišla správa, jej veľkosť a oznam, že ide o IPFIX správu. Za tým nasledujú hodnoty z hlavičky správy a výpis všetkých sád, ktoré správa obsahuje. Potom bol spracovaný záznam o toku posunutý dispečerovi – `FlowRecordDispatcher`, ktorý ho poslal na export. Tam sa záznam o toku naspäť zabalil do nového IPFIX paketu a ten bol odoslaný smerom na kolektor.

```
=====
root Beem INFORMATION Expiration reason value -> 2
root Beem INFORMATION Type of used protocol -> [TCP]
root Beem INFORMATION Source IP address -> 192.168.1.35
root Beem INFORMATION Destination IP address -> 67.214.223.103
root Beem INFORMATION Speed -> 0 kbps
root Beem INFORMATION Amount of DATA sent -> 7095:0
root Beem INFORMATION Flow_ID value: 5668716075047983316
=====
```

Obr. 5–2 Test 1: Konektivita medzi exportérom a Mediátorom – strana exportéra

```
UDPPProcessor - Packet read from /192.168.1.35:56265 (104 bytes)
UDPPProcessor - Length of packet: 104
UDPPProcessor - It's IPFIX packet ...
IPFIXParser - ***** IPFIX MESSAGE HEADER *****
IPFIXParser - Version number: 10
IPFIXParser - Message Length: 104 bytes
IPFIXParser - Export Time: 1365898787
IPFIXParser - Export time human readable: 2013-04-14 01:19:47+0100
IPFIXParser - Sequence number: 26
IPFIXParser - Observation domain ID: 0
IPFIXParser - ***** ***** ***** ***** *****
IPFIXParser - ***** IPFIX SET HEADER *****
IPFIXParser - Set ID:257(DATA SET)
IPFIXParser - Set Length: 88 bytes
IPFIXParser - ***** ***** ***** ***** *****
IPFIXParser - ***** DATA Record *****
IPFIXParser - data record length = 84
IPFIXParser - padding size = 0
IPFIXParser - >>>Sending flowRecord, sourceName to RecordDispatcher<<<
FlowRecordDispatcher - Sending data for export!
ExportCache - ExportCache queue remaning capacity: 100
DPExporter - SENDING PACKET
```

Obr. 5–3 Test 1: Konektivita medzi exportérom a Mediátorom – strana Mediátora

Konektivita medzi Mediátorom a kolektorom bola tiež potvrdená. Dôkazom boli logovacie výstupy kolektora, ktoré sú veľmi podobné ako tie z Mediátora. V databáze,

na ktorú je JXColl pripojený, pribúdali správne záznamy. Tento experiment môžeme považovať za úspešný.

### 5.3 Test správnej reprezentácie údajov

Druhý experiment nadväzoval na predchádzajúci test konektivity. Testovacia topológia aj konfigurácie jednotlivých nástrojov boli rovnaké ako v predchádzajúcom prípade. V Mediátore stále neboli spustené žiadne sprostredkovateľské moduly.

Myšlienkou tohto overenia bolo porovnať dáta, ktoré posielal exportér s dátami, ktoré prejdú celou topológiou cez Mediátor a JXColl a sú uložené v databáze. Na zachytenie správ z BEEM-u bol použitý program Wireshark, ktorý odchytil sieťovú prevádzku na prvom počítači, na rozhraní *eth0* s IP adresou *192.168.1.35*. Detail jednej zo správ, ktoré zachytil Wireshark je na Obrázku 5–5. Na Obrázku 5–4 sú zase vyfiltrované posledné záznamy, ktoré boli uložené do databázy. Môžeme porovnať aspoň na niekoľkých znázornených informačných elementoch, že záznamy o toku sa po prechode cez topológiu s Mediátorom bez zapnutých sprostredkovateľských procesov nezmenili. Tento experiment považujeme za úspešný.

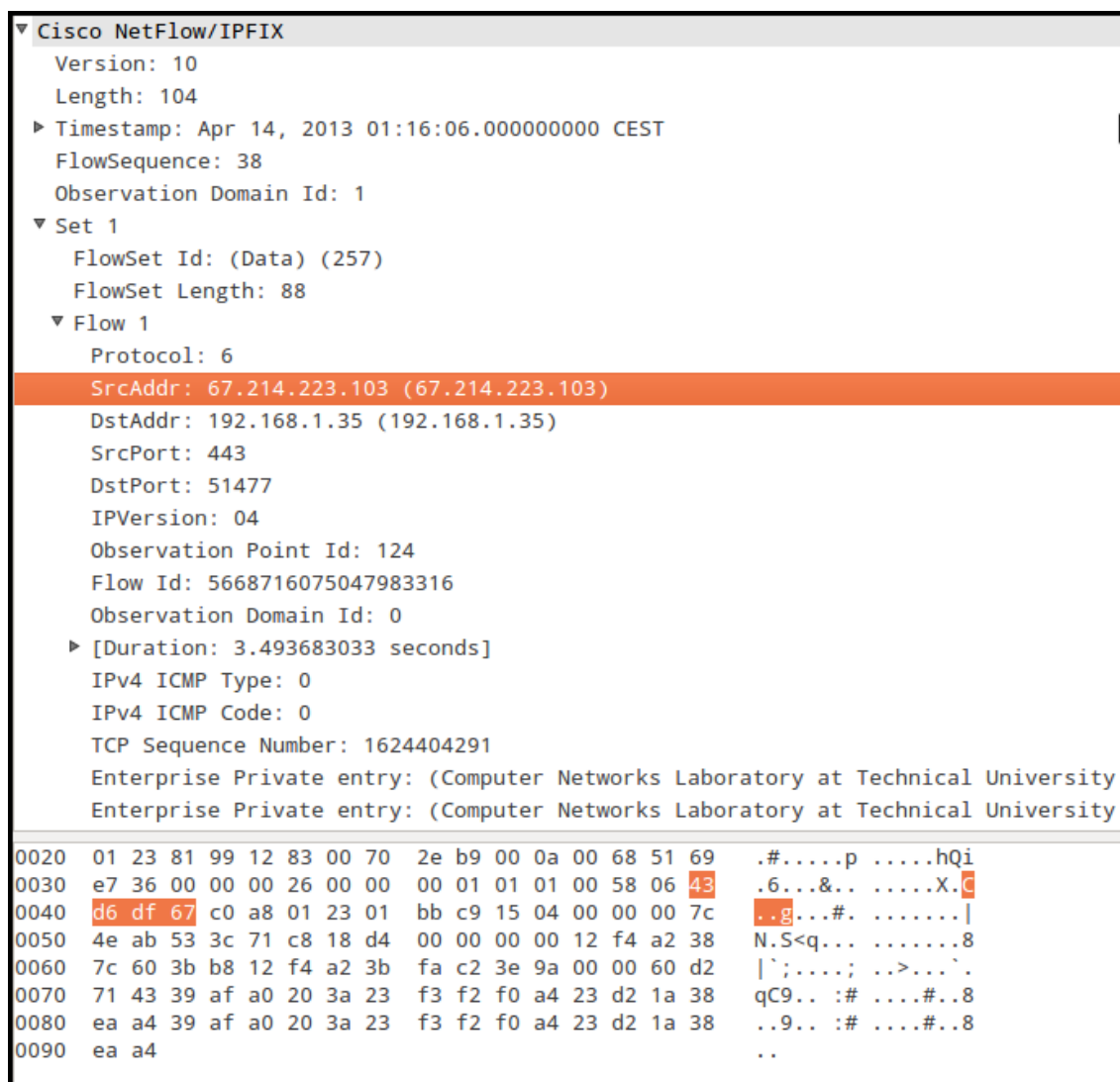
rid	sourceipv4address	destinationipv4address	sourcetransportport	destinationtransportport	flowid
72297	192.168.1.34	192.168.1.35	33177	4739	13954769630869584656
72296	192.168.1.35	192.168.1.34	33916	4739	5801356863763569128
72295	67.214.223.103	192.168.1.35	443	51477	5668716075047983316
72294	192.168.1.35	67.214.223.103	51477	443	1102623916668138460
72293	192.168.1.35	173.194.39.117	47805	443	2076010102190056414
72292	173.194.39.117	192.168.1.35	443	47805	10142187004514802383
72291	192.168.1.34	192.168.1.35	33177	4739	13954769630869584656
72290	192.168.1.35	192.168.1.34	33916	4739	5801356863763569128

Obr. 5–4 Test 2: Výpis obsahu databázy

### 5.4 Test Mediátora s anonymizačným modulom

Tento experiment je miernou úpravou predchádzajúceho. Testovacia topológia aj konfigurácie nástrojov boli rovnaké s tým rozdielom, že v Mediátore bol zapnutý





Obr. 5 – 5 Test 2: Správy odosielané exportérom zachytené programom Wireshark

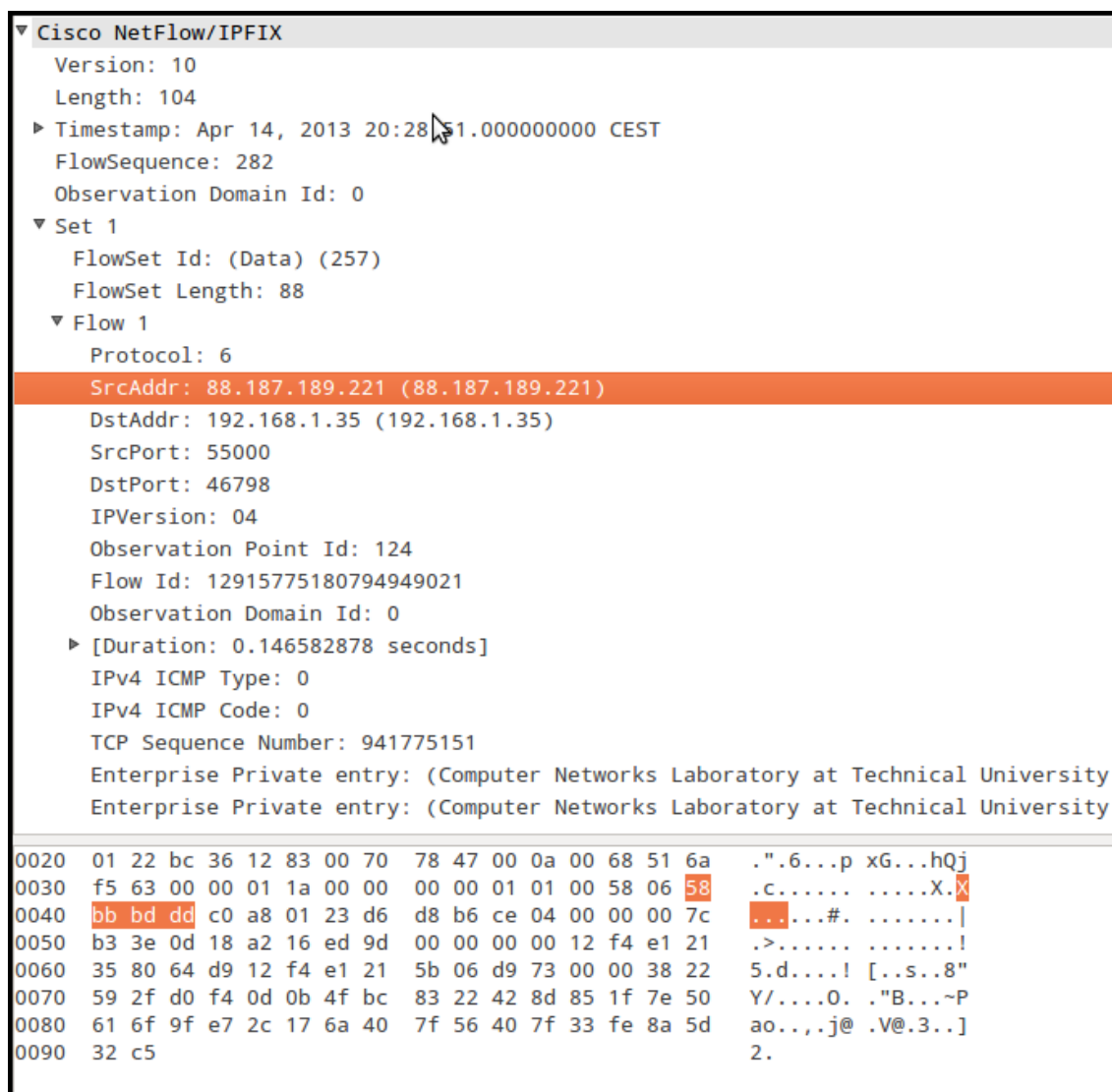
jednoduchý anonymizačný proces - `ExampleProcess`. Tento modul bol naprogramovaný kvôli demonštrácii implementácie sprostredkovateľského procesu. Cieľom experimentu bolo dokázať dve tvrdenia:

- Aplikačný rámec poskytuje sprostredkovateľským procesom správne metódy na dekodovanie a zakódovanie dátových záznamov. To platí vtedy, keď hodnoty informačných elementov sa rovnajú hodnotám záznamov v databáze po prechode celou topológiou aj keď sú spustené sprostredkovateľské moduly.

- Anonymizačný modul správne modifikuje zdrojovú a cieľovú IP adresu toku.

Výsledkom je to, že posledný oktet je v oboch prípadoch nahradený nulou.

Na Obrázku 5–6 je zobrazený detail jednej zo správ, ktorú exportoval BEEM. Na nasledujúcom Obrázku 5–7 vidieť posledné záznamy z databázy. Môžeme porovnať, že hodnoty zdrojového a cieľového transportného portu a hodnota *flowId* sa nezmenili. Avšak zároveň, všetky zdrojové a cieľové IP adresy boli anonymizované. Tento experiment považujeme za úspešný, dokázal správnosť oboch tvrdení.



Obr. 5–6 Test 3: Správy odosielané exportérom zachytené programom Wireshark

rid	sourceip4address	destinationip4address	sourcetransportport	destinationtransportport	flowid
73741	90.206.221.0	192.168.1.0	63149	51413	8601468596564156237
73740	192.168.1.0	90.206.221.0	51413	63149	14239456355096976069
73739	88.187.189.0	192.168.1.0	55000	46798	12915775180794949021
73738	192.168.1.0	69.171.248.0	53379	443	16277749152667872155
73737	69.171.248.0	192.168.1.0	443	53379	1888078803143614872
73736	58.109.69.0	192.168.1.0	47133	53155	10741961727492761765
73735	187.143.12.0	192.168.1.0	13167	51413	10464130345034868280
73734	192.168.1.0	90.41.178.0	48322	14857	17966693341759876638

Obr. 5 – 7 Test 3: Výpis obsahu databázy

## 5.5 Závažový test

Posledný experiment sa nazýva aj „*stress test*“. Jeho úlohou je intenzívne a náročné testovanie za účelom overenia stability systému. Pre tento experiment nebola použitá topológia ako v ostatných testoch. Závažový test prebiehal na virtuálnom serveri *monica-med.monica.cnl.sk* v infraštruktúre Laboratória počítačových sietí na Technickej univerzite v Košiciach.

Na serveri bol nainštalovaný Mediátor 1.0 z inštalačného *.deb* balíčka a BEEM poslednej verzie, ktorá má podporu pre Mediátor. Myšlienkou experimentu bolo zistiť, ako sa bude správať Mediátor pri dlhodobom behu a pri vysokom zaťažení v podobe veľkého množstva súbežných tokov. Protokol *BitTorrent* (BitTorrent.org, 2009) je známy práve tým, že nadväzuje veľa spojení. Preto sa o generovanie sieťovej prevádzky postarala konzolová verzia BitTorrent klienta *Transmission*, ktorá stahovala rôzne distribúcie OS Ubuntu.

Experiment odhalil úzke hrdlo nielen Mediátora, ale aj exportéra BEEM. Keď sa spojili dva faktory: enormný počet dátových tokov a rýchlosť prenosu medzi 20 MB/s až 40 MB/s, tak sa niekedy stávalo, že vstupná pamäť sprostredkovateľského procesu *ExampleProcess* sa preplnila a začala zahadzovať pakety. Analýza vyťaženia vlákien vo vývojárskom prostredí *NetBeans 7.3* ukázala, že príčina zahadzovania paketov nie je spôsobená tým, že *ExampleProcess* nestíha v dostatočnej rýchlosti spracovávať pakety. Proces bol až 95% času v stave *wait*, teda čakal na záznamy o toku. Avšak BEEM exportoval správy v takých zhlukoch, ktoré nárazovo zaplnili celú vstupnú pamäť procesu. Stávalo sa, že do vstupnej pamäte bol zapísaný zhluk

30 záznamov behom 0,003 sekundy! Zvýšenie veľkosti vstupného buffera z 25 na 100 záznamov odstránilo tento nedostatok. Stálo by za úvahu, či by nebolo vhodné triedu zodpovednú za delegovanie toku zaznamov - `FlowRecordDispatcher` implementovať ako samostatné vlákno. Dispečerovi by bola predradená dostatočne veľká vyrovnávacia pamäť typu `ArrayBlockingQueue`, použitá aj na iných miestach architektúry, ktorá by mohla zmierniť nárazový nápor záznamov. Zhromažďovací proces a moduly by tak nevolali priamo synchronizovanú metódu dispečera ako je tomu teraz, ale svoje záznamy by zapisovali do tejto pamäte.

Druhý nedostatok bol odhalený v exportéri. Nástroj BEEM rovnako hlásil preplnenie pamäte pre pakety a preto zahadzoval správy. Tento fakt bol oznámený riešiteľskému tímu zodpovednému za exportér.

Pozitívnym výsledkom experimentu bolo to, že dlhodobé spustenie Mediátora neprinieslo žiadnu nestabilitu systému. Spotreba pamäte nemala stúpajúcu tendenciu, bola ustálená na hodnote 32 MB počas celých 10 dní trvania experimentu. Využitie procesora nepresiahlo 6%. Na základe týchto zistení môžeme záťažový experiment považovať za čiastočne úspešný.

Uvedomujeme si, že časový horizont desiatich dní ťažko považovať za dlhodobé testovanie programu. Tu sa skôr hovorí o mesiacoch, alebo rokoch. No z pochopiteľných dôvodov nebolo možné vykonať takéto dlhé testovanie.

## 6 Zhodnotenie riešenia

Najdôležitejším cieľom tejto diplomovej práce bolo navrhnúť a implementovať aplikačný rámec pre sprostredkovateľskú entitu (IPFIX Mediátor), ktorá by bola medzičlánkom v komunikácii medzi exportérom BEEM a kolektorom JXColl nástroja SLAmeter. Mediátor má poskytovať rozhranie pre manipuláciu so sprostredkovateľskými modulmi, ktoré budú poskytovať rôzne funkcie na modifikáciu IPFIX správ ešte pred ich spracovaním v kolektore. Práca túto úlohu aj splnila, čo bolo potvrdené v poslednej časti práce venovanej experimentálnemu overeniu.

Úvodom sa čitateľ oboznámil s analýzou formátu IPFIX správ rovnako ako s výhodami použitia IPFIX Mediátora v meracej architektúre. Samostatné časti boli venované príkladom použitia, ale aj úskaliam spojeným s implementáciou tohto nástroja. Nechýbalo ani stručné predstavenie projektov výskumnej skupiny MONICA.

Najvýznamnejším výsledkom práce je funkčná implementácia aplikačného rámca v jazyku Java. Jeho návrh je odpoveďou na definované požiadavky a analyzované implementačno-špecifické problémy. Zhromažďovací proces bol prispôsobený, no vychádzal z existujúcej implementácie kolektoru JXColl. Dôležitou časťou práce je podpora pre sprostredkovateľské procesy. Tok dát medzi modulmi, či už sériový, alebo paralelný sa jednoducho nastavuje v konfiguračnom XML súbore podľa navrhnutého formátu. Na základe tejto konfigurácie riadi dispečer distribúciu záznamov o tokoch medzi procesmi. Definované moduly sa však najprv musia pospúšťať. Toto zabezpečila pokročilá technológia dynamického načítavania tried pomocou systémového *class loadera* a reflexie. Ďalším pokročilým riešením bolo zabezpečenie toho, že všetky sprostredkovateľské moduly musia mať len jednu unikátnu inštanciu. Keďže dizajn jazyka Java nepovoľuje *abstract static* metódy, bolo potrebné využiť návrhový vzor *Factory method* a hash mapu uchovávajúcu existujúce inštancie modulov.

Úlohou aplikačného rámca je aj poskytnúť metódy na dekodovanie a zakódovanie hodnôt informačných elementov. Pri zakódovaní bol kladený veľký dôraz na efekti-

vitú a výpočtový výkon. Preto navrhnuté metódy namiesto využívania tried jazyka pracujú na úrovni bitových operácií. Budúci riešitelia určite ocenia prítomnosť vzorového sprostredkovateľského modulu, ktorý demonštruje navrhnuté zásady programovania modulov a využíva všetky dostupné metódy, ktoré mu poskytuje abstraktná rodičovská trieda.

Implementácia bola nakoniec overená niekoľkými experimentmi, ktoré dokázali funkčnosť riešenia. Ďalšie smerovanie nástroja predpokladá implementovanie sprostredkovateľských modulov, ktoré zlepšia monitorovacie možnosti aplikácie SLAmeter. Najväčší prínos sa vidí v anonymizačnom module, konverznom module z protokolu NetFlow na IPFIX, prípadne v module, ktorý bude na základe istých pravidiel robiť selekciu IPFIX správ a tie následne exportovať distribuovaným kolektorom. Z pohľadu aplikačného rámca by bolo vhodné rozšíriť jeho zhromažďovací proces o podporu ďalších transportných protokolov ako TCP a SCTP. Rovnaké rozšírenie je možné aj na strane exportovacieho procesu.

## Literatúra

Tom Sheldon's Linktionary: *Socket* [online], 2001. Dostupné z: <<http://www.linktionary.com/s/socket.html>>.

SADASIVAN, G. et al.: *Architecture for IP Flow Information Export* RFC 5470. 2009

Information Sciences Institute, University of Southern California: *Internet protocol* RFC 791. 1981

DEERING, S., HINDEN, R.: *Internet Protocol, Version 6 (IPv6) - Specification* RFC 2460. 1998

CLAISE, B. et al.: *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. RFC 5101. 2008

QUITTEK, J. et al.: *Information Model for IP Flow Information Export* RFC 5102. 2008

QUITTEK, J. et al.: *Requirements for IP Flow Information Export (IPFIX)*. RFC 3917. 2004

KOBAYASHI, A. – CLAISE, B. et al.: *IP Flow Information Export (IPFIX) Mediation: Problem Statement*. RFC 5982. 2010

KOBAYASHI, A. et al.: *IP Flow Information Export (IPFIX) Mediation: Framework*. RFC 6183. 2011

CLAISE, B.: *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. 2004

MILLS, D. L.: *Network Time Protocol (Version 3) Specification, Implementation and Analysis*. RFC 1305. 1992

ZSEBY, T. et al.: *Sampling and Filtering Techniques for IP Packet Selection* RFC 5475. 2009

- 
- IETF: *IP Flow Information Export (ipfix)* [online], 2013. Dostupné z: <<http://www.ietf.org/html.charters/ipfix-charter.html>>.
- Javvin network management & security: *IPFIX: Internet Protocol Flow Information eXport* [online]. Dostupné z: <<http://www.javvin.com/protocolIPFIX.html>>.
- JUVHAUGEN, P.: *Exporting IP flows using IPFIX*. Diplomová práca. Oslo: Oslo University College FI, 2007. 18 s. Dostupné z: <<http://hdl.handle.net/10642/434>>.
- VEREŠČÁK, T.: *Optimalizácia zhromažďovacieho procesu nástroja BasicMeter*. Diplomová práca. Košice: Technická univerzita. Fakulta elektrotechniky a informatiky. Katedra počítačov a informatiky, 2012.
- IANA: *Private Enterprise Numbers* [online], 2013. Dostupné z: <<http://www.iana.org/assignments/enterprise-numbers>>.
- CHO, K., FUKUDA, K., ESAKI, H., KATO, A.: *The Impact and Implications of the Growth in Residential User-to-User Traffic*, SIGCOMM2006, pp. 207-218, Pisa, Taliansko, September 2006.
- IEEE Computer Society: *Link Aggregation*, IEEE Std 802.3ad-2000, March 2000.
- PEKÁR, A.: *Monitorovanie prevádzkových parametrov siete v reálnom čase*. Bakalárska práca. Košice: Technická univerzita. Fakulta elektrotechniky a informatiky. Katedra počítačov a informatiky, 2009.
- Výskumná skupina MONICA: *Nástroj BasicMeter* [online], 2013. Dostupné z: <<http://wiki.cnl.sk/Monica/BasicMeter>>.
- KUDLA, R.: *Experimentálne prostredie pre nástroj BasicMeter*. Bakalárska práca. Košice: Technická univerzita. Fakulta elektrotechniky a informatiky. Katedra počítačov a informatiky, 2010.
-



- Výskumná skupina MONICA: *SLAmeter* [online], 2013. Dostupné z: <<http://wiki.cnl.sk/Monica/SLAmeter>>.
- Výskumná skupina MONICA: *Vyhodnocovač* [online], 2013. Dostupné z: <<http://wiki.cnl.sk/Monica/VyhodnocovacSLA>>.
- MCMANIS, CH.: *The basics of Java class loaders* [online], 1996. Dostupné z: <<http://www.javaworld.com/javaworld/jw-10-1996/jw-10-indepth.html>>.
- TRAVIS, G.: *Understanding the Java ClassLoader* [online], 2001. Dostupné z: <<http://www.ibm.com/developerworks/java/tutorials/j-classloader>>.
- TechJava: *Java Class Loading* [online], 2008. Dostupné z: <<http://www.techjava.de/topics/2008/01/java-class-loading/>>.
- ANTL, M.: *Rámec vyhodnocovača a webového rozhrania nástroja SLA Meter* Diplomová práca. Košice: Technická univerzita. Fakulta elektrotechniky a informatiky. Katedra počítačov a informatiky, 2012.
- CHRISTUDAS, B.: *Internals of Java Class Loading* [online], 2005. Dostupné z: <<http://www.onjava.com/pub/a/onjava/2005/01/26/classloading.html>>.
- Oracle: *Class ClassLoader* [online], Java 6 - oficiálna špecifikácia API, 2011. Dostupné z: <<http://docs.oracle.com/javase/6/docs/api/java/lang/ClassLoader.html>>.
- GALLAGHER, N.: *Inherited Java Singleton Problem* [príspevok na diskusnom fóre], 2010. Dostupné z: <<http://c2.com/cgi/wiki?InheritedJavaSingletonProblem>>.
- Oracle: *Java Language and Virtual Machine Specifications* [online], 2013. Dostupné z: <<http://docs.oracle.com/javase/specs/>>.
- Oracle: *Class ArrayBlockingQueue<E>* [online], Java 6 - oficiálna špecifikácia

API, 2011. Dostupné z: [<http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ArrayBlockingQueue.html#offer\(E\)>](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ArrayBlockingQueue.html#offer(E)).

BitTorrent.org: *The BitTorrent Protocol Specification* [online], 2009. Dostupné z: [<http://www.bittorrent.org/beps/bep\\_0003.html>](http://www.bittorrent.org/beps/bep_0003.html).

# Zoznam príloh

**Príloha A** Systémová príručka Mediator v1.0

**Príloha B** Používateľská príručka Mediator v1.0

**Príloha C** CD médium obsahujúce:

diplomovú prácu v elektronickej podobe (PDF,  $\LaTeX$ )

systémovú príručku Mediator v1.0 (PDF,  $\LaTeX$ )

používateľskú príručku Mediator v1.0 (PDF,  $\LaTeX$ )

zdrojové súbory Mediator v1.0 (Java)

DEB inštalačný balík pre Mediator v1.0 (DEB)