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 Feciľak, PhD. |
| Konzultant: | Ing. Adrián Pekár |

**Košice 2013** **Bc. Rastislav Kudla**

**Abstrakt v SJ**

Abstrakt je povinnou súčasťou každej práce. Je výstižnou charakteristikou obsahu dokumentu. Nevyjadruje hodnotiace stanovisko autora. Má byť taký informatívny, ako to povoľuje podstata práce. Text abstraktu sa píše ako jeden odstavec. Abstrakt neobsahuje odkazy na samotný text práce. Mal by mať rozsah 250 až 500 slov. Pri štylizácii sa používajú celé vety, slovesá v činnom rode a tretej osobe. Používa sa odborná terminológia, menej zvyčajné termíny, skratky a symboly sa pri prvom výskyte v texte definujú.

**Kľúčové slová**

Sprostredkovanie, Mediátor, Kolektor, Exportér, IPFIX, SLAMeter, BasicMeter

**Abstrakt v AJ**

Text abstraktu v svetovom jazyku je potrebný pre integráciu do medzinárodných informačných systémov. Ak nie je možné cudzojazyčnú verziu abstraktu umiestniť na jednej strane so slovenským abstraktom, je potrebné umiestniť ju na samostatnú stranu (cudzojazyčný abstrakt nemožno deliť a uvádzať na dvoch strabách).

**Kľúčové slová v AJ**

Mediation, Mediator, Collector, Exporter, IPFIX, SLAMeter, BasicMeter

# Zadanie práce

Namiesto tejto strany vložte naskenované zadanie úlohy. Odporúčame skenovať s rozlíšením 200 až 300 dpi, čierno-bielo! V jednej vytlačenej ZP musí byť vložený originál zadávacieho listu!

**Čestné vyhlásenie**

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

Košice 26. 4. 2013 .....

*Vlastnoručný podpis*

**Poďakovanie**

Na tomto mieste sa chcem poďakovať vedúcemu diplomovej práce Ing. Petrovi Feciľakovi, 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

Predhovor je povinnou náležitosťou záverečnej práce, pozri (Gonda, 2001). V predhovore autor uvedie základné charakteristiky svojej záverečnej práce a okolnosti jej vzniku. Vysvetlí dôvody, ktoré ho viedli k voľbe témy, cieľ a účel práce a stručne informuje o hlavných metódach, ktoré pri spracovaní záverečnej práce použil.

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Zoznam symbolov a skratiek

a pod. a podobne

ACP Analyzer-Colector Protocol

atd. a tak dalej

BEEM BasicMeter Exporting and Measuring process

BGP Border Gateway Protocol

BMAnalyzer 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

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 XML Collector

LAN Local Area Network

LTS Long Term Support

MAC Media Access Control

MONICA Monitoring and Optimization of Network Infrastructures Communications and Applications

napr. napríklad

PEN Private Enterprise Number

resp. respektívne

SCTP Stream Control Transmission Protocol

SLAmeter Service-level agreement metering

SQL Structured Query Language

TCP Transmission Control Protocol

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 vstupne 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 poli. K hashovacim kľúčom priraďuje 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 Institute, 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 bytov dátových typov v pamäti počítača. Najvýznamnejší byte je uložený na najnižšej (prvej) pozícii. Používa sa pri komunikácii v počítačových sieťach, preto sa tiež nazýva „sieťové poradie bytov“. Jeho presným opakom je Little-Endian.

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

**paket**

**soket**

# Úvod

V úvode autor podrobnejšie ako v predhovore, pritom výstižne a krátko charakterizuje stav poznania alebo praxe v špecifickej oblasti, ktorá je predmetom záverečnej práce. Autor presnejšie ako v predhovore vysvetlí ciele práce, jej zameranie, použité metódy a stručne objasní vzťah práce k iným prácam podobného zamerania. V úvode netreba zachádzať hlbšie do teórie. Nie je potrebné podrobne popisovať metódy, experimentálne výsledky, ani opakovať závery prípadne odporúčania, pozri (Katuščák, 1998).

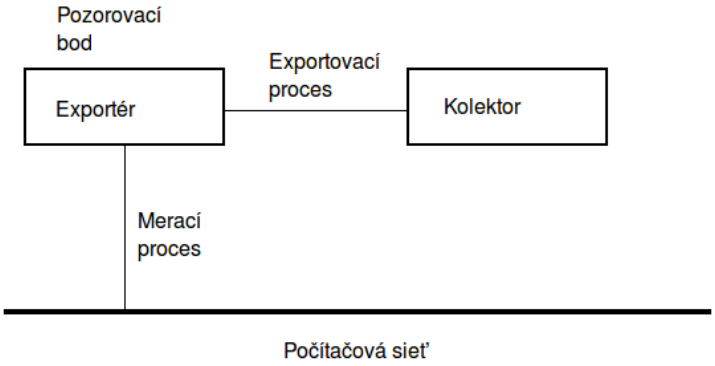
# 1 Formulácia úlohy

Na písanie textu záverečnej práce sa používajú štýly udené v tejto šablóne (Nadpis záverečnej práce, Podnadpis záverečnej práce, Text záverečnej práce [riadkovanie 1.5, Times New Roman 12] a ďalšie podľa potreby). Text záverečnej práce musí obsahovať kapitolu s formuláciou úlohy resp. úloh riešených v rámci záverečnej práce. V tejto časti autor rozvedie spôsob, akým budú riešené úlohy a tézy formulované v zadaní práce. Taktiež uvedie prehľad podmienok riešenia.

# 2 Analýza

## 2.1 Analýza protokolu IPFIX

Protokol IPFIX (Claise et al., 2008; IPFIX protocol; Juvhaugen, 2007; Vereščák, 2012) 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 (IPFIX charter, 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.




**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 splna 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 bytov, 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 odchytávanie hlavičiek paketov, vytváranie časových známok, 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.

**Šablóna** špecifikuje formát odosielaných záznamov o tokoch pomocou zoznamu informačných elementov. Musí byť odoslaná kolektoru z exportovacieho procesu ešte pred odoslaním samotných dát. Neskor 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)

**Tabuľka 2 – 1** Základné skupiny informačných elementov podla (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. | Padding |

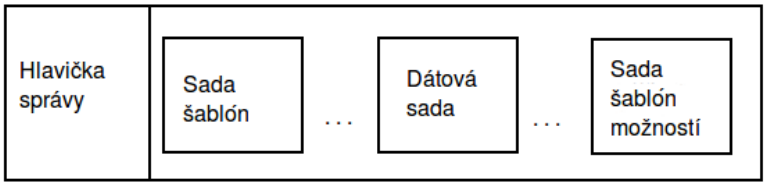
**Informačné elementy** sú protokolovo nezávislým popisom atribútov záznamov o tokoch. Informačný model IPFIX (Quittek, et al., 2008) obsahuje základnú množinu informačných elementov, vrátane ich popisu, významu, dátového typu a pod. Informačné elementy sú rozdelené do 12 skupín, pozri tabuľku 2 – 1, 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 [1].

### 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 bytov (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.



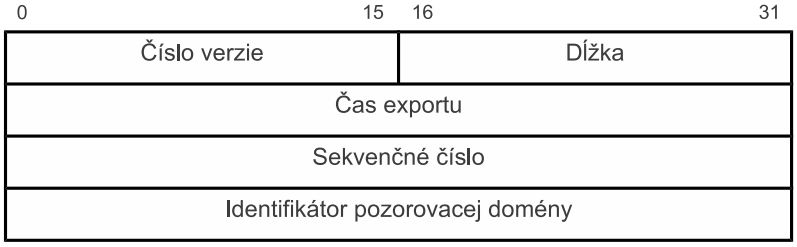

**Obr. 2 – 2** Príklad formátu IPFIX správy

**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.

[1] http://www.iana.org/assignments/enterprise-numbers

- **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.



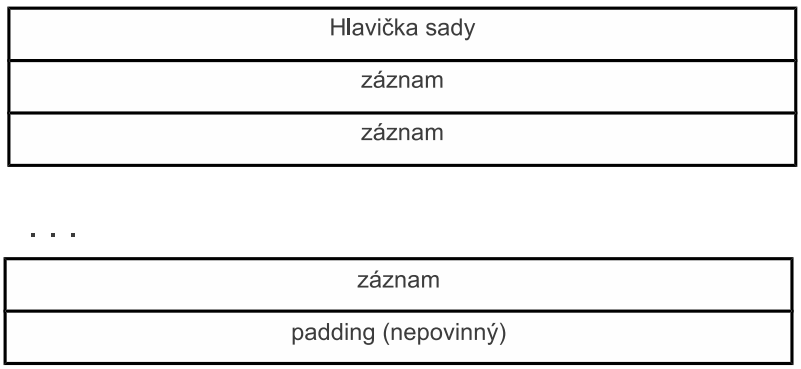

**Obr. 2 – 3** Formát hlavičky IPFIX správy

**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 sad sa skladá z hlavičky sady a jedného, alebo viacerých záznamov sady, obrázok 2 – 4. Na úplnom konci môže byt vložený 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 2 – 2. Sada istého druhu nemôže obsahovať záznamy iného typu. Hodnotu 2 je rezervovaná pre sadu šablón. Sada šablón možností nadobúda hodnotu 3. Iden-



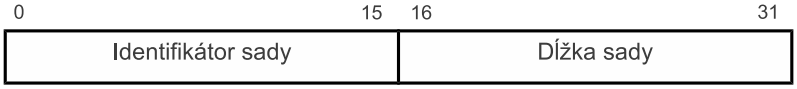

**Obr. 2 – 4** Formát sady

tifikátory 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.

**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 |

*Dĺžka sady* zahŕňa celková dĺžka všetkých sád, vrátane hlavičky sady a prípadne paddingu. Na jej základe sa určuje začiatok ďalšej sady, pretože sada môže obsahovať rôzny počet záznamov.

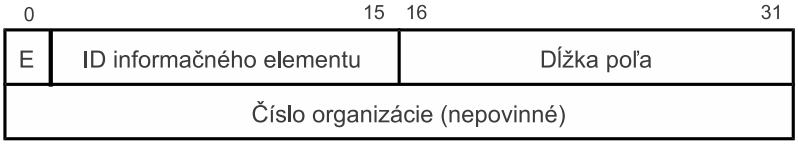



**Obr. 2 – 5** Formát hlavičky sady

**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.

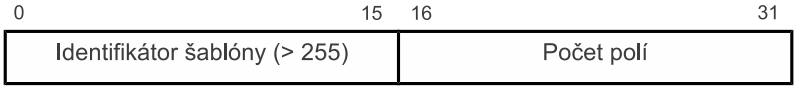
Prvý bit sa nazýva *Enterprise bit*. Ak je nastavený na 0, tak hovoríme o oficiálnom informačnom elemente charakterizovanom organizáciou IETF a registrovanom v IANA. V tomto prípade je *číslo organizácie (PEN)* nevyplnené. V opačnom prípade, keď je tento bit nastavený na 1, ide o organizáciou špecifikovaný 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 dekódovať dátové záznamy. Šablóny môžu obsahovať akúkoľvek kombináciu informačných elementov. Či už oficiálnych, navrhnutých spoločnosťou IANA, alebo vlastných, vytvorených organizáciami.

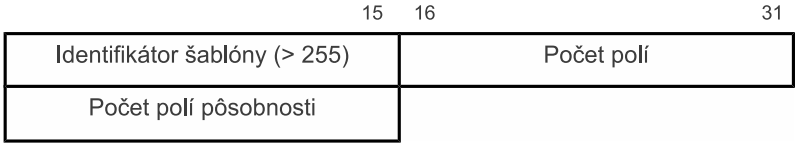
Záznam šablóny tvorí *hlavička*, pozri 2 – 7, nasledovaná *špecifikátormi poľa*. Každá šablóna musí mať v rámci transportnej relácie a pozorovacej domény jedinečný *identifikátor*. Čísluje sa od 255 do 65535, rovnako ako identifikátory dátových sád, pretože každá šablóna referuje na dátovú sadu, ktorej 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 posielaných dát, ktoré by zo samotných záznamov tokov nevedel vyčítať. Príkladom týchto informácií sú kľúče tokov, alebo konfigurácia šablóny, vzorkovacie parametre a pod.

Formát záznamov 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 naviac 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.




**Obr. 2 – 8** Formát hlavičky záznamu šablóny možností

**2.1.2.6 Formát dátových záznamov** Dátové záznamy sú posielané v dátových sadách. Ich formát je veľmi jednoduchý. Pozostávajú len z hodnôt polí, nemajú ani vlastnú hlavičku. Sú kódované podla 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 dekódovať, musí poznať formát šablóny už pred prijatím prvého dátového záznamu.

## 2.2 Analýza sprostredkovania správ v IPFIX

Výhodou monitorovania sieťovej prevádzky na báže 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* vypĺňa 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 podkapitole 2.1.1 na strane 3. Dodatočne zadefinujeme nasledujúce termíny:

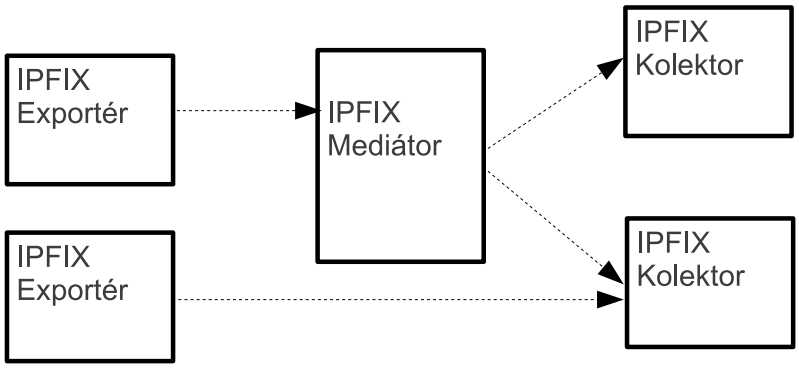
**Prúd záznamov** - *record stream* je sled dát nesúcich informácie o tokoch.

**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 zapi-

sovaču 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 architektúry bez Mediátora

Problematika sprostredkovania IPFIX správ je podrobne spracovaná v (Kobayashi, Claise, 2010). Hovori o tom, že sieťoví administrátori často celia 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 prispôsobovania niektorých parametrov meracích nástrojov zdrojom meracieho systému zatiaľ čo musia

naplniť patričné podmienky ako sú *presnosť nameraných dát*, *granularita toku*, či *spoľahlivosť 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** Veľké 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. Podla (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 agregované toky a pod.

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, udržovanie presnosti dát a granularity tokov tak, aby boli splnené požiadavky rôznych monitorovacích aplikácii vyžaduje škálovateľnú a flexibilnú zhromažďovaciu infraštruktúru.

Toto zhrnutie implikuje, že nové sprostredkovateľské funkcie sú potrebné 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. Uveďme aspoň niektoré.

#### 2.2.3.1 Prispôsobovanie granularity tokov

Najzákladnejšia sada kľúčov toku je pätica: *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.

#### 2.2.3.2 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 aplikácii 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 aplikáciam.

#### 2.2.3.3 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 implementácii 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 nezaťažuje exportér ďalšími výpočtami.

**2.2.3.4 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 podla 802.3ad (IEEE 802.3ad, 2000).
2. **Spájanie priestoru viacerých pozorovacích domén jedného exportéra**
   Tak isto ako v predchádzajúcom príklade 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 klientskych 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.

**2.2.3.5 Anonymizácia dátových záznamov** IPFIX exporty krížom cez administratívne domény, tak ako to bolo popísané v prípade 4 v podkapitole 2.2.3.4, na strane 16 môžu byt 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.

**2.2.3.6 Distribúcia dátových záznamov** Trendom v moderných dátových sieťach je súčasný prenos dát, hlasu a video komunikácie jednou spoločnou infraštruktúrou. Takéto siete nazývame konvertovanými 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 zátaže. Pod typom dátového záznamu máme na mysli napríklad typ sieťového protokolu, ktorým boli dáta prenášane.

Jedna z možných implementácií v tomto prípade používa replikáciu IPFIX správy v exportéri pre viac kolektorov. Každý kolektor potom dekóduje 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.

**2.2.3.7 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.

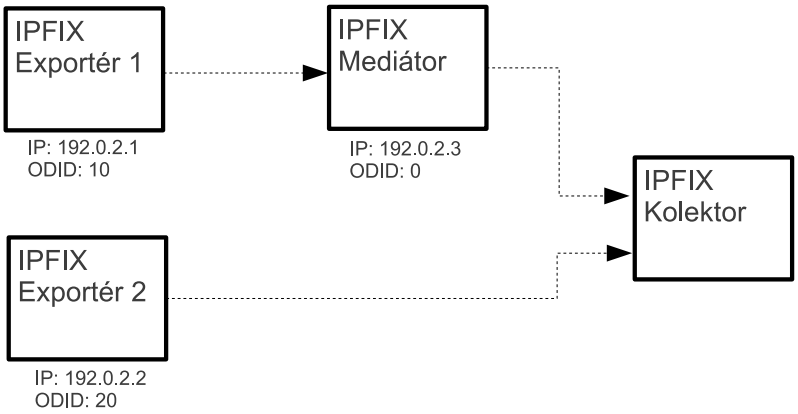
Jedna z možností je použiť IPFIX Mediátor, ktorý bude konvertovať starší protokol na IPFIX.

### 2.2.4 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 informácii. V prvom rade to je IP adresa exportéra, ktorá bola získavaná zo zdrojovej IP adresy transportnej relácie, 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, alebo dokonca ID pozorovacieho bodu. Ak Mediátor tieto informácie o exportéri neoznámi 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. (Me-

diátor) 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.

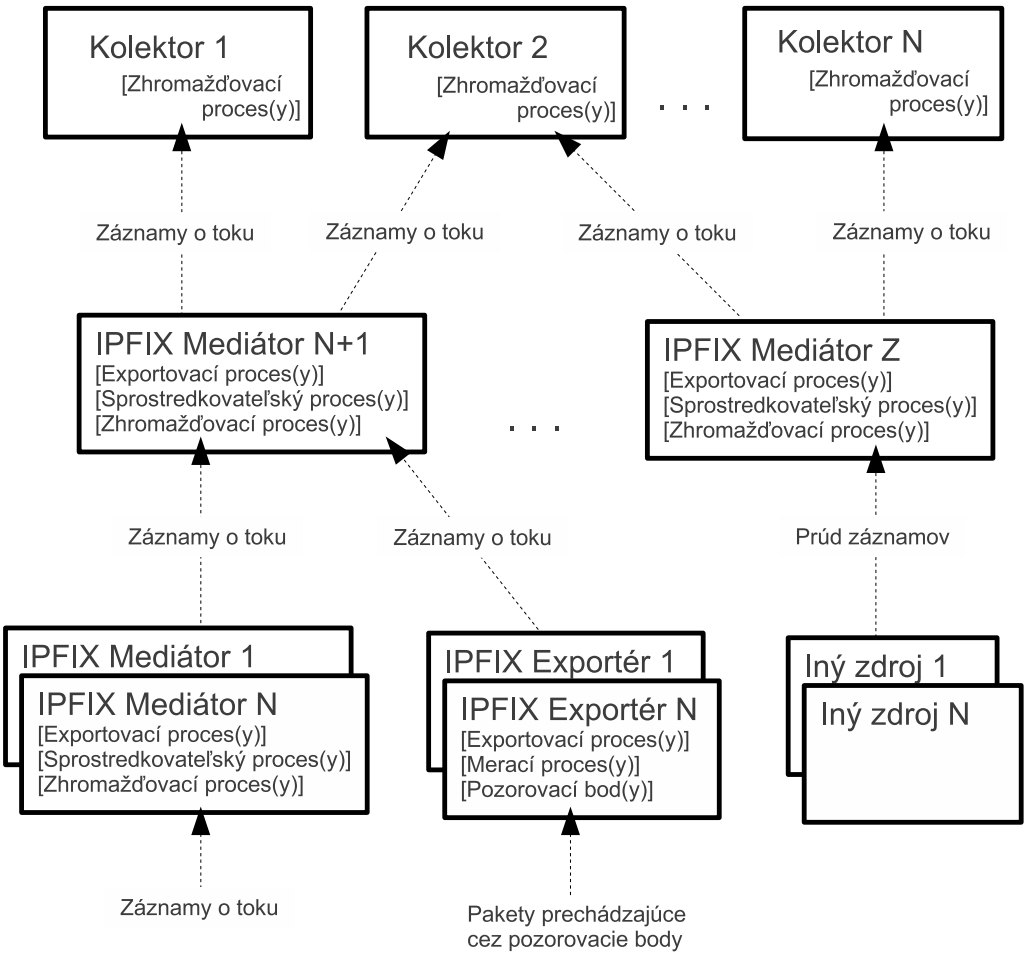
V prípade spájania času, kolektor musí poznať minimálne aktívny timeout *active timeout* pôvodných záznamov o tokoch. Pri spájaní priestoru je potrebné poznať nad akou oblasťou bola vykonaná kompozícia dátových záznamov.(Kobayashi, Claise,

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 jednotlivých kapitolách si podrobnejšie priblížime referenčný model, vybrané funkčné bloky aplikačného rámca a TODO ...

### 2.3.1 Referenčný model sprostredkovania správ v IPFIX

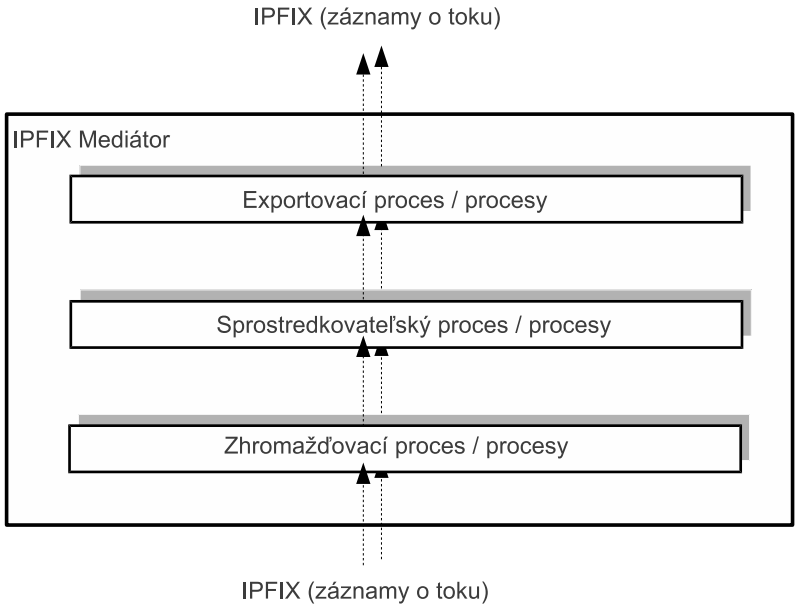



**Obr. 2 – 11** Referenčný model sprostredkovania správ v IPFIX

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). Spracovane dáta vo forme záznamov o toku potom exportuje jednému alebo viacerým kolektorom a mediátorom.




**Obr. 2 – 12** Zjednodušený model komponentov IPFIX Mediátora

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.

### 2.3.2 Komponenty sprostredkovania správ v IPFIX

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

**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 naviac je odovzdanie sady dátových záznamov a riadiacich informácii jednému, alebo viacerým komponentom, tj. sprostredkovateľským procesom, alebo ďalším aplikáciam. 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 Message).*
- Z dôvodu uvedeného v kapitole 2.2.4.1 na strane 18, je potrebné preposielať informácie o pôvodcovi dat (exporterovi), napríklad ID pozorovacieho bodu a pozorovacej domény, IP adresa exportéra atď. Tieto dáta zakóduje do prídavných dátových záznamov, bud s využitím informačných elementov skupiny 2 (tabuľka 2 – 3), alebo organizáciou špecifikovaných elementov.

**2.3.2.3 Sprostredkovateľské procesy** Sprostredkovateľské procesy sú kľúčovými funkčnými blokmi sprostredkovania správ v IPFIX. Musia pokryť každý príklad použitia sprostredkovania správ z kapitoly 2.2.3 na strane 14. 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 sprostred-

**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 |

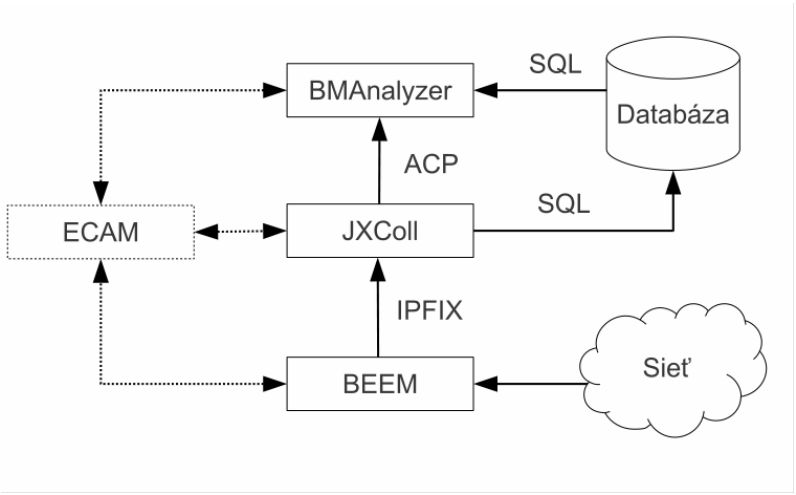
kovateľskými procesmi paralelne tak, aby boli splnené požiadavky koncových aplikácií. V tomto scenári, každý sprostredkovateľský proces dostáva kópiu celého prúdu záznamov ako vstup.

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

# 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 posiela komponentu *JXColl* vo formáte IPFIX správ. Kolektor dekóduje prijaté spravy od jedného alebo viacerých exportérov a ukladá ich do databázy kvôli neskoršej analýze. Za účelom analyzovania dát a ich grafickeho zobrazenia vo forme grafov v reálnom čase ich posiela *BMAnalyzeru* 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 pripadné útoky, resp. anomálie. *ECAM* umožňuje centrálne 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

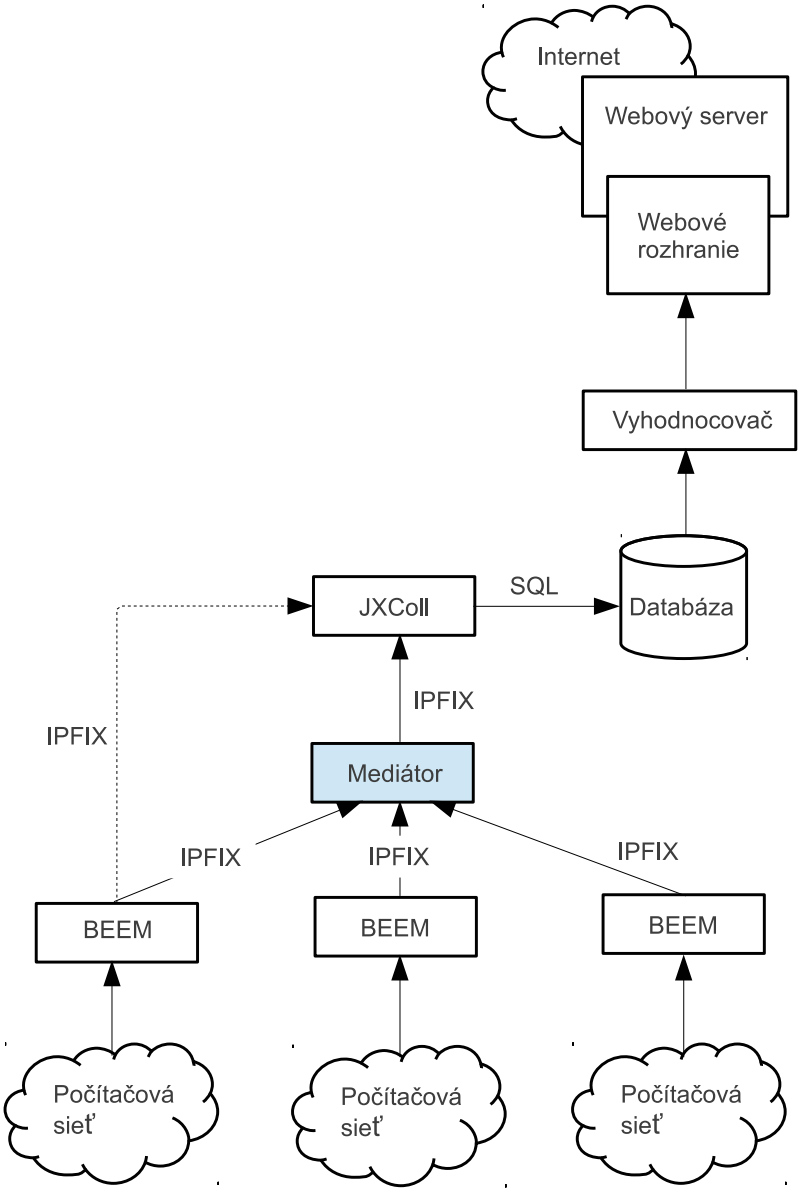
SLAmeter je merač parametrov sieťovej prevádzky vyhodnocujúci dodržiavanie zmluvy o úrovni poskytovanej služby *(SLA)*. V tomto prípade sa pod poskytovanou službou rozumie prístup do siete Internet. Základ nástroja je postavený na komponentoch nástroja BasicMeter, a rovnako je projektom výskumnej skupiny 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 sprav v IPFIX (pozri kapitolu 2.3, strana 21) a analýzy exportovacieho a zhromažďovacieho procesu v RFC 5101 (Claise et al., 2008) som zhrnul požiadavky a navrhol samotnú architektúru IPFIX Mediátora. Jeho jednotlivým komponentom, ktoré sú vyssie znázornené na obrázku 2 – 12 a požiadavkám sú venované nasledujúce kapitoly. Upozorňujem, že termíny „sprostredkovateľský proces“ a „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 ich jednoduché a flexibilné pridávanie resp. odoberanie. Zdôrazňujem, 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 dekódovanie dátových záznamov po svojom. Preto je potrebné navrhnúť a implementovať rozhranie, prostredníctvom ktorého budú musí 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.

- **Transparentnosť vzhľadom na kolektor** - 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.
- **Komunikácia pomocou UDP** - v tejto fáze projektu som zvolil 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.
- **Jednoduchá konfigurácia** - ako už bolo spomenuté v analýze (kapitola 2.3.2.3, strana 23), sprostredkovateľské procesy spracúvajú prijaté záznamy o tokoch bud 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 a zhromažďovacím a exportovacím procesom. Tieto spôsoby musia byť pre procesy jednotné, transparentné a bez možnosti zmeny z vnútra sprostredkovateľského procesu.
- **Jediná inštancia modulov** - navrhol som, 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** - zhromažďovací a exportovací proces aplikačného rámca sa nesmú líšiť od charakteristík týchto procesov daných špecifikáciou IPFIX protokolu v RFC 5101 (Claise et al., 2008).
- **Programovací jazyk Java** - pre implementáciu som zvolil programovací jazyk Java. Hlavným dôvodom bol fakt, že zhromažďovací proces Mediátora a IPFIX kolektora sú veľmi podobné a kolektor JXColl je naprogramovaný v tomto jazyku. Ďalším faktom je relatívne jednoduchá tvorba modulárnych aplikácii, 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

Ulohou hlavnej triedy Mediatora je postupne spustit vsetky vlakna a procesy potrebne pre beh programu. Najprv sa precitaju a spracuju argumenty prikazoveho riadku. Program vie rozpoznavat dva druhy argumentov. Prvym je cesta ku konfi-

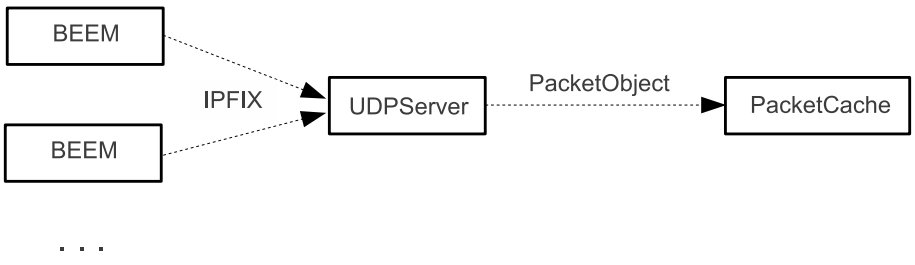
guracnemu suboru. Ak nie je zadana, pouziva sa vychodiskovy konfiguracny subor. Druhym argumentom moze byt zadana moznost `--logtofile`. Vtedy su vsetky logovacie vystupy presmerovane zo standardneho vystupu do suboru.

Potom ako program nacita vsetky nastavenia z konfiguracneho suboru, spusti vsetky svoje moduly - sprostredkovateľské procesy pomocou triedy `IPLoader`. Nasleduje spustenie vlakna, ktore prijima IPFIX pakety prostrednictvom protokolu UDP a vlakna, ktore ich spracovava. Hovorime o `UDPServer` a `UDPProcessor`. Nakoniec je spustene exportovacie vlakno - `UDPExporter`. Kedykolvek ked nastane chyba je Mediator korektne ukonceny a to tak, ze uvolni vsetku pamat a zastavi beziace vlakna. Rovnako je Mediator zastavny po stalceni kombinacie klaves `Ctrl+c`. Podrobne o kazdom spomenutom vlakne a procese bude povedane v nasledujucich kapitolach.

## 4.3 Zhromažďovací proces

Na základe analýzy a zhodnotenia požiadaviek na zhromažďovací proces som navrhol jeho architektúru. 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




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

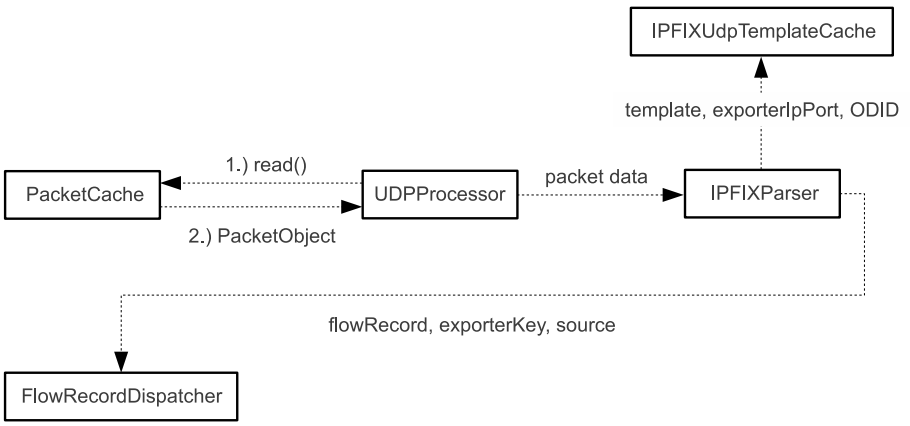
Prvá fáza je znázornená na Obr. 4 – 1 a predstavuje najnižšiu vrstvu celého nástroja. Jej jadrom je UDP server, bežiaci v samostatnom vlákne. V jeho hlavnej

metóde *run()* cyklicky vykonáva kód, dokiaľ nie je prerušený výnimkou *InterruptedException*. Tento cyklicky kód odchytáva údaje posielané protokolom UDP na úrovni bytov a ukladá ich do buffera. 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 (formát 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 pozorovacej domény) spolu s obsahom správy, časom prijatia a adresy z ktorej bol prijatý. Prijatá IPFIX správa vo forme inštancie triedy PacketObject sa uloží do vyrovnávacej pamäte pre správy, 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. Tato vyrovnávacia pamäť medzi prvou a druhou fázou zhromažďovacieho procesuje kritická vo vysoko rýchlostných sieťach.

Návrh do budúcnosti umožňuje jednoduché rozšírenie o serveri iných protokolov, napr. TCP a SCTP. Tieto serveri budú rovnako ako UDPServer bežiace v samostatných vláknach.

### 4.3.2 2. fáza zhromažďovacieho procesu

Schému druhej fázy vidíme na Obr. 4 – 2. Hlavná metóda *run()* vlákna *UDPProcessor* cyklicky vyberá dáta z PacketCache a predáva ich *parseru*, reprezentovaného triedou *IPFIXParser*, dokiaľ nie je prerušený výnimkou *InterruptedException* vyhodenou pri čítaní z vyrovnávacej pamäte. UDPProcessor 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 z vyrovnávacej pamäte. Trieda *IPFIXParser* sa používa na spracovanie prijatých paketových binárnych dat, ktorého výsledkom je hotový objekt IPFIX správy,




**Obr. 4 – 2** Schéma druhej fázy zhromažďovacieho procesu Mediátora

obsahujúci všetky komponenty správy, pozri kapitolu 2.1.2, strana 6. Metódy tejto triedy najprv vykladajú kompletnú hlavičku správy a potom sa pustia do parsovania IPFIX sád. Na základe identifikátora sady spracujú a vytvoria objekty pre sadu šablón, sadu šablón možnosti a dátovú sadu. Sady následne naplnia príslušnými záznamami.

Prichádzajúce záznamy šablón a záznamy šablón možnosti sú posielané triede *IPFIXUdpTemplateCache*, ktorá má na starosti spravu prijatých šablóny osobitne pre každý exportér. Ako je zrejme 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 ukladá nove šablóny a objekty šablón, ktoré už má uložené 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. Druhým parametrom, ktorý sa posiela triede *FlowRecordDispatcher* je reťazec, ktorý určuje odkiaľ záznam vystupuje *(inputProcess)*. V tomto prípade je to *„exportér“*. Trieda FlowRecordDispatcher rozdistribuuje prijaté

záznamy o tokoch príslušným sprostredkovateľským procesom, alebo ich pošle na export. O tom podrobnejšie v samostatnej kapitole 4.5, na strane 44.

## 4.4 Rozhranie a podpora pre sprostredkovateľské procesy - moduly

Vyššie boli definované viaceré požiadavky na sprostredkovateľské procesy, ktorý musí zabezpečiť aplikačný rámec. Jedná sa o modulárnu implementáciu, oddelenie logiky rámca od logiky procesov, ich dynamické načítavanie, jediná inštancia procesov atď.

### 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 samo zavádzací *(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`. Jeho potomkom 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. Sys-

témový class loader získavame metódou `ClassLoader.getSystemClassLoader()`. (TechJava, 2008; Antl, 2012)

Abstraktná trieda `ClassLoader` je umiestnená v balíčku `java.lang`. Vývojári môžu pridať vlastnú funkcionalitu 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)`. Tuto metódu som použil pri načítavaní modulov definovaných v konfiguračnom súbore a aj pri získavaní inštancii sprostredkovateľských procesov, ktoré boli potrebné pri riadení toku dát medzi procesmi. Podrobnejšie sa tomu venujem 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 Viacvláknovosť** Každý modul musí byť vykonávaný v samostatnom vlákne. Preto trieda *AIntermediateProcess* dedi 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.
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 vytvo-

riť novu inštanciu potomka a nie rodičovskej triedy. Avšak inštanciu ktorého potomka?

4. Druhý prístup je deklarovať metódu *getInstance()* abstraktnou. Konkrétna implementácia 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 [2]. *Static* metóda patrí triede, kde je definovaná. Pričom *abstract* naznačuje, že funkcionalita bude definovaná až v potomkoch. Tu dochádza k logickému rozporu.

Preto bolo potrebné navrhnúť a implementovať iný spôsob, ktorý by zabezpečil jedinú inštanciu pre všetky moduly z abstraktnej rodičovskej triedy. Riešenie navrhol britsky 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 statickú metódu *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);
      }
```

[2] Niektoré jazyky, napr. Python túto možnosť dovoľujú.

```
      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 inštancií používa reflexiu, ktorá je pomalá. Avšak, keďže vytvárame Singletony, volanie *newInstance()* sa vykoná pre každý modul pravé raz.

Aby bolo možné získavať inštancie modulov aj na základe mena triedy a nie len cez class objekty, vytvoril som ďalšiu metódu *getInstance(String processName)*. Parameter *processName* je práve meno procesu, napr `SelectionProcess`. Premenná *name* je binárne meno procesu, podľa špecifikácie jazyka Java (Oracle, 2013), napr. `sk.tuke.cnl.Mediator.SelectionProcess`. Tato metóda načíta *class* objekt sprostredkovateľského procesu cez systémový class loader, tak ako to bolo vyššie spomínané. Potom zavolá pôvodnú metódu *getInstance(Class clazz)* a 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 aplikačný 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 informácii 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 `IpfixFieldAttributes`. 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, druha fáza dekóduje byty 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 daný informačný element sa 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 byty, obalí ich do objektu triedy *ByteBuffer* a ten pošle spolu s pomenovaním dátového typu na dekódovanie triede `IPFIXDecoder`.

Autorom tejto triedy je Tomáš Vereščák. Na základe dátového typu je určená metóda, ktorá dekóduje byty 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. Vymenujem aspoň niektoré: znamienkové a bezznamienkové celé čísla na 8, 16, 32 a 64 bitoch, čísla v pohyblivej rádovej čiarke 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 dekódovanými 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 dekódovania), 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. Tuto triedu som navrhol a implementoval analogicky k dekodéru. Aj tu sú pokryté všetky dátové typy, ktoré podporuje IPFIX protokol vrátane jedného naviac - bezznamienkového celého čísla na 128 bitoch - *unsigned128*. Tento dátový typ využívajú niektoré informačné elementy definované Laboratóriom Počítačových Sietí na Technickej Univerzite v

Košiciach. Podla špecifikácie (Claise et al., 2008) musia byť zakódované informačné elementy posielané v sieťovom poradí bytov, 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 vyhodená príslušná výnimka a kódovanie je úplne ukončené, dátový záznam sa neexportuje. Je neprípustné, aby Mediátor posielal kolektoru dátové záznamy s prázdnymi hodnotami z dôvodu, že nebolo možné zakódovať hodnotu danú v zlom formáte.

Pri kódovaní je veľmi dôležitá rýchlosť a pamäťová nenáročnosť kódovacích funkcii. Jedna funkcia, napr. na zakódovanie znamienkového celého čísla na 32 bitoch môže byť zavolaná veľakrát v rámci kódovania jediného dátového záznamu. Tento počet zavisí od dátových typov informačných elementov v dátovom zázname. Nemusím zdôrazňovať aké množstvo IPFIX paketov a dátových záznamov môže Mediátor v čase prijímať. Preto som dával 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 bytov. Najjednoduchším a najpohodlnejším riešením je použiť triedu *ByteBuffer*, alokovať pole veľkosti 4, nastaviť poradie bytov 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 bytov. V tejto metóde sa nedá explicitne nastavovať poradie bytov, pole je stále zoradené v *Big-Endian*, čo nám vyhovuje. Nepríjemnosťou je dodatočné orezanie na potrebný počet bytov.

```
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 som pre všetky konverzie implementoval metódy pomocou bitových posunov a bitových operátorov. Tieto metódy prevádzajú všetky číselne primitívne typy jazyka Java (byte, short, int, long, float a double) na pole bytov. Ukážme si túto konverziu na 4-bytovom 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 bytov sa uloží do pomocného poľa spomínaného vyššie, na rovnakú pozíciu ako je pozícia špecifikátor poľa v šablóne. Keď sa prejdu všetky hodnoty pripravene na zakódovanie, pomocné pole je vykladané 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 synchronizovanej metóde `dispatchFlowRecord(...)`. Jej jedinou úlohou je zavolanie rovnomennej metódy distribútora záznamov - triedy `FlowRecordDispatcher`. O tom podrobne v samostatnej kapitole 4.5 na strane 44.

### 4.4.3 Príklad implementácie modulu - ExampleProcess

Pre budúcich riešiteľov som pripravil 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 *vstupne záznamy*. Vstupný buffer jej napĺňa 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 bude 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>`. Tato 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 uvediem v nasledujúcej kapitole 4.5.

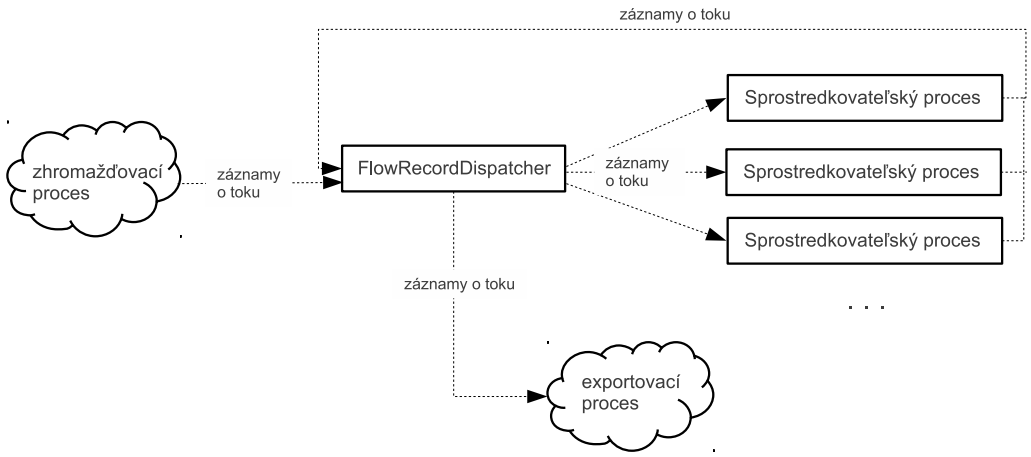
Parser konfiguračného súboru spracuje tieto položky a vytvorí zoznam modulov, ktoré sa majú načítať. Dynamické načítavanie tried podľa ich mena bolo analyzované v kapitole 4.4.1 na strane 34. Načítavanie modulov má na starosti trieda `IPLoader`. Jej hlavná metóda `loadProcesses()` cyklicky prechádza zoznam reťazcov obsahujúcich názvy modulov. Meno každého modulu prevedie na binárny názov a vďaka *systémovému class loader*-u, získa jeho *class objekt*. Podmienkou, však je, že hlavná trieda modulu musí byť v rovnakom balíčku ako trieda `AIntermediateProcess`. Ta 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 výstupnom 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 hovorime 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 „exportér“. Pri konfigurácii ako je daná v príklade by zoznam obsahoval dva reťazce: SelectionProcess a AggregationProcess. Teraz potrebujeme získať inštancie týchto tried a ich vstupným bufferom 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 som sa jej venoval v časti *Jediná inštancia modulov* na strane 35. Každej takto získanej inštancií sprostredkovateľského procesu zapíše do vstupného buffera *inputBuffer* záznam o toku.

Vstupný buffer implementuje trieda `IPInputBuffer`, ktorá je analógiou k triede `PacketCache`. Samotnú buffer rovnako predstavuje synchronizovaný a vysoko výkonný *FIFO* front, implementovaný pomocou `ArrayBlockingQueue`. Do cache 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)

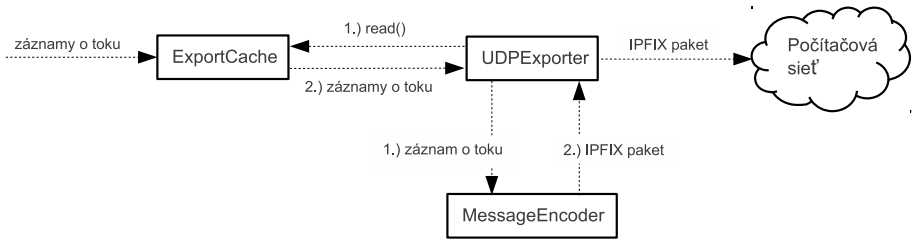
V zhromažďovacom procese, v kapitole 4.3 na strane 31, sa do frontu zapisuje metódou `put()`. V prípade, že sa *PacketCache* naplní, je na mieste zablokovať *UDPServer* a počkať a uvoľnenie miesta vo vyrovnávacej pamäti. Avšak na tomto mieste by to nebolo vhodné. Stačilo by, že by jeden sprostredkovateľský 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 vstupného buffera 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 povedané v predchádzajúcej kapitole, záznamy o tokoch, ktoré sú výstupom „koncových“ sprostredkovateľských procesov sú prostredníctvom dispečera tokov pripravené na export zapísaním do exportovacej pamäte.

Exportovaciu pamäť predstavuje trieda `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 blokované, ale 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 soket 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 posiela triede `MessageEncoder`, ktorá z neho poskladá IPFIX paket.

`MessageEncoder` vo svojich metódach postupne tvori obsah IPFIX správy podľa formátu, ktorý som analyzoval v kapitole 2.1.2 na strane 6. 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 prislúchajúca dátovým záznamom obsiahnutým v zázname o toku z ktorého vytvára spravu. 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 bytov. 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 bytoch. V prípade, že ide o element definovaný spoločnosťou, tak najvyšší bit sa nastaví na 1. Nasleduje dĺžka informačného elementu a v prípade potreby číslo spoločnosti definované organizáciou 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 bytov 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 bytov. Napokon je potrebné pred tento prúd bytov 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 bytov dátovej sady.

Ako posledný sa zostaví prúd bytov hlavičky IPFIX správy. Dĺžka správy je určená súčtom veľkosti hlavičky správy s veľkosťou sady šablóny a dátovej sady. Ako prvé sa do prúdu bytov 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 pozorovaciu doménu v ktorej sídli Mediátor, nie exportér. O určovaní času exportu podrobnejšie neskôr.

V analýze som uviedol implementačno-špecifické problémy Mediátora, kapitola 2.2.4,

strana 18. Prvým bola strata informácii o pôvodnom exportéri. Ako som uviedol neskôr, v analýze exportovacieho procesu v kapitole 2.3.2.2, na strane 23, spôsobom ako posielať 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 posiela akýkoľvek „*delta-*“ informačný element, napr. *flowStartDeltaMicroseconds*, tak musí byť použitý prvý spôsob určenia času exportu. Aby Mediátor vyhovoval obom prípadom, navrhol som, že spôsob určovania času exportu definuje administrátor v konfiguračnom súbore, v elemente `<exportTime>`. Ak zadá reťazec `KEEP`, tak trieda `MessageEncoder` zakóduje do prúdu bytov hlavičky pôvodnú hodnotu. V prípade reťazca `RENEW` sa zakóduje aktuálna hodnota.

Poslednou úlohou triedy `MessageEncoder` je pospájať jednotlivé prúdy bytov do výsledného prúdu IPFIX správy. Prvým je prúd bytov 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 vytvori 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 soketom je správa odoslaná.

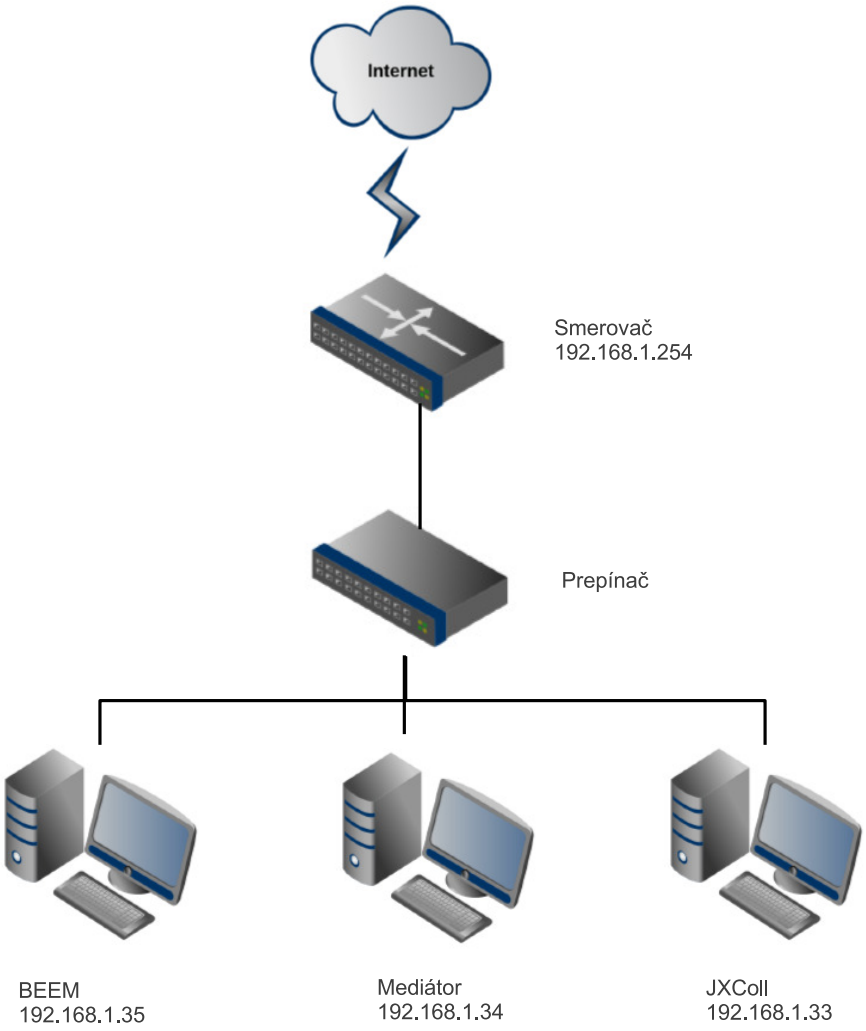
# 5 Experimentálne overenie funkčnosti riešenia

Tato kapitola je venovana overeniu spravnosti implementacie. Boli vykonane nasledujuce styri experimenty, ktore mali potvrdit, alebo vyvratit spravnu funkcnost riesenia.

1. **Prvy test** overil konektivitu a prenos udajov medzi exporterom a mediatorom a nasledne medzi mediatorom a kolektorom.
2. **Druhy test** porovnal hodnoty udajov exportovanych mybeem-om s hodnotami ulozenymi v databaze po prechode cez Mediator bez sprostredkovatelskeho procesu.
3. **Treti test** demonstroval spravnost anonymizacie udajov v podani sprostredkovatelskeho procesu `ExampleProcess`, spominaneho vyssie v navrhu a implementacii
4. **Stvrty test** bol tzv. zatazovym testom, overoval stabilitu Mediatora pri dlhodobom behu a pri spracovavani prevadzky s vysokym poctom tokov.

## 5.1 Testovacia topológia

Pri vsetkych testoch bol pouzity virtualizacny nastroj VirtualBox, v ktorom boli vytvorene tri virtualne pocitace, zapojene do topologie, ktoru mozno vidiet na Obrazku 5 – 1. Na vsetkych troch pocitacoch sa pouzival operacny system Ubuntu 12.04 LTS v desktopovej verzii. Na prvom pocitaci bol nainstalovany exporter mybeem (verzia 1.1-6 s podporou pre IPFIX Mediator), ktory exportoval spravy druhemu pocitacu na UDP port vyhradeny pre IPFIX komunikaciu - *4739*, kde bezal Mediator verzie 1.0. Ten zase preposielal spravy na treti pocitac (taktiez port *4739*), kde bol spusteny kolektor JXColl (verzia 3.9) s funkcnou PostgreSQL databazou. Pocitace boli zapojene v jednej lokalnej sieti a vsetky mali pristup na Internet.




**Obr. 5 – 1** Testovacia topológia

## 5.2 Test konektivity

Prvy experiment overoval zakladnu konektivitu medzi exporterom a kolektorom v pripade, ze je medzi ne zaradeny mediator.

```
================================================
 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** Dôkaz konektivity medzi exportérom a Mediátorom - strana exportéra

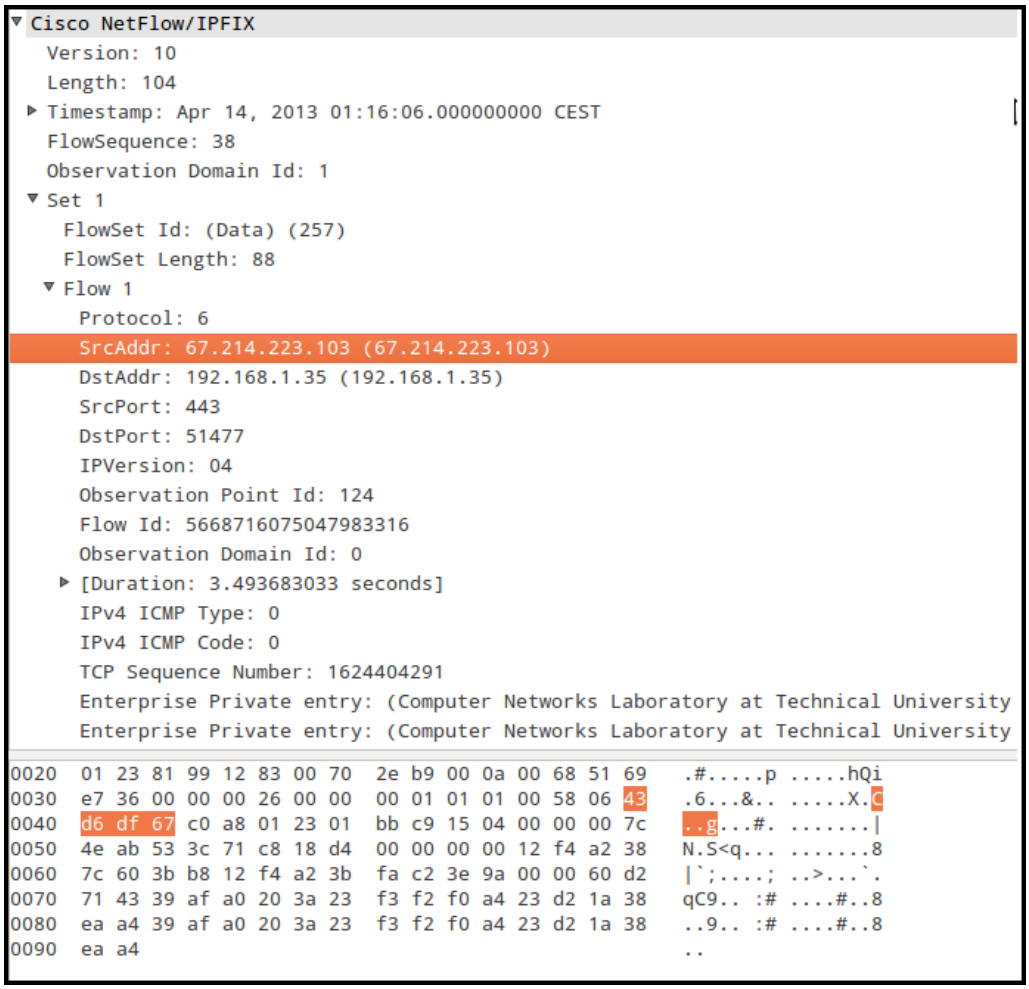
```
UDPProcessor - Packet read from /192.168.1.35:56265 (104 bytes)
UDPProcessor - Length of packet: 104
UDPProcessor - It's IPFIX packet ...
IPFIXParser - ****** IPFIX MESSAGE HEADER ******
IPFIXParser - Version number: 10
IPFIXParser - Message Length: 104 bytes
IPFIXParser - Export Time: 1365898787
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
UDPExporter - SENDING PACKET
```

**Obr. 5 – 3** Dôkaz konektivity medzi exportérom a Mediátorom - strana Mediátora

## 5.3 Test spravnej reprezentacie udajov

## 5.4 Test Mediatora s anonymizacnym modulom

## 5.5 Zatazovy test




**Obr. 5 – 4** Správy odosielané exportérom zachytené programom Wireshark

| rid | protocolidentifier | sourceipv4address | destinationipv4address | sourcetransportport | destinationtransportport |
|---|---|---|---|---|---|
| 72297 | 17 | 192.168.1.34 | 192.168.1.35 | 33177 | 4739 |
| 72296 | 17 | 192.168.1.35 | 192.168.1.34 | 33916 | 4739 |
| 72295 | 6 | 67.214.223.103 | 192.168.1.35 | 443 | 51477 |
| 72294 | 6 | 192.168.1.35 | 67.214.223.103 | 51477 | 443 |
| 72293 | 6 | 192.168.1.35 | 173.194.39.117 | 47805 | 443 |
| 72292 | 6 | 173.194.39.117 | 192.168.1.35 | 443 | 47805 |
| 72291 | 17 | 192.168.1.34 | 192.168.1.35 | 33177 | 4739 |
| 72290 | 17 | 192.168.1.35 | 192.168.1.34 | 33916 | 4739 |

**Obr. 5 – 5** Výpis obsahu databázy

# 6 Záver (zhodnotenie riešenia)

Táto časť záverečnej práce je povinná. Autor uvedie zhodnotenie riešenia. Uvedie výhody, nevýhody riešenia, použitie výsledkov, ďalšie možnosti a pod., prípadne načrtne iný spôsob riešenia úloh, resp. uvedie, prečo postupoval uvedeným spôsobom.

# Literatúra

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

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

The IPFIX Charter: *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)>.

BARANČOK, D. et al. 1995. *The effect of semiconductor surface treatment on LB*

*film/Si interface.* In: Physica Status Solidi (a), ISSN 0031-8965, 1995, vol. 108, no. 2, pp. K 87–90

BENČO, J. 2001. *Metodológia vedeckého výskumu.* Bratislava : IRIS, 2001, ISBN 80-89018-27-0

GONDA, V. 2001. *Ako napísať a úspešne obhájiť diplomovú prácu.* Bratislava : Elita, 2001, 3. doplnené a prepracované vydanie, 120 s. ISBN 80-8044-075-1

*Jadrová fyzika a technika: Terminologický výkladový slovník.* 2. rev. vyd. Bratislava : ALFA, 1985. 235 s. ISBN 80-8256-030-5

KATUŠČÁK, D. 1998. *Ako písať vysokoškolské a kvalifikačné práce.* Bratislava : Stimul, 1998, 2. doplnené vydanie. 121 s. ISBN 80-85697-82-3

LAMOŠ, F. – POTOCKÝ, R. 1989. *Pravdepodobnosť a matematická štatistika.* 1. vyd. Bratislava : Alfa, 1989. 344 s. ISBN 80-8046-020-5

SÝKORA, F. a iní. 1980. *Telesná výchova a šport.* 1.vyd. Bratislava : SPN, 1980. 35 s. ISBN 80-8046-020-5

STEINEROVÁ, J. 2000. *Základy filozofie človeka v knižničnej a informačnej vede.* In: Kimlička, Š., Knižničná a informačná veda na prahu informačnej spoločnosti. Bratislava : Stimul, 2000. ISBN 80-2274-035-2, s. 327–334

ŠUMICHRAST, Ľ. 1995. *On the performance of higher approximations of radiation boundary conditions for the simulation of wave propagation in structures of integrated optics.* In: Photonics '95. Prague : CTU, 1995, pp. 159–161

# Zoznam príloh

# Príloha A

## Prílohy

Táto časť záverečnej práce je povinná a obsahuje zoznam všetkých príloh vrátane elektronických nosičov. Názvy príloh v zozname musia byť zhodné s názvami uvedenými na príslušných prílohách. Tlačené prílohy majú na prvej strane identifikačné údaje – informácie zhodné s titulnou stranou záverečnej práce doplnené o názov príslušnej prílohy. Identifikačné údaje sú aj na priložených diskoch alebo disketách. Ak je médií viac, sú označené aj číselne v tvare $I/N$, kde $I$ je poradové číslo a $N$ je celkový počet daných médií. Zoznam príloh má nasledujúci tvar:

**Príloha A** CD médium – záverečná práca v elektronickej podobe, prílohy v elektronickej podobe.

**Príloha B** Používateľská príručka

**Príloha C** Systémová príručka

Prílohová časť je samostatnou časťou kvalifikačnej práce. Každá príloha začína na novej strane a je označená samostatným písmenom (Príloha A, Príloha B, ...). Číslovanie strán príloh nadväzuje na číslovanie strán v hlavnom texte. Pri každej prílohe sa má uviesť prameň, z ktorého sme príslušný materiál získali.

# Príloha B

## Bibliografické odkazy

Táto časť záverečnej práce je povinná. V zozname použitej literatúry sa uvádzajú odkazy podľa normy STN ISO 690–2 (01 0197) (Informácie a dokumentácia. Bibliografické citácie. Časť 2: Elektronické dokumenty alebo ich časti, dátum vydania 1. 12. 2001, ICS: 01.140.20). Odkazy sa môžu týkať knižných, časopiseckých a iných zdrojov informácií (zborníky z konferencií, patentové dokumenty, normy, odporúčania, kvalifikačné práce, osobná korešpondencia a rukopisy, odkazy cez sprostredkujúci zdroj, elektronické publikácie), ktoré boli v záverečnej práci použité.

Forma citácií sa zabezpečuje niektorou z metód, opísaných v norme STN ISO 690, 1998, s. 21. Podrobnejšie informácie nájdete na stránke `http://www.tuke.sk/anta/` v záložke Výsledky práce/Prehľad normy pre publikovanie STN ISO 690 a STN ISO 690-2.

Existujú dva hlavné spôsoby citovania v texte.

- Citovanie podľa mena a dátumu.
- Citovanie podľa odkazového čísla.

*Preferovanou metódou citovania* v texte vysokoškolskej a kvalifikačnej práce je podľa normy ISO 7144 citovanie podľa mena a dátumu (Katuščák, 1998; Gonda, 2001). V tomto prípade sa zoznam použitej literatúry upraví tak, že za meno sa pridá rok vydania. Na uľahčenie vyhľadávania citácií sa zoznam vytvára v abecednom poradí autorov.

Príklad: ... podľa (Steinerová, 2000) je táto metóda dostatočne rozpracovaná na to, aby mohla byť všeobecne používaná v ...

Druhý spôsob uvedenia odkazu na použitú literatúru je uvedenie len čísla tohto zdroja v hranatých zátvorkách bez mena autora (autorov) najčastejšie na konci

príslušnej vety alebo odstavca.

Príklad: . . . podľa [13] je táto metóda dostatočne rozpracovaná na to, aby mohla byť všeobecne používaná v . . . ako je uvedené v [14].

Citácie sú spojené s bibliografickým odkazom poradovým číslom v tvare indexu alebo čísla v hranatých zátvorkách. Odkazy v zozname na konci práce budú usporiadané podľa týchto poradových čísel. Viacero citácií toho istého diela bude mať rovnaké číslo. Odporúča sa usporiadať jednotlivé položky v poradí citovania alebo podľa abecedy.

Rôzne spôsoby odkazov je možné dosiahnuť zmenou voľby v balíku `natbib`:

```
% Citovanie podla mena autora a roku
\usepackage[]{natbib}\citestyle{chicago}
% Možnosť rôznych štýlov citácií. Príklady sú uvedené
% v preambule súboru natbib.sty.
% Napr. štýly chicago, egs, pass, anngeo, nlinproc produkujú
% odkaz v tvare (Jones, 1961; Baker, 1952). V prípade, keď
% neuvedieme štýl citácie (vynecháme \citestyle{}) v "options"
% balíka natbib zapíšeme voľbu "colon".
```

Keď zapneme voľbu `numbers`, prepneme sa do režimu citovania podľa odkazového čísla.

```
% Metoda ciselnych citacii
\usepackage[numbers]{natbib}
```

Pri zápise odkazov sa používajú nasledujúce pravidlá:

V odkaze na knižnú publikáciu (pozri príklad zoznamov na konci tejto časti):

- Uvádzame jedno, dve alebo tri prvé mená oddelené pomlčkou, ostatné vynecháme a namiesto nich napíšeme skratku et al. alebo a i.

- Podnázov sa môže zapísať vtedy, ak to uľahčí identifikáciu dokumentu. Od názvu sa oddeľuje dvojbodkou a medzerou.
- Dlhý názov sa môže skrátiť v prípade, ak sa tým nestratí podstatná informácia. Nikdy sa neskracuje začiatok názvu. Všetky vynechávky treba označiť znamienkami vypustenia „. . . “

Pri využívaní informácií z elektronických dokumentov treba dodržiavať tieto zásady:

- uprednostňujeme autorizované súbory solídnych služieb a systémov,
- zaznamenáme dostatok informácií o súbore tak, aby ho bolo opäť možné vyhľadať,
- urobíme si kópiu použitého prameňa v elektronickej alebo papierovej forme,
- za verifikovateľnosť informácií zodpovedá autor, ktorý sa na ne odvoláva.

Pre zápis elektronických dokumentov platia tie isté pravidlá, ako pre zápis „klasických“. Navyše treba uviesť tieto údaje:

- druh nosiča [online], [CD-ROM], [disketa], [magnetická páska]
- dátum citovania (len pre online dokumenty)
- dostupnosť (len pre online dokumenty)

Poradie prvkov odkazu je nasledovné: Autor. Názov. In Názov primárneho zdroja: Podnázov. [Druh nosiča]. Editor. Vydanie alebo verzia. Miesto vydania : Vydavateľ, dátum vydania. [Dátum citovania]. Poznámky. Dostupnosť. ISBN alebo ISSN.

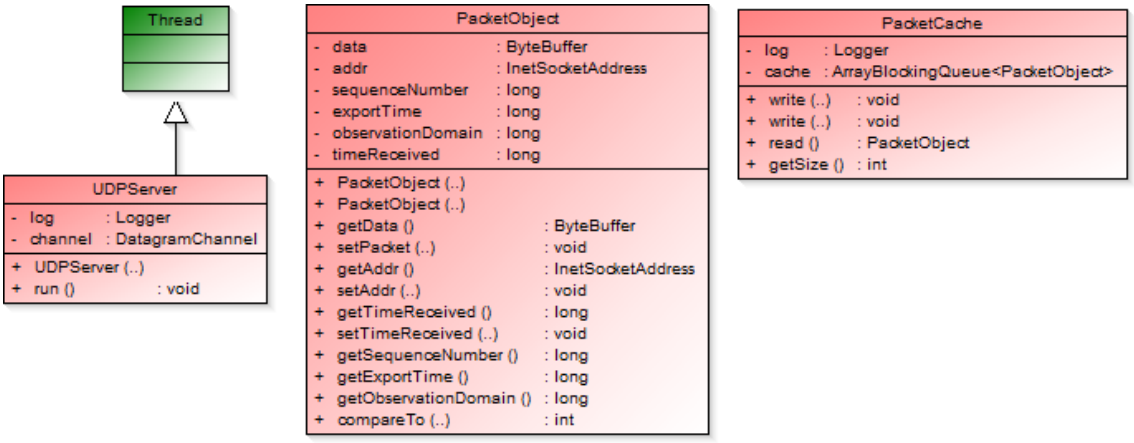
# Príloha C

## Vytvorenie zoznamu skratiek a symbolov

Ak sú v práci skratky a symboly, vytvára sa *Zoznam skratiek a symbolov* (a ich dešifrovanie). V prostredí LaTeXu sa takýto zoznam ľahko vytvorí pomocou balíka `nomencl`. Postup je nasledovný:

1. Do preambuly zapíšeme nasledujúce príkazy
   ```
   \usepackage[slovak,noprefix]{nomencl}
   \makeglossary
   ```
2. V mieste, kde má byť vložený zoznam zapíšeme príkaz
   ```
   \printglossary
   ```
3. V miestach, kde sa vyskytujú skratky a symboly ich definíciu zavedieme, napr. ako v našom texte, príkazmi
   ```
   \nomenclature{$\upmu$}{mikro, $10^{-6}$}
   \nomenclature{V}{volt, základná jednotka napätia v sústave SI}
   ```
   a dokument „preLaTeXujeme“.
4. Z príkazového riadka spustíme program `makeindex` s prepínačmi podľa použitého operačného systému, napr. v OS GNU/Linux s distribúciou Ubuntu 10.04 a verziou `texlive 2009-7` napíšeme:
   `makeindex␣tukedip.glo␣-s␣nomencl.ist␣-o␣tukedip.gls`
   v OS Win XP s verziou `TeXLive 2010` napíšeme:
   `makeindex␣-o␣tukedip.gls␣-s␣nomencl.ist␣tukedip.glo`
5. Po opätovnom „preLaTeXovaní“ dokumentu sa na požadované miesto vloží *Zoznam skratiek a symbolov*.

Zjednodušený diagram tried tejto fázy môžeme vidieť na Obr. 6 – 1.

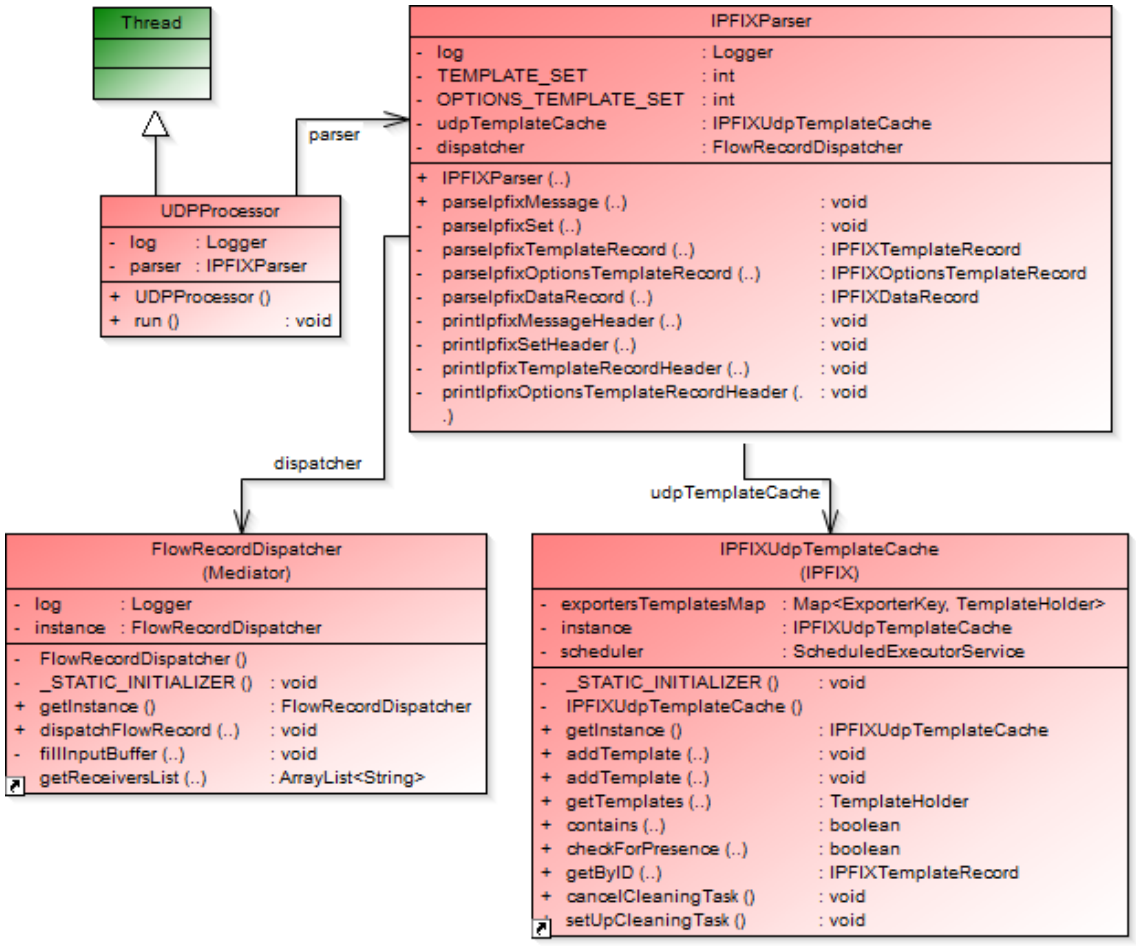



**Obr. 6 – 1** Diagram tried prvej fázy zhromažďovacieho procesu

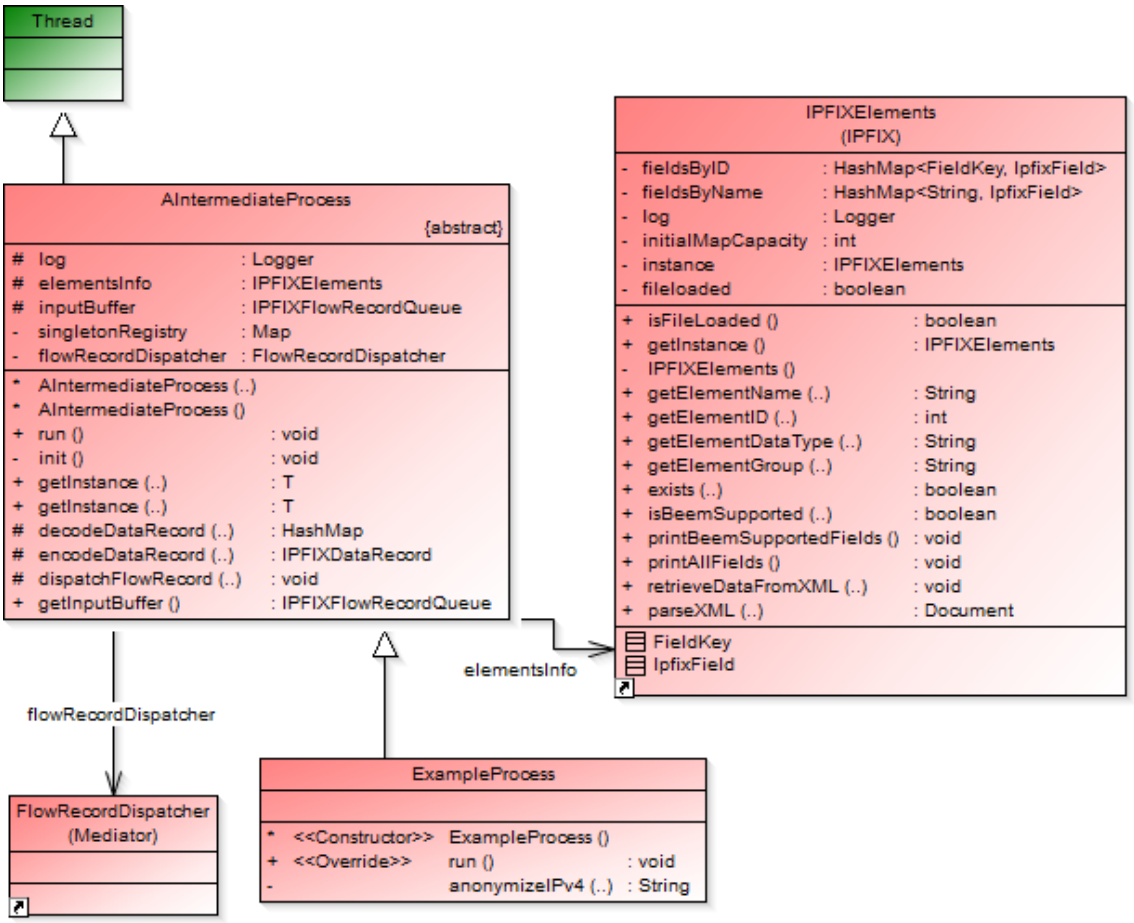
Zjednodušený diagram tried druhej fázy sprostredkovateľského procesu je na Obr. 6 – 2.

Diagram tried rozhrania pre sprostredkovateľské procesy, vrátane triedy `ExampleProcess` je znázornený na obrázku 6 – 3.

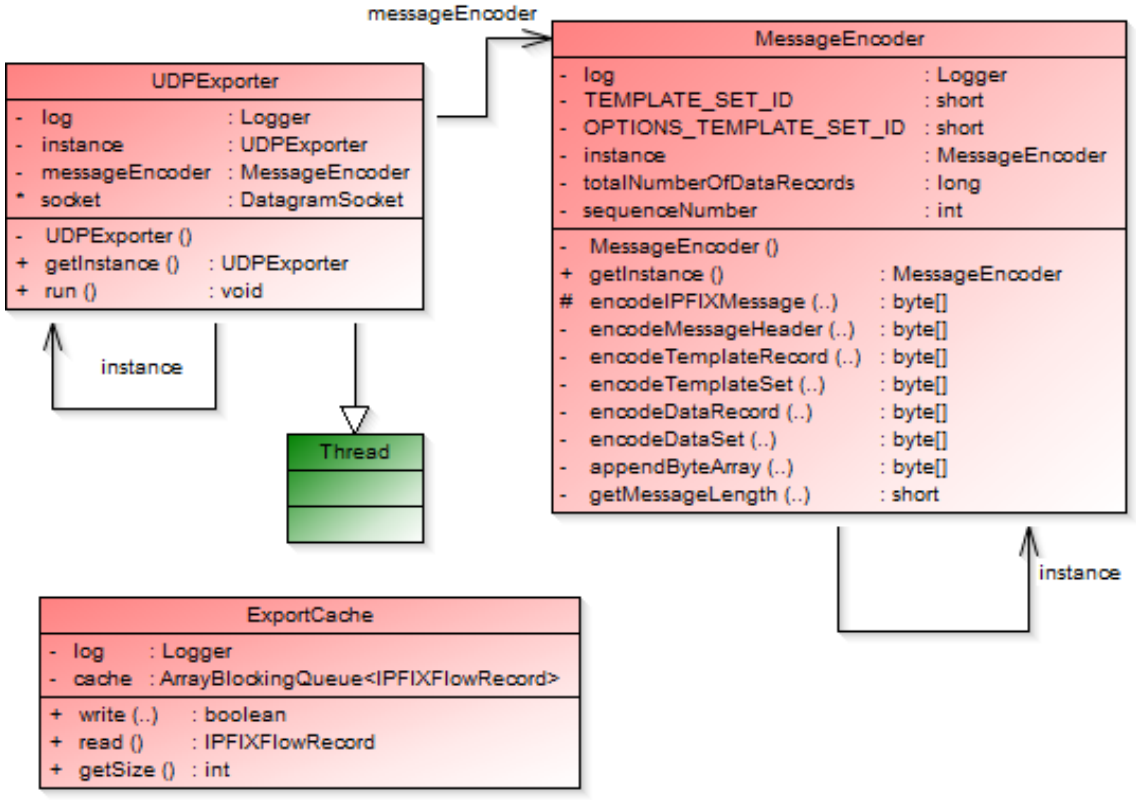
Diagram tried exportovacieho procesu.




**Obr. 6 – 2** Diagram tried druhej fázy zhromažďovacieho procesu




**Obr. 6 – 3** Diagram tried rozhrania pre sprostredkovateľské procesy




**Obr. 6 – 4** Diagram tried exportovacieho procesu