Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky
Katedra počítačov a informatiky

Aplikačný rámec pre sprostredkovanie IPFIX správ v nástroji SLAmeter

Diplomová práca

Príloha A

SYSTÉMOVÁ PRÍRUČKA Mediator v1.0

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

Košice 2013 **Bc. Rastislav Kudla**

Obsah

Zoznam obrázkov

Zoznam tabuliek

1 Funkcia programu

Program Mediator je implementaciou aplikacneho ramca pre problem sprostredkovania sprav v protokole IPFIX *(IP Flow Information Export (IPFIX) Mediation Problem)* vyvijany vyskumnou skupinou MONICA sidliacou v Laboratoriu pocitacovych sieti *(CNL)* na Technickej Univerzite v Kosiciach. Je sucastou meracej architektury SLAmeter, ktorej ulohou je pasivne meranie parametrov sietovej prevadzky na baze tokov. Na zaklade nameranych hodnot urcuje triedu kvality sluzieb a Internetoveho pripojenia poskytovatelov Internetu. Trieda kvality vypoveda o dodrziavani zmluvy o urovni poskytovanej sluzby - *SLA*.

Mediaotrebnú pre široký rad meracích aplikácii. Sprostredkovatelske moduly Mediatora mozu z pohľadu manipulácie s dátami poskytovať agregáciu, koreláciu, filtrovanie, anonymizáciu a iné úpravy záznamov o tokoch za účelom šetrenia výpočtových zdrojov meracieho systému a vykonávania predspracovania úloh pre kolektor. Z hľadiska interoperability nástrojov rôznych vývojárov, môžu poskytovať konverziu z iných protokolov na IPFIX, respektíve zvyšovať spoľahlivosť exportov napríklad prevodom z nespoľahlivého, bezspojovo orientovaného protokolu UDP na spoľahlivý SCTP.

Program bol v roku 2013 vytvorený Rastislavom Kudlom v ramci jeho diplomovej prace.

2 Analýza problému

Problematika sprostredkovania IPFIX správ je podrobne spracovaná v [1]. 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.

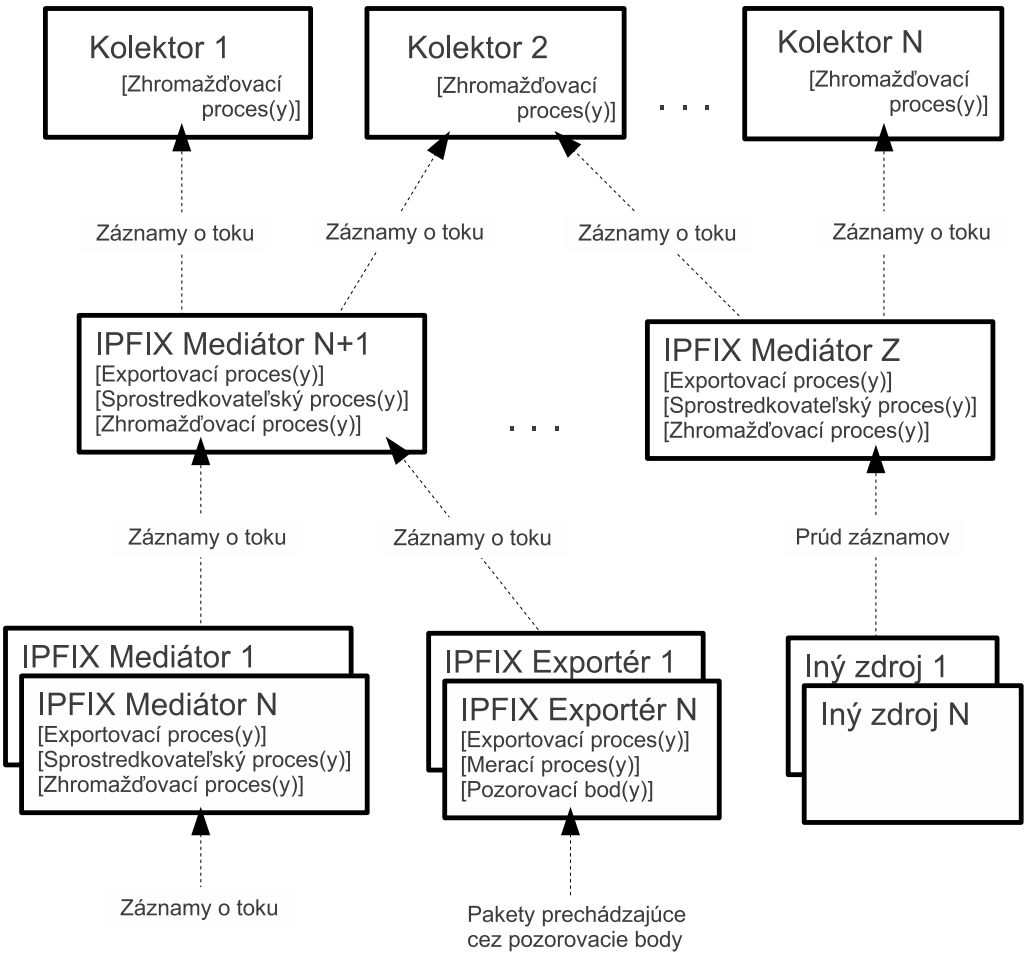
2.1 Vybrané príklady použitia sprostredkovania správ

RFC 5982 [1] uvádza viacero príkladov zaradenia IPFIX Mediátora do klasickej exportér - kolektor architektúry. Uveďme aspoň niektoré:

- prispôsobovanie granularity tokov,
- distribuovaná zhromažďovacia infraštruktúra,
- spájanie času,
- spájanie priestoru,
  - spájanie priestoru v rámci jednej pozorovacej domény,
  - spájanie priestoru viacerých pozorovacích domén jedného exportéra,
  - spájanie priestoru niekoľkých exportérov,
  - spájanie priestoru administratívnych domén,
- anonymizácia dátových záznamov,
- distribúcia dátových záznamov,
- konverzia z protokolu nizsej verzie na IPFIX,

2.2 Analýza aplikačného rámca pre IPFIX Mediátor

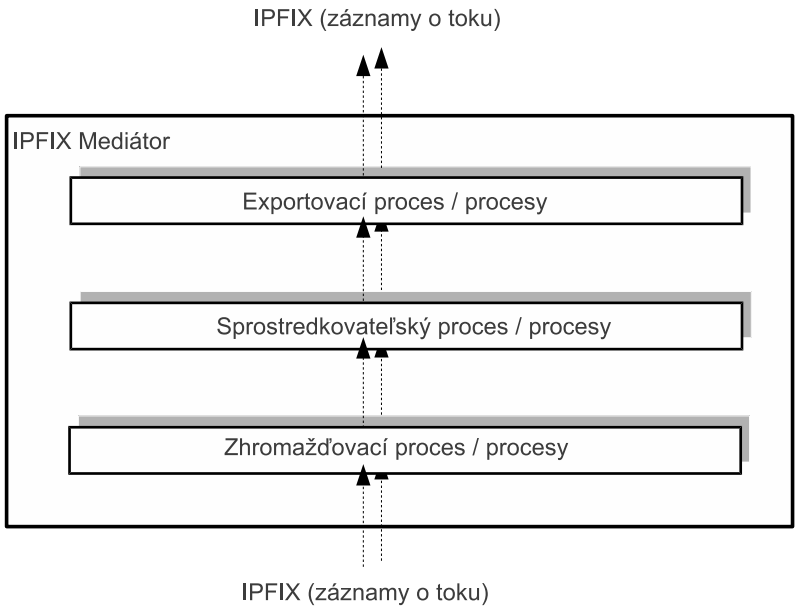
Analýze aplikačného rámca pre sprostredkovanie správ v IPFIX sa venuje RFC 6183 [2]. Na Obrazku 2 – 1 je zobrazeny referenčný model sprostredkovania správ v IPFIX ako rozšírenie referenčného modelu IPFIX, popísaného v *Architecture for IP Flow Information Export* [3]. Táto schéma zobrazuje možné scenáre, ktoré môžu existovať v meracej architektúre.



Obr. 2 – 1 Referenčný model sprostredkovania správ v IPFIX

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



Obr. 2 – 2 Zjednodušený model komponentov IPFIX Mediátora

Zjednodušený model komponentov IPFIX mediátora je predstavuje Obrázok 2 – 2. 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.

Sprostredkovateľské procesy sú kľúčovými funkčnými blokmi sprostredkovania správ v IPFIX. Musia pokryť každý príklad použitia sprostredkovania správ z kapitoly 2.1. Mediátor je schopný súčasne podporovať viac ako jeden sprostredkovateľský proces.

- **Paralelné spracovanie** - Prúd záznamov je spracovaný viacerými sprostredkovateľskými procesmi paralelne. V tomto scenári, každý sprostredkovateľský proces dostáva kópiu celého prúdu záznamov ako vstup.
- **Sériové spracovanie** - Sprostredkovateľské procesy sú zapojene sériovo. Výstupný prúd záznamov jedného procesu je vstupným prúdom nasledujúceho.

3 Popis programu

Jednotlivé časti programu sú umiestnené v nasledujúcich balíkoch:

- `sk.tuke.cnl.bm.Mediator.collecting` - implementacia zhromazdovacieho procesu
- `sk.tuke.cnl.bm.Mediator.exporting` - implementacia exportovacieho procesu
- `sk.tuke.cnl.bm.Mediator.IntermediateProcesses` - triedy tvoriace podporu pre sprostredkovatelske procesy. Hlavne triedy novych modulov musia byt umiestnene v tomto baliku!
- `sk.tuke.cnl.bm.Mediator.IPFIX` - triedy s manuálnou implementáciou protokolu IPFIX
- `sk.tuke.cnl.bm.Mediator.exceptions` - vlastne vynimky aplikacie
- `sk.tuke.cnl.bm.Mediator` - hlavné triedy samotného programu

3.1 Popis riešenia

4 Popis tried, členských premenných a metód

Kedže niektore triedy Mediatora su kvoli jednotnosti rieseni v ramci vyskumnej skupiny MONICA totozne s triedami nastroja JXColl, v nasledujúcich častiach budú uvedené len tie, ktoré sa tykaju vyhradne Mediatora. Popis ostatných tried a metód je uvedený v systemovej príručke programu JXColl [5].

4.1 Balík sk.tuke.cnl.bm.Mediator

4.1.1 Trieda Default

Trieda predstavuje rozhranie obsahujuce vychodiskove hodnoty konfiguracneho suboru. Neobsahuje konstruktor ani ziadne metody, iba verejne pristupne staticke konstanty.

4.1.2 Trieda DropsCounter

Sluzi na vypocet statistiky zahodenych entit. Pod entitou sa myslia zaznamy o tokoch, datove zaznamy, alebo IPFIX pakety. Obsahuje len staticke metody.

Metódy

public static void ***inputBufferDropsUP****()*

Zvysuje pocet strat sposobenych preplnenim vstupneho buffera sprostredkovateľských porcesov o jeden.

Parametre:

String processName - meno procesu

public static void ***exportCacheDropsUP****()*

Zvysuje pocet strat sposobenych `ExportCache` o jeden.

public static void ***encodingDropsUp****()*

Zvysuje pocet strat sposobenych chybou pri kodovani o jeden.

public static void ***decodingDropsUp****()*

Zvysuje pocet strat sposobenych chybou pri dekodovani o jeden.

public static void ***packetDropsUp****()*

Zvysuje pocet IPFIX paketov zahodenych UDP serverom o jeden.

public static void ***printStats****()*

Vypise statistiku vsetkych zahodenych entit.

4.1.3 Trieda FlowRecordDispatcher

Ulohou tejto triedy je na zaklade konfiguracneho suboru distribuovat prijate zaznamy o toku sprostredkovatelskym procesom (seriovo alebo paralelne) a exportovaciemu porcesu. Distribucia prebieha v sulade s IPFIX Mediator Framework (RFC 6183). Tato trieda je implementovana podla navrhoveho vzoru *Singleton*.

Metódy

public static FlowRecordDispatcher ***getInstance****()*

Implementuje vzor *Singleton*. Vytvori a vrati jedinecnu instanciu v pripade ze neexistuje, v opacnom pripade ju iba vrati.

Návratová hodnota:

Jedinecny objekt typu `FlowRecordDispatcher`.

public synchronized void ***dispatchFlowRecord****(IPFIXFlowRecord flowRecord, String inputProcess)*

Posiela prijate zaznamy o tokoch prislusnym sprostredkovatelskym procesom, alebo exportovaciemu procesu podla konfiguracie. Najprv ziska zoznam prijimatelov toku na zaklade mena povodcu. Ak je zoznam prazdny - zaznam o toku je urceny na export, preto ho zapise do `ExportCache`. Ak zoznam nie je prazdny, ziska si instancie prijimatelov toku a zaznam im zapise do vstupneho buffera. Tato metoda je synchronizovana, lebo je pristupna viacerym vlaknam.

Parametre:

IPFIXFlowRecord flowRecord - zaznam o toku, ktory sa ma distribuovat dalej

String inputProcess - meno povodcu zaznamu o toku.

private void ***fillInputBuffer****(AIntermediateProcess process, IPFIXFlowRecord flowRecord)*

Metoda, ktora zapisuje zaznamy o tokoch do vstupneho buffra sprostredkovatelskych procesov. V pripade neuspechu za zvysi pocitadlo v `DropsCounter` a vypise error.

Parametre:

AIntermediateProcess process - instancia sprostredkovatelskeho procesu.

IPFIXFlowRecord flowRecord - zaznam o toku, ktory sa ma zapisat.

private ArrayList<String> ***getReceiversList****(String inputDevice)*

Ziskava zoznam prijemcov zaznamu o toku od zadaneho povodcu toku.

Parametre:

String inputDevice - povodca zaznamu o toku

Návratová hodnota:

Zoznam prijemcov toku, typ `ArrayList`.

4.1.4 Trieda IPLoader

Trieda je zodpovedna za dynamicke nacitavanie sprostredkovatelskych procesov definovanych v konfiguracnom subore. Implementuje navrhovy vzor *Singleton*.

Metódy

public static IPLoader ***getInstance****()*

Implementuje vzor *Singleton*. Vytvori a vrati jedinecnu instanciu v pripade ze neexistuje, v opacnom pripade ju iba vrati.

public void ***loadProcesses****()*

Hlavna metoda triedy, dynamicky nacitava sprostredkovatelske moduly definovane v konfiguracnom subore. Najprv ziska systemovy *classLoader*. V cykle prechadza zoznam spostredkovatelskych modulov. Kazdy retazec obsahujuci meno prevedie na binarne meno (meno triedy vratane balickov) a pomocou classLoader-a ziska jeho `Class` objekt. Na zaklade tohto objektu ziska jedinecnu instanciu modulu a kedze sa jedna o vlakno, spusti ho tak, ze zavola jeho metodu `start()`.

Hádže:

`IPLoaderException` - V pripade akejkolvek chyby, ktora moze nastat pri vykonavani metody. Chyby, ktore su zachytavane su typov:

- `SecurityException`
- `ClassNotFoundException`
- `IllegalAccessException`
- `NoSuchMethodException`

- `InvocationTargetException`

4.1.5 Trieda Mediator

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 konfiguracnemu 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 zastaveny po stlaceni kombinacie klaves `Ctrl+c`.

Metódy

public static void ***main****(String[] args)*

Hlavna metoda triedy.

Parametre:

String[] args - argumenty prikazoveho riadku.

public static void ***stopMediator****()*

Metoda, ktora korektne ukoncuje beh programu. Zastavi vsetky spustene vlakna a uvolni vsetky druhy pamate.

public static void ***interruptThread****()*

Prerusi vykonavanie vlakna.

Parametre:

Thread thread - objekt vlakna, ktore sa ma zastavit.

private static void ***loggingToFile****()*

Metoda, ktora vykonava logovanie do suboru namiesto standardneho vystupu.

4.1.6 Trieda Support

Podporna trieda, ktora obsahuje pomocne metody potrebne pri de(kodovani) a pri validacii formatu dat. Uvadzam iba vlastne metody.

Metódy

public static byte[] ***byteToByteArray****(byte x)*

Konvertuje primitivny typ *byte* na pole bytov v usporadani bytov Big Endian.

Parametre:

byte x - hodnota, ktora sa ma zakodovat.

Návratová hodnota:

Pole bytov, typ *byte[]*.

public static byte[] ***shortToByteArray****(short x)*

Konvertuje primitivny typ *short* na pole bytov v usporadani bytov Big Endian.

Parametre:

short x - hodnota, ktora sa ma zakodovat.

Návratová hodnota:

Pole bytov, typ *byte[]*.

public static byte[] ***intToByteArray****(int x)*

Konvertuje primitivny typ *int* na pole bytov v usporadani bytov Big Endian.

Parametre:

int x - hodnota, ktora sa ma zakodovat.

Návratová hodnota:

Pole bytov, typ *byte[]*.

public static byte[] ***longToByteArray****(long x)*

Konvertuje primitivny typ *long* na pole bytov v usporadani bytov Big Endian.

Parametre:

long x - hodnota, ktora sa ma zakodovat.

Návratová hodnota:

Pole bytov, typ *byte[]*.

public static byte[] ***floatToByteArray****(float x)*

Konvertuje primitivny typ *float* na pole bytov v usporadani bytov Big Endian.

Parametre:

float x - hodnota, ktora sa ma zakodovat.

Návratová hodnota:

Pole bytov, typ *byte[]*.

public static byte[] ***doubleToByteArray****(double x)*

Konvertuje primitivny typ *double* na pole bytov v usporadani bytov Big Endian.

Parametre:

double x - hodnota, ktora sa ma zakodovat.

Návratová hodnota:

Pole bytov, typ *byte[]*.

public static boolean ***validateMAC****(String macAddress)*

Validuje format MAC adresy.

Parametre:

String macAddress - adresa, ktorej format sa ma overit.

Návratová hodnota:

true - v pripade, ze je adresa v spravnom formate.

false - opacne.

4.2 Balík sk.tuke.cnl.bm.Mediator.IPFIX

4.2.1 Trieda IPFIXEncoder

Trieda so statickými metódami slúžiacimi na zakódovanie retazcovej reprezentacie hodnot informacnych elementov na abstraktne datove typy podla RFC 5101 a RFC 5102. Je presnym opakom triedy `IPFIXDecoder`.

Metódy

public static byte[] ***encode****(String dataType, String value)*

Zakóduje hodnotu datoveho typu do pola bytov podľa špecifikácie IPFIX. Priamo nevykonáva zakódovanie, volá konkrétne metódy podľa kategórie dátového typu.

Parametre:

String dataType - reťazec definujúci dátový typ obsiahnutý v buffri.

String value - samotné dáta, ktoré sú predmetom zakódovania

Návratová hodnota:

Pole bytov reprezentujúce interpretovanú retazcovu hodnotu na základ predaného typu.

Hádže:

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`OutOfBoundsException` - Ak je hodnota mimo povoleneho rozsahu

`UnknownHostException` - Ak program nevie rozpoznat host, alebo IP adresu

`DataException` - Ak je chyba v formate kodovanej hodnoty vzhladom na datovy typ

public static byte[] ***encodeUnsignedIntegralType****(String dataType, String value)*

Zakóduje celočíselné bezznamiekové dátové typy unsigned8, unsigned16, unsigned32, unsigned64 a unsigned128.

Parametre:

String dataType - reťazec definujúci dátový typ obsiahnutý v buffri.

String value - samotné dáta, ktoré sú predmetom zakódovania

Návratová hodnota:

Pole bytov reprezentujúce interpretovanú retazcovu hodnotu na základ predaného typu.

Hádže:

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formate kodovanej hodnoty vzhladom na datovy typ

`OutOfBoundsException` - Ak je hodnota mimo povoleneho rozsahu

public static byte[] ***encodeSignedIntegralType****(String dataType, String value)*

Zakóduje celočíselné znamienkové dátové typy signed8, signed16, signed32 a signed64.

Parametre:

String dataType - reťazec definujúci dátový typ obsiahnutý v buffri.

String value - samotné dáta, ktoré sú predmetom zakódovania

Návratová hodnota:

Pole bytov reprezentujúce interpretovanú retazcovu hodnotu na základ predaného typu.

Hádže:

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formate kodovanej hodnoty vzhladom na datovy typ

public static byte[] ***encodeFloatType****(String dataType, String value)*

Zakóduje desatinné dátové typy float32 a float64.

Parametre:

String dataType - reťazec definujúci dátový typ obsiahnutý v buffri.

String value - samotné dáta, ktoré sú predmetom zakódovania

Návratová hodnota:

Pole bytov reprezentujúce interpretovanú retazcovu hodnotu na základ predaného typu.

Hádže:

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formate kodovanej hodnoty vzhladom na datovy typ

public static byte[] ***encodeAddressType****(String dataType, String value)*

Zakóduje dátové typy obsahujúce adresy: ipv4Address, ipv6Address a macAddress.

Parametre:

String dataType - reťazec definujúci dátový typ obsiahnutý v buffri.

String value - samotné dáta, ktoré sú predmetom zakódovania

Návratová hodnota:

Pole bytov reprezentujúce interpretovanú retazcovu hodnotu na základ predaného typu.

Hádže:

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`UnknownHostException` - Ak program nevie rozpoznat host, alebo IP adresu

`DataException` - Ak je chyba v formate kodovanej hodnoty vzhladom na datovy typ

public static byte[] ***encodeBooleanType****(String value)*

Zakóduje boolean reprezentujúci pravdivostnú hodnotu.

Návratová hodnota:

Pole bytov reprezentujúci pravdivostnú hodnotu, ”true” alebo ”false”.

Hádže:

`OutOfBoundsException` - Ak je hodnota mimo povoleneho rozsahu

public static byte[] ***encodeStringType****(String value)*

Zakóduje retazec v kódovaní UTF-8 do pole bytov.

Návratová hodnota:

Pole bytov retazca v kódovaní UTF-8.

public static byte[] ***encodeOctetArrayType****(String value)*

Reťazec v kódu Base64 prevedie na pole bytov.

Návratová hodnota:

Pole bytov predstavujúce binárne dáta zakódované v Base64.

public static byte[] ***encodeDateTimeType****(String dataType, String value)*

Zakóduje dátové typy časových známok: dateTimeSeconds, dateTimeMilliseconds, dateTimeMicroseconds a dateTimeNanoseconds do pola bytov.

Parametre:

String dataType - reťazec definujúci dátový typ obsiahnutý v buffri.

String value - samotné dáta, ktoré sú predmetom zakódovania

Návratová hodnota:

Pole bytov reprezentujúce interpretovanú hodnotu retazca na základe predaného typu. Dátové typy dateTimeSeconds a dateTimeMilliseconds predstavujú počet sekúnd, resp. milisekúnd od Unix epochy (00:00 1.1.1970 UTC). Dátové typy dateTimeMicroseconds a dateTimeNanoseconds sú zakódované vo formáte časovej známky NTP Timestamp.

Hádže:

`UnsupportedDataException` - Ak dátový typ nie je podporovaný

`NumberFormatException` - Ak je chyba v formate kodovanej hodnoty vzhladom na datovy typ

`OutOfBoundsException` - Ak je hodnota mimo povoleneho rozsahu

public static void ***checkStringNumbersRange****(String min, String max, String value, String dataType)*

Overi, ci ciselna hodnota v retazci spada do rozsahu daneho datovym typom.

Parametre:

String min - dolna hranica rozsahu

String max - horna hranica rozsahu

String value - hodnota, ktora sa ma overit

String dataType - datovy typ hodnoty

Hádže:

`OutOfBoundsException` - Ak je hodnota mimo povoleneho rozsahu

4.2.2 Trieda IPFIXFlowRecord

Tato trieda je reprezentaciou IPFIX Flow record-u, teda zaznamu o toku.

Konštruktor

public ***IPFIXFlowRecord****(IPFIXTemplateRecord referencedTemplate, ArrayList <IPFIXDataRecord> dataRecords, IPFIXMessage.IPFIXMessageHeader messageHeader)*

Konstruktor prostrednictvom predanych parametrov inicializuje clenske premenne.

Parametre:

IPFIXTemplateRecord referencedTemplate - sablona patriaca datovym zaznamom

ArrayList<IPFIXDataRecord> dataRecords - pole datovych zaznamov

IPFIXMessage.IPFIXMessageHeader messageHeader - hlavicka IPFIX spravy, ktora obsahovala tento zaznam o toku

public ***IPFIXFlowRecord****()*

Bezparametricky konstruktor. Iba inicializuje prazdne pole datovych zaznamov.

Metódy:

Metody, ktore tu nie su spomenute, su klasicke gettery a settery.

public int ***getReferencedTemplateID****()*

Návratová hodnota:

Vracia ID sablony z IPFIX spravy, ktora obsahovala tento zaznam o toku.

public void ***addDataRecord****(IPFIXDataRecord dataRecord)*

Prida datovy zaznam do pola datovych zaznamom flow record-u.

Parametre:

IPFIXDataRecord dataRecord - datovy zaznam, ktory sa priradi zaznamu o toku.

4.2.3 Trieda IPFIXMessageHeader

Trieda reprezentujuca hlavicku IPFIX spravy. Implementuje navrhovy vzor Singleton.

Metody:

Metody, ktore tu nie su spomenute, su klasicke gettery a settery.

public static IPFIXMessageHeader ***getInstance****()*

Návratová hodnota:

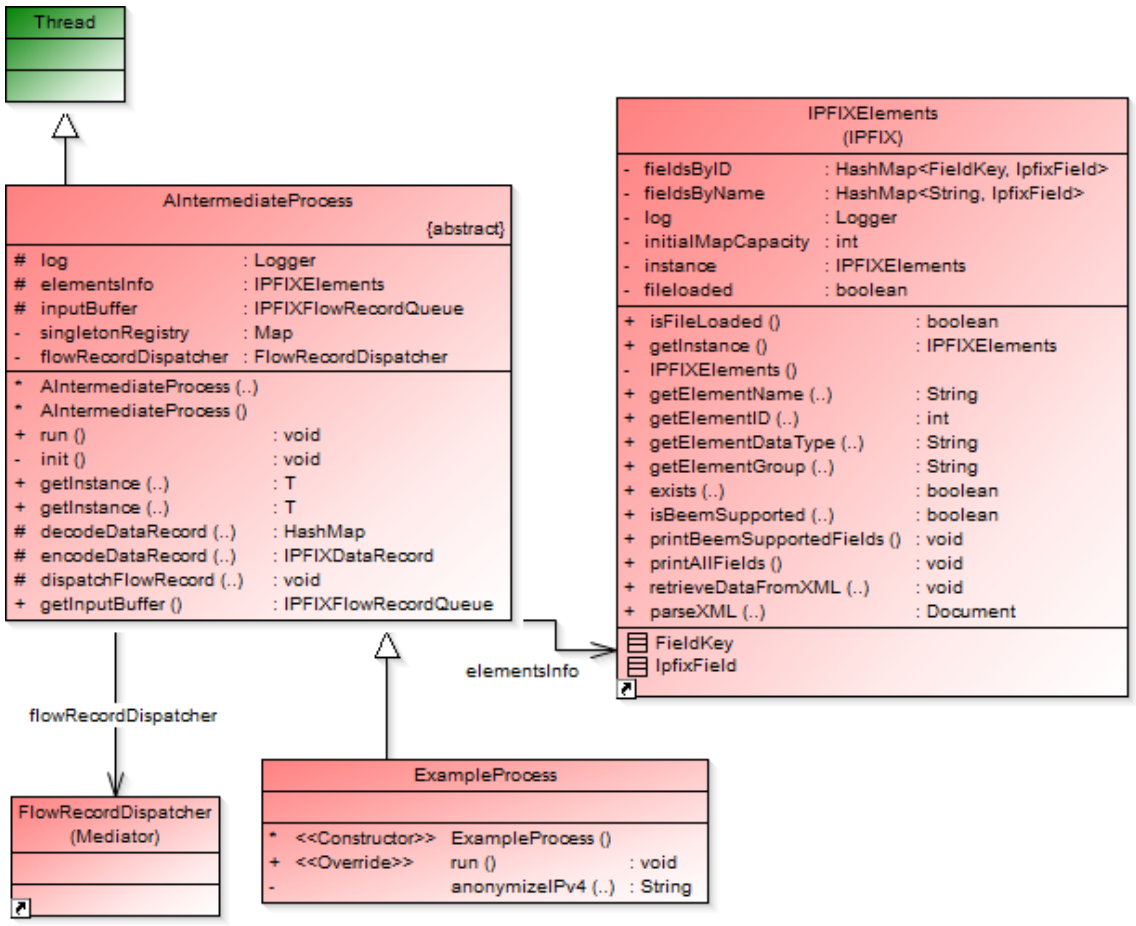
Vracia jedinecnu instanciu objektu.

4.3 Balík sk.tuke.cnl.bm.Mediator.IntermediateProcesses

Diagram tried rozhrania pre sprostredkovateľské procesy, vrátane triedy `ExampleProcess` je znázornený na obrázku 4 – 1.

4.3.1 Trieda AIntermediateProcess

Abstraktna trieda, poskytuje vychodiskove metody sprostredkovatelskym procesom a tvori akesi rozhranie medzi modulmi a aplikacnym ramcom. Dedi od triedy `Thread`.



Obr. 4 – 1 Diagram tried rozhrania pre sprostredkovateľské procesy

Konštruktor

AIntermediateProcess*(String childName)*

Nastavi meno procesu, podla prijateho parametra. Inicializuje vstupny buffer a ziska jedinecnu instanciu triedy `IPFIXElements`.

Parametre:

String childName - meno triedy potomka

AIntermediateProcess*()*

Bezparametricky konstruktor. Inicializuje vstupny buffer a ziska jedinecnu instanciu triedy `IPFIXElements`.

Metody:

public static final synchronized <T extends AIntermediateProcess> T ***getInstance*** *(Class clazz)*

Toto 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)*. 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.

Parametre:

Class clazz = class objekt pozadovanej triedy

Návratová hodnota:

Objekt typu T, pricom T dedi od `AIntermediateProcess`.

public static final synchronized <T extends AIntermediateProcess> T ***getInstance*** *(String processName)*

Aby bolo možné získavať inštancie modulov aj na základe mena triedy a nie len cez class objekty, vytvoril som túto metódu. Premennú *processName* prevedie na binárne meno procesu, podľa špecifikácie jazyka Java, teda nazov triedy vratane balickov,

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.

Parametre:

String processName = meno pozadovanej triedy

Návratová hodnota:

Objekt typu T, pricom T dedi od `AIntermediateProcess`.

protected final HashMap ***decodeDataRecord****(IPFIXTemplateRecord template, IPFIXDataRecord dataRecord)*

Pri prvom prechode funkciou sa generuje pamatovy zaznam o informacnych elementoch (ie) z XML suboru. Vytiahnu sa informacie o ie, ktore sa nachadzaju v sablone, dekoduju sa ich datove typy a prislusnost k skupine.

Parametre:

IPFIXTemplateRecord template - sablona dat

IPFIXDataRecord dataRecord - datovy zaznam

Návratová hodnota:

Dekodovane data ako objekt typu `HashMap`.

protected final IPFIXDataRecord ***encodeDataRecord****(IPFIXTemplateRecord template, HashMap<String, String> dataMap)*

Zakoduje vsetky hodnoty z hashmapy obsahujucej hodnoty informacnych elementov podla sablony do datoveho zaznamu.

Parametre:

IPFIXTemplateRecord template - sablona dat

HashMap<String, String> dataMap - hodnoty informacnych elementov v hashmape,

ktora sa ma zakodovat

Návratová hodnota:

Vracia objekt datoveho zaznamu - `IPFIXDataRecord`.

Hádže:

`EncodingException` - Ak nastane chyba pri kodovani.

protected final void ***dispatchFlowRecord****(IPFIXFlowRecord flowRecord, String inputProcess)*

Vytvara rozhranie pre pristup k metode frameworku.

Parametre:

IPFIXFlowRecord flowRecord - zaznam o toku, ktory sa ma posunut dalej

String inputProcess - povodca zaznamu o toku

4.3.2 Trieda ExampleProcess

Tato trieda je vzorovym riesenim jednoducheho sprostredkovatelskeho procesu, vykonavajuceho anonymizaciu. Ucelom triedy je pomoc dalsej generacii riesitelov.

Konštruktor

ExampleProcess*()*

Vola rodicovsky konstruktor a predava mu svoje meno ako parameter.

Metody:

public void ***run****()*

Hlavna metoda vlakna. 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.

private String ***anonymizeIPv4****(String address)*

Metoda na velmi jednoduchu anonymizaciu IP adresy, cislo v poslednom oktete zmeni na 0.

Parametre:

String address - IP adresa, ktora sa anonymizovat

Návratová hodnota:

Anonymizovana IP adresa, vratena ako retazec.

4.3.3 Trieda IPInputBuffer

Reprezentuje vstupny buffer sprostredkovatelskych modulov. Tato trieda je cache pre zaznamy o tokoch. Jej pouzitie je kriticke vo vysokorychlostnych sietach, pretoze udrziava elementy a tym padom moze vyrovnavat narazovy napor. Je synchronizovana a jej implementacia je FIFO front typu `ArrayBlockingQueue`.

Metody:

public boolean ***write****(IPFIXFlowRecord flowRecord)*

Zapisuje zaznamy o tokoch do frontu. Ak je front plny, zaznam sa zahadzuje.

Parametre:

IPFIXFlowRecord flowRecord - zaznam o toku

Návratová hodnota:

Pravdivostna hodnota podla toho, ci zaznam bol, alebo nebol zapisany do cache.

public void ***write****(IPFIXTemplateRecord template, ArrayList<IPFIXDataRecord> dataRecords, IPFIXMessage.IPFIXMessageHeader messageHeader)*

Metoda obali prijate parametre do objektu tiredy `IPFIXFlowRecord` a zavola predchadzajucu metodu.

Parametre:

IPFIXTemplateRecord template - sablona

ArrayList<IPFIXDataRecord> dataRecords - pole datovych zaznamov

IPFIXMessage.IPFIXMessageHeader messageHeader - hlavicka IPFIX spravy, z ktorej tento zaznam o toku pochadza

public IPFIXFlowRecord ***read****()*

Precita a zmaze vrchol frontu. Ak je front prazdny caka dokial sa tam nejaky element neprida.

Návratová hodnota:

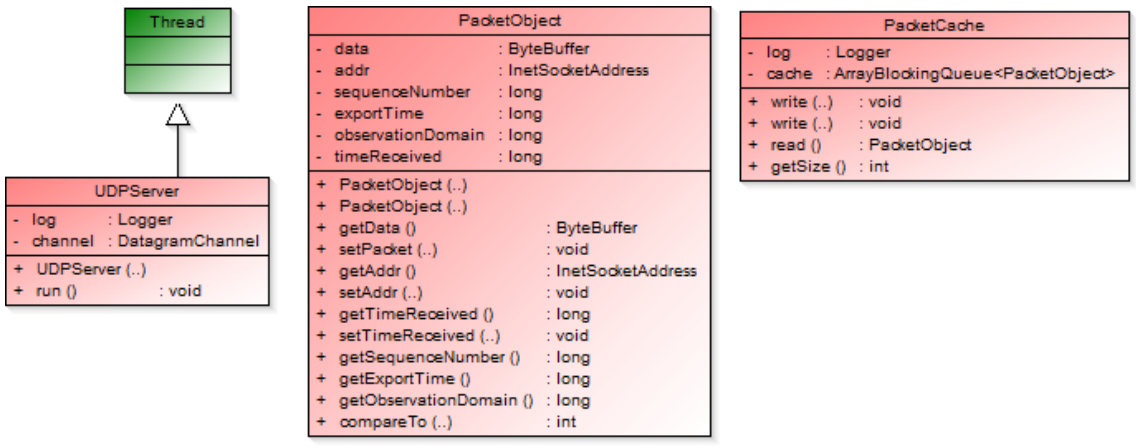
Objekt triedy `IPFIXFlowRecord`.

Hádže:

`InterruptedException` - Ak nastala chyba pri synchronizovani vlakien, alebo ak bolo vlakno prerusene pocas cakania.

4.4 Balik sk.tuke.cnl.bm.Mediator.collecting

Zjednodušený diagram tried tohto balika môžeme vidieť na Obr. 4 – 2 a na Obr. 4 – 3.



Obr. 4 – 2 Diagram tried prvej fázy zhromažďovacieho procesu

4.4.1 Trieda UDPServer

Slúži ako UDP server. Prijíma UDP datagramy cez `DatagramChannel` a uklada ich do `PacketCache`.

Konštruktor

public ***UDPServer****(int port)*

Konštruktor inicializuje `DatagramChannel`, nastaví mu blokovací režim a priviaže ho k portu definovanom v konfiguračnom súbore, ktory mu je predany ako parameter. Nastaví meno vlákna.

Parametre:

int port - cislo portu

Metódy

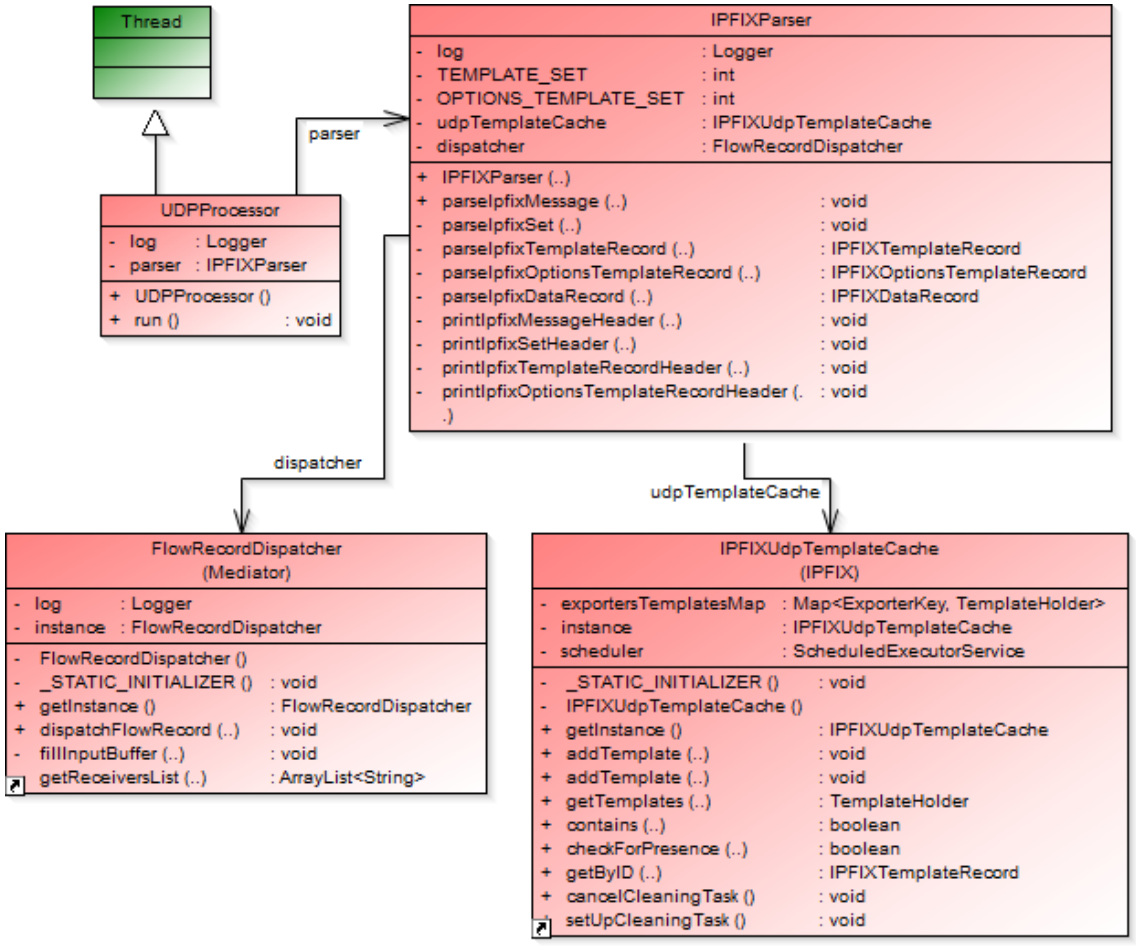
public void ***run****()*

Hlavná metóda vlákna. Pokiaľ nedôjde k prerušeniu, prijima cez vytvoreny kanal data od exportéra. Prijate data obali do objektu `ByteBuffer` a predá ich spolu s casom prijatia a IP adresou a portom exportera metode *write()*, ktora ich zapise do

`PacketCache`.

public void ***cleanUp****()*

Táto metóda zruší čistiace vlákno pre UDP Template Cache. Je volaná pri prerušení tohto vlákna.



Obr. 4 – 3 Diagram tried druhej fázy zhromažďovacieho procesu

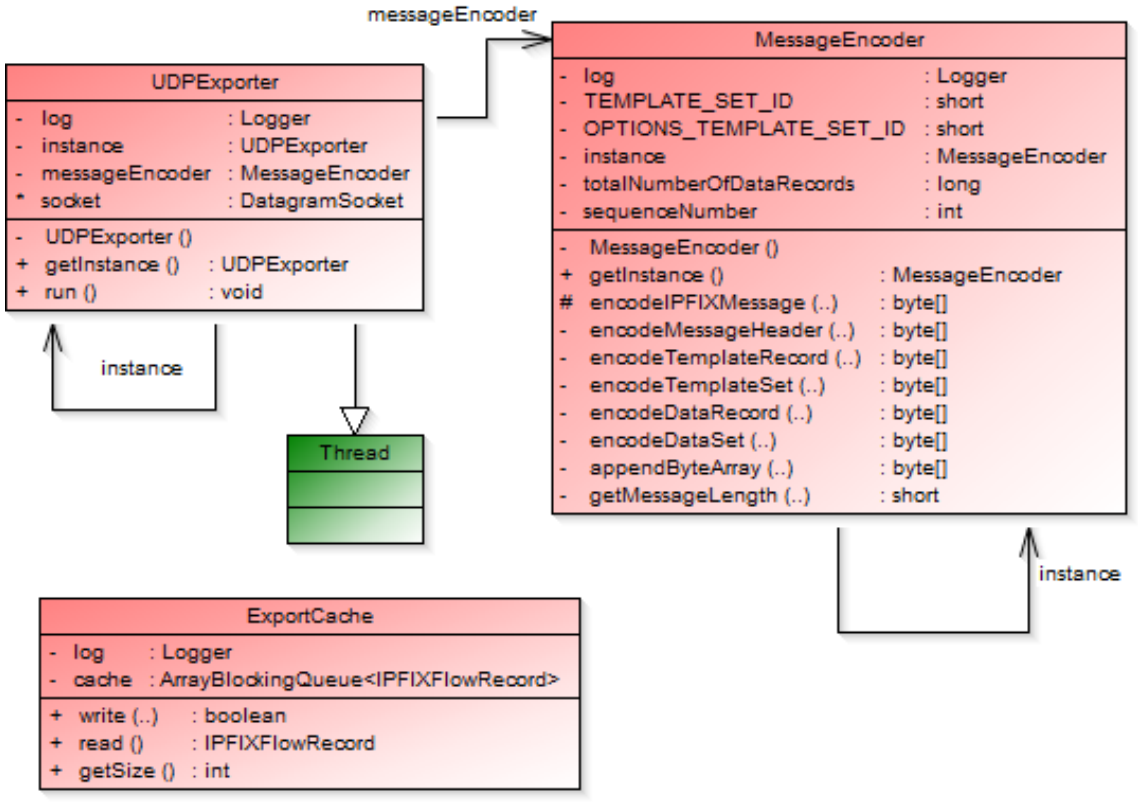
4.4.2 Trieda IpfixParser

Táto trieda sa používa na parsovanie IPFIX správ a ich spracovanie. V porovnani s verziou v aplikacii JXColl bola precistena. Boli vypustene sekcie spracovavajuce TCP a SCTP spojenia. Zasadnejsia zmena prisla na vystupe z triedy. Sparsovane

datove zaznamy su zabalene do vytvoreneho objektu triedy `IPFIXFlowRecord`, spolu s prislusnou sablonou a hlavickou prijatej IPFIX spravy. Vytvoreny zaznam o toku je spolu s retazcom predstavujucim zdroj zaznamu (v tomto pripade „exporter“) posunuty triede `FlowRecordDispatcher`.

4.5 Balik sk.tuke.cnl.bm.Mediator.exporting

Diagram tried exportovacieho procesu je na Obr. 4 – 4.



Obr. 4 – 4 Diagram tried exportovacieho procesu

4.5.1 Trieda ExportCache

Analogia k triede `IPInputBuffer`. Reprezentuje exportovaciu cache, ktora je synchronizovana a jej implementacia je FIFO front typu `ArrayBlockingQueue`.

Metody:

public static boolean ***write****(IPFIXFlowRecord flowRecord)*

Zapisuje zaznamy o tokoch do frontu. Ak je front plny, zaznam sa zahadzuje.

Parametre:

IPFIXFlowRecord flowRecord - zaznam o toku

Návratová hodnota:

Pravdivostna hodnota podla toho, ci zaznam bol, alebo nebol zapisany do cache.

public static IPFIXFlowRecord ***read****()*

Precita a zmaze vrchol frontu. Ak je front prazdny caka dokial sa tam nejaky element neprida.

Návratová hodnota:

Objekt triedy `IPFIXFlowRecord`.

Hádže:

`InterruptedException` - Ak nastala chyba pri synchronizovani vlakien, alebo ak bolo vlakno prerusene pocas cakania.

public static int ***getSize****()*

Návratová hodnota:

Vracia pocet elementov v cache.

4.5.2 Trieda MessageEncoder

Trieda sluzi na zakodovanie resp. zabalenie zaznamu o toku do IPFIX paketu podla specifikacie v RFC 5101 a RFC 5102.

Metody:

public static MessageEncoder ***getInstance****()*

Návratová hodnota:

Metoda vracia jedinecnu instanciu objektu triedy podla navrhoveho vzoru *Singleton*.

protected byte[] ***createIPFIXMessage****(IPFIXFlowRecord flowRecord)*

Na zaklade zaznamu o toku vytvara prud bytov z IPFIX spravy. Vola jednotlive motody, ktore robia ciastkove ulohy ako zakodovanie sad, hlavicky a podobne.

Parametre:

IPFIXFlowRecord flowRecord - zaznam o toku

Návratová hodnota:

Pole bytov prudu IPFIX spravy.

private byte[] ***encodeMessageHeader****(IPFIXMessage.IPFIXMessageHeader header, short length)*

Zakoduje hlavicku IPFIX spravy.

Parametre:

IPFIXMessage.IPFIXMessageHeader header - hlavicka, ktora sa ma zakodovat

short length - celkova dlzka IPFIX spravy

Návratová hodnota:

Pole bytov prudu hlavicky IPFIX spravy.

private byte[] ***encodeTemplateRecord****(IPFIXTemplateRecord templateRecord)*

Zakoduje zaznam sablony.

Parametre:

IPFIXTemplateRecord templateRecord - zaznam sablony, ktory sa ma zakodovat

Návratová hodnota:

Pole bytov prudu zaznamu sablony.

private byte[] ***encodeTemplateSet****(byte[] templateRecordBytes)*

Zakoduje sadu sablon.

Parametre:

byte[] templateRecordBytes - pole bytov prudu zaznamov sablony, ktore obsahuje sada sablony

Návratová hodnota:

Pole bytov prudu sady sablon.

private byte[] ***encodeDataRecord****(IPFIXDataRecord dataRecord)*

Zakoduje datovy zaznam.

Parametre:

IPFIXDataRecord dataRecord - datovy zaznam, ktory sa ma zakodovat

Návratová hodnota:

Pole bytov prudu datoveho zaznamu.

private byte[] ***encodeDataSet****(ByteArrayOutputStream dataRecordsStream, int templateID)*

Zakoduje datovu sadu.

Parametre:

ByteArrayOutputStream dataRecordsStream - prud bytov datovych zaznamov, ktore obsahuje sada sablony

int templateID - ID prisluchajucej sablony

Návratová hodnota:

Pole bytov prudu datovej sady.

private byte[] ***appendByteArray****(byte[] first, byte[] second)*

Pomocna metoda, ktora na koniec prveho pola bytov pripoji druhe pole bytov .

Parametre:

byte[] first - prve pole bytov

byte[] second - druhe pole bytov

Návratová hodnota:

Pole bytov vysledneho pola.

private short ***getMessageLength****(byte[] templateSetBytes, byte[] dataSetBytes, byte[] optionsTemplateSetStream)*

Vracia celkovu dlzku IPFIX spravy.

Parametre:

byte[] templateSetBytes - pole bytov sady sablon

byte[] dataSetBytes - pole bytov datovej sady

byte[] optionsTemplateSetStream - pole bytov sady sablon moznosti

Návratová hodnota:

Celkova dlzka IPFIX spravy, vratane hlavicky. Navratovy typ je *short*.

4.5.3 Trieda UDPExporter

Trieda je samostatnym vlaknom, zaznamy o toku cita z ExportCache. Tie potom posle triede `MessageEncoder`, ktora ich zabali do IPFIX sprav. Zakodovane spravy exportuje kolektoru na IP adresu a port, ktore su definovane v konfiguracnom subore.

Konštruktor:

private ***UDPExporter****()*

Bezparametricky konstruktor. Vola rodicovsky konstruktor s parametrom svojho

mena. Potom sa pokusi vytvorit socket pomocou triedy `DatagramSocket`.

Metódy:

public static MessageEncoder ***getInstance****()*

Návratová hodnota:

Metoda vracia jedinecnu instanciu objektu triedy podla navrhoveho vzoru *Singleton*.

public void ***run****()*

Hlavna metoda vlakna. Dokial nie je vlakno prerusene, cyklicky cita zaznamy o toku z ExportCache. Prostrednictvom treidy `MessageEncoder` vytvori zo zaznamu o toku vystupny prud bytov. Ten spolu s dlzkou prudu, IP adresou a portom zabali do paketu - objektu triedy `DatagramPacket`. Vzniknuty paket odosle.

5 Preklad programu

5.1 Zoznam zdrojových textov

Zdrojové texty sú k dispozícii v prílohe diplomovej práce.

Sú k dispozícii tieto zdrojové texty:

- balík sk.tuke.cnl.bm.Mediator:
  - Config.java
  - Default.java
  - DropsCounter.java
  - FlowRecordDispatcher.java
  - IPLoader.java
  - Mediator.java
  - Support.java
- balík sk.tuke.cnl.bm.Mediator.IPFIX:
  - ExporterKey.java
  - FieldSpecifier.java
  - IPFIXDataRecord.java
  - IPFIXDecoder.java
  - IPFIXElements.java
  - IPFIXEncoder.java
  - IPFIXFlowRecord.java
  - IPFIXMessage.java
  - IPFIXOptionsTemplateRecord.java
  - IPFIXSet.java
  - IPFIXTemplateRecord.java
  - IpfixUdpTemplateCache.java
  - TemplateHolder.java
- balík sk.tuke.cnl.bm.Mediator.IntermediateProcesses:
  - AIntermediateProcess.java
  - ExampleProcess.java
  - IPInputBuffer.java
- balík sk.tuke.cnl.bm.Mediator.collecting:
  - IPFIXParser.java
  - PacketCache.java
  - PacketObject.java
  - UDPProcessor.java
  - UDPServer.java
- balík sk.tuke.cnl.bm.Mediator.exporting:
  - ExportCache.java

```
   MessageEncoder.java
   UDPExporter.java
- balík sk.tuke.cnl.bm.exceptions:
   DataException.java
   DataFormatException.java
   EncodingException.java
   IPLoaderException.java
   MediatorException.java
   OutOfBoundsException.java
   TemplateException.java
```

5.2 Požiadavky na technické prostriedky pri preklade

Preklad programu si vyžaduje nasledovnú hardvérovú konfiguráciu:

- CPU Intel Pentium III 1Ghz alebo ekvivalent
- operačná pamäť 512MB
- pevný disk s 1GB voľného miesta
- sieťová karta 100Mb/s

5.3 Požiadavky na programové prostriedky pri preklade

- operačný systém GNU/Linux s verziou jadra 2.6 a vyššou
- Java Runtime Environment (JRE) verzie 1.7.0_03 a vyššej
- knižnice dodávané na inštalačnom médiu

5.4 Náväznosť na iné programové produkty

Program umožňuje ukladanie dát do databázy alebo ich sprístupnenie priamym pripojením, ktoré budú následne vyhodnotené príslušnými prídavnými modulmi. Je

implementáciou zhromažďovacieho procesu architektúry BasicMeter. Z toho vyplýva jeho náväznosť na merací a exportovací proces - BEEM, alebo iné implementácie.

5.5 Vlastný preklad

Preklad programu spočíva v nakopírovaní zdrojových súborov a spustení kompilátora jazyka Java s potrebnými parametrami a parametrom classpath nastaveným na prídavné knižnice. Odporúča sa použiť váš obľubený java IDE, kde stačí jednoducho nastaviť verziu JDK na 7.0 alebo vyššie a do cesty classpath pridať cesty ku všetkým potrebným knižniciam. V prostredí Netbeans IDE potom stačí kliknúť na tlačidlo *Clean and Build*.

5.6 Vytvorenie inštalačného DEB súboru

Stačí spustiť skript `buildDeb.sh`, ktorý sa nachádza v priečinku jxcoll/deb.

```
sh buildDeb.sh
```

Výstupom tohto skriptu je súbor s názvom `debian.deb`, ktorý môžme následne premenovať podľa verzie JXColl (napríklad na jxcoll_3.9_i386.deb). Tento skript vykonáva nasledovné:

1. v prípade, ak neexistuje priečinok debian, extrahuje ho z archívu debian.tar.gz, inak tento krok preskočí
2. v prípade, ak neexistuje priečinok debian, extrahuje ho z archívu debian.tar.gz, inak tento krok preskočí
3. skopíruje binárny súbor z projektu do DEB balíčka (predpokladá sa, že bol program kompilovaný v Netbeans IDE pomocou Clean and Build tlačidla)
4. skopíruje konfiguračný súbor z projektu do DEB balíčka

5. skopíruje IPFIX definičný súbor z projektu do DEB balíčka
6. vymaže prípadné dočasné súbory z DEB balíčka
7. vygeneruje MD5 kontrolné súčty pre všetky súbory DEB balíčka
8. zabezpečí maximálnu kompresiu manuálových stránok a changelog súborov
9. skopíruje binárny súbor z projektu do DEB balíčka a nastaví mu práva na vykonávanie
10. vytvorí samotný DEB balíček
11. overí ho pomocou programu lintian - ten vypíše prípadne varovania a/alebo chyby
12. archivuje vytvorený DEB balíček do archívu debian.tar.gz

Pred spustením skriptu je nutné skompilovať JXColl pomocou Netbeans IDE tlačidlom *Clean and Build*. Prípadné zmeny control alebo changelog súboru, manuálových stránok je nutné vykonať ručne. Manuálové stránky je vhodné upraviť pomocou programu *GmanEdit* . Po spustení skriptu sa vytvorí DEB balíček s názvom `debian.deb`. Ten je vhodné premenovať podľa aktuálnej verzie. Vytvorí sa aj archív `debian.tar.gz`, ktorý obsahuje najaktuálnejšiu adresárovú štruktúru DEB balíčka pre budúce využitie (ak neexistuje priečinok debian, vytvorí sa extrakciou z tohto archívu). Ak je potrebné len aktualizovať kód, stačí spustiť skript a ten sa o všetko postará, pričom vytvorí aj adresár debian. Súbory možno v ňom upravovať až kým nie je všetko podľa predstáv. Ak je všetko hotové, v Netbeans IDE je potrebné vymazať priečinok debian (vykoná sa SVN DELETE, namiesto obyčajného odstránenia zo súborového systému) a projekt "commitnúť".

5.7 Opis známych chýb

V súčasnosti nie sú známe žiadne vážne chyby.

6 Zhodnotenie riešenia

Hlavným cieľom práce bolo zvýšiť interoperabilitu s inými IPFIX riešeniami pomocou zvýšenia konformity so štandardom IPFIX. V práci boli vyriešené problémy, ktoré doteraz znemožňovali dekódovanie viacerých záznamov sade, informačncých elementov s variabilnou dĺžkou, informačných elementov s redukovaným kódovaním alebo niektorých predtým neimplementovaných dátových typov.

Súčasťou práce bolo rozšírenie podpory prenosu údajov o tokoch prostredníctvom transportných protokolov TCP a SCTP, čo zvyšuje možnosti nasadenia nástroja BasicMeter aj v podmienkach s vyššou náchylnosťou na preťaženie v sieti.

Možnosti budúceho vývoja zhromažďovacieho procesu nástroja BasicMeter predstavuje implementácia podpory pre dátové typy umožňujúce prenos štruktúrovaných dát a podpora pre zabezpečené pripojenia od exportérov.

Literatúra

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

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

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

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

[5] VEREŠČÁK, T.: Optimalizácia zhromažďovacieho procesu nástroja BasicMeter, 2012, Diplomová práca, Príloha A, Systémová príručka JXColl v3.9, KPI FEI TU, Košice