Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Optimalizácia zhromažďovacieho procesu nástroja BasicMeter

Diplomová práca

2012 **Bc. Tomáš Vereščák**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Optimalizácia zhromažďovacieho procesu nástroja BasicMeter

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Juraj Giertl, PhD.
Konzultant:	Ing. Martin Révés, PhD.

Košice 2012 **Bc. Tomáš Vereščák**

Abstrakt v SJ

Táto diplomová práca sa zaoberá optimalizáciou zhromažďovacieho procesu nástroja BasicMeter v oblasti interoperability s inými IPFIX riešeniami na základe zvýšenia jeho konformity so špecifikáciou IPFIX. Práca v sebe zahŕňa aj analýzu, návrh a implementáciu spracovania údajov o tokoch prostredníctvom transportných protokolov TCP a SCTP. Záver práce sa venuje overeniu funkčnosti implementovaných riešení.

Kľúčové slová

Optimalizácia, zhromažďovací proces, SCTP, TCP, konformita, IPFIX, BasicMeter, interoperabilita

Abstrakt v AJ

This diploma thesis deals with the optimization of BasicMeter collecting process in relation to interoperability with other IPFIX solutions on the basis of enhancing conformity with IPFIX specification. The thesis also contains analysis, design and implementation of flow data processing over transport protocols TCP and SCTP. Final part of the thesis is devoted to the verification of functionality of implemented solutions.

Kľúčové slová v AJ

Optimization, collecting process, SCTP, TCP, conformity, IPFIX, BasicMeter, interoperability

TECHNICKÁ UNIVERZITA V KOŠICIACH

Fakulta elektrotechniky a informatiky

Katedra počítačov a informatiky

ZADANIE
DIPLOMOVEJ PRÁCE

Študijný odbor: **9.2.1 Informatika**

Študijný program: **Informatika**

Názov práce:

Optimalizácia zhromažďovacieho procesu nástroja BasicMeter
Optimization of the BasicMeter Tool Collecting Process

Študent (tituly, meno, priezvisko): **Bc. Tomáš Vereščák**

Školiteľ (tituly, meno, priezvisko):	**Ing. Juraj Giertl, PhD.**
Školiace pracovisko:	**Katedra počítačov a informatiky**
Konzultant práce (tituly, meno, priezvisko):	**Ing. Martin Révés, PhD.**
Pracovisko konzultanta:	**Katedra počítačov a informatiky**

Pokyny na vypracovanie diplomovej práce:

1. Analyzovať súčasné riešenie zhromažďovacieho procesu nástroja BasicMeter z hľadiska podpory transportných protokolov.
2. Navrhnúť a implementovať podporu transportných protokolov TCP a SCTP v zhromažďovacom procese nástroja BasicMeter.
3. Optimalizovať zhromažďovací proces nástroja BasicMeter s ohľadom na konformitu so špecifikáciou IPFIX.
4. Overiť funkčnosť vykonaných zmien.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje:	slovenský
Termín pre odovzdanie práce:	27.04.2012
Dátum zadania diplomovej práce:	31.10.2011

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

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY 12

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

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

Košice 27. 4. 2012 .....

Vlastnoručný podpis

Poďakovanie

Chcem sa poďakovať vedúcemu tejto diplomovej práce Ing. Jurajovi Giertlovi, PhD. a konzultantovi Ing. Martinovi Révésovi, PhD. za ich cenné rady a pripomienky týkajúce sa tejto práce. Ďakujem tiež svojim rodičom, ktorí mi umožnili študovať a nesmierne ma podporovali počas celého štúdia.

Predhovor

Počítačové siete sú v súčasnosti neoddeliteľnou súčasťou života a vďaka nim je práca omnoho efektívnejšia ako v minulosti. S vývojom sieťových aplikácií sa zvyšujú aj nároky na kvalitu poskytovaných služieb. Dnešné konvergované siete spájajú dátové, a telekomunikačné služby do spoločnej infraštruktúry.

Problémy súvisiace so zabezpečením kvality služieb sa snaží riešiť nástroj BasicMeter, ktorý je vyvíjaný ako projekt výskumnej skupiny MONICA v rámci Laboratória počítačových sietí pri Technickej univerzite v Košiciach. Projekt sa zaoberá zberom a vyhodnocovaním prevádzkových parametrov počítačových sietí.

Cieľom tejto práce je rozšíriť zhromažďovací proces nástroja BasicMeter o podporu transportných protokolov TCP a SCTP. Týmto rozšírením sa zvýšia možnosti nasadenia tohto nástroja aj v prostredí náchylnom na preťaženie. Ďalšou úlohou je optimalizovať zhromažďovací proces implementovaním doposiaľ nepodporovaných funkcií, ktoré mu zabraňujú korektne spracovávať údaje od iných riešení, a zvýšiť tak konformitu so špecifikáciou IPFIX.

Tému *Optimalizácia zhromažďovacieho procesu nástroja BasicMeter* som si vybral, pretože ma okrem programovania zaujímajú vo veľkej miere aj počítačové siete a zabezpečovanie kvality služieb. Na tomto projekte som dostal možnosť rozvíjať svoje záujmy a k výberu ma tiež viedla viacročná skúsenosť s projektom BasicMeter a snaha o vylepšenie zhromažďovaciho procesu tohto nástroja.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam symbolov a skratiek

API	Application Programming Interface
ECAM	Exporter Collector Analyzer Manager
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information eXport
JDK	Java Development Kit
JXColl	Java XML Collector
LKSCTP	SCTP for the Linux Kernel
MONICA	Monitoring and Optimization of Network Infrastructures Communications and Applications
MTU	Maximum Transmission Unit
SCTP	Stream Control Transmission Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
Vermont	VERsatile MONitoring Toolkit
XML	eXtensible Markup Language
YAF	Yet Another Flowmeter

Slovník termínov

Pozorovací bod je miesto v sieti, kde môžu byť sledované IP pakety. Zvyčajne je to centrálny uzol v sieti, napríklad smerovač, prepínač, alebo externá sonda pripojená k takémuto bodu.

Merací proces vytvára záznamy tokov. Vstupom tohto procesu sú hlavičky paketov sledovaných na pozorovacom bode. Pozostáva z množiny funkcií, ktoré zahŕňajú zachytávanie hlavičiek paketov, vytváranie časových známok, vzorkovanie, klasifikácia a údržba záznamov o tokoch.

Exportovací proces posiela záznamy o tokoch na jeden alebo viac zhromažďovacích procesov. Záznamy tokov sú generované jedným alebo viacerými meracími procesmi.

Zhromažďovací proces prijíma záznamy tokov z jedného alebo viacerých exportovacích procesov. Zhromažďovací proces môže spracovávať alebo ukladať prijaté záznamy tokov.

BasicMeter predstavuje merací nástroj vytvorený na základe špecifikácie IPFIX, na účely merania prevádzkových parametrov počítačovej siete.

JXColl je komponentom nástroja BasicMeter, ktorý slúži na zachytávanie a predspracovávanie informácií o tokoch v sieťach získané od exportérov. Umožňuje ukladať prijaté informácie o tokoch do databázy pre neskoršiu analýzu, prípadne ich sprístupniť v reálnom čase analyzujúcej aplikácii pomocou protokolu ACP. Predstavuje implementáciu zhromažďovacieho procesu.

Parser nástroj analyzujúci vstupné dáta, rozdeľujúc ich na jednotlivé významové časti, ktoré môžu byť následne spracované.

Singleton je návrhový vzor umožňujúci vytvoriť najviac jednu inštanciu triedy.

Úvod

Dnešné mimoriadne vysoké nároky na kvalitu poskytovaných služieb v oblasti počítačových sietí so sebou prinášajú rôzne problémy. S narastajúcimi požiadavkami na prenos zvukových a obrazových dát spoločne s ostatnými dátami sa zvyšujú aj nároky na hardvérové a softvérové vybavenie počítačovej siete a koncových zariadení. Pri zabezpečovaní funkčnosti počítačovej siete organizácie je potrebné reagovať na neustále sa zvyšujúce nároky a priebežne optimalizovať hardvérové a softvérové vybavenie, či zmeniť pravidlá smerovania a prepínania.

Sledovanie rôznych prevádzkových parametrov počítačových sietí by malo byť v každej organizácii samozrejmosťou. Pre potreby zistenia krokov nutných na skvalitnenie počítačovej siete, je nutné sledovať nielen typ a objem prevádzky vyskytujúci sa v sieti, ale aj prevažujúce aplikácie a používateľov. Po vynaložení úsilia na skvalitnenie služieb je znovu potrebné vykonávať merania nakoľko sa daný problém investíciou vyriešil. Je potrené overiť, či sieť zodpovedá kritériám definovaným v zmluve o dohodnutej úrovni poskytovania služieb. Sieťoví operátori navyše potrebujú mať detailný prehľad o sieťovej prevádzke z bezpečnostných dôvodov.

Merať je možné zaznamenávaním jednotlivých paketov, prechádzajúcich centrálnymi prvkami počítačovej siete (najmä smerovačmi a prepínačmi). Tento spôsob však predstavuje veľmi vysoké nároky na tieto zariadenia. Efektívnejším spôsobom je zlúčiť pakety s niektorými spoločnými vlastnosťami do sieťových tokov a umožniť tým prenos menšieho množstva dát.

Protokol IPFIX (IP Flow Information eXport) predstavuje prokotol umožňujúci export informácií o sieťovej prevádzke v podobe sieťových tokov. V rámci výskumnej skupiny MONICA pôsobiacej v Laboratóriu počítačových sietí pri Technickej univerzite v Košiciach je vyvíjaný nástroj Basicmeter. Tento nástroj je snahou o implementáciu pasívneho merania sieťových tokov podľa protokolu IPFIX.

V apríli 2011 sa v Prahe uskutočnila udalosť DEMONS IPFIX Interoperability Testing Event, ktorá poukázala na niektoré nedostatky tohto nástroja v porovnaní s inými existujúcimi implementáciami. Prejavili sa nedostatky v oblasti dekódovania dátových typov, skrátených, variabilných alebo organizáciou definovaných informačných elementov. Možno spomenúť tiež neexistujúci mechanizmus expiráciu šablón alebo chýbajúcu podporu pre viacero záznamov v sade.

Podstatným nedostatkom je napríklad aj neschopnosť zhromažďovacieho nástroja JXColl prijímať informácie o sieťových tokoch pomocou protokolu SCTP a TCP.

Cieľom tejto práce je implementovať podporu transportných protokolov TCP a SCTP a taktiež vyriešiť problémy, ktoré boli zistené na spomínanej udalosti. Bližšie informácie o nich poskytnú príslušné kapitoly tejto práce.

V úvodnej kapitole je formulovaná úloha práce. Druhá kapitola sa zaoberá prehľadom protokolu IPFIX a nástroja BasicMeter. Obsahom tretej kapitoly je stručná analýza transportných protokolov UDP, TCP a SCTP. V štvrtej kapitole sa čitateľ dozvie o súčasnom stave a hlavných nedostatkoch zhromažďovacieho procesu nástroja BasicMeter. V piatej kapitole je uvedený návrh a implementácia podpory transportných protokolov TCP a SCTP v nástroji JXColl a tiež návrh a implementácia zmien zvyšujúcich konformitu nástroja so špecifikáciou IPFIX. Šiesta kapitola je venovaná overeniu funkčnosti implementovaných zmien. V závere je zhodnotený prínos tejto práce.

1 Formulácia úlohy

Úlohou tejto práce je rozšíriť zhromažďovací proces nástroja BasicMeter o podporu príjmu údajov o sieťových tokoch prostredníctvom transportných protokolov TCP a SCTP. V súvislosti s touto úlohou je nutné uviesť základné informácie o týchto transportných protokoloch, ich súčasnú podporu v zhromažďovacom procese nástroja BasicMeter a požiadavky pre ich realizáciu. Je potrebné navrhnúť a implementovať podporu týchto transportných protokolov v zhromažďovacom procese.

Podľa zistení na vzájomnom testovaní niektorých IPFIX implementácií v Prahe na udalosti DEMONS IPFIX Interoperability Testing event, zhromažďovací proces nástroja BasicMeter má nedostatky v dekódovaní časových známok, informačných elementov s redukovaným kódovaním, informačných elementov s variabilnou dĺžkou a informačných elementov vytvorených organizáciou. Nedostatky možno nájsť aj v systéme správy šablón, či v dekódovaní viacerých záznamov v sade. Je potrebné analyzovať tieto problémy a v zhromažďovacom procese nástroja BasicMeter, navrhnúť a implementovať zmeny riešiace tieto problémy a prispieť tak k zvýšeniu jeho konformity so špecifikáciou IPFIX.

Súčasťou práce má byť overenie funkčnosti prenosu údajov o IP tokoch prostredníctvom protokolov TCP a SCTP spoločne s overením funkčnosti zmien v zhromažďovacom procese zvyšujúce konformitu so špecifikáciou IPFIX.

2 Analýza protokolu IPFIX

Protokol IPFIX [1, 2, 3, 4] je protokol vyvíjaný rovnomennou pracovnou skupinou v rámci IETF (Internet Engineering Task Force). Štandardizovaný protokol IPFIX slúži na export informácií o sieťových tokoch. Je navrhnutý ako rozšírenie protokolu Netflow verzia 9, ktorý bol vyvinutý spoločnosťou Cisco.

Tok predstavuje súbor sledovaných IP paketov, ktoré prechádzali pozorovacím bodom v sieti. Všetky tieto pakety zdieľajú niekoľko spoločných atribútov, ktoré vytvárajú kľúč toku. Najčastejšie sa ako kľúč toku používa zdrojová a cieľová IP adresa, transportný protokol, zdrojový a cieľový port transportného protokolu. Okrem kľúčových atribútov, záznam toku obsahuje aj iné informácie. Najčastejšie sa vyskytujúce sú celkový počet prenesených paketov a bajtov, spoločne s časmi, kedy bol zachytený prvý alebo posledný paket.

Odosielateľ dát o tokoch sa nazýva exportér. Kolektor predstavuje prijímateľa dát o tokoch v sieti. Protokol IPFIX je flexibilný z toho pohľadu, že je možné nakonfigurovať záznam toku podľa potreby.

Protokol IPFIX používa počas exportu dva rozdielne druhy dát:

- metadáta, ktoré definujú sémantiku a kódovanie záznamu toku,
- samotný záznam toku.

Metadáta sú v pojmoch IPFIX zvané šablóny. Šablóna je usporiadaná množina špecifikátorov polí. Špecifikátor poľa obsahuje informácie o type a dĺžke informačného elementu. Šablóna teda určuje štruktúru a význam jednotlivých polí dátového záznamu. Príklad šabóny pozostávajúcej z deviatich polí s celkovou dĺžkou 45 bajtov je uvedený v Tabuľke 2 – 1.

Štanardné IPFIX informačné elementy sú definované v informačnom modeli protokolu IPFIX [5]. Informačný model uvádza informácie o identifikátoroch informačných elementov, ich sémantike, dátovom type, ich príslušnosti do skupín a ďalšie infor-

Tabuľka 2 – 1 Vzorová IPFIX šablóna [3]

Informačný element	**Dĺžka poľa**
sourceIPv4Address (8)	4
destinationIPv4Address (12)	4
protocolIdentifier (4)	1
sourceTransportPort (7)	2
destinationTransportPort (11)	2
flowStartMilliseconds (152)	8
flowEndMilliseconds (153)	8
octetTotalCount (85)	8
packetTotalCount (86)	8
Celková dĺžka	**45**

mácie. Informačný model je verejne prístupný aj vo forme XML dokumentu, aby mohol byť strojovo spracovaný. Na základe tohto dokumentu je možné dekódovať a interpretovať jednotlivé dátové záznamy.

Ak organizácia pre svoje potreby vyžaduje použitie informačného elementu, ktorý sa nenachádza medzi IETF štandardnými informačnými elementami, môže si vytvoriť vlastné. Špecifikátor takéhoto informačného elementu potom obsahuje aj číslo organizácie.

Aby mohol kolektor interpretovať dátový záznam, musí poznať definíciu šablóny, na ktorú referuje. Preto je nevyhnutné, aby ju exportér poslal pred akýmikoľvek dátovými záznamami. Ak kolektor nemá v evidencii šablónu pre prichádzajúce dátové záznamy, musí ich buď zadržiavať alebo zahadzovať. Príklad interpretovaných dát je možné vidieť v Tabuľke 2 – 2.

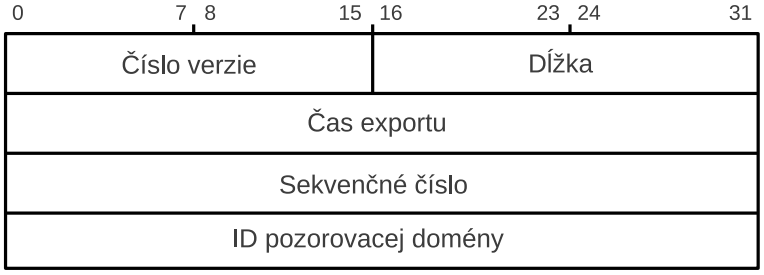
V nasledujúcich častiach kapitoly je uvedený formát IPFIX správ. Tieto informácie budú prínosné pre pochopenie ďalších kapitol.

Tabuľka 2 – 2 Dekódované vzorové dáta toku [3]

Informačný element	**Hodnota**
sourceIPv4Address	192.168.2.104
destinationIPv4Address	147.232.40.94
protocolIdentifier	TCP (6)
sourceTransportPort	50808
destinationTransportPort	80
flowStartMilliseconds	1334168954836
flowEndMilliseconds	1334168956687
octetTotalCount	7502
packetTotalCount	44

2.1 Formát IPFIX správ

IPFIX správy majú jednoduchú štruktúru definovanú v špecifikácii protokolu [1], začínajúcu s hlavičkou, ktorá špecifikuje celkovú dĺžku správy a informácie o konkrétnej správe. Medzi tieto informácie patrí číslo pozorovacej domény, čas exportu správy exportovacím procesom, sekvenčné číslo. Verzia protokolu musí byť nastavená na číslo 10. Formát hlavičky IPFIX správy je uvedený na Obrázku 2 – 1.



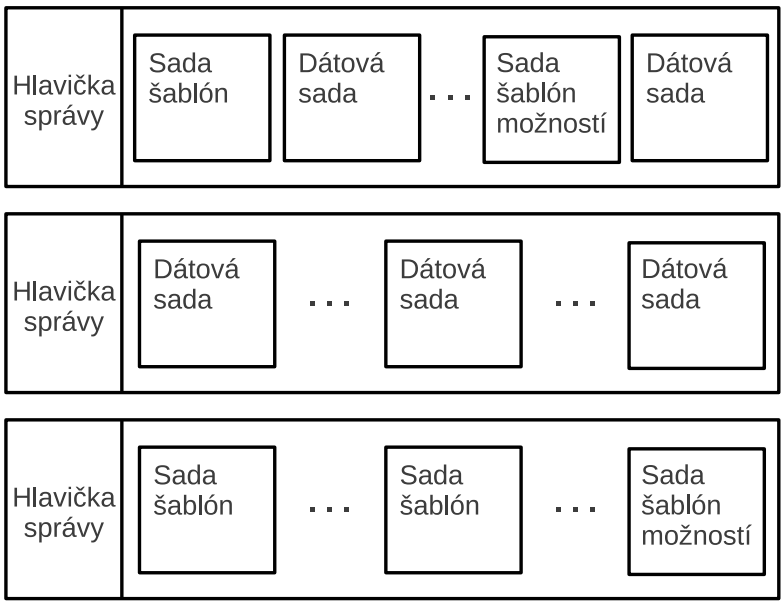
Obrázok 2 – 1 Formát IPFIX správy

2.1.1 IPFIX sada

IPFIX správa obsahuje jednu alebo viacero sád, v rôznych kombináciách. Existujú tri duhy sád: sada šablón, sada šablón možností a dátová sada. Jednou z možností je akákoľvek kombinácia druhu sád. Ak ešte ostáva miesto v IPFIX správe pre ďalšiu sadu, tá je zaradená za poslednú sadu. Napríklad ak je nutné znovu poslať šablónu, môže sa pribaliť k dátovej sade.

Najčastejšie posielanými sú dátové sady. Ak je definovaných viacero šablón, môže byť poslaných viacero dátových sád, z ktorých každá obsahuje dátové záznamy pre inú šablónu.

Treťou možnosťou kombinácie je IPFIX správa pozostávajúca výhradne zo sád šablón a sád šablón možností. Príklady kombinácií sád v IPFIX správe sú uvedené na Obrázku 2 – 2.



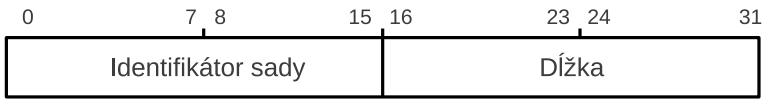
Obrázok 2 – 2 Rôzne kombinácie sád v IPFIX správe

Sada v závislosti od svojho druhu obsahuje:

- záznamy šablón,

- záznamy šablón možností,
- dátové záznamy.

Druh sady je definovaný v hlavičke sady v poli identifikátor sady. Tento identifikátor nadobúda hodnotu 2, ak sa jedná o sadu šablón. Hodnota 3 je rezervovaná pre sady šablón možností. Hodnoty 0 a 1 sa nepoužívajú z historických dôvodov, hodnoty 4 až 255 sú rezervované pre budúce použitie. Hodnoty väčšie ako 255 znamenajú, že sa jedná o dátovú sadu a toto číslo zároveň identifikuje šablónu, ktorá definuje dátové záznamy. Súčasťou hlavičky je aj celková dĺžka sady zahŕňajúca aj túto hlavičku. Hlavička sady je zobrazená na Obrázku 2 – 3.

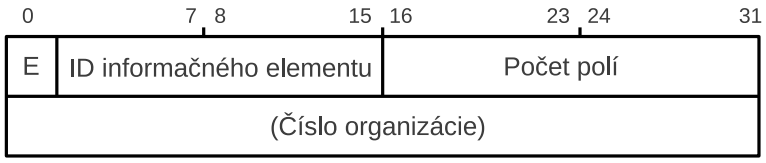


Obrázok 2 – 3 Formát hlavičky sady

2.1.2 Špecifikátor poľa

Informačné elementy sú rozlišované identifikátorom informačného elementu. Špecifikátor poľa obsahuje pole definujúce dĺžku informačného elementu, ktorú zhromažďovací proces využíva pri dekódovaní dát v dátových záznamoch. Dĺžka sa uvádza ako v [5], avšak môže byť aj menšia, ak sa použije redukované kódovanie.

Špecifikátor poľa umožňuje v šablóne indikovať použitie vlastného informačného elementu. Ak je nastavený prvý bit (Enterprise bit) špecifikátora poľa, jedná sa o informačný element špecifický pre organizáciu (Enterprise-specific). V tomto prípade musí byť prítomné aj číslo organizácie (Enterprise Number). Inak, toto číslo nesmie byť prítomné. Formát špecifikátora poľa je uvedený na Obrázku 2 – 4.



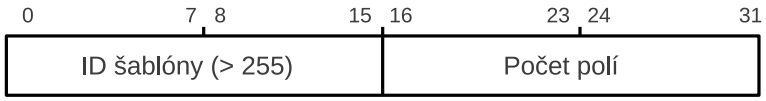
Obrázok 2 – 4 Formát špecifikátora poľa

2.1.3 Záznam šablóny

Záznam šablóny je veľmi dôležitý druh záznamu. Šablóny veľmi rozširujú flexibilitu formátu záznamu, pretože umožňujú zhromažďovaciemu procesu spracovať IPFIX správy aj bez toho, aby poznal interpretáciu všetkých dátových záznamov.

Záznam šablóny obsahuje akúkoľvek kombináciu identifikátorov informačných elementov. Či už ide o štandardné informačné elementy priradené organizáciou IANA (Internet Assignet Numbers Authority) a/alebo informačné elementy špecifické pre organizáciu.

Záznam šablóny začína hlavičkou uvedenou na Obrázku 2 – 5. Obsahuje identifikátor šablóny spoločne s počtom polí vyskytujúcich sa v nej. Identifikátor šablóny nadobúda hodnoty od 255 do 65535. Hodnoty 0 až 255 sú vyhradené pre sady šablón, sady šablón možností a možné ešte nevytvorené druhy sád. Za hlavičkou nasleduje množina špecifikátorov polí v počte uvedenom v hlavičke.



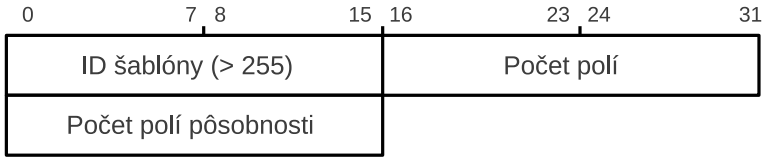
Obrázok 2 – 5 Formát hlavičky záznamu šablóny

2.1.4 Záznam šablóny možností

Záznam šablóny možností umožňuje exportéru poskytovať dodatočné informácie kolektoru. Bežným spôsobom využitia záznamu šablón možností je informovanie

zhromažďovacieho procesu o kľúčoch toku, vzorkovacích parametroch, či pravidelný export štatistických informácií o meracom či exportovacom procese, napríklad celkový počet exportovaných, zahodených alebo ignorovaných paketov.

Záznam šablóny možností má podobnú štruktúru ako záznam šablóny. Hlavička obsahuje navyše pole obsahujúce počet polí pôsobnosti. Pôsobnosť udáva kontext, na ktorý sa informácie vzťahujú. Ak je prítomných viacero polí pôsobností, celková pôsobnosť je definovaná kombináciou týchto polí. Napríklad exportované dáta sa budú vzťahovať na konkrétny merací proces a na konkrétnu šablónu. Počet polí pôsobnosti nesmie byť rovný nule. Pole počet polí udáva celkový počet polí, vrátane polí pôsobnosti. Polia pôsobnosti sú v zázname šablóny možností vždy na začiatku. Formát hlavičky záznamu šablón možností je uvedený na Obrázku 2 – 6



Obrázok 2 – 6 Formát hlavičky záznamu šablóny možností

2.1.5 Dátový záznam

Dátový záznam pozostáva z jednej alebo viacerých hodnôt polí. Číslo šablóny, ku ktorej patrí dátový záznam je uvedené v hlavičke dátovej sady, v poli identifikátora sady. Dátový záznam teda nemá svoju hlavičku. Preto všetky dátové záznamy v sade sú definované jednou šablónou. Hodnoty polí sú zakódované, ako je uvedené v informačnom modeli IPFIX [5]. Ako bolo spomenuté vyššie, interpretovanie dátových záznamov je možné, len ak je príslušná šablóna evidovaná zhromažďovacím procesom.

2.2 Redukované kódovanie informačných elementov

Špecifikácia IPFIX [1] uvádza, že informačné elementy obsahujúce celé alebo desatinné číslo, môžu byť zakódované s menším počtom bajtov, ako je definované v informačnom modeli IPFIX [5], za predpokladu že menšia veľkosť je postačujúca na prenesenie akejkoľvek hodnoty, ktorú môže exportér vyžadovať poslať. Napríklad, ak exportér nikdy nebude potrebovať poslať hodnotu väčšiu ako 4294967295, môže použiť dátový typ *unsigned32*.

Napríklad, centrálny smerovač môže vyžadovať 8 bajtov, zatiaľ čo menej vyťažený smerovať si vystačí so štyrmi bajtmi.

Na uplatnenie tohto princípu je jednoducho v šablóne uvedená menšia dĺžka poľa, ako tá, ktorá je asociovaná s daným typom informačného elementu. V príklade vyššie, exportér by v šablóne vložil ako dĺžku poľa rovnú 4 namiesto 8.

Ak sa používa redukované kódovanie, musí sa použiť iba s týmito typmi: *unsigned64*, *signed64*, *unsigned32*, *signed32*, *unsigned16* a *signed16*. Redukcia veľkosti môže byť o lubovoľné množstvo oktetov, pokiaľ sa hodnota zmestí do poľa, teda, aby sa zahodili úvodne nuly. Napríklad, typ unsigned64 môže byť redukovaný na 7, 6, 5, 4, 3, 2 alebo 1 oktet.

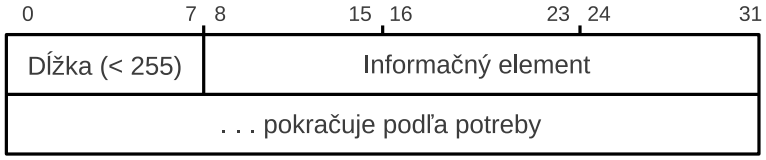
Redukované kódovanie môže byť tiež použité na typ *float64*. Výsledkom by bol typ *float32*. Keďže typ *float32* má nielen menší rozsah hodnôt ale vďaka menšej mantise je aj menej presné.

Redukované kódovanie nesmie byť použité na typy *dateTimeMicroseconds* alebo *dateTimeNanoseconds* [1].

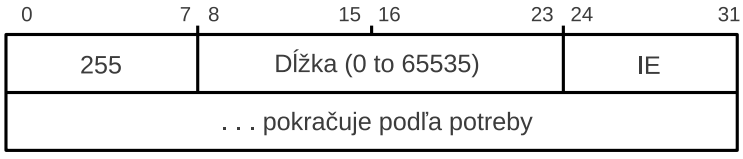
2.3 Informačné elementy s variabilnou dĺžkou

Mechanizmus šablón v protokole IPFIX je optimalizovaný hlavne na informačné elementy s fixnou dĺžkou, avšak je možné použiť aj informačné elementy s variabilnou dĺžkou. Informačný element s variabilnou dĺžkou je v zázname šablóny definovaný špecifikátorom poľa, ktorého dĺžka je nastavená na hodnotu 65535. Táto rezervovaná hodnota informuje zhromažďovací proces, že dĺžka informačného elementu sa ukrýva v obsahu samotného informačného elementu v dátovom zázname.

Vo väčšine prípadov, dĺžka informačného elementu je menšia ako 255 bajtov. Prvý bajt v hodnote poľa dátového záznamu tohto informačného elementu udáva počet nasledujúcich bajtov tohto informačného elementu. Táto konfigurácie je zobrazená na Obrázku 2 – 7. Ak je veľkosť informačného elementu väčšia ako 255, musí sa prvý bajt nastaviť na hodnotu 255 a veľkosť informačného elementu je zakódovaná v nasledujúcich dvoch bajtoch ako bezznamienkové 16-bitové číslo. Tento prípad je možné vidieť na Obrázku 2 – 8. Bajty nesúce informácie o dĺžke informačného elementu nie sú zahrnuté do tejto dĺžky, avšak počíta sa s nimi v poli dĺžka v hlavičke IPFIX správy a sady ..



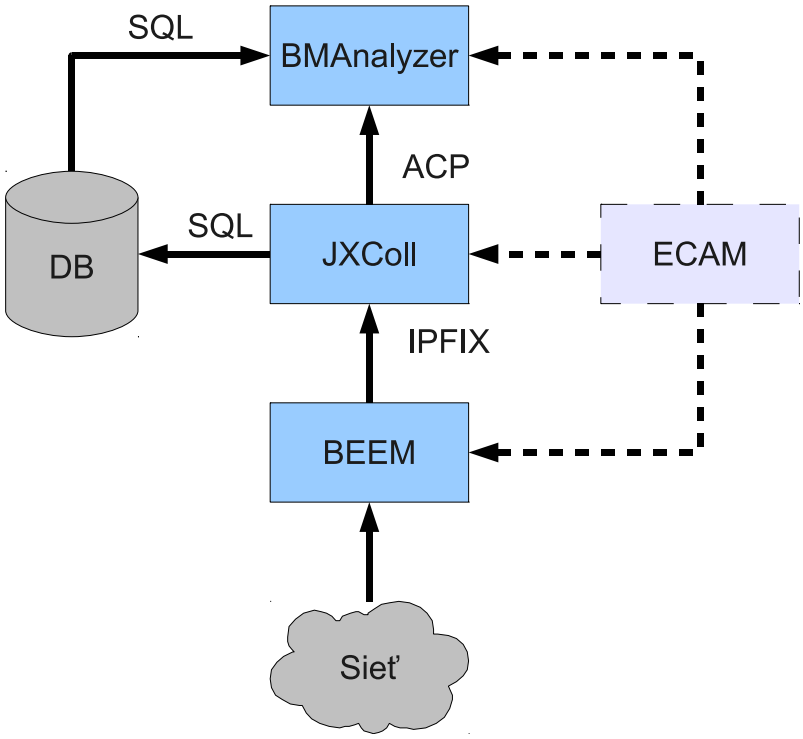
Obrázok 2 – 7 Príklad informačného elementu s variabilnou dĺžkou menšou ako 255B



Obrázok 2 – 8 Príklad informačného elementu s variabilnou dĺžkou väčšou ako 255B

2.4 Nástroj BasicMeter

BasicMeter [7] je nástroj umožňujúci vykonávať meranie prevádzkových parametrov počítačovej siete podľa štandardu IPFIX a PSAMP (Pcaket Sampling). Je vyvíjaný výskumnou skupinou MONICA ako projekt Laboratória počítačových sietí na Technickej univerzite v Košiciach. Vývoj začal v roku 2003 a každým rokom pribúdajú nové funkcie. Architektúru nástroja BasicMeter, kde ako analyzujúca aplikácia figuruje BMAnalyzer, ukazuje Obrázok 2 – 9.



Obrázok 2 – 9 Architektúra nástroja BasicMeter [9].

Nástroj BasicMeter sa skladá z týchto komponentov:

- **BEEM** - Jedná sa o merací a exportovací proces. Jeho hlavnou úlohou je zaznamenávať sieťovú prevádzku, klasifikovať ju do tokov na základe kľúčov toku, viesť o nich záznamy. Po ich ukončení ich exportuje smerom k zhromažďovaciemu procesu vo forme IPFIX správ.

- **JXColl** - Úlohou zhromažďovacieho procesu je prijímať a ukladať záznamy o tokoch prijatých od jedného alebo viacerých exportérov, za účelmi neskoršej analýzy, prípadne posielanie týchto záznamov priamo na analyzujúcu aplikáciu pomocou protokolu ACP. [8].
- **BMAnalyzer** - Predstavuje analyzujúcu aplikáciu, ktorá slúži hlavne na grafické zobrazenie dát vo forme grafov. Zobrazuje dáta, ktoré zhromaždil JXColl.
- **bmIDS** - Predstavuje systém pre detekciu narušenia v počítačovej sieti. Jeho úlohou je analyzovať prebiehajúcu komunikáciu v uzle siete a následne určiť mieru pravdepodobnosti prebiehajúceho útoku, resp. signalizovať anomálie. Údaje získava z JXColl v reálnom čase pomocou aplikačného rozhrania ACP. Na vyhodnotenie prevádzky a detekciu jednotlivých útokov bmIDS používa fuzzy subsystémy.
- **ECAM** - ECAM predstavuje nástroj pre centrálnu správu nástroja BasicMeter. Jeho účelom je ovládanie exportovacích, meracích a zhromažďovacích procesov. Pozostáva z aplikácií EcamDaemon, predstavujúci klientsku časť systému, ktorý je spustený na každom stroji, ktorý má byť ovládaný. EcamServer poskytuje grafické používateľské rozhranie a slúži ako centrum ECAM architektúry. Umožňuje vytvoriť, zmazať jednotlivé inštancie exportérov či kolektorov, alebo zmeniť ich konfiguráciu.

3 Analýza transportných protokolov

V tejto kapitole sú uvedené základné informácie a špecifické vlastnosti transportných protokolov TCP a SCTP, ktoré je možné použiť ako prostriedok na prenos informácií o dátových tokoch.

3.1 Protokol UDP

Protokol UDP (User Datagram Protocol) [10] je nespojovaný a nespoľahlivý transportný protokol. Dáta aplikačnej vrstvy sú posielané ako samostatné správy, čo je výhodné pre protokoly orientované na správy, akým je aj IPFIX.

Protokol negarantuje doručenie paketu cieľovej aplikácii. Odosielateľ nemá žiadnu možnosť zistiť, či bola správa doručená, pretože nedostáva žiadne potvrdenia. Správa sa môže na ceste k adresátovi stratiť alebo zahodiť kvôli vysokému preťaženiu siete, prípadne pakety môžu dôjsť v inom poradí, v akom boli odoslané.

Nedoručenie UDP paketu obsahujúceho záznam šablóny môže znamenať stratu potenciálne veľkého množstva nasledujúcich údajov, keďže kolektor bez šablóny nie je schopný prijaté dáta dekódovať.

Protokol UDP by mal byť použitý na prenos údajov o IP tokoch len za predpokladu, že linka medzi exportérom a kolektorom nie je náchylná na preťaženie, alebo ak je táto linka na túto prevádzku iným spôsobom pripravená. [1]

3.2 Protokol TCP

Protokol TCP (Transmission Control Protocol) [11] je spojovo orientovaný, spoľahlivý transportný protokol, ktorý umožňuje prenášať údaje medzi dvoma koncovými bodmi, aplikáciami. Protokol je určený pre aplikácie, pre ktoré spoľahlivosť doručenia kompletných, nepoškodených dát v správnom poradí je nevyhnutnosťou. TCP

je najčastejšie používaným transportnným protokolom.

Aby mohli dve aplikácie navzájom komunikovať, musia vytvoriť spojenie. Spojenie je definované práve dvoma koncovými bodmi, ktorým je kombinácia IP adresy a portu aplikácie. Na vytvorenie spojenia sa používa procedúra zvaná Three-way handshake. Iniciátor (klient) pošle SYN segment na počúvajúcu stanicu servera. Tento segment ma nastavený bit SYN v TCP hlavičke a obsahuje počiatočné sekvenčné číslo. Server následne rezervuje zdroje pre komunikáciu a odpovie paketom s nastavenými bitmi SYN a ACK. Súčasťou segmentu je sekvenčné číslo strany servera a potvrdzovacie číslo. Potvrdzovacie číslo je inkrementáciou sekvenčného čísla, ktoré v prvom kroku poslal klient. V poslednej fáze klient odpovedá poslaním ACK segmentu, v ktorom pošle potvrdzovacie číslo klientovi. Nevýhodou tohto procesu je fakt, že server už pri obdržaní SYN segmentu alokuje zdroje a umožňuje vykonať útok založený na princípe odmietnutia služby (Denial of Service). Pri poslaní množstva falošných SYN segmentov dôjde k vyčerpaniu zdrojov servera.

Spojenie medzi dvoma koncovými bodmi umožňuje obojsmernú komunikáciu vo forme dvoch samostatných prúdov. Jeden pre každý smer komunikácie. Na rozdiel od UDP, ktoré zachováva hranice správy, je pre protokoly pracujúce so správami z programátorského hľadiska zložitejšie oddeliť jednotlivé správy od seba.

TCP garantuje, že dáta budú doručené druhej strane presne tak, ako boli poslané, bez stratených, zle usporiadaných alebo duplicitných dát. Za týmto účelom používa sekvenčné čísla, potvrdzovacie segmenty a kontrolné súčty.

V prípade nepotvrdenia dát prijímateľom a vypršaním časovača pre retransmisiu, je nutné segment poslať znova, až kým prijímateľ nepotvrdí jeho prijatie. S tým súvisí aj problém zvaný *head-of-line blocking*. Je to problém, kde posielanie nezávislých správ cez TCP spojenie spôsobuje ich neskoré doručenie, pretože sú zadržiavané vo vyrovnávacej pamäti prijímateľa, kým ten neobdrží predchádzajúce stratené správy.

Súčasťou potvrdení je aj pole veľkosť okna. Toto okno informuje o veľkosti voľnej

vyrovnávacej pamäte prijímateľa a predstavuje dáta, ktoré odosielateľ môže vyslať tak, aby ich vedel prijímateľ spracovať. Na základe tejto hodnoty odosielateľ prispôsobuje množstvo súčasne posielaných dát. Podrobnejšie informácie možno nájsť v [12].

Na ukončenie spojenia sa štandardne používa výmena štyroch segmentov. Jedna zo staníc (A) pošle FIN segment. Tým oznámi druhej strane (B), že už jej nebude posielať ďalšie dáta. Stanica B potvrdí prijatie FIN segmentu poslaním ACK segmentu.

Rovnakú procedúru, ale v opačnom smere vykoná aj stanica B. Kým stanica A neprijme FIN segment, stále môže čítať dáta od stanice B. Tejto vlastnosti protokolu TCP sa hovorí aj polovične uzavreté spojenie (half-closed connection). Ide o slušné ukončenie spojenia. FIN segment je vložený do fronty na odoslanie ako normálny segment, teda aplikácia na druhej strane prijme všetky dáta a potom narazí na koniec prúdu.

Ukončiť spojenie možno aj násilne pomocou RST segmentu. Ak akákoľvek strana vyšle RST segment, znamená to, že celé spojenie sa predčasne ukončí a všetky dáta prítomné vo vstupnej či výstupnej fronte môžu byť zahodené. RST segment sa vyšle napríklad aj pri haváriách operačného systému. Po naštartovaní počítača už neexistuje kontext pôvodného pripojenia a preto po prijatí neznámych dát je odosielateľ o neaktuálnom spojení informovaný RST segmentom [13].

3.3 Protokol SCTP

Podľa [14], zvyšujúce sa množstvo nových aplikácií považuje protokol TCP za príliš limitujúci a vznikli protokoly zebezpečujúce spoľahlivý prenos dát ako samostatná vrstva nad protokolom UDP.

Hlavné nedostatky, ktoré si uživatelia žiadajú odstrániť sú:

- Ako bolo uvedené vyššie, TCP poskytuje spoľahlivý a striktne zoradzovaný

prenos dát. Niektoré aplikácie vyžadujú spoľahlivosť, ale nie zoradené doručenie, zatiaľ čo iné by si vystačili s čiastočne spoľahlivým prenosom. Head-of-line blocking prítomný v TCP spôsobuje nežiadúce zdržania.

- Často je prúdová povaha TCP protokolu nevýhodou, pretože aplikácie musia používať vlastné značky, aby mohli jednotlivé správy od seba oddeliť.
- TCP sockety sú limitované na kombináciu jednej IP adresy a portu aplikácie. Ak dôjde k výpadku spojenia, musí sa nadviazať nové. TCP nie je schopné využiť možnosti staníc s viacerými sieťovými rozhraniami.
- TCP je relatívne náchylné na DoS útoky, napríklad SYN útoky.

Tvorba protokolu SCTP (Stream Control Transmission Protocol) [14] [2] [4] [16] [17] bola motivovaná najmä s cieľom prenášať signalizácie telefónnych sietí po IP sieti, kde všetky tieto nedostatky sú relevanté. Autori protokolu však vytvorili všeobecný protokol, ktorý môžu použiť aj iné aplikácie.

SCTP je spojovo orientovaný protokol, zabezpečujúci spoľahlivý prenos dát. Keďže je spoľahlivý, znamená to, že dokáže zaznamenať stratené, duplicitné alebo zle usporiadané dáta a súčasne obsahuje mechanizmy na kontrolu toku dát a kontrolu zahltenia, podobne ako protokol TCP. Tieto mechanizmy sú založené na využívaní kontrolných súčtov, sekvenčných čísiel a selektívnych retransmisiách.

SCTP zachováva hranice správy, podobne ako protokol UDP. Správy sú posielané ako jeden celok, nie sú súčasťou súvislého prúdu dát. SCTP paket je zložený z kusov, ktoré predstavujú buď samotné používateľské dáta alebo dáta slúžiace na správu asociácie.

V rámci jedného SCTP paketu môže byť poslaných viacero kusov dát. SCTP podporuje fragmentáciu dátových kusov. To znamená, že ak je požadovaná správa väčšia ako MTU (Maximum Transmission Unit) na ceste od jednej k druhej stanici (Path MTU), SCTP túto správu rozdelí do viacerých kusov a pošle ich samostatne aby

zabránil IP fragmentácii. Na druhej strane sa všetky kusy spoja do jedného celku a posunú sa aplikácii ako celistvá správa. Protokol tiež za účelom zvýšenia efektivity posielania dát umožňuje vložiť do jedného SCTP paketu viacero dátových kusov patriacich rôznym správam. Kontrolné kusy sú vždy prvé v poradí, a až za nimi nasledujú dátové kusy.

Jeden koniec asociácie nie je identifikovaný portom a len jednou IP adresou, ale množinou IP adries (IPv4 a súčasne aj IPv6) nastavených na svojich rozhraniach. Stanica s viacerými rozhraniami (multihomed peer) je dostupná cez viacero transportných adries. Ak sa jedna z nich stane nedostupnou, dáta je možné doručiť cez inú transportnú adresu, bez toho, aby si to všimla aplikácia. SCTP používa vlastnosť *multihoming* iba na zabezpečenie redundancie, nie na vyrovnávanie záťaže. Každý koncový bod si zvolí jednu primárnu cieľovú transportnú adresu, cez ktorú posiela údaje počas životnosti asociácie.

SCTP používa na vytvorenie spojenia proces pozostávajúci zo štyroch krokov (*four-way handshake*). Tento proces znemožňuje využiť SYN útoky ako pri protokole TCP.

Proces iniciácie asociácie stanicou A so stanicou B je nasledovný:

1. Stanica A vyšle INIT kus stanici B.
2. Keď stanica B prijme INIT kus, vráti stanici A INIT-ACK kus. Tento INIT-ACK kus obsahuje cookie zloženú z informácie, ktorú si môže overiť len stanica B. Cookie pomáha overiť, či je stanica A legitímna. Stanica B zatiaľ nealokuje žiadnu pamäť a neuchováva stav potencionálnej asociácie.
3. Keď stanica A prijme INIT-ACK, odpovie s kusom COOKIE-ECHO. Ako názov hovorí, vracia späť cookie, ktoré stanica B poslala v predchádzajúcom kroku.
4. Po príjme COOKIE-ECHO kusu, stanica B overí platnosť tohto cookie. Po úspešnom overení je asociácia úspešne nadviazaná.

V posledných dvoch krokoch sa už môžu vyskytovať prvé dáta, čo zrýchľuje čas potrebný na prvé prenesenie informácie.

Počas vytvárania spojenia si obe strany dohodnú počet prúdov, ktoré budú používať. SCTP podporuje viacero nezávislých logických prúdov správ v rámci asociácie. Každá správa poslaná cez SCTP asociáciu je priradená k prúdu. Správy v rámci prúdu sú doručené v poradí, v akom boli poslané. Stratené dáta jedného prúdu neblokujú príjem dát v iných prúdoch. Eliminuje sa tým nepriaznivý efekt *head-of-line blocking*, prítomný v protokole TCP.

SCTP ponúka tiež spoľahlivé doručovanie správ bez zoradzovania, čím sa líši od protokolu UDP, ktorý síce ponúka nezoradzovaný prenos, ale žiadnu spoľahlivosť. To či bude prenos zachovávať poradie je možné nastaviť pre každú správu zvlášť. Ak je správa poslaná bez zoradzovania, príslušnosť k prúdu je irelevantná, jedná sa o samostatnú kategóriu prenosu.

Každej správe je možné nastaviť životnosť. Táto vlastnosť dovoľuje vysielajúcej aplikácii definovať, ako dlho je správa užitočná. Ak tento čas vyprší skôr, ako môže byť správa poslaná príjemcovi, odosielateľ môže preskočiť túto správu. Ak už správa bola poslaná, ale stratila sa na ceste k príjemcovi, dochádza k blokovaniu ďalších už prijatých dát na tomto prúde, kým správu prijímateľ neobdrží.

S použitím rozšírenia čiastočnej spoľahlivosti protokolu SCTP [15] je však možné poslaním FORWARD TSN kusu informovať cieľovú stanicu o tom, že danú správu už nemá očakávať a že si má posunúť svoje kumulatívne transportné sekvenčné číslo prijatých dát na hodnotu uvedenú v tomto kuse. Týmto spôsobom dokáže aplikácii sprístupniť dáta, ktoré doteraz museli čakať, kvôli nedoručeniu tých predchádzajúcich.

Protokol SCTP narozdiel od TCP nepozná polovične uzatvorené spojenia. Na "slušné" ukončenie asociácie sa používa proces pozostávajúci z troch krokov. Iniciátor ukončenia spojenia pošle kus SHUTDOWN, druhá strana odpovie kusom SHUTDOWN-

ACK a nakoniec iniciátor pošle kus SHUTDOWN-COMPLETE. Nekorektné ukončenie je indikované poslaním kusu ABORT.

4 Analýza súčasného stavu programu JXColl

V tejto kapitole je čitateľ uvedený do zhromažďovacieho procesu nástroja BasicMeter. Zhromažďovacím procesom v nástroji BasicMeter je program JXColl (Java XML Collector). Budú tu predstavené základné pojmy tohto nástroja, zistené nedostatky a tiež súčasný stav podpory pre transportné protokoly TCP a SCTP.

Hlavnou úlohou programu JXColl je príjem správ vo formáte IPFIX od exportéra. Program dekóduje informácie obsiahnuté v týchto správach a ukladá ich do centrálneho úložiska pre neskoršiu analýzu. Centrálnym úložiskom je databáza vytvorená v databázovom systéme PostgreSQL. Program taktiež umožňuje prenos informácií o tokoch v reálnom čase na analyzujúcu aplikáciu prostredníctvom protokolu ACP. JXColl je konzolová aplikácia, u ktorej sa predpokladá, že bude bežať na pozadí ako služba. V tomto prípade sa všetok výstup zapisuje do log súboru.

4.1 Hlavné nedostatky v oblasti konformity so špecifikáciou IPFIX

Na testovaní interoperability IPFIX nástrojov v Prahe v roku 2011 sa zistilo niekoľko závažných nedostatkov v zhromažďovacom procese nástroja BasicMeter, ktoré vyplývajú z nedostatočnej konformity so špecifikáciou IPFIX.

4.1.1 Viacero záznamov v sade

Jedným z takýchto nedostatkov bola skutočnosť, že pri pokuse o export informácií o IP tokoch s použitím viacerých šablón, JXColl takejto definícii sady šablón nerozumel. JXColl sa začal správať veľmi zvláštne a do databázy boli ukladané nesprávne informácie. Dáta boli interpretované chybným spôsobom. Pri analýze kódu sa zistilo, že v sade bolo uvažované len s jedným záznamom. Ako bolo spomenuté pri analýze protokolu IPFIX, záznamov môže byť viacero, či už v sade šablón, sade

šablón možností alebo v dátovej sade. JXColl v tomto prípade interpretoval ďalší záznam šablóny ako novú sadu. Keďže záznam šablóny začína identifikátorom šablóny (pozri Obrázok 2 – 5), ktorý je vždy väčší ako 255, JXColl interpretoval tieto údaje ako dátovú sadu. Na tento fakt sa prišlo takým spôsobom, že do databázy boli ukladané záznamy, v ktorých číslo pozorovacieho bodu bolo zhodné s číslom jednej zo šablón.

Tento problém sa teda týka nie len zlého spracovania viacerých záznamov šablón, ale aj všetkých druhov záznamov. Dochádzalo by pritom k ukladaniu nesprávnych dát do databázy, alebo by sa neuložili údaje vôbec. Opísaný jav nebol spozorovaný pri predchádzajúcom vývoji aj z toho dôvodu, že exportovací proces nástroja BasicMeter (BEEM) posielal vždy najviac jeden záznam šablóny v sade šablón, nasledovaný dátovou sadou s jedným dátovým záznamom. Každá IPFIX správa obsahovala najviac jednu dátovú sadu s jedným dátovým záznamom.

4.1.2 Informačné elementy s redukovaným kódovaním

JXColl tiež nebol schopný prijímať informačné elementy, pri ktorých sa použije redukované kódovanie (pozri kapitolu 2.2). Najčastejšie sú nimi informačné elementy *octetDeltaCount* a *packetDeltaCount*. Ak boli od exportéra prijaté dátové záznamy s redukovaným kódovaním, dochádzalo k ich preskočeniu, pretože JXColl kontroloval, či veľkosť prijatých dát súhlasí s informáciou uvedenou v informačnom modeli. Tieto hodnoty predstavujú maximálnu hodnotu, s ktorou treba uvažovať pri návrhu databázy. Pri prenose údajov od exportéra je nutné počítať aj so skrátenými informačnými elementmi.

4.1.3 Nesprávne alebo chýbajúce dátové typy

Nedostatky sa takisto prejavili pri dekódovaní niektorých abstraktných dátových typov. Problémy sa vyskytli s časovými známkami používajúcimi abstraktný údajový

typ *dateTimeMicroseconds* a *dateTimeNanoseconds*. Tie nie sú dekódované podľa definície v informačnom modeli protokolu IPFIX [5]. Treba napraviť túto skutočnosť. Navyše, kolektor nepodporuje dekódovanie niektorých abstraktných dátových typov uvedených v informačnom modeli, menovite *boolean*, *macAddress* a *octetArray*.

4.1.4 Variabilná veľkosť informačných elementov

Niektoré informačné elementy, napríklad *octetArray* a *string*, môžu byť buď fixnej, alebo variabilnej dĺžky. Kolektor nemá podporu pre informačné elementy variabilnej dĺžky, čo znova predstavuje problém interoperability. Pri príjme dát s takýmto informačným elementom by JXColl havaroval, pretože by sa snažil z dátového záznamu vybrať 65535 bajtov, namiesto toho aby zistil skutočnú veľkosť dát (pozri kapitolu 2.3). Pri výbere by došlo k neočakávanému nárazu na koniec buffra.

4.1.5 Spracovanie organizáciou definovaných informačných elementov

Pri analýze kódu sa zistilo, že spracovanie oranizáciou definovaných informačných elementov (enterprise-specific) je nekorektné. Všetky IETF informačné elementy spoločne s informačnými elementmi špecifickými pre organizáciu JXColl očakáva v súbore *ipfixFields.xml*. Organizáciou definované informačné elementy v tomto XML súbore obsahujú atribút *enterpriseID* obsahujúci číslo organizácie. Ak by však identifikátor tohto informačného elementu bol zhodný s identifikátorom štandardného IETF informačného elementu, JXColl by mal informácie len o prvom výskyte informačného elementu v XML súbore s takýmto identifikátorom. Číslo organizácie by nebral do úvahy. Pri exporte tohto informačného elementu v uvedenej situácii by došlo k nesprávnemu dekódovaniu a teda k chybám. Podobne, ak by JXColl prijímal v dátovom zázname organizáciou definovaný informačný element, ktorý síce nie je uvedený v súbore *ipfixFields.xml*, ale identifikátor tohto informačného elementu by bol zhodný s niektorým štandardným IETF informačným elementom, dekódova-

nie by prebiehalo podľa definície tohto štandardného informačného elementu. To je samozrejme neprípustné, a je potrebné danú skutočnosť napraviť.

4.2 Podpora transportných protokolov

Ako bolo spomenuté v úvode, JXColl podporuje príjem dát o IP tokoch podľa protokolu IPFIX len cez transportný protokol UDP. Pre interné potreby výskumnej skupiny MONICA bol tento protokol doposiaľ postačujúci, avšak podľa špecifikácie IPFIX [1] je preferovaným transportným protokolom SCTP s rozšírením čiastočnej spoľahlivosti (PR-SCTP). Pomocou tohto protokolu možno nastaviť, ako dlho sa má exportovací proces snažiť poslať dátový záznam, alebo zadefinovať maximálny počet retransmisií, po ktorých sa dátový záznam zahodí.

V prípade, ak protokol SCTP nie je možné nasadiť (napríklad z dôvodu chýbajúcej podpory v operačnom systéme), je možné použiť protokol TCP alebo UDP.

V súvislosti so šablónami špecifikácia IPFIX udáva, že rôzne transportné protokoly majú pristupovať k správe šablón rôznym spôsobom.

4.2.1 Správa šablón pre protokol UDP

Pri prenose údajov cez transportný protokol UDP by mal zhromažďovací proces uvažovať životnosť šablón. Šablóny neobnovené exportovacím procesom do vypršania ich životnosti majú byť expirované a následne zahodené. Zároveň majú byť zahodené aj všetky dátové záznamy referujúce na túto expirovanú šablónu. Je nutné informovať o tejto udalosti. Životnosť šablón by mala byť konfigurovateľná a mala by byť nastavená aspoň na trojnásobok periódy obnovy šablón exportérom. Identifikátory šablón sú jedinečné na UDP reláciu a pozorovaciu doménu. Zhromažďovací proces by mal uchovávať nasledujúce údaje pre všetky aktuálne záznamy šablóny (možností):

- IPFIX zariadenie - IP adresa,
- zdrojový UDP port exportéra,
- číslo pozorovacej domény,
- číslo šablóny,
- definícia šablóny,
- čas posledného prijatia.

JXColl neuchováva zdrojový UDP port exportéra. Ak by bolo spustených viacero inštancií exportéra na jednom stroji s totožnou pozorovacou doménou, a ak by posielali rovnaký identifikátor šablóny, ale s inou definíciou, šablóny z oboch exportérov by sa navzájom prepisovali. Dochádzalo by k nesprávnemu dekódovaniu a ukladaniu údajov do databázy, prípadne by k tomuto kroku ani nedošlo.

Takisto nie je vyriešené expirovanie šablón. Neexistuje žiadna evidencia času príchodu záznamov šablón (možností) z exportéra. Je potrebné preto implementovať aj túto požiadavku.

4.2.2 Správa šablón pre protokoly SCTP a TCP

Pri prenose údajov o IP tokoch cez transportný protokol SCTP podľa špecifikácie IPFIX [1] treba uvažovať s tým, že exportovací proces priraďuje a spravuje identifikátory šablón osobitne pre každú SCTP asociáciu a pozorovaciu doménu.

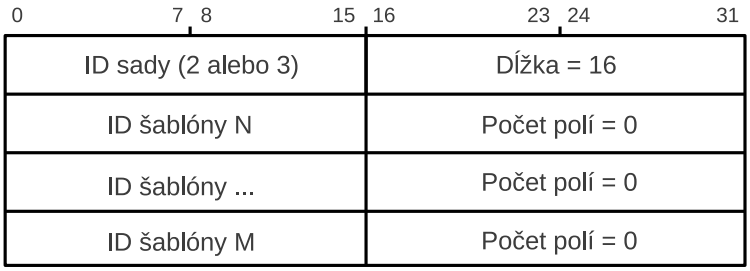
Zhromažďovací proces musí uchovávať informáciu o zázname šablóny počas trvania SCTP asociácie, teda nedochádza k periodickému obnovovaniu šablóny ako v prípade protokolu UDP. Mal by ich poslať na začiatku asociácie , aby mohli byť interpretované dátové záznamy referujúce na tieto šablóny. Ak kolektor príjme už evidovanú šablónu z rovnakej asociácie a pozorovacej domény bez toho, aby bola predtým zrušená, potom musí kolektor asociáciu vypnúť. Po zrušnení asociácie musí

zhromažďovací proces zahodiť všetky šablóny prijaté cez túto asociáciu a zastaviť dekódovanie IPFIX správ, ktoré používajú tieto šablóny.

Rozdielne pozorovacie domény z rovnakej SCTP asociácie môžu použiť rovnaký identifikátor šablóny na odkazovanie sa k odlišným šablónam. Šablóny, ktoré už viac nie sú používané, by mali byť vymazané použitím špeciálneho typu šablóny, Template Withdrawal Message.

Pred poslaním tejto správy by mal exportér čakať určitý čas (napríklad 5 sekúnd), aby mal kolektor dostatok času na spracovanie posledného dátového záznamu použitím pôvodnej šablóny. Definícia novej šablóny s rovnakým identifikátorom a prípadné dáta vzťahujúce sa na túto šablónu by nemala byť exportérom poslaná pokiaľ neubehne dostatočný čas, na to aby kolektor spracoval Template Withdrawal Message.

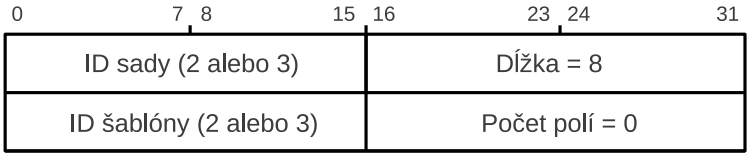
Je to záznam šablóny s počtom polí nastaveným na 0. Identifikátor záznamu šablóny určuje, ktorá šablóna má byť vymazaná. Ukážkový formát správy Template Withdrawal Message je uvedený na obrázku 4 – 1.



Obrázok 4 – 1 Formát Template Withdrawal Message

Zrušenie šablóny sa používa najmä ak sa zmenia meracie parametre na meracom procese, alebo ak sa reštartuje merací proces. Ak sa reštartuje merací proces, tak exportovací proces buď použije pôvodné šablóny, alebo ich musí zrušiť pomocou Template Withdrawal Message pred ich znovupoužitím. Na zrušenie všetkých šablón danej pozorovacej domény sa môže použiť špeciálna verzia Template Withdrawal Message. Jej formát je na Obrázku 4 – 2. Počet polí je nulový a číslo šablóny je buď 2 alebo 3, podľa toho či sa jedná o All Data Template Withdrawal Message (vymaže

všetky šablóny pre danu pozorovaciu doménu) alebo ide o All Options Template Withdrawal Message (vymaže všetky šablóny možností pre danú pozorovaciu doménu). Pozorovacia doména je uvedená v hlavičke IPFIX správy ako si je možné všimnúť na obrázku 2 – 1. Keď sa reštartuje SCTP asociácia, exportovací proces musí znovu poslať všetky záznamy šablón.



Obrázok 4 – 2 Formát All (Data alebo Options) Template Withdrawal Message

Pre prenos údajov pomocou protokolu TCP platia rovnaké pravidlá správy šablón ako pri protokole SCTP, s ohľadom na to, že v protokole TCP sa jedná o spojenia, nie o asociácie.

4.2.3 Hlavné požiadavky na prenos dát pomocou protokolu SCTP

Zhromažďovací proces by mal počúvať na porte 4739, ale má byť umožnená zmena tohto portu. Mal by umožniť viacero asociácií od exportérov. Exportér si vyžiada počet prúdov, ktoré bude používať na prenos.

Ak zhromažďovací proces prijíma poškodené IPFIX správy, musí zrušiť SCTP asociáciu, zrušiť IPFIX správu a mal by zaznamenať chybu.

Sady šablón a sady šablón možností môžu byť poslané na akomkoľvek SCTP prúde a to vždy spoľahlivo použitím zoradzovaného doručovania.

Zhromažďovací proces nesmie predpokladať, že dátové sady a asociované sady šablón (možností) sú exportované v rovnakej IPFIX správe.

Exportovací proces si môže vyžiadať viac ako jeden prúd na asociáciu. Každý z týchto prúdov môže byť použitý na prenos IPFIX správ obsahujúcich dátové sady, sady

šablón alebo sady šablón možností.

Podľa požiadaviek aplikácie, exportovací proces môže posielať dátové sady s úplnou alebo čiastočnou spoľahlivosťou, použitím zoradzovaného alebo nezoradzovaného doručovania, cez akýkoľvek SCTP prúd.

Keď zhromažďovací proces zistí, že SCTP asociácia bola abnormálne ukončená, musí pokračovať v počúvaní na novú asociáciu.

Ak zhromažďovací proces príjme Template Withdrawal Message pre záznam šablóny, ktorý predtým neprijal na konkrétnej SCTP asociácii, musí zrušiť SCTP asociáciu, zrušiť IPFIX správu, a mal by zaznamenať chybu, podobne ako to robí pre poškodené IPFIX správy.

4.2.4 Hlavné požiadavky na prenos dát pomocou protokolu TCP

Zhhromažďovací proces by mal počúvať na porte 4739, ale musí byť umožnená konfigurácia iného portu. Mal by umožniť viacero pripojení od exportérov. Keď sa vypne exportovací proces, mal by vypnúť TCP spojenie. Keď zhromažďovací proces viac nechce prijímať IPFIX správy, mal by ukončiť svoj koniec pripojenia. Zhromažďovací proces by mal pokračovať v čítaní IPFIX správ pokiaľ exportovací proces nezavrie svoj koniec.

Ak zhromažďovací proces prijíma poškodené IPFIX správy, musí zrušiť TCP spojenie, zahodiť IPFIX správu a mal by zaznamenať chybu.

Keď exportovací proces detekuje, že TCP spojenie na zhromažďovací proces je abnormálne ukončené, mal by sa opätovne pokúsiť o vytvorenie spojenia. Časové limity spojenia a intervaly, ako často sa má exportér znovu pokúšať o pripojenie by mali byť konfigurovateľné. V predvolenom nastavení, exportovací proces sa nesmie pokúšať zahájiť pripojenie častejšie ako raz za minútu.

IPFIX správy sú posielané cez TCP spojenie bez akéhokoľvek špeciálneho kódovania.

Pole *dĺžka* v hlavičke IPFIX správy definuje koniec každej IPFIX správy a teda začiatok nasledujúcej IPFIX správy. To znamená, že IPFIX správy nemôžu byť prekladané.

5 Návrh a implementácia zmien v programe JXColl

V tejto časti sú uvedené bližšie informácie o jednotlivých zistených problémoch spoločne s ich riešeniami. Medzi hlavné prvky programu JXColl patrí trieda *PacketListener*. Toto vlákno slúži ako UDP server, prijíma UDP datagramy a ukladá ich do *PacketCache*. *PacketCache* je trieda, ktorá obsahuje statickú premennú triedy predstavujúci vyrovnávaciu pamäť. Je to front typu *java.util.ArrayBlockingQueue*. Z tejto vyrovnávacej pamäte potom vlákno *NetXMLParser* vyberá jednotlivé prijaté správy a snaží sa ich interpretovať. Na uchovávanie informácií o šablónach využíva objekt *IPFIXTemplateCache*. *NetXMLParser* takisto pracuje s objektom *RecordDispatcher*, ktorý spracované dáta dekóduje a podľa nastavenia konfiguračného súboru ich ukladá do databázy, prípadne ich tiež posúva triedam slúžiacim na export dát v reálnom čase prostredníctvom protokolu ACP, alebo modulu na účtovanie. Táto práca sa venuje optimalizácii procesov vo vstupnej časti programu, preto oblasť výstupu nebude bližšie popisovaná.

5.1 Podpora abstraktných dátových typov IPFIX

Dekódovanie jednotlivých informačných elementov nastáva v triede *RecordDispatcher*. Metóda exportujúca údaje do databázy, *dbExport()*, má prístup k spracovanému údajovému záznamu a záznamu šablóny. Na dekódovanie dát je nevyhnutná tiež trieda *IpfixElements*, ktorá pri spustení programu načíta informačný model protokolu IPFIX z XML súboru *ipfixFields.xml*. Táto trieda obsahuje metódy na prístup k informáciam o informačných elementoch: abstraktný údajový typ, názov, skupina a tiež identifikátor. Na základe záznamu šablóny a informácií z informačného modelu, pre každé pole v zázname šablóny je dekódovaný kúsok dát obsiahnutý v dátovom zázname. Metóda slúžiaca na dekódovanie testuje dátový typ informačného elementu, podľa tohto typu interpretuje binárne dáta a výsledkom je reťazec obsahujúci hodnotu informačného elementu. Metóda na dekódovanie sa nachádza aj

v triede *RecordDispatcher* a aj v triede *ACPIPFIXWorker* slúžiacej na prenos dát cez protokl ACP. Je preto žiadúce vytvoriť samostatnú triedu dekódera, pretože v aktuálnej implementácii môže ľahko dôjsť k nekonzistencii oboch metód.

Takáto trieda by mohla byť IpfixDecoder. Hlavným vstupným bodom dekódera by mala byť metóda *decode()*, kde prvým argumentom by bol reťazec predstavujúci dátový typ informačného elementu, a druhým by bol objekt typu *java.nio.ByteBuffer*, obsahujúci samotné dáta. Každá skupina dátových typov je dekódovaná vo vlastnej metóde, čo prispieva k prehľadnosti. Prípadné nové údajové typy je ľahké doplniť.

Doposiaľ neboli implementované niektoré dátové typy. Dátový typ *octetArray* predstavuje konečný reťazec oktetov [5]. Keďže exportné triedy ukladajúce údaje do databázy a aj protokol ACP očakávajú ako výstup dekódovania reťazec znakov, je nutné ho na reťazec skonvertovať takým spôsobom, aby mohla byť hodnota obnovená do pôvodnej podoby.

Vhodným kódovaním sa zdá byť kódovanie Base64 [19]. Toto kódovanie je určené na prevod binárnych dát do textovej podoby a naspäť z textovej do binárnej podoby. Používa sa najmä pri prenose príloh v elektronickej pošte alebo na kódovanie binárnych údajov v XML súboroch.

Dátový typ *macAddress* predstavuje reťazec 6 oktetov [1]. Je preto nutné previesť binárne údaje na ich textovú reprezentáciu, aby výsledný reťazec dosiahol tvar XX:XX:XX:XX:XX, kde XX sú znaky 0 až F. Na to sa použil pomocný reťazec HEXES, ktorý obsahuje všetky hexadecimálne znaky zoradené podľa ich hodnoty vzostupne. Dekódovanie vykoná pomocná metóda *parseMacAddress(byte[] data)*. Je nutné zistiť hodnotu horných aj spodných štyroch bitov každého bajtu. Na základe týchto hodnôt na koniec výsledného reťazca bude priradený znak z pomocného reťazca HEXES z pozície, akú udáva daná hodnota. Medzi jednotlivými bajtmi zo vstupu sa vloží znak ":". Kvôli efektivite sa použil na vytváranie reťazca *StringBuilder*. Pred celým procesom sa testuje dĺžka vstupného poľa. Ak je veľkosť iná ako

6, metóda informuje volajúcu metódu výnimkou *DataException* (dedí od základnej výnimky programu *JXCollException*).

Dátový typ *boolean* je reprezentovaný ako binárna hodnota, nadobúdajúca len dva stavy: "pravda" a "nepravda" [5]. Na základe tejto definície je hodnota 0 uvažovaná ako "nepravda" a hodnota ako "pravda". Všetky ostatné hodnotu sú považované za chybu a vyhodí sa výnimka DataException.

Najzávažnejším problémom je nesprávne dekódovanie niektorých časových známok. Celkovo na tento účel existujú v informačnom modeli IPFIX štyri dátové typy:

- dateTimeSeconds,
- dateTimeMilliseconds,
- dateTimeMicroseconds,
- dateTimeNanseconds.

Prvé dva menované údajové typy obsahujú bezznamienkové číslo obsahujúce počet sekúnd, resp. počet milisekúnd od Unix epochy (1.1.1970 00:00 UTC). údajové typy *dateTimeMicroseconds* a *dateTimeNanoseconds* však používajú na uloženie informácie o čase špeciálnu štruktúru časovej známky vo formáte NTP Timestamp [18]. Ide o 64bitové číslo, kde horných 32 bitov reprezentuje počet sekúnd od stanovenej epochy (1.1.1900 00:00 UTC) a zvyšok predstavuje desatinnú časť sekúnd. Táto štruktúra umožňuje uložiť informáciu o čase s presnosťou až na 232 pikosekúnd. Je potrebné túto štruktúru uložiť do databázy, resp. poslať cez protokol ACP v nezmenenej podobe. Interpretovať túto hodnotu musí až analyzujúca aplikácia. Nesprávne dekódovanie spočíva v interpretovaní týchto údajových typov ako bezznamienkové celé číslo. Číslo sa teda musí dekódovať ako dátový typ *long* (znamienkové číslo). Analyzujúca aplikácia môže na prácu s uloženým údajom použiť napríklad knižnicu *Apache Commons Net* [20], ktorá obsahuje triedu na prácu s NTP časovou známkou.

5.2 Redukované kódovanie informačných elementov

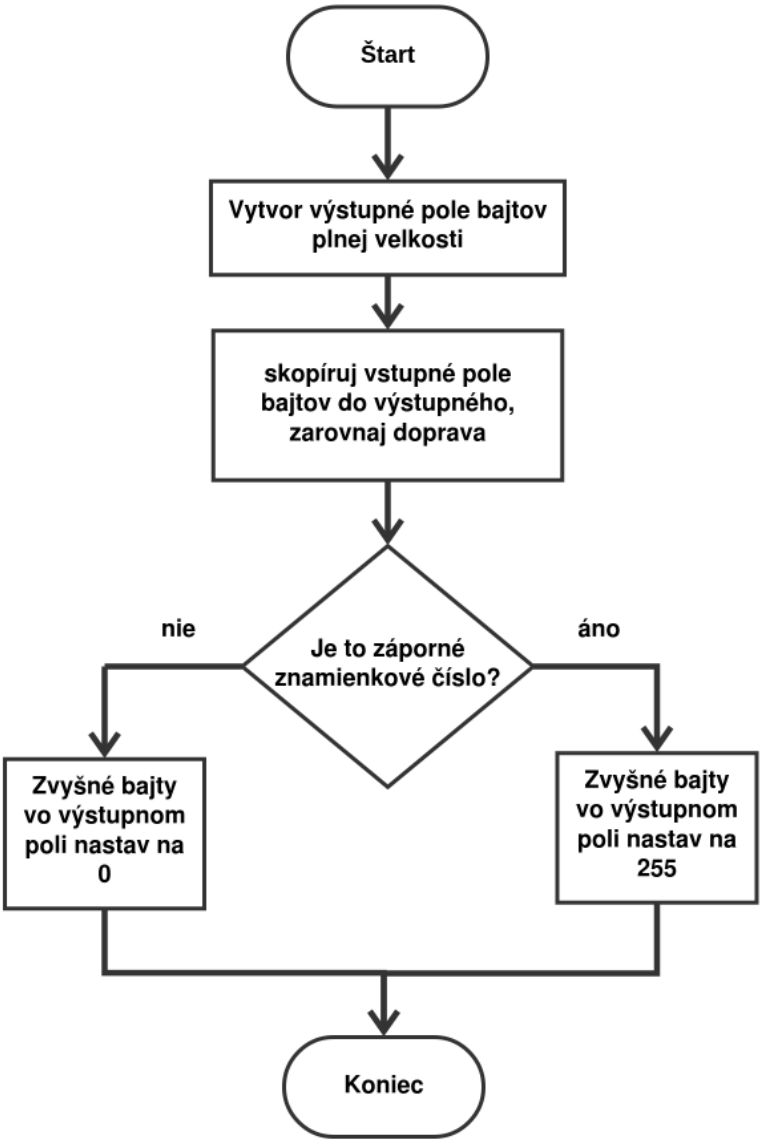
Na to, aby sa použilo redukované kódovanie informačných elementov, je potrebné v šablóne uviesť veľkosť informačného elementu menšiu, ako je jeho maximálna veľkosť definovaná v informačnom modeli IPFIX. Na základe tejto informácie sa z údajového záznamu vyberie časť dát pre každý informačný element. Dekóder dát testuje kapacitu predaného buffra voči definícii. Štandardne, ak je veľkosť iná, ako uvedená v definícii typu, volajúca metóda je informovaná výnimkou *DataException* s informáciami o predanej a očakávanej veľkosti dát.

Pri bezznamienkových aj znamienkových celočíselných typoch pri veľkosti dát menšej, ako je udávaná v definícii typu, musí sa vytvoriť buffer štandardnej (úplnej) veľkosti a obsah dát sa musí do tohto buffra skopírovať s požiadavkou, aby sa význam čísla nezmenil. Pri skracovaní informačných elementov sa odstraňujú len počiatočné nuly.

Uvedené sa dosiahne tak, že najprv sa vytvorí výstupné pole bajtov plnej veľkosti daného dátového typu. Z pravej strany vstupného poľa sa skopírujú jednotlivé bajty a ukladajú sa taktiež sprava do výstupného poľa bajtov plnej veľkosti. Zvyšné bajty sa nastavia na hodnotu 0. Ak je číslo znamienkové, a ak je nastavený najvýznamnejší bit najľavejšieho bajtu (znamienko) vstupného poľa, priradí sa zvyšným bajtom hodnota 255. Tento algoritmus je uvedený na Obrázku 5 – 1.

Dôvodom je skutočnosť, že záporné číslo v doplnkovom kóde má oproti číslu v priamom kóde všetky bity sprava, až po prvú jednotku totožné, a zvyšné bity okrem znamienkového sú invertované. Ak sa posunie znamienkový bit na správnu pozíciu, teda na najľavejší bit poľa bajtov plnej veľkosti daného údajového typu, všetky zvyšné bity musia byť pre záporné číslo nastavené na 1. Ak je znamienkové číslo kladné, platí pre neho to, čo pre bezznamienkové číslo, keďže kladné číslo v doplnkovom kóde má rovnakú podobu ako v priamom kóde.

Po získaní údajov správnej veľkosti ich možno štandardným spôsobom interpretovať.



Obrázok 5 – 1 Algoritmus doplnenia veľkosti informačného elementu

V príklade (a) na Obrázku 5 – 2 je uvedený o dátový typ *unsigned16*, ktorého dáta boli skrátené na jeden bajt. Po predradení čísla nulovým bajtom je možné ho dekódovať ako *unsigned short*. Keďže programovací jazyk Java nepozná bezznamienkové typy, tak musí byť číslo vhodne uložené do najbližšieho väčšieho typu, v tomto prípade do typu *int*. Na tieto účely slúži už implementovaná trieda Support, obsahujúca metódy na získanie bezznamienkových verzií dátových typov. Príklad (b) ukazuje

prevod čísla s rovnako nastavenými bitmi, ale dátový typ týchto dát je *signed16*. Keďže je znamienkový bit nastavený, jedná sa o záporné číslo. Predradí sa teda bajtom o hodnote 255. Stačí interpretovať toto výsledné pole bajtov ako dátový typ *short*.

Keďže číslo môže byť skrátené o akýkoľvek počet bajtov, tento prístup je všeobecný a použiteľný na každý možný prípad. Tento algoritmus vykonáva metóda *handleReducedSizeEncoding*(), ktorá očákáva vstupné pole bajtov, výsledný počet bajtov, a pravdivostnú hodnotu udávajúcu, či je číslo znamienkové alebo nie. Táto metóda je volaná pred dekódovaním údajového typu, ak je jeho veľkosť menšia ako sa očakáva.

Obrázok 5 – 2 Príklad redukovaného kódovania znamienkového (a) a bezznamienkového (b) čísla.

Skrátenie elementov s číslami s pohyblivou rádovou čiarkou je tiež možné. Desatinné číslo s dvojitou presnosťou (*float64*, v Jave typ *double*) môže byť skrátené z ôsmich na štyri bajty. Vtedy sa dekóduje ako desatinné číslo s jednoduchou presnosťou (*float32*, v Jave typ *float*). Treba ale počítať so stratou presnosti, kvôli menšej mantise. Všetky ostatné veľkosti vstupu iné ako 4 alebo 8 bajtov sú považované za chybnú veľkosť a je o tom informovaná volajúca metóda vyhodením výnimky *DataException*.

5.3 Organizáciou definované informačné elementy

Ako bolo spomenuté v predchádzajúcej kapitole, trieda *IpfixElements* slúži ako zdroj informácií o jednotlivých informačných elementoch, ktoré sú popísané v súbore *ipfixFields.xml*. Obsahuje kolekciu *HashMap*, ktorá ako kľúč obsahuje len číslo infor-

mačného elementu. Hodnotou je objekt popisujúci konkrétny informačný element. Keďže organizáciou definovaný informačný element môže mať rovnaký identifikátor ako štandardný IETF informačný element, evidencia len na základe identifikátora je nepostačujúca. Preto sa kľúč v tejto kolekcii zmenil a použil sa objekt pozostávajúci z identifikátora informačného elementu a čísla organizácie. Tento objekt bol nazvaný *FieldKey*. V prípade štandardného IETF informačného elementu sa použije číslo organizácie rovné nule. Všetky metódy na prístup k informáciám o informačnom elemente (názov, údajový typ, skupina) museli byť doplnené o argument definujúci číslo organizácie. Zároveň metódy v triedach RecordDispatcher a ACPIPFIXWorker využívajúce triedu IpfixElements museli byť prispôsobené tejto skutočnosti. Pri dekódovaní údajov v dátovom zázname sa odteraz uvažuje aj s číslom organizácie. Informačné elementy už nemôžu byť zamenené. Pri príjme organizáciou definovaného informačného elementu, ktorá nie je uvedená v *ipfixFields.xml* sa dekódovanie tohto informačného elementu preskočí.

5.4 Optimalizácia IPFIX parsera

Parser IPFIX správ bol súčasťou a hlavnou pracovnou náplňou vlákna NetXMLParser. Toto vlákno vyberá nazhromaždené IPFIX správy z PacketCache a priamo ich spracováva. Keďže sa už očakáva príjem dát nielen cez transportný protokol UDP, ale aj cez protokoly TCP a SCTP, je vhodné, aby existoval jeden objekt slúžiaci na spracovávanie IPFIX správ. Ak by ich bolo viacero, ich údržba by bola náročná. Nový parser je nutné od základu prepísať, pretože pôvodný je veľmi neprehľadný (celé spracovanie bolo umiestnené v jednej metóde) a nekonzistentný. Nekonzistentný napríklad v tom zmysle, že údaje z hlavičiek IPFIX správy boli raz interpretované ako bezznamienkové, inokedy ako znamienkové čísla. Toto bolo potrebné zmeniť. Všetky čísla z hlavičiek sú v novom parseri považované za bezznamienkové.

Nový parser dostal meno *IpfixParser* a celkový problém sa rozdelil na metódy rie-

šiace jednotlivé podproblémy. Existuje hlavná metóda na spracovanie celej správy, *parseIpfixMessage()*. Tá po prečítaní hlavičky volá metódu na spracovanie sady, *parseIpfixSet()*, až kým sa pozícia v buffri nedostane na koniec prijatých dát. Na základe druhu sady sa spracovanie deleguje na metódy zaoberajúce sa spracovaním daného druhu záznamu, až kým pozícia v buffri nedosiahne koniec sady.

V prípade dátovej sady prevezme spracovanie metóda *parseIpfixDataRecord()*, sadu šablón spracováva metóda *parseIpfixTemplateRecord()* a sadu šablón možností metóda *parseIpfixOptionsTemplateRecord()*.

Kód je omnoho prehľadnejší a ľahšie udržiavateľný. Každá metóda vracia príslušný objekt záznamu, ktorý sa vkladá do nadradeného objektu (napríklad dátový záznam do sady, sada do správy a pod.). Výslednú štrutkúru je potom možné prehľadne spracovať.

5.4.1 Viacero záznamov v sade

Ako bolo spomenuté v analýze súčasného stavu programu JXColl, nepredpokladalo sa, že niekedy bude prijatých viac záznamov v sade, ako jeden. Toto je neprípustné. V starom parseri neexistoval cyklus, ktorý by opakovane vyberal záznamy zo sady, až kým sa nedôjde na jej koniec. Očakávalo sa len, že v správe bude existovať viacero sád. V metóde *parseIpfixSet()* je nutné pre každý druh sady vložiť cyklus, ktorý by testoval aktuálnu pozíciu v buffri. Ak je rozdiel aktuálnej pozície a pozície začiatku sady menší ako celková veľkosť sady, je nutné pokračovať v spracovávaní záznamov daného typu.

Podľa druhu sady, metódy zodpovedné za získanie objektu daného záznamu pridajú do objektu sady získané záznamy. Potom sa všetky záznamy sady interpretujú.

5.4.2 Detekcia poškodených správ

Je nevyhnutné zisťovať, či správa nie je poškodená. Jednou z možných chýb môže byť údaj o veľkosti v hlavičke IPFIX správy (pozri Obrázok 2 – 1), ktorý nezodpovedá skutočnej veľkosti prijatých dát. Stávalo sa, že bola prijatá správa, ktorej veľkosť podľa hlavičky mala byť 630 bajtov, no skutočne sa prijalo len 20 bajtov. Takáto správa spôsobovala haváriu JXColl. Pôvodná verzia JXColl mala vyhradené pole bajtov o veľkosti 65540 bajtov pre každú prijatú IPFIX správu. Toto pole bajtov bolo priradené do objektu *PacketObject*, ktoré okrem týchto dát obsahovalo aj zdrojovú IP adresu UDP paketu. *PacketObject* sa ukladá v *PacketCache*. Nebolo však možné určiť skutočnú dĺžku prijatých dát, keďže pole bajtov neobsahuje žiadne indikátory limitu alebo pozície.

Výhodnejšie je uložiť prijaté dáta ako objekt *ByteBuffer*. Po prijatí všetkých dát sa buffer otočí pomocou metódy flip() a aktuálna pozícia sa zmení na limit (pravú hranicu prijatých dát) a pozícia sa nastaví na 0. Takto možno pomocou metódy *remaining()* nad týmto buffrom zistiť, koľko bajtov bolo skutočne prijatých, resp. koľko ešte zostáva do konca.

IpfixParser je týmto spôsobom schopný zistiť veľkosť prijatých dát. V prípade, ak bol prijatý iný počet bajtov, ako uvádza hlavička, parser vyhodí výnimku *DataFormatException* (dedí od základnej výnimky programu *JXCollException*). Vlákno spracúvajúce UDP správy pri príjme tejto výnimky od parsera ukončí spracovanie aktuálnej správy, je vyhlásená chyba na konzolu a pokračuje sa v spracovaní ďalšej správy, ak je prítomná v *PacketCache*.

Ďalšou štruktúrou v rámci IPFIX správy je sada. Je potrebné otestovať, či zostávajúca časť v buffri je postačujúca na prečítanie celej sady. Veľkosť sady je uvedená v hlavičke sady, ako bolo ukázané na Obrázku 2 – 3. Pri nedostatočnom počte voľných bajtov sa opäť vyhodí výnimka *DataFormatException*.

Ak by bol záznam väčší, ako zostávajúca časť sady, tiež je nutné túto situáciu de-

tekovať. Pri vyberaní špecifikátorov polí zo záznamu šablóny sa teda otestuje, či je dostatok zvyšného miesta v sade. Ak nie, vyhodí sa výnimka *DataFormatException*. Podobne pri výbere dátového záznamu, ak nie je dostatok miesta v dátovej sade na výber požadovaného množstva dát, ohlási sa to touto výnimkou. Trieda *IPFIXSet* bola v súvislosti s týmito testami obohatená o možnosti nastavenia a získania začiatočnej a koncovej pozície sady. Vlákno využívajúce *IpfixParser* sa na základe prijatej výnimky rozhodne, aký bude ďalší postup.

5.4.3 Variabilná veľkosť informačných elementov

V metóde *parseIpfixDataRecord()* sa testuje veľkosť informačného elementu, tak ako je daná v špecifikátore poľa v zázname šablóny. Informačné elementy s variabilnou dĺžkou nemajú informáciu o dĺžke zakódovanú v šablóne, ale priamo v dátovej časti informačného elementu. Kolektor sa dozvie o takomto informačnom elemente, ak dĺžka informačného elementu v zázname šablóny má hodnotu 65535.

Samotná dĺžka informačného elementu sa zistí získaním nasledujúceho bajtu v buffri obsahujúcom dátový záznam. Ak je hodnota tohto bajtu menšia ako 255, z buffra sa vyberie počet bajtov určený touto hodnotou. Ak by bola veľkosť väčšia ako 255, vyberú sa ešte ďalšie dva bajty, ktoré sa interpretujú ako bezznamienkové 16-bitové číslo. Na základe tejto hodnoty sa vyberie daný počet bajtov.

Toto vylepšenie umožňuje napríklad spracovať textové reťazce (dátový typ *string*) alebo reťazce oktetov (dátový typ *octetArray*), ktoré majú variabilnú veľkosť. Pred implementáciou tohto mechanizmu by prípadný príjem takýchto dát spôsobil haváriu programu JXColl, pretože by sa snažil vybrať z dátovej časti 65535 bajtov.

5.5 Správa šablón

Evidencia správy šablón vyžaduje dva rozdielne prístupy v závislosti od použitého transportného protokolu.

5.5.1 Správa šablón pre protokol UDP

Od prenosu údajov prostredníctvom protokolu UDP sa očakáva, že šablóny budú posielané v pravidelných intervaloch. Je potrebné implementovať mechanizmus expirácie šablón. Za týmto účelom je nutné modifikovať viaceré časti programu. V prvom rade je potrebné zisťovať čas prijatia každej IPFIX správy a spolu s týmto časom ju uložiť do *PacketCache*. Za týmto účelom musí byť rozšírená trieda *PacketObject* o údaj príchodu správy. Modifikácie si vyžiadali aj ďalšie triedy, ktoré pristupujú do *PacketCache*. Čas poslednej aktualizácie musí byť doplnený do triedy *IPFIXTemplateRecord*, ktorá reprezentuje záznam šablóny. Zároveň je nevyhnutné túto triedu doplniť o metódu *isValid()*, ktorá vyhodnotí aktuálnosť šablóny. Ak je aktuálny čas väčší ako čas poslednej aktualizácie zvýšený o nakonfigurovanú životnosť šablón, znamená to, že šablóna expirovala. Na základe tohto výsledku je možné vyhodnotiť aktuálnosť šablóny.

Je nutné zmeniť samotnú triedu spravujúcu šablóny, *IPFIXTemplateCache*. Je neprehľadná a neposkytuje niektoré funkcie, napríklad spomenutú expiráciu šablón alebo rozlíšenie viacerých inštancií exportérov bežiacich na jednom stroji.

Nová trieda sa bude volať *IpfixUdpTemplateCache*, pretože spravuje len šablóny prijaté pomocou protokolu UDP. V programe bude existovať len jedna inštancia tejto triedy, preto bude použitý návrhový vzor Singleton. Trieda by mala obsahovať podobne ako predchádzajúca trieda kolekciu typu *Map*, ktorá bude reprezentovať jednotlivé exportéry. Kľúčom v tejto kolekcii by mal byť objekt definujúci konkrétny exportér, uvažujúc jeho IP adresu, zdrojový UDP port a číslo pozorovacej domény. Tento objekt vznikne pridaním zdrojového portu do definície triedy *KeyObject*, ktorý

sa premenuje na *ExporterKey*, aby názov bol viac informatívny.

V pôvodnej triede boli hodnotami kolekcie *Hashtable* ďalšie kolekcie *Hashtable*, čo nepridávalo na prehľadnosti a zrozumiteľnosti riešenia. Preto bude vytvorená nová trieda *TemplateHolder*, ktorá bude spravovať všetky šablóny pre daný kľúč exportéra. Bude obsahovať kolekciu *HashMap*, kde kľúčom bude číslo šablóny a hodnotou bude samotný záznam šablóny, *IPFIXTemplateRecord*.

TemplateHolder by mal poskytovať metódy na vloženie šablóny, vymazanie šablóny, vymazanie všetkých šablón, overenie dostupnosti šablóny, získanie konkrétnej alebo všetkých šablón a tiež na zistenie, či je TemplateHolder prázdny.

Vlákna spracúvajúce dáta by mali pristupovať len k triede *IpfixUdpTemplateCache*, ktorá poskytuje metódy na pridanie šablóny, overenie dostupnosti šablóny, výber šablóny s konkrétnym číslom od konkrétneho exportéra, alebo aj všetkých šablón, ktoré tento exportér poslal.

Metóda *checkForPresence()* slúžiaca na zistenie dostupnosti šablóny musí zisťovať stav platnosti šablóny. Ak je expirovaná, musí vratiť false, a tiež vymazať šablónu z cache. Dátový záznam preto nebude môcť byť spracovaný a bude zahodený, čo bude oznámené používateľovi.

Tento princíp expirovania šablón so sebou prináša jeden problém. Ak by exportér prestal posielať akékoľvek IPFIX správy, nikdy by nedošlo k vymazaniu záznamu šablóny. JXColl je aplikácia, pri ktorej sa predpokladá, že bude spustená dlhodobo. Ak by na JXColl posielalo údaje krátkodobo mnoho exportérov, všetky ich šablóny by boli stále dostupné v programe, aj po ukončení prenosu. To by mohlo po istom čase spôsobiť vyčerpanie voľnej pamäte programu a došlo by k pádu JXColl.

Preto bolo vytvorené špeciálne vlákno. Toto vlákno každých 10 minút prehľadáva úložisko v *IpfixUdpTemplateCache* na expirované šablóny. Ak takú nájde, odstráni ju. Ak po odstránení šablóny ostane *TemplateHolder* pre daný exportér prázdny, vymaže sa z evidencie aj tento exportér. Vlákno sa aktivuje pri spustení programu (ak

je zapnutý príjem prostredníctvom UDP). Týmto spôsobom sa zabráni neželanému zapĺňaniu pamäte programu.

5.5.2 Správa šablón pre protokoly TCP a SCTP

Na správu šablón prijímaných cez spojovo orientovaný protokol nie je nutné evidovať šablóny na základe kľúča pozostávajúceho z IP adresy a portu exportéra. V prípade protokolu SCTP, asociácia môže obsahovať aj viacero IP adries. Preto, každé spojenie (asociácia) bude pracovať so svojim vlastným úložiskom šablón, v ktorom kľúčom bude len číslo pozorovacej domény. Trieda na správu šablón pre tieto protokoly sa bude nazývať *IpfixSingleSessionTemplateCache*. Samotné šablóny budú uložené v už spomenutom objekte *TemplateHolder*.

Na rozdiel od protokolu UDP, pri protokoloch TCP a SCTP sa musí jedna šablóna poslať počas životnosti spojenia (asociácie) iba raz. Nie je teda nutné evidovať čas prijatia šablóny. Pri pokuse o vloženie už existujúcej šablóny, musí sa vyhodiť výnimka *TemplateException* (dedí od základnej výnimky JXCollException).

IpfixParser bol doplnený aj o funkciu rozpoznania špeciálnych šablón určených na zrušenie šablón, Template Withdrawal Messages. Pri príjme takej šablóny je číslo šablóny, ktorú exportér žiada vymazať, uvedené v hlavičke záznamu šablóny a počet polí je rovný 0. Z hlavičky IPFIX správy je možné zistiť číslo pozorovacej domény, pre ktorú je táto šablóna definovaná. Preto IpfixSingleSessionTemplateCache poskytuje metódu na vymazanie šablóny -metóda *remove(int templateId, long od)*. Keďže Template Withdrawal Message môže indikovať aj žiadosť o vymazanie všetkých šablón z danej pozorovacej domény, trieda obsahuje aj metódu slúžiacu na tento účel - metóda *removeAll(long od)*. Ak by bol pokus o zrušenie šablóny, ktorá nie je evidovaná programom JXColl, alebo už bola zrušená, *IpfixParser* vyhodí výnimku *TemplateException*.

5.6 Podpora transportného protokolu TCP

Je potrebné podporovať viacero pripojení pomocou protokolu TCP. Každé pripojenie by malo obhospodarovať samostatné vlákno. Port na pripojenie pomocou protokolu by mal byť nastaviteľný cez konfiguračný súbor. Nastavenie portu v konfigurácii je platné pre všetky transportné protokoly. Cieľom tejto práce s ohľadom na implementáciu transportných protokolov je zabezpečiť, aby mohol JXColl prijímať záznamy o tokoch prostredníctvom všetkých transportných protokolov súčasne. Preto sa v konfiguračnom súbore vytvorí prepínač na zapnutie alebo vypnutie podpory pre daný transportný protokol. Na základe týchto prepínačov sa v programe aktivujú vlákna konkrétneho transportného protokolu. Súčasťou konfigurácie transportných protokolov je nastavenie maximálneho počtu aktívnych pripojení. Za týmto účelom musela byť aktualizovaná trieda Config, ktorá získava informácie z konfiguračného súboru.

Pripojenia sú iniciované exportérmi, JXColl je v roli servera. Preto bolo vytvorené vlákno *TCPServer* obsahujúce cyklus, v ktorom sa čaká na pripojenie exportérov. Ak sa exportér pripojí, *ServerSocket.accept()* vráti objekt Socket, ktorý sa predá konštruktoru vlákna *TCPProcessor*, ktoré zároveň bude spustené. Toto vlákno už prijíma a spracováva správy od konkrétneho exportéra.

Pokiaľ vlákno *TCPServer* nebude prerušené (napríklad pri žiadosti o ukončenie programu), bude čakať na pripojenie ďalších exportérov. Maximálny počet spojení je definovaný v konfiguračnom súbore. Po dosiahnutí maximálneho počtu spojení pripojenie exportéra nebude možné, kým sa niektoré zo spojení nezruší.

Keďže komunikácia prostredníctvom protokolu TCP predstavuje súvislý prúd dát, je potrebné vhodne rozdeľovať prijaté údaje na jednotlivé správy. Za týmto účelom *TCPProcessor* najprv načíta prvé 4 bajty IPFIX správy. Podľa Obrázka 2 – 1, sú to polia číslo verzie a dĺžka. Ak je verzia rovná hodnote 10, je to IPFIX správa a môže sa teda pokračovať ďalej, inak sa spojenie zruší. Ostáva prečítať toľko bajtov,

koľko je uvedených v pole dĺžka mínus 4 práve načítané bajty. Výsledok sa predá objektu *IpfixParser* na spracovanie. Celý proces sa opakuje až kým exportér neuzavrie spojenie, prípadne kým nepríde požiadavka na ukončenie programu.

Vlákno odchytáva výnimky TemplateException a DataFormatException. V prípade zachytenia niektorej z týchto výnimiek, spojenie je ukončené poslaním RST segmentu, správa zahodená a vlákno zrušené. Exportéru je poslaný RST segment.

Pri požiadavke o ukončenie programu by JXColl mal vypnúť svoju stranu každého TCP pripojenia pomocou *Socket.shutdownOutput()* a spracovávať všetky nasledujúce dáta, až kým nezaregistruje ukončenie spojenia zo strany exportéra, príjem hodnoty -1. Problémom je skutočnosť, že protokol IPFIX je navrhnutý tak, že dáta putujú len smerom od exportéra ku kolektoru. Jediným spôsobom, ako môže exportér zistiť, že sa uzavrel druhý koniec spojenia je, že sa bude snažiť čítať dáta od kolektora. Toto však žiadny z exportérov, ktoré budú predstavené v nasledujúcej kapitole, nezachytil. Dodržanie požiadaviek štandardu by viedlo k nemožnosti vypnúť JXColl štandardným spôsobom. Preto sa autor rozhodol ukončiť spojenie RST segmentom. Exportér sa tak o nedostupnosti kolektora dozvie okamžite, inak by sa to dozvedel až pri ďalšom pokuse o poslanie dát. Ako sa uvádza v [13], na poslanie RST segmentu je nutné aplikovať *setLinger(true, 0)* nad objektom socketu a následne ho uzavrieť pomocou volania *close()*.

5.7 Podpora transportného protokolu SCTP

Keďže protokol SCTP je relatívne novým transportným protokolom, Java API (Application Programming Interface) verzie 1.6, ktoré využíval JXColl doposiaľ, neobsahuje jeho podporu. Takisto podpora zo strany operačného systému nie je zaručená. Našťastie existuje projekt LKSCTP (SCTP for the Linux Kernel) [21], ktorý implementoval podporu pre tento transportný protokol a je súčasťou operačného systému Linux od verzie jadra 2.6. Na funkciu programov využívajúcich protokol SCTP je

potrebné aby systém obsahoval balík *lksctp-tools*.

Jorgensen vytvoril pre svoje vlastné potreby knižnicu, ktorú pomenoval Java SCTP [22]. Avšak neimplementoval všetky požiadavky, ktoré uvádza špecifikácia protokolu. Z tohto dôvodu nie je vhodné použiť túto knižnicu v programe JXColl.

V súčasnosti je SCTP API prístupná Oracle JDK 1.7. Toto API bolo definované a referenčná implementácia bola vytvorená ako OpenJDK SCTP projekt. Táto práca bola integrovaná do Oracle JDK 7 [24].

Existuje aj postup [25], ako umožniť Jave verzii 6 spolupracovať s SCTP API od OpenJDK, avšak tento proces by bol pre reálne nasadenie konfiguračne príliš náročný. Najvhodnejším riešením bol prechod na OpenJDK 7 (prípadne Oracle JDK 7).

Podobne ako pri protokole TCP, je vhodné použiť vlákno, ktoré bude čakať na SCTP asociácie iniciované exportérmi. Takýmto vláknom je *SCTPServer* a na tento účel využíva objekt *SctpServerChannel*. Java SCTP API [23] umožňuje použiť aj *SctpMultiChannel*, ktorý dokáže obsluhovať viacero asociácií. Avšak kvôli prehľadnosti a zrozumiteľnosti bol použitý prvý spomenutý prístup.

Vlákno *SCTPServer* čaká na požiadavku o vznik asociácie od exportérov pomocou blokujúceho volania *SctpServerChannel.accept()*. Navrátený objekt *SctpChannel* je predaný novému vláknu *SCTPProcessor*, ktorý už pracuje s konkrétnou asociáciou. Toto vlákno prijíma celistvé správy vďaka zachovávaniu hraníc správ protokolom SCTP. Vlákno SCTPServer po vytvorení asociácie naďalej čaká na novú asociáciu. Maximálny počet asociácií je nastaviteľný v konfiguračnom súbore.

SCTPProcessor správy predá svojmu objektu *IpfixParser*, ktorý spracováva samotnú správu. Ak parser vyhodí výnimku TemplateException alebo DataFormatException (v prípade nepovolenej operácie so šablónami alebo pri príjme poškodených dát), vlákno *SCTPProcessor* zahodí správu, zruší asociáciu, a ukončí svoju činnosť. Všetky záznamy šablón sú tak zahodené, pretože každá asociácia používa samos-

tatný *IpfixParser* a samostatnú IpfixSingleSessionTemplateCache. O zrušení alebo nedostupnosti asociácie sa vlákno *SCTPProcessor* dozvie prostredníctvom notifikácií.

Implementácia podpory čiastočnej spoľahlivosti zo strany kolektora vyžaduje len jeho konfiguráciu v LKSCTP. Štandardne je podpora pre PR-SCTP zapnutá. Uistiť sa o nastavení možno prostredníctvom príkazu:

```
echo '1' > /proc/sys/net/sctp/prsctp_enable
```

Vlákna spracúvajúce dáta prijaté protokolom UDP (*UDPProcessor*), TCP (*TCPProcessor*) a SCTP (*SCTPProcessor*) majú svoj vlastný objekt IpfixParser. Podľa toho, aký druh úložiska šablón bol predaný jeho konštruktoru, parser volí spôsob práce so šablónami. Na export dát parser využíva metódu *dispatchIPFIXRecord()* nad objektom *RecordDispatcher*. *RecordDispatcher* bol upravený, aby existovala iba jedna inštancia v programe. Metóda *dispatchIPFIXRecord()* bola nastavená na synchronizovanú, aby dáta z jednotlivých vlákien boli úplne exportované bez prelínania, ktoré by mohlo spôsobiť neželané javy.

6 Overenie funkčnosti vykonaných zmien

V tejto kapitole je overovaná funkčnosť prezentovaných riešení. Za týmto účelom boli použité nasledovné implementácie IPFIX exportérov.

6.1 Nástroj Vermont

Vermont (VERsatile MONitoring Toolkit) [27] je modulárny open-source nástroj na export a spracovanie dát o sieťových tokoch, založených na monitorovanej prevádzke. Podporuje štandardy IPFIX, Netflow v9 a PSAMP (Packet Sampling). Vermont beží na operačných systémoch Linux a FreeBSD.

Exportovací modul podporuje prenos IPFIX správ cez transportné protokoly UDP a SCTP. Umožňuje nastaviť životnosť SCTP správ obsahujúcich dátové záznamy vďaka využitiu čiastočnej spoľahlivosti protokolu SCTP. Vermont má veľmi bohaté možnosti konfigurácie. Je vyvíjaný na FAU Erlangen a TU München.

6.2 Nástroj Maji

Nástroj Maji [28] je open-source implementácia IPFIX meracieho nástroja založená na knižnici *libtrace*, slúžiacej na zachytávanie a spracovanie paketov. Hlavné vlastnosti tohto nástroja patrí tvorba vlastných šablón. Je možné použiť viac než 50 štandardných informačných elementov. Umožňuje export IPFIX správ cez protokoly SCTP, TCP alebo UDP, na štandardný výstup alebo do SQLite databázy. Je vyvíjaný vo WAND Research Group na University of Waikato na Novom Zélande.

6.3 Nástroj YAF

Nástroj YAF (Yet Another Flowmeter) sa používa ako sonda na zachytávanie informácií o tokoch v sieti a export týchto informácií vo formáte IPFIX. Číta údaje

paketov z pcap dump súborov generovaných programom tcpdump, zo živého zaznamenávania na rozhraní použitím knižnice pcap. Následne agreguje tieto pakety do tokov a exportuje ich cez protokoly SCTP, TCP alebo UDP, alebo do serializovaných prúdov IPFIX správ (IPFIX súborov). Natívne podporuje aj meranie obojsmerných tokov.

Je vyvíjaný skupinou CERT NetSA na Carnegie Mellon University.

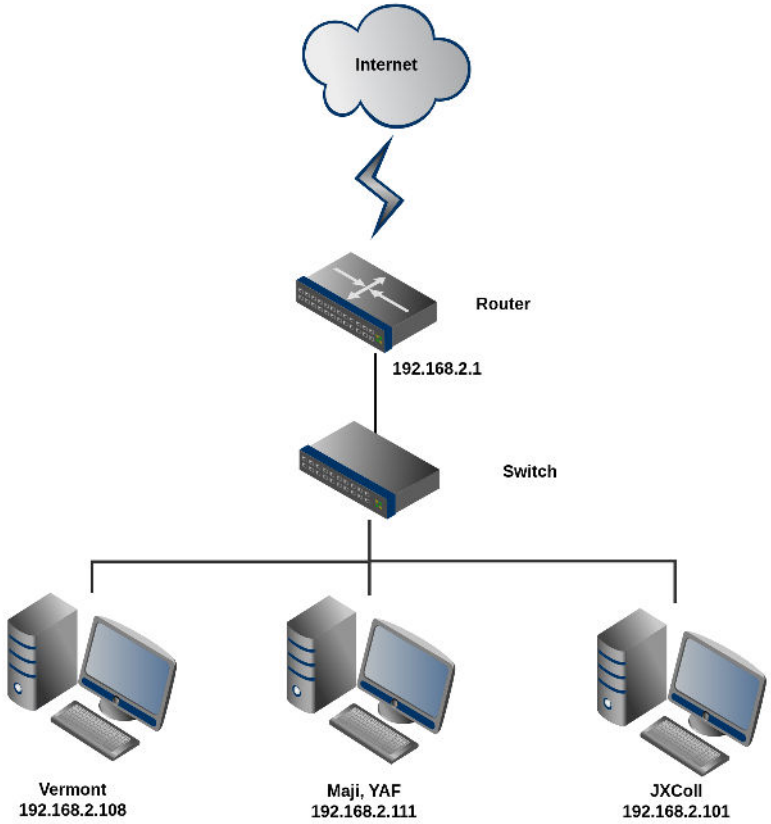
6.4 Testovacia topológia

Vyššie opísané nástroje boli používané už počas nápravy jednotlivých nedostatkov, resp. pri testovaní funkčnosti prenosu údajov pomocou protokolu TCP alebo SCTP. Na Obrázku 6 – 1 možno vidieť testovaciu topológiu. Na jednom počítači bol nainštalovaný nástroj Vermont (verzia 1.0). Použil sa operačný systém Ubuntu 10.04 LTS. Na druhom počítači s operačným systémom Fedora 16 boli nainštalované nástroje YAF (verzia 2.2.2) a Maji (verzia 1.0.2). JXColl bežiaci na treťom počítači bol nainštalovaný na operačnom systéme Ubuntu 11.04. Aby sa otestovali jednotlivé implementované funkcie v tejto práci, boli spúšťané inštancie exportérov podľa požiadaviek konkrétneho testu.

6.5 Priebeh testovania

6.5.1 Správa šablón pre protokol UDP

Na otestovanie správnej funkčnosti správy šablón pre protokol UDP boli spustené dve inštancie exportéra Maji. Obidva posielali šablónu s číslom 257, kde jedna obsahovala 11 informačných elementov a druhá obsahovala 13 informačných elementov. JXColl jednotlivé šablóny korektne uložil a bol schopný dekódovať dáta od oboch exportérov. Viacero inštancií na jednom počítači už teda nepredstavuje problém.



Obrázok 6 – 1 Testovacia topológia

Tieto inštancie mali nastavené rozdielne intervaly posialania šablón (60 a 300 sekúnd). JXColl mal nastavenú expiračnú dobu šablón na 180 sekúnd. Prvá inštancia nemala problém s exportom aj po 180 sekundách, keďže každú minútu sa šabóna aktualizovala. Druhej inštancii však vypršala životnosť šablóny a preto nasledujúce dátové záznamy až po príchod novej šablóny boli zahadzované. Po vypnutí oboch inštancií sa čistiace vlákno postaralo o vymazanie všetkých expirovaných šablón a exportérov z evidencie.

6.5.2 Prenos cez protokol TCP

Keďže protokol TCP podporuje len YAF a Maji, na otestovanie tohto transportného protokolu boli použité súčasne tieto dva nástroje. JXColl umožnil pripojenie

obidvom exportérom. JXColl mal nastavený maximálny počet spojení na hodnotu 3. Pri pokuse o pripojenie ďalšieho exportéra, spojenie nebolo umožnené. JXColl správne rozdeľoval správy a ich dekódovanie prebiehalo bez problémov.

Obidva exportéry správne posielali šablóny iba na začiatku spojenia. Na otestovanie reakcie na opätovné poslanie záznamu šablóny bol použitý nástroj BEEM, ktorý bol modifikovaný na periodický export šablón aj na protokole TCP. JXColl správne zaznamenal chybu, zahodil správu a ukončil spojenie s exportérom. Umožnil tak pripojiť sa ďalšiemu exportéru pomocou protokolu TCP. Chyba počas prenosu cez TCP nemala vplyv na stabilitu ostatných častí programu.

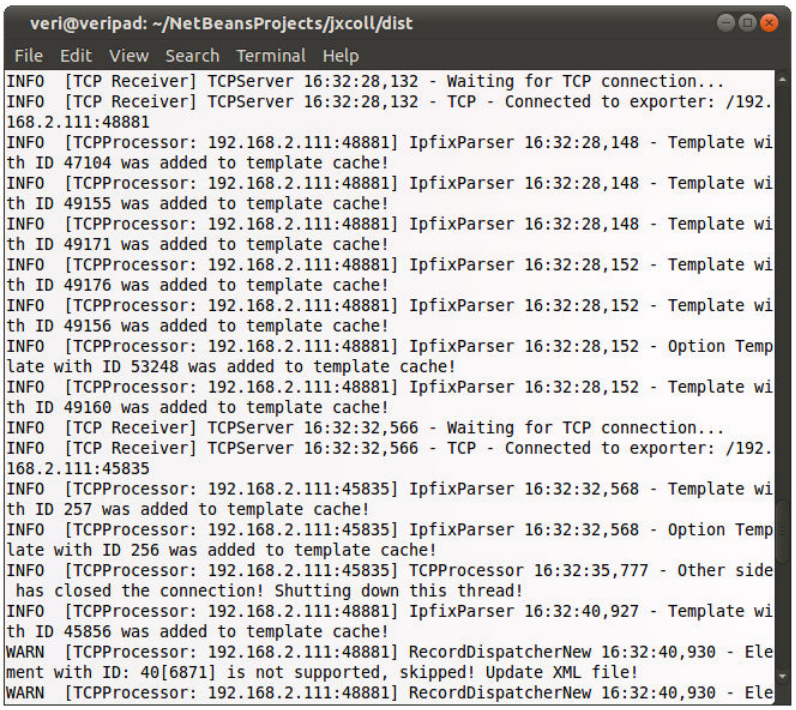
Po vypnutí exportérov, JXColl túto skutočnosť zaznamenal a ukončil činnosť príslušných vlákien. Znova, uvoľnili sa zdroje a ďalším exportérom bolo umožnené sa pripojiť.

Pri požiadavke o vypnutie programu JXColl boli všetky pripojenia zrušené RST segmentom, aby sa exportéry dozvedeli o neprítomnosti spojenia čo najskôr. JXColl sa korektne ukončil.

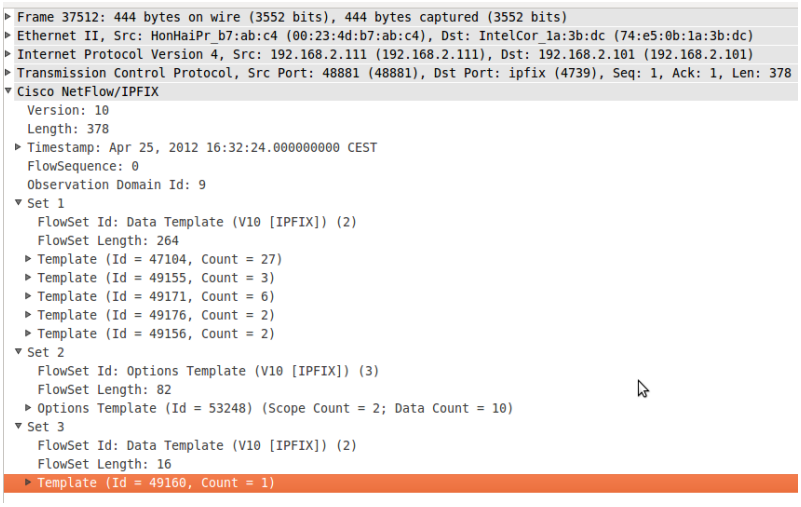
Na Obrázku 6 – 2 je možné hneď na začiatku vidieť pripojenie nástroja YAF. Ten okamžite poslal svoje šablóny. Potom sa pripojil aj exportér Maji, ktorý poslal svoje šablóny. Šablóny boli korektne uložené do cache. Následne bol exportér Maji vypnutý, čo je tiež zobrazené na obrázku. Na Obrázku 6 – 3 je vidieť prijaté dáta zachytené programom Wireshark, ktoré korešpondujú s výpisom JXColl.

6.5.3 Prenos cez protokol SCTP

Export protokolom SCTP umožňujú všetky spomenuté nástroje. Boli vykonané rovnaké testy ako pri protokole TCP. Rovnako ako pri TCP je ohraničený počet aktívnych asociácií. Po dosiahnutí maximálneho množstva asociácií, ako je nastavené v konfiguračnom súbore, nie je možné pripojenie ďalšieho exportéra.



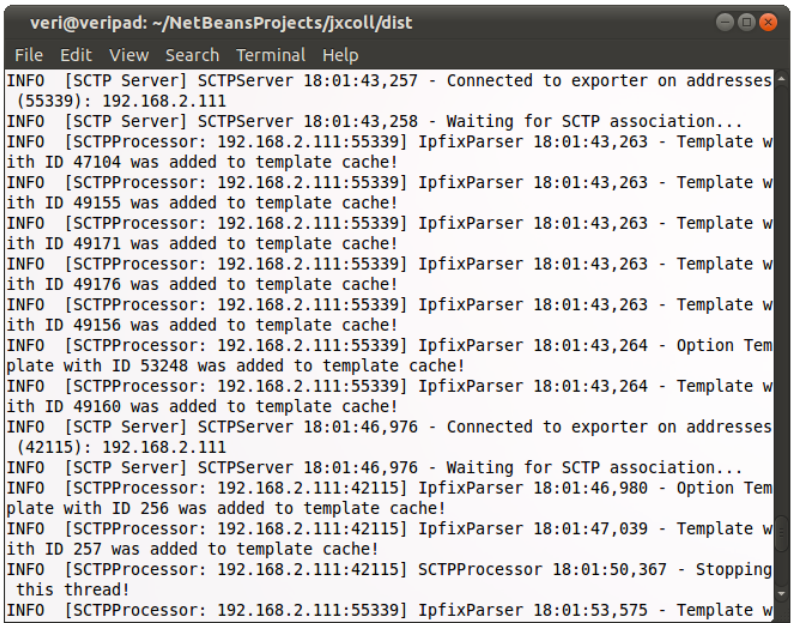
Obrázok 6 – 2 Výstup JXColl pri príjme správ cez protokol TCP



Obrázok 6 – 3 Prijaté správy cez TCP zachytené programom Wireshark

JXColl prijímal dáta od všetkých exportérov. Po ich vypnutí, JXColl ukončil vlákna SCTP procesorov a počúval na nové asociácie. Pri ukončovaní programu JXColl všetky vlákna SCTP procesorov boli prerušené a program sa korektne ukončil. Nástroj Vermont pred zrušením asociácie poslal správu Template Withdrawal Message,

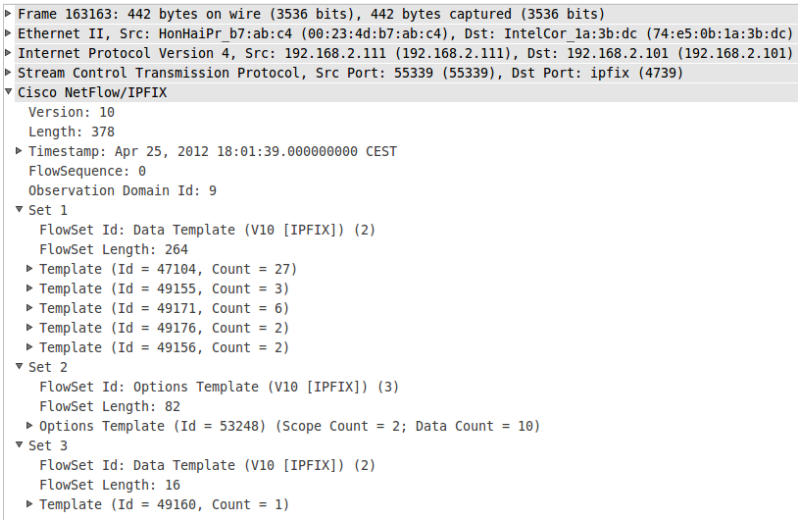
ktorou zrušil aktívnu šablónu. Exportér Vermont sa pokúšal opätovne vytvoriť asociáciu, čo sa po zapnutí JXColl podarilo. Keďže znovu poslal aj šablónu, dátové záznamy boli dekódované. Obrázok 6 – 4 ukazuje výstup programu JXColl pri vytvorení asociácie s exportérom YAF a Maji. Je zobrazená tá istá situácia, ako pri teste s protokolom TCP. Obidva programy boli v tomto príklade spustené len za účelom demonštrovať funkcionalitu podpory protokolu SCTP. Na Obrázku 6 – 5 sú zobrazené dáta korešpondujúce s týmto výpisom, ktoré zachytil program Wireshark. Na Obrázku 6 – 6 sú ukázané vlákna bežiaceho programu JXColl s viacerými TCP spojeniami a SCTP asociáciami.



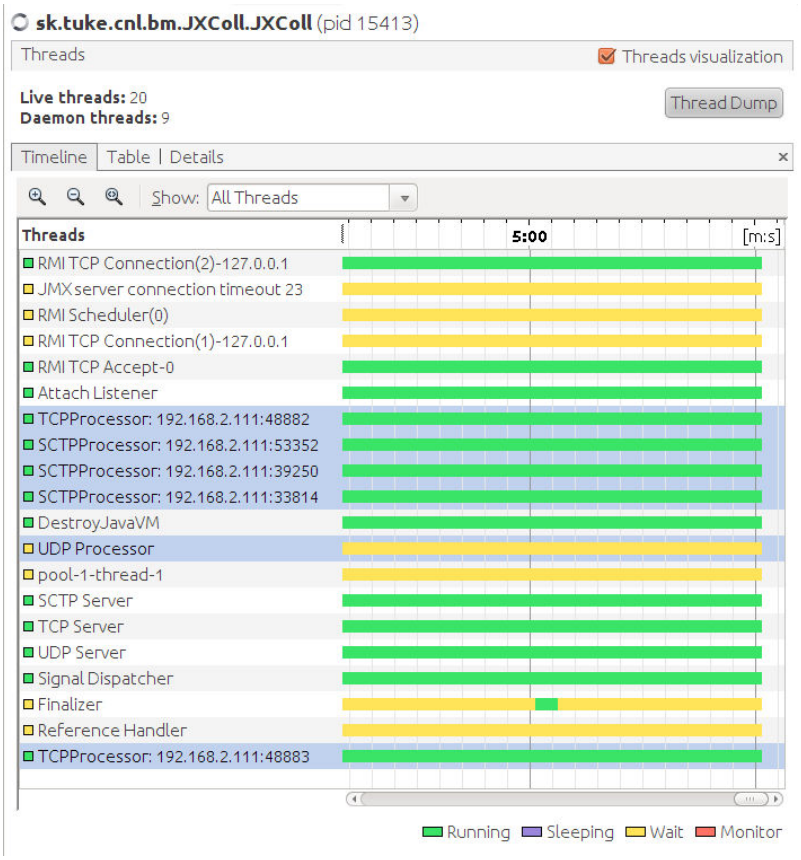
Obrázok 6 – 4 Výstup JXColl pri príjme správ cez protokol SCTP

6.5.4 Zmeny zvyšujúce konformitu so špecifikáciou IPFIX

Na Obrázku 6 – 2, zobrazujúcom prenos pomocou protokolu TCP je vidieť, že informačný element s číslom identifikátora 40 a číslom organizácie 6871 bol preskočený, pretože JXColl nenašiel definíciu takéhoto informačného elementu v súbore *ipfixFields.xml*, napriek tomu, že štandardný informačný element s číslom 40 sa v súbore



Obrázok 6 – 5 Prijaté správy cez SCTP zachytené programom Wireshark



Obrázok 6 – 6 Vlákna bežiaceho programu JXColl s viacerými pripojeniami

nachádza. JXColl sa zachoval korektne, v súlade so špecifikáciou IPFIX. Z tohto obrázku je tiež zrejmé, že JXColl už je schopný spracovať viacero záznamov v sade. V tomto prípade sa jedná o viacero záznamov šablón.

```
DEBUG RecordDispatcher 19:35:01,835 - Processing data within data record!
DEBUG RecordDispatcher 19:35:01,835 - n: flowStartMilliseconds | d: dateTimeMilliseconds | g: main | v: 1335375278526
DEBUG RecordDispatcher 19:35:01,835 - n: flowEndMilliseconds | d: dateTimeMilliseconds | g: main | v: 1335375297743
DEBUG IpfixDecoder 19:35:01,835 - Reduced size encoding is in use!
DEBUG RecordDispatcher 19:35:01,835 - n: octetTotalCount | d: unsigned64 | g: main | v: 19107164
DEBUG IpfixDecoder 19:35:01,835 - Reduced size encoding is in use!
DEBUG RecordDispatcher 19:35:01,835 - n: packetTotalCount | d: unsigned64 | g: main | v: 12741
DEBUG RecordDispatcher 19:35:01,835 - n: sourceIPv4Address | d: ipv4Address | g: main | v: 212.23.6.76
DEBUG RecordDispatcher 19:35:01,835 - n: destinationIPv4Address | d: ipv4Address | g: main | v: 192.168.2.111
DEBUG RecordDispatcher 19:35:01,836 - n: sourceTransportPort | d: unsigned16 | g: main | v: 80
DEBUG RecordDispatcher 19:35:01,836 - n: destinationTransportPort | d: unsigned16 | g: main | v: 35122
WARN  RecordDispatcher 19:35:01,836 - Element with ID: 40[6871] is not supported, skipped! Update XML file!
DEBUG RecordDispatcher 19:35:01,836 - n: protocolIdentifier | d: unsigned8 | g: main | v: 6
DEBUG RecordDispatcher 19:35:01,836 - n: flowEndReason | d: unsigned8 | g: main | v: 4
DEBUG RecordDispatcher 19:35:01,836 - n: tcpSequenceNumber | d: unsigned32 | g: transportHeader | v: 2232177828
WARN  RecordDispatcher 19:35:01,836 - Element with ID: 14[6871] is not supported, skipped! Update XML file!
WARN  RecordDispatcher 19:35:01,836 - Element with ID: 15[6871] is not supported, skipped! Update XML file!
DEBUG RecordDispatcher 19:35:01,836 - n: vlanId | d: unsigned16 | g: subIpHeader | v: 0
ERROR RecordDispatcher 19:35:01,837 - i.e. 'subTemplateMultiList' - Cannot decode datatype: subTemplateMultiList
ERROR RecordDispatcher 19:35:01,837 - Skipping this element DB exportation!
DEBUG RecordDispatcher 19:35:01,837 - Data fields processing finished!
DEBUG RecordDispatcher 19:35:01,837 -  Storing 10 IPFIX fields into table: records_main
DEBUG RecordDispatcher 19:35:01,843 -  Storing 1 IPFIX fields into table: records_subIpHeader
DEBUG RecordDispatcher 19:35:01,853 -  Storing 1 IPFIX fields into table: records_transportHeader
```

Obrázok 6 – 7 Dekódovanie dát od exportéra YAF

```
▼ Flow 2
  ▶ [Duration: 19.217000000 seconds]
    Permanent Octets: 19107164
    Permanent Packets: 12741
    SrcAddr: 212.23.6.76 (212.23.6.76)
    DstAddr: 192.168.2.111 (192.168.2.111)
    SrcPort: 80
    DstPort: 35122
    Enterprise Private entry: (CERT Coordination Center) Type 40: Value (hex bytes): 00 00
    Protocol: 6
    Flow End Reason: Forced end (4)
    TCP Sequence Number: 2232177828
    Enterprise Private entry: (CERT Coordination Center) Type 14: Value (hex bytes): 12
    Enterprise Private entry: (CERT Coordination Center) Type 15: Value (hex bytes): 18
    Vlan Id: 0
  ▶ [Enterprise Private entry: ((null)) Type 293: Value (hex bytes): 00 (Variable Length)]

0350  01 36 ea 91 fc cf 01 23  8d 5c 00 00 31 c5 d4 17   .6.....# .\..1...
0360  06 4c c0 a8 02 6f 00 50  89 32 00 00 06 04 85 0c   .L...o.P .2......
0370  54 a4 12 18 00 00 ff 00  01 00 d0 00 00 44 00 00   T....... .....D..
0380  00 00 00 00 00 0e 00 00  00 00 00 00 3d be 00 00   ........ ....=...
```

Obrázok 6 – 8 Dáta od exportéra YAF zobrazené v programe Wireshark

Dekódovanie informačných elementov s redukovaným kódovaním je možné vidieť na obrázku 6 – 7. Informačné elementy *octetTotalCount* a *packetTotalCount* sú expor-

térom YAF zakódované v 4 bajtoch, namiesto 8. Dôkaz správnosti dekódovania je viditeľný na Obrázku 6 – 8. JXColl si poradil s dekódovaním týchto dát a nevyhlásil žiadnu chybu. Výstup z programu Wireshark dokazuje, že údaje boli správne dekódované.

Na Obrázkoch 6 – 7 a 6 – 8 si možno tiež všimnúť, že informačný element *subTemplateMultiList* bol vynechaný z dekódovania, pretože mu JXColl nerozumel. Tento informačný element variabilnej dĺžky patrí do skupiny informačných elementov slúžiacich na prenos štruktúrovaných dát pomocou protokolu IPFIX. Implementácia týchto nových informačných elementov nebola predmetom tejto práce, avšak podpora pre správne spracovanie informačných elementov s variabilnou dĺžkou je základným predpokladom budúcej podpory aj týchto abstraktných dátových typov. JXColl vybral z dátového záznamu správnu porciu dát a všetky ostatné údaje tohto záznamu o toku boli úspešne uložené do databázy.

Príklad správneho dekódovania dátového typu *dateTimeNanoseconds* od exportéra Vermont je uvedený na Obrázku 6 – 9. Zobrazené číslo je možné z databázy načítať ako typ *long* a pomocou knižnice *Apache Commons Net* jednoducho konvertovať na typ *Date*.

Nástroj Maji síce umožňuje export časovej známky v údajovom type *dateTimeMicroseconds* (Obrázok 6 – 10), ale hodnota je nesprávna. Správne by mal byť čas zakódovaný vo formáte časovej známky *NTP Timestamp*, Maji však uvažuje v tomto dátovom type počet mikrosekúnd od Unix epochy (00:00 1.1.1970 UTC). Na tomto obrázku je vidieť aj správne dekódovanie typu *macAddress*.

```
DEBUG RecordDispatcher 23:42:36,313 - Processing data within data record!
DEBUG RecordDispatcher 23:42:36,313 - n: sourceIPv4Address | d: ipv4Address | g: main | v: 192.168.2.109
DEBUG RecordDispatcher 23:42:36,313 - n: sourceIPv4PrefixLength | d: unsigned8 | g: ipHeader | v: 32
DEBUG RecordDispatcher 23:42:36,313 - n: destinationIPv4Address | d: ipv4Address | g: main | v: 192.168.2.255
DEBUG RecordDispatcher 23:42:36,313 - n: destinationIPv4PrefixLength | d: unsigned8 | g: ipHeader | v: 32
DEBUG RecordDispatcher 23:42:36,313 - n: protocolIdentifier | d: unsigned8 | g: main | v: 17
DEBUG RecordDispatcher 23:42:36,313 - n: sourceTransportPort | d: unsigned16 | g: main | v: 17500
DEBUG RecordDispatcher 23:42:36,313 - n: destinationTransportPort | d: unsigned16 | g: main | v: 17500
DEBUG RecordDispatcher 23:42:36,313 - n: flowStartNanoseconds | d: dateTimeNanoseconds | g: timestamp | v: -3223750425978780594
DEBUG RecordDispatcher 23:42:36,313 - n: flowEndNanoseconds | d: dateTimeNanoseconds | g: timestamp | v: -3223750425978780594
DEBUG RecordDispatcher 23:42:36,313 - n: octetDeltaCount | d: unsigned64 | g: main | v: 196
DEBUG RecordDispatcher 23:42:36,313 - n: packetDeltaCount | d: unsigned64 | g: main | v: 1
DEBUG RecordDispatcher 23:42:36,313 - n: tcpControlBits | d: unsigned8 | g: minMax | v: 0
DEBUG RecordDispatcher 23:42:36,313 - Data fields processing finished!
DEBUG RecordDispatcher 23:42:36,313 -  Storing 7 IPFIX fields into table: records_main
DEBUG RecordDispatcher 23:42:36,328 -  Storing 1 IPFIX fields into table: records_minMax
DEBUG RecordDispatcher 23:42:36,336 -  Storing 2 IPFIX fields into table: records_timestamp
DEBUG RecordDispatcher 23:42:36,345 -  Storing 2 IPFIX fields into table: records_ipHeader
```

Obrázok 6 – 9 Dekódovanie údajov od exportéra Vermont

```
DEBUG RecordDispatcher 14:05:59,432 - Processing data within data record!
DEBUG RecordDispatcher 14:05:59,432 - n: sourceIPv4Address | d: ipv4Address | g: main | v: 192.168.2.111
DEBUG RecordDispatcher 14:05:59,432 - n: destinationIPv4Address | d: ipv4Address | g: main | v: 192.168.2.101
DEBUG RecordDispatcher 14:05:59,432 - n: protocolIdentifier | d: unsigned8 | g: main | v: 132
DEBUG RecordDispatcher 14:05:59,432 - n: sourceTransportPort | d: unsigned16 | g: main | v: 56697
DEBUG RecordDispatcher 14:05:59,432 - n: destinationTransportPort | d: unsigned16 | g: main | v: 4739
DEBUG RecordDispatcher 14:05:59,432 - n: packetTotalCount | d: unsigned64 | g: main | v: 2
DEBUG RecordDispatcher 14:05:59,432 - n: octetTotalCount | d: unsigned64 | g: main | v: 160
DEBUG RecordDispatcher 14:05:59,432 - n: sourceMacAddress | d: macAddress | g: subIpHeader | v: 00:0C:29:98:47:F3
DEBUG RecordDispatcher 14:05:59,432 - n: destinationMacAddress | d: macAddress | g: subIpHeader | v: 74:E5:0B:1A:3B:DC
DEBUG RecordDispatcher 14:05:59,432 - n: flowStartMilliseconds | d: dateTimeMilliseconds | g: main | v: 1335441925155
DEBUG RecordDispatcher 14:05:59,432 - n: flowEndMilliseconds | d: dateTimeMilliseconds | g: main | v: 1335441926197
DEBUG RecordDispatcher 14:05:59,432 - n: flowStartMicroseconds | d: dateTimeMicroseconds | g: timestamp | v: 1335441925155018
DEBUG RecordDispatcher 14:05:59,432 - n: flowEndMicroseconds | d: dateTimeMicroseconds | g: timestamp | v: 1335441926197917
DEBUG RecordDispatcher 14:05:59,432 - Data fields processing finished!
DEBUG RecordDispatcher 14:05:59,432 -  Storing 9 IPFIX fields into table: records_main
DEBUG RecordDispatcher 14:05:59,440 -  Storing 2 IPFIX fields into table: records_subIpHeader
DEBUG RecordDispatcher 14:05:59,448 -  Storing 2 IPFIX fields into table: records_timestamp
```

Obrázok 6 – 10 Dekódovanie údajov od exportéra Maji

7 Záver

Hlavné ciele tejto práce pozostávajú zo zvýšenia konformity zhromažďovacieho procesu nástroja BasicMeter so špecifikáciou IPFIX, súčasne s jeho rozšírením o podporu transportných protokolov TCP a SCTP.

V rámci zvýšenia konformity zhromažďovacieho procesu nástroja BasicMeter bolo nutné vyriešiť viacero problémov, ktoré sa prejavili na vzájomnom testovaní IPFIX nástrojov v roku 2011. Jedným z problémov bola absencia podpory viacerých záznamov v sade šablón, sade šablón možností a v dátovej sade. Vážnym problémom bola neschopnosť programu JXColl spracovať informačné elementy s redukovaným kódovaním, variabilnou veľkosťou, či organizáciou definované informačné elementy s identifikátorom zhodným s niektorým štandardným informačným elementom. V práci bol predstavený návrh a implementácia riešení uvedených nedostatkov. Navyše boli implementované doposiaľ nepodporované údajové typy *macAddress*, *boolean* a *octetAray* spoločne s nápravou interpretácie údajových typov *dateTimeMicroseconds* a *dateTimeNanoseconds*.

Výsledkom práce je aj rozšírenie funkcionality programu JXColl o prenos údajov o tokoch pomocou transportných protokolov TCP a SCTP, čo prináša možnosť nasadenia nástroja v prostredí so zvýšenou náchylnosťou na preťaženie. V rámci tejto úlohy bolo nutné zmeniť mechanizmus správy šablón v závislosti od druhu transportného protokolu. Pri prenose údajov pomocou protokolu UDP bola zavedená funkcia expirácie šablón a súčasne bolo umožnené spracovanie dát od viacerých exportérov bežiacich na jednom počítači. Expirácia šablón zabraňuje zadržiavaniu záznamov šablón v pamäti počas dlhodobého behu programu.

Správa šablón pre protokoly TCP a SCTP zohľadňuje spojovo orientovanú povahu prenosu údajov. Šablóny sú platné len pre konkrétne TCP pripojenie alebo SCTP asociáciu a bola implementovaná podpora pre ich zrušenie exportérmi pomocou Template Withdrawal správ. Podľa požiadaviek protokolu IPFIX, prijatie poškode-

ných dát, už evidovanej šablóny alebo zrušenie neexistujúcej šablóny vyústi k zrušeniu TCP pripojenia alebo SCTP asociácie.

Predstavené zmeny boli overené experimentmi s inými implementáciami IPFIX exportérov (Maji, YAF a Vermont). Na základe týchto experimentov bola dokázaná funkčnosť prezentovaných riešení. Možným budúcim smerovaním vývoja zhromažďovacieho procesu nástroja BasicMeter môže byť implementácia podpory nových abstraktných údajových typov určených na prenos štruktúrovaných dát protokolom IPFIX či podpora zabezpečeného prenosu údajov o tokoch.

Zoznam použitej literatúry

[1] Claise, B.: *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information.* RFC 5101. 2008.

[2] Brownlee, N.: *Flow-Based Measurement: IPFIX Development and Deployment.* IEICE TRANS. COMMUN., VOL.E94–B, NO.8. 2011.

[3] Mentz, D.: *Secure and efficient transmission of traffic measuring data.* Diplomová práca. Mníchov: TU München FI, 2010. 48 s.

[4] Juvhaugen, P.: *Exporting IP flows using IPFIX.* Diplomová práca. Oslo: Oslo University College FI, 2007. 69 s.

[5] Quittek, J. et al.: *Information Model for IP Flow Information Export.* RFC 5102. 2008.

[6] Boschi, E. et al.: *IP Flow Information Export (IPFIX) Implementation Guidelines.* RFC 5153. 2008.

[7] Výskumná skupina MONICA: *Nástroj BasicMeter [online].* 2012. [cit. 2012-04-10]. Dostupné na internete: <`http://wiki.cnl.sk/Monica/BasicMeter`>.

[8] Pekár, A.: *Monitorovanie prevádzkových parametrov siete v reálnom čase.* Bakalárska práca. Košice: TUKE, FEI, 2009. 50 s.

[9] Vereščák, T.: *Zhromažďovací proces nástroja BasicMeter.* Bakalárska práca. Košice: TUKE, FEI, 2010. 42 s.

[10] Postel, J. - ISI: *User Datagram Protocol.* RFC 768. 1980.

[11] ISI: *Transmission Control Protocol.* RFC 793. 1981.

[12] Haden, R.: *TCP [online].* In: Data Network Resource. Dátum neznámy. [cit. 2012-04-02] Dostupné na internete: <`http://www.rhyshaden.com/tcp.htm`>

[13] Oracle: *Orderly Versus Abortive Connection Release in Java [on-*

line]. Dátum neznámy. [cit. 2012-02-16]. Dostupné na internete: <http://docs.oracle.com/javase/1.5.0/docs/guide/net/articles/connection_release.html>

[14] Stewart, R.: *Stream Control Transmission Protocol.* RFC 4960. 2007.

[15] Stewart, R. et al.: *Stream Control Transmission Protocol (SCTP) Partial Reliability Extension.* RFC 3758. 2007.

[16] Stewart, R. et al.: *SCTP: What is it, and how to use it?* In: BSDCan. Technická BSD konferencia. Kanada, 2008. 10 s.

[17] Shaojian, F. - Mohammed, A.: *SCTP: State of the Art in Research, Products, and Technical Challenges.* In: IEEE Communications Magazine. 2004 ISSN 0163-6804, s. 64 – 76.

[18] Milllis, D.: *Network Time Protocol (Version 3) : Specification, Implementation and Analysis.* RFC 1305. 1992.

[19] Josefsson, S.: *The Base16, Base32, and Base64 Data Encodings.* RFC 4648. 2006.

[20] The Apache Software Foundation: *Apache Commons Net [online].* 2012. [cit. 2012-03-10]. Dostupné na internete: <http://commons.apache.org/net/>

[21] LKSCTP Maintainers: *SCTP for the Linux Kernel [online].* 2012. [cit. 2012-03-11]. Dostupné na internete: <http://commons.apache.org/net/>

[22] Jorgensen, I.: *Java SCTP Library [online].* Dátum neznámy. [cit. 2012-03-16]. Dostupné na internete: <http://i1.dk/JavaSCTP/>

[23] Oracle: *A Java API for Stream Control Transport Protocol [online].* 2011. [cit. 2012-03-7], Dostupné na internete: <http://docs.oracle.com/javase/7/docs/jre/api/nio/sctp/spec/com/sun/nio/sctp/package-summary.html>

[24] Hegarty, Ch.: *Stream Control Transport Protocol (SCTP) in Java [online].* In: Oracle Sun Developer Network (SDN). 2009. [cit. 2012-03-18]. Dostupné na internete: <http://java.sun.com/developer/technicalArticles/javase/jdk7-sctp/>

[25] Oracle: *Using the SCTP API from OpenJDK with Sun's JDK6 [online].* Dátum neznámy. [cit. 2012-03-18]. Dostupné na internete: <http://openjdk.java.net/projects/sctp/html/sctp6.html>

[26] Carnegie Mellon University: *Yet Another Flowmeter [online].* CERT NetSA Security Suite. Dátum neznámy. [cit. 2012-04-13] Dostupné na internete: <http://tools.netsa.cert.org/yaf/index.html>

[27] FAU Erlang - TU München: *Vermont [online].* 2012. [cit. 2012-04-13]. Dostupné na internete: <https://github.com/constcast/vermont/wiki>

[28] University of Waikato: *Maji [online].* Dátum neznámy. [cit. 2012-04-15]. Dostupné na internete: <http://research.wand.net.nz/software/maji.php>

Zoznam príloh

Príloha A Systémová príručka JXColl v3.9

Príloha B Používateľská príručka JXColl v3.9

Príloha C CD médium obsahujúce:

diplomovú prácu v elektronickej podobe (PDF, LaTeX)

systémovú príručku JXColl v3.9 (PDF, LaTeX)

používateľskú príručku JXColl v3.9 (PDF, LaTeX)

zdrojové súbory JXColl v3.9 (Java)

inštalačný skript na vytvorenie databázovej štruktúry (SH, SQL)

Javadoc dokumentáciu tried programu JXColl v3.9 (HTML)

DEB inštalačný balík pre JXColl v3.9 (DEB)