FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Univerzální agregační modul pro systém NEMEA
Student:	Bc. Michal Slabihoudek
Vedoucí:	Ing. Tomáš Čejka
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Seznamte se s modulárním systémem NEMEA [1,2] pro analýzu síťových toků a detekci anomálií, zaměřte se na používaný datový formát UniRec [3].
Navrhněte vhodný (uživatelsky použitelný) způsob zápisu/konfigurace pro agregování informací o síťovém provozu ve formátu UniRec v reálném čase. Tento zápis by měl podporovat specifikaci agregačního intervalu, klíče pro agregování, základní sadu operací (count, sum, avg, ...).
Navrhněte a implementujte softwarový modul do systému NEMEA pro agregaci proudu dat ve formátu UniRec v reálném čase. Při implementaci se soustřeďte na vysokou propustnost vyvíjeného modulu, aby jej bylo možné použít nad daty z vysokorychlostních sítí.
Vzniklý modul otestujte na datech s různými šablonami UniRec formátu a s různě nastavenými agregačními funkcemi, změřte propustnost modulu.

Seznam odborné literatury

[1] T.Cejka, et al.: "NEMEA: A Framework for Network Traffic Analysis," in *12th International Conference on Network and Service Management (CNSM 2016)*, Montreal, Canada, 2016.
[2] https://github.com/CESNET/NEMEA
[3] https://github.com/CESNET/Nemea-Framework/tree/master/unirec

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Univerzální agregační modul pro systém NEMEA

Bc. Michal Slabihoudek

Katedra počítačových systémů
Vedoucí práce: Ing. Tomáš Čejka

3. května 2018

Poděkování

Tímto děkuji svému vedoucímu práce Ing. Tomáši Čejkovi za podporu a spolupráci s orientací v prostředí systému NEMEA a vývojem výsledného modulu. Poděkování také náleží dalším členům podílejícím se na vývoji NEMEA systému, zejména Tomáši Jánskému, který byl ochoten pomoci překonávat překážky vzniklé během tvorby závěrečné práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. května 2018 .....

Odkaz na tuto práci

Slabihoudek, Michal. *Univerzální agregační modul pro systém NEMEA*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá návrhem a implementací univerzálního agregačního modulu do existujícího síťového detekčního systému NEMEA. Text práce obsahuje popis systému NEMEA včetně datových formátů využívaných pro komunikační účely. Implementační část práce ukazuje důležité rysy vzniklého agregačního modulu. Výsledky testování potvrzují splnění požadavků a schopnost modulu zpracovat data z vysokorychlostních sítí.

Klíčová slova NEMEA, agregace, UniRec, síťové toky, síťová bezpečnost, analýza síťového provozu, CESNET, LibTrap, modularita, C, C++

Abstract

The main part of this thesis is about design and implementation of an aggregation module for the existing network detection system NEMEA. The thesis also describes system environment of the module (i.e., related tools and systems) with existing data format used for representation of flow data. The implementation part of the thesis shows important features of the aggregation module. The functionality and performance of the developed module were evaluated and the test results confirm requirements fulfilment and the ability to process data from high-speed networks.

Keywords NEMEA, aggregation, UniRec, netflow, network security, network traffic analysis, CESNET, LibTrap, modularity, C, C++

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Stále častěji dochází k zneužívání zařízení připojených do internetové sítě. Cílové i zdrojové stanice, které jsou k síti připojeny jsou směřovány a přizpůsobovány k určité typické činnosti a provoz výpočetně náročného a komplexního ochranného mechanismu se stává kritickým bodem zájmu. Proto se detekce a analýza postupně přesouvají z cílových stanic do prostředí poskytovatele komunikačního kanálu. Nežádoucí datové toky se detekují, analyzují, hlásí, případně i blokují bez aktivní účasti připojených klientů.

Řešení modulárního systému *NEMEA* implementuje výše zmíněné prvky bezpečnosti v prostředí, operátora národní akademické sítě, organizace CESNET. Systém se neustále vyvíjí, a tak roste i počet existujících modulů. Množství komunikačního materiálu zvyšuje výkonnostní požadavky jednotlivých modulů a zároveň zaplavuje detekční i ostatní systémy spoustou záznamů ke zpracování. Tyto záznamy se ve stanovených intervalech dají dle specifických pravidel slučovat (agregovat) bez výrazných ztrát.

Agregace je nutná ke snížení objemu provozních dat celého systému. Uvolní výpočetní kapacity jednotlivých modulů, uleví vytížení linek a umožní snížení kapacit HW prostředků operátorů kritických infrastruktur na jejich ukládání po nezbytně dlouho dobu dle zákona. Sníží se množství výsledných hlášení z detekčních systémů, a tím se zpřehlední celkový stav sítě. Správnou kombinací s filtračními prvky může být také agregace nástroj pro jednoduchou detekční činnost. Na vlastnosti seskupených zpráv jsou založeny i další návrhy modulů a částí systému, proto jsem se rozhodl pomoci v dalším rozvoji a tento agregační modul implementovat.

Výstupem práce bude implementace univerzálního agregačního modulu do open source modulárního detekčního systému NEMEA, který je vyvíjen ve spolupráci sdružení CESNET a předních českých univerzit. Modul bude slučovat přijaté zprávy na základě nastavených pravidel a v daných intervalech agregované zprávy přeposílat. Testování vzniklého modulu a výsledky testů budou součástí práce, stejně jako nasazení v testovacím režimu na kolektoru.

Kapitola 1

Monitorování a analýza síťového provozu

V zájmu každého provozovatele síťové infrastruktury je ji spravovat a udržovat v nenarušeném a kontrolovaném stavu. Jako základní prvek monitorování sítě se dá považovat sledování aktivního či neaktivního připojení cílových stanic do spravované infrastruktury. Tímto způsobem lze sledovat počet stanic a služby, které provozují. Těmito opatřeními lze zajistit stav prostředí z pohledu provozovatele, tím ovšem zajištění bezpečnosti nekončí. Mnoho provozovatelů již bezpečnostní služby (detekce incidentů) ve své infrastruktuře zajišťují. Dochází k rozvoji monitorovacích služeb a nástrojů, které nabízí možnosti informovat správce služby či dokonce přímo konkrétního klienta o detekované skutečnosti. Tímto se ale dostáváme k potřebě hlubšího (detailnějšího) způsobu sledování spravované infrastruktury, a ten představuje analýza přenášených dat.

Dle zdroje [1] existují dva směry metodologie detekce síťových anomálií:

- **Nestatistické** - Využívají znalostí deterministických chování komunikačních protokolů, na jejichž základě lze pozorovat neobvyklé chování pomocí detekce stavů. Na tomto principu je založena např. detekce s názvem *pattern matching*.
- **Statistické** - Založeno na myšlence, že narušení vede ke změnám statistických vlastností sledovaných charakteristik. Neobvyklé chování je zpozorováno metodou *CPD* (change point detection), neboli že v nějakém náhodném čase λ dojde ke změně statistického rozložení (parametru, či celého modelu).

Detekce možného narušení se vyskytuje v podobě bezpečnostních hlášení (*alertů*). Žádný modul není schopen bezchybné detekce, a proto se používá termín *false alarm*, který představuje falešně pozitivní hlášení bezpečnostního incidentu.

1.1 Sekvenční detekce

Mějme sekvenci $X_1, X_2, ..., X_n$ síťových charakteristik sledovaných v časech $t_1, t_2, ..., t_n$. Naměřené hodnoty charakteristik se analyzují a v případě detekce se zvýší hodnota nějaké kontrolní proměnné. Tato hodnota je neustále porovnávána s nastavenou mezní hodnotou zvanou *threshold*. V případě překročení stanoveného *thresholdu* dojde k vygenerování *alertu*.

Detekce pracuje s následujícími pojmy.

- **FAR** - Četnost vygenerovaných falešně pozitivních hlášení.
- **ADD** - Průměrná doba mezi detekcí anomálie a vygenerováním hlášení.
- **PFA** - Pravděpodobnost, že systém vygeneruje falešně pozitivní hlášení.
- **PWR** - Pravděpodobnost úspěšné detekce a nahlášení anomálie.

Vztahy mezi zmíněnými pojmy lze vyjádřit následující tabulkou.

Tabulka 1.1: Vztahy mezi vlastnostmi sekvenční detekce.

Vyšší treshold	nižší FAR	nižší PFA
	delší ADD	*menší* PWR
Nižší treshold	vyšší FAR	vyšší PFA
	kratší ADD	*větší* PWR

S touto metodikou pracují systémy IDS (intrusion detection systems) a IPS (intrusion prevention system) s rozdílem, že IPS anomálii nejen detekuje, ale snaží se o automatickou reakci, která toto narušení omezí, či dokonce eliminuje. Stejně tak jako existuje více možností analýzy dat, jsou i různé pohledy na datové formy, které těmto analytickým procesům podléhají.

1.2 Paketová analýza

Jak již název napovídá zdrojem informací jsou v tomto případě jednotlivé datové pakety. Jedná se o zpracování *Raw* dat, tedy dat zachycených během komunikace bez jakékoli úpravy. Síťový protokol je vystaven na několika úrovních (Obrázek 1.1). Uvažujme implementaci komunikace dle standardu ISO/OSI [2]. Kontrola těchto datových zdrojů může být prováděna na sedmi různých vrstvách. Princip ověřování dat jednotlivých paketů se také označuje jako DPI (deep packet inspection).

Obrázek 1.1: Model síťových vrstev dle standardu ISO/OSI [2].

Takovýto způsob detekce je velice náročný. Zpracování každého datového paketu na všech úrovních v reálném čase je náročné a vyžaduje značné výpočetní zdroje. Záznamy se tedy pořizují s přesným záměrem a před zpracováním podléhají filtrování dle nastavených pravidel. Následným procesem může být zpracování dat nazývané *pattern matching*. Tento pojem představuje hledání hodnot uvnitř paketu dle předem stanovených pravidel (rule-based detekční systémy). Pokud hodnoty uvnitř paketu souhlasí s těmi v nastaveném pravidle, vyvolá se patřičná akce.

Existující nástroje využívající tohoto typu detekce jsou například Snort[1] a Suricata[2].

1.3 Analýza na základě datových toků

Kvůli náročnosti zpracování není paketová analýza vhodná na monitorování páteřních sítí s velkým objemem přenášených dat. Samotná flow analýza také umožňuje zpracovávat provoz z více linek s určitou datovou souvislostí, která poskytuje globální pohled na dění v síti. Umožňuje tak lepší detekci jevů, které se na lokálních částech nemusí zřetelně projevit.

Termín IP flow není jednoznačně definovaný. Jeho popis lze nalézt ve více publikacích a různých podobách, například:

1. „Uměle vytvořený logický ekvivalent hovoru nebo spojení.“ [3]

2. „Sekvence paketů odeslaných z určitého zdroje k vybranému cíli skrze unicast, anycast nebo multicast.“ [4]

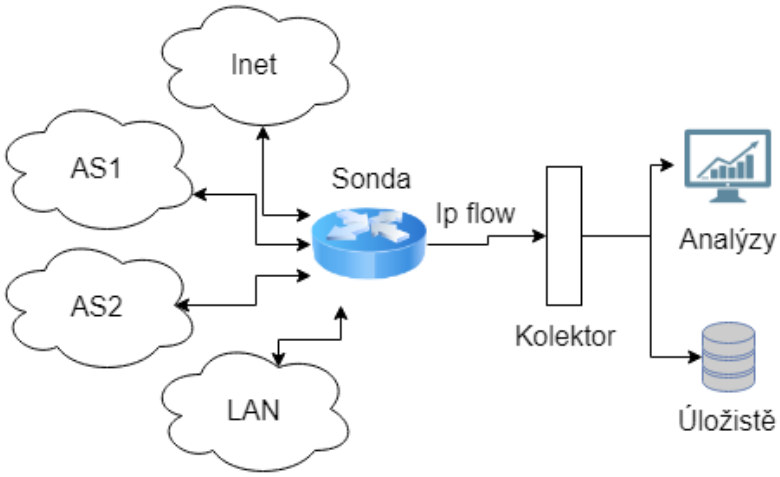
3. „Množina IP paketů stejných vlastností procházející skrze pozorovací bod v síti během určitého časového intervalu.“ [5]

V prvním zmíněném zdroji lze sledovat vnímaní tohoto výrazu jako abstraktního pojmu. Druhý zdroj udává již podobu termínu. Třetí zdroj defi-

[1] https://www.snort.org/
[2] https://suricata-ids.org/

nuje současně nejvíce používanou definici, jejíž znění vychází ze způsobu zachytávání a vytváření samotného datového formátu.

Pozorovacím bodem v síti může být aktivní síťový prvek (switch, router, server), který na základě provozu generuje datové toky a následně je dle nastavených pravidel exportuje (viz. sekce 1.3.2) nebo pasivní prvek reprezentovaný rozbočovačem či zrcadlem datového provozu. Již při vytváření flow záznamu hraje podstatnou roli agregace paketů v rámci stanovených časových intervalů. Ukázkové schéma infrastruktury monitorování pomocí IP flow je znázorněno na Obrázku 1.2.



Obrázek 1.2: Jednoduchá topologie znázorňující vznik a následné zpracování datových toků (IP flow).

Existující nástroje zpracovávající formát flow záznamů jsou například NfSen[3], Flowmon[4] a NEMEA [15]. Detaily systému NEMEA jsou popsány v Kapitole 3.

1.3.1 Kategorie síťových toků (IP flow)

Samotné síťové toky jsou na základě směru zachycených dat rozdělovány do dvou kategorií. Následující vysvětlení a příklady využívají tento jednoduchý a krátký scénář komunikace.

[3] http://nfsen.sourceforge.net/
[4] https://www.flowmon.com/cs

Host A odešle 90 B hostu B a host B odpoví 120 B.

- **Uni-flow** - Obsahuje pouze jeden směr komunikace. IP flow může vypadat následovně.

Tabulka 1.2: Uni-flow ukázka.

srcaddr	**direction**	**dstaddr**	**total Bytes**
Host A	->	Host B	90
Host B	->	Host A	120

- **Bi-flow** - Obsahuje oba směry komunikace. Ip flow může vypadat následovně.

Tabulka 1.3: Bi-flow ukázka.

srcaddr	**direction**	**dstaddr**	**total Bytes**	**src Bytes**	**dst Bytes**
Host A	<->	Host B	210	90	120

1.3.2 Ukončení toku (Flow termination)

Sekce se věnuje definovaným způsobům kdy a jak ukončit agregaci dat a odeslat vytvořený IP flow na výstup (nejčastěji na kolektor). Existují následující časové intervaly exportu (timeouts) [7].

- **Aktivní** (Active) timeout - Datový tok je aktivní po specifikovaný časový úsek t. Tento timeout pomáhá exportovat dlouho trvající aktivity v intervalech t sekund. Typicky používané hodnoty jsou v rozmezí $t = 120$ sekund až $t = 30$ minut. Záznam není z flow cache během exportu odstraněn, čítač je resetován a časové hodnoty počátku a konce aktualizovány dle příchozích dat.

- **Neaktivní** (Idle) timeout - Žádný nový paket patřící ke konkrétnímu flow záznamu nebyl spatřen po dobu t sekund. Typické hodnoty timeoutu jsou od $t = 15$ sekund po $t = 5$ minut.

- **Ukončení spojení** (Natural expiration) - Byl přijat TCP paket s příznakem `FIN` (konec) nebo `RST` (reset) ke konkrétnímu záznamu oznamující ukončení spojení.

- **Násilné** (Emergency) - Ukončení a odeslaní bloku vytvořených flow z důvodu nedostatečné kapacity pro další bezchybný provoz flow exporteru.

- **Jednorázové** (Cache flush) - Odeslání záznamů z důvodů potřeby vyprázdnění úložiště. Může být způsobeno například změnou času důsledkem aktualizace časových informací nebo jinou vlastní potřebou exporteru, např. export záznamů v pravidelných intervalech délky t sekund.

Kapitola 2

Exportní protokoly

V této kapitole jsou popsány vybrané konkrétní datové reprezentace a jejich implementace. Jednou z prvních implementací, která využívala data v podobě datových toků byl proprietární protokol společnosti Cisco zvaný NetFlow.

2.1 NetFlow

Vznikl jako integrovaný nástroj uvnitř Cisco IOS softwaru za účelem sledování síťových operací jako odpověď na požadavky síťových operátorů, kteří sledování chování sítě považovali za kritické. Za nejdůležitější vlastnosti byly považovány parametry typu jaké aplikace využívají síť, využití síťových zdrojů, projevy změn na síti nebo dodržování stanovených podmínek.

Za jeden unikátní datový tok je považován jednosměrný (Sekce 1.3.1) proud paketů identifikován sedmi políčky.

1. zdrojová/cílová IP adresa
2. zdrojový/cílový port
3. typ protokolu třetí vrstvy (L3)
4. typ služby (TOS)
5. vstupní logické rozhraní (interface)

Záznamy jsou ukládány a agregovány dle stanovených pravidel, blíže popsaných v Sekci 1.3.2. Každý flow záznam je po vypršení platnosti uskupen do tzv. „NetFlow Export“ datagramu a odeslán skrze UDP na kolektor. NetFlow Export datagram se skládá z hlavičky a sekvence flow záznamů (Tabulka 2.1).

Tabulka 2.1: Typický datagram pro Netflow formáty fixní délky.

IP hlavička
UDP hlavička
NetFlow hlavička
Flow záznam
...

2.1.1 Formáty exportu

NetFlow prošel značným vývojem a za dobu existence byly publikovány následující verze, z nichž pouze verze 9 byla přijata jako standardní formát exportního protokolu.

Verze 1

První vydaný formát exportu datových toků. Nedoporučuje se používat, pokud není vyžadován starším systémem sběru dat. Formát samotného flow záznamu je pevně daný a hodnoty mohou být dosaženy pomocí stanovených offsetů [8].

Tabulka 2.2: Hlavička NetFlow verze 1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Verze																Počet															
Doba běhu systému																															
UNIX čas (sekundy)																															
UNIX čas (nanosekundy)																															

Version 5

Nejrozšířenější používaný formát. Vylepšená verze formátu obohacená o BGP-AS informace a sekvenční čísla. Verze 2-4 nebyly nikdy veřejně uvolněny. Jelikož jsou flow záznamy odesílány skrze UDP, přidáním sekvenčního čísla do hlavičky exportovaných záznamů se umožnila detekce ztracených dat. Formát samotného flow záznamu je pevně daný a hodnoty mohou být dosaženy pomocí stanovených offsetů [8].

Tabulka 2.3: Hlavička NetFlow verze 5.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Verze 1 ...																															
Sekvenční číslo																															
Typ zdroje								ID zdroje								Rezervováno															

Version 8

Speciální formát pro podporu exportu z agregačních exportérů (NetFlow aggregation caches). Obsahuje podmnožinu flow ve formátu verze 5. Vzorkovací interval představuje nastavené časové okno agregace, pokud je nakonfigurováno.

Tabulka 2.4: Hlavička NetFlow verze 8.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31			
Verze 1 ...			
Sekvenční číslo			
Typ zdroje	ID zdroje	Typ agregace	verze agregace
Vzorkovací interval		Rezervováno	

Version 9

Nejnovější flexibilní a rozšiřitelný formát založený na šablonách. Díky šablonám a variabilním polím podporuje nové typy záznamů a políčka, která mohou být dokonce definována samotným administrátorem v konfiguraci exportéru.

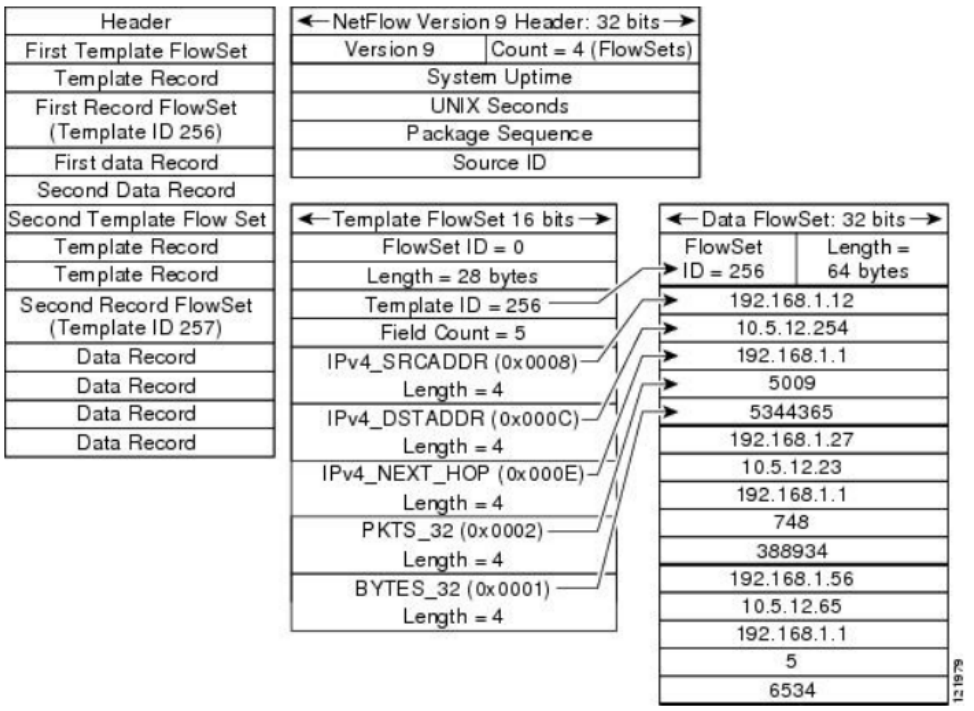
Tento formát není zpětně kompatibilní s předchozími (v1, v5, v8).

- Obsah záznamů (pořadí dat) není pevně stanoven, ale je dán použitou šablonou.
- Popis šablon je vyměňován mezi routerem a sběrným zdrojem NetFlow (kolektorem).
- Flow záznamy jsou odeslány na kolektor s minimální sadou dat identifikující použitou šablonu, takže kolektor si data spojí s konkrétní šablonou bez potřeby jejího přenosu.
- Přenos datagramů je nezávislý na transportním protokolu (UDP, TCP, SCTP, atd.).

Šablona definuje formát a atributy jednotlivých políček (např. délka) v záznamu. Router (exporter) vygeneruje šablonu a přidělí jí unikátní identifikátor, který je poté předán na kolektor i s definicí šablony. Během další komunikace se již přenáší pouze ID šablony.

Datagram se skládá z hlavičky a 2 typů tzv. flowsets (množin datových toků) (Obrázek 2.1).

- **Template flowset** (Množina šablon) - Popisuje pole, která se vyskytují v datové množině nebo flow záznamech.
- **Data flowset** (Datová množina) Obsahuje hodnoty nebo statistiky jednoho či více datových toků se stejnou šablonou.



Obrázek 2.1: Ukázka NetFlow datagramu verze 9 [8].

2.2 IPFIX

Dalším typem formátu pro přenos IP flow je IPFIX (Internet Protocol Flow Information Export). Byl vytvořen pro roli univerzálního protokolu pro export IP flow informací ze síťových zařízení (Switche, Routery, firewally, sondy) na kolektor. Veškerá struktura vychází z definice IP flow jako množiny IP paketů míjející pozorovací bod. Za pozorovací bod se považuje síťový prvek generující IP flow, které přímo exportuje na kolektor, nebo je předává na exportér, který je nadále může ve vzorkovacích intervalech agregovat. V roce 2004 byl organizací IETF ustanoven jako standard [5].

IPFIX byl vyvinut na základech a principech protokolu NetFlow v9 (Obrázek 2.1), na který většina provozovatelů do té doby spoléhala. Definuje vlastní podobu šablon, záznamů a zpráv.

- **Šablona** - Uspořádaná sekvence dvojic <typ,délka>, která kompletně definuje strukturu množiny informací, která je přenášena z IPFIX exportéru na kolektor. Stejně jako u konkurentního protokolu NetFlow je šablona identifikovatelná pomocí unikátního id (Tabulka 2.5).

Tabulka 2.5: IPFIX šablona. [6].

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Id šabony	Počet políček
Informační element id (id políčka)	Délka políčka
Informační element id (id políčka)	Délka políčka
...	

- **Hlavička** - První část přenášené zprávy. Poskytuje základní informace o zprávě (verzi, délku zprávy, sekvenční číslo,...).

Tabulka 2.6: Hlavička IPFIX [6].

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Verze	Délka
Čas exportu	
Sekvenční číslo	
Id pozorovací domény	

- **Template record** (záznam šablony) - Definuje strukturu a interpretaci políček v datovém záznamu.
- **Data record** (datový záznam) - Obsahuje hodnoty parametrů definovaných v odpovídající šabloně.

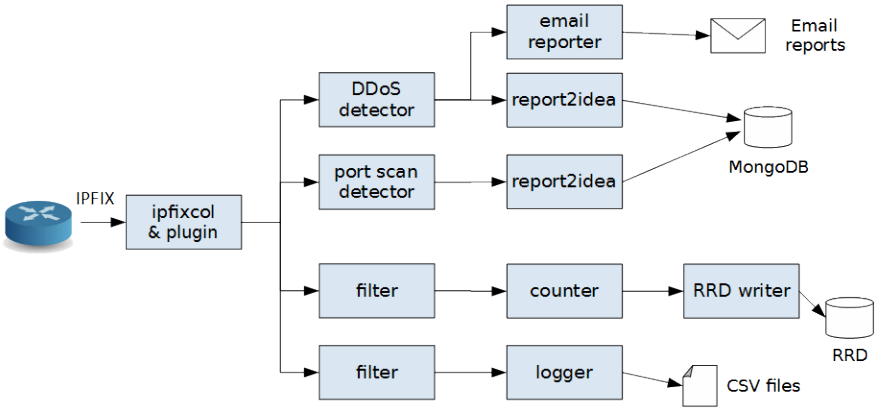
Šablony i samotné datové zprávy jsou shlukovány do množin (např. k jedné šabloně patří více datových záznamů). Možné podoby IPFIX zpráv jsou naznačeny na Obrázku 2.2.

Obrázek 2.2: Možné podoby IPFIX zprávy [6].

Kapitola 3

Systém NEMEA

NEMEA (Network Measurements Analysis) je proudově orientovaný, modulární detekční systém pro analýzu síťového provozu ve formě síťových toků (IP flow). Systém je vhodný k použití okamžité (on-the-fly) analýze datových toků. Díky více možnostem vstupu a výstupu podporuje zpracování živého i zachyceného síťového provozu. Kompletní systém včetně potřebných modulů je také připraven jako *STaaS*[5] (Security tools as a service). V důsledku toho je nasazení celého systému do infrastruktury o to snazší a rychlejší. Obrázek 3.1 zobrazuje ukázku možného zapojení NEMEA modulů.



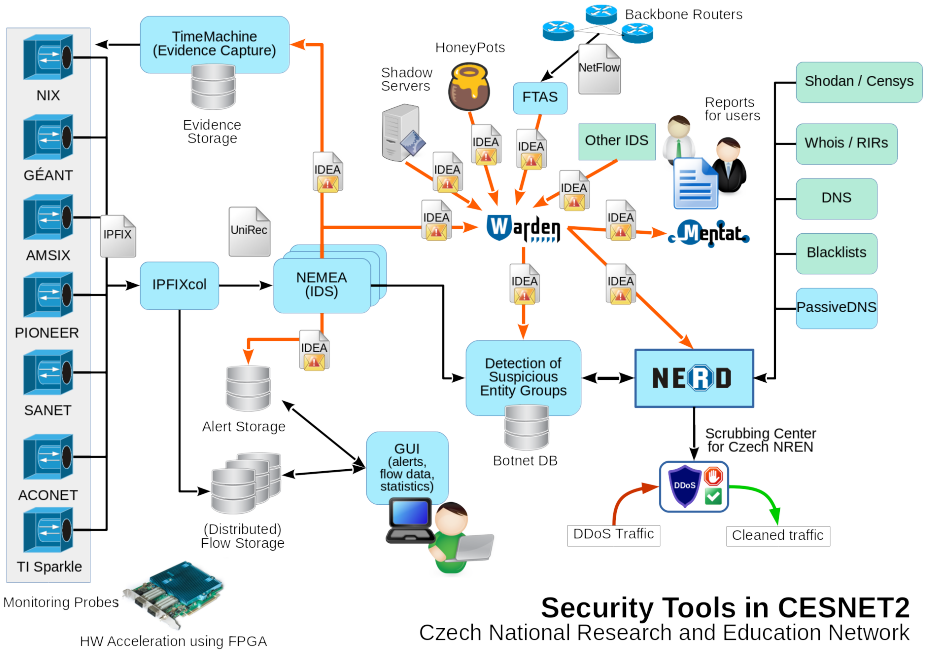
Obrázek 3.1: Současně existující prostředí systému NEMEA [15].

Celý systém je tvořen z nezávisle běžících NEMEA modulů, které průběžně zpracovávají příchozí data (zprávy). Zpracované zprávy či jiné výstupy mohou být dále odeslány do jiného modulu či systému k dalšímu zpracování nebo analýze. Veškerá komunikace uvnitř prostředí systému NEMEA je založena na komunikačním rozhraní LibTrap (Sekce 3.2). Zpráva obvykle obsahuje informace o síťových tocích ve formátu UniRec (Sekce 3.1). Zpráva ovšem může

[5]https://github.com/CESNET/STaaS

obecně obsahovat i jiná data, například hlášení o detekci bezpečnostního incidentu. NEMEA modul reprezentuje instanci či několik instancí spustitelného kódu na daném OS.

Za dobu existence NEMEA systému se rozvinula spolupráce s ostatními systémy (službami) v rámci organizace CESNET, pro které je díky svým schopnostem zpracování dat v reálném čase a dostatečné univerzálnosti zdrojem dat. Výstup systému NEMEA se používá například v systémech *Warden* a *NERD*. NEMEA ve spolupráci s ostatními systémy je součástí scrubbing centra pro českou národní výzkumnou a univerzitní síť, jehož schéma je zobrazeno na Obrázku 3.2.



Obrázek 3.2: Ukázka propojení NEMEA systému s dalšími systémy, které jsou součástí scrubbing centra pro českou NREN [16].

3.1 UniRec

UniRec neboli Unified record (jednotný záznam) představuje vlastní implementaci reprezentace dat vyvinuté pro účely komunikace v systému NEMEA. UniRec využívá návrhů Netflow v9 a IPFIX, které přizpůsobil svým potřebám.

Návrh protokolu využívá šablony k zachování dostatečné univerzálnosti. Do formátu je možné dodefinovat libovolné nové hodnoty políček, které budou přenášeny mezi aplikacemi využívajícími tento formát. Šablony také umožňují přenášení datových polí s variabilní délkou.

Tabulka 3.1: Podporované datové typy v UniRec záznamech [11].

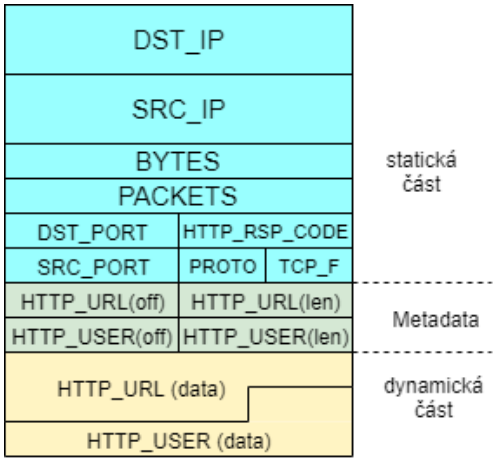
Jméno	**Velikost (B)**	**Popis**
int{8, 16, 32, 64}	1, 2, 4, 8	celá čísla
uint{8, 16, 32, 64}	1, 2, 4, 8	přirozená čísla vč. 0
char	1	samotný ASCII znak
float	4	číslo v plovoucí řádové čárce
double	8	číslo v plovoucí řádové čárce (dvojitá přesnost)
ipaddr	16	IPv4/IPv6 adresa
macaddr	6	MAC adresa
time	8	Časové značky
string	–	Pole variabilní délky (zobrazitelné znaky)
bytes	–	Pole variabilní délky

3.1.1 Struktura záznamu

Ke správné identifikaci datových záznamů slouží *šablona*. Ta je složena z definic jednotlivých datových políček, které mají být přenášeny mezi výstupním a vstupním rozhraním systému. Každé políčko reprezentující popis přenášených dat je definováno jako pár <název, datový typ>, kde název slouží jako unikátní identifikátor daného pole. Aktuálně podporované datové typy jsou zobrazeny v Tabulce 3.1.

Samotný datový záznam se skládá ze tří částí. *Statické* části, kde všechna políčka mají pevnou délku stanovenou v šabloně, *meta informace* políček s variabilní délkou (offset, velikost) a *dynamické* části, která obsahuje samotná data. Samotný záznam je omezen maximální velikostí $(2^{16} - 2) = 65534$ B. Podrobnější popis a vizualizace záznamu je naznačena na Obrázku 3.3, který zobrazuje informace o HTTP připojení. Šablona tohoto záznamu vypadá následovně.

```
ipaddr SRC_IP, ipaddr DST_IP, uint16 SRC_PORT, uint16 DST_PORT,
uint8 PROTOCOL, uint8 TCP_FLAGS, uint32 PACKETS, uint32 BYTES,
uint16 HTTP_RSP_CODE, string HTTP_URL, string HTTP_USER_AGENT
```



Obrázek 3.3: Ilustrace struktury UniRec záznamu [11].

3.1.2 UniRec API

K jednoduššímu vývoji nových aplikací spolupracujících s tímto formátem je k dispozici API [12], které umožňuje manipulovat se šablonami i jednotlivými záznamy. Mezi základní operace patří:

- **Definice políček zprávy** ve formátu šablony (typ název).

  ```
  UR_FIELDS(type name [, type name [, ...] ])
  ```

- **Definice políčka zprávy** za běhu aplikace. Pro definování více políček najednou se používá formát šablony (typ název), kde jsou jednotlivá pole oddělena čárkami.

  ```
  int ur_define_field(const char *name, ur_field_type type)
  int ur_define_set_of_fields(const char *ifc_data_fmt)
  ```

- **Generování šablony** z již definovaných polí v globální struktuře. Parametr `field` je řetězec názvů polí oddělených čárkami.

  ```
  ur_template_t *ur_create_template(const char *fields,
                                    char **errstr)
  ```

- **Uvolnění generované šablony**

  ```
  void *ur_free_template(ur_template_t *tmplt)
  ```

- **Získání dat** ze záznamu dle uvedené šablony.

  ```
  ur_get(tmplt, rec, field)
  ur_get_ptr_by_id(tmplt, rec, field)
  ```

- **Zápis dat** do záznamu dle uvedené šablony. V případě pole s variabilní délkou se získá ukazatel na data a do něj se zapíše.

  ```
  ur_set_var(tmplt, rec, field, val_ptr, val_len)
  ```

- **Získání typu** políčka z globální struktury.

  ```
  ur_is_varlen(field)
  ur_is_fixlen(field)
  ```

- **Vytvořit záznam** dle šablony.

  ```
  void* ur_create_record(const ur_template_t *tmplt,
                         uint16_t max_var_size);
  ```

- **Zkopírovat data** dle šablony do jiného záznamu.

  ```
  void ur_copy_fields(dst_tmplt, dst, src_tmplt, src);
  ```

- **Uvolnit záznam**.

  ```
  void ur_free_record(void *record);
  ```

Globální struktura je kontejner obsahující všechny potřebné informace o již definovaných UniRec políčkách.

3.2 LibTRAP

LibTrap je knihovna, která slouží ke zprostředkování komunikace mezi vstupním a výstupním rozhraním. Implementuje všechny potřebné komunikační metody a přenos dat mezi rozhraními. Abstraktní vrstva tohoto rozhraní zaručuje, že z pohledu vývojáře (vlastní implementace aplikace) je zcela jednotné použití pro všechny podporované typy komunikačních kanálů. Tuto vlastnost podporuje skutečnost, že výběr skutečného komunikačního kanálu použitého pro danou běžící instanci aplikace provádí sám uživatel pomocí parametru při spuštění.

3.2.1 Typy komunikačních rozhraní

- **TCP** - Komunikace skrze TCP soket. Výstupní rozhraní poslouchá na daném portu, vstupní rozhraní se k němu připojí a poslouchá na něm. Více vstupních rozhraní může být připojeno na jedno shodné výstupní rozhraní. V takovém případě se na každý vstup doručí stejná data (kopie).

  Pro použití TCP kanálu je třeba specifikovat cílovou/zdrojovou adresu, záleží na použití vstup/výstup, a cílový port. Pro výstupní rozhraní také existuje možnost omezit počet najednou připojených vstupních rozhraní (klientů).

- **TLS** - Komunikace skrze TCP soket (stejně jako TCP výše). Dochází k navázání šifrovaného spojení (SSL handshake). Více vstupních rozhraní může být připojeno na jedno shodné výstupní rozhraní. V takovém případě se na každý vstup doručí stejná data (kopie).

  Pro použití TLS kanálu je třeba oproti TCP specifikovat navíc cesty k souborům reprezentujícím platný certifikát, privátní klíč, soubor obsahující řetězec certifikačních autorit s důvěryhodnou certifikační autoritou v PEM formátu. Pro výstupní rozhraní také existuje možnost omezit počet najednou připojených vstupních rozhraní (klientů).

- **UNIX** - Komunikace skrze UNIX soket. Tento typ je použitelný pouze pro komunikaci v rámci stejného UNIXového operačního systému mezi procesy nebo jednotlivými vlákny. Výstupní rozhraní vytvoří soket a poslouchá na něm, vstupní rozhraní se k tomuto portu připojuje. Více vstupních rozhraní může být připojeno k jednomu konkrétnímu výstupnímu. V takovém případě každé takové rozhraní obdrží stejnou kopii dat.

  Pro použití tohoto typu komunikace je zapotřebí definovat identifikátor pro vytvořený, či připojovaný soket. Identifikátor je řetězec znaků, který musí splňovat požadavky pro vytvoření souboru. Pro výstupní rozhraní také existuje možnost omezit počet najednou připojených klientů.

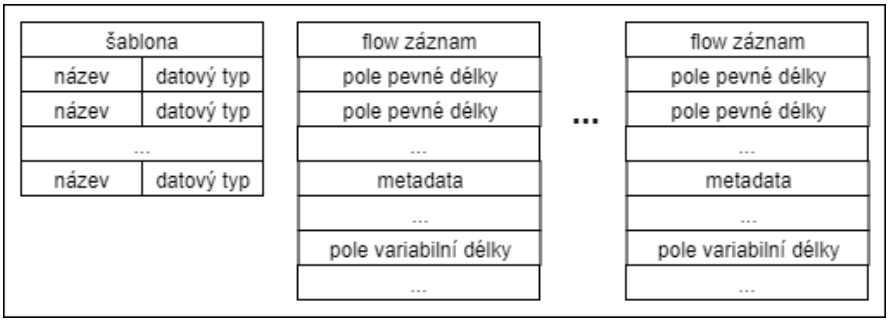
- **Blackhole** Veškerá komunikace směřována do tohoto typu rozhraní je zahozena. Může být nastaveno pouze jako výstupní rozhraní. Tento typ komunikačního kanálu slouží pro účely testování nebo vývoje aplikací.

- **File** Komunikace prostřednictvím souborů. Vstupní rozhraní čte data z přiložených souborů, výstupní rozhraní ukládá data do specifikovaných souborů. Tento způsob je použitelný pro více vstupních souborů i pro zapisování nebo tvorbu více souborů dle zadaných pravidel prostřednictvím konfiguračních parametrů. Lze použít soubory obsahující zachycený provoz s doporučovanou příponou *.trapcap*.

Více informací o zmíněných pravidlech a příkladech použití lze nalézt na oficiálních webových stránkách [13].

3.2.2 Komunikace

Způsob komunikace mezi vstupním a výstupním rozhraním LibTrap probíhá přenosem přidělené šablony a k ní vztaženého datového proudu. Během navázání komunikace mezi výstupním a vstupním rozhraním dojde k ověření, zda druhá strana umožňuje příjem políček obsažených v šabloně. Pokud cílový modul (aplikace) umožní přijmout všechna používaná políčka, dojde k vytvoření spojení a mohou se začít přeposílat data. Pro každé spojení platí, že přenášená data vyhovují právě jedné šabloně.



Obrázek 3.4: Komunikace v UniRec formátu skrze LibTrap (datový proud).

Na Obrázku 3.4 je znázorněna proudová komunikace. Po úvodní inicializaci se už zasílají pouze nekonečné proudy dat, dokud se spojení neukončí. Tento způsob komunikace slouží k rychlejšímu a datově méně náročnému přenosu mezi zdrojem a cílem, například oproti IPFIX (Obrázek 2.2), kde se dávkově přenáší vždy i šablona pro obsažená data.

3.2.3 LibTrap API

Za účelem zjednodušení používání této knihovny je vytvořeno vývojářské aplikační rozhraní, které významně zjednodušuje základní operace a práci při zprostředkování komunikace mezi aplikacemi (moduly). Poskytnuté rozhraní nemůže pokrýt všechny aspekty použití, a tak může nastat situace, kdy je třeba použít interní funkce a metody knihovny pro implementaci požadované komunikace. K dispozici jsou následující volání.

- **Inicializace** komunikačního rozhraní.

```
trap_init (trap_module_info_t *module_info,
           trap_ifc_spec_t ifc_spec)
```

- **Navázání komunikace a přijetí dat**.

```
trap_recv (uint32_t ifcidx, const void **data,
           uint16_t *size)
```

- **Odeslání dat**.

```
trap_send (uint32_t ifcidx, const void *data,
           uint16_t size)
```

- **Ukončení** všech aktivních rozhraní a komunikace.

```
trap_terminate ()
```

- **Nastavení** komunikačních parametrů knihovny.

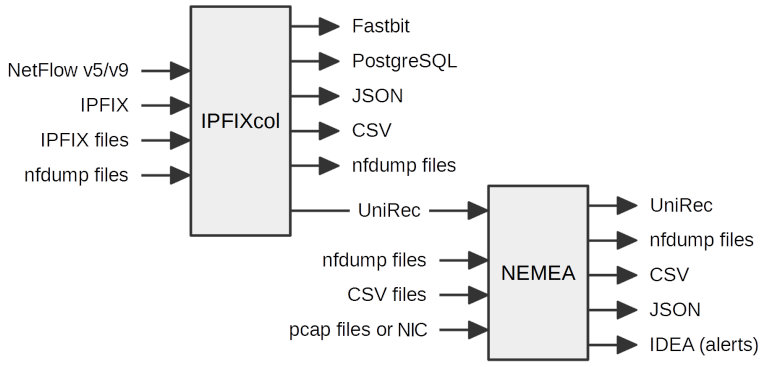
```
trap_ifcctl (int8_t type, uint32_t ifcidx,
             int32_t request,...)
```

- **Uvolnění prostředků** knihovny

```
trap_finalize ()
```

3.3 Zdroj dat

Běžné síťové prvky v současné době nepodporují export dat ve formátu UniRec, používaném moduly uvnitř NEMEA systému. IPFIXcol (IPFIX kolektor) dokáže exportovat data ve více formátech (Obrázek 3.5). Jednou z dostupných přídavných částí kolektoru je flow exportér pro UniRec formát. Kolektor je proto jedním z nejdůležitějších prvků stojící na samém okraji celého systému.



Obrázek 3.5: Schopnost systému spolupracovat i s infrastrukturami jiného exportního formátu [15].

3.4 Existující moduly

Systém NEMEA obsahuje moduly, které jsou schopny detekce různých typů podezřelého provozu, počítání statistik provozu, filtrování určitých typů zpráv nebo hlášení detekovaných incidentů. Funkcionalita každého modulu je přesně zaměřena na konkrétní problematiku, či oblast problematiky. Vývoj všech modulů je ovlivněn politikou projektu a výsledné moduly musí splňovat požadované parametry. Každý modul používá stejný NEMEA framework, který implementuje komunikační funkce mezi moduly, používaný datový formát, často využívané datové struktury a algoritmy.

Mezi existujícími moduly jsou i takové, které jsou schopny určitým způsobem data mezi vstupem a výstupem agregovat. Ve většině případů se tato data agregují rovnou během zpracování, za účelem využití k analýze specifického bezpečnostního incidentu, a nejsou součástí výstupu modulu.

3.4.1 Vportscan_detector

Neboli vertical port scan detector. Tento detekční modul odhaluje vertikální port scan, což je případ, kdy zdrojová adresa prochází více portů na stejné cílové IP adrese. Dle nastavené prahové hodnoty, představující počet otestovaných portů vůči konkrétní adrese, pak reportuje aktivní scan.

```
Klíč: SRC_IP, DST_IP
Počet unikátních: DST_PORT
```

V tomto případě se jedná o specializovanou agregaci příchozích zpráv s příslušnými příznaky (modul tedy provádí i filtraci), kde jako klíč působí dvojice zdrojová a cílová IP adresa, ke které se sledují dotazované cílové porty.

3.4.2 Haddrscan_detector

Neboli horizontal scan detector. Tento detekční modul odhaluje horizontální port scan, což je případ, kdy zdrojová adresa testuje stejný cílový port na více cílových IP adresách. Dle nastavené prahové hodnoty, představující počet otestovaných adres se stejnou hodnotou cílového portu, pak reportuje aktivní scan.

```
Klíč: SRC_IP, DST_PORT
Počet unikátních: DST_IP
```

V tomto případě se také jedná o specializovanou agregaci příchozích a filtrovaných zpráv, kde jako klíč figuruje dvojice zdrojová IP adresa a cílový port, ke které se sledují dotazované cílové adresy.

3.4.3 Scalar-aggregator

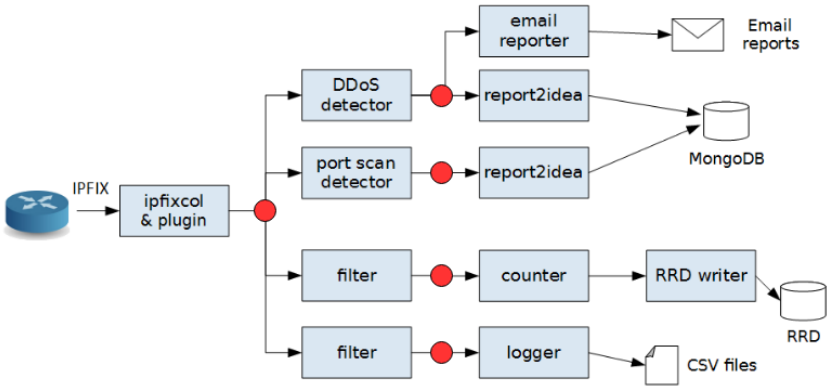
Jedná se o agregační modul univerzální z pohledu definice libovolných políček obsažených ve vstupním datovém proudu. Modul provádí operace `SUM`, `COUNT`, `AVG`, `RATE`, `COUNT_UNIQ`. Z pohledu nabídky agregačních funkcí téměř vyhovující modul. Jelikož výstupní záznam obsahuje pouze časový interval agregace a výslednou hodnotu agregovaného políčka, není vhodný pro použití, kdy je potřeba, aby data měla stejnou informační hodnotu na vstupu i výstupu. Agregační funkce podléhá pouze konkrétnímu filtrovacímu pravidlu. Tento princip je vhodný pro detekci, či hledání specifické vlastnosti, nikoli na univerzální použití pro skládání záznamů dle požadovaných obecných pravidel.

Kapitola 4

Návrh agregačního modulu

4.1 Požadavky

I přes existující skalární řešení agregace je v NEMEA systému potřeba univerzální agregace, která není závislá na konkrétním umístění a je použitelná v libovolném sektoru celého systému. Modul by měl podporovat agregační pravidla aplikovatelná na libovolnou sadu dat v UniRec formátu a odesílat data dle pravidel exportu datových toků specifikovaných v Sekci 1.3.2. Na Obrázku 4.1 jsou naznačena možná umístění nového agregačního modulu. Lze říci, že modul může být nasazen před či za libovolným modulem, který přijímá nebo odesílá data v podobě UniRec zpráv.



Obrázek 4.1: Ukázka možného nasazení agregačního modulu ve struktuře uvedené na Obrázku 3.1. Místo nasazení označuje červená tečka.

Tento modul by měl být zároveň schopný pracovat nejlépe v reálném čase, aby mohl být použit na živé infrastruktuře bez zpožďování dat procházejících skrze modul o jiné než nastavené časové intervaly. Modul by měl umět agregovat libovolná specifikovaná políčka UniRec záznamu dle jim přiřazené funkce. Samozřejmostí je umožnění určitých způsobů agregace pouze tam, kde má

takové použití logický význam a výsledkem není pouze aritmeticky správný nesmysl. Mezi podporované agregační funkce by měly patřit:

- **Suma** provádějící celkový součet hodnot políčka v přijatých zprávách.
- **Průměrná hodnota** daného políčka ve všech přijatých zprávách.
- **Minimum/Maximum** hodnoty pole mezi přijatými zprávami.
- **Bitové operace** s daným políčkem prováděné s každou přijatou zprávou.
- **Ponechat** původní první nebo poslední hodnotu políčka.

Klíčem k agregaci je opět uživatelem definovaná sekvence UniRec políček, jejichž hodnoty plní roli agregačního klíče.

4.2 Datové typy a transformace

Z analýzy struktury UniRec záznamů (Sekce 3.1) a údajů potřebných k implementaci modulu byly navrženy následující datové struktury, které jsou určeny k uchování uživatelské konfigurace (Sekce 4.2.1), držení potřebných informací o UniRec políčkách (Sekce 4.2.2) a uchování hodnot z přijatých záznamů. Hodnoty jsou v těchto strukturách i modifikovány a připravovány k exportu.

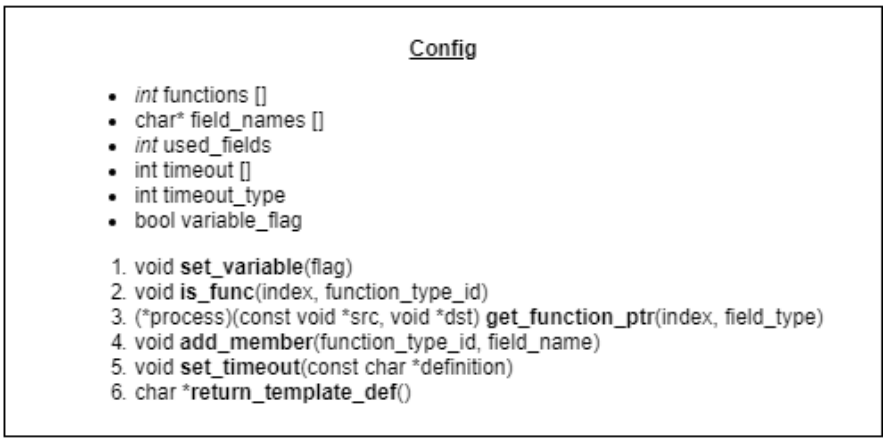
4.2.1 Uživatelská konfigurace

Jedním z požadavků je uživatelsky přívětivý způsob konfigurace modulu, který spočívá v uložení určitých dat získaných od uživatele do proměnných či hodnot používaných modulem k dalšímu zpracování dat. Po zamyšlení nad otázkou co vše modul umožní uživateli nastavit a jakou minimální množinu hodnot musí mít nastavenou k tomu, aby mohl správně plnit svou funkci, vyvstaly parametry detailněji zobrazené na Obrázku 4.2.

Modul k agregaci využívá definovaných funkcí, jejichž příslušnost k danému políčku určí uživatel při spuštění. Návrh vychází z myšlenky pole hodnot, na jehož příslušném indexu se nachází nastavení patřičného UniRec políčka. Do definice UniRec políčka `i` tedy patří:

- `functions[i]` - náležející agregační funkce
- `field_names[i]` - název políčka, který bude spárován s daty z příchozího a výstupního rozhraní.

Stejně jako dvojici hodnot definice políčka a funkce lze definovat vlastnosti modulu ohledně exportu agregovaných datových toků (Sekce 1.3.2) a metody manipulace s daty (Sekce 4.3). Tyto vlastnosti jsou zahrnuty v podobě pojmu `timeout`. Modul definuje 4 typy exportů (timeouts).



Obrázek 4.2: Návrh datové struktury, která reprezentuje uživatelem modifikovatelnou část modulu ovlivňující jeho činnost.

- **Aktivní** - Implementace aktivního ukončení toku (Sekce 1.3.2).
- **Pasivní** - Implementace pasivního ukončení toku (Sekce 1.3.2).
- **Globální** - Implementace možného scénáře *cache flush* (Sekce 1.3.2).
- **Mix** - Reprezentuje kombinaci aktivního a pasivního, kdy jsou oba spuštěny zároveň s možným nastavením různých hodnot, jelikož by bylo vhodné detekovat i dlouho neaktivní (neaktualizované) záznamy během právě používaném aktivním timeoutu a předejít tak stárnutí takovýchto záznamů v paměti modulu.

Tyto možnosti i s délkami jejich intervalu jsou ukládány opět jako jednoduché pole hodnot časového intervalu pro každý typ zvlášť. Hodnota typu exportu `x` určuje index právě platného intervalu.

- `timeout[]` - Pole hodnot délek pro jednotlivé časové intervaly.
- `timeout_type` - Hodnota označující nastavený typ exportu (timeout).

Za účelem podpory políček s variabilní délkou existuje detekce vlastnosti políčka, a to právě délky. V případě zjištění přítomnosti libovolného políčka proměnné délky je tato situace signalizována hodnotou `variable_flag`.

Uživatelský vstup musí být validován a zpracován do nastavení požadovaných hodnot. Nejlepším způsobem pro spojení navržené třídy s přijatými daty je využití třídních metod, které mají, díky členství, přístup k soukromým proměnným třídy. Metody potřebné pro manipulaci s daty do výše popsaných podob jsou následující:

- `add_member()` - Slouží k přidání nově definovaného páru (název, funkce) políčka do seznamu modulu.

- `set_timeout()` - Přijaté hodnoty definující nastavení typu exportu a hodnoty intervalu musí být zpracovány a validní hodnoty přijaty do konfigurace modulu.
- `return_template_def()` - Z přijatých definic políček, které se účastní agregačních metod, je třeba uspořádat definici políček použitých k vytvoření šablony pro výstupní rozhraní.

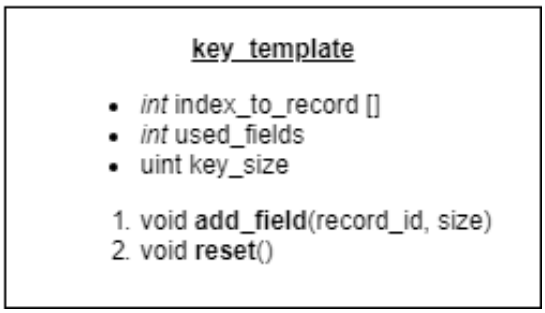
4.2.2 Manipulace s daty

Data jsou v modulu reprezentována využitím dvou úrovní abstrakce. Nejprve jsou modulem na základě uživatelského vstupu generovány vstupní a výstupní šablony, které obsahují tzv. metadata, tedy informace o výsledné podobě používaných dat. Šablony agregačního modulu slouží, stejně jako v návrhu UniRec formátu (Sekce 3.1), k rychlejšímu přístupu k datům a zároveň také k tvorbě datových struktur, kde jsou konkrétní hodnoty uchovávány.

Jak již samotný princip agregace vyžaduje, je zapotřebí definovat si data, dle kterých se bude agregovat. Tato data budeme dále nazývat *klíčem*, který je reprezentován strukturou `key`.

Šablony agregačního modulu

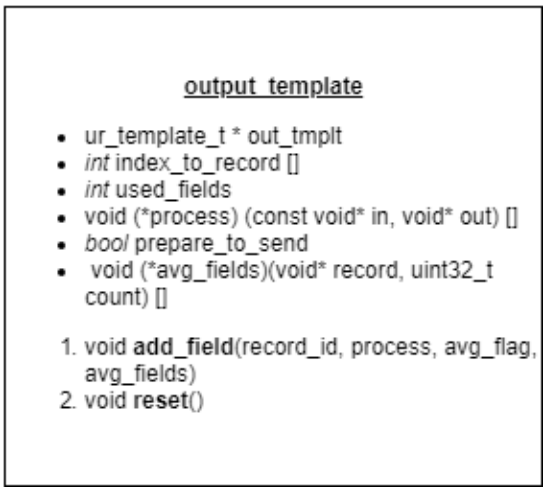
Modul využívá dvou definovaných šablon, a to šablony klíče (`key_template`) a polí určených k provádění agregace (`output_template`).



Obrázek 4.3: Návrh datové struktury, která reprezentuje uživatelem definovanou podobu klíče.

Šablona obsahující metadata klíče popisuje UniRec políčka tvořící klíč. Klíč je obecně platný pro všechny přijaté záznamy. Definujme název struktury jako `key_template`, potom je návrh její podoby zobrazen na Obrázku 4.3. Za políčka tvořící klíč jsou považována pouze taková, která mají pevnou a předem danou velikost, jejíž hodnota může být zaznamenána jako součást celkové délky klíče uložené v šabloně. Celková velikost klíče, představující součet velikostí jednotlivých obsažených políček, je uložena v hodnotě `key_size`. Z důvodu rychlosti generování samotného klíče obsahuje šablona informace

o tom, kde přesně se zúčastněná políčka v příchozích zprávách nachází. Tyto informace jsou uloženy v poli hodnot `index_to_record[]`, kde každý `i`-tý prvek odkazuje na index daného prvku do přijaté UniRec zprávy. Tento prvek slouží v podstatě jako mezipaměť (cache) pro opakované vyhledávání stejných políček v UniRec zprávách. Odpadá tedy potřeba vyhledání indexu dle definovaného jména skrze UniRec API (Sekce 3.1.2). Existence této šablony je podmíněna jejím generování, které probíhá pomocí metody `add_field()`.



Obrázek 4.4: Návrh datové struktury, která reprezentuje definice agregačních procesů.

Nejvýznamnější schopností modulu je agregace dat dle již zmíněného klíče. I tato vlastnost je definována uživatelem a je vhodné ji pro rychlejší průběh reprezentovat šablonou. Šablona je opět obecně platná pro všechny přijímané UniRec záznamy. Definujme ji jako datovou strukturu `output_template`, zobrazenou na Obrázku 4.4. Nejdůležitější část této struktury vychází, stejně jako v případě konfigurace (Sekce 4.2.1), z návrhu jednoduchého pole hodnot, kde pro každé definované `i`-té UniRec pole existují hodnoty určující jeho umístění ve výstupním záznamu a jemu přiřazenou agregační funkci.

- `index_to_record[]` - Hodnota indexu daného pole ve výstupní šabloně nahrazující definici políčka názvem. Opakovaný přístup je rychlejší než za využití překladu skrze UniRec API.

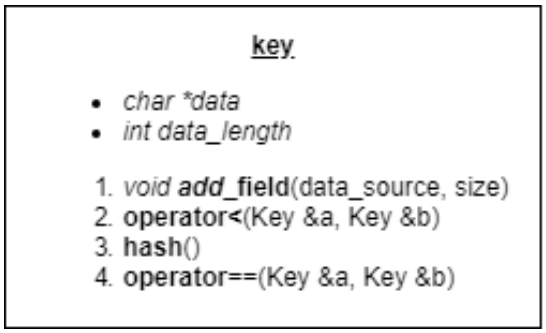
- `process[]` - Agregační funkce přiřazená danému prvku na `i`-tém indexu.

Další informace o zpracování dat se týkají zpracování agregační funkce průměrné hodnoty. Tato funkce byla již od počátku navržena pro použití s metodikou *post processing*, tedy závěrečnému zpracování dat před exportem (odesláním). Za tímto účelem existuje pole `avg_fields[]`, které představuje funkci průměrné hodnoty přiřazené prvku na daném indexu a hodnota `prepare_to_send`,

která označuje potřebu provést dodatečné zpracování dat před exportem. Existence šablony `output_template` je, stejně jako v případě šablony klíče, podmíněna generováním na základě vstupu. Metoda `add_field()` je navržena za účelem registrace nového prvku (UniRec pole) ke zpracování patřičnými agregačními metodami.

Datové struktury

Vlastní realizace klíče spočívá na struktuře `key` (Obrázek 4.5), která je vytvořena na základě informací obsažených v šabloně `key_template` (Obrázek 4.3).



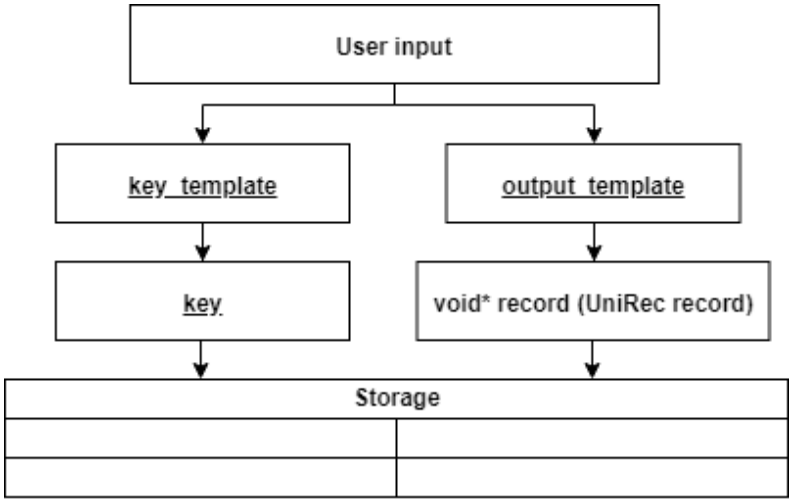
Obrázek 4.5: Návrh datové struktury, která reprezentuje agregační klíč naplněný daty z přijatých UniRec záznamů.

Jedná se o datovou sadu, která připomíná datový typ `string`, ale liší se definovanými specifickými vlastnostmi dodanými skrze navržené metody. Klíč se skládá z pole znaků uvedených jako `data` spolu s hodnotou délky zapsaných dat `data_length`. Za znaky jsou považovány informace v podobě bajtů nikoli jejich ASCII reprezentace. Pomocí metody `add_field()` jsou hodnoty políček z přijatých zpráv definovaných v šabloně postupně překopírovány do souvislého pole bajtů, které je nadále považováno za unikátní klíč k provádění agregace. Ostatní uvedené metody včetně operátorů pouze definují chování této datové struktury potřebné pro správnou manipulaci během implementace. Samotná operace `hash()` představuje metodu transformace hodnoty v podobě pole bajtů do numerické podoby sloužící pro rychlejší přístup k datům uloženým v paměti modulu.

Data určená k agregaci a exportu jsou ukládána a modifikována přímo jako UniRec záznam (UniRec record). Tato realizace je zvolena především kvůli rychlosti, s jakou může modul data exportovat bez jejich další potřebné konverze z vlastních struktur. Ukládání dat v podobě UniRec záznamu znamená, že do této struktury již nelze přidávat hodnoty, jenž nejsou součástí výstupu.

Na Obrázku 4.6 lze najít shrnutí navržených datových struktur pro manipulaci s daty i s jejich zamýšleným způsobem použití. Klíč slouží jako iden-

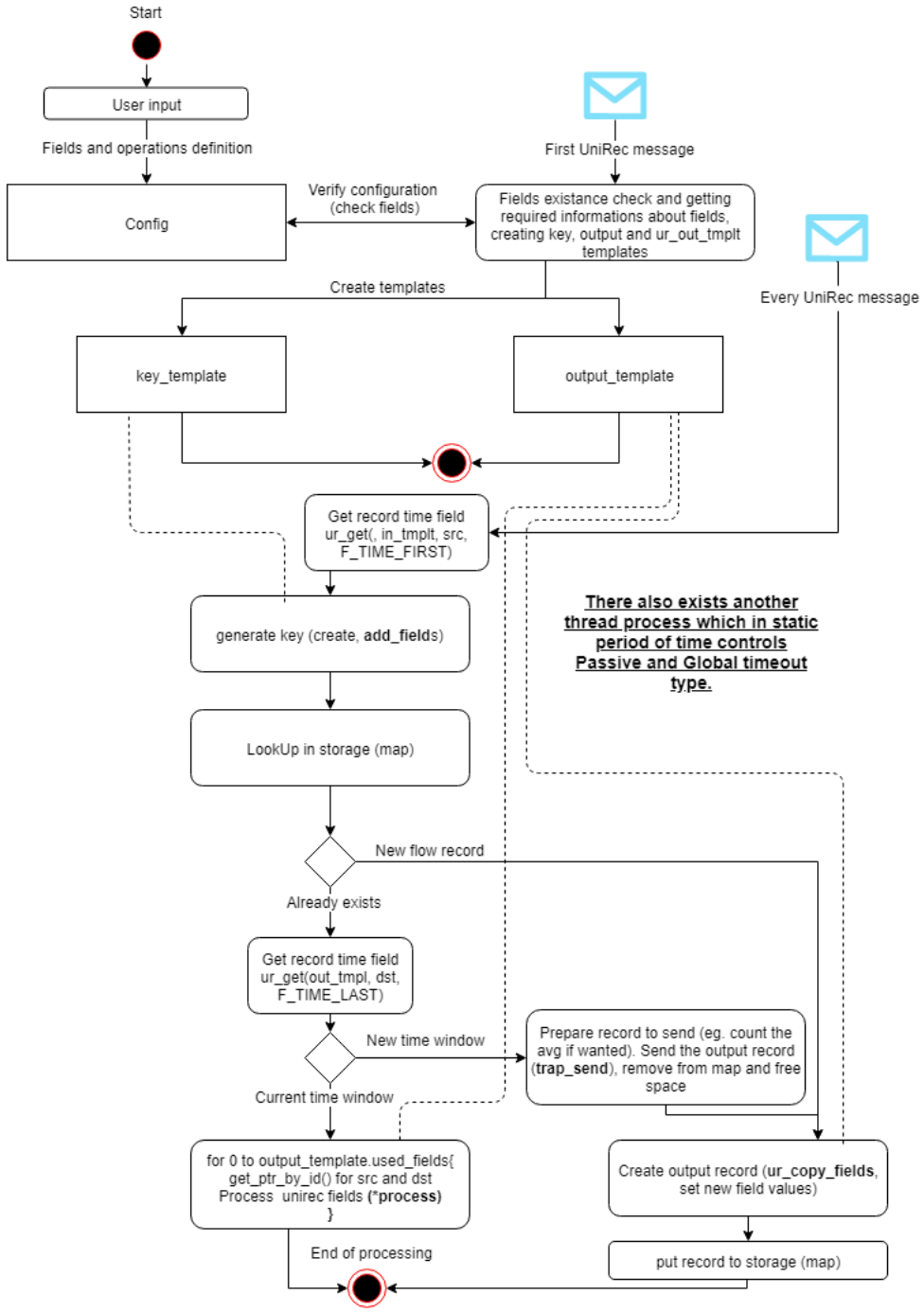
tifikace agregovaných záznamů. Výstupní šablona drží přehled o agregačních funkcích, které mají být použity na jednotlivá políčka obsažená v UniRec záznamu identifikovaného právě jedním klíčem.



Obrázek 4.6: Zobrazení závislostí a použití jednotlivých datových struktur modulu pro manipulaci s daty.

4.3 Zpracování dat

Celý princip fungování modulu včetně datových proudů pro manipulaci s daty je zobrazen na Obrázku 4.7. Popisy funkcí jsou orientační a nepředstavují kompletní podobu existujícího příkazu či volání API. Obdélníky reprezentují datové struktury a zaoblené hrany označují procesy.



Obrázek 4.7: Návrh kompletního zpracování příchozích dat při průchodu modulem až k jejich exportu. Oblé rohy reprezentují akce nebo procesy, obdélníky reprezentují objekty. Přerušovaná čára mezi objektem a procesem znamená využití objektu procesem, ke kterému směřuje.

Vše začíná spuštěním modulu a předáním vstupních parametrů.

- Definice potřebných LibTrap rozhraní (Sekce 3.2). Modul je navržen pro použití s právě dvěma rozhraními.
  1. Vstupní
  2. Výstupní
- Definice UniRec políček:
  - Název a přiřazení role agregačního klíče.
  - Název a přiřazení agregační funkce.
- Nastavení typu exportu (timeout) a délky intervalu agregace.

Takto získané hodnoty jsou uloženy do již zmíněné struktury `Config` (Sekce 4.2.1). Modul poté čeká na úspěšné navázání spojení skrze definované vstupní rozhraní. Když obdrží první zprávu obsahující používanou šablonu pro tuto komunikaci, spustí ověření vstupu od uživatele. V případě že jsou všechna definovaná políčka součástí komunikační šablony, pokračuje se dalším krokem. Pokud je detekováno zadané políčko, které se mělo účastnit agregace, ale není součástí přijímaných zpráv, modul skončí s chybou, kterou reportuje uživateli. V době validace dochází k postupnému vytváření finální podoby používaných šablon `key_template` a `output_template` (Sekce 4.2.2). Podrobnější postup validačního a zároveň inicializačního procesu je popsán v Sekci 4.3.1.

Po úspěšném navázání komunikace a zpracování šablony modul ve smyčce čeká na UniRec záznam, který ihned po přijetí zpracovává. Po obdržení zprávy dojde k získání časových údajů o daném datovém toku (jeho začátek a konec). Na základě vygenerované šablony pro klíč dojde k vytvoření agregačního klíče z patřičných políček přijatého UniRec záznamu (Sekce 4.3.3). Následně je úložiště modulu dotázáno na existenci vytvořeného klíče a k němu patřícího UniRec záznamu. Zde se chování modulu rozděluje dle stavu existence záznamu v úložišti.

- **Záznam již existuje**, pak dojde k porovnání časových údajů mezi přijatým záznamem a uloženým. Dle nastaveného typu exportu se vyhodnotí, zda je třeba před již existující záznam připravit k odeslání a exportovat, či se provede zpracování všech přidělených agregačních funkcí (Sekce 4.3.5), jejichž výsledek je uložen do existujícího záznamu.
- **Záznam je nový**, tudíž dojde k vytvoření nového prázdného záznamu, do něhož jsou nakopírována všechna zúčastněná políčka agregace (Sekce 4.3.4). Hotová kopie je poté uložena s daným klíčem do úložiště.

Dokončením zpracování všech agregačních funkcí nebo uložením nových dat končí životní cyklus přijatého záznamu. Pro zvýšení rychlosti zpracování dat se po exportu uložený záznam neuvolňuje z úložiště, ale je pouze přepsán novými daty. O exportu dat v případě aktivovaného jiného než aktivního typu exportu rozhoduje nezávislý proces, který v definovaných periodách dle specifikací ukončení datových toků (Sekce 1.3.2) provádí odeslání vybraných záznamů.

Následující sekce se věnují podrobnějšímu návrhu jednotlivých zde zmíněných procesů, které ke svému naplnění vyžadují provedení více spolupracujících činností.

4.3.1 Kontrola vstupu a tvorba šablon

Tento proces, znázorněn na Obrázku 4.8, je první akce modulu po zpracování uživatelského vstupu do podoby konfigurace modulu a úspěšného navázání komunikace. Představuje validaci uživatelského vstupu vůči přijaté komunikační šabloně za současného generování potřebných šablon `key_template` a `output_template`.

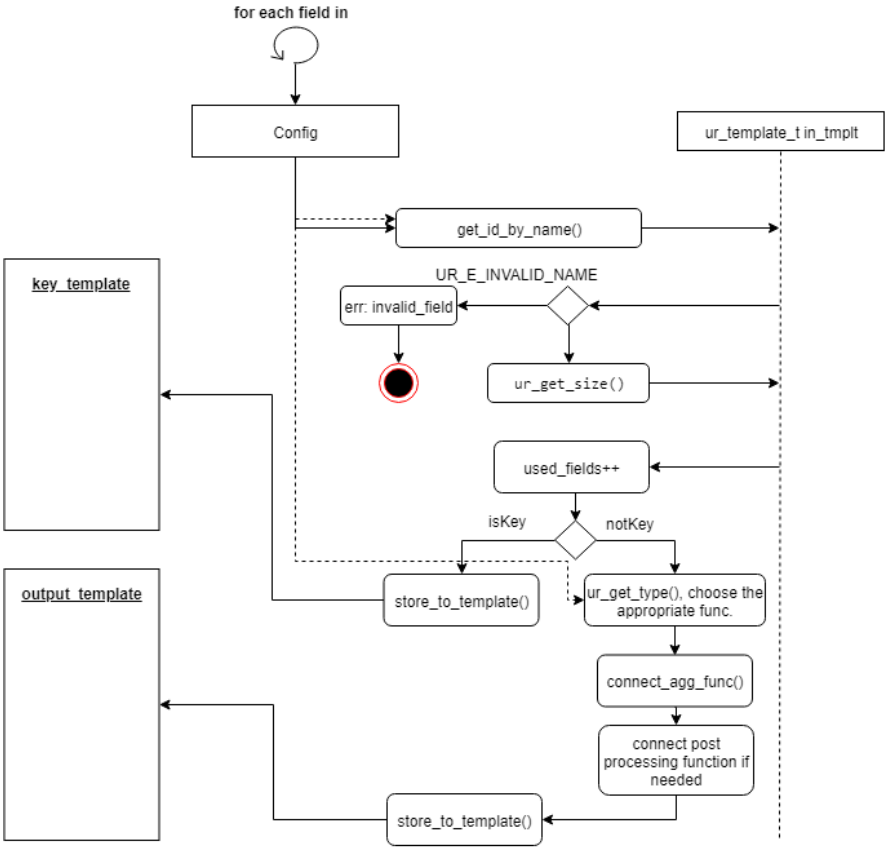
Konfigurace modulu disponuje jmény políček, které nejsou příliš efektivní pro opakované použití. Proto se během ověřování získává číselná hodnota jejich pořadí (index) v globální struktuře UniRec záznamu, která je poté ukládána do šablon a slouží jako hodnota mezipaměti (cache) pro rychlejší přístup k datům. Validace probíhá pro každé definované UniRec políčko s využitím dostupných API volání dle následujícího postupu.

1. Dotaz na index pole určeného názvem a následné vyhodnocení jeho existence dle návratové hodnoty.

2. Pro potřeby modulu následuje detekce, zda se jedná o variabilní, či pevnou délku.

3. Dle hodnoty přidružené funkce se detekuje použití v agregačním klíči, nebo účast v procesu agregace.

   - **Agregační klíč** - pole je přidáno do šablony klíče (`key_template`) i s hodnotou délky pole.
   - **Agregační funkce** - K danému poli je dle uživatelem zadané hodnoty přiřazena implementace agregační funkce pro konkrétní datový typ z aktuálně podporovaných (Tabulka 3.1), který je získán opět skrze volání API. Hodnoty jsou poté přidány do výstupní šablony (`output_template`).



Obrázek 4.8: Zpracování procesu ověření uživatelské konfigurace a generování šablon. Oblé rohy reprezentují akce nebo procesy, obdélníky reprezentují objekty. Přerušovaná čára mezi objektem a procesem znamená využití objektu procesem, ke kterému směřuje.

4.3.2 Tvorba výstupní UniRec šablony

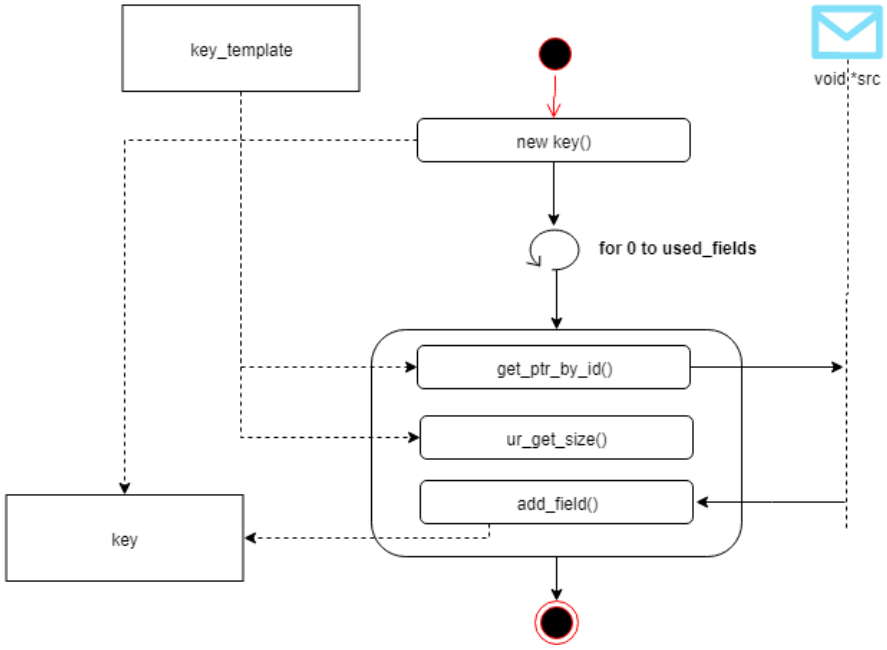
Po dokončení procesu ověření uživatelského vstupu jsou k dispozici finální podoby šablon. K tomu, aby se data mohla exportovat skrze výstupní rozhraní definované s pomocí knihovny LibTrap, je třeba definovat výstupní šablonu, která bude použita během navázání spojení s cílovým rozhraním. Proces tvorby výstupní šablony vychází ze zpracování uživatelského vstupu do vhodné konfigurace modulu, a toho že knihovna LibTrap během zahájení komunikace sama aktualizuje dostupná políčka v globální struktuře. Využitím této vlastnosti se proces definice výstupní šablony zjednoduší pouze na jedno API volání, do kterého jako parametr vstupuje seznam všech políček s nimiž modul manipuluje (klíčová pole, pole určená k agregaci, časové značky a počet agregovaných záznamů).

4.3.3 Generování klíče

K využití agregace je třeba každý záznam určitým způsobem identifikovat. Klíč (`key`, Obrázek 4.5) představuje podobu dat, která slouží k jednoznačnému rozlišení záznamů. Dle metadat v šabloně (`key_template`) je postupně zkonstruována datová struktura obsahující data z definovaných klíčových UniRec políček. Následující postup je možno pozorovat i na Obrázku 4.9. Po vytvoření prázdného objektu reprezentující klíč dochází k iteraci přes všechna klíčová políčka v šabloně klíče, nad kterými je prováděná sekvence akcí.

- Získání místa v záznamu kde jsou data daného políčka.
- Ověření velikosti daného bloku dat
- Volání metody přidání klíče, která dle vyhovujících postupů překopíruje data do vymezeného bloku paměti struktury klíče.

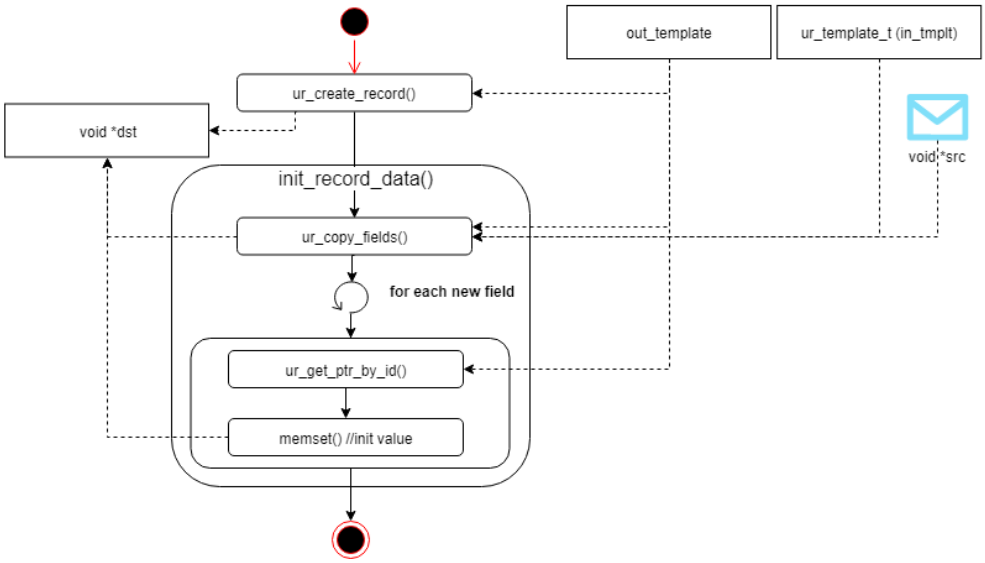
Po dokončení zpracování všech polí je již podoba klíče kompletní a připravena k použití pro identifikaci záznamů v paměti modulu.



Obrázek 4.9: Detail procesu tvorby agregačního klíče dle definované šablony klíče. Oblé rohy reprezentují akce nebo procesy, obdélníky reprezentují objekty. Přerušovaná čára mezi objektem a procesem znamená využití objektu procesem, ke kterému směřuje.

4.3.4 Tvorba UniRec záznamu

Při identifikaci přijatého záznamu daného klíče, který nebyl nalezen v úložišti modulu, je třeba vygenerovat nový záznam, jenž bude následně spojen s daným klíčem a uložen. Tento záznam se vyskytuje v podobě definované šablonou, která byla přenesena knihovnou LibTrap při zahájení komunikace mezi vstupním a výstupním komunikačním rozhraním. Takový způsob byl zvolen v důsledku zkrácení doby exportu daného flow záznamu, kdy již není zapotřebí žádné další datové konverze. UniRec API nabízí funkce, které umožňují kopírování dat z jednoho záznamu do jiného s dostatečným datovým prostorem dle přiložené UniRec šablony. Celá tvorba záznamu spočívá ve vymezení dostatečného datového prostoru pro data z příchozího záznamu a jejich nakopírování do nového záznamu v podobě výstupní UniRec šablony. Datová pole definovaná samotným modulem (v současnosti pouze hodnota agregovaných záznamů) jsou nastavena na své inicializační hodnoty. K tvorbě nového záznamu dochází především z důvodu libovolné výstupní šablony, která je definována uživatelem při spuštění. Předpokládá se, že na výstupu modulu bude použita odlišná UniRec šablona, než je využívaná na vstupu. Po dokončení inicializace je záznam připraven k dalšímu zpracování nebo odeslání. Postup pro vytvoření cílové podoby dat je možné vidět také na Obrázku 4.10.



Obrázek 4.10: Inicializace nového záznamu z přijatých dat. Oblé rohy reprezentují akce nebo procesy, obdélníky reprezentují objekty. Přerušovaná čára mezi objektem a procesem znamená využití objektu procesem, ke kterému směřuje.

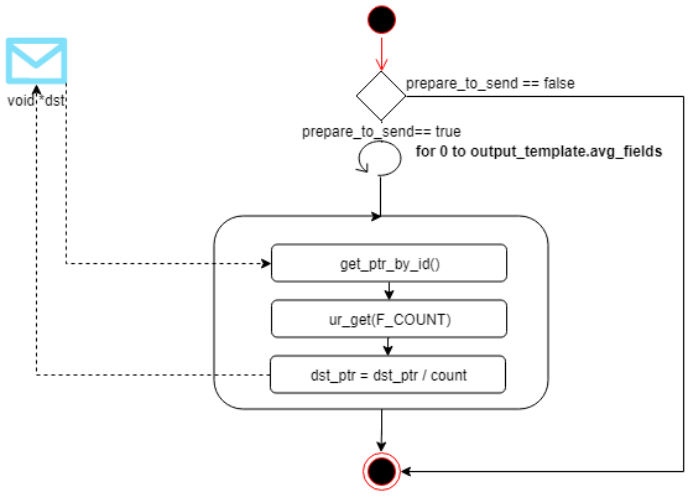
4.3.5 Zpracování UniRec záznamu

Pod tímto procesem se skrývá veškerá agregační činnost modulu. Jedná se o aplikaci přidružené agregační funkce napříč všemi políčky obsaženými ve výstupní šabloně (`output_template`). Existující agregační funkce neprovádí žádné výpočetně ani návrhově složité operace. Dle požadavků vypsaných v Sekci 4.1 modul implementuje jednoduché aritmetické a logické operace.

Po nalezení požadovaného záznamu v úložišti modulu je vždy automaticky inkrementována hodnota `COUNT` označující počet zpracovaných záznamů s identickým klíčem. Poté jsou dle výstupní šablony v cyklu volány přiřazené agregační funkce, které manipulují s uloženými a právě příchozími daty. Dokončením poslední agregační funkce je proces běžného zpracování záznamu dokončen.

4.3.6 Postprocessing

Některé agregační funkce, jako například výpočet průměrné hodnoty, mohou vyžadovat dodatečné zpracování některých dat až v momentě, kdy je známo, že konkrétní záznam již nebude aktualizován. V takovém případě výstupní šablona obsahuje ukazatel potřeby tohoto zpracování. Tato hodnota je ověřována po označení skladovaného záznamu jako expirovaného. Pokud je finální zpracování aktivní, dojde k manipulaci se všemi políčky, které mají takovouto funkci přiřazenou. Procházení políček se provádí v cyklu na základě výstupní šablony (`output_template`) stejně jako v případě běžného zpracování agregačních funkcí (Sekce 4.3.5).



Obrázek 4.11: Ukázka finálního zpracování záznamu před odesláním v případě aktivní agregační funkce průměru. Oblé rohy reprezentují akce nebo procesy.

4.4 Architektura

Celý NEMEA framework je dostupný ve dvou implementačních architekturách. Pro rychlý vývoj bez nutnosti studovat používané datové typy a strukturu nabízeného řešení je připravena knihovna pro skriptovací jazyk Python zvaná *Pytrap*. Druhým řešením je programovací jazyk C, ve kterém jsou potřebné knihovny optimalizovány za účelem rychlosti zpracování pomocí maker pro preprocesor. Vzhledem k potřebné rychlosti zpracování příchozích dat byla zvolena varianta programovacího jazyka C/C++. V implementaci modulu nalezneme prvky čistého jazyka C, které reprezentují zahrnuté hlavičkové soubory a knihovny pro manipulaci s UniRec formátem a LibTrap. Prvky jazyka C++ jsou obsaženy především ve vlastním návrhu objektových datových struktur a uzavřené manipulaci s nimi pouze skrze definované třídní metody. Pro obecnou reprezentaci procesů v rámci modulu je použito i procedurální programování (funkce), jelikož se procesy vždy nevztahují ke konkrétním datovým strukturám.

V modulu chceme ukládat pouze unikátní flow záznamy specifikované daným klíčem. Pro toto použití je vhodný standardizovaný kontejner *map*. Pro návrh modulu bylo využito předpokladu opakovaných přístupů k již skladovaným záznamům. Vzhledem k tomu byla jako hlavní datové úložiště modulu zvolena *hash* mapa, kde přístup k datům je okamžitý ($\mathcal{O}(1)$), oproti klasické implementaci využívající seřazenou posloupnost prvků s dobou přístupu $\mathcal{O}(log(n))$. K návrhu byla tedy využita varianta s teoreticky větší datovou propustností.

Modul je navržen pro orientaci v jednotkách sekund. Přesnost v jednotkách sekund vyhovuje použití v reálném nasazení, kde jsou časové intervaly běžně voleny v řádu minimálně desítek sekund. Tento předpoklad vychází i z definic standardů exportních formátu, kde je ukončení datových toků (Sekce 1.3.2) prováděno s časovým intervalem v rámci jednotek sekund.

Kapitola 5

Implementace agregačního modulu

Tato kapitola popisuje výslednou implementaci agregačního modulu na základě vytvořeného návrhu. Vysvětluje proč a jakým způsobem byly vybrané procesy a datové typy implementovány. V následujících sekcích jsou také ukázány nedostatky již připravených funkcí a maker vytvořených pro rychlejší vývoj NEMEA modulů. Celá implementace je dle návrhu architektury provedena v programovacím jazyce C++ za přítomnosti některých importovaných řešení v programovacím jazyce C. V důsledku využití standardizovaného kontejneru `unordered_map` využívajícího hash hodnoty k identifikaci záznamu, je verze použitá pro překlad stanovena na C++11.

Modul ke svému běhu využívá vícevláknový přístup zpracování. Jednotlivá vlákna jsou synchronizována z pohledu přístupu k datům, jinak jsou výpočetně zcela nezávislá. Modul také pracuje s časovými záznamy, a je tedy závislý na systémovém čase. Libovolná změna systémového času, v případě již spuštěného modulu, může ovlivnit způsob, jakým modul manipuluje s příchozími záznamy i export uložených dat. Modul nemá implementovanou detekci podobných událostí, a tak je detekce případného podezřelého chování ponechána na samotném uživateli.

5.1 Typy exportu

Jednou z hlavních předností modulu je podpora více typů exportu (timeouts), které lze libovolně nastavit. K danému typu lze také zvolit libovolnou délku časového okna, která v kombinaci s časovým údajem záznamu vytváří interval k rozhodování o platnosti záznamu dle aktivního typu exportu. Implementace byla vytvořena na základě stanovených pravidel zmíněných ve zdroji [7], týkajícího se datových toků (Sekce 1.3.2). Modul podporuje *aktivní*, *pasivní*, *globální* a *mix* typy exportu datových toků. Zpracování dat dle zmíněných

pravidel je implementováno pomocí dvou POSIX vláken. Vlákna mají na starosti hlídat stanovená pravidla nastaveného typu exportu. Každé vlákno může provádět právě jeden typ exportu. Manipulace prováděné jednotlivými vlákny jsou zcela výpočetně nezávislé, což umožňuje použití kombinace *aktivního* a *pasivního* exportu nazvaného *mix*, kdy jsou obě pravidla prováděna odděleně jednotlivými vlákny.

5.1.1 Aktivní timeout

Kontrolu tohoto typu exportu provádí hlavní vlákno. Pokud modul přijme zprávu, u které již dle daného klíče eviduje uložený záznam a je nastaven *aktivní* typ exportu, vlákno ověřuje časové údaje dle podmínek splňujících definici tohoto pravidla. Stav, kdy se má daný záznam exportovat z úložiště pro délku časového okna daného hodnotou t_n je definován jako:

$$Stored(time\_first + t_n) < Received(time\_first). \tag{5.1}$$

V případě, kdy hodnota `time_first` přijatého záznamu je mimo interval $\langle$`time_first`, `time_first` $+ t_n\rangle$ stanovený uloženým záznamem, je záznam odeslán na výstupní rozhraní. Struktura uloženého záznamu není odstraněna z úložiště a nadále existuje. Její data jsou přepsána hodnotami z právě zpracovávané nové zprávy a hodnoty vytvářené modulem jsou nastaveny na původní hodnoty. Výsledkem tohoto scénáře je stav uloženého záznamu ekvivalentní nově vytvořenému záznamu bez nutnosti uvolňovat a opětovně alokovat potřebný paměťový prostor pro záznam. V opačném případě, kdy časový údaj spadá do daného intervalu, jsou hodnoty záznamu aktualizovány na základě definovaných agregačních funkcí a pokračuje se přijetím další datové zprávy na otevřeném komunikačním rozhraní. Zmíněné části procesu jsou ukázány ve Výpisu zdrojového kódu 5.1.

Výpis 5.1: Hlavní body zpracování aktivního typu exportu.

```
if (stored_first + config.get_timeout(TIMEOUT_ACTIVE) < record_first ) {
   new_time_window = true;
}
if (new_time_window) {
   if(!send_record_out(OutputTemplate::out_tmplt, stored_rec)) {
      break;
   }
   init_record_data(in_tmplt, in_rec, OutputTemplate::out_tmplt,
                    stored_rec);
}
else {
   process_agg_functions(in_tmplt, in_rec, OutputTemplate::out_tmplt,
                         stored_rec);
}
```

5.1.2 Pasivní timeout

Kontrolu provádí nové vlákno vytvořené hlavním. Tento typ exportu není nijak závislý na době přijetí zprávy. Kontrola tohoto typu exportu je závislá na systémovém čase. V pravidelných intervalech daných hodnotou t_n, přijatou od uživatele během nastavení modulu, se provádí kontrola všech uložených záznamů dle podmínek splňujících definici tohoto pravidla. Stav, ve kterém se má daný záznam označit za expirovaný a odeslat na výstupní rozhraní je vyjádřen jako:

$$Stored(time\_last + t_n) < current\_time. \tag{5.2}$$

Celý životní cyklus vlákna probíhá ve funkci `check_timeouts(void *input)` definované v main souboru *aggregator.cpp*. Nutno dodat, že hodnota reprezentována proměnnou *current_time* není aktuální systémový čas. Tato hodnota reprezentuje časovou informaci ovlivňovanou plynutím času (aktualizovanou skrze systémový čas), ovšem její původní hodnota pochází z první přijaté UniRec zprávy po navázaní komunikace.

Z toho vyplývá, že modul je na základě tohoto typu exportu schopen zpracovávat i záznamy opětovně přehrané z nějakého úložiště (soubor, databáze, atd.) a není omezen pouze na živé přenosy. V případě takto skladovaných záznamů modul zpracuje veškerý příchozí proud dat a odesílání dat probíhá průběžně pro všechny záznamy dle uplynulého času od přijetí prvního záznamu daného datového proudu. Toto zpracování má i své nedostatky, které vyplývají ze způsobu aktualizace aktuálního času (*current_time)*. V případě, že dojde k přehrání datového proudu, který obsahuje záznamy za časový úsek například 30 minut, lze očekávat export naposledy aktualizovaných záznamu až v době 30 $minut + t_n$.

Expirovaný záznam je odeslán na výstupní rozhraní a poté odstraněn z úložiště modulu, včetně uvolnění alokovaného paměťového prostoru. Zmíněné části procesu jsou ukázány ve Výpisu zdrojového kódu 5.2.

Výpis 5.2: Hlavní body zpracování pasivního typu exportu.

```
for (std::unordered_map<Key, void*>::iterator it = storage.begin();
     it != storage.end(); ) {
   if (ur_time_get_sec(ur_get(OutputTemplate::out_tmplt,
                              it->second, F_TIME_LAST))
       < time_last_from_record - timeout) {
      send_record_out(OutputTemplate::out_tmplt, it->second);
      ur_free_record(it->second);
      it = storage.erase(it);
   }
   else {
      ++it;
   }
}
```

5.1.3 Mix timeout

Existují situace, kde by bylo vhodné použít oba dva již zmíněné typy exportu najednou. Například pro pravidelné rozdělení dlouhých flow pomocí aktivního timeoutu po velikosti časového okna t_n, ale zároveň nechceme, aby nám v paměti modulu zůstávaly záznamy, které již nejsou aktivní po dobu větší t_m. Tento případ lze vyřešit právě typem *mix*, jenž implementuje tuto kombinaci. Stav, ve kterém dochází k exportu je dán vztahem:

$$(St(time\_first + t_n) < Recv(time\_first)) \wedge (time\_last + t_m < curr\_time). \tag{5.3}$$

Každé vlákno zpracovává samostatně jeden typ exportu. Hlavní vlákno kontroluje data z pohledu *aktivního* a nové vlákno vytvořené hlavním se stará o *pasivní* timeout.

Vše je implementováno pomocí rozhodovacího mechanismu, zobrazeného ve Výpisu zdrojového kódu 5.3, který na základě nastavené hodnoty spouští kontroly časových oken. Komentáře uvnitř výpisu označují pokračování zpracování dle *aktivního* typu (Výpis 5.1) a *pasivního* typu (Výpis 5.2).

Výpis 5.3: Rozhodovací body exportu typu mix.

```
/* Active timeout */
if ((config.get_timeout_type() == TIMEOUT_ACTIVE) ||
    (config.get_timeout_type() == TIMEOUT_ACTIVE_PASSIVE)) {
   /* Active timeout handle */
}
/* Passive timeout */
if ((timeout_type == TIMEOUT_PASSIVE) ||
    (timeout_type == TIMEOUT_ACTIVE_PASSIVE)) {
   int timeout = configuration->get_timeout(TIMEOUT_PASSIVE);
   /* Passive timeout handle */
}
```

5.1.4 Globální timeout

Globální typ exportu datových toků je prováděn novým vláknem vytvořeným hlavním. Reprezentuje situaci, kdy je potřeba v pravidelných intervalech stanovených hodnotou časového okna t_n všechny záznamy exportovat a vyprázdnit tak celé úložiště modulu pro zpracování dalšího časového okna. Tato podoba manipulace s daty vytváří dojem dávkového zpracování na výstupu modulu. Pravidlo pro splnění podmínky spuštění exportu je následující:

$$last\_export + t_n < current\_time. \tag{5.4}$$

Celý životní cyklus vlákna probíhá ve funkci `check_timeouts(void *input)` definované v main souboru *aggregator.cpp*. Na rozdíl od *pasivního* timeoutu hodnota *current_time* představuje aktuální systémový čas. Periodicita provádění exportu je zajištěna uspáním vlákna na rozdílovou hodnotu časového okna t_n a dobu provedení operace odeslání všech záznamů. Zajištění správné periodicity exportu je znázorněno ve Výpisu zdrojového kódu 5.4.

Výpis 5.4: Zajištění periodicity.

```
int elapsed = difftime(end, start);
int sec_to_sleep = (timeout - elapsed);
if (sec_to_sleep > 0)
   sleep(sec_to_sleep);
```

5.1.5 Synchronizace vláken

V současné době obě existující vlákna přistupují ke stejnému datovému úložišti, se kterým manipulují. Tyto přístupy je třeba synchronizovat, aby nedocházelo k neočekávanému chování modulu. Přístup k datům je řízen dle principu kritických sekcí vytvořených pomocí POSIX mutexů. Kritické sekce jsou implementovány zámkem na celou strukturu úložiště již od zahájení možné manipulace s daty.

K synchronizaci nedochází pouze během přístupu k uloženým datům, ale i v momentě aktualizace časových údajů používaných pro pasivní timeout. Jelikož modul umožňuje automatický restart po navázání nové komunikace na stejném vstupním komunikačním rozhraní, je třeba ošetřit kritickou sekcí i manipulaci s touto časovou proměnou, která označuje čas prvního přijatého záznamu aktualizovaný s plynutím systémového času. Nastavení této časové proměnné je ukázáno ve Výpisu zdrojového kódu 5.5.

Výpis 5.5: Aktualizace časové proměnné.

```
pthread_mutex_lock(&time_last_from_record_mutex);
time_last_from_record += (elapsed + sec_to_sleep);
pthread_mutex_unlock(&time_last_from_record_mutex);
```

5.1.6 Nastavení

Typ exportu a délka časového okna jsou nastavitelné skrze vstupní parametry při spuštění modulu. Modul akceptuje tři možné způsoby nastavení.

- **Základní nastavení** - Jedná se o `default`ní hodnoty modulu. Tento způsob je uplatněn, když uživatel nedefinuje žádné parametry nastavení exportu. Výsledná konfigurace modulu je:
  - **Typ exportu**: *Aktivní*
  - **Časové okno**: 10 sekund
- **Pouze časové okno** - Ponechá se `default`ní nastavení typu, ale změní hodnotu délky časového okna. Příklad použití parametru `-t "300"`. Výsledná konfigurace modulu je:
  - **Typ exportu**: *Aktivní*
  - **Časové okno**: 300 sekund

- **Kompletní redefinice** - Modul nastaví požadované hodnoty předané parametrem. Příklad použití parametru `-t "G:180"` je výsledná konfigurace:
  - **Typ exportu**: *Globální*
  - **Časové okno**: 180 sekund

Implementace zpracování takto získaných parametrů se vyskytuje jako metoda konfigurační datové třídy `Config` s názvem `set_timeout(const char *input)` v souboru *configuration.cpp*. Předaný řetězec znaků je zde rozdělen dle dělícího znaku „:“ a následně jsou aktualizovány hodnoty patřičných proměnných. V případě detekce nesprávného zadání jsou ponechány původní hodnoty a spuštění modulu pokračuje běžným způsobem.

5.2 Agregační funkce

Agregační funkce představují z hlediska požadavků uživatele hlavní činnost celého modulu. Pro dostatečnou jednoduchost implementace stávajících i případných nových funkcí jsou implementovány tak, aby dodržovaly jednotnou podobu definice návratového typu i parametrů. Tato unifikovanost umožňuje přiřadit všechny implementace agregačních funkcí do společného pole ukazatelů právě na tyto funkce. Volání a provedení samotné funkce je poté z pohledu implementace modulu vždy stejné.

Důsledkem toho je jednoduchost implementace nových agregačních funkcí, pro niž není potřeba znát či studovat implementaci modulu, která s agregačními funkcemi nesouvisí. Stačí dodržet stanovenou podobu definice funkce a následně ji doplnit mezi dostupné funkce modulu, aby ji bylo možné přiřadit k políčku. Dostupné funkce modulu jsou definovány jako součást konfigurace používané třídou `Config` v souboru *configuration.h*. Dalším přínosem této implementace je přehlednost a přímočarost kódu.

Modul v aktuální podobě implementuje dvě podoby funkcí.

1. Agregační funkce
   ```
   typedef void (*agg_func)(const void *src, void *dst);
   ```
2. Postprocessing funkce
   ```
   typedef void (*final_avg)(void *record, uint32_t count);
   ```

Ukazatel `*agg_func` na implementaci agregační funkce definuje již zmíněný formát. Tento ukazatel reprezentuje libovolnou agregační funkci dodržující požadované rozhraní.

src označuje ukazatel na přijatá data

dst označuje ukazatel na data určená k aktualizaci

void funkce nemá žádnou návratovou hodnotu

Úkolem samotné agregační funkce je pouze zpracovat předané hodnoty a výsledek uložit do již existujícího záznamu daného ukazatelem `dst`.

Funkce pro finální zpracování jsou implementovány jako součást výstupní šablony modulu. Aktuální stav využívá pouze výpočet konečné podoby průměrné hodnoty. Rozhraní vyžadované použitým ukazatelem je následující.

record ukazatel na proměnnou, která slouží jako zdroj i cíl výsledné hodnoty

count představuje počet agregovaných zpráv potřebný k výpočtu průměru

void funkce nemá žádnou návratovou hodnotu

5.2.1 Šablony

Unirec definuje seznam podporovaných datových typů (Tabulka 3.1), které je třeba zahrnout i v implementaci konkrétních agregačních funkcí. Implementace agregační funkce pro každý datový typ by byla časově i prostorově náročná. Pro jednoduchost je tedy většina implementována pomocí šablon. Definice a konkrétní podoby implementace se vyskytují v souborech *agg_functions.cpp* a *agg_functions.h*. Ukázka implementace agregační funkce `min` pomocí šablony je obsažena ve Výpisu zdrojového kódu 5.6.

Výpis 5.6: Příklad šablony agregační funkce pro minimum.

```
template <typename T>
void min(const void *src, void *dst)
{
   if (*((T*)src) < *((T*)dst))
      *((T*)dst) = *((T*)src);
}
```

Ne všechny datové typy (například struktura `ip_addr_t`) mají definované potřebné operace skrze operátory. V takovém případě je třeba manipulaci s daty implementovat přímo v dané funkci. Tímto se odliší od dané šablony. Tato skutečnost je důsledkem importu výchozí implementace UniRec datových typů v hlavičkovém souboru *unirec.h*, který je postaven na strukturách jazyka C. V implementaci těchto struktur nejsou vlastnosti definované skrze operátory jako v C++ třídách vytvořených pro agregační modul. Součástí Výpisu 5.7 je ukázka implementace agregační funkce `min` pro datový typ `ip_addr_t`.

Výpis 5.7: Příklad specializované agregační funkce pro minimum.

```
void min_ip(const void *src, void *dst)
{
   int ret = ip_cmp((const ip_addr_t*)src, (const ip_addr_t*)dst);

   if (ret < 0)
      *((ip_addr_t*)dst) = *((ip_addr_t*)src);
}
```

Stejným způsobem jsou realizované i ostatní funkce vyjmenované v požadavcích pro modul. Jejich deklaraci a implementaci lze najít na githubu [17] ve výše jmenovaných souborech.

5.3 Datové typy

5.3.1 Config

Po spuštění modulu se jako první vytvoří datová třída `Config`, jejíž data jsou inicializována z uživatelského vstupu. Tato třída reprezentuje konfiguraci pro běh modulu a zajišťuje převzetí nové konfigurace od uživatele. Její zdrojový kód se nachází v souborech *configuration.cpp* a *configuration.h*.

Výpis 5.8: Deklarace proměnných konfigurační třídy modulu.

```
class Config {
private:
   int functions[MAX_KEY_FIELDS];
   char *field_names[MAX_KEY_FIELDS];
   int used_fields;
   int timeout[TIMEOUT_TYPES_COUNT];
   int timeout_type;
   bool variable_flag;
};
```

Deklarace třídy ve Výpisu zdrojového kódu 5.8 není kompletní a neobsahuje metody definované u této třídy. Pole `functions` reprezentuje přiřazenou funkci políčku registrovaném pro daný index, stejným způsobem pole `field_names` přiřazuje danému indexu název registrovaného UniRec políčka. Hodnota proměnné `used_fields` ukazuje počet již definovaných políček. Následující text obsahuje důležité metody třídy `Config` pro funkce agregačního modulu a ukázky částí jejich implementace.

5.3.1.1 add_member()

Provádí přidání nového uživatelem definovaného políčka do konfigurace modulu. Dostupné parametry z uživatelského vstupu jsou agregační funkce (`func`) a název UniRec políčka (`field_name`). Výpis 5.9 ukazuje hlavní operace prováděné během volání této metody.

Výpis 5.9: Konfigurace nového políčka.

```
strncpy(field_names[used_fields], field_name, name_length + 1);
functions[used_fields] = func;
used_fields++;
```

5.3.1.2 get_function_ptr()

Jedná se o jednu z klíčových funkcí agregačního modulu podporující univerzálnost přiřazení agregačních funkcí. Agregační funkce musí splňovat unifikované rozhraní, které je využíváno v přiřazení ukazatele na tuto funkci do pole ukazatelů agregačních funkcí. Tato funkce zajišťuje přiřazení správné implementace agregační funkce k datovému typu daného políčka. Využívá k tomu datový typ políčka předaný parametrem, hodnota parametru `index` v tomto případě vyjadřuje agregační funkci přiřazenou právě zpracovávanému políčku. Funkce vrací ukazatel ve formě definovaného datového typu `agg_func`. Deklarace funkce vypadá následovně:

```
agg_func get_function_ptr(int index, ur_field_type_t field_type)
```

K výběru správné implementace slouží více úrovňový switch, který vybere správnou agregační funkci dle předaného parametru `index` a na základě UniRec datového typu přiřadí správnou implementaci šablony. Část s výběrem správné implementace a přiřazení jejího ukazatele do proměnné návratové hodnoty je zobrazena ve Výpisu zdrojového kódu 5.10.

Výpis 5.10: Přiřazení konkrétní implementace funkce.

```
agg_func out = &nope;
switch (functions[index]) {
   case MIN:
      switch (field_type) {
         case UR_TYPE_INT32:
            out = &min<int32_t>;
            break;
         case UR_TYPE_IP:
            out = &min_ip;
           break;
           /* .... */
      }
}
```

Ukázka obsahuje vybranou část kódu s agregační funkcí minima, zmíněné v Sekci agregačních funkcí (5.2), pro lepší představu fungování výběru ukazatele na konkrétní implementaci a jeho přiřazení.

5.3.1.3 return_template_def()

Metoda využívající přístupu k proměnným konfigurace, ze kterých je schopna vytvořit řetězec definující novou UniRec šablonu pro výstupní rozhraní. Výsledná šablona obsahuje všechna políčka zadané uživatelem a pole používaná modulem (`time_first`, `time_last`, `count`). Generování výsledného řetězce touto metodou je znázorněno ve Výpisu zdrojového kódu 5.11.

Výpis 5.11: Tvorba definice výstupní šablony.

```
char *tmplt_def = new char [len];
for (int i = 0; i < used_fields; i++) {
   strcat(tmplt_def, field_names[i]);
   strcat(tmplt_def, ",");
}
strcat(tmplt_def, static_fields);
```

5.3.2 KeyTemplate

Datová třída obsahující data potřebná k tvorbě třídy agregačního klíče. Pole `indexes_to_record` obsahuje index daného políčka v přijaté UniRec zprávě pro přímé získání dat, které tvoří agregační klíč. Hodnota `key_size` udává velikost klíče, jelikož může být složen pouze z políček s pevnou délkou. Metoda `reset()` slouží k nastavení šablony na původní prázdnou hodnotu, jelikož modul umožňuje změnu šablony po zahájení nové komunikace na vstupním LibTrap rozhraní. Jak již ukázka ve Výpisu zdrojového kódu 5.12 napovídá, šablona je využívána jako statická třída za účelem dostupnosti z libovolné části modulu.

Výpis 5.12: Třída KeyTemplate.

```
class KeyTemplate {
public:
   static int indexes_to_record [MAX_KEY_FIELDS];
   static uint used_fields;
   static uint key_size;
   static void add_field(int record_id, int size);
   static void reset();
};
```

5.3.3 Key

Třída reprezentující agregační klíč. Její tvorba je závislá na šabloně `KeyTemplate`. Instance této třídy je součástí uloženého záznamu a slouží jako klíč pro hledání prvku v mapě (množině unikátních záznamů). Výpis 5.13 obsahuje důležité proměnné a metody třídy.

Výpis 5.13: Třída Key.

```
class Key {
private:
   char* data;
   int data_length;
public:
void add_field(const void *src, int size);
friend bool operator< (const Key &a, const Key &b);
friend bool operator== (const Key &a, const Key &b);
```

Skládá se z pole bajtů, které v sobě ukrývá hodnoty klíčových políček z přijaté zprávy a počtu takto zapsaných bajtů. Třída obsahuje definici vlastních operátorů, jelikož implementace modulu vyžaduje použití těchto vlastností. Instance

třídy se musí být schopny navzájem porovnat kvůli kolizím hashovací funkce. Jedná se o porovnání sekvence bytů pomocí funkce `memcmp()`.

5.3.3.1 hash()

Z důvodu rychlejšího přístupu k uloženým datům je úložiště tvořeno *hash* mapou. Implementace *hash* mapy vyžaduje definici hashovací funkce pro daný datový typ. Pro toto použití byla vytvořena specializace funkce `std::hash()`, zobrazena ve Výpisu zdrojového kódu 5.14.

Výpis 5.14: Specializace std::hash().

```
namespace std {
   template<>
   struct hash<Key>
   {
      size_t operator()(const Key &k) const
      {
         return SuperFastHash(k.get_data(), k.get_size());
      }
   };
}
```

Tvorbu *hash* hodnoty provádí funkce `SuperFastHash()` importovaná z *NEMEA framework*u.

5.3.4 OutputTemplate

Tato datová třída poskytuje návod, jakým způsobem se má každé registrované políčko zpracovávat. Pole `indexes_to_record` obsahuje index daného políčka v přijaté UniRec zprávě pro přímé získání dat ke zpracování. Proměnná `process` eviduje přiřazené agregační funkce a `avg_fields` funkce pro finální zpracování. Hodnota `out_tmplt` reprezentuje vygenerovanou výstupní šablonu. Metoda `reset()` slouží k nastavení šablony na původní prázdnou hodnotu, jelikož modul umožňuje změnu šablony po navázání nové komunikace na vstupním LibTrap rozhraní. Výpis zdrojového kódu 5.15 ukazuje implementaci této třídy. Výstupní šablona je využívána jako statická třída za účelem dostupnosti z libovolné části modulu.

Výpis 5.15: Třída OutputTemplate.

```
class OutputTemplate {
public:
   static ur_template_t *out_tmplt;
   static int indexes_to_record[MAX_KEY_FIELDS];
   static int used_fields;
   static agg_func process[MAX_KEY_FIELDS];
   static bool prepare_to_send;
   static final_avg avg_fields[MAX_KEY_FIELDS];
   static void add_field(int record_id, agg_func foo, bool avg_flag,
                         final_avg foo2);
   static void reset();
```

5.3.5 Pomocné třídy

V modulu existují i pomocné třídy, které slouží k předání parametrů metodám nebo funkcím. Díky návrhu předávání většiny parametrů jako `void*` je možné parametrem předat ukazatel na libovolnou potřebnou datovou strukturu. Jako příklad této skutečnosti lze zmínit strukturu `var_params`, jejíž podoba je obsažena ve Výpisu 5.16. Struktura je použita v agregační funkci `last_variable(const void *src, void *dst)`.

Výpis 5.16: Pomocná struktura pro nestandardní parametry agregační funkce last_variable().

```
typedef struct {
   void *dst;
   int field_id;
   int var_len;
} var_params;
```

5.4 Zpracování dat

Zpracování dat a způsob manipulace s nimi jsou klíčové procesy celé agregace. V této sekci jsou popsány implementované postupy zpracování dat. S daty procházejícími modulem se provádí následující operace:

- Příjem dat.
- Uložení nových dat.
- Aktualizace již skladovaných záznamů.
- Odeslání dat.

5.4.1 Příjem dat

Modul využívá knihovny LibTrap, tedy příjem dat probíhá skrze rozhraní, která tato knihovna nabízí. Nad uživatelem definovaným rozhraním se provádí blokující operace čekající na data.

V případě aktivity rozhraní, se pomocí již existujícího makra provede základní řízení chyb během příjmu dat. Reakci lze definovat jako parametr makra. Modul zachytává návratový kód `TRAP_E_FORMAT_CHANGED`, který oznamuje příjem nové komunikační UniRec šablony. V tomto okamžiku se resetují veškeré konfigurační struktury a spustí procesy validace uživatelského vstupu vůči přijaté UniRec šabloně. Ukončením tvorby nových šablon agregačního modulu `KeyTemplate` a `OutputTemplate` je vše připraveno k běžnému provozu a zpracování příchozích záznamů. Pokud modul během validace detekuje neplatné nastavení, dojde k vypsání chybového hlášení a ukončení modulu, jelikož nemůže dále pracovat.

5.4.2 Zpracování záznamu

Po úspěšném přijetí záznamu dojde k vytvoření agregačního klíče, jehož instance je naplněna daty z přijaté zprávy. Pokusem o vložení klíče do úložiště (Výpis 5.17) s prozatím prázdným ukazatelem na možný budoucí záznam (`init_ptr`) se proces zpracování, dle návratové hodnoty v proměnné `inserted`, rozděluje na dvě různé metody zpracování přijatého záznamu. Prázdný ukazatel bez alokované struktury záznamu je využit z důvodu úspory času procesoru v případě, že nový záznam nebude třeba vytvářet (dojde k aktualizaci již skladovaného). Všechny ukázky kódu v této sekci jsou části implementace ze souboru *aggregator.cpp*.

Výpis 5.17: Pokus o vložení nového záznamu.

```
std::pair<std::unordered_map<Key, void*>::iterator, bool> inserted;
inserted = storage.insert(std::make_pair(rec_key, init_ptr));
```

- **Nový záznam** je metoda zpracování navazující na úspěšné vložení klíče do úložiště. Na přiděleném ukazateli se poté alokuje nová datová struktura UniRec záznamu s potřebnou velikostí, do které jsou z přijaté zprávy překopírována všechna potřebná políčka. Tvorba nového záznamu je znázorněna ve Výpisu 5.18.

  Výpis 5.18: Tvorba nového záznamu.

  ```
  int var_length = config.is_variable() == false ? 0 : 2048;
  void *out_rec = create_record(OutputTemplate::out_tmplt,
                                var_length);
  init_record_data(in_tmplt, in_rec, OutputTemplate::out_tmplt,
                   out_rec);
  inserted.first->second = out_rec;
  ```

- **Existující záznam** označuje proces aktualizace již skladovaného záznamu provedením agregačních funkcí s nově přijatými daty. Celý proces aktualizace provádí funkce `process_agg_functions()`. V následujícím Výpisu zdrojového kódu 5.19 je zobrazeno využití implementace agregačních funkcí pomocí ukazatelů na funkce zmíněné v Sekci 5.2.

Výpis 5.19: Zpracování agregačních funkcí.

```
for (int i = 0; i < OutputTemplate::used_fields; i++) {
   if (ur_is_fixlen(i)) {
      ptr_dst = ur_get_ptr_by_id(out_tmplt, dst_rec,
                                 OutputTemplate::indexes_to_record[i]);
      ptr_src = ur_get_ptr_by_id(in_tmplt, src_rec,
                                 OutputTemplate::indexes_to_record[i]);
      OutputTemplate::process[i](ptr_src, ptr_dst);
   }
   else {
      var_params params = {dst_rec, i,
                           ur_get_var_len(in_tmplt, src_rec,
                             utputTemplate::indexes_to_record[i])};
      ptr_src = ur_get_ptr_by_id(in_tmplt, src_rec,
                                 OutputTemplate::indexes_to_record[i]);
      OutputTemplate::process[i](ptr_src, (void*)&params);
   }
}
```

5.4.3 Odeslání dat

Důležitou fází odeslání dat je možnost finálního zpracování dat před samotným odesláním, k němuž dojde na základě hodnoty proměnné označující požadavek na provedení tohoto procesu. Tento typ zpracování zajišťuje funkce `prepare_to_send()`. Následuje samotné odeslání expirovaných záznamů skrze výstupní LibTrap rozhraní definované ve funkci `send_record_out()`. Implementace modulu je zaměřená na vyšší úroveň spolehlivosti. Vyznačuje se snahou přijít o co nejmenší počet dat během zpracování. Za tímto účelem je manuálně nastaven timeout operace odeslání záznamu na výstupním rozhraní skrze volání `trap_ifcctl()`, po jehož vypršení modul okamžitě danou zprávu nezahodí, ale opakuje proces odeslání, a to až po hodnotu `MAX_TIMEOUT_RETRY` (v aktuální podobě 3x). Pokud se ani poté nepodaří zprávu úspěšně odeslat, teprve pak je zahozena a pokračuje se dalším expirovaným záznamem. V případě zahozené odchozí zprávy je vypsáno upozornění na standardní chybový výstup. Zajištění opakovaného odeslání po selhání je znázorněno ve Výpisu 5.20.

Výpis 5.20: Proces opakovaného odesílání.

```
for (; i < MAX_TIMEOUT_RETRY; i++) {
   int ret = trap_send(0, out_rec, ur_rec_fixlen_size(out_tmplt) +
                       ur_rec_varlen_size(out_tmplt, out_rec));

   TRAP_DEFAULT_SEND_ERROR_HANDLING(ret, continue, break);
   return true;
}
```

5.5 Ukončení modulu

Většina z existujících modulů reaguje na ukončení datového proudu jako na ohlášení splněné činnosti a ukončí se. V tomto ohledu je agregační modul výjimka, jelikož jediné správné ukončení modulu nastane až v případě, že tomu tak chce sám uživatel. Modul stejně jako ostatní odchytává data indikující konec komunikace, ale díky implementované možnosti restartu se neukončuje a stále čeká na vstupním rozhraní na navázání další komunikace, na kterou bude uplatňovat stejné uživatelské nastavení jako na předchozí ukončenou.

5.5.1 Restart

Možnost restartu znamená, že modul není třeba ukončit, aby se mohla navázat nová komunikace na vstupním rozhraní (jiný zdroj dat, jiná šablona). Jelikož modul registruje změnu na vstupu zachytáváním návratové hodnoty a reaguje na ni novým nastavením vlastních šablon z přijatých dat. Stačí tedy, aby došlo k přerušení současného datového proudu a byla zahájena nová komunikace (například ohlášena změna UniRec šablony). Tato možnost je implementována především na základě myšlenky, že chci spustit modul s požadovanými parametry, ale zpracování chci stále stejné, i když data přijdou z různých zdrojů.

5.5.2 Ukončení

Na počátku této sekce bylo řečeno, že modul lze ukončit pouze na pokyn uživatele. Modul zachytává signály pomocí vlastního *signal handleru* a na základě daného signálu nastavuje proměnné, dle kterých se modul orientuje. Bylo třeba zrealizovat vlastní implementaci tohoto odchytávače signálu, jelikož verze poskytovaná v podobě makra knihovnou LibTrap v tomto případě nelze použít. Základní poskytované makro využívá po zachycení signálu možnost ukončit veškerou činnost všech LibTrap rozhraní voláním `trap_terminate()`. Modul má v držení uložené záznamy, které by bylo vhodné před řádným ukončením ještě vyexportovat. Dalším důvodem je vícevláknová implementace, jelikož není předem známo jaké vlákno signál zachytí a spustí proces zpracování odchytávaného signálu. V tomto případě může, v kombinaci se zámky kritických sekcí, docházet k uváznutí (*deadlock*).

Vlastní signal handler pouze změní hodnotu řídící proměnné stop na hodnotu označující konec běhu modulu (`stop` = 1). V tomto okamžiku lze ovšem předpokládat blokující volání čekající na data ze vstupního rozhraní. Bylo tedy třeba ještě dodefinovat hodnotu timeoutu pro toto volání, aby po stanovené době došlo k otestování zda má modul stále ještě běžet. Řídící proměnnou sdílí všechna vlákna využívána modulem, žádné vlákno není nuceně ukončeno skrze `pthread_cancel()`, a hlavní vlákno tedy čeká na ukončení všech spuštěných vláken. Tento způsob postupného a řádného ukončení se může projevit delší

dobou čekání na ukončení ostatních vláken. Především pokud se čeká na ukončení vlákna, které je po dobu pasivního či globálního typu exportu uspáno po dobu intervalu t_n sekund. Na hlavním vlákně je, aby navázalo odesláním všech skladovaných záznamů, uvolnilo veškeré alokované paměťové struktury a následně ukončilo celý modul.
Výpis 5.21 obsahuje fragmenty zdrojových kódů procesů zmíněných v této sekci.

Výpis 5.21: Realizace zmíněných postupů.

```
/* Register own signal handler */
signal(SIGTERM, my_signal_handler);
signal(SIGINT, my_signal_handler);

/* Set timeout for trap interface */
trap_ifcctl(TRAPIFC_INPUT, 0, TRAPCTL_SETTIMEOUT, TRAP_RECV_TIMEOUT);

/* Check control variable for run */
while (!stop) {
/* ... */
}
/* Wait for other running thread*/
pthread_join(timeout_thread, NULL);

/* Send out all stored records */
flush_storage();

/* Send end of stream message */
trap_send(0, "", 1);

/* Close all trap interfaces */
trap_terminate();
```

5.5.2.1 Návratové hodnoty

Modul při svém ukončení vrací následující hodnoty.

-1 Nepodařilo se alokovat místo pro potřebnou datovou strukturu.

0 Řádné ukončení signalizované uživatelem.

1 Zadané UniRec políčko není obsaženo v přijaté šabloně.

Kapitola 6

Testování

6.1 Testovací prostředí

Modul byl spouštěn a testován ve dvou různých prostředích. Prvním je lokální stanice pro testování propustnosti a druhým testovací kolektor (`collector-nemea-test.liberouter.org`).

6.1.1 Lokální stanice

Metodika testování na osobním počítači byla vybrána z důvodu dosažení ekvivalentních podmínek pro všechny testy.

- Stejné provozní zatížení pro všechny testy.
- Stejná vstupní data pro všechny testy skrze uložené flow záznamy.
- Bez použití sítě (pouze UNIX sokety a lokální soubor)

Jedná se o osobní přenosný počítač *ASUS 7265NGW* s následujícími parametry.

- **CPU**: Intel® Core i7-6500U
- **RAM**: SAMSUNG 2x4GB DDR3, 1600MT/s
- **SSD**: Kingston 512GB SSD, SATA 3.1 (6Gb/s)
- **OS**: Fedora 27, 64-bit

6.1.2 Testovací data

Testovací data reprezentují 640 sekundový záznam IP flow z kolekoru v UniRec formátu. Jsou uložena v šesti souborech o velikostech 501 MB (3006 MB). Dohromady testovací data obsahují 46 462 185 flow záznamů.

6.1.3 Kolektor

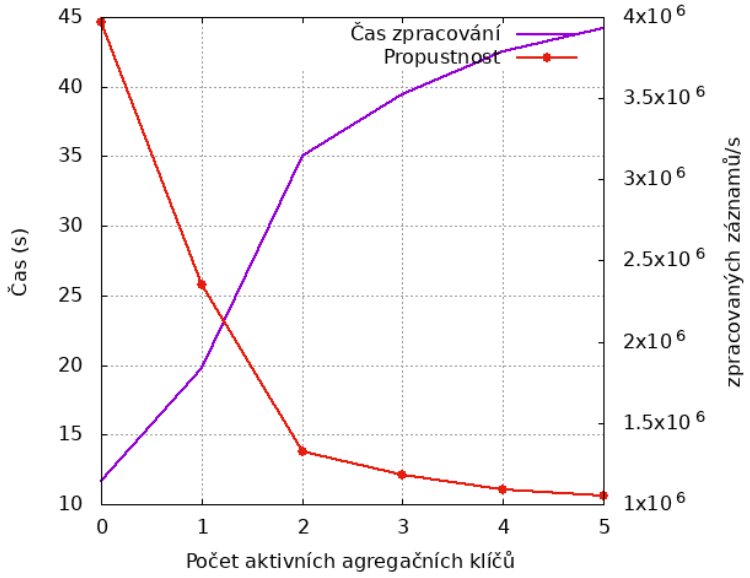
Kolektor je zdroj dat z reálného provozu. Reálný provoz nenabízí vhodné podmínky pro srovnávací testy, jelikož generovaná hustota provozu není stálá a výsledky by neměly požadovanou informační hodnotu. Kolektor je testem nasazení modulu v prostředí využívaného ostatními NEMEA systémy (moduly). Je také správnou volbou pro otestování funkcí modulu s nekonečným proudem dat, který přichází v nerovnoměrných intervalech a dávkách.

6.2 Výsledky testování propustnosti

6.2.1 Délka agregačního klíče

Graf na Obrázku 6.1 ukazuje vliv počtu UniRec políček (`URFIELD`) agregačního klíče na celkovou propustnost modulu. Políčka určené k roli agregačního klíče jsou definována příkazem `-k URFIELD`. Sekvence postupně přidávaných políček je `SRC_IP`, `DST_IP`, `SRC_PORT`, `DST_PORT`, `PROTOCOL`. Přiřazené agregační funkce reprezentují součet bajtů a paketů.

```
/usr/bin/nemea/agg -i u:input,u:aggr -t 120 -s BYTES -s PACKETS
                   [-k URFIELD ...]
```



Obrázek 6.1: Vliv počtu agregačních klíčů na čas zpracování a propustnost agregačního modulu.

Z výsledků výše je patrné, že způsob tvorby agregačního klíče (kopírováním dat pro každý záznam) se s rostoucím počtem registrovaných políček značně projeví i na propustnosti modulu.

6.2.2 Agregační funkce

Testování agregačních funkcí je rozděleno do dvou částí. První část ukazuje dobu zpracování vstupních dat v závislosti na konkrétní jedné instanci agregační funkce (Tabulka 6.1). Druhá část zobrazuje propustnost při kombinaci agregačních funkcí (Tabulka 6.2). Uvedené hodnoty jsou průměrem pěti nezávislých spuštění modulu.

Tabulka 6.1: Doba zpracování a propustnost pro jednotlivé agregační funkce.

Agregační funkce	Čas zpracování (s)	Propustnost/s
Sum	34,9	1,333 mil.
Avg	35,7	1,300 mil.
Min	35,1	1,322 mil.
Max	35,3	1,315 mil.
First	34,8	1,337 mil.
Last	34,9	1,333 mil.
And	35,0	1,328 mil.
Or	34,9	1,330 mil.

Tabulka 6.2: Doba zpracování pro kombinace agregačních funkcí. Funkce přiřazovány v pořadí dle Tabulky 6.1.

#použitých funkcí	Čas zpracování (s)	Propustnost/s
1	35,0	1,328 mil.
2	36,0	1,290 mil.
3	36,1	1,289 mil.
4	36,4	1,275 mil.
5	36,7	1,266 mil.
6	36,9	1,259 mil.
7	37,1	1,254 mil.
8	37,1	1,253 mil.

Z výsledků je patrné, že samotné zpracování záznamů modulem je dostatečně rychlé pro nasazení v reálném prostředí sítě CESNET2, kde provoz při shromažďování testovacích dat dosahoval počtu 75 597 záznamů/s. V současné době se maximální datový tok této sítě pohybuje na úrovni cca 300 000 záznamů za sekundu.

6.3 Zpracování dle typu exportu

V této sekci je zobrazen vliv jednotlivých typů exportu a délky časového okna na propustnost agregačního modulu. Ve výsledcích není zahrnut *Passive* a

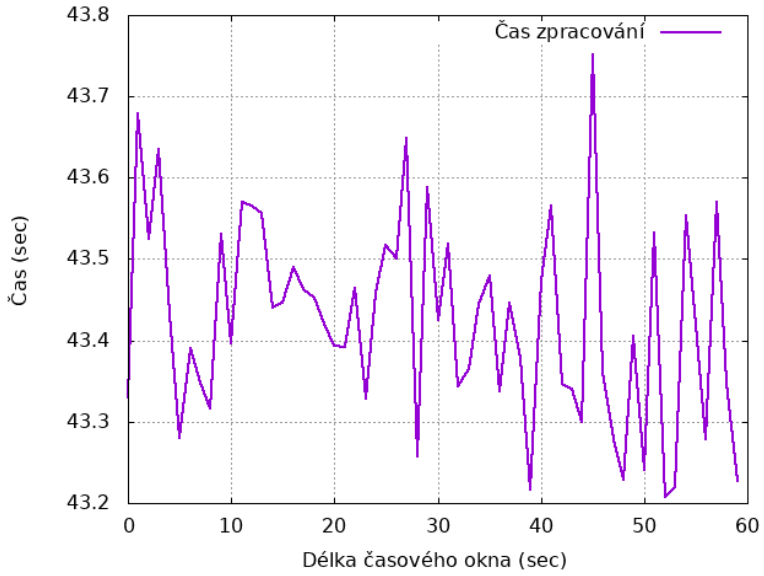
Mix timeout, jelikož jejich výsledky se během dávkového zpracování dat ze souboru neprojeví. *Passive* se vůbec neuplatní, jelikož se orientuje dle času prvního přijatého záznamu a poté je ovlivňován systémovým časem (Sekce 5.1). *Mix* timeout se dle vlastností pasivního projeví jako *Active*, který je ve výstupech zahrnut.

Testovací provoz uplatňuje agregaci běžného flow dle následujících pravidel.

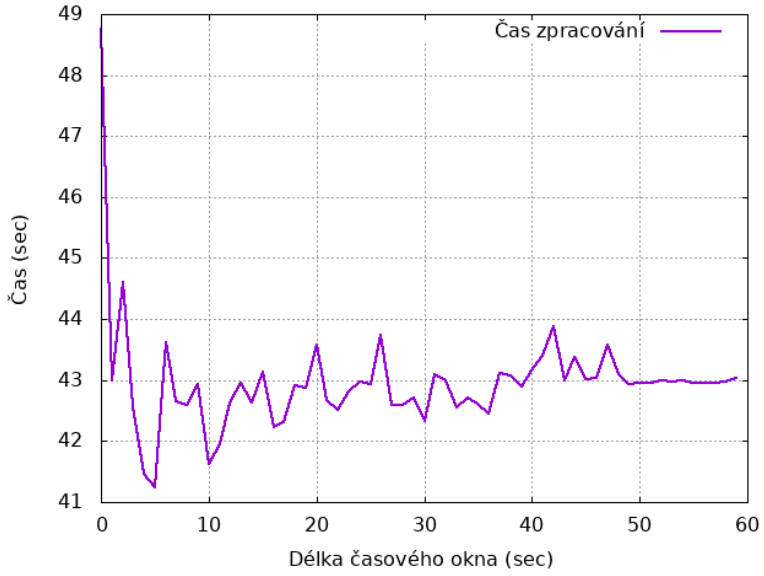
- **Agregační klíč**: `SRC_IP`, `DST_IP`, `SRC_PORT`, `DST_PORT`, `PROTOCOL`
- **Suma**: `BYTES`, `PACKETS`
- **Or**: `TCP_FLAGS`

Jedná se tedy již o měření spuštění modulu, které může být běžně využíváno ke snížení objemu provozu.

Naměřené hodnoty ukazují, že exportovací subsystém nemá na běh modulu větší vliv než přidání dvou až tří nových políček jako součást agregačního klíče. Z grafů na Obrázku 6.2 lze vysledovat trend vlivu exportu záznamů na čas zpracování testovacích dat dle časových oken v rozmezí 1-60 sekund. Testování aktivního typu exportu nepřináší žádné viditelné známky vlivu délky časového okna na dobu zpracování, jelikož tento případ je ovlivněn každým příchozím záznamem stejného agregačního klíče a časovými značkami. Viditelné výchylky grafu jsou způsobeny nedostatkem počtu opakování měření. Na grafu s globálním typem timeoutu je krásně vidět konec vlivu exportu v momentě kdy je odeslání záznamů vyvoláno až po zpracování všech záznamů na vstupním rozhraní. Konečná podoba grafu a délka zpracování je také dána strukturou zpracovávaných dat, jelikož míra jejich agregace dle stanoveného klíče také ovlivňuje výkon modulu.



(a) Aktivní timeout.



(b) Globální timeout.

Obrázek 6.2: Vliv délky časového okna na dobu zpracování.

Závěr

Cílem této práce byl návrh a implementace univerzálního agregačního modulu do modulárního systému NEMEA, využívajícího datový formát UniRec. Na základě získaných informací o prostředí systému a samotném datovém formátu byl navržen koncept univerzálního agregačního modulu pro zprávy ve formátu UniRec k použití nad daty z vysokorychlostních sítí. Tento modul byl dle vytvořených scénářů implementován a řádně otestován na testovacích vstupních datech i v reálném provozu české univerzitní a výzkumné sítě CESNET2.

Agregační modul je spolu s filtračním modulem základním kamenem pro prototypování streamwise detekčních modulů. Na uživatelské úrovni se dá snadno spustit experiment k získání prvotní představy o efektivnosti případné implementace optimálního řešení detektoru. V prostředí systému NEMEA tento agregační modul do této doby nebyl k dispozici. Jelikož se z pohledu datového formátu UniRec jedná o univerzální modul, může být nasazen kdekoli je potřeba agregace datového provozu. Výsledek této práce je přínosem pro rozvoj detekčních modulů i pro snížení komunikačního zatížení infrastrukturních linek mezi jednotlivými systémovými moduly. Umožní také snížit požadované výkonnostní nároky hardwaru pro provoz jednotlivých modulů redukcí počtu vstupních dat určených ke zpracování.

Podařilo se navrhnout a vytvořit univerzální agregační modul s mnoha možnostmi použití v rámci prostředí systému NEMEA. Samotný modul byl z důvodu potřeby rychlosti implementován v jazyce C/C++. Modul byl navržen, aby umožňoval snadné rozšíření o další agregační funkce bez nutnosti zásahu do implementace samotného modulu. V současnosti je pro nasazení v reálném prostředí požadovaná propustnost alespoň 250 000 záznamů/s. Testování výsledné aplikace ukázalo, že modul je schopen zpracovat 10^6 záznamů/s, a splňuje tak aktuální požadavky na zpracování dat v reálném čase. Modul se stal součástí oficiální distribuce balíku NEMEA Modules a je zahrnut v instalaci NEMEA systému (Příloha B). Příklady použití vytvořeného agregačního modulu i s ukázkami konfigurace pro spuštění jsou uvedeny v Příloze C.

Možné budoucí práce

Aktuální stav modulu splňuje požadavky, ale během testování byly identifikovány vlastnosti, které by se daly v budoucnu vylepšit nebo přidat.

- Kontrola přetečení datového typu v aritmetických agregačních funkcích.
- Nové agregační funkce:
  - zřetězení datového typu `UR_TYPE_STRING`
  - standardní odchylka (stdev)
  - počet unikátních záznamů (`COUNT_UNIQUE`)
- Kontrola maximální velikosti výstupního záznamu (65534 B).
- Aplikace více agregačních funkcí na stejné políčko.
- Možnost vytváření bi-flow záznamů.
- Samostatné výpočetní vlákno pro odesílání záznamů.

Literatura

[1] R. Blažek. (2016, Říjen). „Introduction to Statistical Methodology for Anomaly Detection“. *Síťová bezpečnost*. [online]. Dostupné z: `https://edux.fit.cvut.cz/courses/MI-SIB.16/_media/lectures/mi-sib-p07-statisticaldetectionmethodology.pdf` [24.03.2018].

[2] ISO/IEC JTC 1. *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. Switzerland: ISO/IEC Copyright office, 1994.

[3] *Traffic Flow Measurement: Architecture*. N. Brownlee, C. Mills, G. Ruth. October 1999. (Obsoletes RFC2063) (Status: INFORMATIONAL) (DOI: 10.17487/RFC2722).

[4] *IPv6 Flow Label Specification*. J. Rajahalme, A. Conta, B. Carpenter, S. Deering. March 2004. (Obsoleted by RFC6437) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC3697).

[5] *Requirements for IP Flow Information Export (IPFIX)*. J. Quittek, T. Zseby, B. Claise, S. Zander. October 2004. (Status: INFORMATIONAL) (DOI: 10.17487/RFC3917)

[6] *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. B. Claise, Ed.. January 2008. (Obsoleted by RFC7011) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC5101)

[7] R. Hofstede et al. *Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX*. Fourthquarter: IEEE Communications Surveys & Tutorials, 2014.

[8] Cisco Systems, Inc. „Configuring NetFlow and NetFlow Data Export“. [online]. Dostupné z: `https://www.cisco.com/c/en/us/td/docs/`

ios-xml/ios/iproute_pi/configuration/15-s/nf-15-s-book/cfg-nflow-data-expt.pdf [26.03.2018].

[9] Cisco Systems, Inc. „Introduction to Cisco IOS NetFlow“. [online]. Dostupné z: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html [26.03.2018].

[10] T. Čejka, V. Bartoš, M. Švepeš, Z. Rosa, H. Kubátová. „NEMEA: A framework for network traffic analysis“. 2016 12th International Conference on Network and Service Management (CNSM), Montreal, QC, 2016. (DOI: 10.1109/CNSM.2016.7818417)

[11] CESNET, z.s.p.o. „UniRec: Overview“. [online]. Dostupné z: http://nemea.liberouter.org/doc/unirec/md_doc_intro.html [27.03.2018].

[12] CESNET, z.s.p.o. „UniRec: API“. [online]. Dostupné z: http://nemea.liberouter.org/doc/unirec/group__urtemplate.html [27.03.2018].

[13] CESNET, z.s.p.o. „TRAP Interface Specifier“. *NEMEA - System for network traffic analysis and anomaly detection.* [online]. Dostupné z: http://nemea.liberouter.org/trap-ifcspec/ [29.03.2018].

[14] CESNET, z.s.p.o. „Libtrap: Overview“. [online]. Dostupné z: http://nemea.liberouter.org/doc/libtrap/ [29.03.2018].

[15] CESNET, z.s.p.o. „General Information“. *NEMEA - System for network traffic analysis and anomaly detection.* [online]. Dostupné z: http://nemea.liberouter.org/ [29.03.2018].

[16] T. Čejka (2017). „Network Monitoring and Anomaly Detection“. *Síťová bezpečnost.* [online]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-SIB.16/_media/tutorials/cejkat-network-monitoring-detection.pdf [30.03.2018].

[17] M. Slabihoudek (2018). „Aggregator module“. *Github: NEMEA modules.* [online]. Dostupné z: https://github.com/CESNET/Nemea-Modules/tree/master/aggregator [13.4.2018].

[18] M. Slabihoudek (2018). „Aggregation module“. *NEMEA, System for network traffic analysis and anomaly detection.* [online]. Dostupné z: http://nemea.liberouter.org/aggregation/ [23.4.2018].

[19] Kalina, Miroslav. „Sledování provozu 100Gb/s síťových infrastruktur“. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Příloha A

Seznam použitých zkratek

ADD Average detection delay

API Application programming interface

AS Autonomous system

ASCII American Standard Code for Information Interchange

BGP Border gateway protocol

CPD Change point detection

DDoS Distributed denial of service

DoS Denial of service

DPI Deep packet inspection

FAR False alarm rate

HTTP Hypertext transfer protocol

IDS Intrusion detection system

IETF Internet engineering task force

IP Internet protocol

IPFIX Internet Protocol Flow Information Export

IPS Intrusion prevention system

NEMEA Network Measurements Analysis

NREN National Research and Education Network

OS Operating system

PEM Privacy-enhanced electronic mail

PFA Probability of false alarm

POSIX Portable Operating System Interface

PWR Detection power

SCTP Stream control transmission protocol

STaaS Security tools as a service

TCP Transmission communication protocol

TLS Transport layer security protocol

UDP User datagram protocol

UniRec Unified record

Příloha B

Instalační a uživatelská příručka

B.1 Instalace

Modul je závislý na prostředí NEMEA frameworku. Ke kompilaci modulu jsou vyžadovány následující balíčky.

- autoconf
- automake
- gcc
- gcc-c++
- libtool
- libxml2-devel
- libxml2-utils (contains xmllint on Debian)
- make
- pkg-config

Modul lze spolu s prostředím instalovat dvěma způsoby, v binární podobě z repozitáře `https://copr.fedorainfracloud.org/coprs/g/CESNET/NEMEA/`. Po instalaci repozitáře lze instalovat jako běžný balíček s aplikací.

```
yum install nemea
```

Nebo zkompilovat přímo ze zdrojového kódu dostupného na Githubu. Před samotnou kompilací musí být již nainstalovány veškeré požadované závislosti.

```
git clone --recursive https://github.com/CESNET/nemea
./bootstrap.sh
./configure --enable-repobuild --prefix=/usr
 --bindir=/usr/bin/nemea --sysconfdir=/etc/nemea
 --libdir=/usr/lib64
make -C aggregator
```

Po instalaci lze modul nalézt v `/usr/bin/nemea` pod názvem `agg` nebo v aktuálním adresáři. Výsledná pozice záleží na místě určení a způsobu instalace. Více informací a detailněji popsaný postup instalace naleznete na `https://github.com/CESNET/Nemea`.

B.2 Použití

Obecné možnosti spuštění a nastavení jsou popsány v samotné nápovědě modulu

```
/usr/bin/nemea/agg -h
```

nebo na webových stránkách `http://nemea.liberouter.org/aggregation/`, které slouží zároveň jako manuálové stránky s více tipy a nápady na použití agregačního modulu.

B.3 Možnosti modulu

Modul nabízí následující možnosti spuštění (výstup nápovědy modulu zobrazující definici a popis jednotlivých parametrů).

```
Module specific parameters
-k --key <URFIELD>
   Defines received UniRec field name as part of aggregation
   key.Use individually on each field as -k FIELD_NAME.
   When no key specified every record is considered to match
   the empty key (every record is processed as with
   the equal key).
-t --time_window <string>
   Represents type of timeout and #seconds for given type
   before sending record to output.
   Use as [G,A,P]:#seconds or M:#Active,#Passive
   (eg. -t "m:10,25").When not specified the default value
   (A:10) is used.
-s --sum <URFIELD> Makes sum of UniRec field values
   identified by given name.
-a --avg <URFIELD> Makes average of UniRec field values
```

```
   identified by given name.
-m --min <URFIELD> Keep minimal value of UniRec field
   identified by given name.
-M --max <URFIELD> Keep maximal value of UniRec field
   identified by given name.
-f --first <URFIELD> Keep first value of UniRec field
   identified by given name.
-l --last <URFIELD> Keep first value of UniRec field
   identified by given name.
-o --or <URFIELD> Make bitwise OR of UniRec field
   identified by given name.
-n --and <URFIELD> Make bitwise AND of UniRec field
   identified by given name.

Common TRAP parameters
-h [trap,1] Print help message for this module
   for libtrap specific parameters.
-i IFC_SPEC Specification of interface types and their
   parameters.
-v Be verbose.
-vv Be more verbose.
-vvv Be even more verbose.
```

Příloha C

Příklady použití

C.1 Tvorba grafu

Když chceme vytvářet grafy, které shrnují celkový provoz, chceme celkový součet jednotlivých sledovaných atributů sítě. Například nás zajímají hodnoty přenesených bajtů a paketů. Agregovat můžeme buď vše do jednoho kompletního výsledku, nebo například odlišit jednotlivé sondy sloužící jako zdroje dat.

- **Celkový provoz** - V aktuální podobě nechceme definovat žádný agregační klíč, jelikož chceme agregovat všechny záznamy do jednoho. Výsledné spuštění modulu je následující.

  ```
  ./agg -i u:input,u:output -s BYTES -s PACKETS
  ```

- **Jednotlivé zdroje** - Zdroj je rozlišený hodnotou `LINK_BIT_FIELD`. Tento název UniRec políčka se tedy stane naším agregačním klíčem. Výsledné spuštění modulu je následující.

  ```
  ./agg -i u:input,u:output -k LINK_BIT_FIELD -s BYTES
        -s PACKETS
  ```

C.2 Agregace výstupu NEMEA modulu

C.2.1 Agregace flow

Lze využít například chceme-li delší časové úseky datových toků, ale sondy nebo kolektory je agregují s menším časovým intervalem než požadujeme. Klíčem v této podobě jsou všechna UniRec políčka potřebná k jednoznačné identifikaci flow záznamu. V tomto případě se agregačním klíčem stala políčka `SRC_IP`, `DST_IP`, `SRC_PORT`, `DST_PORT`, `PROTOCOL`. Naším cílem je získat součet bajtů, paketů a všechny TCP příznaky, které se během stanoveného okna v síťovém toku objevily. Spuštění modulu je následovné.

```
./agg -i u:input,u:output -k SRC_IP -k DST_IP -k SRC_PORT
-k DST_PORT -k PROTOCOL -s BYTES -s PACKETS -o TCP_FLAGS
```

C.3 Jednoduchá detekce

Agregační modul v kombinaci s filtračním modulem tvoří jednoduchý detekční mechanismus. Agregační modul v daném časovém okně utvoří požadovaný výstup, na který bude reagovat připravený filtrační mechanismus a generovat případné bezpečnostní hlášení.

C.3.1 DDoS

Distribuované DoS útoky se vyznačují velikým provozem z různých zdrojů na jednu cílovou adresu a port. Chceme-li sledovat distribuované DoS útoky, lze spustit agregační modul s klíčem v podobě `DST_IP`, `DST_PORT`. Výstup bude sledovat filtrační modul, který reaguje dle nastavené prahové hodnoty počtu agregovaných záznamů stejného typu.

```
./agg -i u:input,u:output -k DST_IP -k DST_PORT
```

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `app` ..... spustitelná forma implementace pro Fedora 27 x64
- `text` ..... Texty práce
  - `DP_assignment.pdf` ..... zadání práce ve formátu PDF
  - `DP_Slabihoudek_Michal_2018.pdf` ..... text práce ve formátu PDF
- `src`
  - `app` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojové soubory práce ve formátu LaTeX