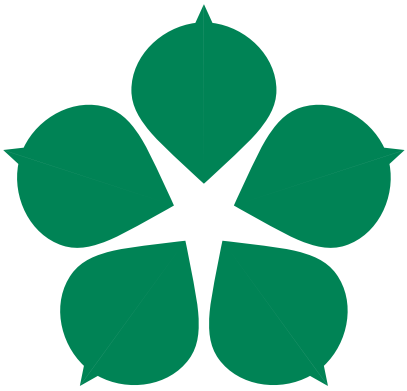
Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta



Rozšíření kolektoru Ipfixcol2 o modul pro export dat do Apache Kafka

Bakalářská práce

Ondřej Zeman

Vedoucí práce: Konopa Michal, Mgr.

České Budějovice 2021

Bibliografické údaje

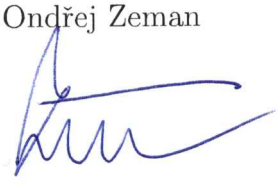
Zeman, O., 2021: Rozšíření kolektoru Ipfixcol2 o modul pro export dat do Apache Kafka. [The Extension of the IPFIXcol2 Collector of the Module for Data Export to Apache Kafka . Bc.. Thesis, in Czech.] – 43 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Abstract

The main aim of this bachelor thesis is to construct the output plugin for collector IPFIXcol2 made by the CESNET company and to describe this construction. The output plugin can process data in the IPFIX format and convert it into JSON format for further export to Apache Kafka. This plugin uses the multithread message procession to reach the maximal permeability. The thesis includes the comparison of this plugin with the plugin made by CESNET and the parallel solution PMACCT.

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 7. 12. 2021 Ondřej Zeman



Poděkování

Chtěl bych poděkovat svému vedoucímu Mgr. Michalovi Konopovi za vedení této práce a odborné rady během vývoje.
Dále bych rád poděkoval své rodině, která mě během studia vždy podporovala.

Contents

1 Úvod **4**

1.1 Cíle práce 5

2 Analýza **6**

2.1 Teoretická východiska práce 6

2.2 Sběr dat síťového provozu a distribuce záznamů 6

2.2.1 Technologie IPFIX a NetFlow 7

2.2.2 Exportér (sonda síťového provozu) 8

2.2.3 Kolektor 10

2.2.4 Message broker 11

2.3 Systémy pro sběr dat síťového provozu 13

2.3.1 IPFIXcol2 14

2.3.2 PMACCT 14

2.4 Systémy pro distribuci dat 15

2.4.1 Apache Kafka 15

2.4.2 RabbitMQ 16

2.5 Metodika práce 18

2.5.1 Dílčí výstupy vlastního výzkumu 19

2.6 Funkční požadavky 19

2.7 Nefunkční požadavky 19

3 Návrh a implementace **20**

3.1 Použité technologie 20

3.1.1 IPFIXcol2 20

3.1.2 Apache Kafka 22

3.1.3 Libfds 23

3.1.4 Librdkafka 24

3.1.5 Easylogging++ 24

3.2 Vývoj pluginu pro IPFIXcol2 24

3.2.1 Podmínky pro vytvoření výstupního pluginu 25

3.2.2 Návrh pluginu pro export dat do Apache Kafky 26

3.3 Výsledné provedení pluginu 27

3.3.1 Architektura pluginu 28

3.4 Testování 31

3.4.1 Měření rychlosti 32

Závěr **35**

Seznam použité literatury **36**

Seznam příloh **39**

Příloha 1 - Instalace pluginu **40**

Příloha 2 - Spuštění pluginu **41**

Příloha 3 - Nahrané soubory **43**

Seznam použitých zkratek

Plugin = rozšiřující modul pro program, díky kterému se doplní funkcionalita, která dříve v programu zahrnuta nebyla. Základní předpoklad je, že původní program bude tuto rozšiřitelnost podporovat.

PCAP = formát souboru, ve kterém jsou ukládány celé pakety síťové komunikace.[1]

IPFIX = formát pro sběr dat v počítačové síti, kde se oproti PCAP neukládá celá komunikace, ale pouze statistické údaje, např. odesílatel, příjemce, čas a celková velikost přenesených paketů. [2]

IPFIXcol2 = kolektor určený pro sběr dat ve formátu NetFlow v5/v9 a IPFIX od společnosti CESNET, který je navržen jako zcela modulární systém bez problému specifické rozšiřitelnosti. [3]

Lifds = knihovna použitá v IPFIXcol2 pro práci se záznamy se strukturou IPFIX a Netflow v5/v9. [4]

PMACCT = kolektor s možností nasazení sondy síťového provozu. [5]

JSON = strukturovaný textový formát, který je strojově i lidsky dobře čitelný.[6]

Message Broker = systémy určené pro předávání zpráv. Pomocí těchto systémů můžeme propojovat aplikace, které produkují data s libovolným počtem aplikací, které data konzumují. V základu je zde podpora dvou modelů pro předávání zpráv: Point-to-Point a Publish/Subscribe. Pro implementaci předávání zpráv je nejčastěji použita fronta (FIFO) [7]

Apache Kafka = systém určený pro předávání zpráv více konzumentům. Oproti ostatním message brokerům umožňuje i předávání zpráv v podobě streamu.[8]

Memory leak = únik paměti; nejčastěji způsoben ruční alokací a následně neprovedenou dealokací [9]

1. Úvod

Monitorování a sběr dat síťového provozu je nezbytným předstupněm a součástí ochrany počítačových sítí před možnými hrozbami, které jsou s rychlým tempem vývoje nových technologií, např. umělé inteligence, neustále sofistikovanější. Kritickým problémem je především rychlost a efektivita zpracování velkých toků dat, které v současnosti dosahují desítky až stovky GB/s v běžných sítích.

Jedním z nástrojů pro sběr dat síťového provozu je IPFIXcol2. Tento nástroj podporuje rozšíření své funkcionality pomocí pluginů, díky čemuž je možné vytvořit specifické prostředí dle preferencí uživatele. IPFIXcol2 je možné rozšířit o vstupní(input), vnitřní(intermediate) a výstupní(output) pluginy.

Vstupní pluginy se starají o příjem dat. V základu jsou s kolektorem distribuovány pluginy, pomocí kterých lze prostřednictvím TCP a UDP přijmout IPFIX a NetFlow záznamy. Rovněž je zde podpora čtení souborů typu FDS a IPFIX.

Vnitřní pluginy jsou zaměřeny na zpracování a úpravu záznamů. V tomto případě je s kolektorem distribuován plugin, který se stará o anonymizaci IPv4 a IPv6 adres.

Úkolem výstupních pluginů je distribuce dat z kolektoru do libovolného systému bez nutnosti vytvářet další programy, které by se staraly o napojení kolektoru k požadovanému systému.

Záměrem této práce je rozšíření kolektoru IPFIXcol2 o export dat ve formátu JSON do prostředí, které dokáže provést distribuci do dalších systémů a je uzpůsobené pro práci s velkým množstvím dat. Původně dokázal kolektor IPFIXcol2 exportovat výsledná data do databáze, souboru nebo je posílat v roli klienta či serveru další aplikaci pomocí TCP spojení.

Paralelně s průběhem této práce byl společností CESNET rovněž vyvinut plugin JSON-Kafka, který byl přidán do základní sady výstupních pluginů. Tento plugin taktéž nabízí podporu exportu dat do platformy Apache Kafka. Veškeré dostupné implementace těchto pluginů jsou provedeny jednovláknově, a proto se zde při větším zatížení kolektoru může objevit problém s rychlostí zpracování vstupních dat - konverze na JSON formát probíhá sekvenčně záznam po záznamu.

Oproti výše zmíněnému pluginu společnosti CESNET mnou vytvořený plugin zpracovává data paralelně, což představuje značnou výhodu.

Oba dva pluginy shodně umožňují vytvoření efektivní a modulární struktury systému složené z libovolné IPFIX sondy, IPFIXcol2 kolektoru a platformy Apache Kafka.

1.1 Cíle práce

Cílem práce je vytvoření funkčně odladěného výstupního pluginu pro kolektor IPFIXcol2 od společnosti CESNET. Důraz bude kladen především na maximální časovou efektivitu zpracování příchozích zpráv a následný export zpracovaných dat do Apache Kafky.

Součástí práce bude rovněž srovnání výsledného pluginu se systémem PMACCT z hlediska časové efektivity. Do tohoto srovnání bude pak přirozeně zahrnut i plugin JSON-Kafka.

2. Analýza

Pro monitorování provozu počítačových sítí dnes existuje široká paleta nástrojů. Od těch nejjednodušších, které poskytují pouze základní funkce s volbou parametrů prostřednictvím příkazové řádky, po systémy, v jejichž grafickém uživatelském rozhraní lze nakonfigurovat a zobrazit výsledky pokročilé analýzy zachycených dat. Zástupcem jednodušších nástrojů je třeba tcpdump, který umožňuje zachytávat komunikaci a ukládat ji jako pcap soubor, nebo IPFIXcol2, který zpracovává data ve formátu NetFlow a IPFIX. Naopak nástroj Kemp Flowmon kolektor umožňuje krom monitorování sítě také identifikaci bottleneck míst na síti, sledování provozu u SaaS aplikací a ukládání dat na cloud [10].

2.1 Teoretická východiska práce

Fundovaný vhled do problematiky a fungování kolektoru IPFIXcol2 nabízí diplomová práce Lukáše Hutáka [11]. Huták tento kolektor vytvářel z původního IPFIXcol. V jeho práci je popsané napojení pluginů k jádru kolektoru, princip interního předávání zpráv a zároveň i všeobecné informace o sběru dat síťového provozu za pomocí technologie NetFlow / IPFIX.

Krom teoretických poznatků z práce Lukáše Hutáka se vycházelo i ze zdrojových souborů kolektoru, které obsahují veškeré potřebné informace týkající se základního rozhraní pluginu pro napojení ke zbytku kolektoru s velice dobrou dokumentací jednotlivých metod.

Dalším zdrojem, ze kterého se čerpali informace, je diplomová práce Davida Bohmanna [12]. Autor se v této práci zabývá srovnáváním jednotlivých message brokerů a detailnějším popisem jejich nasazení. Na základě výsledků této práce jde poměrně jasně formulovat, proč je ve srovnání s ostatními klasickými message brokery nejvhodnější právě platforma Apache Kafka, která se zdá být nejuniverzálnější platformou pro předávání zpráv mezi systémy.

2.2 Sběr dat síťového provozu a distribuce záznamů

Monitorování síťového provozu a jeho následná analýza patří dnes mezi standardní prvky ochrany počítačových sítí. Pro monitorování existuje několik technologií, díky nimž lze tento proces zrealizovat. Nejčastěji se používají tech-

nologie založené na standardech NetFlow a jeho nástupci IPFIX. Ústředním prvkem těchto technologií je kolektor, do kterého proudí data z předem vybraných míst v monitorované síti. Kolektor tato data dle požadavků uživatele zpracuje a případně dále exportuje. Většina kolektorů poskytuje export do nejrůznějších databázových systémů pro dlouhodobé ukládání nebo do programů typu message broker, které jsou určené pro předávání zpráv. [11]

2.2.1 Technologie IPFIX a NetFlow

Nejznámější technologie pro monitorování sítí jsou IPFIX a NetFlow verze 5 a verze 9. Tyto pojmy označují jak samotný princip práce s daty (analýza a testování), tak samotné komunikační protokoly. Dá se říct, že technologie IPFIX se stává nástupcem NetFlow.

Použití technologie NetFlow v praxi mnohdy vyžadovalo dílčí úpravy provedené uživatelem. Na tento problém reaguje technologie IPFIX, kterou lze v tomto ohledu považovat za uživatelsky přívětivější a modulárnější. [11]

Technologie IPFIX se od přístupu uchovávání celých paketů odlišuje v samotné podstatě ukládaní informací. V případě ukládání celých paketů uchováváme celistvou komunikaci ze sítě i s obsahem jednotlivých zpráv, což je proces velice náročný na paměť kvůli ukládání, zvláště pokud stanice, na které provádíme monitorování, je v rušném prostředí. V tomto případě jsou získaná data přirozeně mnohonásobně větší. Samotné vyhledávání na takto velkém vzorku dat bývá náročné, v určitých případech dokonce zcela nemožné. IPFIX neukládá celou komunikaci, ale pouze statistické údaje komunikací. Díky této vlastnosti mají data zachycená pomocí technologie IPFIX menší paměťovou náročnost než data, která se zaznamenala pomocí celých paketů za stejný časový úsek. [13]

Nejčastěji ukládané položky v IPFIX:

- zdrojová adresa
- zdrojový port
- cílová adresa
- cílový port
- transportní protokol

- počet paketů
- počet přenesených bajtů

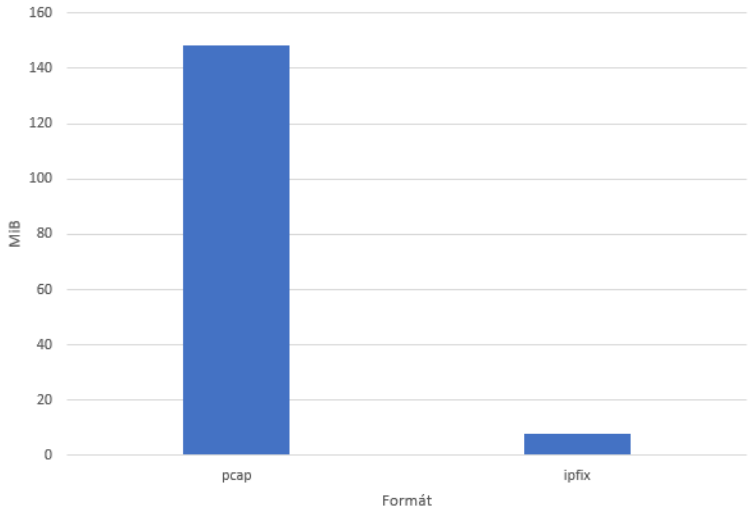


Figure 2.1: Velikost PCAP a IPFIX za stejný časový úsek

Rozdíly těchto technologií by se daly přirovnat k situaci, kdy jsou výpisy hovorů, u kterých se zná pouze, kdo komu volal a jak dlouho hovor trval, toto je případ technologie IPFIX, respektive výpisy hovorů včetně jejich kompletních nahrávek, což je případ zachytávání celých paketů.[11]

Jak lze vidět na obrázku 2.1, rozdíly velikosti uložených souborů jsou nezanedbatelné. Uživatel by se tedy měl rozhodovat i na základě tohoto parametru, jakou technologii použije, aby dosáhl svého záměru.

IPFIX se snaží zastřešit veškeré potřeby, které mohou u odchytu dat nastat, a nasadit tak jednotný protokol pro monitorování počítačových sítí. K IPFIX existují i různé alternativy. Jednou z nich je například technologie sFlow, která vybírá náhodné pakety, z jejichž hlaviček načte hodnoty parametrů, které potom předá k dalšímu zpracování. Tento postup tedy poskytuje méně realistický pohled na provoz počítačové sítě a může být v určitých ohledech zavádějící. [11]

2.2.2 Exportér (sonda síťového provozu)

Pro samotný monitoring je potřeba umístit do internetové sítě exportér, který dokáže předávat zachycená data do kolektoru.

Za zmínku dále stojí fakt, že standard IPFIX [14] mírně upravil definici exportéru, kdy u něj nově oddělujeme činnosti na tzv. měřící process (angl. Metering Process), který provádí činnosti od zachycení provozu až po tvorbu záznamů o tocích, a exportní process (angl. Exporting Process), jehož náplní je čistě přenos dat ke kolektoru [11]

Exportérem může být pouhý switch / router s možnou podporou exportu dat v NetFlow / IPFIX do kolektoru, jak lze vidět na obrázku 2.2 [15]. Toto řešení ovšem není vždy ideální, jelikož v některých případech není export řešen v reálném čase, ale je použita vyrovnávací paměť. Data se nejprve uloží do vyrovnávací paměti, ze které jsou po určitém čase exportována dále do kolektoru. Tento post-up by mohl způsobit problémy ve chvíli, kdy je potřeba pracovat s daty v reálném čase, případně je důležitý přesný časový údaj uložení záznamu.

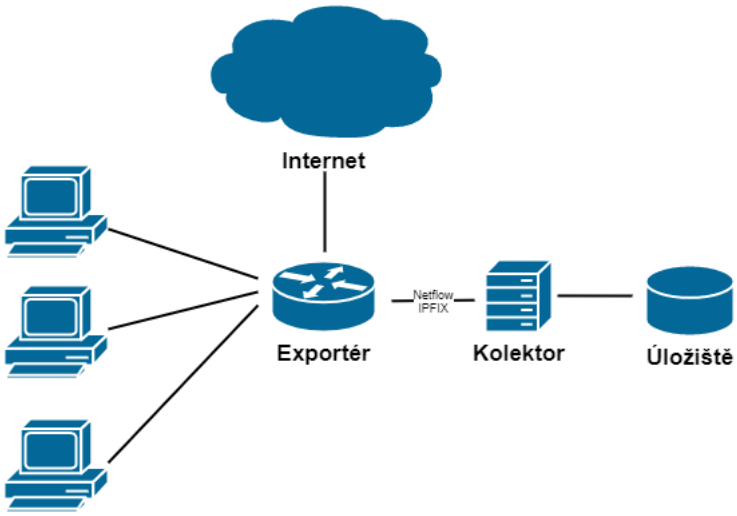


Figure 2.2: Exportér zabudovaný ve směrovači

Další možností je použití exportéru, který je v síti přidaný jako samostatné fyzické zařízení, přičemž komunikace je na něj pouze odkloněna, jak lze vidět na obrázku 2.3 [15]. Takovéto exportéry již dokáží komunikaci předávat bez nutnosti použití vyrovnávací paměti jak tomu bylo v předchozím případě.

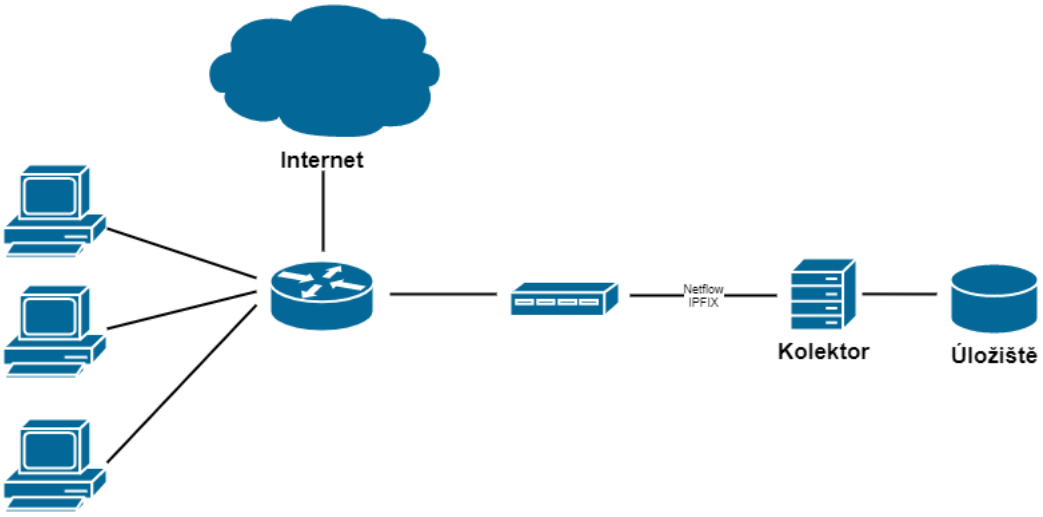


Figure 2.3: Exportér jako samostatné zařízení v síti

Výše popsané přístupy lze také nazvat jako Pasivní monitorování sítě, kdy je zařízení v síti pouze pro odposlech, data nijak neupravuje a nekontaktuje koncová zařízení v síti. Další možností je nasazení Aktivního monitorování sítě, což se provádí pomocí periodického dotazování na pozorované zařízení, a pokud toto zařízení neodpovídá nebo jeho odpověď nekoresponduje s odpovědí očekávanou, uživatel je na tuto skutečnost okamžitě upozorněn. [11]

Při aktivním monitorování se nesmí opomíjet na fyzické uspořádání sítě a s tím související závislost jedné služby na službách jiných. Dosti ilustrativním případem je situace výpadku směrovače (nebo přepínače), za kterým se nachází servery poskytující jednu nebo více sledovaných služeb. Jeden výpadek tam má za následek masivní příval hlášení nedostupnosti celé řady služeb. [11]

2.2.3 Kolektor

Hlavní úlohou kolektoru je centralizovaný sběr dat ze všech sond síťového provozu a následná práce s těmito zachycenými daty. Poněkud problematické může být u kolektorů nastavení příjmu dat, kdy sondy síťového provozu mohou být vůči nastavení kolektoru nastaveny mírně odlišně. Lišit se mohou jak v nastavení transportního protokolu (TCP, UDP), tak i protokolu aplikačního (NetFlow, IPFIX, sFlow). Pokud je v síti nasazeno více sond síťového provozu s rozdílnou konfigurací exportu dat a je potřeba tyto sondy napojit do jednoho místa, bude zapotřebí kolektor, který umožňuje sběr dat z více protokolů. Kolektory mohou provádět základní zpracování zachycených dat, zároveň je mohou připravit pro

pokročilejší analýzu, během které by původní formát záznamů nebyl vhodný. Po zpracování se záznamy předávají buď k finálnímu uložení, anebo k dalšímu zpracování. Zde je důležitý výběr kolektoru, jelikož každý kolektor může podporovat pouze určité exporty dat. [11]

2.2.4 Message broker

Message brokery se používají pro předávání zpráv mezi více systémy v situacích, kdy není potřeba tyto zprávy dlouhodobě ukládat. Díky message brokerům se mohou data mezi aplikacemi předávat rychle a spolehlivě. Další výhodou message brokerů je jejich možnost budoucí škálovatelností systému. Aplikace potřebují vědět pouze o existenci message brokeru, nikoliv o sobě navzájem, díky čemuž není problém jednotlivé aplikace upravit, popřípadě zcela nahradit.

Z hlediska architektury se dají aplikace rozdělit do dvou skupin:

- producenti
  - produkují data a ukládají je do message brokerů
- konzumenti
  - vybírají data z message brokerů a dále s nimi pracují

Existují tři nejčastější scénáře předávání zpráv pomocí message brokeru:

- Point-to-Point
- Publish-Subscribe
- Fan-Out

V případě použití scénáře Point-to-Point využívá message broker pro předávání zpráv strukturu fronta neboli FIFO (first in first out), kdy zpráva, která přijde do fronty jako první, jako první frontu také opustí. Point-to-Point se používá v případech kdy jedna nebo několik aplikací vytváří zprávy, které potom další aplikace (nebo i několik) konzumuje. Producent ukládá zprávy do fronty message brokera, odkud je následně vyzvedávají konzumenti. [12]

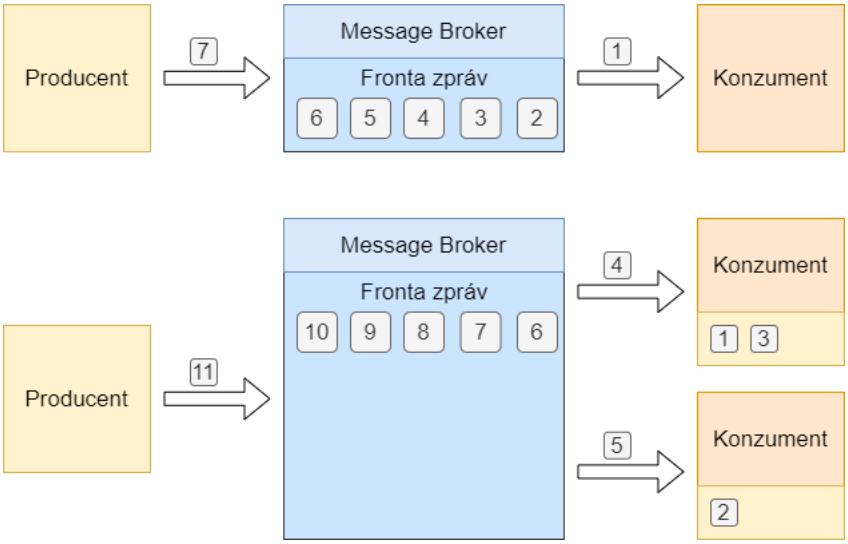


Figure 2.4: Architektura point to point

Scénář pro Publish-Subscribe je od předchozího mírně odlišný tím, že se zde neuchovává zpráva ve frontě, ale zpráva si s sebou nese svoje téma (topic) a je předaná aplikacím, které jsou v aktuální chvíli přihlášené k odběru tohoto tématu. V tomto případě tedy zprávu může obdržet více aplikací najednou, nikoli pouze jedna, jak tomu bylo v předchozím případě. Tento scénář, ale přináší problém ve chvíli, kdy zpráva přijde s tématem, které není odebíráno žádnou konzumující aplikací. Pokud tento případ nastane, existují dvě možnosti, jak s touto zprávou dále nakládat. V prvním případě se zpráva zahazuje. Ve druhém případě se zpráva uloží a dokud se nepřihlásí k tomuto topicu nějaký konzument zpráva zůstává uložena. Dalším problematickým bodem tohoto scénáře je situace, kdy se konzument přihlásí k odběru později (například po pádu aplikace, restartu, nebo je nově nasazen). V takové situaci totiž neobdrží předchozí posílané zprávy, jelikož se tyto zprávy v tomto scénáři nikam přechodně neukládají. [12]

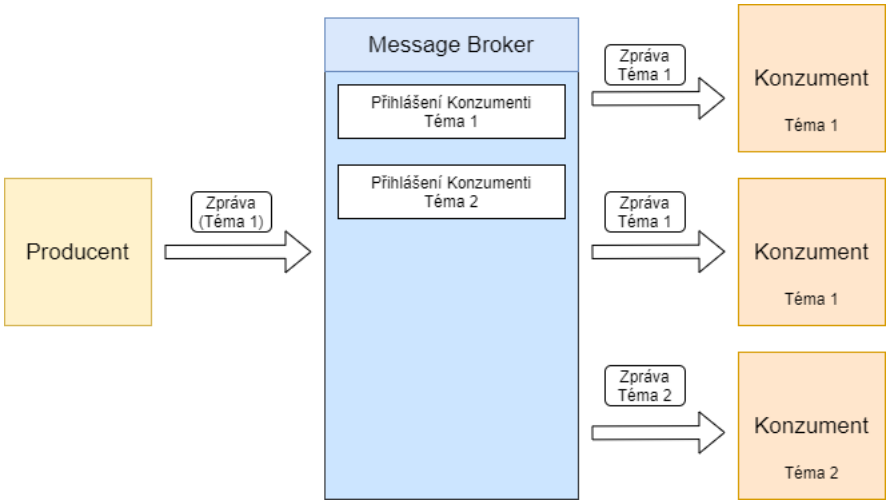


Figure 2.5: Architektura publish subscribe

Poslední scénář Fan-out kombinuje oba předchozí přístupy. Každý konzument má pro každý topic, který odebírá, svou vlastní, oddělenou frontu zpráv. [12]

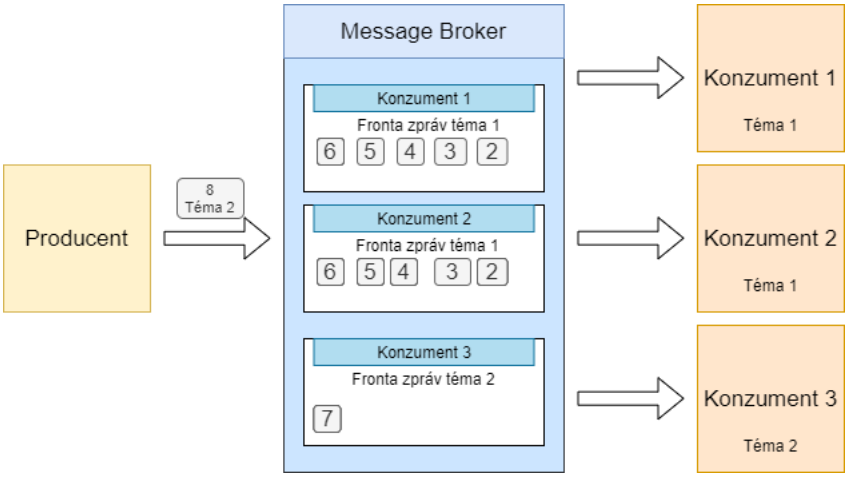


Figure 2.6: Architektura fan out

2.3 Systémy pro sběr dat síťového provozu

Jak bylo výše uvedeno, existuje nepřeberné množství hotových řešení pro sběr dat síťového provozu. Komplexnější systémy, které zastřešují kompletní zpracování, jsou většinou komerční, a je tedy potřeba si zakoupit licenci na jejich legální použití. Nevýhodou těchto řešení bývá takřka nulová rozšiřitelnost od koncového uživatele, jelikož tato řešení nemívají otevřený zdrojový kód. Menší systémy jsou často neplacené a distribuované se zdrojovým kódem, který se musí pro spuštění zkompilovat. V takovýchto systémech není problém pro osobní potřeby zdro-

jový kód modifikovat z důvodu přidání specifické činnosti. Toto se ale kvůli spolehlivosti příliš nedoporučuje, jelikož na tyto změny nebylo během vývoje pamatováno, a systémy se tak stávají potenciálně nespolehlivé. Existují však i systémy, které rozšiřování své funkcionality, často ve formě pluginů, přímo podporují.

Jako zástupce systémů s otevřeným zdrojovým kódem a podporou rozšiřitelnosti lze jmenovat například IPFIXcol2 a PMACCT.

2.3.1 IPFIXcol2

IPFIXcol2 je kolektor síťového provozu vyvíjený společností CESNET. Je nástupce IPFIXcol a řeší řadu nedostatků původního řešení. Má modulární architekturu, která uživateli umožňuje přizpůsobit podle svých požadavků, takřka celý proces zpracování dat síťového provozu. Důležitá část modulární architektury je možnost přidat další možné výstupy kolektoru, což je velmi důležité pro případné napojení kolektoru do dalších fází zpracování dat.

IPFIXcol2 lze používat zdarma a je určen výhradně pro systémy typu Unix. Je napsán v jazyce C a C++ ve verzi 11. Pro svou funkci vyžaduje knihovnu libfds, kterou rovněž vyvíjí společnost CESNET.

IPFIXcol2 pracuje pouze s formáty zpráv NetFlow verze 5/9 a IPFIX. [3] Primární defaultní podpora exportu podporuje export do souborů formátů FDS a IPFIX, popřípadě může být výsledek posílán pomocí TCP/UDP spojení. V průběhu vzniku této bakalářské práce byl také nově přidán plugin pro export dat do platformy Apache Kafka.

2.3.2 PMACCT

Řešení PMACCT je ve srovnání s kolektorem IPFIXcol2 mírně komplexnějšího rázu, jelikož PMACCT již v základu obsahuje nástroje pro realizaci systému tvořeného z pasivní sondy síťového provozu (za pomocí knihovny libpcap) a kolektoru. Oproti IPFIXcol2 je zde širší paleta podporovaných protokolů, kde kromě IPFIX a NetFlow nalezneme také protokol sFlow a použitím mírně odlišnější protokol NetLink, který se používá pro komunikaci mezi jádrem operačního systému Linux a procesy v uživatelském prostoru [16].

Podpora exportu dat je zde rovněž zastoupena v širší míře. K dispozici jsou zde databázové systémy jako například MySQL, PostgressSQL, MongoDB a rovněž nejpoužívanější message brokery RabbitMQ a Apache Kafka. Defaultní export dat je nastaven na memory table, ale je zde i možnost zapisovat data do souborů.

Jednotlivé části systému jsou spuštěny jako démoni. Jakýmsi ekvivalentem kolektoru IPFIXcol2 je démon nfacctd, který slouží jako kolektor dat pro protokoly NetFlow a IPFIX.

PMACCT je napsán v jazyce C a je určen pouze pro systémy typu Unix, podobně jako IPFIXcol2. [5]

2.4 Systémy pro distribuci dat

U výběru systému pro předávání dat se jako hlavní aspekty považují možné použité scénáře pro posílání dat, propustnost a škálovatelnost. Většina dostupných implementací těchto systémů podporuje širokou škálu programovacích jazyků, díky čemuž není problém propojit zcela odlišné systémy.

Tyto systémy se použitím nejvíce hodí pro časté předávání velkých dat, což se v případě této práce vyžaduje. Díky využití těchto systémů se programy spolehlivě propojí, zajistí rychlý přenos dat a umožní možnou škálovatelnost do budoucího rozvoje. Alternativní metody předávání dat jako je například využití TCP spojení nebo předávání pomocí souboru nedokáží poskytnou ani zlomek toho co dokáží tyto programy.

2.4.1 Apache Kafka

Apache Kafka není čistokrevná implementace message brokeru, ačkoliv podporuje klasické message broker scénáře. Hlavním rozdílem odlišujícím Apache Kafku od ostatních message brokerů je, že zde se zprávy mohou ukládat a následný přístup k nim je zajišťován offsetem. Tento rozdíl zásadně odlišuje Apache Kafku od klasických message brokerů, které zprávu pouze předají a poté ji automaticky zahazují. Tato vlastnost činí Apache Kafku jednou z nejuniverzálnějších platforem pro předávání zpráv.

Apache Kafka používá takzvaný "Dump broker" model (jinak nazvaný "smart consumer" model), kdy se message broker nestará o rozlišování zpráv na přečtené

a nepřečtené, které se mají smazat, ale pouze zprávu uloží do patřičného oddílu podle tématu a uchovává ji po danou dobu. Konzumenti si následně potřebné zprávy mohou přečíst podle offsetu.[17]

Za značnou popularitou Apache Kafky stojí rovněž široká podpora různých programovacích jazyků, od C / C++, C a Javy až po Python, Golang či Rust.

Platforma Apache Kafka je napsaná v jazyce Java, a není tedy problém s jejím nasazením na různé operační systémy, kde je nainstalované prostředí Java. Apache Kafka pro komunikaci nepoužívá protokoly známé z jiných systémů, jako je například MQTT, ale používá svůj vlastní binární protokol. [8]

2.4.2 RabbitMQ

RabbitMQ je často označována za nejúspěšnější implementaci message brokeru a oproti Apache Kafce se jedná o čistokrevnou implementaci podporující klasické scénáře. Rovněž je zde podpora klasických protokolů, jako jsou MQTT, STOMP a AMQP, díky čemuž se dá RabbitMQ použít v mnoha případech, kde je potřeba klasický message broker.

RabbitMQ používá takzvaný "Smart broker" model (jinak nazvaný "Dump consumer" model), kdy message broker rozlišuje, která zpráva je, nebo není přečtená, a kterou zprávu má tedy odstranit, popřípadě kterou má zachovat.[17]

RabbitMQ je naprogramován v jazyce Erlang, který je vhodný pro tvorbu škálovatelných aplikací, u kterých je kladen důraz na zajištění maximální odolnosti proti selhání [18]. Podpora programovacích jazyků je zde stejně dobře zastoupená, jako tomu je v případě Apache Kafky.[19]

Srovnání rychlosti Apache Kafky a RabbitMQ

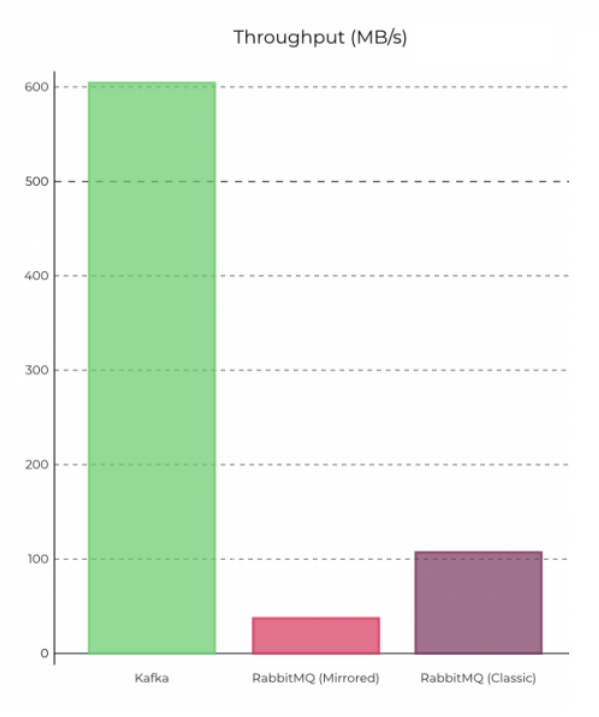


Figure 2.7: Kafka vs RabbitMQ propustnost

Z obrázku 2.7 [20] v celku jasně vyplývá, že Apache Kafka vykazuje oproti RabbitMQ značně větší propustnost. V každém řešení bylo vytvořeno 100 oddílů, do kterých čtyři producenti ukládali zprávy o velikosti 1KB, které byly čteny čtyřmi konzumenty. [20]

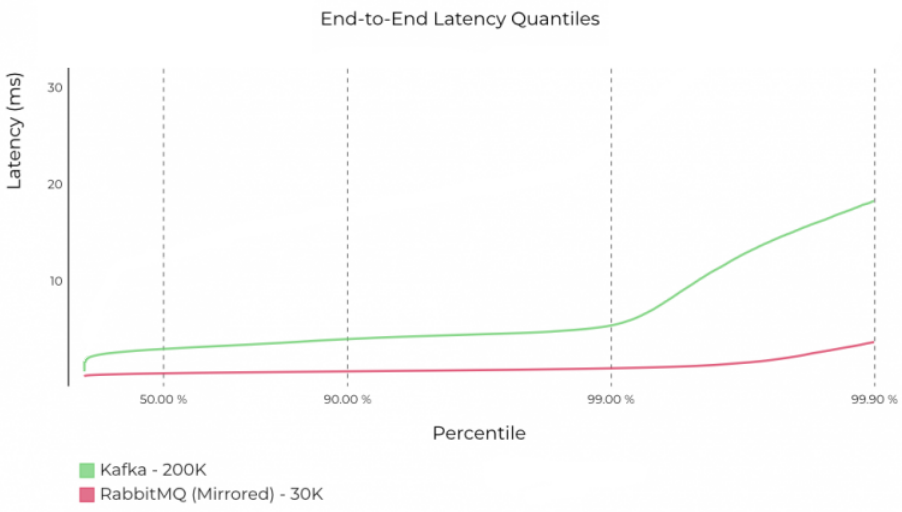


Figure 2.8: Kafka vs RabbitMQ odezva

Na obrázku 2.8 [20] lze vidět odezvu, přičemž nižší číslo reprezentuje lepší výsledek. Apache Kafka zde zpracovával 200 000 zpráv za sekundu a RabiitMQ pouze 30 000 zpráv za sekundu, jelikož následně přicházel bootleneck ze strany CPU. [20]

2.5 Metodika práce

Struktura vývoje této práce se dá přirovnat ke spirálovému modelu, kde se během iterací střídají fáze analýzy, plánování, vývoje a hodnocení [21]. Větší úkoly budou rozděleny na menší pod-úkoly, které se budou postupně řešit. V optimálním případě se při změně iterace půjde z jednoho funkčního celku do dalšího.

Pro vývoj se použilo vývojové prostředí Clion od společnosti JetBrains, které patří mezi jedny z nejlepších nástrojů pro vývoj aplikací v jazyce C a C++. Uživatelské rozhraní je rovněž velice přívětivé a patří rovněž mezi jedno z nejlepších. [22] Samozřejmostí je možnost snadné konfigurace dle preferencí vývojáře. Mezi další výhody, které toto IDE nabízí, patří výkonný refactoring, smart completetion, velké množství dostupných pluginů a pohodlná práce s CMake [23].

Velkou výhodou kolektoru IPFIXcol2 je existence jednoduchého nástroje IPFIXsend, který dokáže simulovat příjem dat ze sondy síťového provozu a díky tomu plugin otestovat na malé dávce dat. Nástroj lze konfigurovat různými parametry, což umožňuje simulovat různá síťová prostředí.

Pomocí nástroje IPFIXsend se bude plugin průběžně testovat. Testování bude prováděno na předem připraveném datasetu, u kterého budou výsledky zpracování známy. Jako dodatečnou kontrolu pro ověření správnosti fungování pluginu se použije porovnání získaných výsledků s výsledky získanými z pluginu Json.

IPFIXsend se krom kontroly konverze použije i na testování dlouhodobého provozu. Díky parametru, který určuje počet opakovaného zaslání konkrétního datasetu. Není problém vyzkoušet několikadenní provoz pluginu a získat tak výsledky bližší reálnému provozu. Tyto výsledky jsou vzhledem k realitě daleko přesnější oproti výsledkům, které by mohl poskytnout malý nárazový test, který má tendenci objevit pouze zásadní nedostatky a neodhalí například memory leak bez nástroje Valgrind, kterým bude výsledný plugin rovněž testován.

Měření propustnosti jednotlivých řešeních bude prováděno za pomocí větších datasetů. Testování každého řešení bude prováděno vícekrát, z čehož se následně vyvodí průměrný čas zpracování konkrétního datasetu. Pro komplexnější přehled bude použito více různě velkých datasetů.

2.5.1 Dílčí výstupy vlastního výzkumu

Během dosavadního výzkumu se zjistilo, že pluginy nejsou primárně zamýšlené jako vícevláknové aplikace, protože předávání zprávy ke zpracování není záležitostí pouhého předání ukazatele. V takovémto případě je totiž nutné udělat hlubokou kopii, aby byla zaručena relevantní existence zprávy v procesu zpracování pluginem. Pro kopírování jsou již před-připravené metody, které ale u určitých položek předají pouze ukazatel a nekopírují prvek po prvku. Tyto položky se tedy musí překopírovat ručně, aby nedocházelo k dvojitému uvolnění paměti.

Kolektor IPFIXcol2 a použitá knihovna jsou napsány z části v jazyce C. Je tedy nutné dát si pozor na dynamickou alokaci paměti, se kterou se bude dále pracovat i mimo vyvíjený plugin, kupříkladu u dynamického zvyšování velikosti zprávy během samotné konverze. Pokud by se toto zanedbalo, plugin by nefungoval správně a s velkou pravděpodobností by nebyl dlouhodobě stabilní.

2.6 Funkční požadavky

- Konverze zprávy z IPFIX do JSON formátu
- Podrobné logování stavu a běhu
- Export konvertovaných zpráv do Apache Kafky
- Nastavitelná velikosti vstupní fronty zpráv

2.7 Nefunkční požadavky

- Výstupní plugin IPFXcol2
- Určeno pouze pro Unixové systémy
- Maximální propustnost zpráv

3. Návrh a implementace

3.1 Použité technologie

3.1.1 IPFIXcol2

V této práci se bude pracovat s kolektorem IPFIXcol2 vyvíjeným společností CESNET, který se vyznačuje především svojí výkonností a vysokou modularitou. Modularita kolektoru spočívá v možné rozšiřitelnosti o případné pluginy (zásuvné moduly), které se dají připojit jako vstupní, vnitřní a výstupní. Architektura kolektoru je znázorněna na obrázku 3.1 [11]. Pluginy se do kolektoru přidávají jako dynamické knihovny a dají se rekonfigurovat za běhu kolektoru. [11]

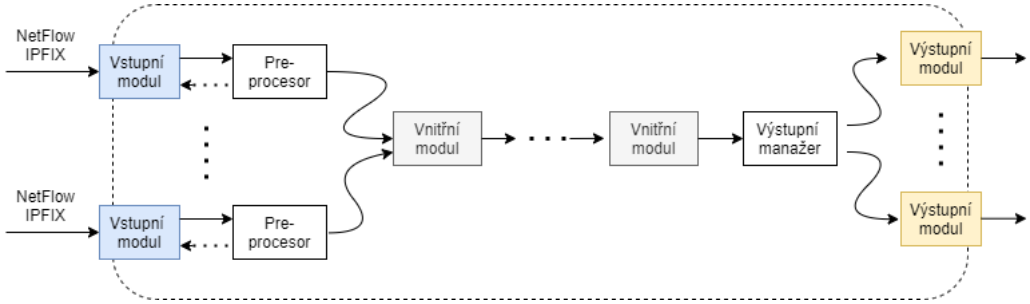


Figure 3.1: Architekrura IPFIXcol2

Počáteční nastavení kolektoru se provádí za pomocí konfiguračních souborů:

- běhová konfigurace
- konfigurace informačních elementů

V běhové konfiguraci se vybírá a nastavuje výčet použitých modulů, které má kolektor používat, a potažmo i jejich samotné nastavení. Tento konfigurační soubor je ve formátu XML a kolektoru se předá jako parametr "c". Struktura nastavení kolektoru je znázorněna na obrázku 3.2 [11]. Vnitřní uspořádání v konfiguračním souboru je odvozeno od struktury nasazování pluginů. Pokud by v tomto souboru byl uveden plugin, který by postrádal hodnoty pro své nastavení, použijí se výchozí hodnoty pluginu, které má každý plugin defaultně nastavené ve zdrojovém kódu.

```
<ipfixcol2>
  <inputPlugins>...</inputPlugins>
  <intermediatePlugins>...</intermediatePlugins>
  <outputPlugins>...</outputPlugins>
</ipfixcol2>
```

Figure 3.2: Zkrácené nastavení kolektoru

Konfigurace informačních elementů se od konfigurace běhu mírně odlišuje. Kolektor je distribuován se základní sadou elementů organizace IANA společně s dalšími elementy, které se využívají v nejběžněji používaných exportérech. V případě potřeby rozšířit tyto sady dalšími potřebnými elementy je zde možnost přiřadit další soubory, které potřebné sady ponesou. Konfigurační soubory informačních elementů se nacházejí v jasně dané adresářové struktuře, která je znázorněna na obrázku 3.3 [11]

- Kořen konfigurační složky
  - system *(soubory distribuované s kolektorem)*
    - elements
      - iana.xml
      - cesnet.xml
      - ...

      aliases.xml
  - user *(soubory modifikované uživatelem)*
    - elements
      - my_elements.xml
      - ...

      aliases.xml

Figure 3.3: Souborová hierarchie

Společně s kolektorem je distribuována i základní sada vstupních (input), vnitřních (intermediate) a výstupních pluginů. Vstupní pluginy umožňují přijímat data v rámci TCP resp. UDP spojení, vnitřní plugin podporuje anonymizaci IP adres (v nachytaných záznamech) a výstupní pluginy ukládají resp. exportují flow záznamy v různých formátech. Je zde obsažen i plugin pro konverzi zpráv z IPFIX a NetFlow do formátu JSON, kde se dají následně vypisovat na standardní výstup konzole, ukládat do souboru nebo posílat jako klient na server či jako server klientovi. V průběhu vytváření této bakalářské práce byl výrobcem do základní sady přidán také plugin na propojení s platformou Apache Kafka.

3.1.2 Apache Kafka

Apache Kafka je jedno z nejvíce univerzálnějších řešení pro předávání zpráv mezi systémy co se dá dnes použít. Oproti klasickým implementacím message brokera se jedná spíše o streaming brokera, který nabízí jak funkcionalitu klasického message brokera, tak i mnoho funkcí navíc. Hlavní rozdíl je zde v uchovávání zpráv, kde v klasické implementaci message brokera zprávy pouze předáme, zde jsou zprávy uchovávány. Zprávě se během uložení přiřadí jednoznačný offset, pomocí kterého se dá ke zprávě přistupovat a číst ji. Výhoda tohoto řešení spočívá v možném vícenásobném čtení jedné zprávy více konzumenty. Konzument si pouze zvolí pomocí offsetu jakou zprávu chce z Apache Kafky obdržet. [8] Tento mechanismus je vyobrazen na obrázku 3.4 [8].

Konzumenti připojeni k Apache Kafce si mohou navolit od jakého offsetu chtějí zprávy dostávat. Nejčastější nastavení jsou:

- od prvního záznamu
- od posledního přečteného záznamu

Zprávy které se ukládají v Apache Kafce jsou immutable, což znamená, že po uložení už nejdou změnit a jediná možnost jak zprávy modifikovat je nechat zprávu načíst konzumentem, upravit ji a uložit jako nový záznam na novém offsetu. [8]

Se zprávami se uloží i jejich časové razítko, kdy byly přesně uloženy do Apache Kafky.

Figure 3.4: Předávání zpráv Apache kafka

Apache Kafka využívá mechanismus "retention time", který se stará o management uložených zpráv a rozhoduje se na základě nastavení, která zpráva se má smazat a která se má ještě nechat uložená. V základu se tento mechanismus

rozhoduje na základě času, jak je daná zpráva dlouho uložena v Apache Kafce.

Apache Kafka také umožňuje nastavit maximální počet uložených zpráv, případně se mohou tato omezení specifikovat pouze na jednotlivá témata, nebo jednotlivé uzly v clusteru. [8]

Apache Kafka umožňuje rozdělení jednotlivých témat na oddíly (není zde problém udělat pro jedno téma jeden oddíl), což umožňuje následné rozdělení zátěže, kdy jednotlivé oddíly mohou běžet odděleně na jiných počítačových stanicích. Další výhody v rozdělení témat na oddíly spočívají v load balancingu, který je znázorněn na obrázku 3.5 [8], kde jednotlivé oddíly mohou být zpracovávány v několika brokerech umístěných v clusteru. Pro každý oddíl lze navíc vytvořit více "partition logů", díky kterým se mohou zprávy zpracovávat paralelně bez ohledu na ostatní konzumenty, které pracují se stejnými daty. [8]

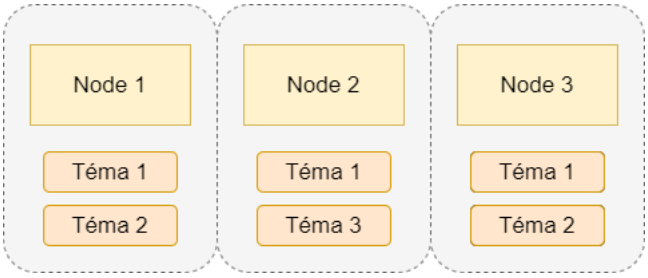


Figure 3.5: Apache kafka load balancing

Tato platforma rovněž podporuje mechanismus replikace, který se používá například v databázových systémech. Oddíly se dají replikovat na více počítačových uzlů, kde je jeden uzel nazvaný "leaderem" a ostatní uzly se nazývají "followeři". Zápis se provádí pouze na leaderovi, změny se poté propagují do všech followerů. Pokud leader přestane z jakéhokoli důvodu pracovat, je nahrazen některým z followerů. [8]

3.1.3 Libfds

Díky knihovně Libfds (flow data storage library) se dokáží zpracovávat IPFIX zprávy. Jsou zde obsaženy jak potřebné struktury, tak i parsery potřebné k jejich převodům, případně další nástroje k jejich zpracování. Jako další funkce této knihovny je jednoduchý XML parser, který dokáže kontrolovat typy. Tento parser je implementovaný jako wraper nad knihovnou libxml2, která se řadí mezi jedny z nejvíce používaných v projektech. Tuto knihovnu vyvíjí společnost CESNET

a kolektor IPFIXcol2 je postaven právě na této knihovně. [4]

3.1.4 Librdkafka

Knihovna Librdkafka implementuje binární protokol pro Apache Kafku a je napsaná v jazyce C. Je navržena s ohledem na vysokou propustnost a spolehlivost doručení zprávy. Rychlost zapisování pomocí této knihovny do Apache kafky se uvádí okolo 1 milionu zpráv za sekundu a čtení zpráv pomocí této knihovny se pohybuje okolo 3 milionů zpráv za sekundu. [24]

Krom implementace producenta na úrovni této knihovny lze využít předpřipravených wrapperů pro řadu programovacích jazyků.

V případě C++ existují dvě implementace:

- cppKafka
- modern-cpp-kafka

Tyto wrapery implementují veškerou potřebnou funkcionalitu již na úrovni C++, kde je plně využito objektového programování.

3.1.5 Easylogging++

Knihovna Easylogging++ je lightweight knihovna pro logování aplikací v jazyce C++, ve kterém je napsaná. Skládá se pouze z jednoho hlavičkového souboru, který se připojí k projektu. Mezi hlavní výhody této knihovny patří velká rychlost, možnost rozsáhlé konfigurace a jednoduchá rozšiřitelnost. Knihovna je "thread-safe" což umožňuje bezproblémové nasazení ve vícevláknových aplikací. Konfigurace této knihovny se provádí ze souboru s následujícím formátem [25]:

* LEVEL:
    CONFIGURATION NAME = "VALUE" ##Comment
    ANOTHER CONFIG NAME = "VALUE"

3.2 Vývoj pluginu pro IPFIXcol2

Kolektor IPFIXcol2 se skládá ze samotného jádra, na které se následně napojují zásuvné moduly, neboli pluginy. Pluginy se implementují jako dynamické knihovny, které následně volají služby poskytované jádrem.

3.2.1 Podmínky pro vytvoření výstupního pluginu

Pro vytvoření pluginu a jeho začlenění do IPFIXcol2 je zapotřebí implementovat k tomu určené rozhraní, definované v souboru "plugins.h". Obrázek 3.6 [11] poskytuje schéma napojení pluginu na jádro kolektoru.

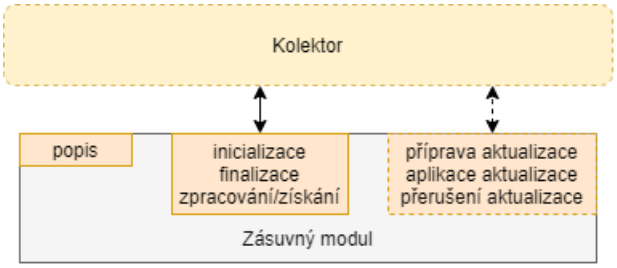


Figure 3.6: Rozhraní pluginu

Pro běh pluginu je nutné implementovat následující metody:

- `plugin_init()`
- `plugin_destroy()`
- `plugin_process()`

Metoda `plugin_init()` je volána při vytváření instance pluginu včetně nastavení jeho běhové konfigurace. Platí zde jedna podmínka, která říká, že pouze úspěšně inicializovaný plugin může být odstraněn pomocí metody `plugin_destroy()`.

Metoda `plugin_destroy()`, slouží k uvolnění dynamicky alokovaných prostředků pluginu.

Díky metodě `plugin_process()` se pluginu předává ukazatel na zprávu, která je určena ke zpracování. Zpráva se po provedení této metody odstraňuje z bufferu kolektoru. Pokud se má zpráva zpracovávat jiným vláknem, je potřeba v této metodě vytvořit hlubokou kopii zprávy. Pokud by se předal pouze ukazatel, po skončení této metody by byl nastaven na nesprávné místo.

Další nezbytná součást implementace pluginu je struktura `ipx_plugin_info`. V této struktuře se nastavuje:

- popis účelu pluginu
- v jaké části kolektoru se plugin nachází

  - vstupní
  - vnitřní
  - výstupní
- flagy
  - v tuto chvíli nejsou kolektorem imlementované
- verze pluginu
- minimální verze kolektoru

3.2.2 Návrh pluginu pro export dat do Apache Kafky

Hlavní prioritou při návrhu pluginu bylo dosažení pokud možno maximální časové efektivity zpracování příchozích zpráv, resp. se k ní přiblížit. Samotné zpracování příchozí zprávy zahrnuje 2 základní operace.

- konverzi na JSON formát
- export zkonvertované zprávy do Apache Kafka

Logickým rámcovým řešením byl vícevláknový návrh pluginu, kdy jednotlivá vlákna fungují jako konzumenti zpráv ukládaných do vnitřního bufferu pluginu.

Výsledný formát pluginu

Plugin by měl produkovat zprávy ve formátu JSON, který je jak dobře čitelný pro člověka, tak i výborně strojově zpracovatelný. Pro konverzi vstupních záznamů ve formátu IPFIX do JSON bylo možné použít knihovnu Libfds, která za tímto účelem poskytuje metodu `fds_drec2json()`.

Celková architektura nového výstupního pluginu

Aby výstupní plugin mohl dosahovat vyšší prostupnosti zpráv, byla již v začátcích práce předpokládána realizace vícevláknového zpracování, která se pro tyto typy úkolů běžně používá. Tím by se vyřešila zbytečná časová prodleva mezi přijetím zprávy ke zpracování a samotným zpracováním.

Zprávy jsou na vstupu vkládány do bufferu implementující FIFO, ze které je jednotlivá vlákna vybírají, konvertují do JSON a následně exportují. Efektivita tohoto provedení závisí na zdrojích, které se pluginu poskytnou. V ideálním

případě dokáže plugin zpracovávat zprávy za stejný nebo menší časový úsek, než co budou zprávy přicházet.

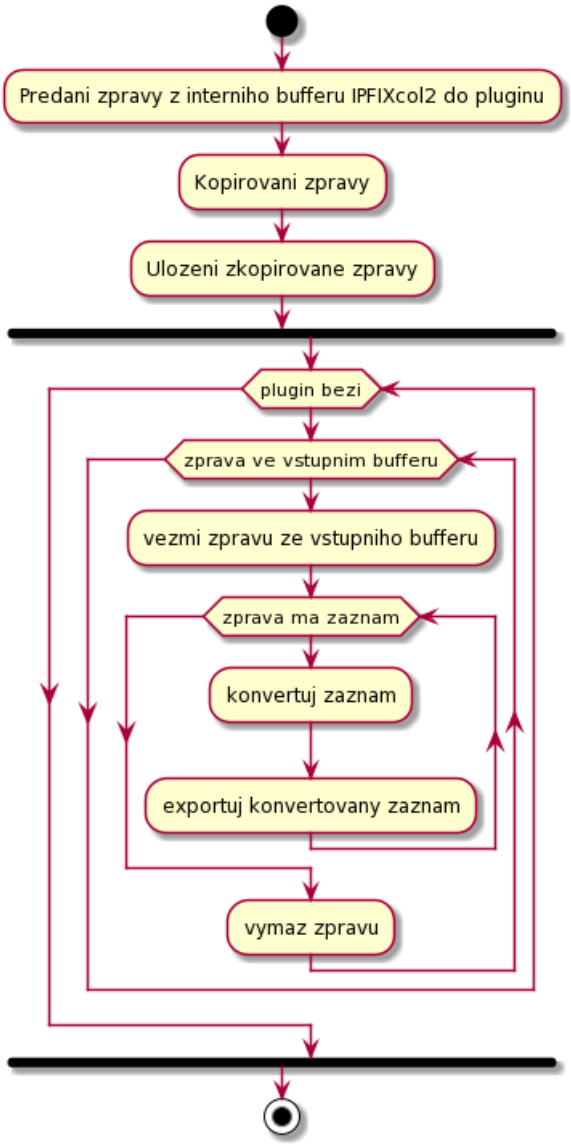


Figure 3.7: Architektura pluginu

3.3 Výsledné provedení pluginu

Plugin pro IPFIXcol2 je možné implementovat buď v jazyce C anebo v C++. V dnešní době je preferovanější C++, jelikož dovoluje využívat objektově orientované programování. Jako vzor pro pochopení napojení výstupního pluginu ke zbytku kolektoru slouží plugin Dummy. Díky jednoduchosti tohoto pluginu se dá jednoduše zorientovat v požadavcích pro správnou implementaci pluginu.

Pro ukázku konverze zpráv posloužil dosavadní plugin JSON, ve kterém je vidět proces zpracování zpráv ve formátu IPFIX na zprávy ve formátu JSON.

3.3.1 Architektura pluginu

Logger

Třída logger je malý wrapper pro knihovnu easylogging++. Použití wrapperu umožňuje využití rozhraní definované programátorem, díky čemuž se odstraní přímé reference na knihovnu. Díky tomuto řešení se dá použitá knihovna lehce zaměnit bez zbytečného přepisování kódu. Pokud by se v kódu vyskytovaly metody z rozhraní knihovny, nahrazení jinou knihovnou by obnášelo přepsat kód na všech místech, kde se volaly metody z původního rozhraní knihovny. Konfigurace této knihovny se provádí z odděleného souboru, na který je odkázáno v konfiguraci kolektoru.

Logovací soubory jsou důležité pro odhalování chyb a jejich následné opravení v dalších verzí programu. Logovací soubory jsou u nasazených aplikací jediný zdroj informací ohledně stavu aplikace.

JsonToKafka

Třída JsonToKafka implementuje potřebné rozhraní pro napojení ke zbytku kolektoru. Definuje se zde struktura `ipx_plugin_info` a metody `ipx_plugin_init()`, `ipx_plugin_destroy()`, `ipx_plugin_process()`.

V metodě `ipx_plugin_init()` se inicializuje nastavení pluginu, které se načítá z konfiguračního souboru kolektoru. Pokud načtení neproběhne korektně, plugin vrátí chybovou hodnotu a instance pluginu se nevytvoří. V případě úspěšného načtení se zde inicializuje Logger a vytvoří instanci Workera, která následně spouští pracovní vlákna.

Pomocí metody `ipx_plugin_process()` se pluginu předá ukazatel na zprávu určenou ke zpracování. Nejdříve se provede pokus o převedení zprávy na formát IPFIX metodou `ipx_msg_ipfix_create()`, které se předává ukazatel na zprávu. Pokud zpráva není ve formátu IPFIX vrací se chybová hodnota a dále se zpráva nezpracovává. V opačném případě se zde vytvoří kopie zprávy, aby její existence byla v režii pluginu.

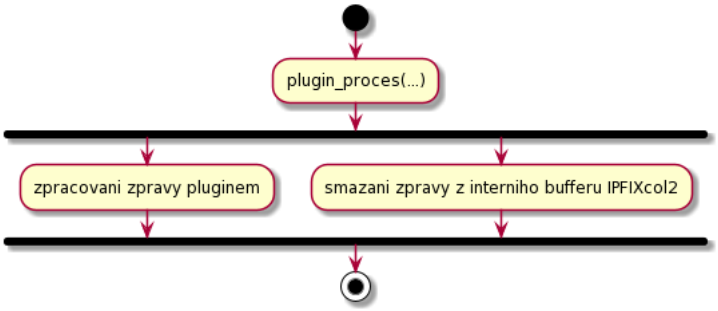


Figure 3.8: Existence zprávy

Pro vytvoření hluboké kopie se použijí metody:

- `ipx_msg_ipfix_create()`
- `add_drec_ref()`
- `fds_template_copy()`

Metoda `ipx_msg_ipfix_create()` vytváří obálku (wrapper) IPFIX zprávy, do které se budou ukládat jednotlivé záznamy.
Parametry této metody jsou:

- kontext pluginu
- kontext zprávy
- data ze zpracovávané zprávy
- velikost dat ze zpracovávané zprávy

Po úspěšném provedení metoda vrátí ukazatel na strukturu `ipx_msg_ipfix`, se kterou se bude dále pracovat.

Metoda `add_drec_ref()` slouží pro vytvoření nového záznamu v předem vytvořené obálce struktury `ipx_msg_ipfix`. Metoda jako parametr bere IPFIX zprávu a vrací ukazatel na nově vytvořený záznam v této IPFIX obálce. Díky tomuto ukazateli se nastaví potřebné hodnoty záznamu během kopírování. Opětovný přístup k tomuto záznamu je umožněn pomocí metody `ipx_msg_ipfix_get_drec()`, která si jako parametry bere zpracovávanou IPFIX zprávu a index požadovaného záznamu. Návratová hodnota této metody je ukazatel na záznam struktury `ipx_ipfix_record`.

Metoda `fds_template_copy()` slouží pro kopírování šablony IPFIX zprávy. Jako jediný parametr je zde šablona původního záznamu a nově vytvořená šablona se obdrží jako návratová hodnota z této metody.

Při vytváření hluboké kopie je potřeba dbát na dodržení interních požadavků kolektoru IPFIXcol2. Jako jeden z nejdůležitějších požadavků je správná alokace paměti pro dynamické struktury. Jádro kolektoru je napsáno z velké části v jazyce C a očekává se zde i dodržení standardů tohoto jazyka. Při vytváření dynamických struktur, které se budou spravovat metodami jádra kolektoru, je potřeba využít metod jako jsou `malloc()` nebo `calloc()`. Alokovaná paměť pomocí těchto metod se uvolňuje pomocí metody `free()`. V případě použití správy paměti pomocí C++ standardu, kde se využívá operátor `new` k alokaci paměti a k jejímu uvolnění operátor `delete`, může docházet k nežádoucím memory leakům (únikům paměti) a program se v praxi stává nepoužitelný.

KafkaProducer

Třída KafkaProducer implementuje producenta pro platformu Apache Kafka za pomocí knihovny Librdkafka. Díky této třídě je realizované spojení s Apache Kafkou a posílání již konvertovaných zpráv, kterým se zde přidá téma z běhové konfigurace.

Config

Třída Config načítá nastavení z běhového konfiguračního souboru. V případě absence tohoto souboru se nastavuje pluginy dle interních hodnot uložených v kódu. Tato třída se skládá z následujících struktur:

- ConfigFormat
- ConfigKafka
- ConfigProcessing

Struktura ConfigFormat uchovává hodnoty spojené s konverzí z formátu IPFIX do formátu JSON. Jedná se boolovské proměnné, pomocí nichž se nastavují flagy, které se předávají do metody `fds_drec2json()`.

Struktura ConfigKafka uchovává hodnoty pro navázání spojení s platformou Apache Kafka. Zde se jedná o textové hodnoty, které se předávají instanci třídy KafkaProducer.

Struktura ConfigProcessing uchovává hodnoty spojené se zpracováním zpráv uvnitř pluginu. Rovněž je zde uložena cesta ke konfiguračnímu souboru Loggera.

Worker

Třída Worker se stará o ukládání zpráv do vstupního bufferu pluginu a jejich následného zpracování, společně s exportem do Apache Kafky.

Vstupní pole je zde řešeno jako ring buffer a jeho velikost se dá nastavit pomocí běhového konfigurace kolektoru. Výchozí velikost bufferu je 1024.

Thread pool je nastaven na optimální počet vláken, který se zjistí pomocí metody `std::thread::hardware_concurrency()`. Tato metoda vrátí počet dostupných vláken. Výsledek této metody nemusí vždy korespondovat se skutečným počtem procesorů nebo jader, jelikož může být podporováno více vláken na jedno jádro, nebo naopak mohou být nějaké zdroje rezervované systémem a pro vyvíjený program nedostupné. [26]

Vlákna postupně vybírají zprávy ze vstupního pole a zpracovávají je. Pokud ve vstupním poli žádná zpráva není, tak se vlákno uspí a čeká, dokud nebude do pole uložená nová zpráva.

Před vlastní konverzí jsou ze zprávy získány jednotlivé záznamy, jejichž celkový počet vrací metoda `ipx_msg_ipfix_get_drec_cnt()`, která si jako parametr bere ukazatel na ipfix zprávu určenou ke zpracování. Jednotlivé záznamy ze zprávy se získají pomocí metody `ipx_msg_ipfix_get_drec()`, do které se předá ukazatel na ipfix zprávu a index záznamu. Po zpracování je třeba záznamy, šablonu a samotnou zprávu odstranit. K dealokaci zpráv je určena metoda `ipx_msg_destroy()`, která předpokládá že paměť pro zprávu byla alokována pomocí standardů jazyka C, tedy prostřednictvím funkce `malloc()` popř. `calloc()`.

3.4 Testování

Nasazení pluginu do provozu je popsáno v příloze 1 a 2, kde je zahrnuta jak instalace, tak následné spuštění.

Vytvořený plugin se musí důkladně otestovat kvůli bezproblémovému chodu. Ne-

jdříve se testuje s pomocí nástroje Valgrind, který odhaluje nedostatky které se týkají úniku paměti. Únik paměti je vždy závažný problém, jelikož aplikace se může chovat nestabilně a nestandardně.
Po úspěšné první fázi kde se kladl důraz především na větší problémy, se plugin začal testovat pomocí nástroje IPFIXsend s datasetem větších rozměrů, aby se otestoval dlouhodobý chod. Tato fáze je značně delšího trvání než fáze předchozí, ale předpokládá se, že plugin funguje korektně a případné chyby již nejsou natolik závažné jako v prvním případě. V případě chyb zde máme logovací soubor, kde se dá vyčíst aktuální stav pluginu a případnou chybu.

Poslední fází je testování stability oproti nasazené sondě síťového provozu jako je například IPFIXprobe od společnosti CESNET, nebo plugin nfprobe pro systém PMACCT. V této fázi se již neočekává nalezení problému, ale spíše ověření dlouhodobé spolehlivosti.

3.4.1 Měření rychlosti

Po otestování pluginu na chyby a stabilitu přichází fáze, kdy se bude vyvíjený plugin testovat oproti ostatním řešením.

Jednotlivá řešení se budou postupně testovat na několika dávkách záznamů, kde výsledný čas běhu pro každou dávku získáme pomocí časových razítek prvního a posledního uloženého záznamu v Apache Kafky. Pro získání těchto časových razítek se spustí jednoduchý konzument, který bude odebírat požadované téma. Zpracování celé dávky se pozná díky ukončení programu IPFIXsend a výpisu z konzumenta, který vypisuje stejné hodnoty. Po zastavení konzumenta se uloží časová razítka do souboru se jménem odebíraného tématu.

Výsledky měření

Měření se provádělo za pomoci nástroje IPFIXsend, díky kterému se dalo zvolit kolikrát a jak rychle se dataset zašle. Díky této architektuře se jednotlivá řešení dala jednoduše otestovat s postupně se zvyšujícím množstvím dat. Řešení postupně zpracovávala data od velikosti 17 GB do 34GB.

Vyvíjený plugin má během testování nastavenou velikost interního bufferu zpráv na 100 000 a velikost pro konvertované zprávy na 2048. Díky tomuto nastavení se minimalizovalo čekání na příchozí zprávy z kolektoru a také se předešlo případné

realokaci paměti během konverze zpráv.

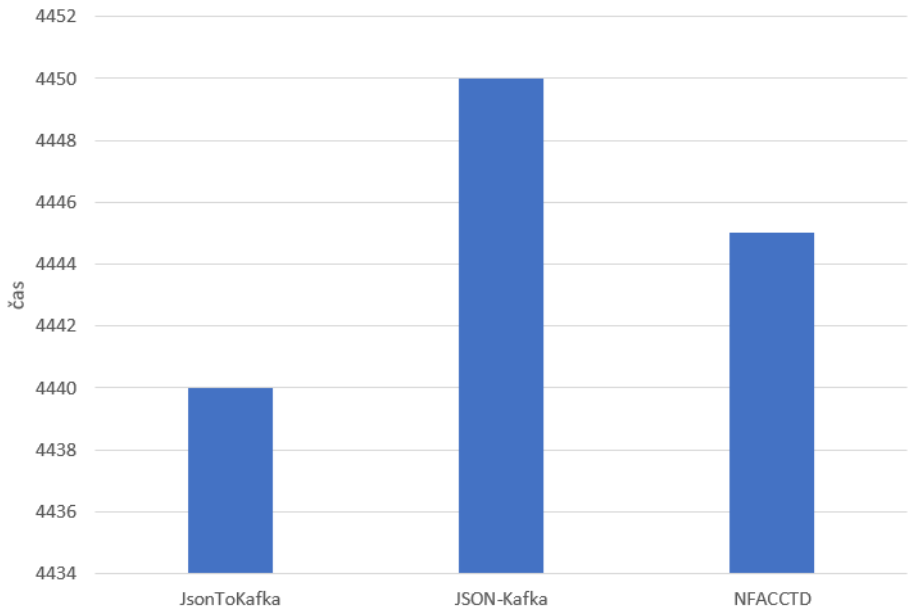


Figure 3.9: Srovnání rychlostí řešení

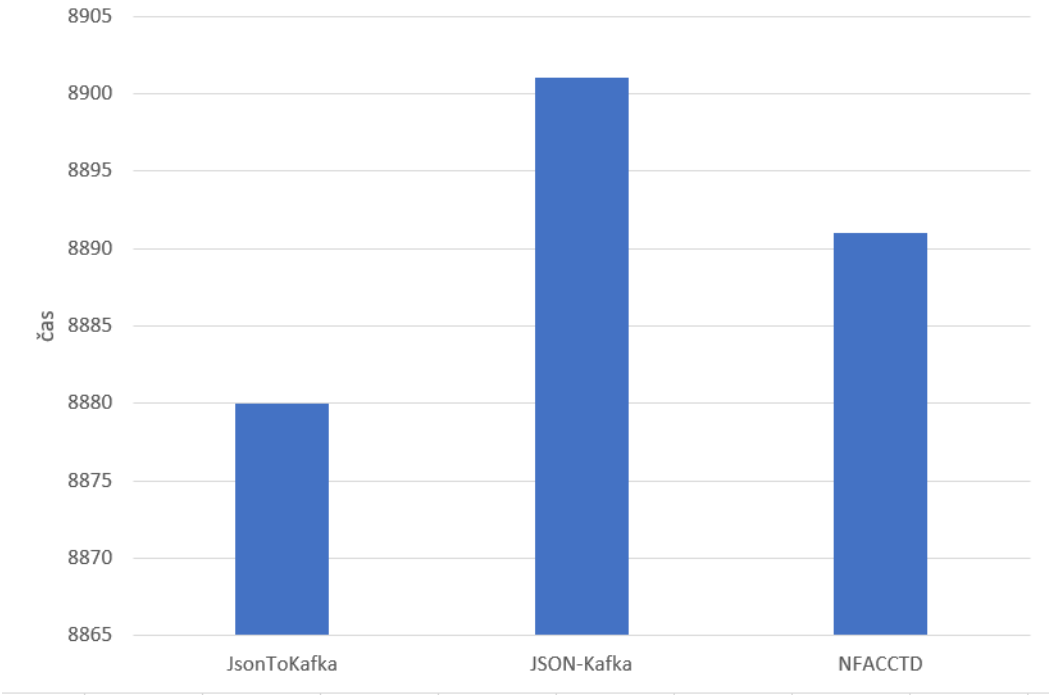


Figure 3.10: Srovnání rychlostí řešení

Na obrázku 3.9 lze vidět rychlost zpracování dat o velikosti 17 GB a obrázek 3.10 prezentuje situaci, kdy řešení zpracovávala data o velikosti 34 GB.

Výsledky jsou zde uvedeny ve vteřinách a platí, že čím menší hodnota, tím konkrétní řešení pracuje rychleji a dokáže zpracovat stejné množství dat za menší časový úsek než konkurenční řešení.

Vícevláknové zpracování zpráv se projevilo především u větších datasetů, kdy byl rozdíl od jednovláknových řešení v řádu vteřin. Důležité bylo nastavení dostatečné velikosti interního bufferu zpráv, který zajišťoval pracovním vláknům nepřetržitý přísun nových zpráv pro zpracování. Druhým důležitým aspektem je počáteční velikost pro konvertovanou zprávu. Pokud by se použila defaultní hodnota 1024, je možné, že během konverze nebude toto místo dostatečné a bude se muset provést alokace nového místa v paměti, kam se bude muset překopírovat původní obsah konvertované zprávy, což zabere určitý čas.

Závěr

Tato práce si kladla za cíl vytvoření plně odladěného pluginu pro systém IPFIXcol2 na export dat do platformy Apache Kafka a porovnání rychlosti zpracování dat oproti ostatním systémům v této kategorii. Testování výsledného pluginu ukázalo, že je schopen dosahovat lepších výsledků než jeho konkurence PMACCT a také nový plugin pro IPFIXcol2 se stejnou funkcionalitou.

Spolehlivost pluginu se během testování rovněž potvrdila a plugin je možné nasadit do běžného provozu bez větších provozních potíží.

Podle výše zmíněných vlastností pluginu se dá cíl práce považovat za splněný.

Seznam použité literatury

[1] Libpcap File Format. WireShark. Harris, 2015. Dostupné také z: https://wiki.wireshark.org/Development/LibpcapFileFormat

[2] NetFlow for Cybersecurity. Cisco press [online]. Hoboken, 2017 [cit. 2021-6-19]. Dostupné z: https://www.ciscopress.com/articles/article.asp?p=2812391&seqNum=4

[3] IPFIXcol2. GitHub [online]. San Francisco: Lukáš Huták, c2015-2017 [cit. 2021-5-5]. Dostupné z: https://github.com/CESNET/ipfixcol2

[4] Libfds: Flow Data Storage library. GitHub [online]. San Francisco: Lukáš Huták, 2021 [cit. 2021-11-29]. Dostupné z: https://github.com/CESNET/libfds

[5] PMACCT [online]. 2003 [cit. 2021-6-18]. Dostupné z: http://www.pmacct.net/

[6] JSON [online]. Mountain View: MDN, 2021 [cit. 2021-6-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

[7] Message Brokers. IBM [online]. Armonk, 2020 [cit. 2021-6-18]. Dostupné z: https://www.ibm.com/cloud/learn/message-brokers

[8] DOCUMENTATION. APACHE KAFKA [online]. Forest hill, c2017 [cit. 2021-10-4]. Dostupné z: https://kafka.apache.org/documentation/

[9] Memory leak in C++ and How to avoid it? Geeksforgeeks [online]. Mishra, 2020 [cit. 2021-5-5]. Dostupné z: https://www.geeksforgeeks.org/memory-leak-in-c-and-how-to-avoid-it/

[10] Kemp Flowmon kolektor. Flowmon [online]. [cit. 2021-11-29]. Dostupné z: https://www.flowmon.com/cs/products/appliances/netflow-collector

[11] HUTÁK, Lukáš. Nová generace IPFIX kolektoru. Antonínská 548/1, 601 90 Brno, 2018. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Jan Wrona.

[12] BOHMANN, David. Srovnání implementací message brokerů. Technická 8, 306 14 Plzeň 3, 2020. Diplomová práce. Západočeská univerzita v Plzni Fakulta aplikovaných věd.

[13] NetFlow AND PCAP (not or). Cisco Blogs [online]. Gavin Reid, 2016 [cit. 2021-11-20]. Dostupné z: https://blogs.cisco.com/security/netflow-and-pcap-not-or

[14] Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. IETF Datatracker [online]. Claise, B.; Trammell, B.; Aitken, P.;, 2013 [cit. 2021-12-05]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7011

[15] Network Traffic Collection with IPFIX Protocol. Žerotínovo nám. 617/9, 601 77 Brno, 2009. Diplomová práce. Masarykova Univerzita. Vedoucí práce Ing. Pavel Celeda, Ph.D.

[16] Netlink. Michael Kerrisk man7.org [online]. Kerrisk, 2021 [cit. 2021-10-29]. Dostupné z: https://man7.org/linux/man-pages/man7/netlink.7.html

[17] Kafka vs. RabbitMQ: Architecture, Performance Use Cases. Upsolver [online]. Eran Levy, 2019 [cit. 2021-11-19]. Dostupné z: https://www.upsolver.com/blog/kafka-versus-rabbitmq-architecture-performance-use-case

[18] Erlang/OTP. Erlang programing language [online]. San Francisco, c2010-2017 [cit. 2021-10-29]. Dostupné z: https://github.com/erlang/otp

[19] Documentation: Table of Contents [online]. c2007-2021 [cit. 2021-11-19]. Dostupné z: https://www.rabbitmq.com/documentation.html

[20] Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest? Confluent [online]. NIKHIL CHANDAR, 2020 [cit. 2021-11-19]. Dostupné z: https://www.confluent.io/blog/kafka-fastest-messaging-system/

[21] METODIKY VÝVOJE SOFTWARE. tř. Kosmonautů 1288, 779 00 Olomouc, 2018. Skripta. Moravská vysoká škola Olomouc, o. p. s.

[22] What are the best IDEs for C++ on Linux? Slant [online]. [2021] [cit. 2021-6-19]. Dostupné z: https://www.slant.co/topics/1411/~best-ides-for-c-on-linux

[23] CLion: A Cross-Platform IDE for C and C++ by JetBrains. JetBrains [online]. Praha: JetBrains, 2021 [cit. 2021-6-14]. Dostupné z: https://www.jetbrains.com/clion/

[24] Librdkafka. GitHub [online]. San Francisco: Magnus Edenhill, c2012-2020 [cit. 2021-11-29]. Dostupné z: https://github.com/edenhill/librdkafka

[25] Easylogging++. GitHub [online]. San Francisco: Majid, 2021 [cit. 2021-11-29]. Dostupné z: https://github.com/amrayn/easyloggingpp#using-configuration-file

[26] Std::thread::hardware_concurrency. Cplusplus [online]. c2000-2021 [cit. 2021-12-05]. Dostupné z: http://www.cplusplus.com/reference/thread/thread/hardware_concurrency/

Seznam příloh

- Příloha 1 - Instalace pluginu
- Příloha 2 - Spuštění pluginu
- Příloha 3 - Nahrané soubory

Příloha 1 - Instalace pluginu

Nejdříve je nutné připravit prostředí pro chod kolektoru IPFIXcol2. Z důvodu závislosti na knihovně `librdkafka` je nutné použít verzi kolektoru 2.2.0.

Zdrojové kódy pluginu se přidají do adresáře kolektoru `/ipfixcol2/src/plugins/output/`, ve kterém se nachází ostatní výstupní pluginy. V tomto adresáři je potřeba upravit `CMakeLists.txt`, do kterého se přidá odkaz na adresář se zdrojovými kódy kolektoru. Toto se provede pomocí příkazu `add_subdirectory(JsonToKafka)`.

Po přidání zdrojových kódů se musí upravit hlavní `CMakeLists.txt` v domovském adresáři kolektoru, ve kterém se změní závislost verze C++11 na verzi C++17 z důvodu použití nových funkcí jazyka.

Po těchto úpravách je možné kolektor zkompilovat a nainstalovat. V domovském adresáři kolektoru se spustí následující příkazy:

```
mkdir build
cd build
cmake ..
make
sudo make install
```

Po úspěšné instalaci se příkazem `ipfixcol2 -V` zjistí informace o nainstalovaném kolektoru a pomocí příkazu `ipfixcol2 -L` se zobrazí seznam dostupných pluginů.

Příloha 2 - Spuštění pluginu

Pro spuštění pluginu se předá kolektoru konfigurační soubor, ve kterém je plugin uveden následujícím příkazem `ipfixcol2 -c startup.xml`.

Ukázka struktury výstupního pluginu:

```
<output>
    <name>...</name>
    <plugin>...</plugin>
    <params>
        ...
        <kafka>
            ...
        </kafka>
        <processing>
            ...
        </processing>
    </params>
</output>
```

Položky v uzlu `<output>` nastavují informace ohledně pluginu, které slouží kolektoru pro identifikaci pluginu.

- `<name>`: jméno spuštěné instance pluginu
- `<plugin>`: jméno pluginu (knihovny) pro spuštění

Položky v uzlu `<param>` patří ke konfiguraci výsledného JSON formátu, které jsou společné s oficiálním pluginem JSON. [3]

- `<tcpFlag>`: převod příznaků TCP (formatted / raw)
- `<timestamp>`: převod časového razítka (formatted/ raw)
- `<protocol>`: převod identifikace protokolu (formatted / raw)
- `<ignoreUnknown>`: přeskočení pole s neznámým názvem (true / false)
- `<ignoreOptions>`: přeskočení záznamů o hlášení (true / false)
- `<nonPrintableChar>`: přehlížení bílých znaků (true / false)
- `<numericNames>`: použití krátké identifikace (true / false)

- `<octetArrayAsUnit >`: převedení všech polí na číslo (true / false)
- `<splitBitflow>`: rozdělení biflow záznamů na dva toky (true / false)

Položky v uzlu `<kafka>` patří ke konfiguraci producenta pro Apache Kafku.

- `<hostName>`: adresa Apache Kafky
- `<port>`: port Apache Kafky
- `<topicList>`: téma zprávy pro Apache Kafku

Položky v uzlu `<processing>` se vztahují ke konfiguraci pluginu samotného.

- `<processMessageLength>`: předem alokovaná paměť pro konverzi zpráv
- `<messagesBufferSize>`: velikost interního bufferu zpráv
- `<loggerConfigFile>`: cesta ke konfiguračnímu souboru loggeru

Pro nastavení logování pluginu se používá samostatný soubor, na který je odkázáno v běhové konfiguraci kolektoru (položka `<loggerConfigFile>`). Formát konfiguračního souboru loggeru:

```
* GLOBAL:
    FORMAT = "%datetime %msg"
    FILENAME = "/tmp/logs/json-to-kafka.log"
    ENABLE = true
    TO_FILE = true
    TO_STANDARD_OUTPUT = false
    MAG_LOGFILE_SIZE = 2097152
    LOG_FLUSH_TRESHOLD = 100
* DEBUG:
    FORMAT = "%datetime{%d%M} %func %msg"
```

Příloha 3 - Nahrané soubory

- JsonToKafka - zdrojové kódy pluginu
- CMakeLists.txt - upravený CMakeLists.txt IPFIXcol2 s podporou C++17
- startup.xml - konfigurační soubor kolektoru s nastaveným vytvořeným výstupním pluginem JsonToKafka
- loggerConfig.conf - konfigurační soubor loggeru pluginu