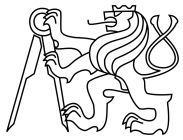
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Sledování provozu 100Gb/s síťových infrastruktur
Student:	Bc. Miroslav Kalina
Vedoucí:	Ing. Tomáš Čejka
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Prostudujte problematiku monitorování počítačových sítí a zaměřte se na sledování síťových toků.
Seznamte se se systémem Nemea pro analýzu síťového provozu.
Navrhněte množinu statistických informací, která bude reprezentovat strukturu a objem síťového provozu v síťové infrastruktuře.
Navrhněte vhodné datové struktury a algoritmy pro real-time analýzu síťového provozu vhodné pro velké a rychlé sítě (100Gb/s).
Implementujte softwarový modul pro sbírání a výpočet navržených statistik jako rozšíření systému Nemea.
Ověřte funkčnost vzniklého modulu s použitím dat z reálné síťové infrastruktury (dodá vedoucí).

Seznam odborné literatury

https://github.com/CESNET/Nemea
V. Bartos, M. Zadnik, T. Cejka: Nemea: Framework for stream-wise analysis of network traffic, CESNET technical report 6/2013.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 10. ledna 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Sledování provozu 100Gb/s síťových infrastruktur

Bc. Miroslav Kalina

Vedoucí práce: Ing. Tomáš Čejka

10. května 2016

Poděkování

Děkuji především Ing. Tomáši Čejkovi za jeho vedení a věcné diskuse k tématu mé práce. Poděkovat bych chtěl ale všem, kteří mě po celou dobu mých studií podporovali, zejména však mé rodině, protože bez nich bych tuto práci rozhodně napsat nedokázal.

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.

V Praze dne 10. května 2016 .....

Odkaz na tuto práci

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

Abstrakt

Obsahem této práce je návrh a následná implementace agregačního modulu pro NEMEA Framework. Výsledný modul bude umožňovat uživateli definovat pravidla pro zpracování velkého množství informací do agregovaných záznamů pro jednodušší následující zpracování.

Klíčová slova NEMEA, TRAP, UniRec, agregace, URFilter, monitorování sítí, NetFlow, IPFIX

Abstract

The main goal of this document is the design and implementation of aggregation module for NEMEA Framework. End user will be able to define rules for the module to progress large amount of information and create aggregated recorde for easier following processing.

Keywords NEMEA, TRAP, UniRec, aggregation, URFilter, network monitoring, NetFlow, IPFIX

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Počítače a výpočetní technika nás v dnešní době obklopují na každém kroku. V zájmu rozšiřování funkčnosti je trend výpočetní techniku vzájemně propojovat do počítačových sítí. Připojení k celosvětové síti Internet již několik let není žádnou výsadou. Ve školách, firmách a domácnostech jsme na dostupnost připojení k Internetu zvyklí a pokládáme jej za samozřejmost.

Internet se stává zejména pro firmy, ale i uživatele, stále důležitější součástí denního života. Není tak divu, že právě služby dostupné z Internetu jsou často cílem různých forem útoků jako projev konkurenčního boje nebo různě závažné kriminální činnosti. Takovému chování se samozřejmě chceme umět co nejlépe bránit a proto existuje obor počítačové bezpečnosti, který se rozvíjí každým dnem.

Součástí počítačové bezpečnosti jsou např. definice procesů minimalizující šanci na průnik do počítačových systémů, pravidelná aktualizace a kontrola systémů, ale i monitorování a analýza síťové komunikace. Právě monitorováním počítačových sítí se budu zabývat v této diplomové práci.

Sledování provozu v počítačové síti a jeho automatická analýza nám dává možnost včas odhalit snahy o průnik do našich systémů či detekovat nestandardní chování některých částí naší sítě. Detekované změny v chování mohou znamenat, že naše systémy byly napadeny, a my tak můžeme včas reagovat a minimalizovat způsobené škody.

Monitorování sítě nám však nemusí sloužit pouze k detekci útoků, ale můžeme jej zároveň využít i ke sledování funkčnosti naší infrastruktury a např. detekci přetěžovaných linek a včasnému navýšení jejich kapacity.

V České republice existují komerční společnosti[1], které se zabývají vývojem potřebných nástrojů, ale mě zaujal především projekt NEMEA rozvíjející se na poli univerzit ČVUT v Praze a VUT v Brně pod křídly výzkumného sdružení CESNET.

[1]Příkladem společnosti zabývající se monitoringem počítačových sítí je Flowmon Networks a.s. sídlící v Brně

NEMEA je modulární open-source nástroj pro sledování počítačových sítí velkého rozsahu na základě analýzy síťových toků. Vyvíjí se již několik let a mě se jeho koncept velice zalíbil a rozhodl jsem se pomoci s vývojem a implementovat jeden z nových systémových modulů. Zpracování velkého množství síťových toků, řádově stovky tisíc toků za sekundu, v reálném čase je výpočetně náročné a proto se v projektu klade velký důraz na efektivitu implementace a použitých datových struktur.

Zpracovávaných dat je velké množství a v projektu aktuálně chybí modul, který by uměl data nějakým způsobem agregovat, aby bylo jejich další zpracování jednodušší. Podobný modul by mohl být rovněž použit např. pro agregování událostí z detektorů.

Mým hlavním přínosem do tohoto projektu bude vytvoření univerzálního modulu pro výpočet agregačních funkcí ze síťových toků v definovaných časových intervalech. Modul bude primárně zaměřen na agregování síťových toků, ale vzhledem k univerzálnímu rozhraní jej bude možné zapojit na kterékoliv místo systému.

Svůj další přínos vidím také v oddělení filtrovací funkcionality z již existujícího modulu tak, aby bylo možné ji jednoduše použít v dalším vývoji.

V práci zprvu uvedu problematiku dnešních počítačových sítí a některé z možností jejich monitorování. Dále se zaměřím na sběr síťových toků a možné způsoby jeho nasazení a zpracování.

Představím NEMEA Framework zprvu obecně a následně detailně vysvětlím jeho stěžejní části včetně jejich typického použití.

Přiblížím motivaci pro vznik agregačního modulu a budu se věnovat teoretickému pozadí a možným datovým strukturám, které bude možné použít pro průběžný výpočet agregačních funkcí. Navrhnu a objasním konkrétní podobu modulu včetně jeho chování, vnitřních datových struktur a dalších součástí.

Popíšu implementaci modulu, objasním prováděná rozhodnutí a shrnu potřebné závislosti. Představím také implementaci kruhové časově orientované databáze TimeDB, která je použita výhradně k výpočtu agregací.

Nakonec výsledný modul otestuji a nabídnu některé konkrétní příklady, jak by bylo možné modul prakticky použít.

Kapitola 1

Počítačové sítě

V úvodu bych rád objasnil několik pojmů a přiblížil problematiku počítačových sítí. Zjednodušeně by se dalo říci, že se jedná o libovolnou skupinu počítačových systémů, které jsou vzájemně propojené. V rámci takové skupiny se předpokládá, že jednotlivé počítače spolu budou komunikovat i když ne nutně každý s každým.

1.1 Síťové prvky

K vzájemnému propojování počítačových systémů a síťových zařízení používáme tzv. síťové prvky, které dělíme na dvě základní skupiny. Pasivní prvky, kam řadíme např. kabely, optická vlákna či konektory [2], a aktivní prvky, z nichž některé zběžně uvedu.

Přepínač

Přepínač, nebo-li switch, je síťový aktivní prvek, který slouží k propojení zařízení v rámci jedné lokální sítě. Jeho úkolem je tedy doručovat síťové rámce cílovým počítačům. Pracuje tedy prakticky výhradně na 2. vrstvě ISO/OSI modelu.

Většina konfigurovatelných přepínačů podporuje virtuální sítě, které dovolují, zjednodušeně řečeno, seskupovat porty jednoho fyzického přepínače do více oddělených logických skupin, což zajišťuje flexibilitu ve správě počítačové sítě.

Směrovač

Směrovač, nebo-li router, je další zástupce aktivních sítových prvků, který přímo odděluje dvě či více počítačových sítí. Jeho hlavním úkolem je zařídit,

[2] Obecně by se dalo říci, že se jedná o součásti sítě, které ke svému provozu nepotřebují elektrickou energii.

aby pakety dorazily do té správné sítě.

Ve své paměti má uloženou tzv. směrovací tabulku, což je soubor informací o tom, kterým směrem má přeposlat pakety, aby byly doručeny do cílových sítí optimální cestou.

Směrovací tabulka může obsahovat pro jednu cílovou síť více záznamů s různou hodnotou metriky. Aktivně se vždy používá pouze jedna z nich a ostatní slouží jako záloha pro případ, že by nebylo možné aktivní cestu použít (např. z důvodu poruchy jiného přímo připojeného směrovače).

Existuje způsob, jak je možné upravovat směrovací tabulku dynamicky bez přímého zásahu správce, což je výhodné například při velkém množství spravovaných směrovačů. Slouží k tomu směrovací protokoly jako jsou např. RIP, OSPF či BGP.

1.2 Struktura počítačových sítí

Středně velká počítačová síť, znázorněná obrázkem 1.1, obsahuje na svém okraji jeden či více směrovačů. Tyto směrovače mají primárně za úkol obsluhovat veškerý kontakt sítě s okolním světem a pokud je síť přímo připojená k celosvětovému BGP směrování [3], tak i oznamovat existenci naší sítě do světa. Více směrovačů má většinou smysl právě pouze ve spojitosti s BGP směrováním, čímž se dosahuje odolnosti proti výpadku HW a vyšší dostupnosti služeb.

Na hraniční směrovače navazuje sada přepínačů a směrovačů, které dohromady tvoří tzv. distribuční vrstvu počítačové sítě. Jejím hlavním úkolem je přenášet data mezi hraničními směrovači a přístupovou vrstvou, připojující koncová zařízení. Tato část sítě je umístěna v jejím topologickém středu, prochází přes ni drtivá většina dat a je proto vždy dimenzována na vysokou propustnost.

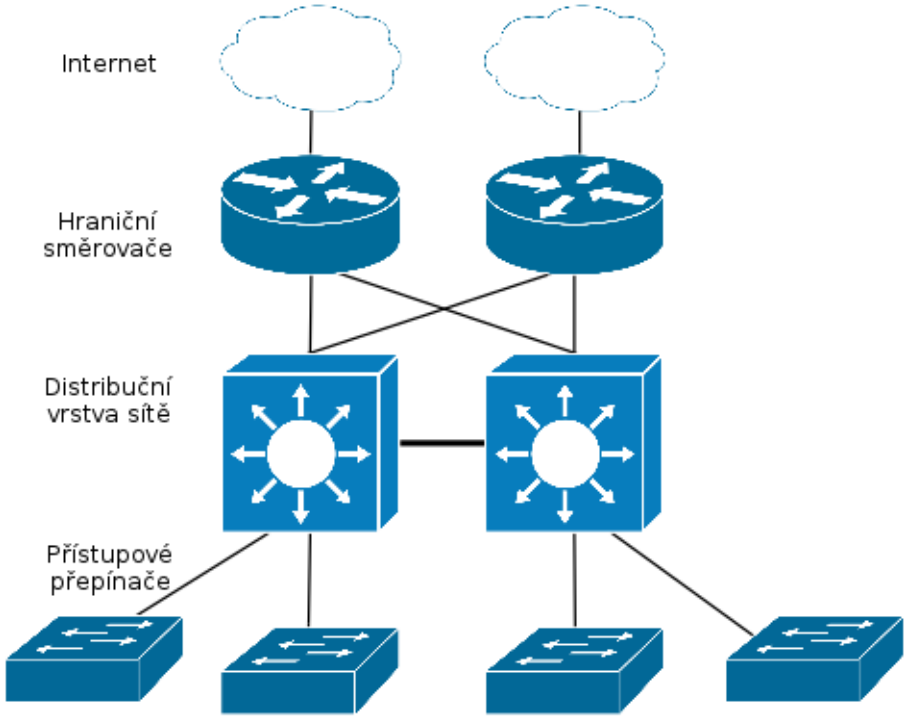
Přístupová vrstva, jak už bylo řečeno, navazuje na distribuční vrstvu. Je složena převážně z přepínačů a přímo připojuje koncové stanice a servery. Před některé segmenty sítě, které chceme chránit, se umisťuje tzv. firewall, což je aktivní zařízení pro filtrování provozu vedoucího do chráněného segmentu. Zmíněným segmentem je typicky kancelářská síť či skupina serverů.

V malých sítích můžeme často vidět, že hraniční a distribuční část sítě jsou sloučeny do jedné společné části. Taková část bývá reprezentována jedním jediným směrovačem.

1.3 Monitorování síťového provozu

Získat přehled o stavu počítačové sítě je jedním ze základních způsobů jak ji provozovat. Hlavním účelem monitorování počítačových sítí je upozornit jejich správce na problémy, které mohou v síti nastat.

[3] Směrovací protokol BGP se používá k distribuci informací o všech dostupných počítačových sítích v rámci celosvětové sítě Internet.



Obrázek 1.1: Příklad větší sítě rozdělené na hraniční, distribuční a přístupový segment.

Základní způsoby monitorování kontrolují dostupnost segmentů sítě a poskytují informaci o velikostech datových přenosů v jednotlivých fyzických segmentech sítě. Dohled se většinou realizuje pomocí kontroly dostupnosti jednotlivých koncových stanic a síťových prvků a sběru statistických informací z jejich rozhraní. Možnosti upozornění na problémy jsou velice malé, získáme pouze informaci o závadě či přetížení síťových segmentů, což v dnešní době není dostačující.

Pokročilejší dohled síťové infrastruktury zahrnuje nástroje, které budou přímo sledovat a analyzovat provoz v počítačové síti. Příkladem takových nástrojů jsou Network IDS systémy (v překladu Síťový systém pro detekci průniků), které získávají přímá či nepřímá data o síťovém provozu a pokoušejí se v nich detekovat různé druhy počítačových útoků. V případě úspěšného odhalení aktivního útoku jej mohou buď pouze ohlásit (pasivní systémy) nebo rovnou provést automatizovanou akci vedoucí k jeho zablokování (reaktivní systémy).

IDS systémy mohou k detekci útoků používat dvě odlišné metody:

- Hledání útoků na základě statistických anomálií. Patří sem zejména tzv. behaviorální analýza, která se snaží účastníky síťové komunikace rozdělovat do skupin s podobným chováním na síti a snaží se detekovat změny v jejich chování.

  Výhodou takových algoritmů je, že se zaměřují na podezřelé chování a jsou schopny detekovat i útoky, které ještě nebyly dříve popsány. Nevýhodou je vyšší náchylnost k falešným alarmům a vyšší zpoždění detekce útoku.

- Detekce na základě známých signatur a obecně podezřelého chování uložených v databázi detektoru.

  Výhodou tohoto přístupu je spolehlivost a rychlost detekce. Hlavní nevýhodou je, že lze detekovat pouze předem známé a popsané typy útoků.

1.3.1 Paketová analýza

Jedním z přístupů k monitorování sítí je přímé sledování provozu na síti a analýza každého paketu, který je prostřednictvím sítě přenášen. Analyzátor tak sleduje, které služby jsou na síti k dispozici a je schopen detailně prověřit každé síťové spojení a určit, zda se jedná o regulérní síťový provoz či potencionální útok.

Velkou výhodou paketové analýzy je, že umožňuje nahlížet do aplikačních vrstev síťového provozu a analyzovat provoz velmi detailně. Sítí však běžně prochází velké množství paketů a analýza každého jednotlivého z nich vyžaduje vysoký výpočetní výkon, protože je potřeba zpracovávat data v reálném čase.

Z toho důvodu se většinou v těchto případech využívá detekce známých signatur provozu, protože má nižší paměťové a výkonnostní nároky než behaviorální analýza.

Tyto analyzátory mívají relativně nízkou propustnost [1] (řádově stovky tisíc paketů za sekundu, což přibližně odpovídá jednotkám gigabitů za sekundu) a jsou tak vhodné především k monitorování sítě omezené velikosti.

Příklady dostupných produktů

Zástupcem paketových analyzátorů je Snort[2]. Jedná se o softwarový produkt, který poskytuje funkcionalitu síťového IDS. Pracuje na bázi hledání signatur a je dostupný zdarma pod otevřenou licencí.

Na trhu je několik různých dodavatelů hardwarových řešení pro analýzu paketů. Příkladem jednoho z nich je TippingPoint od společnosti Trend Micro [4].

Jedná se o IPS zařízení (=Intrusion Prevention System), které je funkčně velice podobné IDS systémům s tím rozdílem, že síťová data přes něj přímo

[4] Ještě před rokem byl TippingPoint dostupný v barvách firmy Hewlett-Packard od které jej v říjnu 2015 odkoupila firma Trend Micro.

protékají. Systém analyzuje provoz v reálném čase stejně jako IDS, ale v případě detekce škodlivého síťového provozu jej ihned zablokuje a nepustí do chráněné sítě.

1.3.2 Analýza síťových toků

Analýza síťových toků je odlišný přístup ke zpracování síťových dat. Koncept spočívá v agregování datových toků v síti do tzv. síťových toků neboli flows či někdy nepříliš správně nazývaných NetFlows.

Zařízení, které sleduje provoz na síti a vytváří záznamy o tocích, se nejčastěji označuje jako sonda či exportér. Jejich úkolem je sledovat síťová data a třídit je do jednotlivých toků. Za jeden síťový tok se označují veškerá data, která mají stejné charakteristické parametry [3]. Sonda pro každý síťový tok udržuje informaci o počtech paketů a bajtů, které sítí protekly a po uplynutí určitého časového úseku je agregovaná informace o síťovém toku odeslána na kolektor k archivaci.

Běžně používaná množina těchto parametrů je:

- Síťové rozhraní, ze kterého tok přichází
- Síťový protokol
- Zdrojová IP adresa
- Cílová IP adres
- Zdrojový TCP/UDP port
- Cílový TCP/UDP port
- ToS – tzv. typ síťové služby. Jedná se o parametr ve struktuře datového paketu, který určuje s jakou naléhavostí by se mělo v paketem nakládat.

Exportované síťové toky obsahují všechny výše zmiňované informace spolu s časem počátku a konce síťového toku a hodnotami naměřenými na čítačích. Současné verze protokolů pro přenos těchto agregovaných síťových toků mohou obsahovat i některé další informace. Blíže se používaným protokolům věnuji níže v sekci 1.3.2.1.

Na první pohled to nemusí být zcela zřejmé, ale je vhodné si uvědomit, že exportovaný síťový tok je vždy jednosměrný. Standardní TCP spojení tedy vygeneruje dva samostatné datové toky, které budou mít vzájemně prohození zdrojové a cílové IP adresy a porty.

Kolektor je software, který přijímá síťové toky ze sond a archivuje je. Záznamy se pak typicky ukládají na lokální disk ve stanovených intervalech (např. 5 minut) k pozdějšímu použití.

Uložené datové toky je možné dále zpracovávat a vyhledávat v nich různé druhy síťových útoků a anomálií, účtovat objem zákaznického provozu či zpětně dohledávat informace k bezpečnostním incidentům.

1.3.2.1 Protokoly exportující síťové toky

V současné době existuje mnoho protokolů určených pro přenos informací o síťových tocích.

Společnost Cisco vyvinula vlastní protokol NetFlow [4], který je určen pro export síťových toků z aktivních síťových zařízení (zejména směrovačů). Ačkoliv tento protokol nebyl dlouhou dobu standardizován, mnoho menších výrobců jej také implementovalo do svých zařízení.

Pravděpodobně je stále nejvíce rozšířen ve verzi NetFlow v5, ačkoliv podporuje pouze IPv4 adresy. Protokol byl několikrát revidován až k verzi NetFlow v9 [5], která přinesla podporu pro IPv6 adresy, MPLS a mnoho dalšího. Ačkoliv definici tohoto protokolu můžeme najít v RFC dokumentech, nebyl nikdy přijat jako Internetový Standard.

Někteří výrobci síťových zařízení si vytvořili vlastní formát pro přenos síťových toků a existují tak protokoly jFlow od firmy Juniper Networks, Traffic Flow od firmy Mikrotik či NetStream od společnosti 3Com / HP. Ačkoliv protokoly nemusí být binárně vzájemně kompatibilní, mohou obsahovat principiálně tytéž informace. Vzhledem k rozšířenosti protokolu NetFlow jej však ostatní výrobci většinou podporují také.

V zájmu zvýšení kompatibility zařízení různých výrobců byl standardizován protokol IPFIX [6], který vychází z NetFlow v9. Protokol podporuje definici vlastních šablon a je tak možné v něm přenášegt prakticky libovolná data včetně celých paketů.

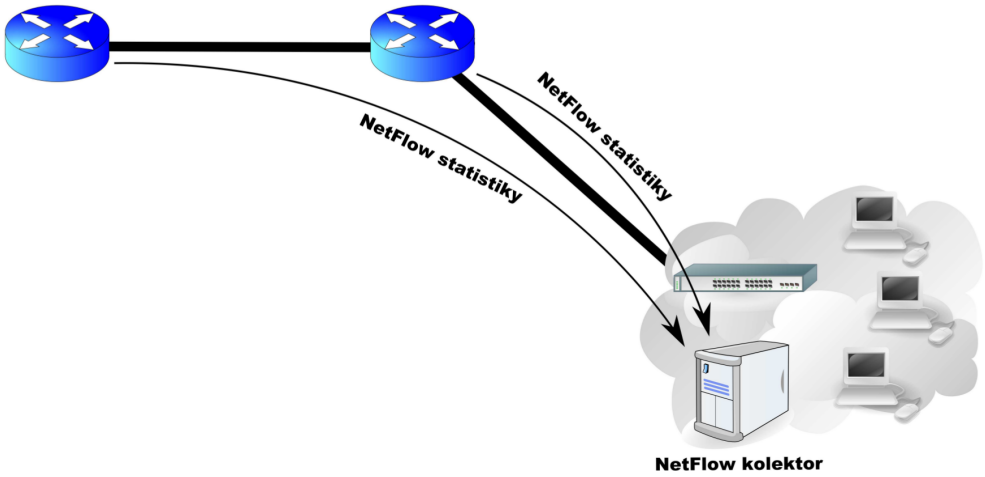
Rád bych ještě zmínil protokol sFlow, který navzdory svému jménu neslouží k přenášení informací o síťových tocích, ale používá se k přenášení L2 síťových rámců. Protokol používá vzorkování, aby bylo možné jej používat i v sítích s vyššími rychlostmi. Směrovače tedy zachytávají prakticelně každý n-tý paket a odesílají jej na kolektor k analýze.

1.3.2.2 Aktivní monitorování

Za aktivní monitorování síťového provozu se označuje takové, kdy aktivní síťové prvky (nejčastěji směrovače) jsou nakonfigurované tak, aby sledovaly a zaznamenávaly agregované informace o síťových tocích a ty následně exportovali na kolektor. Možné použití naznačuje obrázek 1.2.

Směrovače jsou většinou hardwarová zařízení, ale podpora pro sběr síťových toků je obvykle implementována pomocí softwaru, protože jejich hardwarová implementace by mohla zařízení neúměrně prodražit. V důsledku toho směrovače v mnoha případech nemají dostatek výpočetního výkonu, aby mohly sledovat veškerý síťový provoz.

V mnoha případech levných a středně drahých směrovačů se zavádí tzv. vzorkování provozu, což znamená, že do analýzy síťového provozu se zpracovává každý n-tý paket, kde n může nabývat různých hodnot v závislosti na konkrétních možnostech daného zařízení.



Obrázek 1.2: Příklad aktivního sběru síťových toků. Provoz sledují a agregují přímo směrovače.

Směrovače ve většině případů nabízejí export pouze základních informací – počtu přenesených bajtů a paketů. V případě, že bychom chtěli pro analýzu zpracovávat i údaje z aplikačních vrstev, je jedinou možností pasivní monitoring.

1.3.2.3 Pasivní monitorování

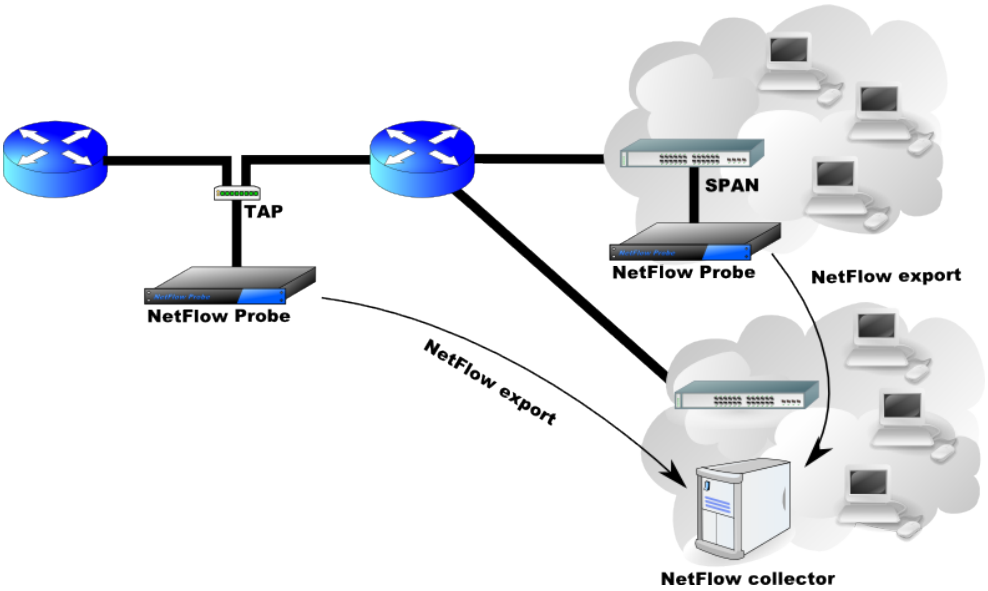
Alternativním přístupem je tzv. pasivní monitoring síťového provozu, kdy zduplikujeme síťový provoz a jeho kopii budeme odesílat do samostatného zařízení – sondy, která má za úkol pouze sběr síťových toků.

Duplikaci provozu můžeme provést dvěma různými způsoby v závislosti na možnostech infrastruktury.

- **Optický rozdělovač**, tzv. splitter, lze použít v případě, že linka, kterou chceme monitorovat, je realizována pomocí optického vlákna. Aktivnímu síťovému prvku pak můžeme předřadit rozdělovač a provoz tak rozdělit mezi směrovač a sondu.

- **Zrcadlení portu**, někdy nazývané též SPAN, je funkce síťového přepínače, která umožňuje zrcadlit definovaný provoz uvnitř přepínače na jedno další síťové rozhraní. K tomuto rozhraní je typicky připojena sonda.

Vzhledem k dostatečnému výkonu dedikovaného zařízení je možné provádět hlubší analýzu síťového provozu a zahrnout do exportovaných síťových toků více informací. Toky se běžně rozšiřují o informace z aplikačních vrstev a mohou tak obsahovat např. volaná telefonní čísla v případě VoIP hovorů,



Obrázek 1.3: Příklad pasivního sběru síťových toků. Síťový provoz je duplikován a předáván sondě, která data agreguje a exportuje.

dotazovaná doménová jména v případě protokolu DNS nebo dotazované URL adresy v HTTP komunikaci.

Nevýhodou tohoto řešení jsou vyšší pořizovací náklady, avšak vzhledem k přidané hodnotě se v dnešní době jedná o preferovanější variantu.

1.3.2.4 Timeout

Monitorovací sonda či aktivní síťový prvek, generující informace o síťových tocích, sleduje datový provoz na síti a zaznamenává napozorované toky spolu s informací o počtech přenesených bajtů a paketů. Pokud k již existujícímu toku zpozoruje další související data, jednoduše je přičte k již existujícím čítačům. V případě, že nebyla zaznamenána žádná data po specifikovanou dobu (např. 30 sekund), je síťový tok odeslán na kolektor ke zpracování a vymazán z lokální paměti sondy. Tuto dobu nazýváme „Neaktivní timeout“.

V síti se vyskytují i toky, které trvají velice dlouhou dobu (např. zálohování dat či navázané VPN spojení), u kterých kdyby sonda čekala na jejich úplné dokončení, docházelo by k velkým zpožděním mezi reálným provozem na síti a odesláním informace o síťovém toku. Z toho důvodu existuje tzv. „Aktivní timeout“, který definuje po jaké době (např. 300 sekund) je ke zpracování odeslán stále probíhající síťový tok. V případě odeslání jsou, stejně jako v předchozím případě, nasbírané čítače daného toku vymazány.

1.3.3 Umístění v síti

V počítačové síti, která má typickou strukturu naznačenou obrázkem 1.1, se můžeme rozhodovat, do které části sítě umístíme monitorovací sondu. Možností je několik[7] a každá má svá pro a proti.

Pro další vysvětlení je třeba se zamyslet, kterým směrem budou proudit data, protože sondu musíme vždy umístit tak, aby měla k datům přístup. Směry datového provozu velice záleží na povaze naší sítě a v zásadě je pouze několik možností:

- Koncové sítě, připojené pomocí přístupového segmentu sítě, komunikují směrem ven ze sítě přes hraniční směrovače. Datový provoz tedy prochází skrz všechny vrstvy sítě a končí mimo ni.

  Tento druh provozu je typický např. pro poskytovatele Internetových služeb.

- Koncové sítě komunikují primárně vzájemně mezi sebou a většina provozu tak zůstává uvnitř spravované sítě. Jen malé procento komunikace směřuje mimo hranice sítě.

  Typickým představitelem mohou být např. akademické či vnitro firemní sítě.

1.3.3.1 V distribuční vrstvě

Při umisťování monitorovací sondy chceme obsáhnout největší možnou množinu datových toků, které se v naší síti vyskytují. Logickým krokem by bylo využití distribuční vrstvy, protože ta obsluhuje komunikaci s okolními sítěmi i tu, která zůstává uvnitř naší sítě.

Distribuční vrstva mívá typicky velké množství, řádově desítky až stovky, propojů zejména s přepínači v přístupové vrstvě, o které nám jde nejvíce. Při takovém množství se již pravděpodobně nevyplatí zavádět pasivní monitorování pomocí dedikovaných sond.

Vzhledem k tomu, že v distribuční vrstvě teče typicky velké množství dat, nemají většinou směrovače dostatečný výkon pro monitorování všech síťových toků a museli bychom stáhnout po vzorkování.

Poslední, však nezanedbatelnou, komplikací by byla duplikace nasbíraných síťových toků v případě komunikace pouze uvnitř naší sítě. Datový tok v takovém případě prochází přes více prvků distribuční vrstvy a některé síťové toky by tak mohly být vyexportovány vícekrát. Deduplikace veškerých síťových toků by byla ve vysokorychlostních sítích velice nákladná, ne-li prakticky nemožná.

1.3.3.2 Na hranici chráněné zóny

Pokud chceme detailněji monitorovat pouze část provozované sítě (např. segment s firemními servery či kancelářskou síť), můžeme umístit dedikované sondy na hranici tohoto segmentu. Typicky bývá takový segment připojen do distribučního jádra naší sítě pomocí několika málo propojů.

Pokud máme této části předřazený firewall nebo směrovač, může být zajímavou variantou využití daného prvku ke sledování síťových toků, pokud je jeho výkon dostačující vzhledem k datovému provozu a vystačíme si se základní analýzou síťových toků.

Hlavní výhodou tohoto nasazení je výrazně nižší počet dat, která je nutno analyzovat. Můžeme si tak dovolit nasadit např. paketovou analýzu nebo výpočetně náročnější detektory na bázi síťových toků.

Potenciální nevýhodou je samozřejmě to, že budeme monitorovat jenom malou část naší sítě.

1.3.3.3 Na hranici sítě

Pokud si můžeme dovolit zanedbat sledování provozu, který neopouští hranice naší sítě, je vhodnou alternativou monitorování na kraji naší sítě, tedy mezi našimi hraničními směrovači a sousedními sítěmi.

Propojů s ostatními sítěmi je většinou relativně malé množství (řádově jednotky) a můžeme si tak dovolit nasadit pasivní monitorování, které nám dává širší možnosti. Sledování síťových toků pomocí směrovačů lze doporučit u malých sítí, které typicky nejsou zapojeny do globálního BGP routování, které je samo o sobě značně náročné na výkon směrovačů.

Hlavní výhodou je, že můžeme analyzovat veškerou komunikaci s okolním světem, protože to je typicky místo, odkud nám hrozí potenciální nebezpečí. Většina počítačových útoků je iniciovaná z prostředí Internetu a cílí na chyby v používaných systémech či špatné bezpečnostní návyky uživatelů (např. slabá hesla).

Kapitola 2

NEMEA

NEMEA je zkratkou za Network Measurements Analysis[8] a představuje především framework určený pro analýzu a zpracování datových toků v počítačových sítích. Narozdíl od tradičního přístupu k datovým tokům se NEMEA orientuje na jejich proudové zpracování. Záznamy tedy nejsou uloženy na disk v blocích a až následně zpracovány, ale jejich zpracování probíhá kontinuálně, což umožňuje rychlejší analýzu dat a dřívější reakci na detekované hrozby.

Monitorovací systém je složen z modulů, které obstarávají pouze jednu konkrétní funkcionalitu. Vzájemné propojení modulů obstarává tzv. TRAP rozhraní, které abstrahuje vývojáře od implementačních detailů komunikace a dovoluje mu soustředit se na vývoj vlastní funkcionality. Zprávy, které systémem proudí, jsou tzv. UniRec záznamy. Ty jsou navržené tak, aby poskytovaly efektivní přístup k informacím v záznamech.

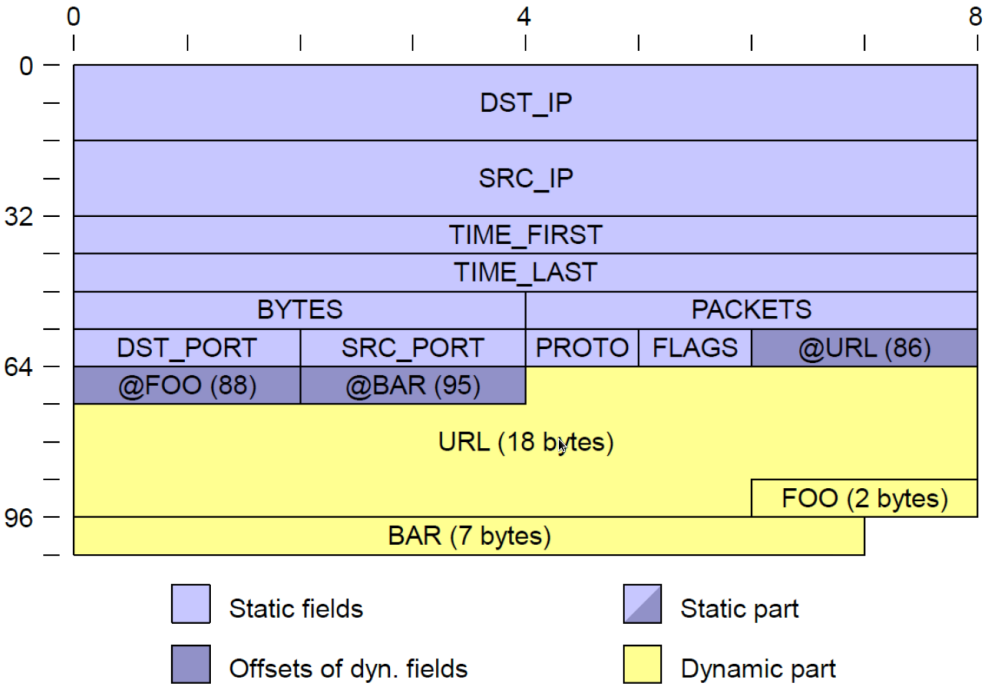
V rámci projektu také vyvíjejí samotné moduly, které je díky jednotnému rozhraní a formátu zprávy možné libovolně uspořádat podle potřeby.

2.1 UniRec záznam

UniRec (=Unified Record) je univerzální a flexibilní formát zpráv, pomocí kterých mezi sebou moduly komunikují. Konkrétní formát je vždy definován tzv. šablonou, která určuje rozdělení zprávy na jednotlivá pole a jejich datový typ. Z těchto polí lze složit prakticky libovolnou zprávu.

Šablona může být definovaná již v době kompilace programu, nebo je možné ji dynamicky vytvářet či měnit po spuštění programu. Definice datových typů obsahuje:

- znaménková i neznaménková celá čísla ve velikosti 8 až 64 bitů
- čísla s plovoucí řádovou čárkou s jednoduchou i dvojitou přesností
- časovou značku s přesností na milisekundy



Obrázek 2.1: Příklad UniRec záznamu, který reprezentuje strukturu síťového toku s několika hodnotami dynamické velikosti [9]

- IP adresu verze 4 či 6
- textový či binární řetězec proměnlivé délky

S UniRec záznamy se pracuje pomocí definovaných maker, což zaručuje dostatečnou přehlednost ve zdrojovém kódu, ale zároveň i efektivní přístup k datovým strukturám záznamu.

Záznam je možné si představit jako plochý seznam hodnot zmíněných datových typů. Do záznamů nelze vkládat vlastní datové typy, ani v nich definovat struktury. Pokud bychom chtěli přidat do záznamu vlastní data, lze k tomu využít textový či binární řetězec.

2.2 TRAP rozhraní

TRAP (=Traffic Analysis Platform) je rozhraní, pomocí kterého mezi sebou moduly komunikují. Abstrahuje uživatele od konkrétních způsobů komunikace a jejich specifického přístupu. Při vývoji modulu tak není třeba myslet na to, zda uživatel bude chtít moduly propojovat pomocí sdílené paměti, UNIX socketů či TCP/IP spojení.

Při navázání spojení mezi moduly je zajištěna výměna UniRec šablony, která dává modulům informaci o struktuře předávaných dat.

Součástí rozhraní je také vyrovnávací vrstva, která agreguje zasílaná data do větších celků, čímž efektivně snižuje režii I/O operací [5] a dovoluje tak dosahovat vyšších propustností modulů.

Programové použití

TRAP rozhraní je úzce svázáno s UniRec šablonami, a proto jejich použití představím společně. Rozhraní zprostředkovává komunikaci modulu s ostatními a zajišťuje vzájemnou výměnu UniRec šablon.

V úvodu je nutné pomocí makra *MODULE_BASIC_INFO()* definovat základní textový popis modulu spolu s počty vstupních a výstupních TRAP rozhraní, a pomocí *MODULE_PARAMS()* dodat výčet parametrů modulu s jejich popisem. TRAP rozhraní zajišťuje výpis nápovědy (pomocí parametru *-h*) a definici rozhraní (parametr *-i*) a proto je nutné mu tyto informace poskytnout ještě před spuštěním programu (funkcí *main()*).

Také je nutné definovat názvy a typy hodnot, se kterými chceme v UniRec záznamech pracovat, pomocí makra *UR_FIELDS()*. Tyto definice jsou uloženy v globálním kontextu programu a makra pro práci s UniRec záznamy jejich definici vyžadují. V případě, že názvy a typy hodnot nejsou známé při kompilaci programu (např. pokud jsou závislé na vstupních parametrech), lze je definovat i v průběhu vykonávání programu, ale je nutné k nim poté přistupovat mírně odlišným způsobem.

O samotnou inicializaci TRAP a parsování přidružených parametrů se postará dvojice maker:

```
INIT_MODULE_INFO_STRUCT(MODULE_BASIC_INFO, MODULE_PARAMS);
TRAP_DEFAULT_INITIALIZATION(argc, argv, *module_info);
```

která má svůj ekvivalent pro uvolnění alokovaných datových struktur, který je vhodné použít před řádným i chybovým ukončením programu:

```
TRAP_DEFAULT_FINALIZATION();
FREE_MODULE_INFO_STRUCT(MODULE_BASIC_INFO, MODULE_PARAMS);
```

Poslední potřebná věc před vlastním spuštěním programu je definice UniRec šablon. Vstupní a výstupní šablony se definují prakticky shodným způsobem, ale jejich přesný význam se mírně liší.

Výstupní šablonu lze vytvořit pomocí funkce:

```
ur_create_output_template(0, "DST_IP,SRC_IP", NULL)
```

kde prvním parametrem je číslo TRAP rozhraní a druhým je řetězec čárkou oddělených hodnot, které budou obsaženy v odesílaných UniRec záznamech.

[5] I/O je zkratka z anglického „Input/Ouput“, tedy vstupně-výstupní. V tomto případě to zahrnuje např. systémová volání pro odesílání a přijímání dat po síti.

Třetí parametr je nepovinný a slouží k navrácení chybového hlášení v případě neúspěchu.

Vstupní šablony se definují obdobným způsobem s tím rozdílem, že druhým argumentem je řetězec povinných hodnot. Jeho zápis je identický, avšak TRAP rozhraní nedovolí navázat spojení s jiným modulem, pokud nabídnutá UniRec šablona neobsahuje všechny vyjmenované hodnoty. Rozdíl je samozřejmě také v názvu funkce:

```
ur_create_input_template(0, "DST_IP,SRC_IP", NULL)
```

Tímto je dokončena veškerá inicializace TRAP rozhraní a modul může začít pracovat. Z důvodu efektivity zpracování je vhodné vyhnout se neustálé dynamické alokaci paměti pro každý odesílaný UniRec záznam a alokovat místo v paměti jednou na počátku a před jeho odesláním obsah vždy přepsat.

```
void *out_rec = ur_create_record(out_tmplt, 0);
```

Pro příjem a čtení dat slouží dvojice maker:

```
int ret = TRAP_RECEIVE(0, in_rec, in_rec_size, in_tmplt);
ur_ipaddr_t = ur_get(in_tmplt, in_rec, F_DST_IP);
```

Obdobně se používají i makra pro zápis a odesílání zpráv:

```
ur_set(out_tmplt, out_rec, F_MY_VALUE, value);
ret = trap_send(0, out_rec, ur_rec_fixlen_size(out_tmplt));
```

Kompletní funkční příklady modulů jsou k nalezení v repozitáři Nemea-framework [6] zveřejněném na stránkách GitHub.com.

2.3 Příklady modulů a typické použití

Moduly lze rozdělit do dvou základních kategorií a mezi *detektory*, které mají na starost analýzu datových toků a detekci specifických typů útoků, a obecně *moduly*, které mají většinou spíše obecnou či podpůrnou funkci jako např. vytváření, filtrování či slučování UniRec toků nebo reportování detekovaných událostí.

Příklady modulů:

- **Flow meter**
  Modul určený k sledování datového provozu a vytváření záznamů o datových tocích. V základní verzi získává záznamy z 2. až 4. vrstvy ISO/OSI modelu, ale je možné jej rozšířit tak, aby zpracovával i data z vyšších vrstev. Existující rozšíření je možné použít např. pro analýzu DNS či HTTP.

[6] https://github.com/CESNET/Nemea-Framework/tree/master/examples

- **NF Reader**
  Modul který načítá data z uložených NetFlow [7] souborů, převádí je na UniRec záznamy, které dále odesílá.

- **UniRec filter**
  Modul pro filtrování záznamů na základě definovaného filtru. Umožňuje filtrovat podle libovolných polí, která jsou v záznamech uložena.

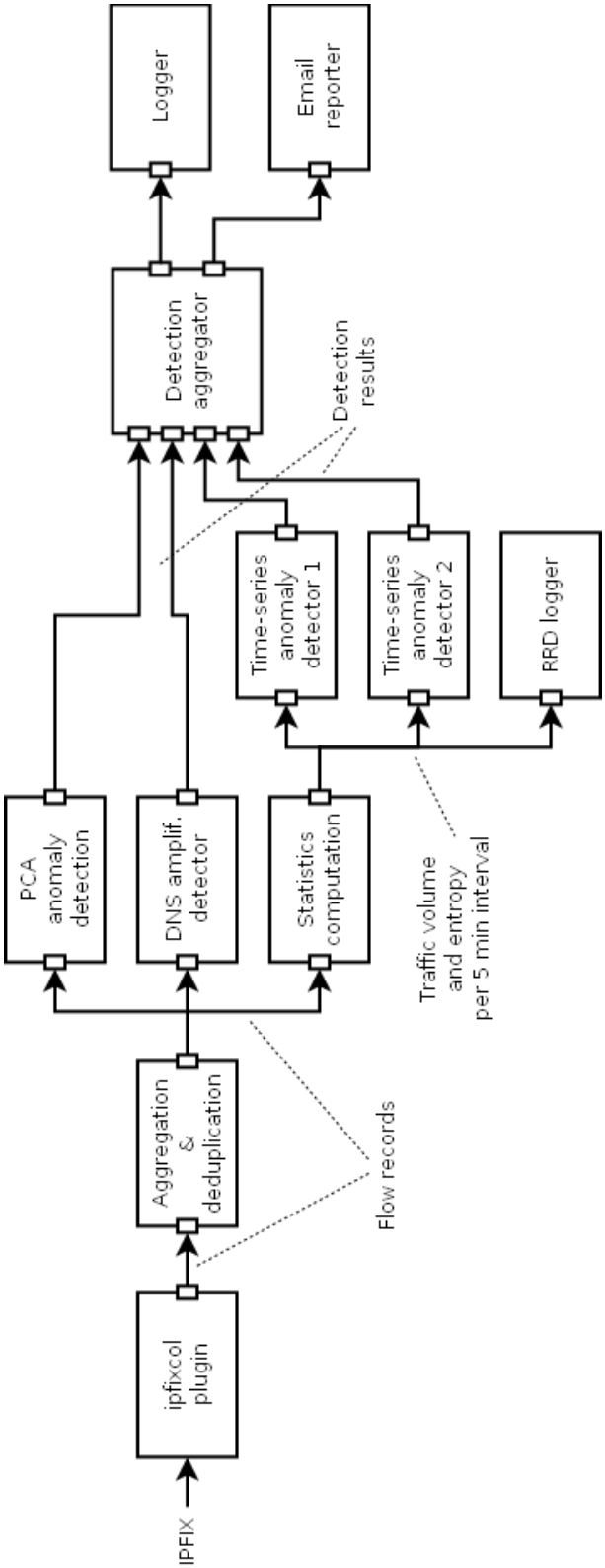
- **Email Reporter**
  Modul určený pro odesílání detekovaných událostí e-mailem na základě definovaných šablon.

Příklady detektorů:

- **SIP BF detector**
  Detektor útoků, které se snaží o prolomení hesel VoIP telefonů komunikujících protokolem SIP.

- **Amplification detector**
  Detektor amplifikačních odrazových útoků, které jsou založené na principu zaslání objemných odpovědí v reakci na malou odpověď. Nejčastěji také v souvislosti s podvržením adresy odesilatele.

[7] NetFlow je Cisco proprietární formát pro ukládání informací o datových tocích s fixně danou množinou atributů, který se postupně stal standardem [5]. Jeho nástupce IPFIX je v mnoha ohledech podobný UniRec záznamům.



Obrázek 2.2: Ukázka možného použití NEMEA systému

Kapitola 3

Návrh rozšíření systému NEMEA

Cílem této práce je rozšířit systém NEMEA o možnost sběru statistických informací z provozu počítačové sítě. Uživateli by měl být ve finále nabídnut flexibilní nástroj, který mu dovolí získávat ze sítě data dle jeho aktuálních potřeb.

3.1 Analýza potřeb

Motivací pro vznik nového modulu je získávání statistických informací o provozu ve sledované počítačové síti. Může nás tedy např. zajímat, kolik dat bylo přeneseno z/do námi sledované sítě nebo jaké byly počty současných aktivních spojení a jejich rozložení v čase.

Modul si nebude klást za úkol cokoliv analyzovat či detekovat, ale bude se zaměřovat pouze na agregování údajů vyskytujících se v UniRec záznamech. Vzhledem k univerzální povaze záznamů by i modul měl být v maximální možné míře univerzální, aby mohl být použit v libovolné části nasazeného NEMEA systému.

Naměřená data není nutné nijak zpracovávat nebo ukládat, postačí je odesílat v podobě UniRec záznamů do navazujícího modulu, který se postará o jejich zpracování či archivaci. Navazující modul je může např. ukládat na disk ve formátu CSV [8] nebo do RRD souborů, InfluxDB [10] či jiné časově orientované databáze.

Vzhledem k tomu, že data mezi NEMEA moduly proudí kontinuálně, je potřeba definovat časový interval, v rámci kterého bude agregační funkce vypočtena. Tento časový interval budu nazývat agregačním intervalem.

[8] Existuje již např. modul Logger, který přijímá libovolná data a ve formátu CSV je vypisuje na standardní výstup

Výpočet bude probíhat průběžně podle toho, jak budou data přicházet na vstupní rozhraní modulu. Ukončení výpočtu jednoho agregačního intervalu nebude založené na aktuálním systémovém čase, ale na informacích uložených v UniRec záznamech a bude možné modul použít i pro zpracování historických záznamů.

3.1.1 Agregace dat

Agregace dat znamená z velké vstupní množiny dat definovaným způsobem vypočíst řádově menší množinu dat, která reprezentuje původní množinu. Výstupem může být i jednoprvková množina.

Způsob redukce vstupní množiny je dán agregační funkcí. Pokud budeme uvažovat data skládající se z více hodnot, jako jsou např. UniRec záznamy, má smysl mluvit i o argumentech funkce, které budou definovat z kterých konkrétních typů hodnot se bude agregace počítat. V závislosti na konkrétní agregační funkci může být argumentů více nebo naopak žádný.

Suma

Suma je agregační funkce, jejíž hodnota vznikne prostým součtem všech zpracovávaných hodnot. Formálně můžeme vyjádřit pomocí zápisu, kde X představuje UniRec záznam a $X[F]$ je zpracovávaná hodnota záznamu.

$$SUM(F) = \sum_{i}^{n} X_i[F]$$

Příklad: `SUM( BYTES )` - součet přenesených bajtů v datových tocích

Průměrná hodnota

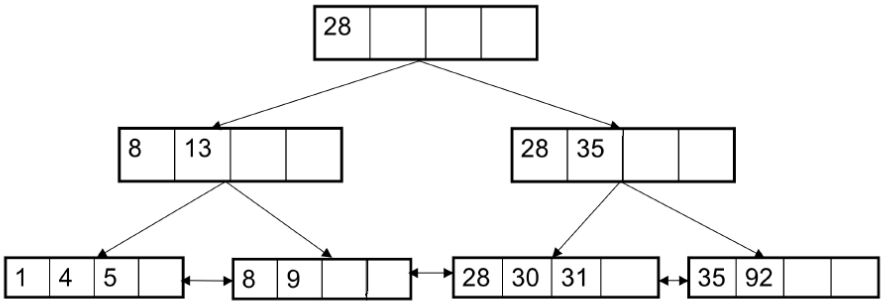
Průměrná hodnota je agregační funkce, jejíž výstupem je součet všech hodnot dělený celkovým počtem zpracovaných záznamů.

$$AVG(F) = \frac{\sum_{i}^{n} X_i[F]}{n}$$

Příklad: `AVG( PACKETS )` - průměrný počet paketů v každém datovém toku

Průměrná změna

Agregační funkce průměrné změny je obdobná jako průměrná hodnota. Vystihuje však průměrný přírůstek hodnoty za jednotku času (např. sekunda). Jedná se tedy o součet hodnot podělený délkou agregačního intervalu.



Obrázek 3.1: Příklad středně zaplněného 4-árního B+ stromu

$$RATE(F) = \frac{\sum_{i}^{n} X_i[F]}{T_{agg}}$$

Příklad: `RATE( BYTES )` - průměrný datový tok v bajtech za sekundu

Počet záznamů

Tato agregační funkce pouze počítá počet záznamů, které projdou zpracováním. Funkce nepotřebuje argument, protože pracuje pouze s informací o záznamu jako celku a ne konkrétní hodnotou.

Příklad: `COUNT()` - počet zpracovaných záznamů

Počet unikátních záznamů

Počítání unikátních záznamů je od prostého počítání záznamů značně odlišné. Vyžaduje minimálně jeden argument, jehož unikátnost je třeba kontrolovat.

U předchozích agregačních funkcí si vystačíme pouze s jedním či dvěma čítači, ale počítání unikátních výskytů bude vyžadovat složitější datovou strukturu.

Pokud by byl modul psán v C++, pravděpodobně by pro tento účel bylo nejvhodnější a nejjednodušší použít *množiny* (tzn. `std::set`), které jsou typicky implementovány pomocí binárních vyhledávacích stromů [11], nebo *neseřazené množiny* (tzn. `std::unordered_set`), které budou nejspíše implementovány pomocí hašovacích tabulek.

Vyvážené binární vyhledávací stromy, jako např. červenočerné stromy [12], jsou binární stromy s upravenými algoritmy pro vkládání a odebírání prvků. Cílem úpravy je udržovat strom vyvážený, tedy aby jedna polovina stromu (či kteréhokoliv podstromu) neměla více než dvojnásobný počet prvků oproti

druhé polovině. Vyvážené stromy mají asymptotickou časovou složitost při vkládání záznamů $O(log\ n)$ a při hledání také $O(log\ n)$. Časová složitost obou případů je shodná v průměrném i nejhorším případě. Paměťová náročnost je $O(n)$.

Hašovací tabulka [13] je pole, které není indexované klíčem, ale hodnotou hašovací funkce vypočtené z klíče. V případě, že více klíčů má shodnou hodnotu hašovací funkce, říkáme, že nastala kolize, a je nutné pro zjištění indexu provést další hašování. S plnící se tabulkou roste pravděpodobnost kolizí a tím se i prodlužuje časová složitost jednotlivých operací.

Nevýhodou hašovacích tabulek je, že v případě naplnění tabulky do určité míry je nutné alokovat větší tabulku a veškerá existující data do ní přesunout, a tedy pro každou hodnotu znovu vypočítat haš. Tato operace je samozřejmě poměrně časově náročná.

Obecně mají hašovací tabulky průměrnou asymptotickou složitost pro vkládání záznamů $O(1)$ a pro jejich hledání také $O(1)$ avšak v nejhorším případě mohou v obě složitosti dosahovat až $O(n)$. Konkrétní složitost závisí především na kvalitě hašovací funkce [14] a aktuálním zaplnění tabulky.

Další datovou strukturou je tzv. B-strom, který je velice podobný binárním stromům s tím rozdílem, že uvnitř uzlu je uloženo d klíčů a každý uzel má až d potomků. Operace nad stromem mají rovněž logaritmickou asymptotickou složitost, ale logaritmus je o základu d narozdíl od *2*.

Obdobou těchto stromů jsou tzv. B+ stromy[15], které používají vnitřní uzly stromu pouze pro průchod stromem. Data jsou ukládána pouze do listů stromu, které navíc obsahují vazbu vždy na sousední listy v řadě. Díky této vazbě jde jednoduše obousměrně iterovat přes všechny vložené záznamy, které jsou z podstaty struktury vždy seřazené. Příklad takového stromu znázorňuje obrázek 3.1.

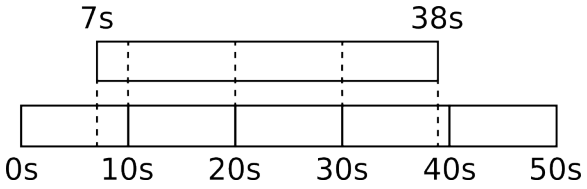
Praktickou výhodou B+ stromů je to, že NEMEA obsahuje již jejich otestovanou implementaci v knihovně *nemea-common* a z toho důvodu jsem se rozhodl ji využít.

Příklad: `COUNT_UNIQ(DST_IP)`- počet unikátních cílových adres

3.1.2 Filtrování záznamů

Pokud by měl modul agregovat vždy všechna data, která dostane na svém vstupu, jednalo by se o velice úzce profilovaný modul. Aby dokázal nabídnout uživateli potřebnou flexibilitu, je potřeba mít k dispozici nástroj na výběr podmnožiny vstupních dat a počítat agregační funkci pouze z této podmnožiny.

Jednotlivé záznamy je tedy potřeba filtrovat a uživatel musí mít možnost srozumitelným způsobem filtr definovat. Měl by mít možnost vytvořit podmínku, která bude zahrnovat libovolné hodnoty ve vstupním UniRec záznamu a bude libovolně složitá.



Obrázek 3.2: Některé síťové toky mohou být delší, než je agregační interval a je nutné je úměrně rozdělit.

Popisovaná funkcionalita je již implementována v modulu *UniRec filter*, ale nelze ji přímo použít v jiném modulu. Nutným úpravám modulu a oddělení samostatné knihovny se věnuji v sekci 4.1.

3.1.3 Rozložení dlouhých toků

Modul bude zaměřen především na zpracování síťových toků, které mohou svou délkou přesáhnout nastavený agregační interval, jak je naznačeno v obrázku 3.2. Je tedy nutné zohlednit zpracování tak, aby vypočtené hodnoty agregačních funkcí popisovaly reálný provoz co možná nejlépe.

Rozložení lze poměrně jednoduše udělat tak, že si pro síťový tok nejprve vypočteme poměrnou část hodnoty odpovídající jednotce času (např. sekundě) a při ukládání hodnoty stanovíme délku průniku toku a aktuálního intervalu, který vynásobíme jednotkovou hodnotou.

Je třeba mít na paměti, že existují i datové toky s nulovým trváním, které jsou typické pro některé síťové útoky jako např. SYN Flood [9].

V případě agregačních funkcí pracujících s počtem, jako jsou *COUNT* a *COUNT_UNIQ* je vhodné přistupovat k přesahujícím síťovým tokům odlišným způsobem. V tomto případě nás totiž spíše zajímá, kolik toků se v danou chvíli vyskytovalo současně, a je tedy třeba daný tok přičíst do všech zúčastněných intervalů.

3.2 Návrh agregačního modulu

Agregační modul bude ke komunikaci s okolím používat TRAP rozhraní a UniRec záznamy, jejichž popisu a použití se věnuje kapitola 2.

Zpracování vstupních dat bude rozděleno do samostatných agregačních pravidel, kde každé bude obsahovat název, podmínku pro filtrování příchozích záznamů a definici agregační funkce. Vnitřní struktura pravidel bude zahrnovat i časově orientovanou databázi, určenou pro průběžný výpočet agregovaných dat, kterou detailněji popisuji níže jako TimeDB v sekci 3.2.3.

[9] SYN Flood je druh síťového útoku, který u oběti udržuje polootevřená TCP spojení a vyčerpává její systémové prostředky s cílem způsobit nedostupnost služby (DoS, Denial-of-Service).

Po připravení všech agregačních pravidel budou inicializována TRAP rozhraní. Na vstupních rozhraních jsou vyžadovanými hodnotami *TIME_FIRST* a *TIME_LAST*, které určují délku síťového toku a jsou nutné ke správnému zařazení v časové databázi.

Šablony na výstupních rozhraních budou sestaveny dynamicky na základě agregačních pravidel. Šablona bude obsahovat hodnotu *TIME*, která definuje začátek agregačního okna zaokrouhleného dle agregačního intervalu. Následovat bude jedna hodnota pro každé agregační pravidlo. Její název je shodný s názvem pravidla a datový typ se bude lišit na základě agregační funkce a věnuji se jim níže v sekci 3.2.2.

S příchodem první zprávy je potřeba inicializovat časové databáze ve všech agregačních pravidlech současně, abychom měli jistotu, že jsou počáteční časy všech databází synchronizované.

Pro každou příchozí zprávu budou následně vyhodnocena veškerá existující agregační pravidla. Nejprve se provede kontrola, zda záznam vyhovuje definovanému filtračnímu pravidlu a pokud ano, uloží se data ze záznamu do agregační databáze.

Při ukládání záznamu může nastat situace, že vkládaný záznam se již nevejde, protože konec síťového toku bude až za hranicí kapacity databáze. V takovém případě je nutné ze všech databází současně vyčíst nejstarší záznamy, odeslat na výstupní rozhraní a zneplatnit je, aby se v databázi uvolnilo místo.

3.2.1 Zápis agregačních pravidel

Agregační pravidlo se bude skládat ze tří částí, z nichž poslední bude nepovinná.

- **Název pravidla**

  První částí definovaného pravidla je jeho název, který bude zároveň také názvem výstupní hodnoty v UniRec záznamech. Z toho důvodu je jeho obsah poměrně striktně omezen a musí začínat malým či velkým písmenem a pak může pokračovat libovolnou kombinací písmen, číslic a pomlčky.

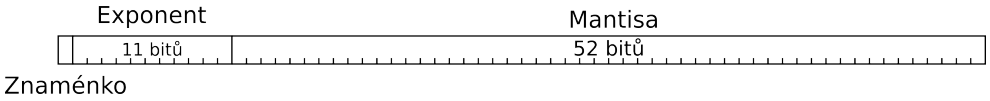
- **Agregační funkce**

  Následující částí je definice agregační funkce, která se skládá z názvu funkce a jejího případného argumentu. Seznam funkcí a styl zápisu je popisován v předchozí kapitole 3.1.1.

  Obecný formát zápisu lze zapsat jako `FUNKCE( ARGUMENT )`

- **Filtrační pravidlo**

  Filtračním pravidlem je řetězec, který vyhovuje parseru knihovny UR Filter. Zjednodušeně řečeno se jedná o logický výraz, jehož jednotlivé části odpovídají formátu `NÁZEV OPERÁTOR HODNOTA`, kde `NÁZEV`



$$Hodnota = (-1)^z \times (1 + \sum_{i=1}^{52} m_{52-i} \times 2^{-i}) \times 2^{e-1023}$$

Obrázek 3.3: Binární struktura plovoucí řádové čárky s dvojitou přesností

je název hodnoty v UniRec záznamu, `OPERÁTOR` je jedním z matematických operátorů[10] a `HODNOTA` je libovolná číselná nebo řetězcová hodnota. Jednotky se mohou vzájemně spojovat pomocí závorek a logických operátorů `||` (= nebo) a `&&` (= a zároveň).

Obsah jednotlivých částí je poměrně striktně omezen implementací UniRec, a proto je možné relativně pohodlně spojit všechny části do jednoho řetězce s použitím jednoznakového oddělovače, jakým je např. dvojtečka – `:`

Jediná možná komplikace, která může nastat je v tom, že filtrační knihovna URFilter umožňuje porovnávat řetězcové hodnoty jak přímo, tak pomocí regulárních výrazů. V takových řetězcích se logicky může vyskytnout libovolný oddělovač, který by mohl narušit parsování celého pravidla. Tento problém lze však elegantně obejít tak, že filtrační pravidlo umístíme na úplný konec řetězce, kde už můžeme s jistotou tvrdit, že veškeré oddělovače jsou součástí definovaného filtru.

Neformálně by pak bylo možné definiční řetězec zapsat jako:

```
NÁZEV : AGREGAČNÍ FUNKCE [: FILTRAČNÍ PRAVIDLO]
```

Formální definice agregačního pravidla je uvedena v sekci 3.2.4, která se zaměřuje na parametry programu.

3.2.2 Výstupní datové typy

Agregační funkci počítající počty prvků, tedy *COUNT()* a *COUNT_UNIQ()* je přiřazený datový typ *uint64*, protože vyjadřuje počet prvků, což je vždy celočíselná hodnota. Všechny ostatní agregační funkce budou pracovat v plovoucí řádové čárce s dvojitou přesností - *double*.

Plovoucí řádovou čárku jsem záměrně zvolil i pro agregační funkci počítající sumu, která je na první pohled celočíselná. Často však budou síťové toky zasahovat do více agregačních intervalů, data se tak mezi ně budou dělit a je pravděpodobné, že budou vznikat desetinná čísla, která by bylo nutné jinak

[10] Matematickými operátory jsou myšleny == resp. =, ! = resp. <>, <, <=, >, >=, které navíc rozšiřují operátory ∼= a =∼ určené pro porovnávání regulárním výrazem.

zaokrouhlit. Datový typ *double* dokáže pojmout bez ztráty přesnosti celá čísla v rozsahu $\pm(2^{53}-1)$ a pokud bychom sčítali přenesená data[11] na plně vytížené síti s kapacitou 1 Tb/s s velikostí agregačního okna 3600 sekund, získáme hodnotu sumy o přibližné velikosti 2^{49}. [12]

Rezervu 3 bitů tedy považuji v tomto případě za dostatečnou. Je třeba si také uvědomit, že po překročení sumární hodnoty 2^{53} nebudou čísla úplně zahozena (jak by se stalo u celočíselného typu), ale dojde pouze k ztrátě nejnižších bitů tedy ke ztrátě přesnosti. Na druhou stranu datový typ s plovoucí řádovou čárkou míří na vyšší univerzálnost modulu, kdy bude možné pracovat i čistě s čísly ryze menšími než 1.

3.2.3 TimeDB

TimeDB ja název kruhové časově orientované datové struktury, která bude sloužit k ukládání mezivýsledků při výpočtu agregačních funkcí. Podporované agregační funkce, které jsou rozepsané v sekci 3.1.1, pracují vždy se součtem agregované hodnoty a počtem sčítaných záznamů, bude tedy dostačující udržovat tyto dva čítače.

Čítač počtu záznamů se liší v případě agregační funkce *COUNT_UNIQ()*, kdy potřebujeme zjistit počet unikátních záznamů. V tomto případě nebude použit celočíselný čítač, ale hodnoty budou vkládány do B+ stromu, který si udržuje informaci o počtu prvků.

Exportované síťové toky přicházejí ke zpracování naprosto neseřazené a při výpočtu agregačních funkcí je nutné tuto neuspořádanost nějakým způsobem kompenzovat. Časová databáze proto bude zahrnovat několik po sobě jdoucích intervalů a příchozí záznamy budou přičítány pouze k čítačům příslušících intervalů. Součet délek všech intervalů vyjadřuje zpoždění databáze a bude konfigurovatelné pomocí parametru.

Pokud tok zasahuje do více intervalů naráz, je ke každému z nich přičtena pouze poměrná část, která do daného intervalu zasahuje, aby nedocházelo k započítávání dat vícekrát. Záznam o síťovém toku neobsahuje žádnou informaci o reálném rozložení provozu v průběhu trvání toku. Pro zjednodušení budu předpokládat rovnoměrné rozdělení, ačkoliv reálná situace může být samozřejmě odlišná zvláště v případě, kdy délka agregačního intervalu je řádově nepoměrná s aktivním timeoutem monitorovací sondy. Možný vnitřní stav databáze naznačuje obrázek 3.4.

[11] Přenesená data jsou pravděpodobně největší možnou číselnou hodnotou síťových toků, které dává smysl sčítat.

[12] Síť s maximální teoretickou rychlostí 1Tb/s přenese za 1 hodinu $3600 * 10^{40}$ bitů. Síťové toky vyjadřují informaci o přenesených bajtech a tedy maximální možná hodnota je $3600 \times 10^{40} \div 8 = 494\ 780\ 232\ 499\ 200$, což odpovídá přibližně hodnotě $2^{48.8}$ a na jeho uložení potřebujeme 49 bitů

Počátek intervalu	Suma	Počet	B+ strom
2016-04-24 12:00:00	100	1	---
2016-04-24 12:00:10	200	1	---
2016-04-24 12:00:20	300	2	---
2016-04-24 12:00:30	140	2	---
2016-04-24 12:00:40	100	1	---
2016-04-24 12:00:50	0	0	---
2016-04-24 12:01:00	0	0	---
2016-04-24 12:01:10	0	0	---
2016-04-24 12:01:20	0	0	---

Síť. tok #1
Od: 12:00:05
Do: 12:00:32
540 bajtů

Síť. tok #2
Od: 12:00:20
Do: 12:00:40
300 bajtů

Pozn.: Samostatné síťové toky (v pravé části obrázku) nejsou součástí databáze a slouží pouze pro vysvětlení situace.

Obrázek 3.4: Příklad TimeDB databáze po vložení dvou síťových toků delších než agregační interval (= 10 sekund).

Přijímaná data

Do databáze bude možné vkládat libovolná data podporovaná v UniRec záznamech, pokud bude dávat agregační funkce smysl. Hlavní podmínkou vkládaných záznamů je, že musí obsahovat hodnoty ohraničující platnost vkládaného záznamu, které jsou klíčové k správné funkčnosti databáze. V případě síťových toků to budou konkrétně hodnoty *TIME_FIRST* a *TIME_LAST*.

Databáze musí odmítnout počítání nelogických agregačních funkcí, jako např. sčítání nečíselných datových typů – znaků nebo textových a binárních řetězců. U těchto typů dává smysl pouze zjišťování počtu záznamů nebo počtu unikátních výskytů.

Vyčítání dat z databáze

Databáze musí odmítnout vložit záznam, který svou délkou bude přesahovat poslední interval v databázi. V takovém případě je vyžadováno, aby byla vyčtena data z nejstaršího intervalu v databázi a jeho místo mohlo být uvolněno pro alokování nového intervalu a mohla být vkládána další data.

Každé agregační pravidlo může počítat jinou agregační funkci a bude mít svoji vlastní instanci časové databáze, která nebude nijak závislá na ostatních. Je však nanejvýš vhodné, aby vyčítání dat probíhalo synchronně ze všech databází současně, aby se předešlo možnému rozsychronizování.

3.2.4 Vstupní parametry

Výsledný agregační modul bude samostatná aplikace, využívající funkce a knihovny NEMEA systému. Možností, jak předat modulu konfiguraci při

startu, je několik a nejjednodušší je pravděpodobně zpracování argumentů z příkazové řádky. Jedná se o běžný způsob, který je používán i ostatními moduly.

Druhým možným způsobem je vytvoření vlastního konfiguračního souboru, který bude obsahovat celou konfiguraci. Při spuštění modulu by stačilo pouze předat cestu ke konfiguračnímu souboru, nebo jej umístit do předem definovaného místa na disku. Vzhledem k tomu, že NEMEA nemá žádný jednotný způsob zápisu konfiguračních souborů, nezdá se mi vhodné, aby každý modul měl svojí vlastní konfigurační syntaxi.

Hlavní výhodou konfiguračních souborů je jejich přehlednost a možnost komentářů. Je nutné jej použít až v případě, kdy hrozí reálné nebezpečí toho, že program by nebylo možné spustit pouze s parametry příkazové řádky. Operační systém omezuje maximální délku argumentů, které mohou být předány programu. Ačkoliv POSIX [13] stanovuje tuto délku na 4kB [16], většina dnešních operačních systémů je mnohem velkorysejší [14] (řádově stovky kB až jednotky MB). Nemyslím si tedy, že maximální délka argumentů by znamenala reálné omezení v použitelnosti modulu.

Popíšu nyní jednotlivé parametry modulu. Konkrétním příkladům praktického použití modulu se věnuje kapitola 6.

Délka agregačního intervalu

Parametr: `-t [celé číslo]`

Argument určuje délku agregačního intervalu, tedy dobu, po kterou bude počítána agregační funkce, než dojde k jejímu resetování.

Zpoždění databáze

Parametr: `-d [celé číslo]`

Argument určuje zpoždění ve zpracování vstupních záznamů a výpočtu agregačních funkcí. Reálně to tedy znamená, že příchozí záznam bude započítán, ale na výstup bude agregovaná hodnota odeslána až po uplynutí tohoto zpoždění plus jedna délka agregačního intervalu.

Účel tohoto argumentu je objasněn v kapitole 3.2.3.

Hodnota argumentu by neměla být menší než je aktivní a pasivní timeout sond, které sbírají síťové toky.

[13] POSIX je zkratkou za Portable Operating System Interface, což je rodina standardů, která se snaží o maximální kompatibilitu napříč různými operačními systémy.

[14] Konkrétní hodnotu lze v Linuxovém prostředí zjistit příkazem `getconf ARG_MAX` a konkrétně na aktuálním systému Debian 8 je maximální hodnota 2MB.

Maximální doba neaktivity

Parametr: `-I [celé číslo]`

Argument definuje dobu, po kterou když nepřijdou žádná data budou zneplatněna veškerá nasbíraná data a vnitřní stavy čítačů budou znovu inicializovány.

Účelem tohoto argumentu je vyhnout se výpisu mnoha nulových hodnot v případě, že by došlo k přerušení dodávky záznamů o síťových tocích. Mohlo by se tak mylně zdát, že žádná data sítí neprotékala.

Agregační pravidlo

Parametr: `-r [řetězec]`

Parametrem je řetězec definující jedno agregační pravidlo. Formát pravidla lze definovat pomocí Backus-Naurovy formy a regulárních výrazů následovně:

```
<pravidlo> ::= název : agregační-funkce
             | název : agregační-funkce : filtrační-pravidlo

název               ~  /^[a-zA-Z][a-zA-Z0-9_]*$/
agregační-funkce    ~  /^[a-zA-Z]+\(([a-zA-Z][a-zA-Z0-9_]*)?\)$/
filtrační-pravidlo  ~  /^[a-zA-Z0-9_=<>~.() ]*$/
```

Každé definované agregační pravidlo vytváří jednu shodně pojmenovanou hodnotu na výstupním rozhraní. Veškeré bílé znaky jsou z názvu a agregační funkce před jejich dalším zpracováním odstraněny.

Následující výstupní TRAP rozhraní

Parametr: `-R`

Tento parametr nemá žádný argument a plní pouze funkci oddělovače jednotlivých výstupních rozhraní. Dává tedy modulu vědět, že byla dokončena konfigurace jednoho rozhraní a následující agregační pravidla se týkají dalšího výstupu.

3.2.5 Vektorový výstup agregace

Součástí původního plánu vývoje agregačního modulu byla i myšlenka, aby výstupem jednoho agregačního pravidla nebyla pouze jedna skalární hodnota, ale výstupem byl vektor hodnot splňujících určité podmínky. Vektor by neobsahoval všechny zpracované hodnoty, ale pouze prvních několik seřazených podle definované agregační funkce.

Příkladem možného použití by bylo např. získání 10 zdrojových adres, které v daném intervalu přenesly největší množství dat do segmentu sítě, který by byl definován pomocí filtračního pravidla.

Jádro modulu

Zamýšlený modul by měl funkční jádro prakticky shodné s tím, který je předmětem této práce. Po spuštění a inicializaci TRAP rozhraní by byly zpracovány vstupní argumenty a inicializovány vstupní a výstupní UniRec šablony podobně jak je uvedeno sekci 2.2 popisující programové použití TRAP rozhraní.

Rozdílný by byl především formát agregačních pravidel, protože obsažená agregační funkce by musela přijímat nové argumenty – minimálně informaci o řazení a počtu záznamů k výpisu.

Hlavní cyklus programu by mohl zůstat prakticky totožný. Přijatý záznam bude vyhodnocen na definované filtrační pravidlo a v případě, že vyhoví, bude vložen do obdobné časově orientované databáze.

Odlišnosti TimeDB

Hlavní odlišnost implementace by spočívala v odlišném použití časově orientované databáze TimeDB.

Ve zpracovávaném modulu jsou součástí intervalů v kruhové databázi hodnoty čítačů, což umožňuje k nim přistupovat ve velmi krátkém čase.

Zamýšlený modul by však všechny agregované hodnoty musel ukládat do interní struktury [15] spolu s hodnotou argumentu funkce. Předpokládám však, že použitá struktura bude přistupovat k datům efektivně a i když bude modul pravděpodobně dosahovat nižších výkonů, stále bude vhodný pro použití ve vysokorychlostních sítích. Záležet bude primárně na konkrétní konfiguraci pravidel.

Při vyčítání dat z databáze by bylo pravděpodobně nutné vyčíst z databáze všechna nasbíraná data. Řadit prvky by nemuselo být teoreticky nutné při využití faktu, že chceme jen určitý počet největších záznamů. Pole by stačilo projít pouze jednou a při konstantním počtu hledaných prvků by asymptotická složitost byla $O(n)$.

Výstupní formát

Výstupní formát a šablony jsou otázkou, která prozatím nezůstala nijak rozhodnuta. V zásadě se nabízejí dva odlišné přístupy, z nichž ani jeden se podle mě nedá označit za ideální.

[15] S největší pravděpodobností by tato struktura byla implementována opět pomocí B+ stromu. Jsou pro tento účel vyhodné, protože nabízejí zaručený čas přístupu $O(log\ n)$ a je možné je jednoduše proiterovat při výpisu.

První z nich navrhuje ve výstupní šabloně alokovat pro každé definované pravidlo počet polí odpovídající dvojnásobku počtu požadovaného počtu prvků. Např. pro zmíněných 10 zdrojových adres s nejvyšším podílem přenosu by bylo třeba použít celkem 20 polí v záznamu – vždy jeden pro zdrojovou adresu a druhý pro počet přenesených dat.

V tomto přístupu spatřuji dva různé potenciální problémy:

- Modul by měl podporovat agregaci dle libovolné hodnoty obsažené v UniRec záznamu včetně těch, které mají variabilní délku (např. URL adresy nebo DNS dotazy). Mohlo by se poměrně jednoduše stát, že v případě výskytu dlouhých řetězců by mohlo dojít až k překročení maximální velikosti UniRec záznamu, která je 64kB. Zvláště pak v případě, že na jedno výstupní rozhraní bude definováno více pravidel.
- Nepříjemností by také mohl být obrovský počet různých hodnot v jednom UniRec záznamu. Pravděpodobně by nedošlo k vyvolání chybového stavu některého z navázaných pluginů, ale obecně práce a ladění nemusí být zrovna nejpohodlnější.

Druhý z možných přístupů k výstupnímu formátu by znamenal, že při výpisu dat z jednoho agregačního intervalu by byl na výstup odeslán jeden UniRec záznam za každé agregační pravidlo. Takový záznam by pak obsahoval časovou známku, název agregačního pravidla a opět dvojnásobný počet hodnot než je požadovaný počet agregovaných záznamů.

Výhodou tohoto přístupu je vzájemné oddělení agregačních pravidel, což by umožňovalo jednodušší zpracování v navazujícím modulu.

Hlavní nevýhodu tohoto přístupu spatřuji v nejednoznačném ukončení bloku UniRec záznamů vygenerovaných pro jeden agregační interval. Přijímající modul by buď musel vědět, kolik má pro každé pravidlo očekávat záznamů, což by znamenalo nutnost duplicitní konfigurace. Nebo by musel existovat speciální UniRec záznam, který by oznamoval konec bloku zpráv.

Takový speciální záznam by byl pravděpodobně pouze proprietární komunikací mezi dvěma konkrétními moduly, což by šlo proti myšlence TRAP rozhraní. Bohužel aktuálně TRAP rozhraní ani UniRec záznamy neobsahují žádný mechanismus podobný např. příznakům v protokolu TCP.

Návrh a implementace

Po dohodě s vedoucím práce však bylo rozhodnuto, že popisovaná funkcionalita by neměla být přímou součástí agregačního modulu, kterému se věnuje zbytek této práce, protože výstupy obou modulů jsou vzájemně velice odlišné. Z toho důvodu zde není konkrétní návrh ani implementace daného modulu zahrnuta.

Kapitola 4

Implementace

V této kapitole bych rád popsal své zásahy do existujících zdrojových kódů systému včetně implementace nového agregačního modulu.

4.1 Úprava filtrovacího modulu

Pro účely agregačního modulu potřebuji funkcionalitu pro filtrování UniRec záznamů na základě uživatelem definovaných pravidel. Filtrační pravidla mohou obsahovat libovolně složitý logický výraz. Pro tuto funkci již v systému NEMEA existuje modul jménem Unirecfilter.

Bohužel filtrační mechanismus aktuálně existuje pouze jako samostatný modul a není možné jej bez úprav použít na jiném místě. Modul funguje tak, že ze zadaného filtračního pravidla nejprve sestaví abstraktní syntaktický strom a následně pro každý UniRec záznam prochází strom a vyhodnocuje filtrační pravidlo.

4.1.1 Aktuální řešení

K vytvoření syntaktického stromu je použita dvojice programů Flex a Bison [16] s vlastní definovanou gramatikou v souborech *scanner.l* a *parser.y*, a sadou funkcí v souboru *functions.c*.

Veškeré funkce, které jsou pro použití filtrování potřebné, jsou obsaženy právě v definované sadě funkcí a mohli bychom je označit ja veřejné rozhraní filtrovací části programu, ačkoliv to není nikde přímo zmíněno. Hlavičkový soubor k funkcím chybí, ale mohl by vypadat např. takto:

```
int evalAST(struct ast *ast, const ur_template_t *in_tmplt, const void *in_rec);
struct ast *getTree(const char *str, const char *port_number);
```

[16] Flex a Bison [17] je dvojice programů používaná k automatizovanému zpracování textů. Nabízejí možnost jednoduchou formou definovat syntaktická pravidla a z nich následně vygenerovat zdrojový kód v C, který definovaný text zpracovává požadovaným způsobem.

Funkce *evalAST()* se používá k vyhodnocení, zda UniRec záznam vyhovuje definovanému filtru. Jako argumenty přijímá syntaktický strom filtru, šablonu UniRec záznamu a vlastní záznam.

Funkce *getTree()* slouží k převodu filtru z řetězce na abstraktní syntaktický strom, který je nutný pro vyhodnocování filtrační podmínky.

Ačkoliv to zde není na první pohled zřejmé, funkce vyžaduje, aby v globálním kontextu byly již definované UniRec šablony obsahující položky, které se mohou vyskytnout ve filtrační podmínce. V důsledku to znamená, že nelze zkompilovat filtr dříve, než budeme znát formát filtrovaných záznamů.

Bez znalosti globální definice šablon by např. nebylo možné ověřit existenci klíčových slov ve filtru, ani zanést do syntaktického stromu informaci, jaké datové typy bude třeba v rámci vyhodnocování filtru porovnávat.

4.1.2 Vlastní úpravy

Veškerá funcionalita filtrování je tedy již v samostatném souboru *functions.c*, ale názvy funkcí nejsou příliš vhodné. Nechtěl jsem příliš zasahovat do existujícího kódu, a proto jsem se rozhodl vytvořit jeden zdrojový soubor, který bude navenek vystupovat pod jednotným názvem a interně bude používat existující funkce *getTree()* a *evalAST()*.

Název knihovny jsem zvolil **urfilter** jako zkratku pro „UniRec filter“ a vytvořil jsem funkce:

- `urfilter_create(const char *filter_str)`
  Připraví filtrační pravidlo, avšak nebude ještě zkompilované. Je možné jej ručně zkompilovat pomocí funkce `urfilter_compile()`, nebo se filtr zkompiluje při prvním volání `urfilter_match()`.

- `urfilter_compile(urfilter_t *unirec_filter)`
  Zkompiluje připravené filtrační pravidlo.

- `urfilter_match(urfilter_t *unirec_filter, const ur_template_t *template, const void * record)`
  Funkce nejprve ověří, zda je pravidlo již zkompilované a případně jej zkompiluje. Ověří zda UniRec záznam *record*, definovaný UniRec šablonou *template*, vyhovuje zkompilovanému filtračnímu pravidlu.

- `urfilter_destroy(urfilter_t *object)`
  Uvolní veškeré alokované datové struktury v paměti.

4.2 Implementace agregačního modulu

Na základě výše uvedeného návrhu jsem implementoval agregační modul jako samostatnou aplikaci s využitím vytvoření knihovny UR Filter a knihovny Crypto z balíku OpenSSL.

Modul je implementován, stejně jako většina NEMEA modulů, v jazyce C a začleněn do struktur projektu Nemea-modules. Vzhledem k tomu, že celý projekt je sestavován pomocí autotools, vytvořil jsem pro modul vlastní složku `modules/aggregator` spolu se souborem `Makefile.am` a odkaz na něj přidal do `modules/configure.ac` a `modules/Makefile.am`. Tím je modul začleněn do Nemea-modules a bude sestaven spolu s celým projektem.

Modul závisí na knihovně UR Filter, která závisí na dvojici programů Bison a Flex. Agregátor je tedy tranzitivně závislý na zmíněných programech a bez jejich přítomnosti nebude modul sestaven.

4.2.1 Implementace TimeDB

V rámci modulu jsem vyvinul kruhovou časově orientovanou databázi pro ukládání síťových toků. Její implementaci jsem umístil do samostatných souborů `aggregator/timedb.h` a `aggregator/timedb.c`, aby bylo možné je případně použít i pro jiný účel. Databázi zatím není zamýšleno využívat z jiného modulu a proto není sestavována jako sdílená knihovna.

Databáze byla implementována s ohledem na její výkon. Interní struktura je shodná s navrhovanou na obrázku 3.4. Kruhová databáze je implementována statickým polem, které je alokováno jednou při vytváření databáze a později již nedochází k realokacím. Vyjímku mohou tvořit B+ stromy, pokud je databáze použita k počítání unikátních výskytů.

Použití databáze

Při práci s databází se používá objekt `timedb_t`, který ji reprezentuje a je vyžadován prakticky všemi funkcemi (s vyjímkou vytváření databáze). Její veřejné rozhraní je následující:

```
timedb_t * timedb_create(int step, int delay,
    int inactive_timeout, int count_uniq);

void timedb_init(timedb_t *timedb, time_t first);

int timedb_save_data(timedb_t *timedb, ur_time_t urfirst,
    ur_time_t urlast, ur_field_type_t value_type, void * value,
    int var_value_size);

void timedb_roll_db(timedb_t *timedb, time_t *time, double *sum,
    uint32_t *count);

void timedb_free(timedb_t *timedb);
```

K vytvoření a zrušení databáze slouží dvojice funkcí `timedb_create()` a `timedb_free()`. Pokud je žádoucí, aby databáze počítala pouze počet uni-

kátní výskytů vkládaných záznamů, je nutné jako čtvrtý argument vložit libovolné přirozené číslo (doporučená je však vkládat číslo jedna).

Databázi je možné inicializovat funkcí `timedb_init()`, ale pokud tak není učiněno dojde k její inicializaci s prvním vloženým záznamem. Databáze je poté připravena tak, aby první vkládaný záznam byl na jejím začátku. Časové značky jednotlivých intervalů v databázi jsou vždy zaokrouhleny na celé násobky agregačního intervalu.

Vkládání dat

Záznamy se do databáze vkládají pomocí funkce `timedb_save_data()`, která přijímá odkaz na libovolnou hodnotu UniRec záznamu a informaci o jejím typu. V případě, že se jedná o typ s variabilní délkou, je nutné specifikovat i její délku, protože databáze nemá k dispozici UniRec šablonu, aby si délku dokázala zjistit sama.

Pokud databázové struktury nejsou inicializované, provede se nejprve inicializace dle časové známky právě vloženého síťového toku.

Datové typy `UR_TYPE_IP`, `UR_TYPE_CHAR`, `UR_TYPE_STRING` a `UR_TYPE_BYTES` lze vkládat pouze pokud je databáze v režimu počítání unikátních záznamů a data jsou tedy ukládána do B+ stromu.

V případě, že vkládáte do databáze záznamy s variabilní délkou, tedy textový nebo bitový řetězec, nejsou ve vnitřním B+ stromu uloženy celé řetězce, ale pouze jejich MD5 otisk. Výpočet otisku je vyřešen pomocí externí knihovny Crypto v balíku OpenSSL. Ačkoliv je hašovací funkce MD5 již dlouhou dobu nedoporučovaná pro kryptografické použití, zde je cílem pouze zkrátit řetězec na fixní délku a předejít kolizím. Tuto funkci jsem zvolil především s ohledem na rychlost jejího zpracování.

Při vkládání dat do databáze je bezpodmínečně nutné kontrolovat návratovou hodnotu funkce, která může být rovna jedné z následujících konstant:

- `TIMEDB_SAVE_ERROR` - Chyba databáze, buď je vkládán neznámý nebo nepovolený typ UniRec záznamu, nebo není dostatek místa v paměti k alokování listu B+ stromu
- `TIMEDB_SAVE_OK` - Uložení proběhlo v pořádku.
- `TIMEDB_SAVE_NEED_ROLLOUT` - Databáze je plná a je nutné z ní vyčíst nejstarší agregovaná data, aby mohlo být uvolněno místo pro další záznamy.

  V případě této návratové hodnoty záznamy nebyli uloženy a je tedy nutné po uvolnění místa vložení opakovat.
- `TIMEDB_SAVE_FLOW_TRUNCATED` - Vložený záznam byl uložen, ale počátek síťového toku je dříve, než dokáže databáze pojmout a došlo tedy ke ztrátě některých dat.

Po inicializaci databáze je běžné, že databáze vrací tento návratový kód, protože síťové toky přicházejí neuspořádaně. Je tedy velice pravděpodobné, že databáze nebude inicializována tokem s nejmenší časovou známkou.

Pokud k těmto chybám dochází v průběhu běhu programu, může to znamenat, že zvolená velikost časové databáze (=její zpoždění) je příliš malé, nebo některé exportéry síťových toků mají nesynchronizované systémové hodiny.

Vyčítání informací z databáze

Čítače se z databáze získávají pomocí funkce `timedb_roll_db()`, která zapíše nasčítané hodnoty do proměnných jejichž odkazy jsou předány jako argumenty. Při čtení je přečtený interval z databáze odstraněn a jeho místo je připraveno pro další použití.

Doporučuji vyčítat hodnoty z databáze až v případě, že vkládání nového záznamu selže z důvodu zaplnění databáze. V opačném případě by se mohlo stát, že z databáze uvolníte časové intervaly, které by ještě využili vkládané záznamy.

4.2.2 Implementace vlastního modulu

Modul má právě jedno vstupní rozhraní a alespoň jedno výstupních rozhraní.

Po startu se zinicializuje TRAP rozhraní a zpracuje všechny argumenty předané z příkazové řádky.

Ke zpracování agregačních pravidel jsem původně plánoval použít funkci `strtok()`, ale narazil jsem na komplikaci s tím, že pokud má vstupní řetězec na začátku oddělovač, funkce jej přeskočí namísto toho, aby vrátila odkaz na prázdný řetězec. Zpracování textového řetězce jsem tedy implementoval svépomocí.

Při načtení pravidla jsou provedeny základní kontroly a připraveny potřebné interní datové struktury – UR Filter a TimeDB.

Řetězec filtru není v tento moment nijak validován, protože v tuto dobu ještě není navázané žádné spojení na vstupní TRAP rozhraní a není známá UniRec šablona. Nelze tedy zkontrolovat správnost názvů uvedených ve filtru. K validaci a zkompilování filtru dojde až po navázání spojení a pokud je ve filtru zanesena chyba, chod programu se ukončí s chybovým hlášením.

Toto chování bohužel nelze nijak jinak vyřešit, protože po startu modul nemá žádné informace o očekávaných vstupních formátech.

Jakmile jsou všechny argumenty příkazové řádky zpracovány, modul zkontroluje, zda je počet výstupních rozhraní definovaný v argumentu TRAP knihovny (parametr `-i`) roven počtu výstupních rozhraní definovaných agregačními pravidly (parametr `-R`).

Modul nyní přejde do čtecí smyčky a čeká na přijetí UniRec záznamu. S prvním přijatým záznamem jsou inicializovány hromadně všechny kruhové databáze, aby se zajistila jejich synchronnost.

Přijatý záznam je zpracován postupně všemi definovanými agregačními pravidly – ověřen vůči filtračnímu pravidlu a případně vložen do kruhové databáze. Pokud vkládání záznamu selže z důvodu zaplnění databáze, jsou postupně ze všech kruhových databází získána nejstarší data a odeslána na výstup. V případě, že je aktivní verbose režim modulu (parametr `-v`) jsou výstupní hodnoty zároveň vypisovány na standardní výstup.

Modul je ukončen podle konvence NEMEA modulů v případě, že přijme speciální UniRec záznam, označující konec datového toku. Tento speciální záznam slouží výhradne pro testovací účely a při běžném nasazení se nikde nepoužívá.

Kapitola 5

Testování

V této kapitole bych se rád věnoval výkonnostním testům agregačního modulu.

Testy budou probíhat v uměle vytvořeném prostředí nezatíženého systému. Modul bude jediná součást NEMEA systému, která poběží a budu se při testování snažit o minimalizaci externích vlivů.

Testovací stroj je notebook Dell Inspirion 13z Touch osazený čtyřjádrovým procesorem Intel® Core™ i5-4210U, operační pamětí o velikosti 8 GB a diskem Samsung SSD 850 EVO.

Testy budou prováděny v sadách po pěti a jejich výsledky zprůměrovány.

5.1 Příprava testovacího prostředí

K testům bude použita sada reálných dat ze sítě společnosti Casablanca INT s.r.o.. Jedná se o tranzitní provoz z nočních hodin. Charakteristické hodnoty získané programem `nfdump` jsou shrnuta v tabulce 5.1.

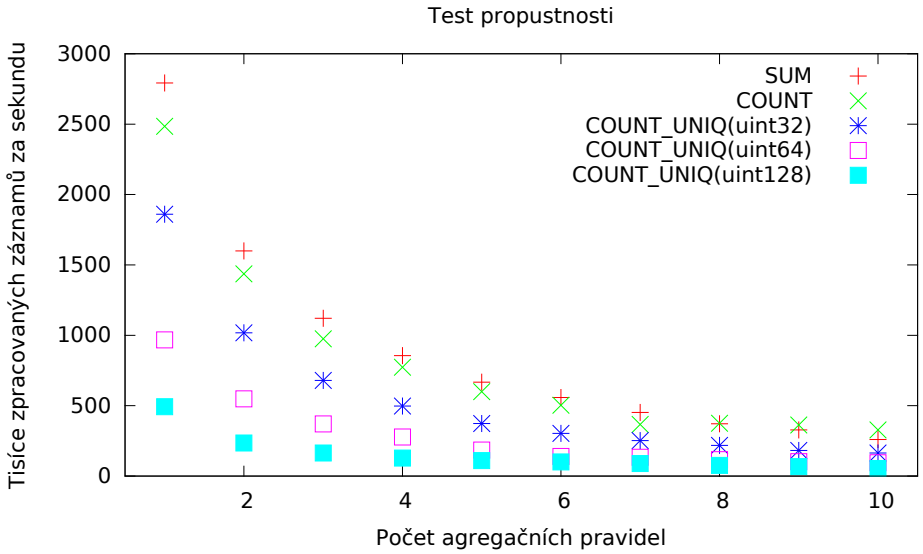
Testovací data jsou pomocí modulu *nfdump_reader* převedena na tok UniRec záznamů a uložena do soubotu `testdata.unirec`.

```
$ ./nfdump_reader -i "f:testdata.unirec" ./nfcapd.201603210435
```

Datový soubor bude umístěn v ramdisku, aby se minimalizoval možný negativní vliv při načítání dat z disku. Výstupní rozhraní modulu bude opět

Počet toků:	2 248 285
Součet bajtů:	31 045 667 896
Součet paketů:	54 027 692
Bitů za sekundu:	396 809 968
Paketů za sekundu:	86 319
Bajtů na paket:	574

Tabulka 5.1: Parametry testovacích síťových toků.



Obrázek 5.1: Výsledky testování propustnosti agregačního modulu.

nasměrováno do souboru `agg_out.unirec` umísteném opět v ramdisku. Po testu bude soubor přečten pomocí modulu *logger* a naměřená data budou ověřena.

Časy budou měřeny pomocí linuxového nástroje `date +%s.%N` vždy bezprostředně před a po experiementu a následně odečteny. Každá instance bude změřena 5krát a výsledná propoustnost zprůměrována.

5.2 Testovací scénář

Cílem testování je zjistit propustnost modulu při výpočtu agregační funkce. Aby nedocházelo ke zkreslování výsledků, bude vždy testu podrobena pouze jedna konkrétní agregační funkce s různým počtem agregačních pravidel.

Vzorový testovací příkaz pro výpočet sumy s dvěma pravidly:

```
./aggregator -i "f:/dev/shm/testdata.unirec,b:10001" -t 10 -d 90 -r "sum_1:
    SUM(BYTES)" -r "sum_2: SUM(BYTES)";
```

K testování výpočtu unikátních záznamů použiji různě velké hodnoty klíčů, abych zjistil jak moc je propustnost závislá na velikosti použitého klíče.

5.3 Výsledky testování

Graf 5.1 a tabulka 5.2 zobrazují výsledky testování propustnosti agregačního modulu. Vertikální grafu osa ukazuje počet zpracovaných záznamů za sekundu

Pravidel	SUM()	COUNT()	UNIQ(32b)	UNIQ(64b)	UNIQ(128b)
1	2792	2484	1859	967	492
2	1599	1436	1017	548	234
3	1120	974	679	370	163
4	855	773	497	278	127
5	667	599	373	184	109
6	558	502	303	138	100
7	452	366	251	133	89
8	371	375	217	116	75
9	328	363	182	102	66
10	259	328	163	94	53

Tabulka 5.2: Tabulka naměřené propustnosti modulu. Uvedené hodnoty vyjadřují propustnost v tisících tocích za sekundu.

v tisících, horizontální osa naopak počet současně vypočítávaných agregačních funkcí.

Z výsledků je patrné, že výpočet sumy či počtu prvků je poměrně jednoduchá operace a modul tak dosahuje poměrně dobré propustnosti i v případě, že je definovaných pravidel několik. Výpočet unikátního počtu záznamů, implementovaný pomocí B+ stromu, je naopak silně závislý na velikosti klíče.

Kapitola 6

Příklady použití

V této kapitole bych rád naznačil některé z možných příkladů použití tohoto modulu v praxi.

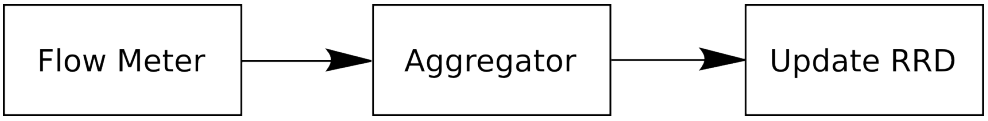
Vzorový scénář bude zahrnovat pouze potřebné moduly pro funkčnost příkladu. Předpokládám, že reálně nasazená struktura bude obsahovat i další NEMEA moduly a detektory, ale pro přehlednost jsem je do příkladů nezahrnoval.

V příkladech budu používat modul *Flow Meter*, který naslouchá na síťovém rozhraní počítače a odesílá síťové toky ve formě UniRec záznamů, a modul *Update RRD*, který přijímá číselná data s časovou známkou a zaznamenává je do lokální RRD databáze. Modul *Update RRD* v současné době není součástí NEMEA systému.

6.1 Účtování zákaznického provozu

Předpokládejme, že jsme středně velký poskytovatel Internetových služeb a chceme účtovat nadlimitní síťový provoz. V rámci smluvního vztahu se zákazníkem máme ujednáno, že prvních 200 GB provozu je zdarma a přenesená data navíc mu budou účtována.

Když už data budeme účtovat na měsíční bázi, rádi bychom zákazníkům nabídli grafický přehled využití smluveného limitu. Svým zákazníkům přidělujeme staticky privátní IP adresy a na jejich základě také chceme provádět účtování.



Obrázek 6.1: Příklad možného nasazení modulu s jedním výstupem.

Na našem serveru, který bude připojen pomocí optického rozdělovače mezi našimi hraničními směrovači a Internetem, spustíme modul *Flow Meter* a nastavíme jej, aby naslouchal na lokálním TCP rozhraní na portu 10000. Na stejném serveru spustíme i modul *Update RRD*, který se bude připojovat na lokální TCP port 10001.

Nyní musíme spustit a správně nastavit agregační modul. Chceme aby se svým vstupním rozhraním připojil k *Flow meteru* na portu 10000 a dále posílal agregovaná data modulu *Update RRD* na portu 10001. Návaznost jednotlivých modulů zobrazuje obrázek 6.1;

Délku agregačního intervalu zvolíme 60 sekund a zpoždění databáze 300 sekund [17],

Celkem máme zatím 3 zákazníky, kteří mají přiřazené adresy `10.0.0.10`, `10.0.0.20` a `10.0.0.30`.

Agregační modul spustíme pomocí parametrů:

```
./aggregator -i "t:localhost:10000,t:10001" \
  -t 60 \
  -d 300 \
  -r "cust_1_in_bytes : SUM(BYTES) : DST_IP = 10.0.0.10" \
  -r "cust_1_in_packets : SUM(PACKETS) : DST_IP = 10.0.0.10" \
  -r "cust_1_out_bytes : SUM(BYTES) : SRC_IP = 10.0.0.10" \
  -r "cust_1_out_packets : SUM(PACKETS) : SRC_IP = 10.0.0.10" \
  -r "cust_2_in_bytes : SUM(BYTES) : DST_IP = 10.0.0.20" \
  -r "cust_2_in_packets : SUM(PACKETS) : DST_IP = 10.0.0.20" \
  -r "cust_2_out_bytes : SUM(BYTES) : SRC_IP = 10.0.0.20" \
  -r "cust_2_out_packets : SUM(PACKETS) : SRC_IP = 10.0.0.20" \
  -r "cust_3_in_bytes : SUM(BYTES) : DST_IP = 10.0.0.30" \
  -r "cust_3_in_packets : SUM(PACKETS) : DST_IP = 10.0.0.30" \
  -r "cust_3_out_bytes : SUM(BYTES) : SRC_IP = 10.0.0.30" \
  -r "cust_3_out_packets : SUM(PACKETS) : SRC_IP = 10.0.0.30"
```

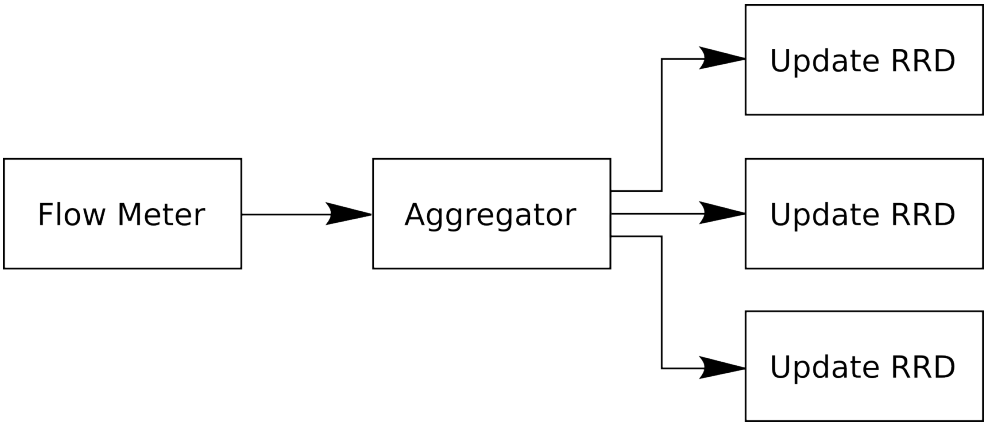
RRD soubory, které modul vytvoří, budeme moci použít k vykreslování grafů pro zákazníky a zároveň pro měsíční vyúčtování provozu.

6.2 Vizualizace provozu chráněné sítě

Předpokládejme, že jsme firma poskytující připojení k Internetu pro domácnosti a zároveň nabízíme naše vlastní e-mailové řešení a webhosting. Chtěli bychom mít přehled o struktuře síťového provozu, který teče na naše servery. Provoz naší kanceláře ani zákaznických přípojek nás v tuto chvíli příliš nezajímá.

Konkrétně by nás zajímalo rozdělení druhů poštovního provozu mezi SMTP, POP3 a IMAP. Rádi bychom znali také rozložení provozu mezi protokoly TCP, UDP a ICMP, protože neočekáváme že by naše servery komunikovali

[17] Nastavené zpožděné databáze by mělo být minimálně stejně velké jako největší z nastavených timeoutů Flow Meteru. Jeho výchozí nastavení je 300 sekund.



Obrázek 6.2: Příklad možného nasazení modulu se třemi výstupy.

jinak než pomocí TCP. A jako poslední bychom rádi znali podíl IPv4 a IPv6 provozu v naší síti.

Připojíme si tedy server k přepínači u našich serverů a nakonfigurujeme zrcadlení veškerého provozu [18] přepínače do našeho serveru. Moduly budeme chtít vzájemně navázat podle obrázku 6.2. Na serveru spustíme modul *Flow meter* a nastavíme jej, aby naslouchal na lokálním TCP portu 10000. Dále na serveru spustíme tři instance modulu *Update RRD* na TCP portech 10001, 10002 a 10003.

Délku agregačního intervalu modulu si opět zvolíme 60 sekund a zpoždění databáze 300 sekund,

Zbývá již jen spustit agregační modul, který bude data agregovat:

```
./aggregator -i "t:localhost:10000,t:10001,t:10002,t:10002" \
   -t 60 \
   -d 300 \
   -r "SMTP_B : SUM(BYTES) : DST_PORT = 25 || SRC_PORT = 25" \
   -r "SMTP_P : SUM(PACKETS) : DST_PORT = 25 || SRC_PORT = 25" \
   -r "SUBM_B : SUM(BYTES) : DST_PORT = 587 || SRC_PORT = 587" \
   -r "SUBM_P : SUM(PACKETS) : DST_PORT = 587 || SRC_PORT = 587" \
   -r "POP3_B : SUM(BYTES) : DST_PORT = 110 || SRC_PORT = 110" \
   -r "POP3_P : SUM(PACKETS) : DST_PORT = 110 || SRC_PORT = 110" \
   -r "IMAP_B : SUM(BYTES) : DST_PORT = 220 || SRC_PORT = 220" \
   -r "IMAP_P : SUM(PACKETS) : DST_PORT = 220 || SRC_PORT = 220" \
   -R \
   -r "TCP_B : SUM(BYTES) : PROTOCOL = TCP" \
   -r "TCP_P : SUM(PACKETS) : PROTOCOL = TCP" \
   -r "UDP_B : SUM(BYTES) : PROTOCOL = UDP" \
   -r "UDP_P : SUM(PACKETS) : PROTOCOL = UDP" \
   -r "ICMP_B : SUM(BYTES) : PROTOCOL = ICMP" \
   -r "ICMP_P : SUM(PACKETS) : PROTOCOL = ICMP" \
```

[18] Tedy konfiguraci tzv. SPAN portu.

```
-R \
-r "IPv4_B : SUM(BYTES) : SRC_IP <= 255.255.255.255" \
-r "IPv4_P : SUM(PACKETS) : SRC_IP <= 255.255.255.255" \
-r "IPv6_B : SUM(BYTES) : SRC_IP > 255.255.255.255" \
-r "IPv6_P : SUM(PACKETS) : SRC_IP > 255.255.255.255"
```

Trojice modulů *Update RRD* nám bude vytvářet soubory, ze kterých můžeme jednoduše vykreslit přehledné grafy.

Závěr

Mým hlavním cílem v této práci byl návrh a implementace univerzálního agregačního modulu pomocí NEMEA Frameworku s ohledem na maximální možnou výkonnost výsledného modulu.

Při vývoji modulu jsem se zaměřoval především na agregaci síťových toků, ale modul je dostatečně univerzální na to, aby zvládal agregovat libovolné UniRec záznamy, které budou obsahovat hodnoty *TIME_FIRST* a *TIME_LAST*.

V rámci této práce na agregačním modulu jsem zároveň i přispěl do projektu oddělením filtrační logiky UniRec filtru do samostatné knihovny, kterou bude nyní možné použít i v jiných modulech, pokud se to někde bude hodit.

Při práci s UniRec filtrem jsem narazil na několik nedostatků, které mi ve filtračním mechanismu chyběly. Modul nedovoluje mezi sebou vzájemně porovnávat dvě hodnoty UniRec záznamu. V případě filtrování síťového provozu to pravděpodobně nemá příliš význam, ale pokud má být modul univerzální, tak myslím že by bylo vhodné toto v budoucnu rozšířit. Obdobná situace je s porovnáváním časových údajů, které filtr aktuálně nepodporuje, protože filtrovat toky na základě přesně určeného času jejich výskytu nedává příliš smysl.

Co mi však u filtrování vyloženě chybí je nemožnost jednoduchého odlišení IPv4 a IPv6 záznamů, které je aktuálně možné obejít pouze díky znalosti používané vnitřní struktury pomocí podmínky $DST\_IP > 255.255.255.255$, která není vůbec intuitivní.

Myslím, že by mohlo být zajímavé agregační modul rozvinout přidáním podpory pro výstupní matematické výrazy s agregační funkcí. Myslím si, že se určitě najdou použití, kdy bude potřeba naměřenou hodnotu vynásobit vhodnou konstantou. Pokud by měla být přidána i možnost do výrazu umístit více agregačních funkcí najednou, bylo by pravděpodobně vhodné přepracovat větší část programu, aby se předešlo vícenásobným výpočtům stejných hodnot.

Vhodným pokračováním ve vývoji by byla také implementace modulu s vektorovým výstupem, který jsem popisoval v sekci 3.2.5.

Literatura

[1] Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata. [cit. 2016-05-05]. Dostupné z: http://people.clarkson.edu/~jmatthew/publications/SPIE_SnortSuricata_2013.pdf

[2] Snort. Org. [cit. 2016-05-05]. Dostupné z: https://www.snort.org

[3] sFlow vs IPFIX. [cit. 2016-05-05]. Dostupné z: https://www.plixer.com/blog/sflow/sflow-vs-ipfix/

[4] Cisco IOS NetFlow. [cit. 2016-05-05]. Dostupné z: http://www.cisco.com/web/go/netflow

[5] Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), Říjen 2004. Dostupné z: http://www.ietf.org/rfc/rfc3954.txt

[6] Claise, B.; Trammell, B.; Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011 (INTERNET STANDARD), Září 2013. Dostupné z: http://www.ietf.org/rfc/rfc7011.txt

[7] Santos, O.: Network Security with Netflow and IPFIX: Big Data Analytics for Information Security. Cisco Press, 2015.

[8] Network Measurement Analysis. [cit. 2016-04-26]. Dostupné z: https://www.liberouter.org/technologies/nemea/

[9] Nemea: Framework for stream-wise analysis of network traffic. CESNET Technical report 6/2013. [cit. 2016-04-26]. Dostupné z: https://www.liberouter.org/technologies/nemea/

[10] The Platform for Time-Series Data. [cit. 2016-05-05]. Dostupné z: https://influxdata.com/

[11] set - C++ Reference. [cit. 2016-05-05]. Dostupné z: http://www.cplusplus.com/reference/set/set/

[12] Red-Black Trees. [cit. 2016-05-05]. Dostupné z: https://www.cs.auckland.ac.nz/software/AlgAnim/red_black.html

[13] Data Structures and Algorithms: Hash Tables. [cit. 2016-05-05]. Dostupné z: https://www.cs.auckland.ac.nz/software/AlgAnim/hash_tables.html

[14] Hash Table. [cit. 2016-05-05]. Dostupné z: http://occcwiki.org/index.php/Hash_Tables

[15] CSci 340: B+-trees. [cit. 2016-05-05]. Dostupné z: http://www.cburch.com/cs/340/reading/btree/

[16] sysconf(3) - Linux manual page. [cit. 2016-05-05]. Dostupné z: http://man7.org/linux/man-pages/man3/sysconf.3.html

[17] What are Flex and Bison. [cit. 2016-05-05]. Dostupné z: http://aquamentus.com/flex_bison.html

[18] Zdrojové kódy NEMEA. [cit. 2016-04-26]. Dostupné z: https://github.com/CESNET/Nemea

Příloha A

Instalační a uživatelská příručka

A.1 Sestavení a instalace

Modul je součástí celého projektu NEMEA [18] a je tedy doporučeno jeho spustitelnou podobu sestavit spolu s celým projektem pomocí doporučené sady příkazů uveřejněné spolu se zdrojovými kódy.

Nejprve je nutné stáhnout rekurzivně repozitář.

`git clone --recursive https://github.com/CESNET/Nemea`

V případě, že bude existovat v repozitíři složka `Nemea/modules/aggregator`, pak tento krok můžete přeskočit. Zkopírujte složky `src/aggregator` a `src/unirecfilter` z přiloženého CD do adresáře `Nemea/modules`.

Vstupte do adresáře `Nemea` a spusťte následující sadu příkazů:

```
./bootstrap.sh
./configure --prefix=/usr --bindir=/usr/bin/nemea \
    --sysconfdir=/etc/nemea --libdir=/usr/lib64
make
```

Nyní byste měli mít zkompilovaný kompletní NEMEA framework se všemi aktuálními moduly včetně agregačního, který je součástí této práce. Jednotlivé moduly jsou k nalezení ve složkách `modules` a `detectors`.

V případě, že chcete zkompilované moduly nainstalovat do systému, stačí jen provést

```
sudo make install
```

A.2 Uživatelská příručka

Použití modulu je poměrně jednoduché a značně přímočaré.

Nejprve je třeba si stanovit délku agregačního intervalu, který je shodný pro celý běžící modul. Tato hodnota definuje jak často bude modul odesílat naměřené hodnoty na výstup.

Druhá důležitá hodnota je délka zpoždění modulu, která by měla být násobkem agregačního intervalu. Tato hodnota je přibližná a může být v případě potřeby automaticky upravena na vhodnou hodnotu (např. násobek agregačního intervalu). Hodnota zpoždění by měla být minimálně tak velká, jako je nejvyšší hodnota timeoutu všech exportérů.

Detailní popis jednotlivých argumentů a jejich význam je uveden v sekci 3.2.4. Zde uvedu jen zkrácený přehled argumentů:

- **-i <definice TRAP rozhraní>** Definice TRAP rozhraní se skládá ze rozhraní oddělených čárkou. Parametry rozhraní jsou odděleny dvojtečkou.

  Možné parametry:

  - `t:localhost:10000` - definice vstupního TCP rozhraní, modul se připojí na port 10000
  - `t:10000` - definice výstupního TCP rozhraní, modul bude naslouchat na připojení klienta
  - `b:` - výstupní rozhraní blackhole, veškeré výstupy jsou okamžitě zahozeny
  - `u:mujsocket` - vstupní či výstupní definice Unix socketu
  - `f:/tmp/file.unirec` - vstupní či výstupní definice souboru

- **-t <agregační interval>** Celočíselná hodnota definující délku agregačního intervalu.

- **-d <zpoždění databáze>** Celočíselná hodnota určující zpoždění časové databáze

- **-r <agregační pravidlo>** Definice agregačního pravidla. Každé pravidlo odpovídá jedné hodnotě na výstupu a skládá se ze tří částí oddělených dvojtečkou. Poslední část není povinná.

  - `NÁZEV` - určitě název výstupního sloupce. Název se nesmí krýt s některým z již existujících názvů (např. na vstupním rozhraní)
  - `FUNKCE(ARGUMENT)` - definuje jakým způsobem a z jaké hodnoty se bude počítat agregační funkce

  - `FILTRAČNÍ PRAVIDLO` - definuje filtrační pravidlo. Všechny příchozí záznamy před začazením do výpočtu zkontrolovány, zda vyhovují tomuto pravidlu.

    K filtrování je použita knihovna UR Filter vyčleněná z modulu UniRec filter. Zběžný popis syntaxe je k dispozici níže.

- **-R** Parametr bez argumentu. Následující agregační pravidla se budou týkat dalšího TRAP rozhraní v řadě.

  Počet TRAP rozhraní definovaných parametrem `-i` musí odpovídat počtu určeným parametry `-R`.

Syntaxe pravidel UR Filtru je definována v dokumentaci UniRec filtru a je k dispozici on-line na adrese:

https://github.com/CESNET/Nemea-Modules/tree/master/unirecfilter

Zevrubně je možné ji popsat jako libovolně uzávorkovaný a libovolně složitý logický výraz, který se skládá z částí, které se zapisují jako `NÁZEV OPERÁTOR HODNOTA`.

Příklad pravidla: `DST_IP = 192.168.1.12 && DST_PORT`

Příloha B

Seznam použitých zkratek

NEMEA Network Measurements Analysis

UniRec Unified Record

TRAP Traffic Analysis Platform

VoIP Voice over Internet Protocol

ISO/OSI Internation Standardization Organization / Open Systems Interconnection

BGP Border Gateway Protocol

IDS Intrusion Detection System

IPS Intrusion Prevention System

ToS Type of Service

RFC Request For Comments

SPAN Source Port Analyzer

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `aggregator` ..... implementace modulu a TimeDB
  - `unirefcilter` ..... upravený filtrovací modul
    - `lib` ..... vyčleněná knihovna pro filtrování UniRec záznamů
    - `Makefile.am` .....
    - `unirecfilter.h` .....
    - `unirecfilter.c` ..... upravená implementace filtračního modulu
  - `thesis`
    - `thesis.tex` ..... zdrojová forma práce ve formátu LaTeX
    - `my.bib` ..... bibliografická data zdrojů práce
    - `rfc.bib` .... souhrnná bibliografická data ke všem existujícím RFC
    - `*.pdf` ..... vložená grafika ve formátu PDF
    - `*.png` ..... vložená grafika ve formátu PNG
    - `*.svg` ..... zdrojová vektorová grafika ve formátu SVG
    - `measurements.csv` ..... naměřené testovací hodnoty
    - `graph.gp` ..... Gnuplot skript pro vykreslení grafu
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF