ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| **Název:** | Rozšíření systému NEMEA pro nasazení v distribuovaném prostředí |
| **Student:** | Bc. Marek Švepeš |
| **Vedoucí:** | Ing. Tomáš Čejka |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Systémové programování |
| **Katedra:** | Katedra teoretické informatiky |
| **Platnost zadání:** | Do konce zimního semestru 2017/18 |

## Pokyny pro vypracování

Prostudujte systém NEMEA [1,2] pro detekci a analýzu síťového provozu.
Prostudujte existující systémy a přístupy paralelního zpracování síťových toků z vysokorychlostních sítí.
Navrhněte úpravy a rozšíření systému NEMEA umožňující zpracování velkého množství síťových toků pomocí paralelního zpracování.
Návrh musí obsahovat analýzu možného dopadu rozdělování zátěže mezi výpočetní uzly na výsledky detekce a zpracování.
Implementujte SW modul v jazyce C pro rozdělování síťových toků na výpočetní uzly.
Experimentálně ověřte vlastnosti rozšířeného systému.

## Seznam odborné literatury

[1] https.//github.com/CESNET/Nemea
[2] Bartoš, Václav, Martin Žádník, and Tomáš Čejka. *Nemea: Framework for Stream-Wise Analysis of Network Traffic.* Technical Report, CESNET, a.l.e., 2013.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 26. září 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Diplomová práce

# Rozšíření systému NEMEA pro nasazení v distribuovaném prostředí

***Bc. Marek Švepeš***

Vedoucí práce: Ing. Tomáš Čejka

21. dubna 2017

# Poděkování

Rád bych poděkoval vedoucímu práce za cenné rady a odborné vedení. Dále děkuji sdružení CESNET, z.s.p.o. za možnost pracovat na tomto projektu. Závěrem děkuji mé rodině a přítelkyni za podporu během celého mého studia na FIT ČVUT.

# Prohlášení

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

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

V Praze dne 21. dubna 2017 .....

### Odkaz na tuto práci

Švepeš, Marek. *Rozšíření systému NEMEA pro nasazení v distribuovaném prostředí.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

# Abstrakt

Množství dat, které musí v dnešní době monitorovací systémy zpracovávat, roste kvůli zvyšující se rychlosti počítačových sítí a přibývajícímu počtu připojených zařízení. Kvůli různým typům bezpečnostních hrozeb musí být většinou data zpracována mnoha detekčními algoritmy najednou s omezenými výpočetními prostředky. Pro zpracování velkého množství síťových dat se začíná používat paralelizace a vznikají škálovatelná řešení monitorovacích systémů. V rámci této diplomové práce byl pro účely výzkumu v oblasti paralelizace zpracování použit modulární open-source systém NEMEA. Tato práce se zabývá návrhem, realizací a testováním rozšíření systému NEMEA, které umožní distribuované nasazení systému, což díky paralelnímu zpracování řádově zvýší propustnost celého systému. Zvolený způsob paralelizace spočívá v rozdělování proudu síťových toků mezi výpočetní uzly. Navržené řešení je realizováno v podobě NEMEA modulu. Práce dále popisuje vytvořené testovací prostředí, ve kterém byly provedeny experimenty k ověření vlastností nového modulu. Všechny experimenty byly provedeny s reálnými daty z akademické sítě CESNET2. V závěru práce je představena škálovatelná architektura paralelního zpracování pomocí systému NEMEA.

**Klíčová slova** Paralelní zpracování, detekce anomálií, NEMEA

# Abstract

As the speed and the size of computer networks grow, monitoring systems have to process increasing volume of data. Moreover, because of various kinds of security threats, all data must be processed by many detection methods at the same time with limited computational resources. Therefore, it is necessary to develop scalable solutions of monitoring systems allowing for parallel processing huge volume of data. This thesis uses modular and open-source system NEMEA for research in the field of parallel processing. The thesis deals with design, implementation and testing of NEMEA system extension, that allows for deployment in distributed environment. The extension increases throughput of the system using parallel processing. The paralelization is done by splitting a stream of flow records among computational nodes. Working prototype is implemented as a NEMEA module. The thesis further describes a testing environment used for experiments verifying the characteristics of the working prototype. All experiments were performed using real data traces from NREN CESNET2. The thesis is concluded by introducing a scalable architecture of parallel processing using NEMEA system.

**Keywords** Parallel processing, anomaly detection, NEMEA

# Obsah

# Seznam obrázků

# Seznam tabulek

# Úvod

S důležitostí počítačových sítí v každodenním životě roste i důležitost monitorovacích systémů. Monitorovací systém dává správci či operátorovi počítačové sítě přehled o stavu síťových prvků, provozu v síti a umožňuje mu informace o provozu zachytit. Tyto informace mohou být poté použity například k účtování služeb, optimalizaci infrastruktury sítě, ale také k bezpečnostní analýze, jejímž cílem je detekování a nahlášení škodlivého provozu. Jedním ze systémů pro analýzu síťového provozu a detekci anomálií je modulární systém NEMEA vyvíjený v rámci sdružení CESNET z.s.p.o.

Postupným zrychlováním a zvětšováním počítačových sítí roste i objem dat, který musí monitorovací systémy zpracovat. Tento problém pomáhá řešit zpracování síťových toků namísto paketů. Síťový tok obsahuje agregované informace z hlaviček paketů a snižuje se tak objem dat ke zpracování. Vedle toho útočníci používají nové typy útoků vyžadující nové detekční algoritmy, které musejí celý objem dat zpracovávat zároveň. Ve výsledku to často znamená, že detekční systém běžící na jednom výpočetním stroji musí postupně zpracovávat více a více dat mnoha detekčními algoritmy najednou. Tento stav nelze udržet a na vysokorychlostních sítích to takto nelze provozovat už nyní.

Možným řešením problémů s nárůstem objemu dat a množstvím detekčních algoritmů je paralelní zpracování v distribuovaném prostředí. Pro paralelní zpracování se nabízí dvě základní možnosti. První možností je paralelizace jednotlivých detekčních algoritmů, která by zvýšila jejich propustnost. To však u některých částí algoritmů není možné a některé algoritmy nelze paralelizovat vůbec. Druhou možností, jak docílit paralelního zpracování, je rozdělení dat ke zpracování na několik částí a ty distribuovat odděleným instancím detekčních algoritmů běžících na více výpočetních uzlech.

Tato práce se zabývá analýzou a návrhem rozšíření systému NEMEA, které umožňuje distribuované nasazení systému a paralelní zpracování velkého množství dat. Systém NEMEA je vhodný pro výzkum v oblasti paralelního zpracování z důvodu modularity, která umožňuje přirozené distribuování na více výpočetních uzlů. Systém je dostupný jako open-source projekt, což

umožňuje provádět potřebné změny. Rozšíření systému umožňující paralelní zpracování představuje nový prvek `Flow scatter`, který rozděluje proud síťových toků mezi výpočetní uzly pomocí různých metod rozhazování. Navržené rozšíření je realizováno v podobě NEMEA modulu.

Práce se dále zabývá vytvořením a konfigurací testovacího prostředí, které simuluje distribuované nasazení NEMEA systému. V tomto prostředí jsou experimentálně ověřeny vlastnosti realizovaného prototypu: rovnoměrnost rozdělení dat, rychlost rozdělení dat a hlavně dopad rozdělení dat na výsledky detekce událostí. Všechny experimenty byly provedeny s reálnými daty zachycenými v akademické síti CESNET2. Práce v závěru představuje architekturu libovolně škálovatelného systému NEMEA.

Kapitola 1

# Analýza

## 1.1 Systém detekce průniku

Systém detekce průniku (Intrusion detection system – IDS) je zařízení nebo program, který hledá škodlivé chování v počítačové síti nebo počítači. Nalezené události obvykle reportuje administrátorovi, zapíše do databáze nebo pošle do systému pro nahlášené události, který události dál zpracovává a vizualizuje. Jde tedy o pasivní monitorování, což znamená, že nezasahuje do provozu v síti nebo počítači. Detekované škodlivé chování je nahlášeno a aktivní opatření je poté na administrátorovi sítě nebo počítače.

Hlavní typy IDS jsou *network IDS (NIDS)* a *host-based IDS (HIDS)*. HIDS analyzuje příchozí a odchozí provoz na jednom stroji a někdy také chrání klíčové systémové soubory daného stroje. Oproti tomu NIDS zpracovává síťový provoz z mnoha zařízení. Obvykle jde o hardware nebo software, který zahrnuje monitorovací sondy rozmístěné v počítačové síti a data procházející přes tyto sondy jsou analyzována.

Se zaměřením na NIDS můžeme dále rozlišovat podle metody detekce *signature-based* a *anomaly-based* systémy. Signature-based přístup využívá definovanou množinu vzorů škodlivého chování a snaží se je v provozu nalézt. Výhodou tohoto přístupu je jistota nalezení známých hrozeb, ale nevýhodou je, že neznámé útoky zůstávají nedetekovány. Anomaly-based přístup hledá v síťovém provozu odchylky oproti „normálnímu“ (běžnému) provozu pomocí statistických metod a strojového učení. Výhodou tohoto přístupu je možné nalezení i nových dosud neznámých útoků. Občas ale může být i legitimní provoz klasifikován jako anomálie a tyto falešné poplachy (tzv. *false-positives*) je nutné vyhodnocovat a detekční metody upravit.

Důležitou charakteristikou systému je i typ zpracovávaných dat. Pokud systém analyzuje celé pakety (včetně obsahu datové části), jedná se o tzv. *packet-based* přístup, který je obvyklý u metod detekce založených na hledání vzorů. Kvůli objemu dat, která musí monitorovací systémy zpracovat, se ale v dnešní době častěji pracuje se síťovými toky (tzv. *flow-based* přístup). Síťový

tok obsahuje agregované informace z hlaviček paketů, konkrétně zdrojovou a cílovou IP adresu, zdrojový a cílový port, protokol, počet přenesených bajtů a paketů a také časové značky. Časové značky vymezují interval komunikace, tj. první a poslední přenesený paket daného toku. V praxi se většinou používá v reálném čase analýza síťových toků a pokud je třeba, může se nad zachycenými daty dodatečně spustit off-line analýza celých paketů (např. analýza *pcap* souborů) pro důkladnější analýzu.

## 1.2 NEMEA systém

NEMEA [1] (**Ne**twork **Me**asurements **A**nalysis) je modulární systém pro analýzu síťového provozu a detekci anomálií. Modularita systému spočívá v rozdělení komplexního úkolu zpracování síťových dat na menší a jednoduší části – filtrování provozu, detekce útoku, reportování události a další. Moduly jsou implementovány jako samostatné procesy, které spolu komunikují prostřednictvím komunikačních rozhraní. Zprávy, které si mezi sebou vyměňují, mohou obsahovat například právě zpracovávaná data, statistiky nebo nahlášenou událost (alert).

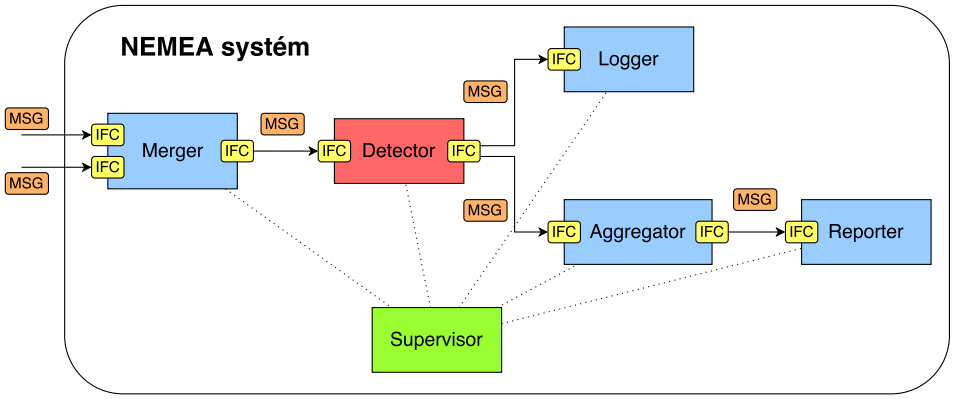
Systém pracuje se síťovými toky a umožňuje analýzu v reálném čase. Doba nahlášení detekované události závisí na čase, za který se data dostanou do systému a také na zpoždění detekčních algoritmů. V praxi tato doba může představovat desítky sekund až jednotky minut. Systém zpracovává data proudově, což znamená, že data systémem prochází bez potřeby ukládání a načítání z disku. Unikátní vlastností systému NEMEA je podpora zpracování rozšířených síťových toků o informace z aplikačních vrstev. To je možné hlavně díky efektivnímu formátu posílaných zpráv mezi moduly.

### 1.2.1 Části NEMEA systému

Na obrázku 1.1 jsou barevně znázorněny všechny důležité části systému NEMEA. Nejdůležitější částí systému je framework implementující knihovny *libtrap* (**Tr**affic **A**nalysis **P**latform), *UniRec* (**Uni**fied **Rec**ord) a *common*.

Knihovna *libtrap* implementuje komunikační rozhraní modulů (na obrázku žlutě). Rozhraní jsou jednosměrná a každý modul může mít libovolný počet vstupních a výstupních rozhraní. Podle potřeby lze využít `TCP` rozhraní pro komunikaci mezi stroji, `UNIXSOCKET` rozhraní pro komunikaci na stejném stroji a nebo `FILE` rozhraní pro čtení a zápis z/do souboru. Během komunikace zastává vstupní rozhraní roli klienta a výstupní rozhraní roli serveru. Knihovna *UniRec* poskytuje efektivní binární formát zpráv posílaných přes rozhraní modulů (na obrázku oranžově). Poslední knihovna *common* obsahuje často používané struktury (např. B+ strom) a funkce (např. rozptylovací funkce).

Moduly jsou základní stavební bloky systému NEMEA a dají se rozdělit do dvou skupin. První skupina je na obrázku znázorněna červeně. Jedná se o detekční moduly, které implementují nějakou detekční metodu. Výstupem




Obrázek 1.1: Mezi základní části systému NEMEA patří moduly (detekční, agregační, reportovací a další), dále framework obsahující knihovny pro komunikační rozhraní modulů (žlutě) a efektivní formát posílaných zpráv (oranžově) a také řídící modul `NEMEA Supervisor` pro centrální správu systému.

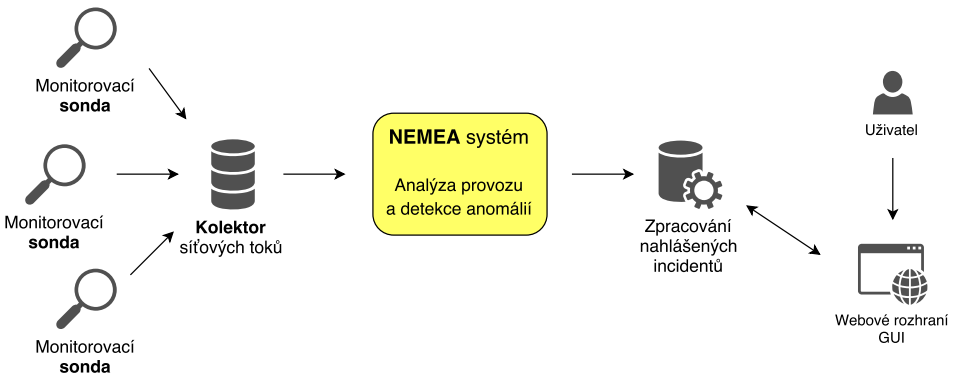
detekčních modulů je obvykle zpráva obsahující informace o nahlášené události (například IP adresa útočníka a oběti, použité porty, čas a typ útoku).

Druhá skupina je na obrázku znázorněna modře a patří do ní ostatní moduly. Podle funkce rozlišujeme: i) zdroje dat, ii) agregační a filtrační moduly, iii) logovací a reportovací moduly. Mezi zdroje dat patří například moduly, které čtou soubory s daty ve formátu `NFDUMP`, `CSV` nebo `JSON`. Filtrace je užitečná například pro vybrání provozu pouze některých adres, na které je třeba se zaměřit. Agregační modul může být použit jako na obrázku k agregaci často hlášených událostí detekčním modulem. Logovací a reportovací moduly jsou zapojeny většinou na konci procesu zpracování a slouží kromě zapsání do souboru i k poslání událostí do databáze pro sdílení událostí Warden [2].

Poslední částí systému je speciální modul `NEMEA Supervisor` [3] (dále nazýván jen `Supervisor`, na obrázku zeleně), který umožňuje celý systém konfigurovat a monitorovat. Na základě uživatelem definované konfigurace umožňuje spouštět a zastavovat moduly, zobrazit stav systému a za běhu změnit konfiguraci. Monitoruje stav modulů, jejich využití systémových prostředků a také statistiky jejich komunikačních rozhraní (například počty přijatých a odeslaných zpráv). Modul `Supervisor` vznikl v rámci mé bakalářské práce a umožňuje spravovat NEMEA systém běžící na jednom výpočetním uzlu. Cílem této diplomové práce je systém NEMEA distribuovat na více výpočetních uzlů a docílit tak škálovatelnosti systému s ohledem na výsledky detekce.

### 1.2.2 Monitorovací infrastruktura

Obr. 1.2 ukazuje infrastrukturu, která se používaná k monitorování počítačové sítě pomocí systému NEMEA. Na páteřních linkách akademické sítě




Obrázek 1.2: Monitorovací infrastruktura počítačové sítě zahrnuje: i) monitorovací sondy exportující informace o provozu; ii) kolektor, který informace ukládá/předzpracovává/přeposílá; iii) detekční systém; iv) systém pro zpracování nahlášených událostí; v) rozhraní pro správce sítě.

CESNET2 jsou umístěny monitorovací sondy, které ze síťového provozu (tj. z jednotlivých paketů) exportují síťové toky na centrální kolektor toků. V současné době se používá `IPFIXcol` kolektor [4], který přijímá data ve formátu IPFIX [5]. Kolektor může přijatá data uložit, předzpracovat nebo přeposlat v jiném formátu na své výstupní rozhraní. V tomto případě posílá data ve formátu UniRec do systému NEMEA.

Systém NEMEA přijatá data proudově zpracovává a detekuje anomálie. Hlavním výstupem NEMEA systému jsou hlášení o detekovaných událostech, která jsou posílána do Warden systému a také do NEMEA Dashboardu [6]. Warden systém slouží ke sdílení informací o nahlášených bezpečnostních událostech mezi oprávněnými účastníky a pomáhá tak doplnit chybějící informace jednotlivých bezpečnostních týmů. NEMEA Dashboard je grafické rozhraní k prohlížení nahlášených událostí, které umožňuje zobrazovat grafy a přehledy, ale také provádět dotazy nad nahlášenými událostmi podle filtračních podmínek.

## 1.3 Útoky v počítačových sítích

Tato sekce popisuje bezpečnostní hrozby, pro které v systému NEMEA existují detekční moduly a pro které tato práce řeší paralelizaci zpracování. Jednotlivé detekční moduly jsou popsány v dalších kapitolách.

### 1.3.1 Skenování portů

Skenování portů je v počítačových sítích často pozorovaný jev. Samo o sobě je skenování neškodné a používá se například při penetračních testech k nalezení

otevřených portů, které mohou být potenciální slabinou využitelnou útočníkem. Jedním z programů pro skenování portů je snadno dostupný *nmap(1)*. Z pohledu útočníka je ale skenování dobrá příležitost, jak odhalit možné slabiny, které poté mohou být využity ke skutečnému útoku. Rozlišujeme následující tři typy skenování portů:

- **Vertikální** sken – na jednom cílovém stroji (jedné cílové IP adrese) je zkoušeno mnoho různých cílových portů
- **Horizontální** sken – na mnoha cílových strojích je zkoušen většinou jeden stejný cílový port
- **Blokový** sken – je kombinace obou předchozích, na mnoha cílových strojích je zkoušeno mnoho cílových portů

### 1.3.2 Útoky hrubou silou

Útok hrubou silou spočívá v systematickém zkoušení všech možností, dokud není nalezeno správné řešení. Tento typ útoku je útočníky většinou použit k prolamování hesel, uživatelských jmen nebo šifrovacího klíče v kryptografii. V případě krátkých hesel je poměrně jednoduché a rychlé vyzkoušet všechny možnosti, ale při zkoušení delších hesel exponenciálně roste počet možností, a proto se většinou používá metoda slovníkového útoku. Slovníkové útoky zkouší *N* nejčastějších hesel jako například *1234* nebo *password* a z důvodu neopatrnosti uživatelů jsou často úspěšné.

V počítačových sítích je tento typ útoku často používán k hádání hesel služeb pro vzdálenou správu (např. *SSH, RDP, TELNET*), hádání uživatelských jmen a hesel v IP telefonii a mnoha dalších.

### 1.3.3 Denial of Service

Denial of Service (DoS) útok je v dnešní době jeden z nejčastějších útoků, který má za cíl učinit nějakou službu nedostupnou pro její běžné uživatele. Může jít například o konkrétní stroj, webovou službu nebo síťovou linku. Následkem tohoto útoku mohou být finanční ztráty, poškození reputace firem, ale také ohrožení na zdraví.

Při tomto útoku jde často o zaplavení cílového stroje nebo linky takovým množstvím dat/požadavků, které není možné zpracovat nebo které vyčerpá prostředky cíle. Dochází poté k zahazování regulérních požadavků a stroj nebo služba se pro uživatele tváří jako nedostupná. Příkladem může být zaplavení SYN pakety (*SYN flood*) nebo zaplavení ICMP pakety (*ICMP flood*).

Zvláštní kategorií DoS útoku je jeho distribuovaná varianta (Distributed DoS), ve které není zdroj útoku jeden, ale mnoho. Může jít například o kontrolovanou skupinu malwarem nakažených počítačů (tzv. *botnet*), který je obvykle používaný k rozesílání nevyžádané pošty, dalšímu šíření malware nebo právě k DDoS útokům.

### 1.3.4 DNS amplifikační útok

Domain Name System (DNS) je klíčový systém a s ním spjatý protokol pro komunikaci mezi DNS servery, jejímž hlavním úkolem je překlad IP adres a doménových jmen. DNS amplifikační útok využívá DNS protokol ze dvou hlavních důvodů. DNS požadavky mohou být posílány pomocí protokolu jako je UDP, což znamená, že nemusí být navázáno spojení, jako v případě TCP protokolu. Proto může být zdrojová IP adresa DNS požadavku podvržena. Druhým důvodem je fakt, že na krátké DNS dotazy může přijít odpověď s násobnou velikostí – z tohoto důvodu v názvu amplifikace (znásobení).

Během útoku útočník posílá DNS dotazy na DNS server, ve kterých je zdrojová IP adresa podvržena za IP adresu oběti. DNS server tedy odpovědi posílá na adresu oběti. Dále útočník volí v dotazech takové domény, které mají registrovány mnoho DNS záznamů. Tím dojde k amplifikaci útoku, protože odpověď bude mít násobnou velikost požadavku.

Následkem tohoto útoku je zahlcení oběti DNS odpovědmi, a proto spadá útok do kategorie DoS. V praxi útočník používá mnoho zdrojů DNS dotazů a DNS serverů najednou a způsobuje tak DDoS útok.

### 1.3.5 Seznamy nebezpečných adres

S přibývajícím množstvím malwarem nakažených počítačů získávají seznamy nebezpečných adres na důležitosti. Tyto seznamy (tzv. *blacklists*) obsahují IP adresy počítačů, které jsou buď nakažené malwarem, slouží k ovládání sítě nakažených počítačů (tzv. *botnet*) nebo vykazují neobvyklé a nebezpečné chování. Je důležité monitorovat komunikaci IP adres na těchto seznamech, protože se tak může zabránit například dalšímu šíření malwaru.

Příklady blacklistu jsou *Zeus Tracker* nebo *Palevo Tracker*, které obsahují IP adresy řídících a ovládacích (*Command and Control*) serverů botnetu.

## 1.4 Existující řešení

V oblasti paralelního zpracování síťových dat za účelem detekce škodlivého provozu bylo publikováno mnoho prací s různými přístupy.

K. Xinidis et al. ve [7] ve své architektuře distribuovaného NIDS použili tzv. *aktivní rozdělovač*, který proud paketů před posláním do sítě rozděloval detekčním senzorům, na kterých běžel detekční systém Snort [8]. Snort patří do skupiny systémů, které v celých paketech hledají definované vzory škodlivého chování. *Aktivní rozdělovač* používal rozptylovací funkci k rozdělování paketů mezi detekční senzory a měl za cíl optimalizovat jejich výkon pomocí tří základních opatření. Tzv. *Cummulative Acknowledgments* optimalizace měla za cíl minimalizovat redundantní posílání paketů mezi rozdělovačem a senzory tím, že pakety přeposlané na senzory k analýze se na okamžik uložily v rozdělovači a senzor místo přeposlání celého paketu zpět na rozdělovač po-

sílá unikátní ID paketu. *Locality Buffering* optimalizace mění pořadí paketů posílaných k analýze tak, aby za sebou byly pakety pokud možno stejných služeb (např. FTP) a detekční senzory tak mohou použít detekční pravidla, která mají načtená ve skryté paměti a tím dojde ke znatelnému zrychlení. Poslední *Early Filtering* optimalizace spočívá v analýze paketů už na rozdělovači. Rozdělovač používá podmnožinu definovaných Snort pravidel, která se aplikují pouze na hlavičky paketů a ty pakety, které jsou v pořádku, rovnou přeposílá do sítě místo na senzory.

H. Sallay et al. ve [9] představili architekturu pro distribuované zpracování síťového provozu, kde switch/router rozděloval pakety podle směrovací tabulky jednoúčelovým senzorům. Tyto senzory prováděly analýzu pomocí detekčního systému Snort, ale pouze s podmnožinou pravidel pro jednu konkrétní službu (např. HTTP, FTP). Jelikož senzory zpracovávají provoz pouze určité služby a objem dat jednotlivých služeb se může velice lišit, není tento přístup pro vysokorychlostní sítě z hlediska rovnoměrného zatížení výpočetních uzlů použitelný.

Nam-Uk Kim et al. se v [10] zabývají statickým a dynamickým vyvažováním zátěže výpočetních uzlů založeném na rozptylovacích funkcích. Představují dynamické (tj. adaptivní) schéma vyvažování zátěže výpočetních uzlů pro NIDS. Toto schéma používá vyhledávací tabulku, která je pravidelně reorganizována podle historického rozdělení paketů a také podle aktuální zátěže jednotlivých výpočetních uzlů. Pokud je některý uzel přetížený, dojde k přesměrování síťových toků s nejmenším objemem dat. Navržený přístup sice reorganizuje síťové toky tak, že nedojde k jejich narušení, ale nebere v potaz sémantické vazby mezi jednotlivými síťovými toky. Také chybí vyhodnocení vlivu rozdělení dat na výsledky detekce.

M. Valentin et al. ve [11] prezentují NIDS klastr pro škálovatelnou detekci anomálií. Architektura klastru obsahuje *front-end* uzly, které rozdělují příchozí pakety mezi *back-end* uzly, na kterých je prováděna detekce anomálií pomocí systému Bro [12]. *Proxy* uzly dále zajišťují propagaci stavových informací mezi *back-end* uzly a všechny výsledky detekce jsou sbírány a agregovány na zvláštním uzlu *central manager*. Každý *front-end* uzel zpracovává pakety z jedné monitorované linky a rozdělování provádí pomocí schématu založeném na jedné rozptylovací funkci. Navržená architektura vyžaduje komunikaci mezi *back-end* a *proxy* uzly pro výměnu informací o mezivýsledcích detekce. Tato diplomová práce používá pro detekci systém NEMEA, který je nasazen na nezávislých výpočetních uzlech, které mezi sebou nekomunikují (nevyměňují si mezivýsledky detekce). Proto je nutné takové rozdělovací schéma, které poskytne výpočetním uzlům všechna potřebná data pro správnou detekci. Navíc požadované schéma nerozděluje proud paketů, ale síťových toků.

Důležitou roli v paralelním zpracování dat hrají tzv. *Big data frameworky*. Jedním z nich je Hadoop [13]. Jde o framework pro ukládání a zpracování velkého množství dat v řádech petabajtů na mnoha výpočetních uzlech. Součástí Hadoopu je distribuovaný souborový systém, který slouží k uložení a

správě dat. Rozdělení dat na jednotlivé uzly probíhá po blocích, což je při paralelním zpracování využito k zachování lokality a minimalizaci výměny dat mezi uzly. K paralelnímu zpracování distribuovaných dat slouží součást zvaná *MapReduce* implementující dvoufázový výpočet. První fáze *Map* spočívá v opakovaném načítání dat z fyzického úložiště, aplikování určité operace a uložení výsledku zpět do fyzického úložiště všemi uzly paralelně. V této fázi se spočítají lokální výsledky jednotlivých uzlů. V druhé fázi *Reduce* se lokální výsledky uzlů spojují do konečného výsledku.

Dalším *Big data frameworkem* je Spark [14] sloužící ke zpracování velkého množství distribuovaných dat. Algoritmus pro paralelní zpracování dat, použitý ve Spark, je násobně rychlejší, než je MapReduce použitý v Hadoopu. Jedním z hlavních důvodů zrychlení je vykonání většiny operací nad načtenými daty z fyzického úložiště najednou v logické paměti RAM. Ve srovnání s Hadoop, který se hodí pro off-line zpracování, je Spark vhodný i pro proudové zpracování dat.

Hashdoop [15] je vylepšení Hadoopu se zaměřením na zpracování síťového provozu a detekci anomálií. Použitím rozptylovacích funkcí při rozdělování paketů jednotlivým uzlům je docíleno zachování určité struktury síťového provozu. K určení cílového uzlu je použito rozptylování zdrojové a cílové IP adresy paketu. Pro paralelní zpracování je opět použit MapReduce, což neumožňuje proudové zpracování.

Vzorkování (sampling) může mít stejný cíl jako paralelní zpracování a to zpracování více dat najednou a rychleji. Analýza vzorkovaného síťového provozu ale nezaručuje správný výsledek. J. Mai et al. se ve [16] zabývají dopadem vzorkování paketů na detekci skenování portů. Ve [17] K. Bartos et al. představili techniky pro vzorkování síťových toků v kontextu detekce anomálií. V této práci jsou ale tyto přístupy nepoužitelné, protože se nedají aplikovat na rozdělení proudu síťových toků.

## 1.5 Rozptylovací funkce

Rozptylovací funkce je funkce, která transformuje vstupní data libovolné velikosti na výstupní data fixní velikosti. Výstupní hodnota funkce je nejčastěji nazývána haš hodnota nebo jen haš.

Častým použitím rozptylovací funkce je datová struktura *rozptylovací tabulka*, ve které se pracuje s dvojicemi *klíč* a *hodnota*. Z klíče (vstupní data funkce) je spočítána haš určující index, na kterém se nachází hodnota příslušející danému klíči. Operace vkládání, mazaní a vyhledávání jsou provedeny u této struktury v konstantním čase (myšleno asymptoticky), v nejhorším případě pak v lineárním.

V bezpečnosti se používají rozptylovací funkce například ke kontrolním součtům, otiskům nebo šifrování. Kontrolní součet (*checksum*) může být použit pro ověření, zda přenesená zpráva odpovídá originálu, tzn. detekce chyb během

přenosu. Spočítaná haš přenesené zprávy je porovnána s přiloženým součtem. Otisk (*fingerprint*) je unikátní identifikátor fixní délky spočítaný rozptýlením libovolně dlouhých dat. Porovnáváním otisků souborů jde například určit jejich modifikace.

Rozptylovací funkce mohou být použity i při rozdělování dat pro paralelní zpracování. Příchozí proud dat může být rozdělen a mapován na výpočetní uzly tak, že z informace obsažené v datech se spočítá haš a ta určí číslo výpočetního uzlu, který příslušná data zpracuje.

Podle účelu rozptylovací funkce jsou požadovány některé z následujících vlastností:

- **Rovnoměrnost** – haše vypočítané z možných vstupních dat by měly být rovnoměrně zastoupeny, tzn. všechny haše by měly mít stejnou pravděpodobnost
- **Minimum kolizí** – pravděpodobnost, že pro dvě různé vstupní hodnoty vyjde stejná haš, by měla být minimální
- **Jednosměrnost** – výpočet haše ze vstupních dat je jednoduchý, ale vypočítat na základě haše vstupní hodnotu by mělo být výpočetně velice náročné, až nemožné
- **Efektivita** – výpočet haše je jednoduchý a nenáročný na výpočetní čas
- **Využití vstupních dat** – k výpočtu výsledné haše jsou použity pokud možno všechny části vstupních dat
- **Důsledek změn vstupních dat** – při změně vstupních dat je v různých případech použití rozptylovací funkce požadován jiný dopad na výslednou haš. Při vyhledávání v rozptylovací tabulce je žádoucí, aby podobné vstupní hodnoty měly podobné haše. Naopak v kryptografii je při malé změně vstupní zprávy požadována velká změna výsledné haše (výsledné hodnoty mají od sebe velkou vzdálenost).

Kapitola 2

# Návrh

Tato kapitola se zabývá návrhem rozšíření systému NEMEA, které umožňuje distribuované nasazení systému. Hlavním cílem je zvýšení propustnosti celého systému pomocí paralelního zpracování velkého množství dat s ohledem na zachování správných výsledků detekce.

## 2.1 Možnosti paralelizace

Stěžejním místem proudového zpracování systémem NEMEA jsou detekční moduly provádějící analýzu síťového provozu. Množství dat, které dokáží zpracovat za určitý čas, závisí na složitosti algoritmů. Pro paralelní zpracování dat mnoha detekčními algoritmy se nabízejí obecně tři základní možnosti:

1. Paralelizace sekvenčních částí detekčních algoritmů
2. Spuštění neupravených detekčních algoritmů na oddělených výpočetních jádrech nebo strojích
3. Rozdělení dat ke zpracování mezi stejné výpočetní uzly

První možnost zahrnuje analýzu jednotlivých detekčních algoritmů a následnou paralelizaci sekvenčně prováděných částí. Důsledkem by byla zvýšená propustnost detekčních modulů a tím i větší množství zpracovaných dat. Většina algoritmů jde ale paralelizovat jen zčásti a některé nejdou paralelizovat vůbec.

Druhá možnost nevyžaduje úpravu algoritmů, ale spuštění každého z nich na odděleném jádru procesoru nebo odděleném výpočetním uzlu. Sníží se tak počet úloh prováděných jedním procesorem a detekční moduly dostanou přiděleno více výpočetních prostředků a zpracují tak více dat.

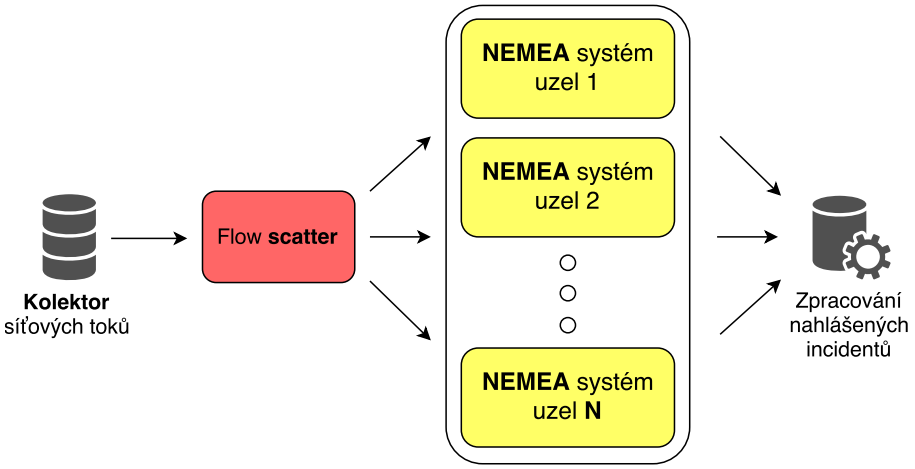
Třetí možnost paralelizace vyžaduje spuštění stejné skupiny neupravených detekčních algoritmů na všech výpočetních uzlech a data ke zpracování jsou mezi uzly pokud možno rovnoměrně rozdělena. S vhodným způsobem rozdělení

dat, který neovlivní výsledky detekce, nabízí tato možnost obecný způsob paralelního zpracování síťových toků, který není závislý na implementovaných algoritmech ani na použitém detekčním systému. Škálování celého zpracování je v tomto případě možné přidáváním dalších výpočetních uzlů se stejnou množinou detekčních algoritmů. Z těchto důvodů byla pro návrh vybrána tato možnost paralelizace.

Následující sekce této kapitoly vycházejí ze zvoleného způsobu paralelizace *rozdělením dat mezi stejné výpočetní uzly*. Postupně popisují návrh potřebného rozšíření monitorovací infrastruktury, požadavky na rozdělení dat a také různé způsoby rozdělení dat mezi výpočetní uzly.

## 2.2 Rozšíření monitorovací infrastruktury

Na Obr. 1.2 bylo ukázáno, že systém NEMEA, běžící na jednom výpočetním uzlu, zpracovává příchozí proud síťových toků z `IPFIXcol` kolektoru, které byly zachyceny na mnoha monitorovacích sondách. Vybraný způsob paralelizace vyžaduje rozdělení dat a odeslání na distribuované výpočetní uzly. Pro zachování stávající monitorovací infrastruktury popsané v Sekci 1.2.2 je nutné rozdělovat data za kolektorem síťových toků. Obr. 2.1 ukazuje vložení nového prvku nazvaného `Flow scatter`, který příchozí síťové toky rozděluje mezi jednotlivé výpočetní uzly.




Obrázek 2.1: Rozšíření monitorovací infrastruktury o prvek `Flow scatter`, který rozděluje příchozí síťové toky mezi výpočetní uzly s detekčním systémem NEMEA.

Proud síťových toků musí být modulem `Flow scatter` rozdělen takovým způsobem, aby byly splněny následující tři požadavky:

1. **rovnoměrnost rozdělení** – každý výpočetní uzel by měl zpracovat rovnoměrné množství dat, aby bylo zatížení pokud možno stejné

2. **efektivita rozdělení** – způsob rozdělování dat musí být jednoduchý a rychlý, aby s přidáváním výpočetních uzlů bylo možné zpracovávat co nejvíce dat a rozdělování nebrzdilo celý proces zpracování

3. **minimální dopad na výsledky detekce** - nahlášené události detekčních modulů na distribuovaných uzlech by měly celkově odpovídat nahlášeným událostem po zpracování všech dat jedním uzlem.

Rozdělování dat může obecně probíhat staticky nebo dynamicky. V kontextu analýzy síťových dat a detekci anomálií mají oba dva způsoby své výhody i nevýhody. Statické rozdělování po celou dobu udržuje neměnná pravidla pro rozdělení. Například při rozdělování paketů mohou být dána pravidla: i) paket se zdrojovou IP adresou 11.12.13.14 patří na uzel číslo 2; ii) paket se zdrojovou IP adresou 21.22.23.24 patří na uzel číslo 5. Výhodou tohoto chování je, že pokud proud paketů takto přiřazený některému uzlu obsahuje bezpečnostní incident, bude detekován. Na druhou stranu, pokud je zatížení výpočetních uzlů nerovnoměrné, statické rozdělování neprovádí žádná opatření k vyrovnání zátěže.

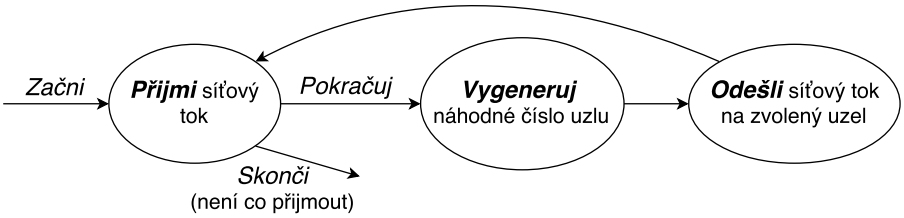
Dynamické rozdělování může v případě nerovnoměrného rozdělení učinit opatření k nápravě. Například může přesměrovat některé pakety z přetíženého uzlu na málo zatížené uzly. Nevýhodou tohoto chování je ale narušení proudu paketů, které mohou obsahovat bezpečnostní incident. Je možné, že rozdělením proudu paketů zůstane incident nedetekován. Z tohoto důvodu byl vybrán statický způsob rozdělování s důrazem na rovnoměrnost.

Následující sekce popisují různá schémata rozdělení, podle kterých `Flow scatter` rozhazuje síťové toky mezi výpočetní uzly.

## 2.3 Schéma náhodného rozhazování

Za předpokladu, že mezi jednotlivými síťovými toky neexistují žádné sémantické vazby, nabízí se možnost toky rozhazovat mezi výpočetní uzly náhodně. S tímto předpokladem lze použít statistické rovnoměrné rozdělení, které je z hlediska rovnoměrnosti optimální. `Flow scatter` proto využívá pseudonáhodný generátor čísel k určení čísel výpočetních uzlů.

Obr. 2.2 ukazuje hlavní cyklus jednoduchého algoritmu `Flow scatteru`, který rozhazuje síťové toky náhodně. Cyklus spočívá v příjetí síťového toku, vygenerování náhodného čísla v rozsahu *1* až *počet uzlů*, které slouží jako index výpočetního uzlu a odeslání toku na výpočetní uzel s daným indexem.

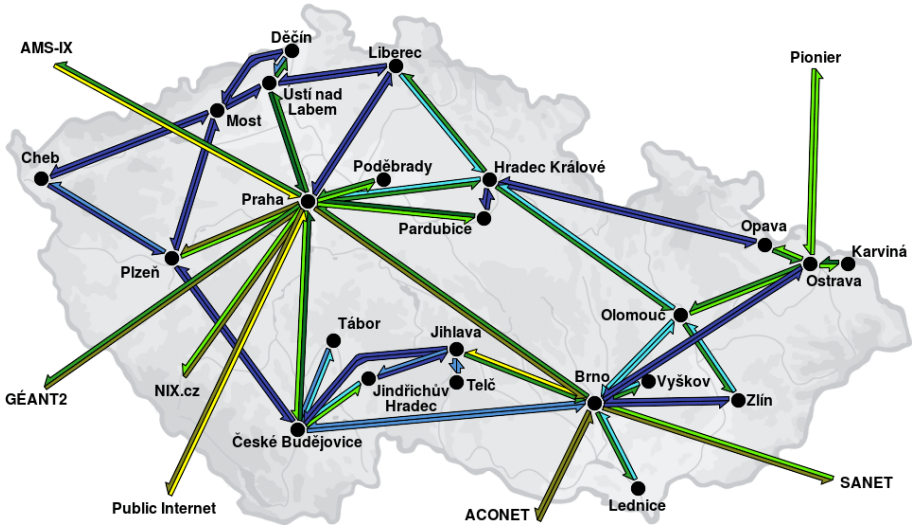



Obrázek 2.2: Schéma popisující hlavní cyklus algoritmu, který síťové toky výpočetním uzlům rozhazuje náhodně.

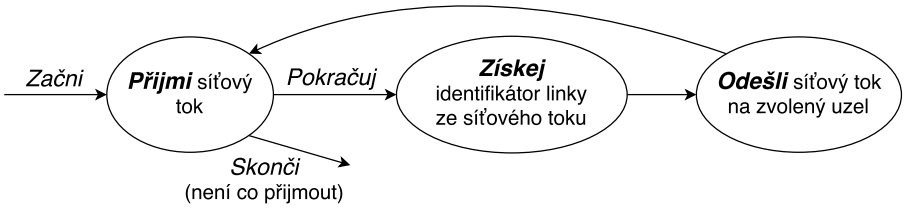
## 2.4 Schéma rozhazování na základě topologie

Akademická síť CESNET2 je páteřní síť, která je využívaná také jako tranzitní. Na linkách, které síť CESNET2 propojují s ostatními sítěmi viz Obr. 2.3, jsou umístěny monitorovací sondy, které exportují informace o provozu v podobě síťových toků. Každý tok mimo obvyklé položky jako jsou IP adresy, porty a další, obsahuje také identifikátor linky, ze které byl exportován. Identifikátory linek jsou reprezentovány jednotlivými bity v položce, která se nachází v exportovaném síťovém toku. Identifikátor je tedy mocninou čísla dva.

Pokud je každá linka přiřazena určitému výpočetnímu uzlu, `Flow scatter` podle identifikátoru linky v každém síťovém toku pozná, na který výpočetní uzel tok pošle. Touto metodou dochází k rozdělení dat na podmnožiny se stejnou geografickou polohou. Obr. 2.4 ukazuje hlavní cyklus jednoduchého algoritmu modulu `Flow scatter`, který rozhazuje síťové toky podle identifikátoru linky.




Obrázek 2.3: Monitorované linky akademické sítě CESNET2.




Obrázek 2.4: Schéma popisující hlavní cyklus algoritmu, který síťové toky výpočetním uzlům rozhazuje podle identifikátoru linky, ze které byly toky exportovány.

Při použití této metody rozhazování by bylo z hlediska úspory zdrojů výhodnější z každé monitorovací sondy posílat data na analýzu rovnou. Se stávající monitorovací infrastrukturou jsou exportovaná data ze sond kumulována na centrální kolektor, a poté znovu rozdělena výpočetním uzlům na stejné části jako před sloučením. Tato otázka je diskutována spolu s experimenty v Sekci 4.4.

## 2.5 Schéma rozhazování rozptylováním

Rozptylovací funkce je modulem `Flow scatter` použita k určení výpočetního uzlu, který zpracuje aktuální síťový tok. Požadované vlastnosti, které byly zmíněny v Sekci 1.5, jsou v tomto případě hlavně rovnoměrnost rozptylování a efektivita. Vstupem rozptylovací funkce je informace obsažená v každém síťovém toku. Je potřeba určit, kterou informaci ze síťových toků k rozptylování využít.

Rozptylovací funkce rozdělí síťové toky na podmnožiny se stejnou charakteristikou, které budou zpracovány stejným výpočetním uzlem. Pokud bude rozptylovací funkce používat jako vstup například zdrojovou IP adresu každého toku, výsledné podmnožiny toků budou mít stejnou zdrojovou IP adresu. Tato závislost zvoleného výpočetního uzlu na vstupních datech rozptylovací funkce může být využita pro účely detekce, konkrétně k poskytnutí všech potřebných dat výpočetním uzlům nutných k úspěšné detekci.

Za předpokladu, že nějaká množina síťových toků obsahuje bezpečnostní incident, který lze detekovat určitou detekční metodou, potom existuje minimální podmnožina síťových toků, která je dostačující k detekování incidentu. Mezi toky v této podmnožině existují sémantické vazby, které pro úspěšnou detekci nesmí být porušeny. Odstranění libovolného toku z minimální podmnožiny může způsobit neúspěšnou detekci. Proto je nutné najít takové charakteristiky, podle kterých rozptylovací funkce toky rozdělí, aby nebyly narušeny sémantické vazby jednotlivých síťových toků. Za tímto účelem byly prozkoumány detekční moduly systému NEMEA. Následující podsekce obsahuje jejich popis spolu s nalezenými charakteristikami.

### 2.5.1 Detekční metody NEMEA systému

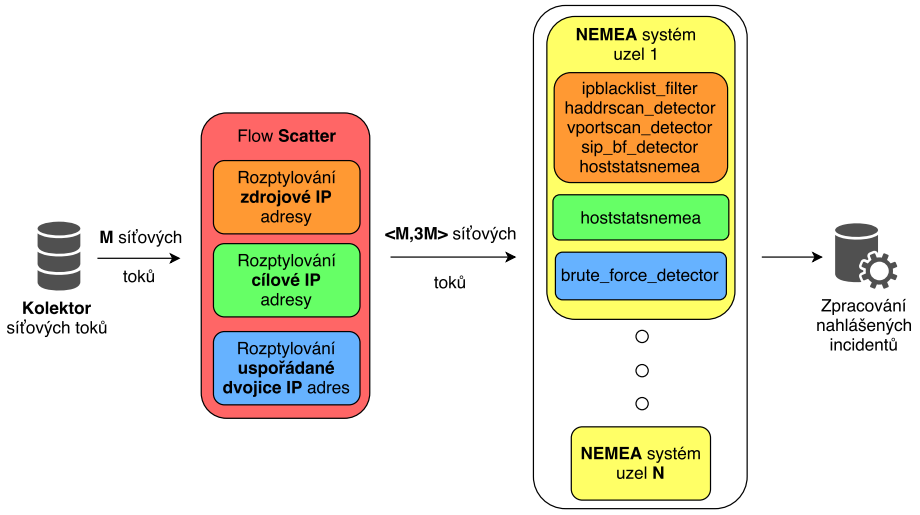
Modul `vportscan_detector` implementuje metodu detekce vertikálního skenování portů publikovanou v [18]. Metoda si pro každou zdrojovou IP adresu ukládá unikátní cílové porty použité ke komunikaci s jednou cílovou IP adresou. Jedním z hlavních parametrů metody je maximální limit použitých portů jednou zdrojovou IP adresou. Pokud nějaká IP adresa tento limit překročí, metoda nahlásí vertikální skenování portů. Aby tato metoda správně fungovala, musí modul obdržet všechny síťové toky se stejnou zdrojovou IP adresou. Toho lze dosáhnout rozptylováním podle zdrojové IP adresy (adresa možného útočníka – zdroje skenu) z každého síťového toku.

Velice podobná metoda pro detekci horizontálního skenování portů je implementována modulem `haddrscan_detector`. Tato metoda si pro každou dvojici zdrojové IP adresy s cílovým portem ukládá unikátní cílové IP adresy, se kterými daná zdrojová IP adresa komunikovala. Pokud nějaká IP adresa překročí limit cílových adres, metoda nahlásí horizontální skenování portů. Modul opět musí obdržet všechny síťové toky se stejnou zdrojovou IP adresou, proto je použito rozptylování podle zdrojové IP adresy.

Modul `hoststatsnemea` implementuje metodu, která si pro každou IP adresu počítá statistiky (např. počet bajtů, paketů, zdrojových a cílových portů) a dokáže detekovat DoS útok, DNS amplifikační útok, horizontální skeny a další. Jednotlivé útoky jsou popsány pravidly/podmínkami, které musí IP adresa splnit pro nahlášení události. Pro správnou detekci tato metoda musí obdržet všechny síťové toky se stejnou adresou. V tomto případě nezáleží na tom, jestli je adresa zdrojová nebo cílová. Z toho důvodu je nutné rozptylovat zvlášť podle zdrojové a zvlášť podle cílové IP adresy každého síťového toku. Pokud jsou výsledné haše rozdílné, síťový tok je poslán na dva rozdílné výpočetní uzly a dojde k duplikaci. Tato situace bude diskutována v Sekci 2.5.2.

Modul `ipblacklist-filter` detekuje IP adresy, které se nacházejí na veřejně dostupných seznamech nebezpečných IP adres. Tato metoda je velice jednoduchá a funguje pro jakékoliv rozdělení dat, protože ke správné detekci stačí jediný síťový tok s danou IP adresou (zdrojovou nebo cílovou). Rozptylovat lze v tomto případě podle jakékoliv informace ze síťového toku, ale vybrána byla zdrojová IP adresa.

Jedním z modulů, který zpracovává síťové toky obohacené o informace z aplikačních vrstev, je `sip_bf_detector`. Modul implementuje metodu, která detekuje pokusy o prolomení hesla a skenování uživatelských jmen SIP protokolu (Session Initiation Protocol), který se často používá jako signalizační protokol v IP telefonii. Metoda kontroluje odpovědi SIP serveru, konkrétně zprávy `401 Unauthorized`, které server posílá po neoprávněném požadavku o registraci. Pokud je překročen maximální limit těchto odpovědí na registraci se stejným uživatelským jménem, dojde k nahlášení pokusu o prolomení hesla. Požadavky o registraci mohou být ale posílány i s různými uživatelskými jmény. Pokud je při neúspěšných registracích překročen limit použitých uživa-



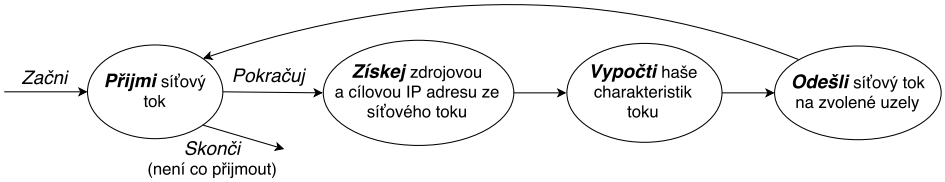

Obrázek 2.5: Při rozhazování rozptylováním detekční moduly vyžadují od modulu `Flow scatter` rozptýlení různých informací obsažených v síťových tocích. Díky tomu jsou zachovány sémantické vazby toků a detekční metody obdrží všechna potřebná data.

telských jmen, dojde k nahlášení skenování uživatelských jmen. Ke správnému fungování musí modul obdržet všechny síťové toky se stejnou zdrojovou IP adresou (adresa SIP serveru, který posílá odpověď). Proto je nutné rozptylování podle zdrojové IP adresy.

Metoda pro detekci pokusů o prolomení hesla u služeb pro vzdálenou správu (SSH, TELNET, RDP) je implementována modulem `bf_detector`. Metoda prohledává okénka o $N$ síťových tocích obousměrné komunikace dvou IP adres a podle pravidel (zahrnují např. TCP příznaky, počet bajtů, počet paketů), která popisují jednotlivé útoky, posuzuje překročení limitu pokusů. Tato metoda tedy potřebuje obdržet všechny síťové toky mezi dvěma stejnými adresami v obou směrech. Toto je zajištěno rozptylováním podle uspořádané dvojice IP adres každého síťového toku.

Z nastudovaných detekčních metod systému NEMEA byly identifikovány 3 skupiny detekčních metod, kde každá skupina potřebuje zpracovat síťové toky se stejnou charakteristikou na jednom výpočetním uzlu. První skupina má společnou charakteristiku zdrojovou IP adresu, druhá skupina cílovou IP adresu a třetí skupina uspořádanou dvojici IP adres. Pokud dojde k rozdělení síťových toků podle těchto charakteristik, nedojde k narušení sémantických vazeb mezi jednotlivými toky a detekční metody zpracují všechna potřebná data k úspěšné detekci. Obr. 2.5 ukazuje všechny tři potřebné rozptylovací funkce v modulu `Flow scatter` a k nim barevně odpovídající skupiny detekčních metod na výpočetním uzlu. Barva skupiny metod souvisí s charakteristikou, podle které rozptylovací funkce síťové toky rozděluje.

Obr. 2.6 ukazuje hlavní cyklus algoritmu modulu `Flow scatter`, který rozhazuje síťové toky rozptylováním zvolených charakteristik. Po přijetí síťového toku jsou z něho získány potřebné charakteristiky a z nich jsou vypočítány haše, které určují čísla výpočetních uzlů.




Obrázek 2.6: Schéma popisující hlavní cyklus algoritmu, který síťové toky výpočetním uzlům rozhazuje rozptýlením určitých charakteristik (např. zdrojové IP adresy).

### 2.5.2 Duplikace síťových toků

Všechny výpočetní uzly obsahují stejnou množinu detekčních metod, které byly v předešlé podsekci rozděleny do tří skupin podle potřeby rozdělení dat. Modul `Flow scatter` musí proto pro každý síťový tok spočítat všechny tři haše. Protože každá rozptylovací funkce používá jako vstup jinou informaci ze síťových toků (zdrojovou IP adresu, cílovou IP adresu, uspořádanou dvojici IP adres), je možné, že všechny tři haše nebudou stejné. V případě, že jsou haše různé, `Flow scatter` odešle aktuální síťový tok na více výpočetních uzlů a tím vznikne duplikace. Jeden síťový tok může být odeslán na jeden až tři výpočetní uzly (podle počtu rozptylovacích funkcí nebo skupin detekčních metod). Vzniklá duplikace je nutná k poskytnutí všech potřebných dat každé detekční metodě.

Popsaná duplikace není závislá na škálování zpracování. Přidáním dalšího výpočetního uzlu se duplikace nezvyšuje a jeden síťový tok bude v nejhorším případě poslán na tři výpočetní uzly. Navíc, konkrétní rozptylovací funkce neurčuje svým výsledkem pouze číslo výpočetního uzlu, ale také určuje skupinu detekčních metod, které síťový tok na výsledném výpočetním uzlu zpracují. To je dáno informací (charakteristikou), kterou rozptylovací funkce ze síťových toku používá. Z toho plyne, že každý síťový tok je zpracován konkrétní skupinou detekčních metod pouze na jednom výpočetním uzlu a na všech výpočetních uzlech dohromady je zpracován každou detekční metodou pouze jednou. Pokud například pro nějaký síťový tok určí rozptylovací funkce modulu `Flow scatter` z Obr. 2.5 výpočetní uzly číslo 2 (oranžová haš), 3 (zelená haš) a 5 (modrá haš), potom je síťový tok zpracován oranžovou skupinou na uzlu 2, zelenou skupinou na uzlu 3 a modrou skupinou na uzlu 5. Duplikace tedy postihne pouze komunikační část mezi modulem `Flow scatter` a výpočetními uzly.

### 2.5.3 Srovnání rozptylovacích schémat

Při srovnání navrženého rozptylovacího schématu s více funkcemi a schématu s jednou funkcí lze ukázat, že pro popsané detekční metody by jedna funkce nestačila. Nechť se rozptylovací funkce rovná $hash = md5(adresa1 + adresa2)$, použitá v [11], kde se pomocí *md5* rozptyluje součet IP adres v paketu. Pokud by byla tato rozptylovací funkce použita k rozdělení dat například pro NEMEA detektor `bf_detector`, výsledky detekce by byly správné, protože všechny síťové toky mezi dvěma stejnými adresami v obou směrech by skončily na stejném výpočetním uzlu (výsledek *md5(A + B)* se rovná *md5(B + A)*, kde *A* a *B* jsou IP adresy).

U detektoru horizontálních skenů `haddrscan_detector` by ale podmínka zachování sémantických vazeb mezi jednotlivými síťovými toky dodržena nebyla. Při tomto skenu je u potřebných síťových toků stejná zdrojová IP adresa, ale cílová adresa se mění. Může se stát, že dva síťové toky se stejnou zdrojovou a různou cílovou IP adresou by skončily na různých výpočetních uzlech, protože *md5(A + B)* se nemusí rovnat *md5(A + C)*, kde *A*, *B* a *C* jsou IP adresy.

Kapitola 3

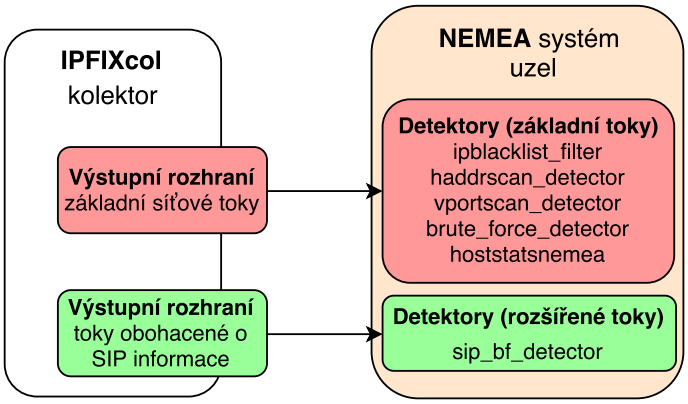
# Realizace

Tato kapitola se zabývá implementací navrženého rozšíření systému `NEMEA` v podobě modulu `Flow scatter`. Nový modul je umístěn mezi kolektorem síťových toků `IPFIXcol` a `NEMEA` systémem, tj. jednotlivými `NEMEA` moduly. Kolektor a moduly spolu komunikují pomocí komunikačních rozhraní implementovaných v knihovně *libtrap* a posílaná data jsou ve formátu *UniRec*, implementovaném ve stejnojmenné knihovně. Obě dvě knihovny jsou součástí `NEMEA` systému. Aby nový modul `Flow scatter` zapadl do stávající architektury, byl implementován jako `NEMEA` modul. Z výkonnostních důvodů byl pro implementaci vybrán jazyk C. Knihovnami podporované možnosti byly jazyky *Python* a *C*

## 3.1 Rozhraní modulu

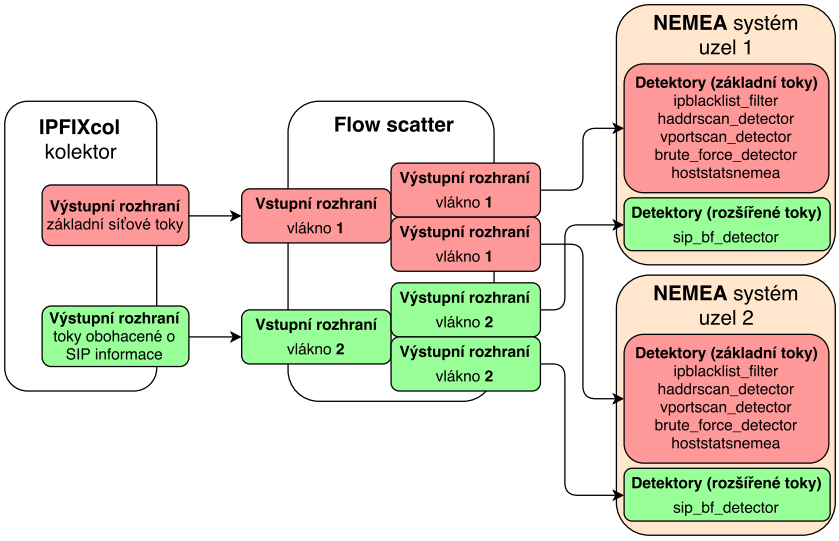
`IPFIXcol` na svá výstupní rozhraní posílá síťové toky různého druhu. Na jedno rozhraní posílá tzv. základní toky, které obsahují informace bez aplikačních vrstev (tzn. do transportní vrstvy včetně). Další rozhraní `IPFIXcolu` slouží k posílání síťových toků obohacených o informace z aplikačních vrstev (např. DNS, SIP). Detekční moduly jsou připojeny podle potřeby dat na jedno z těchto rozhraní. Na Obr. 3.1 jsou vidět detekční moduly diskutované v Sekci 2.5.1 připojené na potřebné zdroje dat. Modul `sip_bf_detector` je přípojený k rozhraní s toky obohacené o SIP informace a ostatní detekční moduly jsou připojeny k rozhraní se základními toky.

Každý výpočetní uzel, na který se z modulu `Flow scatter` posílají síťové toky obsahuje stejnou množinu detekčních modulů, ukázaných na Obr. 3.1. To znamená, že `Flow scatter` musí mezi výpočetní uzly rozhazovat data ze všech výstupních rozhraní kolektoru. Pro zajištění dobré propustnosti modul `Flow scatter` zpracovává výstupní rozhraní `IPFIXcolu` paralelně. Jde tedy o vícevláknový program, kde každé vlákno zpracovává jedno výstupní rozhraní kolektoru. Jednotlivá vlákna podle zvoleného schématu rozhazování posílají síťové toky výpočetním uzlům.




Obrázek 3.1: Výstupní rozhraní `IPFIXcol` kolektoru s různými druhy exportovaných síťových toků a `NEMEA` detektory připojené na potřebná výstupní rozhraní.

Výsledný počet rozhraní modulu `Flow scatter` je závislý na počtu výstupních rozhraní kolektoru a na počtu výpočetních uzlů, mezi které jsou data rozdělována. Pokud například kolektor posílá základní síťové toky, toky obohacené o SIP informace a výpočetní uzly jsou 2, počet vstupních rozhraní modulu `Flow scatter` je 2, počet vláken také 2 a počet výstupních rozhraní bude 2 krát *počet uzlů*, tzn. 4, přičemž každé ze dvou vláken bude rozdělovat toky mezi 2 uzly (2 výstupní rozhraní). Tento příklad je ukázán na Obr. 3.2.




Obrázek 3.2: Data z výstupních rozhraní kolektoru jsou modulem `Flow scatter` zpracovávána paralelně. Každé vlákno v závislosti na zvolené metodě rozhazování posílá toky detekčním modulům, které daný typ toků potřebují.

## 3.2 Použité knihovny

Při implementaci modulu `Flow scatter` byly použity knihovny z `NEMEA` frameworku – *libtrap* a *UniRec*. Knihovna *libtrap* implementuje jednosměrná komunikační rozhraní, mezi kterými probíhá automatické propojování – vstupní klientské rozhraní se připojuje na výstupní serverové rozhraní. Hlavní částí knihovny je *API* pro posílání dat na výstupní rozhraní a přijímání dat ze vstupního rozhraní. Data jsou knihovnou z důvodu vyšší efektivity přenosu ukládána do mezipaměti a odesílána v blocích, což snižuje režii při systémových voláních funkcí *send()* a *recv()*. Nejdůležitější funkce pro přijetí dat ze vstupního rozhraní má následující tvar: *trap_recv(param1, param2, param3)*, kde parametry funkce jsou číslo vstupního rozhraní, ukazatel na data a délka dat. Analogicky vypadá i funkce pro odesílání.

Knihovna *UniRec* implementuje binární formát zpráv posílaných mezi moduly. Umožňuje vytvořit pro každé komunikační rozhraní šablonu, která zprávy rozděluje na políčka. Každé políčko je určeno dvojicí datový typ a jméno. Při vytváření šablony se obvykle specifikuje *N-tice* těchto dvojic. Tento formát je přirozeně aplikovatelný na síťové toky, které `NEMEA` systém zpracovává. Každý tok je také sestaven z množiny různých položek – políček (IP adresy, porty, protokol atd.). Šablona je poté používána v kontextu s nějakým blokem dat, nad kterým se dají provádět operace nastavování hodnoty určitého políčka pomocí funkce *ur_set()* nebo získání hodnoty určitého políčka pomocí funkce *ur_get()*. Stejná políčka v přijaté zprávě se dají zkopírovat na tatáž políčka v odesílané zprávě pomocí funkce *ur_copy_fields()*.

Kombinace těchto dvou knihoven umožňuje jednoduše přijímat na vstupních rozhraních síťové toky jakožto zprávy ve formátu *UniRec*, získat z nich potřebné informace, některé změnit nebo i přidat a odeslat je na příslušné výstupní rozhraní.

## 3.3 Vstup modulu

Modul `Flow scatter` přijímá vstupní parametry zadávané ve formě přepínačů v příkazové řádce. Ty jsou po spuštění programu rozpoznány a ovlivňují například typ použitého rozhazovacího schématu. Každý přepínač má krátkou jednopísmennou podobu a většina z nich i dlouhou podobu. V příkazové řádce jsou zadávány ve formátu *-P argument* nebo *--přepínač=argument*, kde *argument* závisí na konkrétním přepínači. Modul přijímá následující přepínače:

- **N** nebo **nodes-num** – určuje počet výpočetních uzlů, mezi které bude modul síťové toky rozdělovat. Parametr očekává argument v celočíselném formátu.
- **H** nebo **hash-distr** – bude použito schéma rozhazování rozptylováním
- **R** nebo **rand-distr** – bude použito schéma náhodného rozhazování

- **L** nebo **lines-distr** – bude použito schéma rozhazování podle topologie
- **i** – parametr společný pro `NEMEA` moduly určující specifikátor všech rozhraní modulu. Využíván je knihovnou *libtrap* při inicializaci a očekává argument v podobě řetězce.
- **S** nebo **statistics** – umožňuje zapnout periodický výpis statistik modulu. Parametr očekává argument v celočíselném formátu určující délku periody výpisu v sekundách.
- **h** – parametr společný pro `NEMEA` moduly, který zobrazí formátovanou nápovědu pro modul.

Z uvedených přepínačů je vždy povinný jeden z množiny *{H,R,L}*, které určují schéma rozhazování, parametr *N* je povinný vždy, parametr *S* je volitelný (při jeho absenci modul statistiky nevypisuje vůbec) a parametr *i* je povinný vždy. Parametry modulu v tomto formátu jsou načteny a rozpoznány pomocí funkce *getopt()* nebo pokud je v operačním systému dostupná verze i pro dlouhé přepínače *getopt_long()*.

Řetězec popisující specifikátor rozhraní u parametru *i* má předepsaný formát $rozrhani_1, rozhrani_2, rozhrani_3, ..., rozhrani_x$, kde každé $rozhrani_i$ pro i od 1 do $x$ popisuje dvojtečkou oddělené parametry jednoho komunikačního rozhraní modulu. Prvním parametrem je typ rozhraní, který může být jedno z *{t,u,f}* pro `TCP`, `UNIXSOCKET` nebo `FILE` rozhraní. Druhý parametr souvisí s typem rozhraní. V případě `TCP` jsou zadány adresa a port, u `UNIXSOCKET` rozhraní jde o jméno soketu a u `FILE` rozhraní je to jméno souboru. Další parametry jsou volitelné a souvisí s nastavením časového limitu na rozhraní (tzv. timeout) a nebo vypnutí mezipaměti při odesílání a přijímání dat. Rozhraní jsou v pořadí všechna vstupní a poté výstupní. Specifikátor rozhraní pro modul s jedním vstupním `TCP` rozhraním a jedním výstupním `UNIXSOCKET` rozhraním vypadá například takto: *t:1.2.3.4:7500,u:output_sock*.

## 3.4 Výstup modulu

Pokud je modul `Flow scatter` spuštěn s přepínačem pro statistiky, vypisuje periodicky na standardní výstup formátovanou tabulku s přehledy statistik. Pro každé samostatné vlákno, které zpracovává data ze vstupního rozhraní, jsou zobrazeny počty přijatých toků z kolektoru a odeslaných toků na jednotlivé výpočetní uzly. Procentuální vyjádření pomáhá posoudit rovnoměrnost rozdělení. Při rozhazování rozptylováním je navíc zobrazeno, kolikrát byl tok odeslán na jeden, dva a tři uzly z důvodu více rozptylovacích funkcí.

Implicitním výstupem každého `NEMEA` modulu jsou statistiky o všech jeho rozhraních. Statistiky jsou dostupné v JSON formátu z `UNIXSOCKET` rozhraní modulu. Tyto statistiky obsahují: i) počty přijatých zpráv pro vstupní rozhraní, ii) počty odeslaných a zahozených zpráv pro výstupní rozhraní. Tyto

statistiky se dají vyčíst pomocí `trap_stats` nástroje, který je součástí `NEMEA` systému a jsou zároveň používány modulem `Supervisor`, pro zobrazování informací o celém systému.

## 3.5 Algoritmus modulu

Po spuštění modulu je nejprve inicializována knihovna *libtrap* a s ní i komunikační rozhraní modulu. Vstupní rozhraní modulu se automaticky pokouší připojit na výstupní rozhraní `IPFIXcolu` (nebo na jiný zdroj síťových toků podle zadaného portu/soketu) a výstupní rozhraní naopak vyčkávají na připojení dalších `NEMEA` modulů (nejen detekčních) běžících na výpočetních uzlech. Dále jsou rozpoznány argumenty programu (tj. přepínače modulu) vyjmenované v Sekci 3.3. Poté je naalokována paměť pro statistiky a jsou vytvořena 2 nová pracovní vlákna, kde jedno zpracovává základní síťové toky a druhé zpracovává toky obohacené o SIP informace a podle zvoleného typu rozhazování vykonávají jednu z následujících funkcí:

- **links_thr_routine()** – při rozhazování podle topologie vykonávají obě pracovní vlákna
- **rand_thr_routine()** – při náhodném rozhazování vykonávají obě pracovní vlákna
- **basic_hash_thr_routine()** – při rozhazování rozptylováním provádí pracovní vlákno zpracovávající základní síťové toky
- **sip_hash_thr_routine()** – při rozhazování rozptylováním vykonává pracovní vlákno zpracovávající obohacené síťové toky o SIP informace

Základní kostra těchto čtyř funkcí je stejná. Nejprve se pro vstupní rozhraní, ze kterého vlákno data rozhazuje a korespondující výstupní rozhraní vytvoří *UniRec* šablony, nastaví se parametr vlákna pro asynchronní zrušení a poté následuje hlavní cyklus. Ten zahrnuje přijetí zprávy ze vstupního rozhraní, kontrolu chyb, určení čísla cílového uzlu dané zprávy (síťového toku), odeslání zprávy, kontrolu chyb a aktualizaci statistik o rozložení. Implementace určení cílového uzlu se liší podle rozhazovacího schématu a jsou popsány v následující Sekci 3.5.1.

Hlavní vlákno modulu se po dokončení inicializace v cyklu stará o vypisování statistik na standardní výstup (pokud byl zadán parametr $S$). Střídá pasivní čekání pomocí funkce *sleep()* a kontrolu uplynutého času. Po uplynutí stanovené periody parametrem $S$ jsou vypsány statistiky. Hlavní vlákno se také stará o reakci na přijaté signály z operačního systému pomocí funkce *signal_handler()*. Na signály `SIGINT`, `SIGQUIT` a `SIGTERM` hlavní vlákno ukončí pracovní vlákna zpracovávající příchozí síťové toky, vypíše naposledy statistiky, provede uvolnění paměti knihoven a ukončí celý modul.

### 3.5.1 Určení cílového výpočetního uzlu

Ve funkci *rand_thr_routine()*, kterou pracovní vlákna vykonávají při náhodném rozhazování, je po přijetí zprávy určen cílový výpočetní uzel na základě náhodně vygenerovaného čísla. S cílem rovnoměrného rozložení toků byla pro experimentální účely generována čísla uzlů dopředu pomocí programu *R* [19] a vygenerovaná posloupnost čísel je modulem `Flow scatter` načtena při inicializaci do pole čísel. Vlákno pro určení cílového uzlu tedy jen použije hodnotu z pole, kde index prvku se rovná aktuálnímu počtu zpracovaných síťových toků. Bylo otestováno, že i pomocí C funkce *srand()* by bylo dosaženo rovnoměrného rozložení.

Funkce *links_thr_routine()* při rozhazování podle topologie z přijaté zprávy pomocí funkce *ur_get()* nejdříve získá hodnotu políčka `LINK_BIT_FIELD`. Tato hodnota má právě jeden bit nastavený na 1 a ten určuje linku, ze které byl tok exportován. Ke zjištění bitu je použit logický *AND* hodnoty s mocninami čísla dvě. Pořadí bitu určuje číslo výpočetního uzlu, na který tato funkce odešle aktuální síťový tok.

Pro rozhazování rozptylováním jsou potřebné odlišné funkce pro každé pracovní vlákno, protože detekční moduly pracující se základními toky vyžadují rozptylování třemi způsoby (podle zdrojové IP adresy, podle cílové IP adresy a podle uspořádané dvojice IP adres), kdežto detekční modul pracující s toky obohacenými o SIP informace vyžaduje rozptylování pouze podle zdrojové IP adresy. Toto bylo diskutováno v Sekci 2.5.1. Vlákno 1 z Obr. 3.2 používá funkci *basic_hash_thr_routine()* a vlákno 2 funkci *sip_hash_thr_routine()*.

Po přijetí zprávy ve funkci *basic_hash_thr_routine()* jsou pomocí funkce *ur_get()* získány hodnoty položek `SRC_IP` a `DST_IP` jakožto zdrojové a cílové IP adresy síťového toku. Pokud je zdrojová adresa IPv4, je převedena její *UniRec* reprezentace na *uint32* a spočítána její haš. Pokud je zdrojová adresa IPv6, je spočítána haš celých 128 bitů. Výsledná haš je hodnota *uint32* a k získání čísla výpočetního uzlu je použit logický *AND* s *počet_uzlů - 1*. Rozsah výsledku je 0 až *počet_uzlů - 1*, což odpovídá rozsahu indexů výstupních rozhraní. Stejným způsobem je spočítáno i číslo výpočetního uzlu pro cílovou IP adresu. K získání haše uspořádané dvojice IP adres je nejprve zjištěno, zda jsou adresy IPv4 nebo IPv6 kvůli délkám. IP adresy jsou za sebe zřetězeny v pořadí menší, větší. Z bloku paměti velikosti 64 bitů (pro IPv4) nebo 256 bitů (pro IPv6), kde jsou adresy zřetězeny, je spočítána *uint32* haš a stejným způsobem vypočítáno číslo výpočetního uzlu, jako pro samostatné adresy. Když jsou vypočítána všechna 3 čísla výpočetních uzlů, jsou mezi sebou porovnána a síťový tok je odeslán na minimálně jeden a maximálně na *min(počet_uzlů, 3)* uzly.

Funkce *sip_hash_thr_routine()* funguje velice podobně, s tím rozdílem, že počítá pouze jednu haš ze zdrojové IP adresy a nemusí porovnávat výsledná čísla uzlů před odesláním. Následující sekce popisuje implementované rozptylovací funkce.

### 3.5.2 Použité rozptylovací funkce

První z použitých funkcí je `CRC32`. Tato rozptylovací funkce se nejčastěji používá k detekci chyb při přenosu dat, ale testy ukázaly, že má dobré vlastnosti i pro rovnoměrné rozptylování IP adres. Princip této funkce je založený na polynomiálním dělení, kde zbytek po dělení se rovná kontrolnímu součtu (haši) přenášených dat. Aritmetické operace s polynomy jsou prováděny *mod 2*, z čehož plyne veliká výhoda implementace této funkce pomocí logického *XOR* a bitových posunů. Délka kontrolního součtu (haše) se v bitech rovná stupni použitého polynomu (v tomto případě 32 bitů). Pro účely této práce byla použita implementace funkce z [20], konkrétně základní verze *crc32a()* a její optimalizovaná verze *crc32c()*. Základní verze provádí dělení ve dvou cyklech, kde vnější cyklus vstupní data funkce rozděluje na bloky o velikosti jednoho bajtu a vnitřní cyklus bit po bitu pro každý bajt provádí *XOR* a bitový posun (dělení). Optimalizovaná verze využívá vyhledávací tabulku, která obsahuje předpočítané výsledky pro jeden bajt – 256 prvků, a proto může být vynechán vnitřní cyklus oproti základní verzi. Tabulka je vyplněna během inicializace modulu `Flow scatter`.

Další použitou rozptylovací funkcí je *One-at-a-Time* zveřejněná Bobem Jenkinsem. Funkce vychází z jednoduché aditivní rozptylovací funkce, která v cyklu všechny bajty vstupních dat sečte (kombinující krok) a na závěr provede operaci *modulo prvočíslo*. Funkce *One-at-a-Time* provádí stejný kombinující krok a navíc v každém cyklu provádí krok smíchání, kde je k aktuální haši přičten její levý bitový posun a poté pomocí logického *XOR* ještě pravý bitový posun. Délka výsledné haše je také 32 bitů. Tato rozptylovací funkce je v dalších sekcích nazývána jen *Jenkins*. V modulu `Flow scatter` byla použita implementace dostupná z [21].

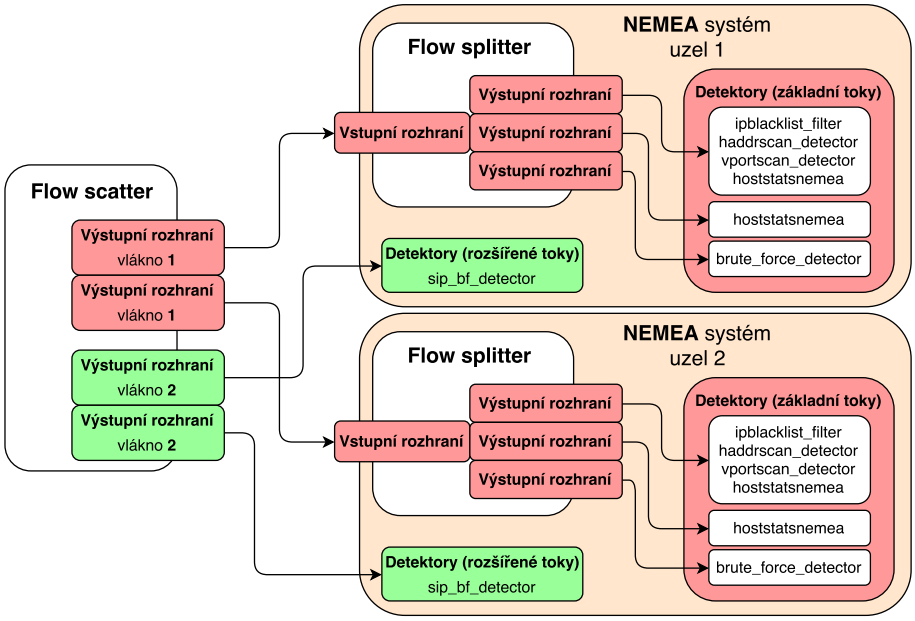
### 3.5.3 Optimalizace rozdělení síťových toků

Aby se zamezilo redundantnímu výpočtu při rozhazování rozptylováním, kdy může být jeden tok zpracován až třemi uzly najednou, bylo provedeno několik úprav. Ve funkci *basic_hash_thr_routine()* byla *UniRec* šablona, která je použita ve výstupních rozhraních, rozšířena o políčko `IP_BIT_FIELD` typu *uint8*. Příchozí zpráva je odeslána s tímto políčkem navíc. Z osmi bitů jsou využity nejnižší tři: `0x1`, `0x2`, `0x4`. Bit `0x1` je ve zprávě před odesláním nastaven na 1, pokud číslo uzlu určila haš uspořádané dvojice IP adres. Bit `0x2` je nastaven na 1, pokud číslo uzlu určila haš cílové IP adresy a bit `0x4` je nastaven na 1, pokud číslo uzlu určila haš zdrojové IP adresy. Políčko `IP_BIT_FIELD` tedy říká, která skupina detekčních modulů na konkrétním výpočetním uzlu by měla zpracovat aktuální síťový tok.

Pouhé přidání políčka by ale nestačilo, protože detekční moduly na výpočetních uzlech jsou přes vstupní `TCP` rozhraní připojeny přímo na výstupní rozhraní `Flow scatteru`. Proto byl naimplementován jednoduchý `NEMEA` mo-

dul `Flow splitter` v jazyce C, který má jedno vstupní a tři výstupní rozhraní. Je umístěn na každém výpočetním uzlu spolu s detekčními moduly a svým vstupním rozhraním se připojuje na výstupní rozhraní vlákna 1 modulu `Flow scatter` (vlákna zpracovávajícího základní toky). Na jeho výstupní rozhraní jsou připojeny tři skupiny detekčních modulů ukázané na Obr. 2.5, které zpracovávají základní síťové toky.

Modul `Flow splitter` v cyklu přijímá zprávy ze vstupního rozhraní, pomocí funkce *ur_get()* získá hodnotu políčka `IP_BIT_FIELD` a podle nastavených třech nejnižších bitů pozná, které skupině (na která výstupní rozhraní) musí zprávu přeposlat. Je-li nastaveno bitů více, přepošle tok na více výstupních rozhraní. Na cílovém výpočetním uzlu už se ale nejedná o redundanci. Obr. 3.3 navazuje na předchozí Obr. 3.2 a ukazuje zapojení rozšířené o tento modul. Díky tomuto rozšíření je síťový tok duplikován pouze na komunikační úrovni jak bylo diskutováno v Sekci 2.5.2 a není zpracován stejným detekčním modulem vícekrát.




Obrázek 3.3: Rozšíření konfigurace o modul `Flow splitter` používaném při rozhazování rozptylováním. Z přidané informace v základních síťových tocích modul pozná, které skupině detekčních modulů tok pošle. Zamezuje se tím redundantnímu výpočtu.

Kapitola 4

# Testování

## 4.1 Testovací prostředí

Pro testování implementovaného prototypu byl vytvořen virtuální stroj simulující distribuované prostředí. Systémové prostředky virtuálního stroje, na kterém vše běželo, byly: 16 jader CPU, 24 GB RAM, 2 TB disk. Celý stroj byl vyhrazen pouze pro účely této práce.

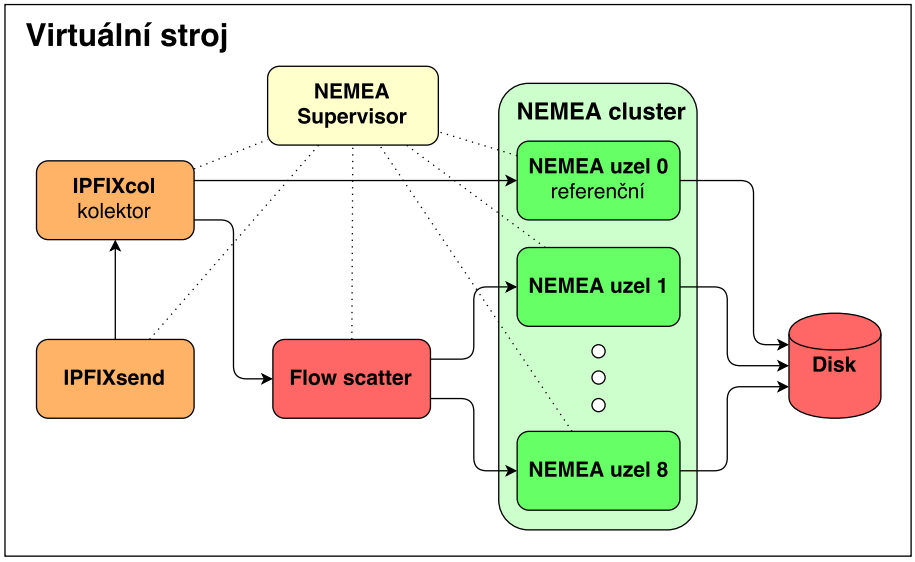
Obr. 4.1 ukazuje všechny hlavní komponenty prostředí. Oranžově jsou zvýrazněny komponenty `IPFIXcol` a `IPFIXsend`, které slouží jako zdroj dat (síťových toků). `IPFIXsend` je nástroj, který umožňuje přehrávání uložených datových souborů ve formátu *IPFIX* v reálném čase podle časových značek jednotlivých síťových toků. Přehrávání dat v reálném čase je důležité například z hlediska detekčních modulů, které vyhodnocují statistiky o provozu v časových oknech podle systémového času. `IPFIXcol` byl v tomto případě použit k přijetí dat v *IPFIX* formátu a následném odeslání na výstupní rozhraní v *UniRec* formátu používaného `NEMEA` moduly.

Zelenou barvou jsou na obrázku zvýrazněny simulované výpočetní uzly, na kterých běží `NEMEA` systém s detekčními moduly. Na všech uzlech byla nasazena stejná množina modulů. `Uzel 0` je nazýván referenční, protože jako jediný zpracovává všechna data od `IPFIXcolu` a výsledky z tohoto uzlu slouží jako referenční. Ostatní (distribuované) uzly 1 až 8 zpracovávají jen podmnožiny dat. Uzlů bylo vytvořeno 8 z důvodu rozhazování podle topologie. V síti CESNET2, ve které byla testovací data zachytávána, je aktuálně 8 monitorovaných linek, ze kterých se data exportují. Při rozhazování podle topologie tedy každý výpočetní uzel obdržel data z jedné monitorované linky.

Červenou barvou jsou zvýrazněny komponenty, které jsou cílem testování: i) modul `Flow scatter` rozdělující data mezi výpočetní uzly 1 až 8, ii) úložiště, na které všechny moduly ukládají své výsledky (zejména detekované události).

Všechny zmíněné komponenty jsou pro jednoduché ovládání spuštěny pod modulem `Supervisor`, který je zobrazený žlutě. Pomocí vytvořené konfigurace, která je popsána v Sekci 4.3, umožňuje `NEMEA` moduly na všech simu-

lovaných uzlech i ostatní komponenty spouštět, zastavovat, měnit konfiguraci za běhu, ale také zobrazovat jejich využití systémových prostředků a statistiky rozhraní modulů. Tyto informace byly používány během celého testování. Obr. D.3 ukazuje použití modulu `Supervisor` pro zobrazení stavu konfigurace.




Obrázek 4.1: Virtuální stroj pro testování implementovaného prototypu. Obsahuje `IPFIXcol` a `IPFIXsend` pro přehrávání dat, 9 výpočetních uzlů, z toho 8 zpracovává podmnožinu dat od `Flow scatteru` a úložiště nahlášených událostí. Všechny komponenty jsou spravovány modulem `NEMEA Supervisor`

## 4.2 Potřebné úpravy

Před začátkem testování i během něj byly objeveny u některých komponent nedostatky a problémy, které musely být odstraněny.

### 4.2.1 Úpravy detektorů

Původní metoda pro detekci vertikálního skenování portů publikovaná v [18] a implementovaná modulem `vportscan_detector`, si pro každou unikátní dvojici IP adres ukládá unikátní cílové porty. Pokud se cílový port opakuje, z tabulky portů je vyhozen. Při vyhodnocování prvních experimentů s rozhazováním podle topologie bylo zjištěno, že některé síťové toky jsou exportovány z více linek najednou. Toto chování způsobovalo vyhazování duplikovaných portů z tabulky u referenčního detektoru, který zpracovával všechna data bez rozdělení (tedy na uzlu 0). Pro další experimenty byl opakovaný port v tabulce ponechán.

Modul `hoststatsnemea`, který zpracovává základní síťové toky a detekuje na základě statistik o IP adresách musel být upraven kvůli metodě rozhazování rozptylováním. V Sekci 2.5.1 bylo řečeno, že modul musí obdržet všechny toky se stejnou IP adresou bez ohledu na to, zda je cílová nebo zdrojová a tudíž je pro něj nutné rozptylování podle zdrojové i cílové adresy. Z každého toku si poté ukládá/aktualizuje statistiky o obou adresách. Pokud ale rozptylovací funkce u modulu `Flow scatter` určí rozdílné uzly pro zdrojovou a cílovou adresu, musí si modul uložit/aktualizovat statistiky pouze u jedné z adres, protože ta druhá na daný uzel nepatří. V takovém případě obdrží modul *UniRec* zprávu, ve které je u políčka `IP_BIT_FIELD` nastaven jen druhý nebo třetí bit zprava (viz Sekce 3.5.3) místo obou dvou zároveň. Proto byl zdrojový kód modulu upraven takto: i) pokud je nastaven třetí bit zprava, ulož/aktualizuj statistiky o zdrojové IP adrese, ii) pokud je nastaven druhý bit zprava, ulož/aktualizuj statistiky o cílové IP adrese.

### 4.2.2 Ostatní úpravy

Z výkonnostních důvodů byl nástroj `IPFIXsend` implementován tak, že celý datový soubor nejdříve načetl do paměti RAM a poté začal přehrávat. Datové sady použité během experimentů ale měly desítky i stovky GB, a proto nebylo možné `IPFIXsend` v této podobě použít. Byl proto upraven, aby data načítal a přehrával postupně.

Detektory skenování portů `vportscan_detector` a `haddrscan_detector` mohou hlásit mnoho událostí za minutu. Proto se za ně zapojují agregační moduly, které v určeném časovém okénku události agregují podle nahlášené IP adresy. Agregační moduly jsou psané v jazyce *Python* a IP adresa z každé přijaté zprávy byla hledána v poli lineárním procházením. Během experimentů agregátor nestíhal zprávy odebírat a detektor musel zprávy zahazovat. Ukládání adres do pole v agregátoru bylo nahrazeno ukládáním do slovníku, což radikálně snížilo výpočetní náročnost a odstranilo tak zahazování na straně detektorů.

## 4.3 Konfigurace prostředí

Pro snadné ovládání testovacího prostředí jsou všechny komponenty prostředí centrálně spravovány modulem `Supervisor`. Modul `Supervisor` přijímá konfigurační soubor ve formátu *XML* a i když je primárně určen pro binární `NEMEA` moduly, lze ho využit ke spuštění a monitorování ostatních binárních programů nebo také skriptů – v tomto případě `IPFIXcol`, `IPFIXsend` a `NEMEA` moduly psané v jazyce *Python*.

Výpis 4.1 ukazuje prázdnou šablonu *XML* konfigurace pro jednu komponentu (nejen `NEMEA` modul). Povinné elementy jsou *name*, *path* a *enabled*, které určují v tomto pořadí unikátní jméno modulu a zároveň jméno budoucího procesu, poté cestu ke spustitelnému souboru (binární soubor nebo například

interpret) a poslední *enabled* je přepínač, který určuje, zdali modul poběží nebo ne. Elementy *params* a *module-restarts* dále určují parametry modulu předané formou argumentů programu a maximální počet automatických restartů modulu za minutu. Konečně element *trapinterfaces* je použit `NEMEA` moduly a popisuje seznam komunikačních rozhraní modulu. Každé rozhraní modulu (element *interface*) je určeno směrem (element *direction*), typem (element *type*) a parametry (element *params*). Z informací o komunikačních rozhraních uvnitř elementu *trapinterfaces* poskládá modul `Supervisor` argument parametru *i*, který byl popsán v Sekci 3.3 a předá jej spuštěnému modulu jako jeden z program argumentů.

```
<module>
  <name/>
  <path/>
  <enabled/>
  <params/>
  <module-restarts/>
  <trapinterfaces>
    <interface>
      <direction/>
      <type/>
      <params/>
    </interface>
    ...
  </trapinterfaces>
</module>
```

Výpis 4.1: Ukázka prázdné šablony *XML* konfigurace používané modulem `NEMEA Supervisor`. Každá kompnenta testovacího prostředí je popsána touto *XML* konfigurací a to umožňuje centrální ovládání a monitorování celého prostředí.

### 4.3.1 Konfigurace jednoho výpočetního uzlu

Prvním krokem k vytvoření celé konfigurace byla konfigurace jednoho simulovaného `NEMEA` uzlu. Konfigurace referenčního uzlu obsahuje šest detekčních modulů vyjmenovaných v Sekci 2.5.1 a pro vyhodnocování výsledků je na každý z nich napojen logovací modul `Logger`, který ukládá přijaté zprávy (v tomto případě nahlášené události) do souboru ve formátu *CSV*. Všechna komunikační rozhraní modulů jsou typu `UNIXSOCKET` pro komunikaci na lokálním stroji. Modul `Logger` má pouze jedno vstupní rozhraní, které se spojuje s korespondujícím výstupním rozhraním detektoru. Použité detektory mají mimo jedno výstupní rozhraní také jedno vstupní, kterým se na referenčním uzlu připojují přímo na výstupní rozhraní `IPFIXcol` kolektoru. Parametry

jednotlivých detekčních modulů (předávané pomocí *params* elementu *XML* konfiguračního souboru) byly převzaty z produkčního nasazení modulů.

Konfigurace ostatních výpočetních uzlů se liší v přidaném rozdělovacím modulu `Flow splitter`. Detekční moduly na těchto uzlech se připojují na výstupní rozhraní modulu `Flow splitter` nebo `Flow scatter` podle druhu síťových toků. Rozdíl v propojení byl ukázán na Obr. 3.3. Na Obr. D.1 je ukázána část konfigurace dvojice `NEMEA` modulů.

### 4.3.2 Celá konfigurace

Konfigurace celého prostředí obsahuje jeden referenční uzel, osm (distribuovaných) uzlů zpracovávající rozdělená data, komponenty pro přehrávání dat a `Flow scatter`. Nástroj `IPFIXsend` posílá přehrávaná data v *IPFIX* formátu na *TCP* port 4739, který je nastavený také v konfiguraci `IPFIXcol` kolektoru. Toto je jediná část komunikace komponent, která neprobíhá přes `UNIXSOCKET` komunikační rozhraní implementovaná knihovnou *libtrap*. Kolektor poté přijatá data posílá na svá dvě výstupní rozhraní v *UniRec* formátu. Modul `Flow scatter` má dvě vstupní rozhraní připojená na výstupní rozhraní kolektoru a spolu s osmi výpočetními uzly z toho plyne 16 výstupních rozhraní `Flow scatteru`. Mezi prvních osm výstupních rozhraní modul rozhazuje základní síťové toky a jsou k nim tudíž připojeny moduly `Flow splitter` z každého distribuovaného uzlu. Mezi druhých osm výstupní rozhraní rozhazuje modul síťové toky obohacené o SIP informace a jsou na ně připojeny detektory `sip_bf_detector` z distribuovaných uzlů.

Výstupní rozhraní všech `NEMEA` modulů mají v konfiguraci nastavený *timeout* na hodnotu `TRAP_WAIT`, tzn. pokud vstupní rozhraní na ně připojené nestíhá zprávy odebírat, výstupní rozhraní blokuje, dokud nejsou zprávy odebrány. V praxi se většinou hlavně u kolektoru používá neblokující režim a neodeslané zprávy jsou zahozeny, ale pro účely testování byl nastaven tento parametr, aby nedošlo ke ztrátě žádné nahlášené události a byly správně vyhodnoceny výsledky detekce.

Protože celá konfigurace obsahuje 119 komponent, byl vytvořen skript pro generování konfigurace celého testovacího prostředí. Tato konfigurace v podobě konfiguračního souboru je vstupem modulu `Supervisor`. Každá komponenta je popsána částí *XML* konfigurace podle šablony ve Výpisu 4.1. Výsledný konfigurační soubor je možné ovlivnit pomocí parametrů skriptu (např. počet uzlů). Po provedení změny (například parametru detektoru) je tedy možné snadno vygenerovat celkovou konfiguraci prostředí. Tento skript i ukázky konfigurace jsou součástí přiloženého CD.

## 4.4 Experimenty

Tato sekce popisuje jednotlivé experimenty, které byly provedeny za účelem ověření vlastností realizovaného prototypu dle úvodních požadavků. Pro ex-

perimenty byla použita reálná data zachycená v síti CESNET2, která byla z důvodu zachování soukromí uživatelů pseudonymizována. Celkem bylo zachyceno přes pět miliard síťových toků v deseti souborech formátu *IPFIX* během dvou měsíců. V průměru se jednalo o 60 000 síťových toků/s. Následující podsekce popisují postup a výsledky vyhodnocení rovnoměrnosti, rychlosti (efektivity) a dopadu na výsledky detekce jednotlivých způsobů rozhazování.

### 4.4.1 Rovnoměrnost rozhazování

Testy rovnoměrnosti rozdělení síťových toků mezi výpočetní uzly byly provedeny přehráváním tří různých `IPFIX` souborů, které obsahovaly celkem přibližně půl milionu zachycených síťových toků. Výsledky byly vyčteny z výstupu modulu `Flow scatter` pomocí přepínače *-S* ukázaného na Obr. D.2.

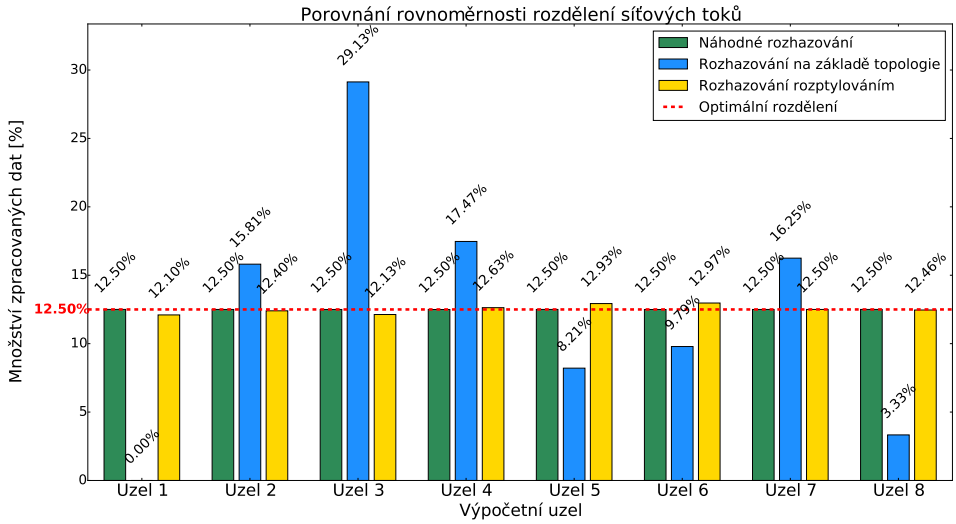
Na Obr. 4.2 je vidět srovnání výsledků testu rovnoměrnosti všech tří způsobů rozhazování. Červená konstantní křivka určuje optimální rozdělení, který je pro 8 výpočetních uzlů 12.50 %. Náhodné rozhazování optimálního rozdělení dosahuje, protože používá statistické rovnoměrné rozdělení. Toto byl cíl náhodného rozhazování – síťové toky rozdělit rovnoměrně mezi výpočetní uzly s předpokladem nezávislosti jednotlivých toků.

Výsledek rozhazování podle topologie je velice nevyvážený. Důvodem je rychlost monitorovaných linek, která se liší a tím i množství dat, které přes linky prochází. Z grafu je vidět, že v průběhu záchytu použitých `IPFIX` souborů se z první linky neexportovala žádná data. Naopak data ze třetí linky představují skoro třetinu celého objemu.

Pro rozhazování rozptylováním byly otestovány tři rozptylovací funkce popsané v Sekci 3.5.2. Tabulka 4.1 ukazuje procentuální rozdělení síťových toků mezi výpočetní uzly v závislosti na vstupu rozptylovací funkce a také na použité funkci. Při rozhazování základních síťových toků modul `Flow scatter` používá jako vstup rozptylovací funkce postupně zdrojovou IP adresu, cílovou IP adresu a uspořádanou dvojici IP adres k určení čísel cílových uzlů. Tabulka ukazuje porovnání funkcí *CRC32* a *Jenkins*. Optimalizovaná verze *CRC32* používající předpočítanou tabulku neměla na rozdělení oproti neoptimalizované verzi vliv. Výsledky jsou zaokrouhleny na jedno desetinné místo. Z tabulky je vidět, že obě funkce dosahovaly srovnatelných výsledků, ale celkově byla lepší funkce *CRC32*, která je také ukázána v grafu 4.2. Rozhazování rozptylováním dosahuje rovnoměrnosti rozdělení, které se blíží optimálnímu rozdělení.

### 4.4.2 Rychlost rozhazování

Dalším důležitým aspektem rozdělování síťových toků výpočetním uzlům je rychlost nebo také propustnost modulu `Flow scatter`. K jejímu změření bylo potřebné zjistit, kolik maximálně síťových toků v podobě *UniRec* zpráv zvládne




Obrázek 4.2: Srovnání rovnoměrnosti rozdělení síťových toků použitím různých metod rozhazování: i) náhodné rozhazování dosahuje optima 12.5 %, ii) topologické rozhazování je ovlivněno rychlostí linek, iii) výsledky rozhazování rozptylováním se blíží optimu.

| Vstup | Funkce | Výpočetní uzly | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| src IP | CRC32 | 12.1 | 12.6 | 11.8 | 12.6 | 12.8 | 13.7 | 12.5 | 11.9 |
| | Jenkins | 11.7 | 14.2 | 11.5 | 12.6 | 13.2 | 12.7 | 12.3 | 11.9 |
| dst IP | CRC32 | 11.5 | 12.5 | 12.0 | 12.5 | 13.6 | 12.8 | 12.3 | 12.7 |
| | Jenkins | 11.8 | 13.2 | 11.7 | 12.9 | 13.0 | 11.8 | 12.4 | 13.1 |
| IP pair | CRC32 | 12.7 | 12.1 | 12.6 | 12.8 | 12.4 | 12.4 | 12.6 | 12.5 |
| | Jenkins | 12.8 | 12.3 | 12.4 | 12.5 | 12.7 | 12.8 | 12.4 | 12.2 |

Tabulka 4.1: Srovnání procentuálního rozdělení síťových toků mezi výpočetní uzly při rozhazování rozptylováním. Rozptylovací funkce určuje číslo uzlu na základě zdrojové (src) IP, cílové (dst) IP a uspořádané dvojice (pair) IP adres. Funkce *CRC32* dosahuje lepších výsledků.

modul zpracovat za určitý čas. Pro tento test byly vytvořeny dva jednoduché `NEMEA` moduly: `generator` a `receiver`. Účelem modulu `generator` je odeslat parametrem zadané množství *UniRec* zpráv maximální možnou rychlostí, kterou omezuje buď procesor nebo propustnost samotné knihovny *libtrap* implementující komunikační rozhraní modulů. Modul `generator` tedy v cyklu odesílá stejnou *UniRec* zprávu na jediné výstupní rozhraní. Modul `receiver` má jedno vstupní rozhraní a tím pouze přijme zprávu a nic dalšího s ní nedělá. Jeho účelem je simulovat odebrání zprávy z výstupního rozhraní modulu `Flow scatter`, jako by to byl například detekční modul.

Během testu byl měřen čas zpracování 20 milionů zpráv poslaných modulem `generator`. Každé měření bylo opakováno desetkrát a výsledný čas byl získán aritmetickým průměrem. Nejdříve byl změřen čas, za který modul `generator` zprávy odešle a modul `receiver` přijme, aby bylo jisté, že test neomezují tyto dva moduly. Modul `receiver` byl tedy připojen přímo na výstupní rozhraní modulu `generator`. Výsledný čas byl 6.3 sekund, což znamená skoro 3.2 milionu zpráv za sekundu.

Pro test propustnosti/rychlosti modulu `Flow scatter` byl napojen modul `generator` na první vstupní rozhraní modulu `Flow scatter`. Druhé vstupní rozhraní a s ním souvisejících 8 výstupních rozhraní nebylo nutné pro měření využít, protože obě vstupní rozhraní jsou v provozu obsluhována paralelně. První rozhraní bylo zvoleno z toho důvodem, že při náhodném a topologickém rozhazování obě pracovní vlákna vykonávají totožnou funkci, ale v případě rozhazování rozptylováním je funkce zpracovávající první vstupní rozhraní složitější díky výpočtu tří haší. Na každé výstupní rozhraní byl připojen modul `receiver`, který pouze odebíral zprávy. Výsledný čas byl měřen od začátku vykonávání hlavního cyklu pracovního vlákna po příjetí poslední zprávy.

Tabulka 4.2 ukazuje výsledný čas zpracování 20 milionů zpráv modulem `Flow scatter` při použití různých metod rozhazování síťových toků. Pro rozhazování rozptylováním byla měřena zvlášť každá testovaná rozptylovací funkce. Pravý sloupec tabulky potom ukazuje průměrnou propustnost modulu, která odpovídá počtu zpracovaných *UniRec* zpráv (síťových toků) za sekundu. Náhodné a topologické rozhazování je o tolik rychlejší, protože výpočet čísla cílového uzlu pro každý síťový tok je velice jednoduchý. Z testovaných rozptylovacích funkcí dosáhla nejlepšího výsledku optimalizovaná verze *CRC32*. Při testu rozhazování rozptylováním byly v posílané zprávě modulem `generator` zvoleny takové IP adresy, aby modul `Flow scatter` po výpočtu všech tří haší odeslal danou zprávu na 3 různé uzly, což odpovídá nejhoršímu případu.

| Metoda rozhazování | Čas [s] | Propustnost [toků/s] |
|---:|---:|---:|
| Náhodné | 9.1 | 2 222 222 |
| Topologické | 8.2 | 2 500 000 |
| Rozptylováním – Jenkins | 18.6 | 1 075 269 |
| Rozptylováním – CRC32 | 22.0 | 909 091 |
| Rozptylováním – CRC32opt | 15.4 | 1 298 701 |

Tabulka 4.2: Srovnání rychlosti zpracování 20 milionů zpráv modulem `Flow scatter`. Pro rozhazování síťových toků byly použity různé metody. Propustnost odpovídá počtu zpracovaných toků modulem za sekundu.

### 4.4.3 Dopad rozhazování na výsledky detekce

K vyhodnocení těchto výsledků bylo nutné zpracovat výstupní soubory `Logger` modulů zapojených za jednotlivými detekčními moduly. Soubory obsahovaly nahlášené události v *CSV* formátu (jeden řádek odpovídá jedné nahlášené události). Bylo nutné porovnat unikátní nahlášené události všech distribuovaných uzlů s nahlášenými událostmi detekčních modulů na referenčním uzlu, který zpracoval všechna data.

Pro automatizované zpracování nahlášených událostí byl pro každou detekční metodu vytvořen skript v awk(1) umožňující snadno zpracovat objemné soubory ve formátu *CSV*. Následující seznam popisuje jejich princip.

- Události nahlášené detektorem vertikálních skenů `vportscan_detector` jsou charakterizovány hlavně IP adresami (zdrojová pro útočníka, cílová pro oběť) a počtem skenovaných portů (konstantní číslo 50 rovné hranici pro detekci). Skript pro tento detektor si tedy ukládá unikátní dvojice cílové a zdrojové adresy a pro každou dvojici si ukládá počet skenovaných portů. Dále je také ukládán celkový počet nahlášených událostí a celkový počet skenovaných portů.

- Pro podobný detektor horizontálních skenů `haddrscan_detector` jsou v nahlášených událostech charakteristické zdrojová IP adresa (útočník), cílový port a počet skenovaných cílových IP adres. Pro každou unikátní dvojici zdrojové IP adresy a cílového portu jsou tedy ukládány počty skenovaných adres. Stejně jako u předchozího jsou ukládány i celkové počty nahlášených událostí a skenovaných cílových IP adres.

- Detekční modul útoků hrubou silou `bf_detector` ve svých nahlášených událostech hlásí zdroj útoku, cílový port určující napadenou službu (SSH, TELNET, RDP) a velikost útoku v počtu síťových toků. Pro tento detektor jsou uloženy unikátní zdrojové IP adresy zvlášť pro každou službu a k nim počty toků. Dále také ukládá celkový počet nahlášených událostí a toků využitých k útokům.

- Detekční modul pro detekci IP adres ze seznamů nebezpečných adres `ipblacklist_filter` nastavuje v nahlášených událostech identifikátor seznamu, na kterém se adresa nachází, zvlášť pro zdrojovou a cílovou IP adresu. Dále detektor provádí agregaci síťových toků se škodlivou IP adresou a v nahlášené události posílá počet škodlivých síťových toků. Skript kontroluje, zda je v nahlášené události nebezpečná zdrojová nebo cílová IP adresa a ukládá si unikátní IP adresy spolu s počtem síťových toků, ve kterých dané adresy figurovaly.

- Nahlášené události detekčním modulem `hoststatsnemea` mají nastavený typ události (útok hrubou silou, horizontální skenování, DoS útok, DNS amplifikační útok) v políčku `EVENT_TYPE`, dále intenzitu útoku

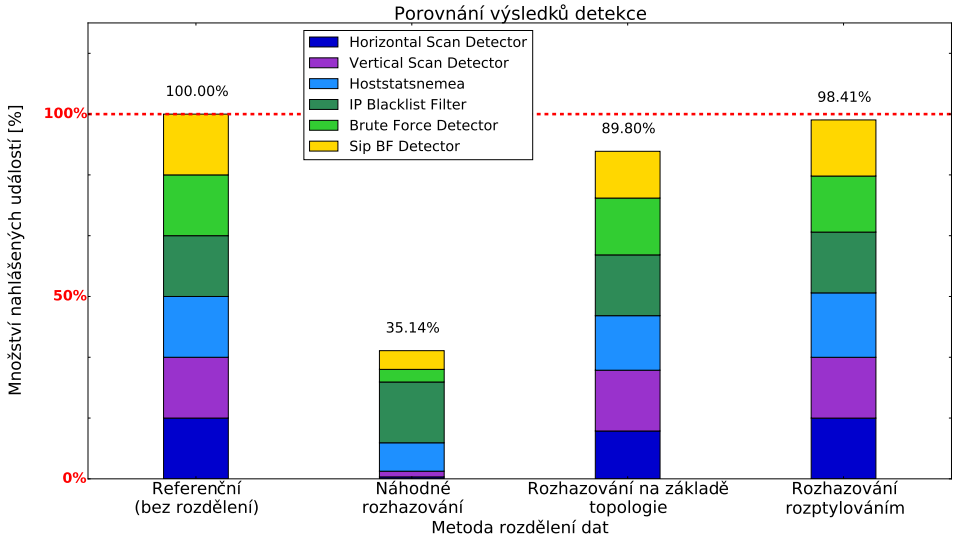
(např. počet pokusů prolomení) v políčku `EVENT_SCALE` a také IP adresy, pokud jsou pro daný typ útoku k dispozici. Skript pro zpracování těchto událostí rozlišuje unikátní IP adresy (zvlášť zdrojovou i cílovou) pro jednotlivé typy útoků a pro každou ukládá intenzitu útoků. Celkově také skript ukládá počet jednotlivých typů útoků a jejich intenzity.

- Poslední detektor `sip_bf_detector` pro detekci útoků hrubou silou na SIP protokol rozlišuje v nahlášených událostech políčkem `EVENT_TYPE` typ útoku (pokus o prolomení hesla, distribuovaný pokus o prolomení hesla, skenování uživatelských jmen), počet pokusů, IP adresy útočníka a SIP serveru a také použité uživatelské jméno. Při zpracování událostí si skript ukládá pro každý typ útoku unikátní dvojice IP adres, k nim unikátní uživatelská jména a počty pokusů.

Každý z těchto skriptů byl spuštěn nejdříve nad souborem s nahlášenými událostmi detektoru z referenčního uzlu a poté nad sloučenými událostmi stejného detektoru z distribuovaných uzlů. Výsledkem bylo porovnání výstupu každého skriptu pro referenční a sloučené distribuované uzly. Tento postup byl opakován zvlášť pro každou metodu rozhazování.

Graf na Obr. 4.3 ukazuje souhrnné výsledky detekce jednotlivých metod rozhazování. Každý sloupec představuje metodu rozhazování a barevné části sloupců představují detekční moduly. Hodnoty na svislé ose jsou v procentech a tato osa ukazuje kolik procent událostí bylo po použití jedné z metod rozhazování nahlášeno. Levý sloupec patří referenčnímu uzlu, který zpracoval data bez rozdělení a tudíž nahlásil 100 % událostí. Každý detektor hlásí různé počty událostí, ale ve výsledku musí mít stejnou váhu jako ostatní. Z tohoto důvodu byly počty nahlášených událostí normalizovány, tzn. i když detektor horizontálního skenování nahlásí desítky událostí za minutu (v součtu např. 500) a detektor pokusu o prolomení SIP jen jednu událost za minutu (v součtu např. 10), výška jejich dílku v referenčním sloupci je stejná.

Výsledek náhodného rozhazování je podle očekávání velice špatný, protože toto rozdělení nezachovává sémantické vazby mezi jednotlivými síťovými toky. Každé náhodné rozdělení by navíc mělo jiné výsledky detekce. Rozhazování na základě topologie dosahuje mnohem lepších výsledků detekce, ale jak naznačuje tmavě a světle modrá část sloupce, tato metoda rozdělení narušuje detekci distribuovaných útoků, obecně jde o útoky 1:N a N:1 (např. DDoS, horizontální skenování). Pokud se odehraje distribuovaný útok počítačů z různých států na jeden počítač v České republice, je velice pravděpodobné, že síťové toky budou exportovány z různých linek, následně zpracovány různými výpočetními uzly a útok zůstane nedetekován. Poslední sloupec ukazuje, že rozhazování rozptylováním podle různých pravidel dosahuje nejlepších výsledků. Nenahlášené události mohou být způsobeny například pravidelným čištěním struktur detektorů nebo dalšími časovými limity detekce.




Obrázek 4.3: Výsledky detekce pro různé metody rozhazování síťových toků mezi výpočetní uzly. Náhodné rozhazování porušuje sémantické vazby mezi toky a rozhazování podle topologie nemusí detekovat distribuované útoky. Nejlepších výsledků dosahuje rozhazování rozptylováním podle různých pravidel.

### 4.4.4 Shrnutí experimentů

Z provedených experimentů rovnoměrnosti, rychlosti a dopadu na výsledky detekce plyne pro každý typ rozhazování několik závěrů.

1. Náhodné rozhazování provádí optimální rozložení zátěže mezi výpočetní uzly díky statistickému rovnoměrnému rozložení. Zároveň je tento způsob rozhazování velice rychlý a proto má modul `Flow scatter` vysokou propustnost. Prioritou je ale v této práci výsledek detekce, který je v případě náhodného rozhazování velice špatný a navíc nepředvídatelný. Důvodem je porušení sémantických vazeb mezi jednotlivými síťovými toky.

2. Rovnoměrnost rozdělení síťových toků metodou rozhazování na základě topologie závisí na rychlosti monitorovaných linek a množství exportovaných toků z každé z nich. Rozdělení může být proto velice nevyvážené. Propustnost modulu `Flow scatter` je při použití této metody velice dobrá, protože i tato metoda určuje číslo výpočetního uzlu jednoduše. Výsledky detekce jsou přijatelné, ale může se stát, že tímto způsobem rozdělení síťových toků nebudou detekovány některé distribuované útoky. To je pozitivní výsledek, protože je potvrzena výhodnost shromažďování síťových toků ze všech monitorovaných linek na centrální kolektor a provádění analýzy nad všemi toky dohromady.

3. Metoda rozhazování rozptylováním dosahuje výborných výsledků detekce. Drobné odchylky oproti referenčním výsledkům mohou být způsobeny časovými limity detekčních modulů. Rovnoměrnost a rychlost rozdělení jsou u této metody rozhazování závislé na použité rozptylovací funkci. Z otestovaných rozptylovacích funkcí dosáhla v rovnoměrnosti i rychlosti rozdělení nejlepších výsledků optimalizovaná verze rozptylovací funkce *CRC32* využívající předpočítanou tabulku.

Metoda rozhazování rozptylováním byla po vyhodnocení všech experimentů vybrána jako nejlepší s použitím optimalizované verze *CRC32* rozptylovací funkce.

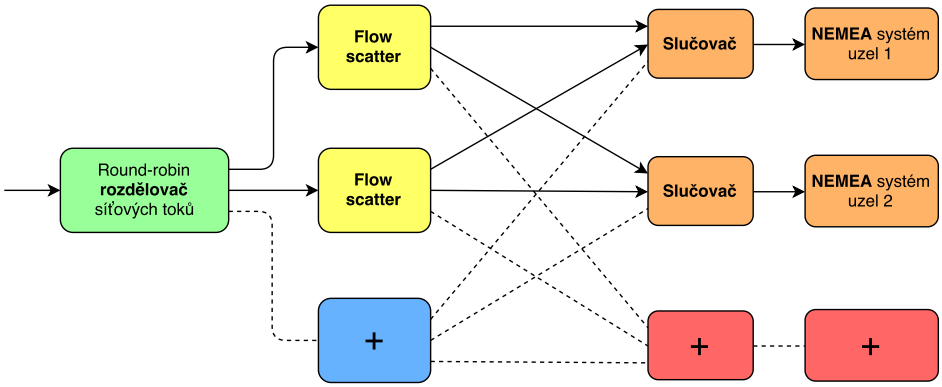
## 4.5 Škálovatelnost systému

Experimenty ukázaly, že modul `Flow scatter` je s finální metodou rozhazování rozptylováním schopný na testovacím stroji zpracovat (rozdělit) téměř 1.3 milionu síťových toků za sekundu. Dále bylo ale nutné zjistit, kolik síťových toků zpracuje jedna instance modulů, tedy jeden simulovaný uzel `NEMEA` systému. Za tímto účelem byly postupně napojeny všechny testované detekční moduly na výstupní rozhraní modulu `generator` pro zjištění jejich propustnosti.

Kvůli své komplexnosti měl nejnižší propustnost detektor `hoststatsnemea`, který zpracoval 20 milionů zpráv v průměru za 35,3 sekund, což znamená propustnost přibližně 567 tisíc toků za sekundu. Celková propustnost jednoho simulovaného uzlu `NEMEA` systému na testovacím stroji je tedy omezena tímto detekčním modulem. Naměřená propustnost jednoho uzlu zároveň udává hranici množství dat, pro které je nutný modul `Flow scatter`.

Z provedených experimentů bylo zjištěno, že paralelní zpracování v připraveném testovacím prostředí je omezeno propustností modulu `Flow scatter` a propustností jednoho simulovaného uzlu. Obr. 4.4 ukazuje možný způsob škálování paralelního zpracování, který by měl překonat propustnosti jednotlivých prvků. Pokud je příchozí množství síťových toků do modulu `Flow scatter` vyšší, než jeho propustnost, modrý prvek znázorňuje přidání dalšího modulu `Flow scatter` označeného žlutě. Pokud nestíhají data zpracovat výpočetní uzly, červené prvky znázorňují přidání dalšího výpočetního uzlu se systémem `NEMEA` označeného oranžově. Návrh škálování obsahuje nový prvek `Rozdělovač`, který příchozí toky střídavě rozděluje modulům `Flow scatter` například metodou *Round-robin*. Prvek `Slučovač` přijímá všechna data patřící konkrétnímu výpočetnímu uzlu od všech modulů `Flow scatter`, protože ty rozdělují data mezi stejnou množinu výpočetních uzlů.

Popsaný princip lze převést i na fyzické výpočetní uzly. Na konkrétních strojích lze stejným způsobem změřit propustnosti detektorů a modulu `Flow scatter` a uplatnit popsaný způsob škálování.




Obrázek 4.4: Návrh škálovatelné architektury paralelního zpracovaní síťových toků systémem NEMEA. Architektura pomáhá překonat omezující propustnosti prvků. Pokud je nedostačující propustnost modulu `Flow scatter`, přidá se další (modrý prvek). Pokud nestíhají množství síťových toků zpracovat výpočetní uzly, přidají se další (červené prvky).

# Závěr

Současné monitorovací systémy, které nepoužívají paralelní zpracování, přestávají být dostačující z důvodu zvyšujícího se množství dat ke zpracování. Existující řešení pro paralelní zpracování síťových toků se nesoustředí na sémantické vazby jednotlivých toků, a proto může být detekce bezpečnostních událostí narušena. Tato diplomová práce představila způsob paralelizace zpracování síťových toků, který sémantické vazby zachovává a nenarušuje tak výsledky paralelní detekce bezpečnostních událostí.

Tato diplomová práce popsala průběh návrhu, vývoje a testování rozšíření modulárního systému NEMEA pro analýzu síťového provozu a detekci anomálií. Realizované rozšíření umožňuje systém distribuovat na více výpočetních uzlů a paralelně zpracovávat velké množství síťových dat. Paralelizace je docíleno rozdělením proudu síťových toků mezi výpočetní uzly.

Výsledky experimentů této diplomové práce se podařilo publikovat v podobě konferenčního článku na mezinárodní konferenci AIMS 2017 [22].

## Shrnutí průběhu práce, vlastního přínosu a výsledků práce

Na začátku práce byl prostudován modulární systému NEMEA pro analýzu síťového provozu a jeho detekční metody. Dále byly nastudovány existující řešení paralelního zpracování velkého množství síťových dat za účelem detekce škodlivého provozu. Poté následoval návrh vhodného způsobu paralelizace zpracování a návrh různých metod rozdělení dat. Navržené metody rozdělení dat byly realizovány v podobě NEMEA modulu `Flow scatter`. V průběhu práce bylo vytvořeno a nakonfigurováno testovací prostředí simulující distribuované nasazení systému NEMEA. Byl vytvořen skript pro generování konfigurace celého prostředí. Tato konfigurace je využita modulem `Supervisor`, který umožňuje celé prostředí spravovat. Provedením řady experimentů byly ověřeny důležité vlastnosti realizovaného prototypu zvlášť pro každou metodu rozdělení dat.

Přínosem této práce je dříve neexistující modul `Flow scatter`, který rozděluje síťové toky mezi výpočetní uzly a umožňuje tak paralelní zpracování systémem NEMEA. Přínosná je také konfigurace celého prostředí pro modul `Supervisor`, která usnadňuje nasazení. Dalším přínosem jsou provedené experimenty s reálnými daty zachycenými v síti CESNET2. Pro vylepšení propustnosti paralelizace byl vytvořen návrh škálovatelné architektury systému NEMEA, která pomáhá překonat omezující propustnosti výpočetních uzlů i modulu `Flow scatter`.

Způsob paralelizace zpracování a metody rozdělení síťových toků popsané v této práci jsou aplikovatelné i na jiné detekční systémy. Stejně tak navržená škálovatelná architektura je použitelná i na komoditním HW. Výsledky této práce mohou tímto pomoci nedistribuovaným monitorovacím systémům zpracovat více dat bez nutnosti větších zásahů.

## Budoucí práce

V rámci pokračování této práce je plánováno testovací nasazení na fyzických výpočetních uzlech v reálném provozu. Dále bude také vytvořeno testovací prostředí pro navrženou škálovatelnou architekturu a bude provedena podobná série experimentů.

# Literatura


[1] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: A framework for network traffic analysis. In *2016 12th International Conference on Network and Service Management (CNSM)*, Říjen 2016, s. 195–201, doi: 10.1109/CNSM.2016.7818417.

[2] CESNET: Warden. [Citováno 2017-01-16]. Dostupné z: `https://warden.cesnet.cz`

[3] Švepeš, M.: Systém pro konfiguraci a monitorování distribuovaného systému NEMEA. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

[4] Velan, P.; Krejčí, R.: Flow Information Storage Assessment Using IPFIXcol. In *Proceedings of the 6th IFIP WG 6.6 International Autonomous Infrastructure, Management, and Security Conference on Dependable Networks and Services*, AIMS'12, Berlin, Heidelberg: Springer-Verlag, 2012, ISBN 978-3-642-30632-7, s. 155–158, doi:10.1007/978-3-642-30633-4_21.

[5] Claise, B.; Trammell, B.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, Září 2013, doi:10.17487/rfc7011.

[6] Stehlík, P.: Vizualizace síťových bezpečnostních událostí. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.

[7] Xinidis, K.; Charitakis, I.; Antonatos, S.; aj.: An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, ročník 3, č. 1, Leden 2006: s. 31–44, ISSN 1545-5971, doi:10.1109/TDSC.2006.6.

[8] Roesch, M.: Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, Berkeley, CA, USA: USENIX Association, 1999, s. 229–238.

[9] Sallay, H.; Alshalfan, K. A.; Fred, O. B.; aj.: A scalable distributed IDS Architecture for High speed Networks. In *IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.8*, Citeseer, 2009.

[10] Kim, N.; Jung, S.; Chung, T.: An Efficient Hash-Based Load Balancing Scheme to Support Parallel NIDS. In *Computational Science and Its Applications - ICCSA 2011 - International Conference, Santander, Spain, June 20-23, 2011. Proceedings, Part I*, Springer, Leden 2011, s. 537–549.

[11] Vallentin, M.; Sommer, R.; Lee, J.; aj.: The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware. In *Recent Advances in Intrusion Detection: 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007. Proceedings*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ISBN 978-3-540-74320-0, s. 107–126, doi:10.1007/978-3-540-74320-0_6.

[12] Paxson, V.: Bro: A System for Detecting Network Intruders in Real-time. *Comput. Netw.*, ročník 31, č. 23-24, Prosinec 1999: s. 2435–2463, ISSN 1389-1286, doi:10.1016/S1389-1286(99)00112-7.

[13] Apache: Hadoop. [Citováno 2017-01-15]. Dostupné z: `http://hadoop.apache.org`

[14] Apache: Spark. [Citováno 2017-01-15]. Dostupné z: `http://spark.apache.org`

[15] Fontugne, R.; Mazel, J.; Fukuda, K.: Hashdoop: A MapReduce framework for network anomaly detection. In *IEEE Conference on Computer Communications Workshops (INFOCOM)*, 2014, [Citováno 2017-01-15].

[16] Mai, J.; Sridharan, A.; Chuah, C. N.; aj.: Impact of Packet Sampling on Portscan Detection. *IEEE Journal on Selected Areas in Communications*, ročník 24, č. 12, Prosinec 2006: s. 2285–2298, ISSN 0733-8716, doi:10.1109/JSAC.2006.884027.

[17] Bartos, K.; Rehak, M.: Towards Efficient Flow Sampling Technique for Anomaly Detection. In *Proceedings of the 4th International Conference on Traffic Monitoring and Analysis*, TMA'12, Berlin, Heidelberg: Springer-Verlag, 2012, ISBN 978-3-642-28533-2, s. 93–106, doi:10.1007/978-3-642-28534-9_11.

[18] Cejka, T.; Svepes, M.: Analysis of Vertical Scans Discovered by Naive Detection. In *Management and Security in the Age of Hyperconnectivity: 10th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2016*, Munich, Germany: Springer International Publishing, 2016, ISBN 978-3-319-39814-3, s. 165–169, doi:10.1007/978-3-319-39814-3_19.

[19] The R Project for Statistical Computing. [Citováno 2017-02-10]. Dostupné z: `https://r-project.org/`

[20] CRC32. [Citováno 2017-02-24]. Dostupné z: `http://hackersdelight.org/hdcodetxt/crc.c.txt`

[21] Jenkins, B.: One-at-a-Time Hash. [Citováno 2017-02-24]. Dostupné z: `http://burtleburtle.net/bob/hash/doobs.html`

[22] Svepes, M.; Cejka, T.: Making flow-based security detection parallel. In *11th International Conference on Autonomous Infrastructure, Management and Security, AIMS 2017*, jul 2017, [Přijato, bude publikováno v červenci 2017].

Příloha A

# Seznam použitých zkratek

**API** Application programming interface

**CPU** Central processing unit

**CRC** Cyclic redundancy check

**CSV** Comma-separated values

**DDoS** Distributed denial of service

**DoS** Denial of service

**DNS** Domain name system

**FTP** File transfer protocol

**HIDS** Host intrusion detection system

**HTTP** Hypertext transfer protocol

**ICMP** Internet control message protocol

**IDS** Intrusion detection system

**IP** Internet protocol

**IPFIX** Internet protocol flow information export

**JSON** Javascript object notation

**NFDUMP** NetFlow dump

**NIDS** Network intrusion detection system

**RAM** Random access memory

**RDP** Remote desktop protocol

**SIP** Session initiation protocol

**SSH** Secure shell

**TCP** Transmission control protocol

**TELNET** Teletype network

**UDP** User datagram protocol

**XML** Extensible markup language

Příloha **B**

# Obsah přiloženého CD

- `configs` ..... konfigurační soubory pro instalaci
- `doc` ..... dokumentace zdrojových kódů
- `DP_Švepeš_Marek_2017.pdf` ..... text práce ve formátu PDF
- `flow_scatter-1.0.0.tar.gz` ... distribuční balík modulu `Flow scatter`
- `flow_splitter-1.0.0.tar.gz` . distribuční balík modulu `Flow splitter`
- `readme.txt` ..... stručný popis obsahu CD
- `text-src` ..... zdrojové soubory textu práce
  - `DP_Švepeš_Marek_2017.tex` .. zdrojová forma práce ve formátu LaTeX
  - `img` ..... obrázky použité v textu práce

Příloha C

# Instalační manuál

Tato sekce popisuje postup instalace a spuštění testovacího prostředí, které bylo použito pro experimenty a testování v Sekci 4.1. Instalační manuál byl vytvořen a otestován na systému *CentOS 7*, který je aktuálně používán na vývojových serverech sdružení CESNET.

## C.1 Instalace

Pro instalaci systému `NEMEA` [1] a kolektoru `IPFIXcol` [2] je nutné nejdříve přidat repozitáře s binárními balíky. Pro jejich přidání je potřeba vytvořit soubory `copr-nemea.repo` a `copr-ipfixcol.repo` v adresáři `/etc/yum.repos.d/`. Obsah souborů ukazuje Výpis C.1 a Výpis C.2.

```
[group_CESNET-NEMEA]
name=Copr repo for NEMEA owned by @CESNET
baseurl=https://copr-be.cloud.fedoraproject.org/results/@CESNET/
    NEMEA/epel-7-$basearch/
type=rpm-md
skip_if_unavailable=True
gpgcheck=1
gpgkey=https://copr-be.cloud.fedoraproject.org/results/@CESNET/
    NEMEA/pubkey.gpg
repo_gpgcheck=0
enabled=1
enabled_metadata=1
```

Výpis C.1: Konfigurace repozitáře `NEMEA` v souboru `/etc/yum.repos.d/copr-nemea.repo`.

[1] https://github.com/CESNET/NEMEA
[2] https://github.com/CESNET/IPFIXcol

```
[group_CESNET-IPFIXcol]
name=Copr repo for IPFIXcol owned by @CESNET
baseurl=https://copr-be.cloud.fedoraproject.org/results/@CESNET/
    IPFIXcol/epel-7-$basearch/
type=rpm-md
skip_if_unavailable=True
gpgcheck=1
gpgkey=https://copr-be.cloud.fedoraproject.org/results/@CESNET/
    IPFIXcol/pubkey.gpg
repo_gpgcheck=0
enabled=1
enabled_metadata=1
```

Výpis C.2: Konfigurace repozitáře `IPFIXcol` v souboru `/etc/yum.repos.d/copr-ipfixcol.repo`.

Z přidaných repozitářů je nutné nainstalovat `IPFIXcol`, `IPFIXsend`, `NEMEA` moduly, `NEMEA Supervisor` a `NEMEA` framework s potřebnými knihovnami *libtrap* a *UniRec*. Tuto instalaci, včetně potřebných závislostí, provede pomocí RPM balíků následující příkaz (spuštěný s právy uživatele *root*):

```
$ yum install nemea nemea-framework-devel ipfixcol ipfixcol-unirec-output
```

Dále je potřeba nainstalovat nové moduly `Flow scatter` a `Flow splitter`. Moduly jsou závislé pouze na knihovnách *libtrap* a *UniRec*, které jsou již nainstalovány z předchozího příkazu. Po stažení archivů *flow_scatter-1.0.0.tar.gz* a *flow_splitter-1.0.0.tar.gz* z přiloženého CD je potřeba v adresáři s archivy provést tyto příkazy:

```
$ tar -zxvf flow_scatter-1.0.0.tar.gz
$ cd flow_scatter-1.0.0
$ ./configure --prefix=/usr --bindir=/usr/bin/nemea --libdir=/usr/lib64 -q
$ make
$ sudo make install
```

```
$ tar -zxvf flow_splitter-1.0.0.tar.gz
$ cd flow_splitter-1.0.0
$ ./configure --prefix=/usr --bindir=/usr/bin/nemea --libdir=/usr/lib64 -q
$ make
$ sudo make install
```

## C.2 Konfigurace

Před spuštěním je nutné změnit konfiguraci některých částí připravovaného prostředí. Nejprve je potřeba z přiloženého CD zkopírovat složku `configs`, která obsahuje všechny potřebné konfigurace. Poté se spuštěním následujících příkazů přepíše výchozí konfigurace modulu `ipblacklist_filter` a `IPFIXcol`:

```
$ cd configs
$ sudo cp ./ipblacklistfilter/* /etc/nemea/ipblacklistfilter/
$ sudo cp ./ipfixcol/* /etc/ipfixcol/
```

Nakonec je potřeba vygenerovat konfiguraci všech komponent prostředí (viz Sekce 4.3.2) pro modul `Supervisor` pomocí skriptu `gen_sup_conf.sh`, který se nachází ve složce `configs`. Skript očekává 3 argumenty: i) počet výpočetních uzlů (zde použito 8), ii) cestu k *IPFIX* souboru, který bude přehrán pomocí `IPFIXsend` nástroje, iii) cestu k existujícímu adresáři, kam budou uloženy výstupní soubory `Logger` modulů s nahlášenými událostmi (alerty). Vygenerovaná konfigurace nahradí výchozí nainstalovanou. Vygenerování a přepsání provedou následující příkazy:

```
$ cd configs/sup_conf/
$ ./gen_sup_conf.sh 8 /cesta/k/soubor.ipfix /cesta/k/alertům > ./conf.xml
$ sudo cp ./conf.xml /etc/nemea/supervisor_config_template.xml
```

## C.3 Spuštění

Spuštění celého prostředí lze provést pomocí modulu `Supervisor` [3]. Příkazem

```
$ sudo service nemea-supervisor restart
```

se lze ujistit, že modul běží a zároveň také znovu-načíst změněnou konfiguraci. Posledním krokem je samotné spuštění všech komponent prostředí pomocí příkazu

```
$ supcli
```

následným zvolením možnosti 1 pro zapnutí profilu nebo modulu a povolením všech dostupných profilů. Pomocí čísla 4 lze poté zkontrolovat stav systému, který by měl být stejný jako na Obr. D.3.

[3] https://github.com/CESNET/NEMEA-Supervisor

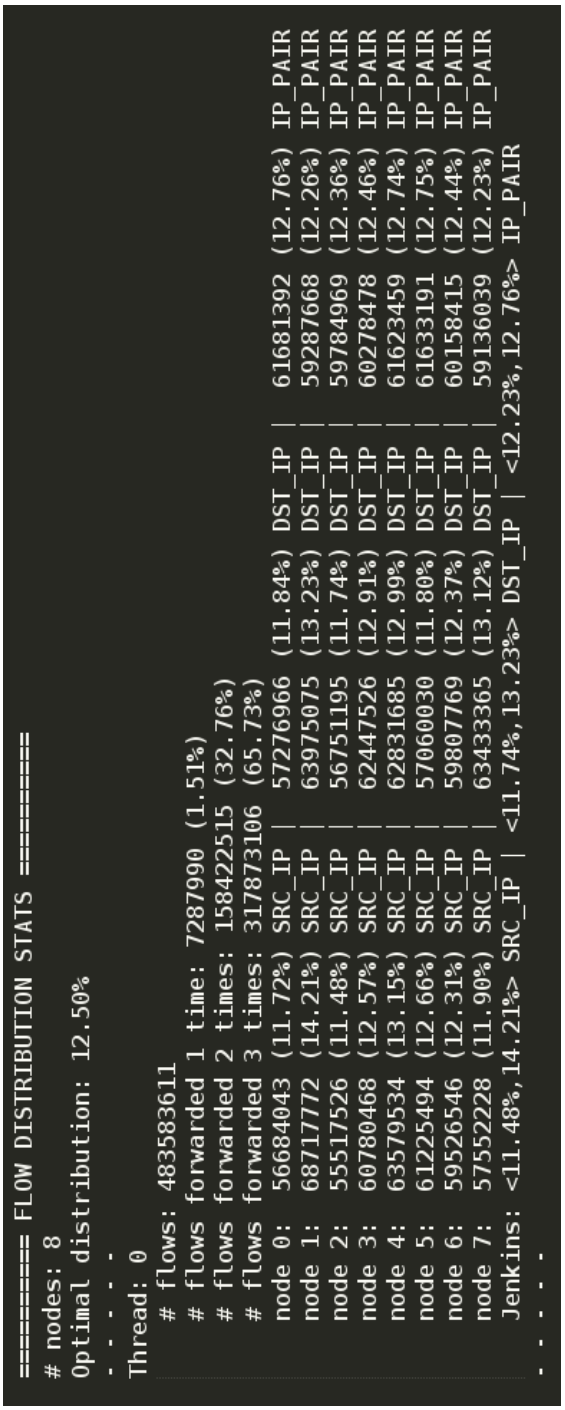
Příloha D

# Obrázkové přílohy

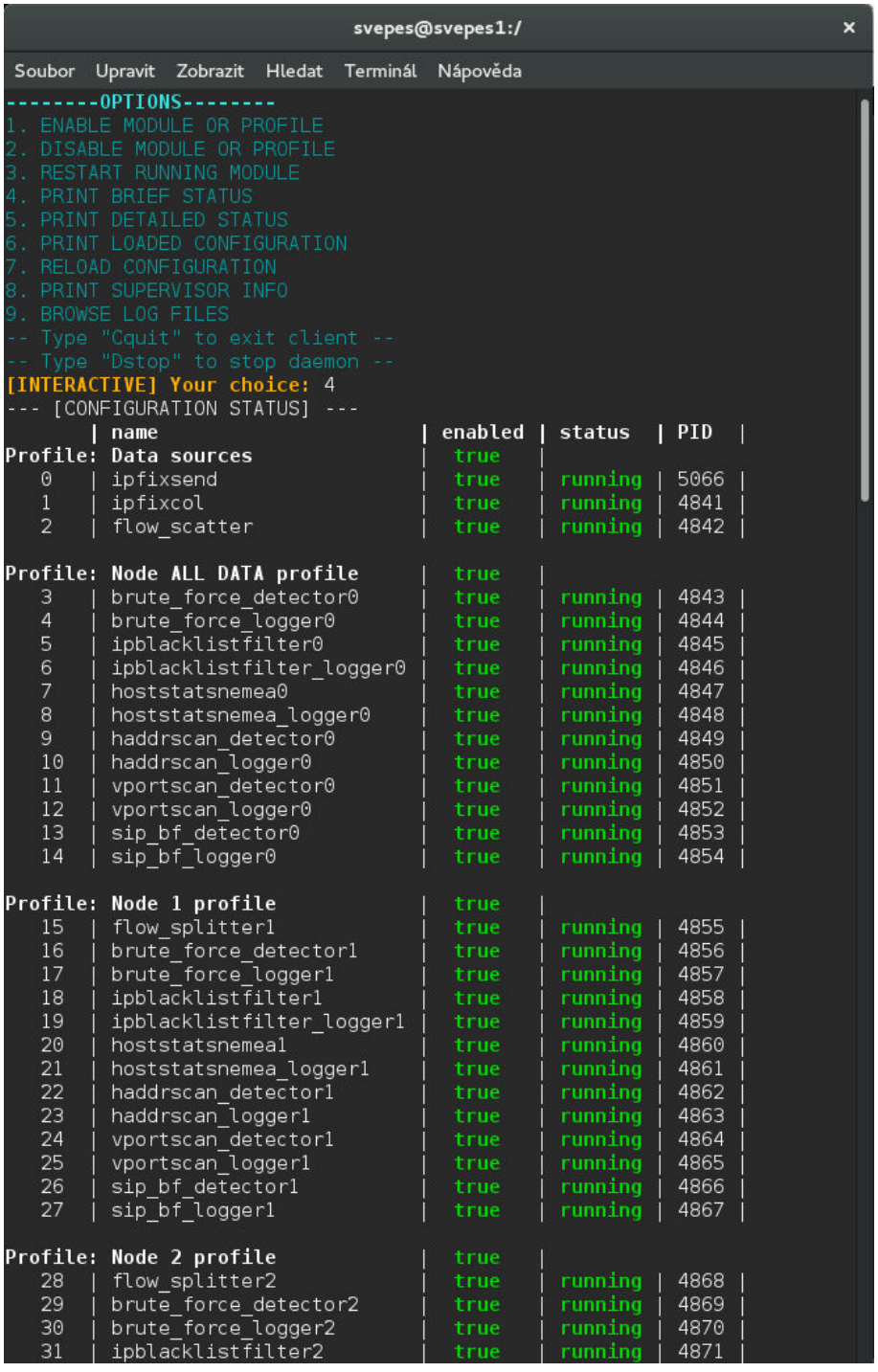
```
<module>
   <name>brute_force_detector1</name>
   <enabled>true</enabled>
   <path>/usr/bin/nemea/brute_force_detector</path>
   <params>-R -S -T</params>
   <trapinterfaces>
      <interface>
         <type>UNIXSOCKET</type>
         <direction>IN</direction>
         <params>flow_data_source_PAIR1</params>
      </interface>
      <interface>
         <type>UNIXSOCKET</type>
         <direction>OUT</direction>
         <params>bfd_data1_out</params>
      </interface>
   </trapinterfaces>
</module>

<module>
   <name>brute_force_logger1</name>
   <enabled>true</enabled>
   <path>/usr/bin/nemea/logger</path>
   <params>-t -T -w /data/brute_force_detector/detected1.log</params>
   <trapinterfaces>
      <interface>
         <type>UNIXSOCKET</type>
         <direction>IN</direction>
         <params>bfd_data1_out</params>
      </interface>
   </trapinterfaces>
</module>
```

Obrázek D.1: Část konfigurace, která popisuje komponenty testovacího prostředí (konkrétně dva `NEMEA` moduly). Tato konfigurace je použita modulem `NEMEA Supervisor`, který celé prostředí spravuje.




Obrázek D.2: Ukázka výpisu statistik o rozdělení síťových toků mezi výpočetní uzly. Statistiky jsou periodicky vypisovány modulem `Flow scatter` při spuštění s přepínačem -S.




Obrázek D.3: Všechny komponenty testovacího prostředí jsou spuštěny a monitorovány modulem `NEMEA Supervisor`. Obrázek ukazuje výpis stavu prostředí (konfigurace).