FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vylepšená knihovna pro komunikaci NEMEA modulů
Student:	Matěj Barnat
Vedoucí:	Ing. Tomáš Čejka, Ph.D.
Studijní program:	Informatika
Studijní obor:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Nastudujte systém NEMEA pro analýzu síťového provozu a detekci anomálií na základě síťových toků v reálném čase. Zaměřte se na NEMEA Framework, který tvoří základ systému.
Proveďte analýzu stávajícího řešení a ve spolupráci s vedoucím práce identifikujte nedostatky v části NEMEA Framework, která se týká včasného doručování dat k bezpečnostní analýze.
Navrhněte novou verzi knihovny pro komunikaci mezi NEMEA moduly, která bude lépe řešit předávání zpráv mezi moduly tak, aby se minimalizovala ztráta dat v souvislosti se zahlcením detekčních modulů.
Implementujte navržené změny.
Novou verzi NEMEA Framework důkladně otestujte (včetně porovnání výkonnosti vzhledem ke stávající verzi) a následně začleňte do distribučních balíků systému NEMEA.

Seznam odborné literatury

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

prof. Ing. Pavel Tvrdík, CSc.
vedoucí katedry

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

V Praze dne 24. ledna 2019

Poděkování

Tímto bych chtěl poděkovat Ing. Tomáši Čejkovi, Ph.D. za odborné vedení a cenné rady při tvorbě této práce. Můj dík patří také sdružení CESNET, z.s.p.o. a týmu Liberouter za možnost podílet se na vývoji projektu NEMEA. V neposlední řadě děkuji své rodině za podporu během celého studia.

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 16. května 2019 ……

Odkaz na tuto práci

Barnat, Matěj. *Vylepšená knihovna pro komunikaci NEMEA modulů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zabývá monitorovacím systémem NEMEA, vyvíjeným ve spolupráci sdružení CESNET a českých vysokých škol za účelem analýzy provozu a detekce anomálií na síti CESNET2. NEMEA je modulární systém, založený na principu zpracování síťových toků v reálném čase.

Cílem práce bylo na základě analýzy současného řešení NEMEA frameworku navrhnout a realizovat vylepšení knihovny libtrap, která tvoří jeho hlavní část. Motivací pro navržené změny je optimalizace komunikace mezi nejvíce zatěžovanými uzly systému NEMEA, kde je vyžadována vysoká propustnost.

Hlavním obsahem práce je analýza současné implementace této knihovny a následné přepracování její části, která se týká předávání dat mezi jednotlivými moduly. Výsledkem je nová verze knihovny libtrap, která byla začleněna do distribučních balíků systému NEMEA.

Klíčová slova Monitorování síťového provozu, detekce anomálií, NEMEA, TRAP

Abstract

This thesis deals with the NEMEA system, developed in cooperation of CESNET association and czech universities with the intention of traffic analysis and anomaly detection on CESNET2 network. NEMEA is a modular system, based on real time analysis of network flows.

Main goal was to analyze current implementation of NEMEA framework and to design and implement improvements of the libtrap library, based on results of the analysis. The aim was to optimize communication between the most heavily loaded modules, which require high throughput.

Essential parts of this thesis are analysis of the libtrap library and redesigning its parts related to data transfer between individual modules. The resulting new version of libtrap has been included in distribution packages of NEMEA framework.

Keywords Network traffic monitoring, anomaly detection, NEMEA, TRAP

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V posledních letech byl v oblasti komunikačních a síťových technologií učiněn velký pokrok. Je přirozené, že úměrně s tím, jak roste všeobecná dostupnost těchto služeb, zaznamenáváme také čím dál tím větší počet útoků zneužívajících komunikační infrastruktury a dalších nežádoucích jevů.

Monitorování síťového provozu je dnes proto nezbytnou součástí každé větší infrastruktury, zejména na straně subjektů provozujících komplexní komunikační sítě, například poskytovatelů internetových služeb.

NEMEA (*Network Measurements Analysis*) [1] poskytuje open-source řešení pro automatizovanou analýzu provozu a detekci anomálií v reálném čase. Systém je založený na *modulární* architektuře a využívá principu proudového zpracování síťových toků.

Konkrétní instanci tohoto systému si lze představit jako soubor programů (modulů), propojených pomocí komunikačních rozhraní poskytovaných společným frameworkem. Každý modul má jeden specifický úkol (detekce, logování, reportování, …) a k přenosu dat a komunikaci s ostatními moduly používá rozhraní poskytované platformou TRAP. NEMEA obsahuje jednak sadu modulů pro detekci běžných síťových útoků a zpracování datových toků, hlavně ale nabízí podporu pro rychlý vývoj a nasazení nových modulů, splňujících konkrétní požadavky uživatele.

Tato práce se zabývá frameworkem systému NEMEA, respektive platformou TRAP. TRAP (*Traffic Analysis Platform*) poskytuje NEMEA modulům API pro vzájemnou komunikaci a definuje formát UniRec, používaný k efektivnímu přenosu tzv. *flow* záznamů (flows). Nabízí také možnost moduly libovolně distribuovat mezi více výpočetních uzlů, díky čemuž je nasazení systému velmi flexibilní.

Cílem práce je na základě analýzy současné implementace navrhnout a realizovat změny v knihovně *libtrap*, která tvoří hlavní část NEMEA frameworku. Motivací je optimalizace komunikace mezi klíčovými uzly systému, kde je vyžadována vysoká propustnost. Těmi jsou zpravidla kolektory — moduly, které shromažďují data z měřících sond a následně je přeposílají detekčním a jiným modulům.

V úvodní části jsou popsány metody monitorování síťového provozu, které jsou v dnešní době standardně využívány, a je představen systém NEMEA. Dále je provedena analýza frameworku tohoto systému, zejména současné implementace jeho komunikační vrstvy — knihovny libtrap. Další dílčí část je návrh a realizace vylepšení této knihovny adresující nedostatky odhalené v analytické části. Nakonec je popsáno testování nové verze knihovny a jsou shrnuty výsledky práce.

Kapitola 1

Monitorování počítačových sítí

1.1 Metody monitorování provozu

Monitorování a detekce anomálií na síti je běžně automatizována a provádí ji tzv. Intrusion Detection / Intrusion Prevention systémy. Tyto systémy můžeme rozdělit do dvou kategorií podle toho, jak k analýze provozu přistupují.

Prvním druhem jsou tzv. *packet-based* systémy, které zpracují zvlášť každý paket a detekce nežádoucího chování je založena zejména na obsahu komunikace. Dále existují *flow-based* systémy, které zkoumají data na vyšší úrovni abstrakce, tzv. flows (síťové toky). Při analýze síťových toků je detekce anomálií prováděna za použití obecnějších údajů o komunikaci, jako je doba jejího trvání a počet přenesených paketů.

1.1.1 Paketová analýza

Běžný přístup k monitorování síťového provozu je záznam probíhající komunikace (například pomocí tzv. *zrcadlení portů*) a následná analýza jednotlivých paketů a jejich obsahu. Detekční algoritmus hledá podezřelé chování, či specifické vzory komunikace a hlásí detekované anomálie, útoky a další nežádoucí události.

Výhodou tohoto přístupu je, že systém má o probíhající komunikaci veškeré informace včetně jejího obsahu. Nevýhodou je vysoká výpočetní náročnost, plynoucí z velkého objemu zpracovávaných dat.

1.1.2 IP flow analýza

Monitorování rozsáhlých sítí pomocí paketové analýzy je často kvůli vysokým nárokům na výpočetní výkon nemožné. Z tohoto důvodu se v komplexních sítích běžně využívá principu analýzy síťových toků, takzvaných *IP flows*.

Jeden tok představuje proud dat (sekvenci síťových paketů) se shodnou pěticí následujících údajů:

— Zdrojová IP adresa

— Zdrojový port

— Cílová IP adresa

— Cílový port

— Typ transportního protokolu

Každý tok tedy popisuje jeden směr komunikace dvou zařízení ve společné síti. Pro efektivní přenos informací o síťových tocích bylo vyvinuto několik protokolů, jako například IPFIX (*IP Flow Information Export)* [3]. Tento protokol definuje tzv. *flow záznamy*. Flow záznam obsahuje kromě identifikace daného toku i další informace, jako například:

— Doba vzniku

— Délka trvání

— Počet přenesených paketů/bajtů

Na rozdíl od analýzy jednotlivých paketů nemají systémy založené na principu analýzy síťových toků o probíhající komunikaci tak podrobné informace, mohou ale pomoci odhalit jiné druhy událostí, jelikož pracují na vyšší úrovni abstrakce [1].

1.2 NEMEA

NEMEA je modulární open-source systém pro monitorování síťového provozu a detekci anomálií v reálném čase. Je založen na principu proudového zpracování síťových toků (flows).

Mezi hlavní výhody patří vysoká propustnost, flexibilita a distribuovatelnost systému. Další klíčová vlastnost, plynoucí z proudového přístupu ke zpracování dat, je minimální prodleva mezi jejich přijetím a následnou analýzou. Příchozí data nejsou ukládána v perzistentní paměti a jejich analýza, či případná detekce nežádoucího chování tedy není nijak zpožděna. Systém NEMEA se skládá z těchto částí:

— Moduly

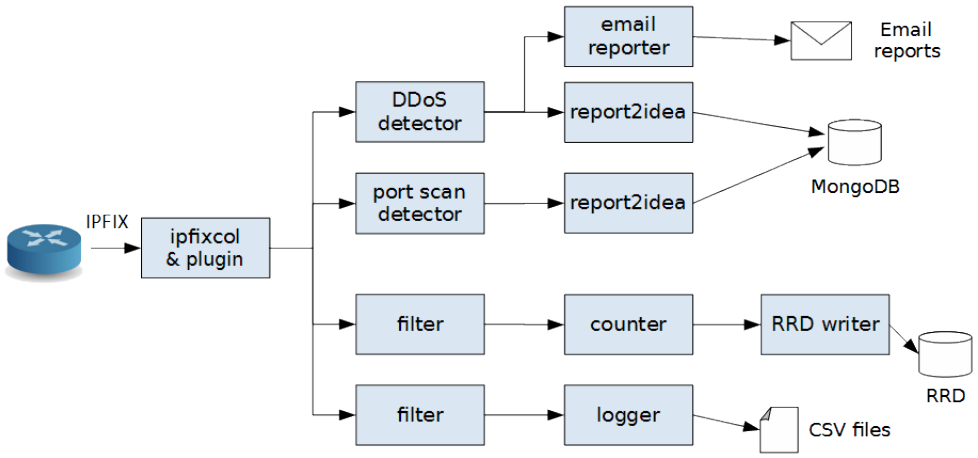
— NEMEA Framework

— Supervisor

1.2.1 Moduly

Instance systému NEMEA je soubor navzájem propojených modulů. Každý modul je nezávislý proces komunikující s ostatními moduly/procesy pomocí TRAP rozhraní.

Obrázek 1.1 ukazuje příklad nasazení tohoto systému. Propojení modulů lze reprezentovat jako orientovaný graf — strom, kde na vstupu modulu představující kořen jsou zpravidla sbírána data z měřících sond a na výstupu modulů představující listy jsou potom výsledky analýzy [2].



Obrázek 1.1: Ukázka nasazení systému NEMEA [4]

Toto řešení nabízí několik výhod. Moduly lze snadno přidávat a odebírat, aniž by tím byly ostatní běžící moduly jakkoliv ovlivněny. Lze také snáze monitorovat využití systémových prostředků různými moduly pomocí nástrojů operačního systému.

Další z výhod, které modulární architektura přináší, je odolnost systému proti selhání jednotlivých modulů a možnost nastavit kvóty operačního systému tak, aby chybný modul nemohl vyčerpat všechny dostupné prostředky a vyřadit tak z provozu celý systém.

Moduly lze do běžícího systému přidávat dynamicky. Po spuštění se modul snaží připojit k rozhraním daných konfigurací a pokud nelze spojení navázat ihned (například pokud ostatní moduly zatím nebyly spuštěny), modul se automaticky pokusí připojit znovu. Díky tomu je přidání (analogicky i odebrání) modulu do běžícího systému velmi jednoduché a vyžaduje pouze spuštění aplikace [2].

Současná verze systému NEMEA obsahuje několik modulů pro detekci útoků, měření statistik, filtrování provozu, reportování detekovaných událostí, atd. Jeho hlavní předností je ale podpora snadné a rychlé implementace nových modulů, splňujících specifické požadavky na monitorování dané sítě.

1.2.2 NEMEA Framework

Dílčí částí každého nasazení systému NEMEA je propojení jednotlivých modulů. Účelem NEMEA frameworku je poskytovat modulům API pro vzájemnou komunikaci a efektivní přenos flow záznamů.

Klíčová vlastnost poskytovaných rozhraní je, že modul je od formy datového přenosu zcela odstíněn. Komunikace může probíhat lokálně (mezi procesy) i vzdáleně (přes síť), transparentně vůči modulům samotným. Systém lze tedy snadno distribuovat mezi více výpočetních uzlů, což může být žádoucí zejména při monitorování komplexní sítě, kde je třeba zpracovávat velké objemy dat. Framework se skládá z těchto částí (knihoven):

— Libtrap

— UniRec

— Common

— Pytrap

— Pycommon

Frameworku NEMEA, respektive platformě TRAP, je věnována samostatná Kapitola 2, kde jsou tyto knihovny popsány podrobněji.

1.2.3 Supervisor

Supervisor je speciální modul, poskytující nástroje ke správě a řízení systému NEMEA. Vstupem tohoto modulu je konfigurační soubor ve formátu XML, který definuje skupiny modulů a způsob jejich propojení (specifikace jednotlivých rozhraní). Poskytované služby výrazně zjednodušují nasazení komplexních topologií systému NEMEA. Správci dávají nejen lepší přehled o tom, které moduly jsou momentálně spuštěny a jak jsou zapojeny, ale i možnost konfigurovat a ovládat více modulů zároveň.

Supervisor se k ostatním modulům připojuje a komunikuje s nimi přes tzv. *servisní* rozhraní, které je implicitně součástí každého NEMEA modulu. Toto speciální rozhraní umožňuje připojit se k běžícím modulům a konfigurovat některé jejich vlastnosti. Lze zapnout či vypnout bufferování odchozích zpráv, nastavit *data timeout* (maximální doba, kterou modul stráví čekáním na odeslání/přijetí jedné zprávy) a *autoflush timeout* (maximální doba, kterou zpráva stráví ve vyrovnávací paměti před jejím odesláním).

Servisní rozhraní dále poskytuje možnost získávat různé údaje z ostatních (vstupních a výstupních) TRAP rozhraní daného modulu ve formátu JSON.

Vstupní rozhraní:

- ID rozhraní
- Typ rozhraní
- Stav připojení
- Počet přijatých zpráv
- Počet přijatých bufferů

Výstupní rozhraní:

- ID rozhraní
- Typ rozhraní
- Počet připojených klientů
- Počet odeslaných zpráv
- Počet odeslaných bufferů
- Počet zahozených zpráv
- Počet vypláchnutí bufferu

Komunikace přes servisní rozhraní probíhá na pozadí v samostatném vlákně a normální činnost modulu tím tedy není nijak ovlivněna.

Kapitola 2

Traffic Analysis Platform

Systém NEMEA je založen na principu modulární architektury. Jeho nedílnou součástí je NEMEA framework, neboli TRAP (*Traffic Analysis Platform*), který jednotlivé moduly spojuje pomocí komunikačních rozhraní v jeden ucelený systém.

TRAP se skládá ze tří hlavních částí (knihoven jazyka C). Knihovna *libtrap* implementuje komunikační rozhraní, knihovna *unirec* poskytuje stejnojmenný datový formát, užívaný k přenosu informací o síťových tocích (flows), a knihovna *common* dodává datové struktury a algoritmy běžně používané při analýze síťového provozu. TRAP obsahuje také moduly *pytrap* a *pycommon*, poskytující podporu pro implementaci NEMEA modulů v jazyce Python.

2.1 Libtrap

Knihovna libtrap tvoří hlavní část TRAP frameworku. Poskytuje jednotlivým modulům rozhraní k vzájemné komunikaci v podobě souboru typu *shared object*, který je k modulu přilinkován v době spuštění.

Velkou výhodou je, že libtrap poskytuje uživateli (vývojáři NEMEA modulů) vrstvu abstrakce od implementace konkrétního rozhraní. Moduly jsou od formy komunikace zcela odstíněny — pomocí společného API odesílají či přijímají zprávy z určeného rozhraní, způsob a parametry přenosu určuje konfigurace TRAP.

2.1.1 Rozhraní

Každý modul má definovaný počet vstupních a výstupních rozhraní. Všechna rozhraní jsou jednosměrná a data přenášejí ve formě samostatných zpráv. Všechny zprávy poslané jedním rozhraním musí být ve stejném formátu, jehož šablona je specifikována dynamicky při připojení modulu do systému. Pro většinu komunikace mezi NEMEA moduly je využíván protokol UniRec (Unified Record), který je podrobněji popsán v Sekci 2.2.

Existuje více druhů rozhraní rozlišujících se formou přenosu dat, kterou používají. Důležité je, že moduly jsou odděleny od implementace konkrétního rozhraní — ke vzájemné komunikaci používají stejné TRAP API, nezávisle na zvoleném typu rozhraní. Formu komunikace lze tedy měnit pouze změnou konfigurace, což značně zjednodušuje distribuci systému mezi více výpočetních uzlů.

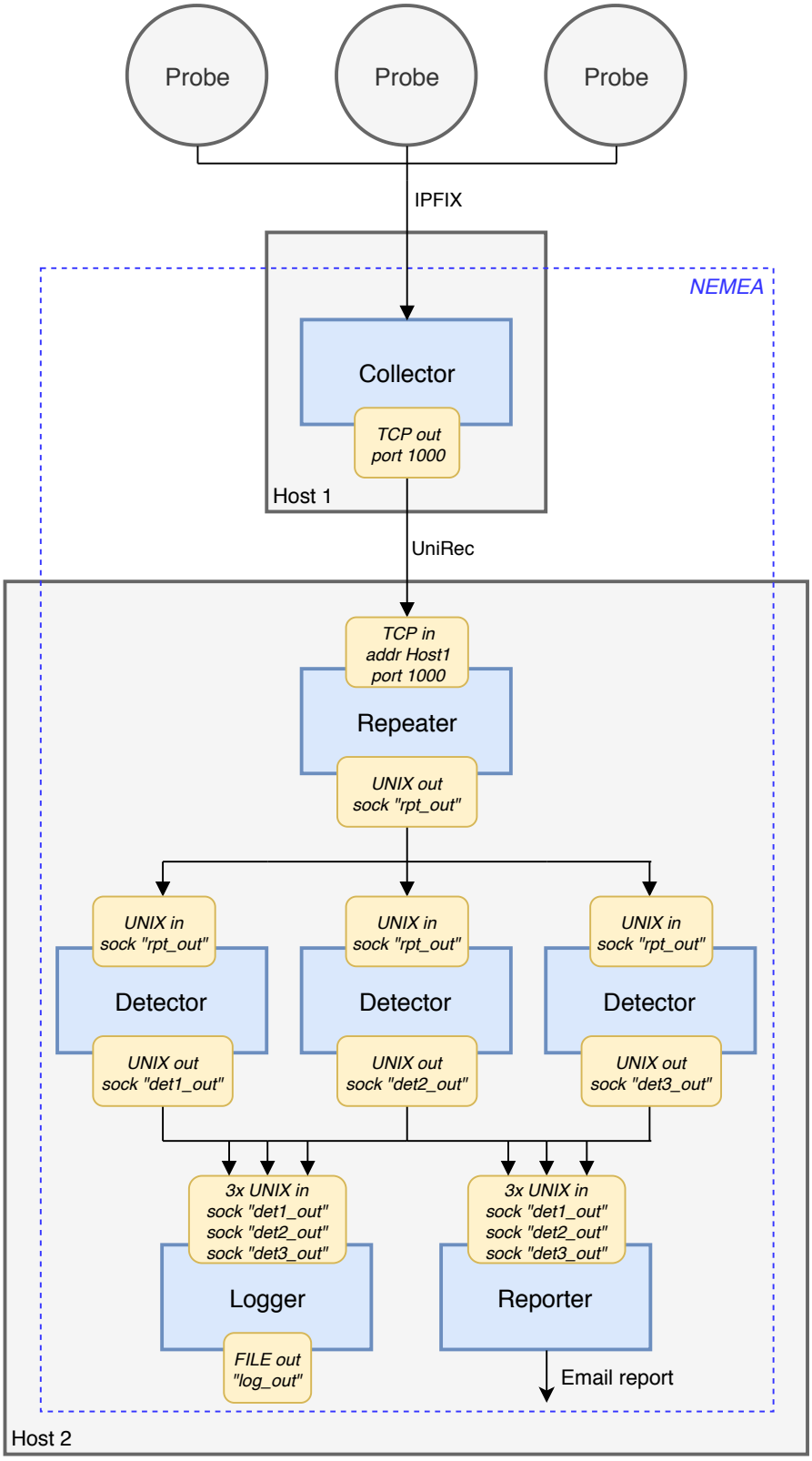
Knihovna libtrap implementuje několik druhů rozhraní pro lokální i vzdálenou komunikaci:

UNIX rozhraní určené pro lokální komunikaci, tedy pro datové přenosy mezi moduly běžícími na stejném stroji. Využívá metodu přenosu přes UNIX socket.

TCP/IP se využívá hlavně pro vzdálené přenosy, lze jej ale využít i pro lokální komunikaci. Přenos probíhá za použití síťových socketů.

TLS nabízí možnost šifrovaného přenosu přes síť pomocí protokolu TLS (Transport Layer Security). Toto rozhraní vyžaduje zadání platného certifikátu a certifikační autority.

File rozhraní se používá zejména pro logování a offline testování detekčních modulů. Data jsou čtena z, či ukládána do souborů.



Obrázek 2.1: Ukázka propojení modulů pomocí TRAP rozhraní

Obrázek 2.1 znázorňuje příklad nasazení systému NEMEA a propojení různých NEMEA modulů (modrá) pomocí TRAP rozhraní (žlutá). Kolektor je základním uzlem celého systému, na jehož vstupním rozhraní jsou přijímána data z měřících sond ve formátu IPFIX [3]. Data jsou následně exportována do některého z formátů, které TRAP podporuje (například UniRec), a odeslána na výstupní rozhraní.

Kolektor běží na jiném stroji než zbytek systému a je tedy třeba pro přenos použít rozhraní pro vzdálenou komunikaci, například TCP/IP. Jelikož jsou exportovaná data odesílána několika detekčním modulům, je pro snížení objemu dat odesílaných přes síť použit modul *repeater*.

Repeater přijatá data beze změny odesílá na výstup, ke kterému jsou připojeny různé detekční moduly — *detektory*. Komunikace probíhá pouze mezi procesy na stejném stroji a lze tedy použít UNIX rozhraní.

Detektory tvoří jádro samotného systému. Tyto moduly provádějí analýzu provozu a detekci útoků a dalších anomálií na základě různých pravidel, algoritmů a statistických metod. Výsledky analýzy jsou odesílány přes výstupní rozhraní logovacím a reportovacím modulům, které tato hlášení dle konfigurace uloží, odešlou jako e-mail, atd.

2.2 UniRec

UniRec je efektivní binární formát pro ukládání a přenos jednoduchých datových záznamů, vyvinutý přímo pro nasazení v rámci systému NEMEA. UniRec jako takový je jen obecná datová struktura (podobná prosté C struktuře), konkrétní formát je definován při inicializaci a je dán šablonou, specifikující jednotlivé položky každého záznamu.

V porovnání s ostatními používanými datovými formáty má UniRec dvě hlavní odlišnosti. Za prvé umožňuje velmi rychlý přístup k jednotlivým položkám záznamů — oproti ostatním formátům, které musí být před přístupem k položce parsovány, k nim UniRec může přistupovat přímo, což mu dává rychlost přístupu srovnatelnou s C strukturou. Za druhé, všechny záznamy posílané přes jedno rozhraní mají stejnou šablonu formátu, což ve většině případů není problém a výrazně to zjednodušuje zpracování těchto záznamů [1].

2.2.1 Automatické nastavení formátu

Před započetím komunikace je nutné určit formát zpráv, respektive šablonu UniRec záznamů, které budou přes dané rozhraní přenášeny. Šablona určuje konkrétní datové položky a jejich velikost.

Při připojení dvou TRAP rozhraní implicitně probíhá tzv. *negociace*, která určuje formát přenášených zpráv. Klient (vstupní rozhraní) specifikuje požadované položky, server (výstupní rozhraní) specifikuje nabízené položky. Pokud je množina požadovaných položek alespoň podmnožinou těch nabízených, je navázáno spojení. V opačném případě je klient zamítnut a spojení je ukončeno.

2.3 Nemea common

Tato knihovna poskytuje modulům systému NEMEA algoritmy a datové struktury, bežně využívané při vývoji softwaru zaměřeného na analýzu síťového provozu. Mezi funkce a struktury nabízené touto knihovnou patří:

— B+ tree

— Prefix tree

— Cuckoo hash

— Counting sort

Knihovna Common zjednodušuje vývoj NEMEA modulů a umožňuje tak vývojářům věnovat se místo programování hlavně vývoji nových metod analýzy provozu a zpracování síťových toků.

Kapitola 3

Analýza současného stavu

V této kapitole je podrobně rozebrána knihovna libtrap a její současná implementace.

Před použitím funkcí TRAP API je třeba knihovnu inicializovat. Při inicializaci se vytváří takzvaný *kontext* — objekt obsahující informace o konfiguraci a interní struktury dané instance libtrap. Součástí kontextu jsou identifikátory pomocných vláken, které libtrap spouští, dále proměnné a struktury jednotlivých rozhraní, vyrovnávací paměti (buffery), atd.

Inicializace také zařizuje parsování parametrů libtrap a na jejich základě vytváří jednotlivá rozhraní definovaná modulem. Hlavním parametrem je tzv. *ifc-spec* [5], což je řetězec obsahující informace o konfiguraci příslušných rozhraní. Jednotlivé položky obsažené v ifc-spec struktuře se mohou lišit v závislosti na typu použitého rozhraní. Konkrétní položky ifc-spec různých rozhraní zahrnují:

Typ rozhraní písmeno označující typ rozhraní — *u* pro Unix, *t* pro TCP/IP, *T* pro TLS, *f* pro File a *b* pro Blackhole

Socket ID název socketu (string) pro UNIX, dvojice IP adresa + port pro TCP/IP a TLS, jméno souboru pro File

Maximální počet klientů výstupní rozhraní umožňují nastavit maximální počet najednou připojených klientů

Timeout počet mikrosekund reprezentující timeout, nebo jedna ze speciálních hodnost (WAIT, NOWAIT, HALFWAIT)

Bufferswitch 0–bufferování vypnuto, nebo 1–bufferování zapnuto

Certifikát / veřejný klíč rozhraní TLS vyžaduje zadání platného veřejného klíče, certifikátu a certifikační autority

Každý program linkující libtrap (NEMEA modul) používá jedno ze dvou různých API, které libtrap poskytuje, a sice *simple API*, nebo *context API* (Obrázek 3.1).

Obě nabízí ekvivalentní funkcionalitu s jedním hlavním rozdílem. Pro volání funkcí context API je třeba parametrem zadat konkrétní kontext, což umožňuje v jednom programu používat více různých instancí libtrap. Volání simple API pracují vždy nad tzv. globálním kontextem.

Simple API

```
+ trap_init(trap_module_info_t *, trap_ifc_spec_t *): int
+ trap_terminate(): int
+ trap_finalize(): int
+ trap_send(uint32_t, const void **, uint16_t): int
+ trap_recv(uint32_t, const void **, uint16_t): int
+ trap_get_verbose_level(): int
+ trap_set_verbose_level(int): void
+ trap_set_help_section(): void
+ trap_print_help(const trap_module_info_t *): void
+ trap_print_ifc_spec_help(): void
+ trap_ifcctl(uint8_t, uint32_t, int32_t) :int
+ trap_send_flush(uint32_t): void
```

```
+ trap_last_error: int
+ trap_last_error_msg: const char *
```

Context API

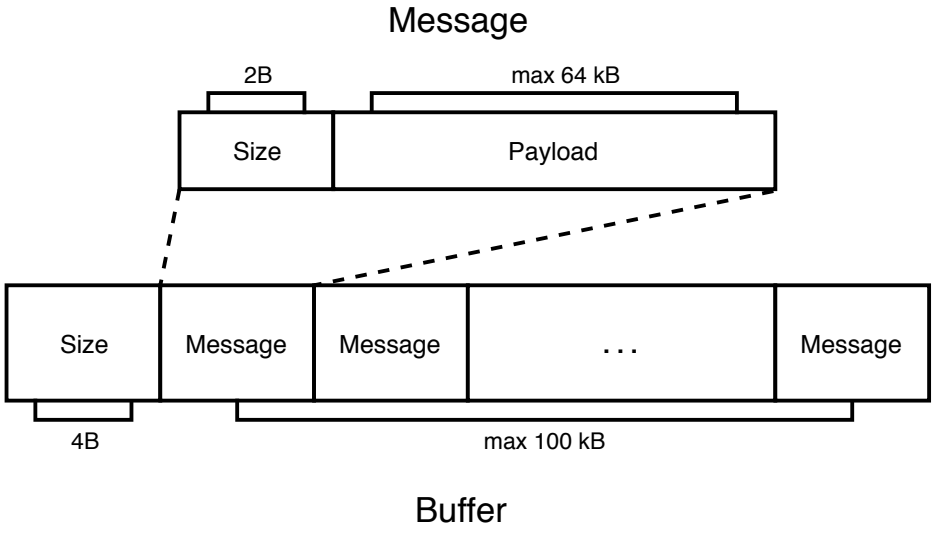
```
+ trap_create_ctx_t(): trap_ctx_priv_t *
+ trap_free_ctx_t(trap_ctx_priv_t *): void
+ trap_ctx_init(trap_module_info *, trap_ifc_spec *): trap_ctx_t *
+ trap_ctx_init2(trap_module_info *, trap_ifc_spec *,
                  const char *): trap_ctx_t *
+ trap_ctx_init3(const char *, const char *, int8_t, int8_t,
                  const char *, const char*): trap_ctx_t *
+ trap_ctx_terminate(trap_ctx_t *): int
+ trap_ctx_finalize(trap_ctx_t *): int
+ trap_ctx_send(trap_ctx_t *, uint32_t, const void *,
                uint16_t): int
+ trap_ctx_recv(trap_ctx_t *, uint32_t, const void **,
                uint16_t *): int
+ trap_ctx_send_flush(trap_ctx_t *, uint32_t): void
+ trap_ctx_ifcctl(trap_ctx_t *, int8_t, uint32_t, int32_t, ...): int
+ trap_ctx_vifcctl(trap_ctx_t *, int8_t, uint32_t, va_list): int
+ trap_ctx_get_last_error(trap_ctx_t *): int
+ trap_ctx_get_error_msg(trap_ctx_t *): const char*
+ trap_ctx_get_verbose_level(trap_ctx_t *): int
+ trap_ctx_set_verbose_level(trap_ctx_t *, int): void
+ trap_ctx_create_ifc_dump(trap_ctx_t *, const char *): void
```

Obrázek 3.1: API definované knihovnou libtrap

3.1 Architektura libtrap

Data v systému NEMEA se posílají ve formátu krátkých zpráv (maximálně 64 KB). Zprávy mohou být flow záznamy, výsledky detekce (alerty), či statistiky získané z flow dat. Obecně ale mohou obsahovat cokoliv [1].

Pro zvýšení propustnosti je využito principu bufferování. Zprávy jsou před odesláním agregovány do bloků (bufferů), což snižuje celkový počet časově náročných operací, které jsou při přenosu dat prováděny, zejména při komunikaci přes síť. Struktura zpráv a interních bufferů pro jejich dočasné ukládání je znázorněna na Obrázku 3.2.



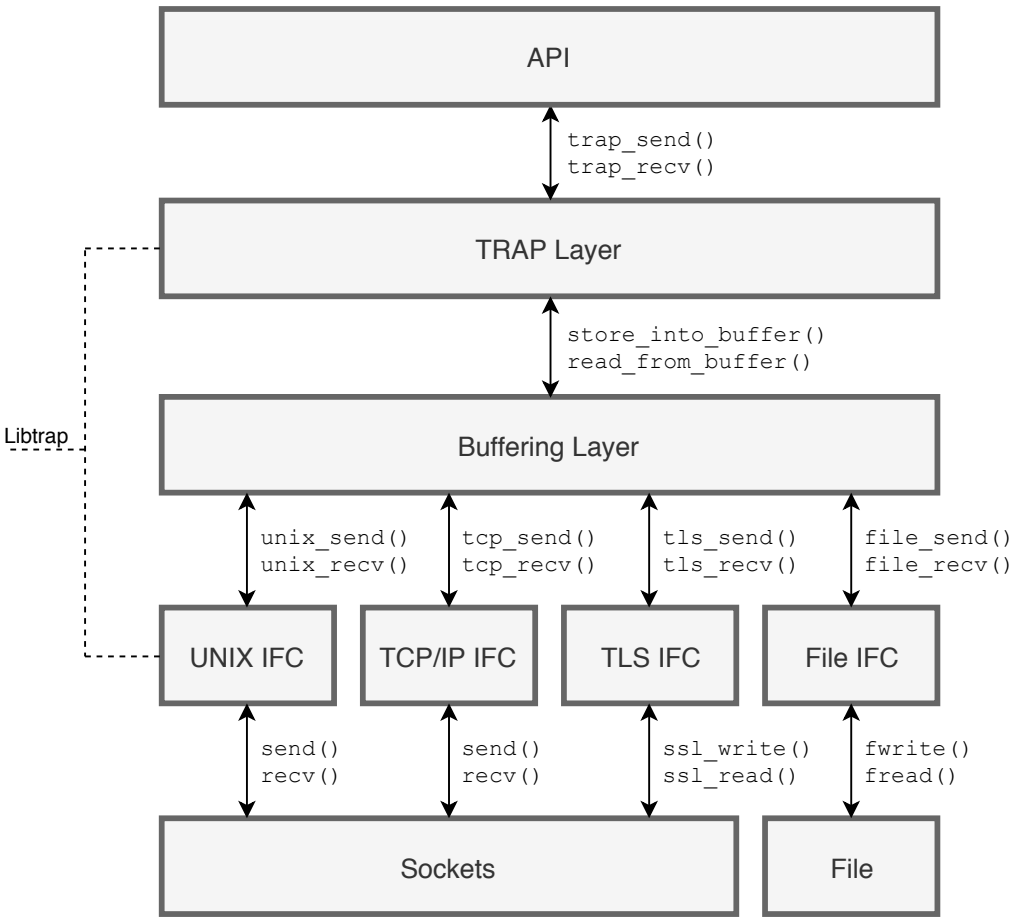
Obrázek 3.2: Struktura zpráv přenášených přes TRAP rozhraní a způsob jejich uspořádání ve vyrovnávací paměti

Obrázek 3.3 znázorňuje současnou architekturu libtrap. Knihovnu lze rozdělit do několika vrstev.

Na nejvyšší vrstvě probíhá komunikace mezi modulem a knihovnou pomocí funkcí TRAP API. Vrstva TRAP implementuje *high-level* funcionality libtrap, jako jsou autoflush, bufferswitch, negociace formátu při připojení, či nastavení timeoutu některého rozhraní. Zajišťuje také konfiguraci jednotlivých rozhraní, jejich inicializaci při spuštění a destrukci při ukončení modulu.

Příchozí zprávy jsou předány *bufferovací* vrstvě, ve které jsou uloženy do vyrovnávací paměti (buffer), kde čekají na odeslání. Když je buffer zcela zaplněn, nebo je signalizován flush (vypláchnutí bufferu), je buffer odeslán na příslušné rozhraní.

Další vrstva je vrstva *rozhraní*. Jejím hlavním účelem je oddělit zbytek TRAP od specifických vlastností konkrétní formy datového přenosu. Vyšší vrstvy se tedy chovají vždy stejně, ať pracují s jakýmkoliv typem rozhraní.



Obrázek 3.3: Architektura knihovny libtrap

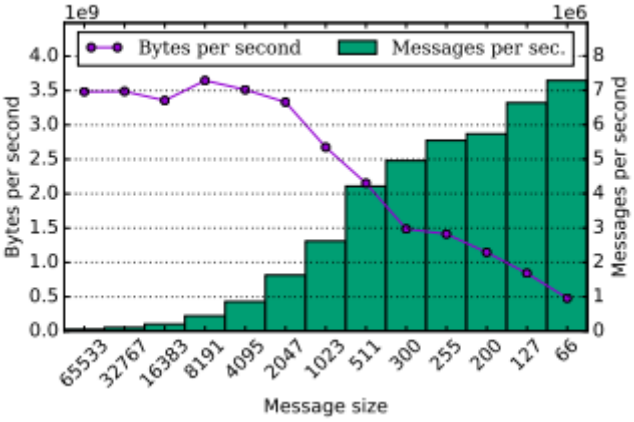
Rozhraní jsou vždy jednosměrná a dělí se na vstupní a výstupní. Jedno rozhraní může tedy data pouze odesílat, nebo pouze přijímat. S výjimkou *file* rozhraní, které vždy pracuje pouze s jedním souborem, jsou všechna TRAP rozhraní postavena na architektuře *klient–server*. Výstupní rozhraní akceptuje požadavky na spojení a odesílá data připojeným klientům. Moduly připojené na stejné výstupní rozhraní dostávají vždy stejná data.

3.2 Bufferování

NEMEA moduly jsou navrženy pro monitorování provozu na rozsáhlých sítích a lze tedy předpokládat, že budou využívány ke zpracování velkých objemů dat. Je proto nutné, aby TRAP rozhraní poskytovala modulům vysokou propustnost. Z tohoto důvodu odchozí zprávy před odesláním agregovány ve vyrovnávací paměti (bufferu). Tím se redukuje tzv. *overhead* spojený s každým datovým přenosem, což zvyšuje efektivitu využití dostupného pásma a tím i rychlost přenosu.

Bufferování zpráv je implicitně využíváno všemi dostupnými rozhraními, modul jej však může explicitně vypnout, například je-li vyžadována minimální prodleva mezi odesláním a doručením zpráv přenášených přes dané rozhraní. Tato možnost je využívána především u modulů produkujících zprávy, které nejsou odesílány příliš často, ale je žádoucí, by byly doručeny co nejdříve — například *alerty* (výsledky detekce).

Implicitní velikost bufferu je 100 KB a byla určena na základě experimentu zkoumajícím závislost propustnosti výstupních rozhraní na velikosti zpráv, které se přes ně odesílají. Tento experiment je popsán v [1] a jeho výsledky jsou znázorněny na Obrázku 3.4.



Obrázek 3.4: Graf závislosti propustnosti TRAP rozhraní na velikosti přenášených zpráv [1]

Je zřejmé, že nejvyšší rychlosti přenosu (3,5 GB/s) rozhraní dosahuje při maximální velikosti zpráv, tedy 64 KB. Zároveň se tím ale snižuje celkový počet poslaných zpráv. Naopak u zpráv velikosti 66 B, což odpovídá jednoduchému UniRec záznamu, je odesláno nejvíce zpráv (přes 7 milionů za sekundu), rychlost přenosu je však minimální (500 MB/s).

Obecně řečeno je tedy při zachování objemu dat efektivnější odesílat dlouhé zprávy s nízkou frekvencí, než naopak.

Při použití bufferování je dosaženo maximální efektivity využití systémových prostředků pro přenos dat, zároveň je ale ponechána schopnost odesílat malé zprávy s vysokou frekvencí. Nevýhodou je již zmiňovaná prodleva plynoucí z faktu, že zpráva bude odeslána až ve chvíli, kdy se zaplní celý buffer.

3.3 Timeout handling

Mezi další z nástrojů k ovládání toku dat přes TRAP rozhraní patří tzv. *timeout handling*. Pojmem timeout se v tomto kontextu myslí časový limit na odeslání, či přečtení jednoho bufferu z rozhraní. Přesněji řečeno, jde o maximální dobu, kterou modul stráví čekáním při volání funkcí `send() a recv()`. Timeout lze nastavit na libovolnou hodnotu s přesností na mikrosekundy.

Libtrap navíc definuje tři speciální hodnoty timeoutů, a sice `TRAP_WAIT`, `TRAP_NOWAIT` a `TRAP_HALFWAIT`. První ze zmiňovaných hodnot představuje nekonečný timeout. Dané funkce se tedy chovají jako *blokující volání* a mohou na odeslání zprávy čekat potenciálně nekonečně dlouho.

Opakem je `TRAP_NOWAIT`. Tato hodnota představuje nulový timeout. Pokud buffer nemůže být odeslán/přečten ihned, funkce se ihned vrací s návratovým kódem značícím timeout.

Poslední významná hodnota — `TRAP_HALFWAIT` — je kompromisem mezi předchozími variantami. Volání se chovají stejně jako v případě `TRAP_WAIT` s tím rozdílem, že modul není blokován v případě, že k rozhraní není zatím nikdo připojen.

3.4 Autoflush

Systém NEMEA je založen na principu analýzy provozu *v reálném čase*. Je tedy nezbytné, aby zprávy odeslané přes TRAP rozhraní byly protistraně doručeny s *minimálním zpožděním*.

Jelikož však každé výstupní rozhraní implicitně data před odesláním agreguje ve vyrovnávací paměti a buffery jsou odesílány až po jejich úplném zaplnění, může u modulů odesílajících zprávy s nízkou frekvencí dojít k nežádoucímu zpoždění (latenci) mezi odesláním zprávy na rozhraní a jejím doručením.

Z tohoto důvodu poskytuje TRAP mechanismus zvaný *autoflush*, který zabraňuje tomu, aby byla data uchovávána ve vyrovnávací paměti příliš dlouho. V případě, že se buffer nestihne naplnit za dobu určenou modulem (autoflush timeout), je buffer odeslán na příslušné rozhraní.

Autoflush je implementován jako pomocné vlákno, které je součástí každého NEMEA modulu.

3.5 Nedostatky v implementaci

Z dlouhodobého pozorování chování modulů systému NEMEA, nasazených na síti CESNET2, vyplynulo několik nežádoucích vlastností současné implementace knihovny libtrap, které jsou v této práci adresovány.

Hlavním problémem je, že systém není odolný vůči stavům, kdy někteří klienti (moduly) nejsou schopni zpracovávat příchozí zprávy dostatečně rychle. To může být zapříčiněno náhlým zvýšením provozu na monitorované síti (například v případě DDoS útoku), krátkodobým výpadkem/zpomalením přenosu na nižší vrstvě komunikace, či pouze výpočetní náročností algoritmu zpracování zpráv, který daný modul používá.

V těchto případech dochází k nežádoucí ztrátě dat plynoucí ze způsobu, jakým výstupní TRAP rozhraní zacházejí s vyrovnávací pamětí. Odesílání probíhá pouze ve chvíli, kdy na rozhraní přijde nová zpráva a v bufferu není dostatek volného místo pro její uložení. Pokud někteří klienti nestihnou přečíst celý buffer než vyprší timeout, je příchozí zpráva zahozena a zbytek bufferu bude odeslán v následujících voláních. Ostatní klienti tedy musí čekat na jeho odeslání a zbytečně tím přicházejí o nová data.

Dalším souvisejícím nedostatkem současné verze libtrap je neschopnost identifikace modulů, kteří datový přenos tímto způsobem zpomalují. NEMEA moduly připojené na stejné výstupní rozhraní dostávají vždy stejná data a je tedy obtížné určit, kteří klienti jsou zdrojem problému (ztráty dat), a kteří jsou těmito klienty pouze brzděni.

Kapitola 4

Návrh a implementace

Cílem praktické části této práce je návrh a následná implementace nové verze knihovny libtrap, která bude lépe řešit přenos dat mezi moduly systému NEMEA. Implementované změny se týkají především výstupních rozhraní a způsobu, jakým tato rozhraní pracují s vyrovnávací pamětí. Dalším přínosem nové verze knihovny je rozšíření nástrojů pro správu systému NEMEA a diagnostiku chybně se chovajících modulů.

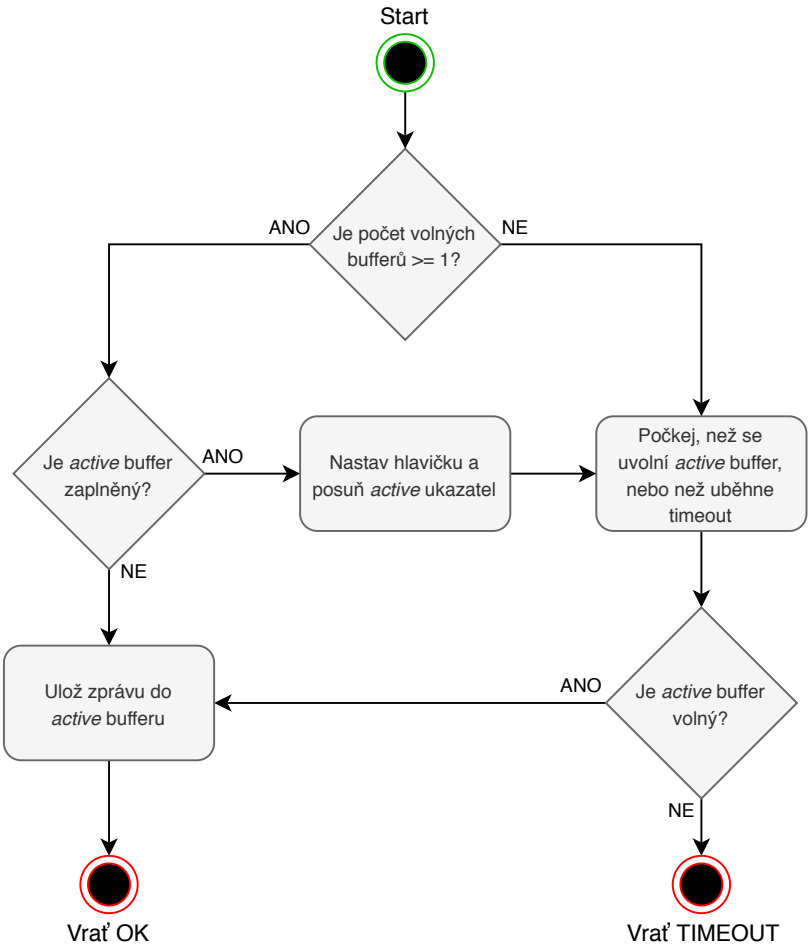
4.1 Nový princip bufferování a odesílání

Pro většinu komunikace v systému NEMEA jsou využívána rozhraní typu TCP/IP, TLS a UNIX. Tato rozhraní typicky pracují s více připojenými klienty a vzniká u nich tedy problém popsaný v Sekci 3.5, plynoucí z nepředvídatelných výkyvů v rychlosti čtení zpráv různými NEMEA moduly.

V nové verzi knihovny libtrap je tento problém vyřešen přepracováním výstupních rozhraní TRAP. Modulům je umožněno při inicializaci kontextu specifikovat velikost vyrovnávací paměti, respektive počet a velikost bufferů pro každé výstupní rozhraní. Další hlavní změna je paralelizace ukládání zpráv a odesílání bufferů.

Dohromady tyto změny umožňují modulům použít více operační paměti za účelem minimalizace ztráty dat při zachování propustnosti daného rozhraní. Tato vlastnost je klíčová u modulů jako jsou kolektory, které slouží jako zdroj dat celého systému a je u nich tedy vyžadována vysoká propustnost.

V ideálním případě (tedy za předpokladu, že je vždy k dispozici alespoň jeden volný buffer) modul při volání funkcí pro odesílání (`trap_send()` a `trap_ctx_send()`) nikdy nečeká na uložení zprávy, zároveň ale nedochází ke ztrátě dat.

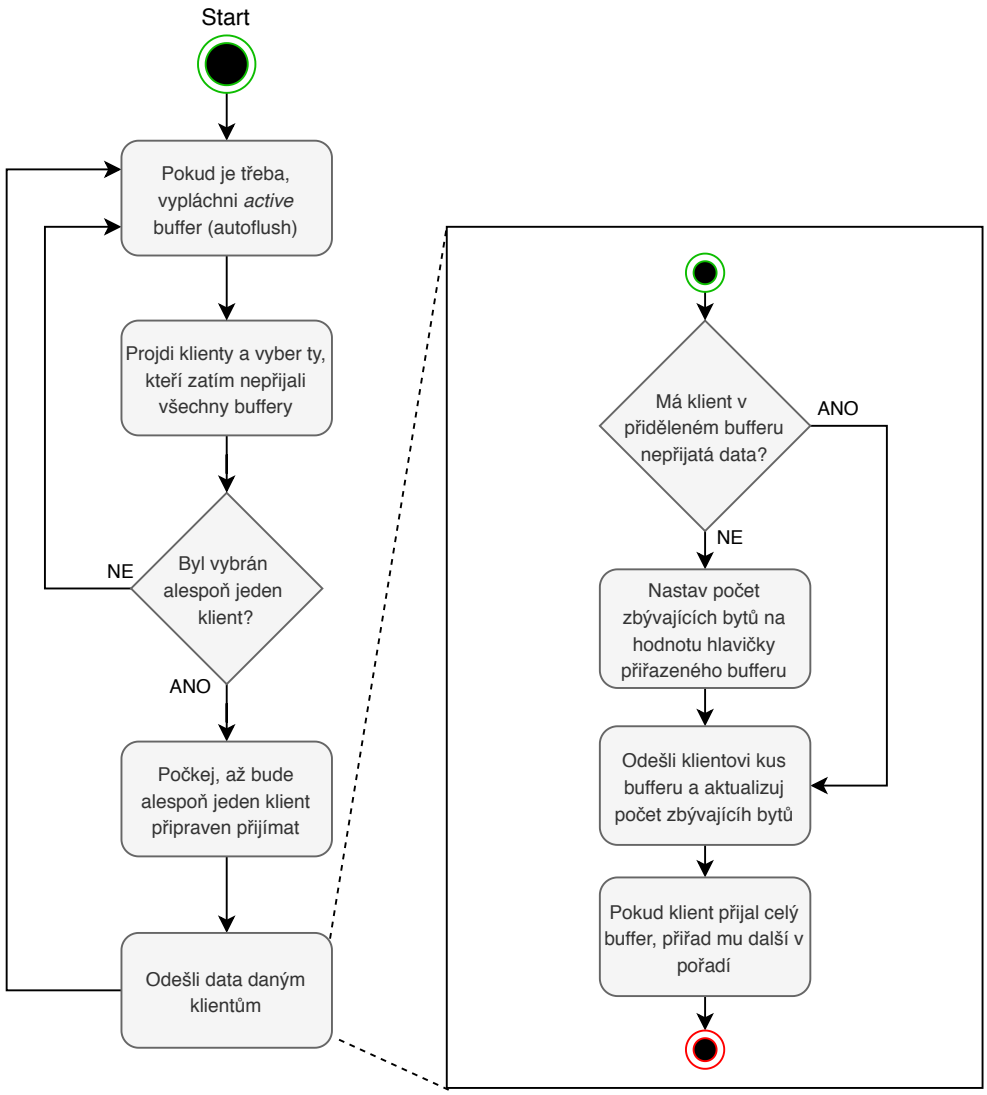


Obrázek 4.1: Návrh algoritmu ukládání zpráv pro *socket-based* rozhraní (TCP/IP, TLS, UNIX)

4.1.1 Socket-based rozhraní

Obrázek 4.1 popisuje algoritmus ukládání zpráv do vyrovnávací paměti, který je v nové verzi knihovny používán pro *socket-based* rozhraní. Rozhraní přistupuje k vyrovnávací paměti přes pole ukazatelů na jednotlivé buffery. Jeden z bufferů je vždy označen jako *active*. Příchozí zprávy jsou ukládány právě do tohoto bufferu.

Když je buffer připraven k odeslání (je již zcela zaplněn, nebo má být vypláchnut), je jeho velikost uložena do hlavičky (Obrázek 3.2) a active ukazatel (index) je posunut na další buffer v pořadí. S tímto bufferem smí dále manipulovat pouze *odesílací vlákno* (Obrázek 4.2), které ho po odeslání všem připojeným klientům opět uvolní.



Obrázek 4.2: Návrh algoritmu odesílacího vlákna

Buffery jsou plněny sekvenčně, tedy od bufferu s nejnižším indexem v poli po ten s nejvyšším indexem. Když dojde k naplnění posledního bufferu v poli, je active ukazatel přesunut zpět na první buffer.

Tento mechanismus byl zvolen z toho důvodu, že výpočetní náročnost ukládání zpráv je nezávislá na velikosti pole bufferů, jelikož přepnutí active bufferu vyžaduje vždy pouze konstantní počet operací. Zároveň se při odesílání (pokud není na příchozí zprávu místo) vždy čeká na buffer obsahující nejstarší (nejdříve přijatá) data. Nemusí se tedy zvlášť ošetřovat, aby zprávy/buffery ke klientům dorazili ve správném pořadí.

Obrázek 4.2 popisuje algoritmus používaný odesílacím vláknem, které je součástí každého výstupního TRAP rozhraní (s výjimkou *file* rozhraní). Hlavím úkolem tohoto vlákna je odesílat zaplněné buffery připojeným klientům. Každý klient má přidělený nějaký buffer, ze kterého čte, a pamatuje si počet chybějících bajtů. Když je tento počet roven nule, je klientovi přidělen další buffer v pořadí.

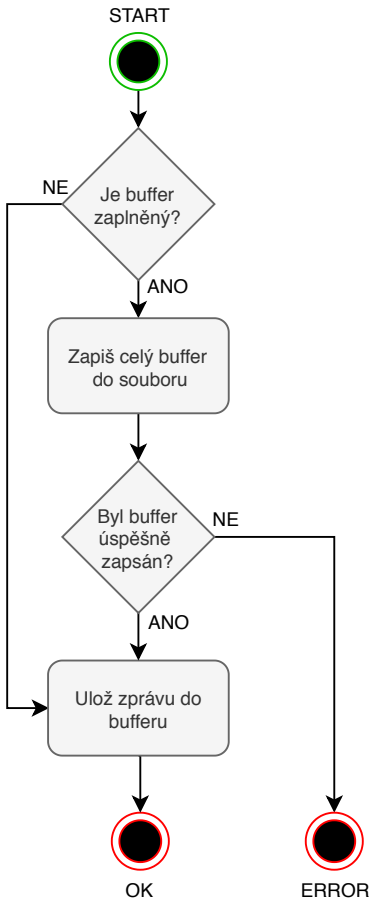
Pořadí, ve kterém jsou klientům přiřazovány buffery, odpovídá pořadí plnění bufferů popsaném výše. Nově připojeným klientům je přidělen současný *active* buffer.

Odesílací vlákno bylo v nové verzi knihovny sloučeno s *timeout* vláknem, které se stará o tzv. *autoflush* (Sekce 3.4), což je funkce libtrap, zajišťující, aby data nebyla uchovávána ve vyrovnávací paměti příliš dlouho v případě, že nové zprávy jsou přijímány s nízkou frekvencí.

4.1.2 File rozhraní

Obrázek 4.3 znázorňuje, jak nová verze knihovny řeší odesílání přes rozhraní typu File. Na rozdíl od socket-based rozhraní popsaných výše toto rozhraní není založené na klient-server architektuře a pracuje najedou vždy pouze s jedním souborem.

Proto je zde použit princip odesílání založený na současné implementaci knihovny libtrap. Používá se vždy právě jeden buffer, jehož odesílání (respektive zápis do souboru) probíhá v rámci stejného volání funkce `file_send()`, místo zvláštního odesílacího vlákna.



Obrázek 4.3: Návrh algoritmu bufferování/odesílání zpráv pro *File* rozhraní

4.2 Změny v architektuře

Dalším rozdílem oproti současné verzi knihovny libtrap jsou změny v její architektuře, plynoucí z návrhu bufferování a odesílání zpráv přes výstupní rozhraní, popsaném v předchozí sekci.

V současné verzi je bufferovací vrstva z logického pohledu nad interface vrstvou (Obrázek 3.3) a je společná pro všechny typy rozhraní.

Podobně jako objektově orientované programovací jazyky pracují s abstraktními třídami, pracuje bufferovací vrstva libtrap s rozhraními jako s *abstraktními objekty*. To znamená, že vyšší vrstvy TRAP jsou od implementace nižších vrstev zcela odstíněny a k rozhraním přistupují pouze přes obecnou strukturu, obsahující položky (proměnné, struktury) společné pro všechny typy rozhraní.

Toto řešení je výhodné zejména proto, že výrazně zjednodušuje vývoj nových typů rozhraní a úpravy těch existujících, jelikož každé rozhraní implementuje pouze části specifické pro danou formu datového přenosu.

Upravená implementace TCP/IP, UNIX a TLS rozhraní je však s principem bufferování nedělitelně spojena. Je totiž nezbytné, aby přístup ke strukturám vyrovnávací paměti mělo i odesílací vlákno (Obrázek 4.2), které pracuje na vrstvě rozhraní.

Použití stejné metody bufferování pro ostatní rozhraní však nemusí dávat smysl. Například u rozhraní typu File, které na rozdíl od ostatních nepracuje s více klienty (soubory) zároveň, by použití stejného principu nepřineslo žádné výhody a naopak by mohlo dojít k nežádoucímu snížení efektivity (maximální propustnosti).

Z tohoto důvodu byla architektura knihovny upravena. Bufferovací vrstva byla sloučena s vrstvou rozhraní a každé výstupní rozhraní tedy může používat odlišný způsob bufferování a/nebo odesílání dat.

4.3 Rozšíření diagnostických nástrojů

Každý NEMEA modul implicitně obsahuje speciální, tzv. *servisní* rozhraní (Sekce 1.2.3), které umožňuje získávat různé statistiky z TRAP rozhraní. V nové verzi knihovny byly výpisy servisního rozhraní rozšířeny o několik údajů, které zjednoduššují identifikaci a diagnostiku chybně se chovajících modulů:

Client ID Výstupní rozhraní nyní ukládá identifikaci připojených klientů — číslo portu pro TCP/IP a TLS rozhraní, PID (process ID) pro UNIX rozhraní. Tento údaj zjednodušuje identifikaci modulů, které jsou na dané rozhraní připojeny.

Client timers Pro klienty připojené na výstupní rozhraní je zaznamenáván čas strávený odesíláním dat. Tato statistika udává počet mikrosekund strávených ve funkci `send()` při odesílání dat danému klientovi. U každého klienta jsou ukládány dva údaje, a sice délka čekání při posledním volání a součet těchto hodnot od navázání spojení.

Client timeouts Pro každého klienta je dále ukládán počet zpráv, které byly zahozeny v důsledku zaplnění vyrovnávací paměti. Tato statistika umožňuje identifikovat klienty (moduly), kvůli kterým dochází na výstupním rozhraní k zahazování nových zpráv.

Module delay U každého rozhraní se měří prodleva mezi odesíláním/čtením zpráv způsobená zpožděním daného modulu. Konkrétně jde o počet mikrosekund mezi posledním voláním funkcí `trap_send()`/`trap_recv()`. Tento údaj nabízí další možnost jak rozpoznat klienty, kteří přenos brzdí, od těch, kteří akorát čekají na nová data.

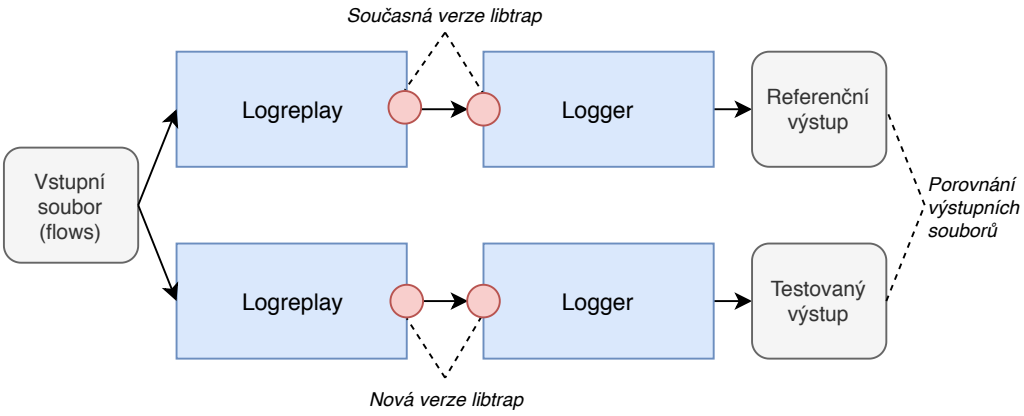
Kapitola 5

Testování

V této kapitole je popsáno testování implementovaných změn NEMEA frameworku, respektive knihovny libtrap. Dále je nová verze knihovny srovnána se současnou implementací z hlediska efektivity (propustnosti). Nakonec jsou probrány výsledky z nasazení vylepšené verze libtrap na testovacím kolektoru.

5.1 Ověření funkcionality implementace

Důležitou částí každé implementace je ověření návrhu, ze kterého vychází. V kontextu libtrap to znamená zejména ověření, že nově implementovaná výstupní rozhraní odesílají zprávy (respektive buffery) ve správném formátu, nepoškozené a že se (kromě případů, kdy je to dáno konfigurací) během přenosu žádná data neztrácí.



Obrázek 5.1: Mechanismus testování implementace nové verze libtrap

Pro tuto formu testování byla použita dvojice modulů *Logreplay* a *Logger*. Způsob, kterým bylo testování realizováno, je popsán na Obrázku 5.1.

Logreplay čte zprávy (flow záznamy) ze vstupního souboru ve formátu CSV a přeposílá je na výstupní rozhraní.

Logger přijímá zprávy ze vstupního rozhraní a vypisuje je na standardní výstup, případně je ukládá do specifikovaného souboru.

5.2 Měření propustnosti

Další fází testování je měření a porovnání maximální propustnosti obou verzí libtrap. Pro testování propustnosti je třeba použít moduly, které jsou samy o sobě velmi efektivní (tj. výpočetně nenáročné), aby zpoždění mezi zpracováním jednotlivých zpráv bylo způsobené pokud možno pouze voláním funkcí TRAP API a nikoliv modulem samotným. Pro tento účel byly implementovány testovací moduly *Sender–test* a *Client–test*:

Sender–test simuluje chování exportovacího modulu. Generuje a odesílá flow záznamy připojeným klientům. Kvůli minimalizaci zpoždění modulu je místo korektních UniRec zpráv odesílán náhodný kus paměti, což v tomto případě nevadí, jelikož se testuje pouze propustnost.

Client–test simuluje chování detekčního modulu. Má jedno vstupní rozhraní, ze kterého čte zprávy ve formátu UniRec. Místo jejich zpracování ale všechna příchozí data z důvodu minimalizace zpoždění zahazuje.

Měření probíhalo na výstupním rozhraní modulu *Sender–test* s deseti připojenými klienty (moduly *Client–test*) a pro datový přenos byl zvolen UNIX socket. Testování probíhalo na jednom stroji s těmito parametry: Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHz, 8 GB RAM, Ubuntu 18.04.1 LTS. Výsledky měření jsou znázorněny v Tabulce 5.1.

Tabulka 5.1: Srovnání maximální propustnosti — pro referenční měření byla použita současná verze libtrap

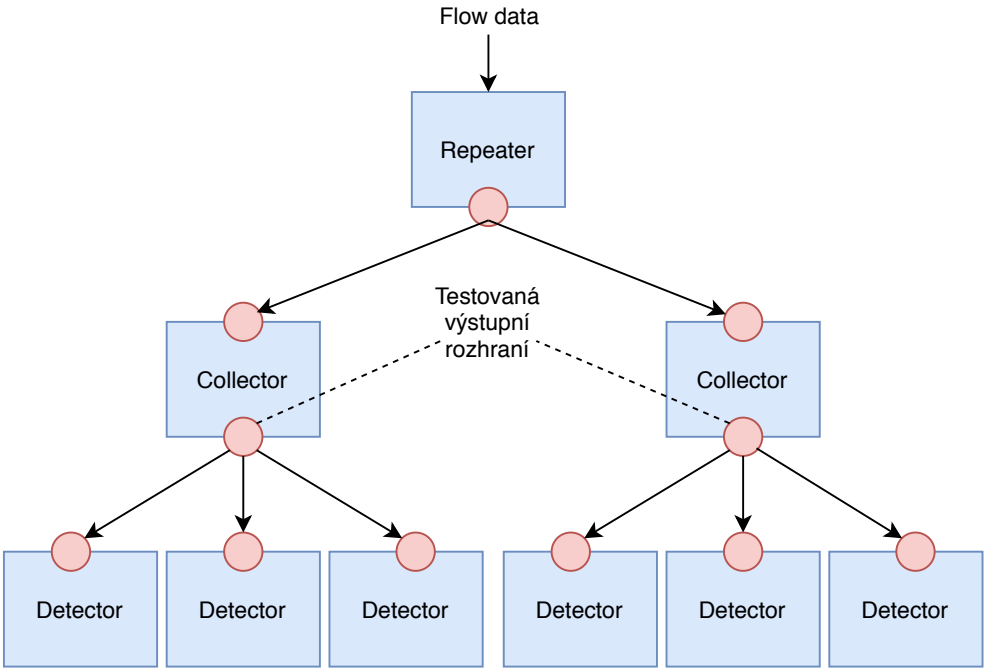
Vyrovnávací paměť [KB]	Propustnost [zprávy/s]		Průměrné zpoždění [ns]	
100	2 625 447	100 %	232	100 %
200	2 765 897	+ 5.3 %	215	− 7.3 %
1 000	2 856 292	+ 8.8 %	202	− 12.9 %
10 000	2 955 061	+ 12.6 %	184	− 20.7 %

5.3 Nasazení na testovacím kolektoru

Jelikož je síťový provoz nepředvídatelný a nestálý, je jeho simulace pro testovací účely velmi složitá. Proto byla nová verze knihovny nasazena na testovacím kolektoru, provozovaném sdružením CESNET, z.s.p.o.

Kolektor je modul, který přijímá data z měřících sond rozmístěných po monitorované síti. Z přijatých dat vytváří flow záznamy, které následně rozesílá ostatním modulům.

Cílem této fáze testování bylo ověření funkcionality návrhu v reálném provozu a srovnání výkonnosti obou verzí libtrap při skutečném nasazení. Testovací topologie systému je znázorněna na Obrázku 5.2.



Obrázek 5.2: Topologie nasazení systému NEMEA na testovacím kolektoru

Na testovacím stroji byly současně spuštěné dvě identické instance systému NEMEA s tím rozdílem, že na jednom z kolektorů byla použita nová verze knihovny libtrap s větší vyrovnávací pamětí (2 MB oproti současným 100 KB). Data byla získána z výstupních rozhraní testovaných kolektorů pomocí nástroje *trap_stats*. Tento nástroj umožňuje připojit se k běžícímu modulu přes servisní rozhraní a získávat různé statistiky (Sekce 1.2.3).

Výsledky zaznamenané zhruba za 80 hodin provozu jsou popsány v Tabulce 5.2. Z tabulky je vidět, že kolektor využívající novou verzi knihovny odeslal během testovacího úseku zhruba o 60 milionů zpráv více. To v časovém rozpětí tohoto testu představuje poměrně nevýznamný nárůst (0,15 %), důležité však je, že propustnost rozhraní v důsledku využití upraveného mechanismu odesílání zpráv neklesla. Zásadním výsledkem je výrazné snížení počtu zpráv, které byly v důsledku vypršení časového limitu na jejich uložení zahozeny.

Tabulka 5.2: Výsledky nasazení nové verze libtrap na testovacím kolektoru

	Odeslaných zpráv [mil.]	Odeslaných bufferů [tis.]	Zahozených zpráv
Současná verze	39 219	31 001	24 907
Nová verze	39 280	31 051	79
Rozdíl	**+ 0,15 %**	**+ 0,15 %**	**− 99,68 %**

Závěr

Cílem této práce bylo na základě analýzy frameworku NEMEA navrhnout a implementovat vylepšení knihovny libtrap. NEMEA je modulární systém, určený k monitorování provozu a detekci anomálií na komplexních sítích a je založen na principu analýzy síťových toků v reálném čase. Pro propojení jednotlivých modulů tohoto systému jsou využívána komunikační rozhraní poskytovaná knihovnou libtrap. Motivací pro navržené změny byla optimalizace komunikace mezi nejvíce zatěžovanými moduly.

Dílčí součástí práce byla analýza současného řešení tohoto systému, respektive knihovny libtrap. Ta odhalila několik nedostatků v implementaci, které se týkají zejména ztráty přenášených dat v důsledku zahlcení detekčních modulů.

Praktická část práce se zabývala návrhem a implementací nové verze libtrap, která tyto nedostatky řeší. Minimalizace nežádoucí ztráty dat bylo dosaženo použitím upraveného mechanismu ukládání zpráv do vyrovnávací paměti a jejich odesílání přes výstupní rozhraní. Další výhodou oproti současné verzi je rozšíření prostředků pro správu systému a diagnostiku chybně se chovajících modulů, které libtrap poskytuje.

Implementace navržených vylepšení byla důkladně otestována a nová verze knihovny byla porovnána s tou současnou z hlediska maximální propustnosti výstupních rozhraní. Nasazení nové verze knihovny na testovacím kolektoru prokázalo funkčnost návrhu — použitím upravených výstupních rozhraní s větší vyrovnávací pamětí byl výrazně snížen počet zahazovaných zpráv při zachování celkové propustnosti.

Literatura

[1] Čejka, T.; Bartoš, V.; Svepeš, M.; Rosa, Z.; Kubátová, H.: NEMEA: A Framework for Network Traffic Analysis. *International Conference on Network and Service Management (CNSM 2016)*, 2016, doi:10.1109/CNSM.2016.7818417. Dostupné z: `http://dx.doi.org/10.1109/CNSM.2016.7818417`

[2] Bartoš, V; Žádník, M; Čejka, T.: Nemea: Framework for streamwise analysis of network traffic. *CESNET Technical Report 9/2013*, 2013. [online] [cit. 2019-04-18] Dostupné z: `https://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf`

[3] Claise, B.; Trammell, B.; Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, IETF RFC 7011, 2013. [online] [cit. 2019-04-22] Dostupné z: `https://tools.ietf.org/html/rfc7011`

[4] CESNET, z. s. p. o: General Information; NEMEA: System for network traffic analysis and anomaly detection. [online], [cit. 2019-4-22]. Dostupné z: `https://nemea.liberouter.org/`

[5] CESNET, z. s. p. o: TRAP Interface Specifier; NEMEA: System for network traffic analysis and anomaly detection. [online], [cit. 2019-5-08]. Dostupné z: `https://nemea.liberouter.org/trap-ifcspec`

Příloha A

Seznam použitých zkratek

API Application Programming Interface

TCP Transmission Control Protocol

IP Internet Protocol

TLS Transport Layer Security

DDoS Distributed Denial of Service

PID Process Identifier

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `BP_Barnat_Matej_2019.pdf` ..... text práce ve formátu PDF