VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO OCHRANU PŘED DOS ÚTOKY S VYUŽITÍM IDS

SYSTEM FOR THE PROTECTION AGAINST DOS ATTACKS USING IDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **IGOR MJASOJEDOV**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN KUČERA**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Mjasojedov Igor**
Program: Informační technologie
Název: **Systém pro ochranu před DoS útoky s využitím IDS**
System for the Protection against DoS Attacks Using IDS
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se se systémy IDS pro detekci bezpečnostních hrozeb v síti, s problematikou útoků typu odepření služby (DoS) a zařízením vyvíjeným v rámci sdružení CESNET pro ochranu před těmito útoky.
2. Analyzujte možnosti využití existujících IDS systémů k potlačení DoS útoků. Zaměřte se především na systém Suricata.
3. Navrhněte způsob integrace IDS systému do existujícího zařízení.
4. Implementujte systém dle návrhu.
5. Vyhodnoťte implementované řešení z hlediska dosažených vlastností.
6. V závěru diskutujte výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kučera Jan, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

Táto bakalárska práca sa zameriava na využitie systému IDS pri ochrane počítačových sietí pred útokom typu DoS. IDS systémom využívaným v tejto práci je systém Suricata. Hlavným cieľom práce je integrácia tohto IDS systému do zariadenia DDoS Protector. Zariadenie DDoS Protector – DCPro využíva pre vysokorýchlostné spracovanie sieťovej prevádzky technológiu DPDK. Z tohto dôvodu bola podpora tejto technológie implementovaná aj do systému Suricata. Tým sa následne taktiež umožnila komunikácia tohto systému so systémom DCPro. Vo výsledku je umožnený chod systému Suricata v dvoch režimoch podporujúcich technológiu DPDK. V samostatnom režime systém Suricata funguje bez závislosti na inom systéme a spracúva pakety priamo z fyzického portu. Druhým režimom je režim integrovaného chodu, kedy systém Suricata spracúva pakety, ktoré mu preposiela systém DCPro. Takýmto spôsobom má systém DCPro možnosť preposielať pakety na podrobnejšiu analýzu do systému Suricata a tým sa jeho celkové možnosti detekcie sieťových útokov výrazne rozšírili.

Abstract

This bachelor's thesis deals with the use of the Intrusion Detection System in the protection of computer networks against Denial of Service attacks. Suricata is the IDS system chosen for this purpose. The main goal of the thesis is to integrate the Suricata system with the DDoS Protector device. DDoS Protector – DCPro is a security network device, which uses, from a software perspective, DPDK technology for high-speed network traffic processing. Due to this fact, this technology was also integrated into the Suricata system. After this integration, the communication between DDoS Protector and Suricata system was allowed more easily. As a result, two DPDK compatible regimes were created in the Suricata system. The individual regime allows Suricata to process network data directly from the network interface card. The second, integrated regime allows DCPro to send network data to the Suricata system for highly precise analysis, which significantly extends DDoS Protector's attack detection abilities.

Kľúčové slová

DoS, DDoS, DDoS Protector, IDS, IPS, Suricata, DPDK

Keywords

DoS, DDoS, DDoS Protector, IDS, IPS, Suricata, DPDK

Citácia

MJASOJEDOV, Igor. *Systém pro ochranu před DoS útoky s využitím IDS*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

Systém pro ochranu před DoS útoky s využitím IDS

Prehlásenie

Vyhlasujem, že som túto bakalársku prácu vypracoval samostatne a pod vedením pána Ing. Jana Kučeru. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje z ktorých som čerpal.

.....
Igor Mjasojedov
4. júna 2020

Poďakovanie

Rád by som poďakoval vedúcemu práce Ing. Janovi Kučerovi za venovaný čas, trpezlivosť a odbornú pomoc, ktorú mi poskytoval v priebehu celého vypracovania práce.

Obsah

1 Úvod **2**

2 Teoretický rozbor **4**
2.1 Princípy počítačových sietí 4
2.2 Útok typu odoprenie služby 7
2.3 Zariadenie DDoS Protector 9
2.4 Systémy pre detekciu bezpečnostných hrozieb 10
2.5 Suricata 13
2.6 Data Plane Development Kit 17

3 Konceptuálny návrh integrácie **22**
3.1 Štruktúra systému Suricata 23
3.2 Samostatný režim 25
3.3 Integrovaný režim 26

4 Popis implementácie **29**
4.1 Štruktúra modulov systému Suricata 29
4.2 Mód `ethdev` 32
4.3 Módy `ethdev-ring` a `native-ring` 33

5 Testovanie a meranie výkonnosti **36**
5.1 Testovacie prostredie 36
5.2 Samostatný beh systému Suricata 39
5.3 Beh integrovaného systému Suricata so systémom DCPro 40
5.4 Zhrnutie dosiahnutých výsledkov 44

6 Záver **46**

Literatúra **48**

A Obsah priloženého pamäťového média **50**

Kapitola 1

Úvod

V dnešnej dobe je Internet neodmysliteľnou súčasťou každodenného života a to so sebou prináša aj bezpečnostné riziká. Ochrana počítačových sietí je teda stále aktuálnejšou témou. Útočníci sa pri útokoch na počítačové siete snažia o odoprenie služieb legitímnym užívateľom a to rôznymi spôsobmi. To môže mať za následok obrovské finančné škody pre poskytovateľov služieb, ku ktorým sa ich zákazníci nemôžu dostať. Počet zariadení pripojených k Internetu z roka na rok výrazne stúpa. To má za následok vyššie objemy prenášaných dát a našou úlohou je pripraviť také sieťové zariadenia, ktoré tieto objemy zvládnu spracovávať po všetkých stránkach a prípadné útoky aktívne eliminovať.

K dosiahnutiu bezpečnosti počítačových sietí služia mnohé aktívne zariadenia, ktoré sa nasadzujú do infraštruktúry jednotlivých sieťových blokov. Medzi takéhto bezpečnostné zariadenia patrí aj zariadenie DDoS Protector, ktoré sa vyvíja pod záštitou združenia CESNET. Toto zariadenie slúži na zabezpečenie sietí pred útokmi zameranými na odoprenie služby legitímnym užívateľom. Okrem hardvérovej podpory spracovávania vysokorýchlostných dátových prenosov disponuje dané zariadenie aj softvérovými modulmi, ktoré slúžia na detekciu a filtráciu niektorých typov útokov.

Táto bakalárska práca sa zaoberá zabezpečením počítačových sietí prostredníctvom využitia systémov IDS. Systémy IDS taktiež slúžia na detekciu a filtráciu prebiehajúceho útoku v sieti na základe ich konfigurovaných pravidiel. Výhodou týchto systémov je ich podrobná analýza prenášaných dát a teda možnosť detekcie aj sofistikovanejších útokov ľubovolného typu. V mojej práci sa venujem rozšíreniu zariadenia DDoS Protector o tieto detekčné schopnosti systémov IDS. Toto rozšírenie dosiahnem integráciou systému DDoS Protector so známym IDS systémom Suricata, ktorého spoľahlivosť je už rokmi overená.

V nasledujúcej kapitole tejto práce je popísaný teoretický základ počítačových sietí, typ útoku odoprenia služby, zariadenie DDoS Protector, ale aj popísaný celkový princíp fungovania systémov IDS. Predposledná sekcia tejto kapitoly obsahuje podrobné informácie o konkrétnom systéme IDS, ktorý v práci využívam. Jedná sa o IDS systém Suricata. Záver tejto kapitoly je venovaný technológii DPDK, ktorá poskytuje možnosti vysokorýchlostného spracovania dát prenášaných v počítačovej sieti. Túto technológiu taktiež využíva zariadenie DDoS Protector a jej využitím v systéme Suricata dosiahnem požadované ciele práce.

Tretia kapitola sa venuje konceptuálnemu návrhu pridania podpory technológie DPDK do systému Suricata a následne aj návrhu integrácie systému Suricata do zariadenia DDoS Protector. Po tejto kapitole nasleduje popis implementačných detailov navrhnutého riešenia. Nasledujúca piata kapitola sa zameriava na prezentáciu výsledkov z výkonnostného testovania funkčného systému. Jednotlivé testy sú vykonávané ako na samostatnom chode systému Suricata po pridaní podpory technológie DPDK, tak aj na chode celého systému po

rozšírení zariadenia DDoS Protector o tento systém. Úplný záver tejto práce popisuje postup, ktorým sa dosiahol požadovaný výsledok. Taktiež sa v záverečnej kapitole spomínajú možné rozšírenia nad takto integrovaným systémom.

Kapitola 2

Teoretický rozbor

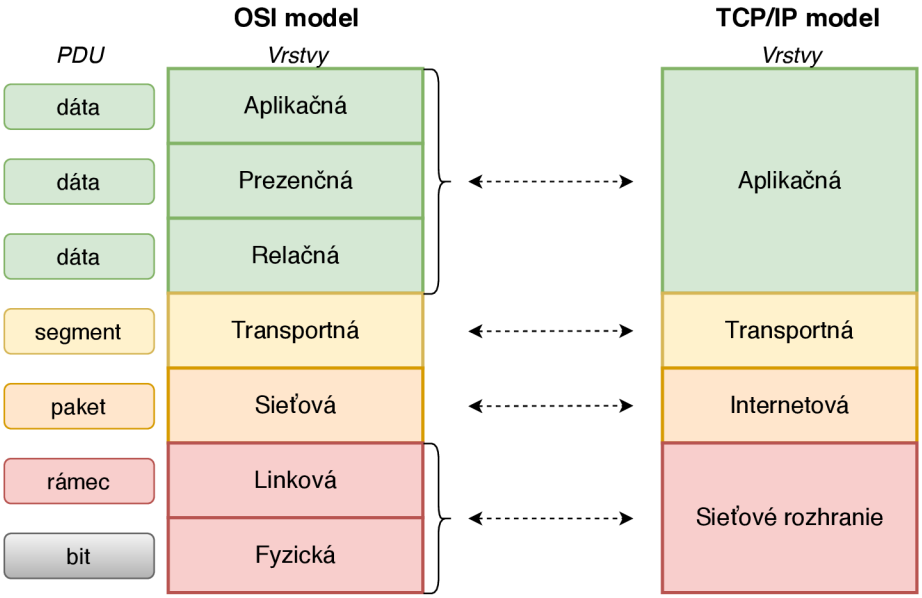
V nasledujúcej kapitole sú zhrnuté a popísané hlavné pojmy a technológie, ktoré sa využívajú v tejto bakalárskej práci. Základné informácie o každej z nich poslúžia aj menej orientovanému čitateľovi v tejto téme pochopiť to, o čom budú nasledujúce kapitoly zaoberajúce sa návrhom riešenia a aj samotnou implementáciou. V prvom rade sa v tejto kapitole predstavia základy a princípy počítačových sietí ako takých. Následne sa text venuje opisu najčastejšieho sieťového útoku, ktorým je DoS útok a taktiež sa predstaví projekt DDoS Protector, ktorý zabezpečuje ochranu pred takýmito útokmi. V ďalšej časti kapitoly budú opísané systémy prevencie a detekcie preniku a aj konkrétny voľne šíriteľný systém Suricata. Ako posledná technológia bude predstavená technológia DPDK.

2.1 Princípy počítačových sietí

Nepostrádateľnou znalosťou pre pochopenie textu tejto práce sú bezpochyby základné informácie o architektúre a princípoch počítačových sietí. Táto sekcia sa bude venovať práve týmto princípom. Informácie pre zhotovenie tejto sekcie som čerpal najmä z knihy [15].

Architektúra počítačových sietí sa od ich počiatkov navrhovala troma základnými komponentmi a to technológiami pre prenos signálu, vrstvou zabezpečujúcou spoľahlivý prenos a aplikačnou vrstvou užívateľských služieb. Avšak dnes už využívame štandardizované vrstvové modely, ktoré definujú jednotlivé vrstvy, vzťahy medzi nimi a protokoly, ktoré nad týmito vrstvami operujú. Taktiež definujeme základné dátové jednotky tzv. PDU (Process Data Unit) pre každú vrstvu. Tieto dátové jednotky sú bloky informácií prenášané po sieti ako jeden celok pozostávajúci zo všetkých vrstevných blokov. Každá vrstva má nastarosti svoj blok informácií s ktorým pracuje. Ku koncu tejto sekcie bude nasledovať podrobnejší popis vzťahu jednotlivých dátových jednotiek a procesu ich zapuzdrenia.

Ako referenčný model počítačových sietí sa používa model OSI (Open Systems Interconnection) vypracovaný organizáciou ISO (International Standards Organisation) v roku 1987, pre podporu štandardizovania počítačových sietí. Model pozostáva zo siedmych vrstiev, ktoré sú jednotlivo popísané v nasledujúcej časti a znázornené aj s prísluchajúcimi PDU v obrázku 2.1.



Obr. 2.1: Porovnanie modelov OSI a TCP/IP.

Fyzická vrstva popisuje a definuje fyzické komponenty sieťovej infraštruktúry ako konektory a ich rozloženie pinov, vlastnosti prenosových liniek, rýchlosť prenosového média, kódovanie pre logickú 1 a 0. Hlavnou funkciou tejto vrstvy je fyzický prenos dát po danej prenosovej linke. Dátová jednotka fyzickej vrstvy je bit.

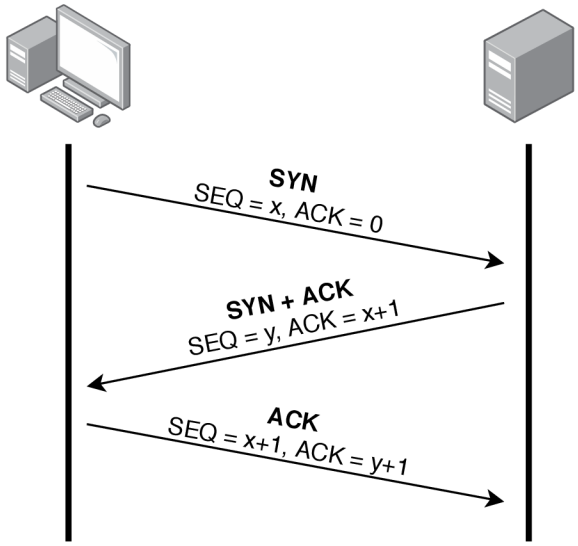
Linková vrstva umožňuje prístup k fyzickému médiu a definuje prenos dát po určitej prenosovej linke. Pre úspešný prenos dát je nutné vytvorenie tzv. rámcov, ktoré sú základnou dátovou jednotkou tejto vrstvy (PDU). Taktiež zabezpečuje adresovanie zariadení, ktoré má lokálny význam na danej linke a kontrolu úspechu nepoškodeného prenosu dát, prípadne aj ich následnú opravu. Najpoužívanejším protokolom tejto vrstvy je v dnešnej dobe protokol Ethernet.

Sieťová vrstva vytvára tzv. datagramy, často nazývané aj pakety, ktoré následne smeruje naprieč počítačovou sieťou. Pre správne smerovanie paketov je nutné poznať adresy cieľových uzlov. Táto podmienka je splnená práve v tejto vrstve a to prostredníctvom adresovania. Smerovanie paketov a ich doručovanie je s najvyšším úsilím (angl. best effort delivery), čo označuje vlastnosť prenosu kedy sa jednotlivé sieťové zariadenia tejto vrtvy snažia o nájdenie čo najkratšej cesty k cieľu. K nájdeniu najkratšej cesty sa využívajú smerovacie protokoly ako OSPF (Open Shortest Path First), BGP (Border Gateway Protocol), atď.. Avšak, spoľahlivosť prenosu nie je doménou tejto vrstvy a teda sa táto dôležitá vlastnosť prenosu prenecháva vrstve, ktorej je venovaná nasledujúca časť.

Transportná vrstva vytvára logické spojenie medzi procesmi aplikácií na zdrojovom a cieľovom zariadení. Adresovanie na tejto vrstve je zabezpečené prostredníctvom tzv. portov. Jedná sa o 16 bitové číslo, ktoré jednoznačne identifikuje komunikujúcu aplikáciu, resp. protokol. Hlavnou činnosťou tejto vrstvy je segmentácia aplikačných dát, riadenie toku dát a zabezpečenie spoľahlivého prenosu. Jednotlivé činnosti sa líšia od toho, ktorý

transportný protokol je práve používaný. Základnými protokolmi transportnej vrstvy sú protokoly TCP (Transmission Control Protocol) a UDP (User Datagram Protocol). V nasledujúcej časti bude podrobnejšie popísaný protokol TCP.

Protokol TCP je stavový a pred samotným prenosom dát vykonáva nadviazanie komunikácie medzi dvoma komunikujúcimi uzlami. Tento mechanizmus nadviazania spojenia sa nazýva „Three Way Handshake“. Ten funguje na základe výmeny niekoľkých paketov s určitým označením prostredníctvom tzv. indikátorov typu (angl. flags). Pakety, ktoré si na začiatku oba uzly vymieňajú, musia byť v správnom poradí. Na nasledujúcom obrázku 2.2 je možné vidieť nadviazanie komunikácie prostredníctvom tohto mechanizmu.



Obr. 2.2: Mechanizmus „Three Way Handshake“.

Prvý paket generuje klient a to paket s označením SYN (angl. Synchronize) v TCP hlavičke. Následne očakáva odpoveď označenú SYN a ACK (angl. Acknowledgement), teda synchronizáciu od servera a potvrdenie prijatého paketu. V tretej fáze, respektíve tretím paketom očakáva server odpoveď od klienta s označením ACK, teda potvrdením prijatia SYN+ACK paketu.

Relačná vrstva zabezpečuje správu relácií medzi komunikujúcimi aplikáciami.

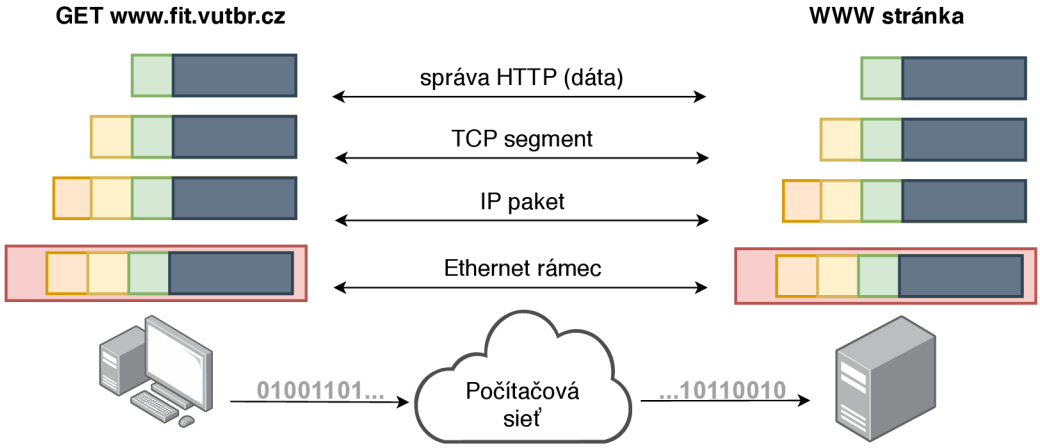
Prezentačná vrstva slúži k formátovaniu zasielaných aplikačných dát a naopak, umožňuje aplikácii porozumieť prijatým dátam v určitom formáte.

Aplikačná vrstva je najvyššia, siedma vrstva OSI modelu. Poskytuje rozhranie pre užívateľov k zariadeniu pripojeného k sieti prostredníctvom aplikácií.

Vyššie uvedené vrstvy sa vzťahujú k refenrenčnému modelu OSI. Avšak, model na ktorom je založený dnešný Internet je model TCP/IP. Jedná sa o protokolový profil, ktorý pozostáva zo štyroch vrstiev. Vzťah medzi referenčným modelom OSI a modelom TCP/IP je taktiež znázornený vo vyššie uvedenom diagrame 2.1. Model TCP/IP znižuje počet vrstiev tým, že spojuje najvyššie tri vrstvy OSI modelu do jednej a to aplikačnej vrstvy. Taktiež

spojuje najnižšie dve vrstvy z OSI modelu do vrstvy označovanej, ako vrstva sieťového rozhrania.

V nasledujúcom obrázku 2.3 je znázornený prenos dát medzi dvoma sieťovými zariadeniami a proces, ktorý tento presnos užívateľských dát sprevádza.



Obr. 2.3: Proces zapuzdrenia a rozbaľovania prenášaného paketu.

Pri procese prenosu dát dochádza k zapuzdreniu (angl. encapsulation) jednotlivých dátových jednotiek (PDU) vyšších vrstiev do tých nižších, až sa nakoniec takýmto spôsobom vytvorený paket prenáša po linke, ako signály reprezentujúce logické jednotky a nuly. V obrázku 2.3 sú užívateľom vytvorené dáta (blok tmavosivej farby), v tomto prípade zadané doménové meno `www.fit.vutbr.cz`, zapuzdrené do jednotlivých vrstiev. Jednotlivé vrstvy zodpovedajú farebnému odlíšeniu na základe predošlého obrázku 2.1. Zariadenie, ktoré prijíma takto prenášaný paket, vykonáva opačný proces, ktorým je rozbaľovanie dát (angl. decapsulation). Pri tomto procese sa paket postupne rozbaľuje od častí paketu pridaných jednotlivými vrstvami pri zapuzdrení, až získa dáta vytvorené v najvyššej a to aplikačnej vrstve na strane odosielateľa.

Po predstavení základných princípov a architektúry počítačových sietí je vhodné upriamiť pozornosť na horzby v nich a zároveň bezpečnosť s nimi spojenú. Nasledujúca sekcia sa bude venovať najčastejšiemu sieťovému útoku, ktorým je útok typu DoS.

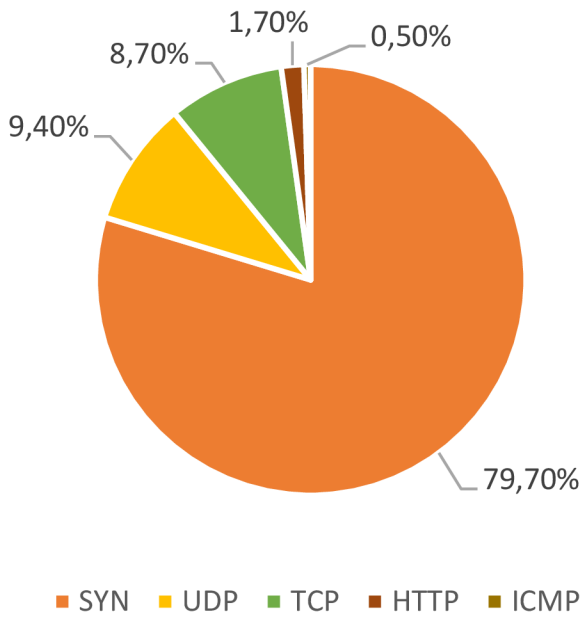
2.2 Útok typu odoprenie služby

Odoprenie služby (angl. Denial of Service - DoS) je typ útoku, ktorý znemožňuje prístup legitímnych užívateľov k internetovej službe. Tento útok spočíva v nadmernom generovaní požiadaviek na server, alebo na sieťovú infraštruktúru cieľa. Následne sa daný server, prípadne napádaná sieť vyťaží natoľko, že už nieje schopná spracovať požiadavky legitímnych užívateľov.

Pri termíne DoS uvažujeme o útoku, ktorý pochádza z jedného zdroja. Avšak, častejšie využívanou metódou je rozšírenie jedného zdroja útoku na mnohé útočníkom napadnuté zdroje, inak nazývané boty. Vykonávanie útoku distribuované na takto napadnuté zariadenia vytvára sieť napadnutých botov, ktorá sa nazýva botnet. Boty následne generujú ešte vačšie množstvo požiadaviek, čím sa sila útoku zvýši a tým sa efektívnejšie znemožní prístup k danému serveru, alebo sieťovej infraštruktúre obete. Táto modifikácia DoS útoku má

označenie DDoS, čo značí distribuované odoprenie služby (angl. Distributed Denial of Service). V nasledujúcom texte práce budem využívať označnie útoku typu odoprenie služby skratkou DDoS.

Tento typ útoku v počítačových sieťach je v dnešnej dobe jedným z najčastejších. Ako aj pri iných útokoch, aj DDoS útok sa rozdeľuje na rôzne varianty jeho priebehu. Nasledujúci graf 2.4 zobrazuje najčastejšie typy DDoS útokov za tretí kvartál roku 2019. Dáta k tvorbe tohto grafu boli získané z reportu vyhotoveným medzinárodnou spoločnosťou Kaspersky Lab, zaoberajúcou sa kybernetickými hrozbami [13].



Obr. 2.4: Najčastejšie typy DDoS útokov za Q3 2019.

Ako je z grafu evidentné, suverénne najčastejším DDoS útokom je útok typu SYN Flood. Jednotlivým typom útokov uvedených v grafe sa bude venovať nasledujúca časť textu.

SYN Flood Tento typ útoku sa zakladá na princípoch fungovania transportného protokolu TCP. Útočník využíva zraniteľnosť jeho mechanizmu „Three Way Handshake“ (obrázok 2.2) a to takým spôsobom, že generuje obrovské množstvo synchronizačných paketov (SYN) a tým nadväzuje spojenie so serverom každým jedným paketom. Server odpovedá paketmi SYN+ACK, avšak útočník už nepotvrdí úspešné nadviazanie spojenia a tým server zostáva v polootvorenom stave spojenia. Takýmto spôsobom sa vyťaží server natoľko, že už nie je schopný odpovedať, respektíve prijímať nové požiadavky o nadviazanie spojenia od legitímnych užívateľov. Tento útok môžeme rozdeliť na tri podtypy a to na priamy útok, spoofing útok, distribuovaný priamy útok [5].

HTTP Flood HTTP Flood je útok na aplikačnej vrstve, respektíve siedmej vrstve podľa referenčného modelu OSI. Cieľom útoku je vyčerpať zdroje na napádanom serveri. Generovaním veľkého množstva GET požiadaviek na server útočník docieli mnohokrát zdĺhavé operácie prístupu do databázy pre zostavenie výsledných stránok, ktoré následne odosiela vo svojej odpovedi. Aplikačné útoky sú náročné na zistenie, teda rozpoznanie legitímnej a nelegitímnej prevádzky [12].

ICMP Flood Jedná sa o jeden z najjednoduchších DDoS útokov, pri ktorom sa útočník snaží zaplaviť obeť ICMP-echo správami a tým spôsobiť zahltenie výpočetných zdrojov obete, alebo linky cez ktorú je obeť pripojená k Internetu.

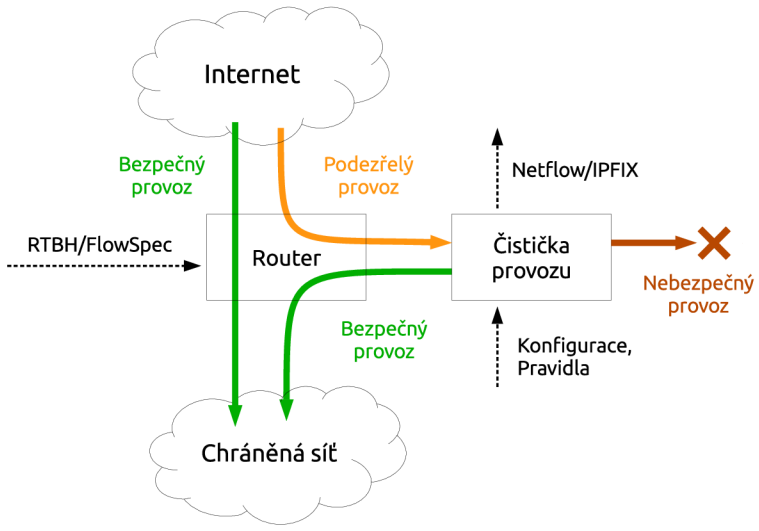
Práve zabezpečeniu pred útokom DDoS sa venuje zariadenie DDoS Protector. Nasledujúca sekcia 2.3 popisuje toto zariadenie.

2.3 Zariadenie DDoS Protector

Projekt DDoS Protector (DCPro) sa zameriava na vývoj centralizovaného zariadenia na detekciu a následné blokovanie DDoS útokov, ako je už zrejmé aj z názvu. Pod pojmom „centralizované“ sa má na mysli, že takéto zariadenie sa nachádza v danej sieti iba raz a celá podozrivá prevádzka sa smeruje naň. Tento návrh je výhodný z hľadiska konfigurácie a administrácie zariadenia. Avšak, dôsledkom toho je nevyhnutné, aby zariadenie disponovalo vysokou kapacitou z hľadiska priepustnosti a výkonnosti pri spracovávaní sieťovej prevádzky. Práve preto DDoS Protector využíva akceleračné sieťové karty s FPGA. Tento projekt je vyvýjaný združením CESNET. Všetky informácie zmienené v tejto sekcii vychádzajú z [6, 7].

DCPro figuruje v sieti, ako zariadenie na sieťovej vrstve OSI modelu. Smerovače v sieti sú konfigurované tak, aby časť nedôveryhodnej a rizikovej prevádzky smerovala práve do tohto zariadenia. Táto konfigurácia je zabezpečená prostredníctvom RTHB/FlowSpec. DCPro následne na základe pravidiel, ktoré sú špecifikované a nakonfigurované administrátorom siete zabezpečí detekciu a prípadne aj filtráciu nevhodných tokov. Následne sú o týchto tokoch dát zozbierané a vyhotovené štatistiky o počte bajtov, paketov a prostredníctvom Netflow/IPFIX sú exportované. Samozrejme, všetka platná a dôveryhodná prevádzka je smerovaná späť, do už chránenej siete.

V nižšie uvedenom obrázku 2.5, je znázorená topológia jedného z najčastejších zapojení DDoS Protectoru v samotnej infraštruktúre sieti.



Obr. 2.5: Schéma nasadenia zariadenia DDoS Protector v sieti [6].

Protector je zapojený združene so smerovačom, ktorý smeruje podozrivú prevádzku na sieti práve na DDoS Protector. V prípade, že sa jedná o nelegitímnu prevádzku sa

DCPro postará o jej vyfiltrovanie, stlmenie a následne sa bezpečná prevádzka smeruje späť do zabezpečenej sieti.

Existujú dve verzie zariadenia DDoS Protector. Prvá, staršia verzia pracuje s proprietárnym rozhraním pre príjem paketov a je úzko naviazaná na využívanie sieťových kariet FPGA. Existuje však framework DPDK (viď sekciu 2.6) pre spracovávanie sieťovej prevádzky, ktorý podporuje rôzne typy sieťových kariet. Aktuálne prebieha vývoj novšej verzie DCPro, ktorá bude postavená nad DPDK. Tým sa umožní viacero typov nasadenia a prevádzky tohto zariadenia, než pri predošlej verzii. V mojej práci som pracoval prevažne s touto verziou DCPro.

Tak, ako naznačuje už názov tejto bakalárskej práce, v zariadení pre ochranu pred DoS útokmi (DDoS Protector – DCPro) bude využitý IDS/IPS systém (sekcia 2.4). Dôvod integrácie tohto typu systému do zariadenia DCPro je rozšírenie jeho možností analýzy nad najrôznejšími typmi sieťových útokov. Systémy IDS/IPS umožňujú vykonávať analýzu a filtráciu až na aplikačnej vrstve L7 OSI modelu (obrázok 2.1), avšak takáto hĺbková analýza paketov je výkonnostne veľmi náročná. Na druhú stranu, zariadenie DDoS Protector pracuje maximálne len do sieťovej L3 vrstvy a vďaka hardvérovej akcelerácii dosahuje vysokého výkonu. Integráciu systému IDS/IPS by sa zariadenie DDoS Protector stalo univerzálnejším. Masívne volumetrické útoky by sa podrobili hrubej filtrácii priamo súčasným riešením tohto zariadenia a to jeho DPDK verziou. Avšak, útoky s menším objemom sieťovej prevádzky, by bolo možné spracovať systémom IDS/IPS a následne na základe jemnej analýzy odflitrovať. Nasledujúca sekcia 2.4 bude venovaná opisu práve tohto typu systému, systému IDS/IPS.

2.4 Systémy pre detekciu bezpečnostných hrozieb

Systém detekcie prieniku (angl. Intrusion Detection System - IDS) je systém pre ochranu počítačových sietí, ktorý monitoruje a analyzuje dátové toky v sieti. Na základe detekčných metód, ktoré budú popísané v nasledujúcom odseku, systém zistí, či sa jedná o legitímnu, alebo o škodlivú prevádzku. Avšak, tento systém nedokáže zabrániť útoku samostatne, bez nutnosti zásahu správcu siete. Pri detekcii útoku, systém zasiela upozornenie správcovi siete, ktorý je zodpovedný za vykonanie príslušného opatrenia pre zmarenie prebiehajúceho útoku. Informácie obsiahnuté v nasledujúcej časti som čerpal z [8, 10, 20].

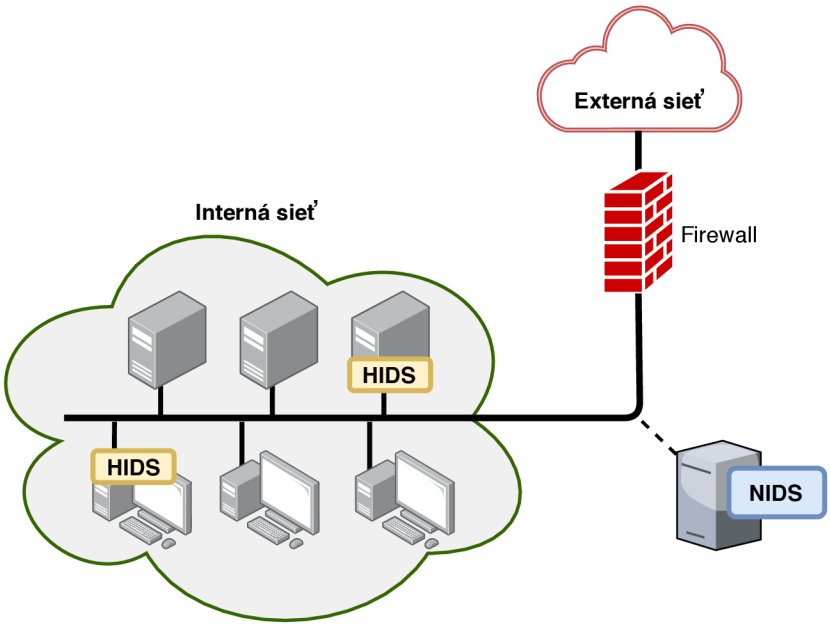
Ako už bolo spomenuté, pre detekciu útoku sa systémami IDS využívajú tzv. detekčné metódy. Najčastejšie používané sú metódy založené na porovnávaní signatúr a na základe monitorovania anomálií v sieťovej prevádzke. Bližšie informácie o oboch typoch detekcií nájdeme v nasledujúcej časti.

Detekcia na základe signatúr Pri tejto metóde sa využívajú predkonfigurované vzory, tzv. signatúry rôznych útokov, s ktorými sa následne porovnáva monitorovaná sieťová prevádzka. Korektnosť signatúr je kľúčová pre úspešné odhalenie aktivity útočníka. V prípade, že sa objaví nový typ útoku, je nutné vytvoriť nové, prípadne modifikovať pôvodné signatúry.

Detekcia na základe anomálií Detekcia útoku je zaistená hľadaním abnormálnej sieťovej prevádzky. Tento typ detekcie teda dokáže zistiť útok aj bez nutnosti jeho špecifikácie. Avšak, nevýhodou tejto metódy je jeho náchylnosť na produkciu falošných alarmov, čo môže spôsobovať problémy v prípade, ak je na sieti implementovaný systém IPS, ktorému bude

venovaný záver tejto sekcie.

Systémy detekcie prieniku (IDS) môžeme kategorizovať aj na základe miesta ich umiestnenia, resp. implementácie v počítačovej sieti. Nasledujúce časť popisuje dva základné typy IDS podľa toho, na akom mieste v sieti sa bude detekcia vykonávať. Obrázok 2.6 znázorňuje rozdiel umiestnenia v sieti týchto dvoch typov IDS, Host-Based IDS a Network-Based IDS.



Obr. 2.6: Schéma zapojenia systému HIDS a NIDS.

Host-based Intrusion Detection System (HIDS) Detekcia a monitorovanie prebieha nad jedným konkrétnym systémom, napríklad nad zabezpečeným serverom, alebo klientskou pracovnou stanicou (viď žltý blok v obrázku 2.6). HIDS monitoruje jedine systém, na ktorom je nasadený a tým pádom jedine sieťovú komunikáciu daného systému. Tento typ IDS je ťažšie udržiavateľný vo väčších počítačových sieťach s rôznymi operačnými systémami a konfiguráciami pracovných staníc, alebo serverov.

Network-based Intrusion Detection System (NIDS) Vhodnejšia alternatíva je typ Network IDS. Tento typ IDS pozostáva zo sieťového zariadenia, ktoré je zapojené do infraštruktúry monitorovanej siete (viď modrý blok v obrázku 2.6). Vďaka umiestneniu na vhodné miesto v sieti je možné monitorovať a následne detegovať útoky nad celou počítačovou sieťou. Vhodným miestom pre zapojenie tohto zariadenia je miesto, cez ktoré zaručene prechádza celá prevádzka monitorovanej siete. Takto zapojené zariadenie potom dokáže monitorovať celú prevádzku a funguje ako sonda a celá prevádzka sa kopíruje a preposiela naň.

Okrem systémov detekcie prieniku (IDS), definujeme aj systémy prevencie prieniku (angl. Intrusion Prevention Systems - IPS). Systém prevencie prieniku je tak ako aj IDS, schopný detekcie útoku, ale je taktiež schopný aj vykonania potrebných opatrení pre od-

stránenie a eliminovanie útoku, bez nutnosti zásahu správcu siete. To je hlavný rozdiel medzi IDS a IPS. V prípade útoku, IPS okamžite zasahuje, čo so sebou prináša aj určité riziká, napríklad ak IPS deteguje útok aj vtedy, keď sa o skutočný útok nejedná. Následne po aplikovaní opatrení môže dôjsť k vytvoreniu problémov samotným IPS.

Existuje mnoho open-source IDS/IPS systémov. Vzhľadom na ich mierne odlišnosti a odôvodnenie výberu mnou používaného IDS/IPS systému v tejto práci, sa nasledujúca časť bude venovať popisu tých najpopulárnejších. V nasledujúcej časti boli využité informácie z práce [14].

Snort

Snort [19] je sieťovo založený (NIDS) systém prevencie a detekcie prieniku. Založený je predovšetkým na princípoch detekcie na základe signatúr. Jeho hlavným vývojárom je spoločnosť Cisco. Jedná sa o najpopulárnejší NIDS systém so širokou podporou. Popularitu si získal aj na základe denne aktualizovanej databázy pravidiel, ako napríklad EmergingThreats [18]. Príklad časti pravidla z tejto databázy, ktoré detekuje pokus o SYN Flood (sekcia 2.2) DDoS útok, je uvedený nižšie:

```
alert tcp 10.10.0.0/24 10000: -> 192.168.1.0/24 0:1023 (msg:ËT DDOS
  Potential Tsunami SYN Flood Denial Of Service Attempt"; flags:S;
    dsize:>900; threshold: type both, count 20, seconds 120, ...)
```

Pravidlo pozostáva troch častí – akcia, hlavička, telo pravidla. Akcia, ktorá sa má vykonať, ak paket odpovedá danému pravidlu (`alert`). V druhej časti nasleduje hlavička, ktorá obsahuje transportný protokol (`tcp`), zdrojový IP prefix (`10.10.0.0/24`), zdrojový port (`any`), cieľový IP prefix (`192.168.1.0/24`), cieľový rozsah portov (`0:1023`). Posladná časť pravidla, telo, je obsiahnutá v zátvorkách. Táto časť pravidla umožňuje definíciu vzorov, ktoré sa v danom pakete majú vyhľadávať, napríklad TCP príznaky (`flags:S`). Taktiež sa v tele uvádza text hlásenia, v prípade paketu odpovedajúcemu tomuto pravidlu (`msg:...`).

Systé Snort môže byť konfigurovaný v troch hlavných módoch:

- Sniffer Mode – jedná sa o odposluch prevádzky real-time,
- Packet Logger mode – jednotlivé pakety sa zaznamenávajú a ukladajú na disk,
- Network Intrusion Detection System Mode – porovnáva pakety s konfigurovanými pravidlami a upozorňuje správcu siete, systému o útokoch.

Jedná sa o jednovláknový systém a teda nastáva limit možného spracovania paketov. To má za následok slabšiu výkonnosť systému a následné zahadzovanie prijatých paketov. Síce sa už aktívne vyvíja nová verzia systému Snort 3.0, ktorá podporuje multivláknové spracovávanie, zatiaľ je k dispozícií len ako beta verzia.

Suricata

Suricata [3] je open-source IDS systém, ktorý tak ako aj Snort, využíva detekciu na základe signatúr. Je vlastnený a podporovaný organizáciou OISF a vyvíjaný pod licenciou GPL verzie 2. Konfigurácia systému Suricata umožňuje jeho nasadenie v sieti ako NIDS, tak aj HIDS. Signatúry, resp. pravidlá, ktoré využíva pri detekcii útokov sú z veľkej časti podobné, až rovnaké so systémom Snort. Taktiež je možné využitie databázy signatúr EmergingThreats [18]. Avšak, hlavnou výhodou oproti konkurenčným riešeniam je podpora

multivláknového spracovania sieťovej prevádzky. Tým výkonnostne predbieha, aj napríklad už zmieňovaný systém Snort. V dôsledku multivláknového spracovania je na základe porovnania [9], zvýšená taktiež presnosť detekcie, než pri systéme Snort.

Zeek

Ako ďalší systém detekcie a prevencie prieniku, som vybral systém Zeek [22, 17], kedysi nazývaný Bro. Je jedným z ďalších NIDS, určený na monitorovanie prevádzky na sieti a taktiež podporuje off-line analýzu paketov. Je podporovaný nad operačnými UNIX systémami ako: Linux, FreeBSD, či MacOS. K špecifikácii detekcie slúži skriptovací jazyk, ktorý poskytuje väčšie možnosti popisu útoku než pravidlá, ktoré sú používané v systémoch Snort, či Suricata. Hlavnou komponentou pri spracovávaní sieťovej prevádzky je udalosťami riadený mechanizmus. Udalosti sú generované na základe analýzy hlavičiek paketov. Obsluhu jednotlivých udalostí je možné definovať pomocou skriptovacieho jazyka. Systém Zeek nepodporuje multivlánkové spracovávanie sieťovej prevádzky, avšak je možné záťaž systému distribuovať na viacero paralelne bežiacich inštancií a to na základe hašovania sieťových tokov.

OSSEC

Jedná sa o open-source multiplatformný systém typu HIDS. Je možné ho sprevádzkovať na platformách Linux, Solaris, AIX, Windows, Mac a rôznych ďalších. Jeho architektúra je nasledovná a tvorí ju, centrálny server, ktorý monitoruje a prijíma informácie od agentov, zo syslog správ, či databáz. Keďže správa všetkých pravidiel, hlavných konfigurácií je centralizovaná na serveri, uľahčuje sa práca administrátorovi siete, ktorý takto môže spravovať veľké množstvo agentov. Komunikácia medzi agentami a serverom prebieha na porte 1514 protokolu UDP [16].

Pre potreby mojej práce bolo nutné vybrať systém, ktorý poskytuje možnosť behu IDS systému ako NIDS. To z dôvodu jeho integrácie do zariadenia DDoS Protector, ktorého nasadenie v sieti zodpovedá práve typu NIDS. Tejto podmienke zodpovedajú jedine prvé tri zmienené systémy Snort, Suricata a Zeek. Z hľadiska jednoduchosti konfigurácie a administrácie systému je výhodnejšie použitie systému Snort, alebo Suricata. Systém Zeek síce disponuje väčšími možnosťami pri definícii popisu útoku, avšak pravidlá využívané v systémoch Snort a Suricata sú dostačujúce. Najdôležitejším faktorom pri výbere IDS/IPS systému pre potreby tejto bakalárskej práce bola však jeho výkonnosť. Vzhľadom na to, že multivláknové spracovanie natívne podporuje jedine systém Suricata a výkonnostne prevyšuje konkurenčné systémy, sa budem v mojej práci zaoberať práve týmto systémom. Mnou využívaný systém Suricata je vo verzii 5.0.2.. Nasledujúca sekcia sa venuje podrobnejšiemu popisu tohto systému.

2.5 Suricata

Táto sekcia je venovaná bližšiemu zoznámeniu sa so softwarom Suricata. Podkapitola v sebe zahŕňa okrem základných všeobecných informácií taktiež nevyhnutné informácie pre správne pochopenie implementačnej časti tejto bakalárskej práce. V prvom rade bude predstavený projekt Suricata ako taký a následne popis formátu pravidiel systému, aj s možnosťami jeho konfigurácie. Avšak, hlavnou časťou z hľadiska mojej práce je časť týkajúca sa spracováva-

nia jednotlivých paketov prostredníctvom reťazca tzv. modulov. Spôsob ako sú jednotlivé moduly uporiadané vytvára rôzne módy behu programu, tzv. runmodes. Aj o tých budú v poslednej časti tejto sekcie zmienené nevyhnutné informácie.

Suricata [3] je open-source bezpečnostný software pre ochranu počítačových sietí. Projekt Suricata je vlastnený a podporovaný organizáciou Open Information Security (OISF) a je vyvíjaný pod licenciou GPL verzie 2. Výkonnosť tohto systému vďaka podpore multivláknového behu je dôvod, kvôli ktorému som si vybral práve tento systém. Tento systém slúži na detekciu a prevenciu proti prieniku a teda sa jedná o IDS/IPS systém (sekcia 2.4). V závislosti od konfigurácie a miesta nasadenia softwaru v sieti, môže fungovať Suricata ako systém typu HIDS/HIPS (varianta „host“), alebo typu NIDS/NIPS (varianta „gateway“). Gateway varianta je častejšie používaná, keďže v tomto prípade analyzuje prevádzku celej siete, ktorá sa nachádza za zariadením na ktorom je systém Suricata spustený.

Signatúry Pre zistenie hrozieb na sieti sa v tomto systéme využíva detekcia na základe signatúr. Signatúry, tiež nazývané ako pravidlá, sú konfigurované administrátorom siete. Každá signatúra má určitý formát a definuje určitý typ prevádzky na sieti a taktiež aj akciu, na základe ktorej sa Suricata rozhoduje, čo s danou sieťovou prevádzkou učiniť. Formát signatúry je zobrazený nižšie, za ktorým nasleduje stručný popis jednotlivých častí pravidla.

```
drop udp $EXTERNAL_NET any -> 192.168.10.0/24 514 (msg:ËT DDOS Cisco 514
  UDP flood DDoS"; content:25 25 25 25 25 58 58 25 25 25 25 25|"; ...)
```

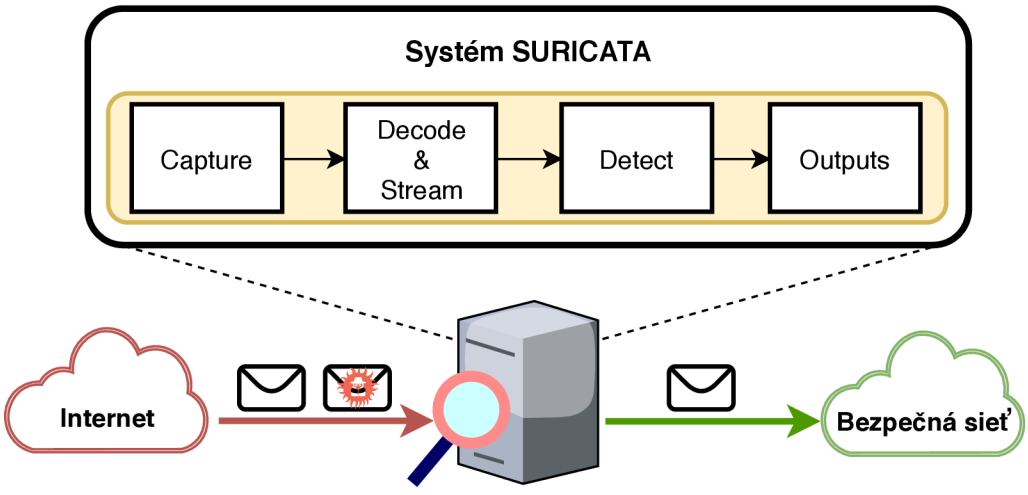
- Akcia definuje, čo sa stane, ak sa dané pravidlo zhoduje s aktuálnou prevádzkou. Suricata môže na základe tejto akcie prevádzku zahodiť (`drop`), povoliť (`pass`), či len upozorniť administrátora prostredníctvom záznamu v logovacom súbore (`alert`).
- Hlavička definuje sieťovú prevádzku, ktorej sa toto pravidlo týka. Pre definíciu danej prevádzky sa využívajú položky vrstvy L3 a L4 OSI modelu (obrázok 2.1). Medzi tieto položky patrí transportný protokol, zdrojový IP prefix s portom, cieľový IP prefix s portom a smer prevádzky. V tejto časti pravidla je možné využívať tzv. premenné (`$EXTERNAL_NET`), definované v konfiguračnom súbore, ktorému bude venovaný bližší popis v dalších častiach tejto sekcie.
- Nastavenie pravidla definuje špecifikácie pravidla. Táto posledná časť pravidla je zároveň aj tou najrozsiahlejšou z hľadiska možností nastavení. V tejto časti môže administrátor definovať vzory, ktoré sa majú vyhľadávať v pakete (položka `content:`). Okrem možností definície vzorov rôznych vrstiev sieťového OSI modelu je taktiež možné definovať napríklad hlášku (položka `msg:`), ktorá sa zaznamená do logovacích súborov v prípade zhody paketu s daným pravidlom.

Takýmto formátom môže administrátor definovať stovky, až tisíce pravidiel a rozdeľovať ich do súborov, tzv. rulesets. Avšak, manuálna definícia pravidiel pre detekciu najrôznejších útokov môže byť časovo veľmi náročná a preto sa často využívajú už existujúce rulesety. S využitím nástroja Suricata-Update [4] má administrátor možnosť sťiahnuť skupiny pravidiel (rulesets) pre rôzne typy útokov nad rôznymi protokolmi. Tieto rulesety sú uložené na Emerging Threats Rule serveroch. Jedná sa o rulesety, ktoré sú denne aktualizovávané a definované tak, aby umožnili detekciu najnovších najrôznejších sieťových útokov vrátane

DDoS. Spravovanie pravidiel je teda týmto spôsobom oveľa jednoduchšie a taktiež obranyschopnosť celého systému sa s pokročilými pravidlami výrazne zvyšuje. Avšak, analýza naprieč rulesetom s tisíckami pravidiel získaných z daných serverov nie je práve jednoduchá úloha pre zariadenie na ktorom Suricata pracuje. Aj z tohto dôvodu máme môžnosť konfigurácie toho, ktoré rulesety sa budú brať do úvahy a teda nad ktorými sa detekcia bude vykonávať. Táto konfigurácia a zároveň aj mnoho ďalších nastavení behu Suricaty sa konfiguruje v konfiguračnom súbore s formátom YAML. V nasledujúcej časti budú predstavené podstatné a najčastejšie používané možnosti konfigurácie systému Suricata.

Konfigurácia systému Suricata sa okrem parametrov pri spúšťaní definuje aj v konfiguračnom YAML súbore (štandardne `/etc/suricata/suricata.yaml`). Tento súbor poskytuje širokú paletu možností nastavenia systému presne podľa aktuálnych potrieb. Ja zmienim niekoľko tých, s ktorými som sa v mojej práci zaoberal a ktoré sú najčastejšie využívané. Medzi tie patrí bezpochyby nastavenie toho, ktoré rulesety sa budú používať pri detekcii, ako už bolo zmienené vyššie. Taktiež tu je možnosť zadefinovať premenné, ktoré sa môžu následne využívať pri definovaní pravidiel. Tak ako aj súbory pravidiel, konfiguračný súbor umožňuje nastavenie adresára s konkrétnym súborom do ktorého má systém zaznamenávať log správy generované na základe pravidiel, ktoré sa zhodujú s aktuálne spracovávanou sieťovou prevádzkou. Keďže Suricata podporuje aj multivláknové spracovávanie paketov, ani nastavenia s tým súvisiace nechýbajú v konfiguračnom súbore. Ďalšou možnosťou konfigurácie je aj nastavenie tzv. runmódu, ktorý predstavuje mód behu celého systému. Jednotlivým typom runmódov, ktoré sú k dispozícií bude venovaná nasledujúca podsekcia.

Runmódy Suricata pozostáva z troch hlavných komponentov a tými sú vlákna, moduly a fronty. To, akým spôsobom sú tieto tri komponenty usporiadané, definuje jednotlivé runmódy. Avšak, pred popisom troch základných runmódov je nutné zoznámenie sa s modulmi, cez ktoré sa jednotlivé pakety spracúvajú. V nižšie uvedenom diagrame 2.7 je zobrazená postupnosť modulov v tom poradí, v akom sa pakety spracúvajú.



Obr. 2.7: Diagram usporiadaných modulov spracúvajúcich sieťovú prevádzku.

Ako je vidieť z diagramu 2.7 vyššie, Suricata disponuje štyrmi modulmi. Prvý modul, modul `Capture` prijíma pakety zo siete. Po ich prijatí, sa pakety presúvajú do modulu `Decode&Stream`, kde sa všetky dekódujú po jednotlivých vrstvách OSI modelu a rozdeľujú

do sledovaných streamov, sieťových tokov. V prípade TCP spojenia sa taktiež rekonštruuje originálny tok, resp. stream a to skladaním viacerých paketov dokopy. Ďalej nasleduje výkonnostne najnáročnejší modul a to modul `Detect`. Najnáročnejší je z toho dôvodu, že práve v tomto module sa vykonáva prehľadávanie a porovnávanie naprieč všetkými signatúrami konfigurovanými v systéme, ktorých môžu byť tisíce. Práve preto, že Suricata podporuje multivláknovosť a hlavne náročné prehľadávanie a porovnávanie signatúr môže byť distribuované do niekoľkých vlákien, má Suricata veľkú výhodu oproti konkurenčným riešeniam. Posledným modulom v spracovaní každého jedného paketu je modul `Output`, ktorý má za úlohu spracovanie všetkých výstražných oznámení o podozrivej prevádzke. V prípade IPS režimu behu Suricaty, je taktiež nutné z tohto modulu odosielať legitímne pakety. Tie pakety, ktoré neboli zahodené z dôvodu úspešnej zhody s nejakou signáturou, ktorej akcia bola konfigurovaná ako `drop`.

Po oboznámení sa s teoretickými konceptami modulov a architektúre spracovania jednotlivých paketov je teda možné definovať rôzne runmódy. Tie sa rozlišujú v tom, nad koľkými vláknami takéto zreťazené spracovávanie paralelne beží. Ako je už uvedené na začiatku tejto časti, Suricata definuje tri základné runmódy: `single`, `autofp`, `workers`.

`Single`
: Runmód `single` je najmenej výkonný a to z dôvodu, že pre spracovanie všetkých paketov je vytvorené len jedno vlákno a teda nedochádza k paralelnému spracovaniu. Tento runmód je užitočný hlavne pri vývoji, avšak do produkčného nasadenia sa nehodí.

`AutoFp`
: Ďalším runmódom je runmód `autofp`, ktorý kladie dôraz na fakt, že najnáročnejšia operácia spracovania paketov je detekcia ich hrozby, a teda porovnávanie a prehľadávanie naprieč signatúrami. Z tohto dôvodu sa v tomto runmóde rozlišujú dve skupiny vlákien. Tou prvou skupinou sú vlákna, ktoré sú vyhradené len pre moduly `Capture`, `Decode&Stream` a `Output`. Tieto vlákna teda zabezpečujú príjem, dekódovanie a prípadné odosielanie paketov. Druhou skupinou sú vlákna vyhradené len pre modul `Detect`, ktoré zabezpečujú jedine porovnávanie naprieč signatúrami. Administrátor má možnosť určiť pomer počtu vytváraných vlákien daných dvoch skupín.

`Workers`
: Avšak, pre potreby vysokorýchlostných sietí je nutné sa spoliehať na multivláknovosť systému naprieč všetkými modulmi. Tretím runmódom je runmód `workers`, ktorý je zároveň tým najvýkonnejším. Tento runmód vytvára vlákna, ktoré spracúvajú pakety všetkými modulmi a teda od modulu pre prijatie paketu `Capture`, až po posledný modul `Outputs`. To je taktiež dôvod jeho vyššej výkonnosti nad runmódom `autofp`. Tým, že v každé vlákno disponuje všetkými modulmi, odpadá zložitá štruktúra predávania paketov medzi dvoma skupinami vlákien, ako je tomu pri `autofp` runmóde. Ovládač sieťovej karty je následne zodpovedný za rozloženie prevádzky do vlákien vytvorených týmto runmódom. V mojej práci budem pracovať práve s týmto runmódom.

Systém Suricata má podporu rôznych typov rozhraní k prijímaniu paketov zo siete, ako napríklad PCAP, AF_PACKET, NETMAP, atď.. Na základe týchto rozhraní sú následne pomenované aj interné Suricata módy, ktoré podporu týchto rozhraní vytvárajú. Medzi tieto rozhrania, resp. módy taktiež patrí aj NFQ [21]. Mód NFQ neprijíma pakety priamo zo sieťovej karty, ale pakety z Linuxového jadra, kernelu. To, ktoré pakety sa na NFQ

rozhranie dostanú a teda budú spracovávanie systémom Suricata, je možné ovplyvniť nastavením Linuxového firewallu. Nastavenie tohto firewallu je zabezpečované konfiguráciou pravidiel prostredníctvom pomocného programu `iptables`. Príklad pravidla, ktoré zabezpečí preposielanie sieťovej prevádzky do systému Suricata bežiacim pod módom NFQ je nasledovný:

```
sudo iptables -I FORWARD -j NFQUEUE
```

V poradí druhým parametrom `-j` sa v `iptables` pravidle nastavuje tzv. *target*, ktorý určuje to, čo učiniť s paketom, ak sa zhoduje s daným pravidlom. *Target* `NFQUEUE` deleguje spracovávanie paketu na iný software v užívateľskej vrstve systému. Keďže systém Suricata má podporu a može pracovať v móde NFQ, dokáže takto delegované pakety prijímať a spracovávať. To, ktoré pakety sa budú takýmto spôsobom delegovať na systém Suricata sa určuje prvým parametrom `-I`. Hodnotou `FORWARD` sa vyberá možnosť behu Suricaty vo variante „gateway“ (NIDS - obrázok 2.6) tým, že sa do systému Suricata preposiela všetka sieťová prevádzka, ktorá nie je určená priamo systému Suricata. Mód NFQ funguje v režime IPS, a teda taktiež odosiela legitímnu prevádzku, späť do už zabezpečenej siete. Z tohto dôvodu som sa v mojej práci zameral hlavne na štúdium práve tohto módu.

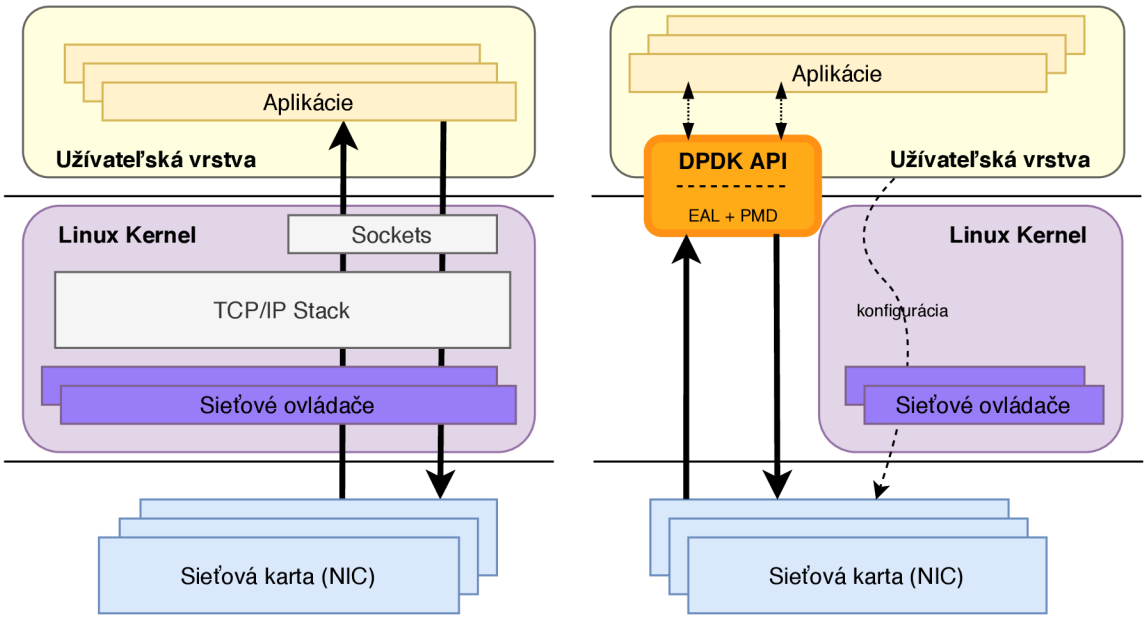
Režim NFQ je však výkonnostne nevhodný pre nasadenie v zariadení DDoS Protector (viz. sekcia 2.5) a to z dôvodu spracovávania všetkých paketov cez Linuxový kernel. Riešením je technológia DPDK (sekcia 2.6), ktorá svojou architektúrou obchádza spracovávanie sieťovej prevádzky cez kernel a využíva k tomu prakticky len užívateľskú vrstvu systému. Táto technológia je taktiež využívaná v novej verzii zariadenia DDoS Protector. Pridaním podpory rozhrania DPDK do systému Suricata, by sa jeho integrácia do systému DCPro stala výrazne jednoduchšou. Práve preto sa v mojej práci budem zaoberať aj pridaním tejto podpory do systému Suricata a nasledujúca sekcia je venovaná detailom práve o technológii DPDK.

2.6 Data Plane Development Kit

Data Plane Development Kit (DPDK) je open-source projekt, ktorého cieľom je poskytnúť framework pre vysokorýchlostné spracovávanie paketov v aplikáciách užívateľskej vrstvy systému. Návrh DPDK umožňuje prácu nad rôznymi architektúrami procesorov, ako napríklad Intel x86, ARM, či PowerPC. Okrem toho taktiež umožňuje prácu nad sieťovými kartami od rôznych výrobcov a to z neho vytvára všeobecné rozhranie, bez ohľadu na konkrétne využívanú sieťovú kartu. V súčasnosti je framework podporovaný hlavne nad operačným systémom Linux, ale aj FreeBSD. Projekt je vyvíjaný pod licenciou BSD [1]. V mojej práci som vužíval framework DPDK vo verzii 19.08.2.

Motiváciou pre vytvorenie takéhoto frameworku bolo relatívne pomalé spracovávanie paketov Linuxovým jadrom. Po prijatí paketu nasleduje pri štandardnom spracovávaní jadrom Linuxu niekoľko operácií, vrátane kopírovania prijatých dát, kým má k danému paketu prístup aplikácia v užívateľskej vrstve systému. To isté platí aj pri odosielaní aplikačných dát, kedy dochádza k vytváraniu socketu, vytvorenia paketov z dát, ich kopírovanie do bufferov ovládačá sieťovej karty a následné odosielanie sieťovou kartou. Vysokorýchlostné siete pri rýchlostiach rádovo desiatok gigabitov za sekundu dát nedokážu takýmto spôsobom spracovávania paketov zabezpečiť spoľahlivosť ich prenosu. Vývoj projektov, akým je aj DPDK rieši tento problém. Na rozdiel od štandardného spracovania paketov jadrom

systému, DPDK spracováva pakety v užívateľskej vrstve systému. Tým je umožnené užívateľským aplikáciám, využívajúcich DPDK framework, pristupovať k prijatým paketom oveľa rýchlejšie a taktiež aj ich rýchlejšie odosielanie. Táto architektúra je navrhnutá spôsobom, kedy sa jadro systému vužíva len pre beh ovládača sieťovej karty, ktorý je špecializovaný pre framework DPDK. Jeho úlohou je hlavne konfigurácia sieťovej karty, avšak pre samotný prenos paketov sa využívajú knižnice frameworku DPDK, ktorým bude venovaná samostatná časť v tejto sekcii. Porovnanie spracovávania paketov cez Linuxové jadro a prostredníctvom DPDK frameworku je zobrazené na nasledujúcom obrázku 2.8.



Obr. 2.8: Rozdiel v spracovávaní paketov frameworkom DPDK.

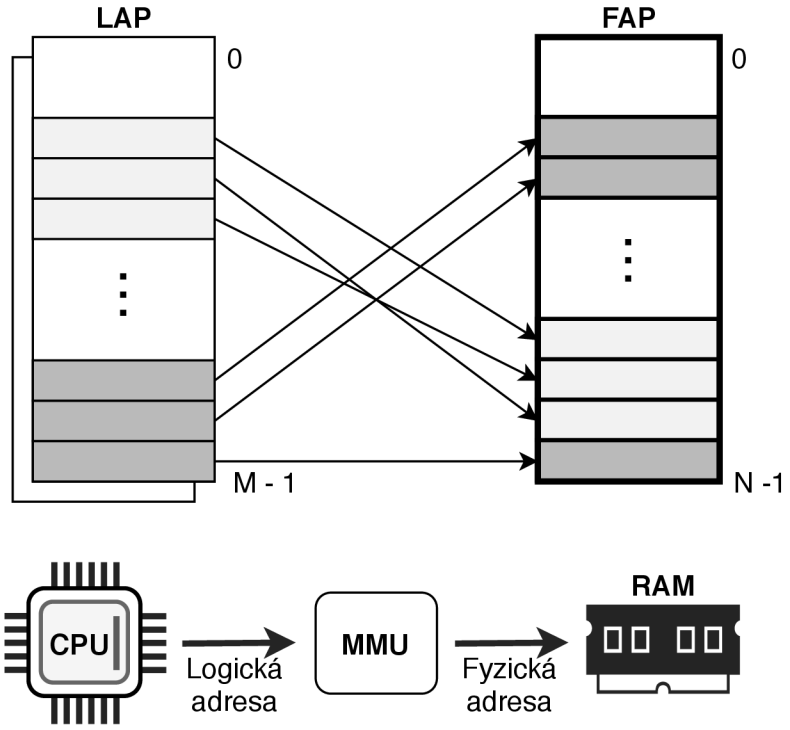
Ako je možné vidieť z obrázku 2.8, riešenie DPDK sa úspešne vyhýba vo veľkej miere práce s jadrom, čím sa výrazne zvyšuje výkonnosť spracovávania sieťovej prevádzky. Jediná komunikácia s kernelom je prostredníctvom sieťových ovládačov, ktoré sa však využívajú len na konfiguráciu sieťových kariet a nie priamo na prenos paketov. Prenos paketov riadi DPDK knižnica, ktorá cez PMD DPDK ovládače priamo komunikuje so sieťovou kartou.

To, akým spôsobom DPDK umožňuje takéto spracovanie bude popísané v nasledujúcej časti. Framework vytvára abstraktnú vrstvu pre špecifické prostredia cez vytvorenie tzv. Environment Abstraction Layer (EAL) [2]. Vrstva EAL zodpovedá za získanie prístupu k nízkoúrovňovým zdrojom ako hardware, či pamäťový priestor. Poskytuje generické rozhranie pre aplikácie a knižnice, čím odstraňuje nutnosť poznania špecifikácií aktuálneho prostredia. Tým pádom sú aplikácie vytvorené nad DPDK frameworkom ľahko prenositeľné nezávisle na aktuálnom prostredí. Hlavnými službami, ktoré poskytuje EAL vrstva sú:

- načítanie a spustenie DPDK, inicializácia,
- afinita jadier procesoru,
- rezervácia systémovej pamäte,
- obsluha prerušení.

Inicializácia DPDK začína ešte pred samotným spustením vykonávania programu, teda pred volaním funkcie `main` a to kontrolou podpory daného typu architektúry, určenej v konfiguračnom súbore. Za inicializáciu vrstvy EAL, ako aj za incializáciu jadier a ich spustenie zodpovedá funkcia `rte_eal_init()`, ktorá na túto úlohu využíva knižnicu `pthread`. Štandardne sa každému jadru procesora vytvára práve jedno vlákno. Inicializáciu prostredia DPDK je možné upraviť a prispôsobiť svojim potrebám prostredníctvom tzv. EAL parametrov, ktoré sa zadávajú pri spúšťaní aplikácie. Tieto parametre sú spoločné pre všetky platformy podporované frameworkom DPDK a po ich zadaní môžu nasledovať štandardné parametre vyvíjanej aplikácie, avšak nutne oddelené dvojicou znakov „`--`“.

Ďalšou službou, ktorú EAL vrstva poskytuje je správa a rezervácia pamäte. Alokáciu vykonáva pomocou `mmap()` v *hugetlbfs*. Hugetlbfs je súborový systém, ktorý umožňuje používanie veľkých stránok, tzv. *hugepages*. Pre lepšie pochopenie bude nasledujúci text venovaný oboznámaniu čitateľa so správou operačnej pamäte RAM. Obrázok 2.9 zobrazuje princípy virtualizácie pamäte.



Obr. 2.9: Princípy virtualizácie pamäte.

Moderné operačné systémy využívajú pre správu hlavnej operačnej pamäte RAM techniku virtualizácie pamäte. Virtualizáciou získavame k fyzickému adresnému priestoru (FAP) aj ten virtuálny, resp. logický (LAP). Logický adresný priestor sa rozdeľuje do tzv. stránok (angl. pages). Stránka je najmenšou jednotkou prideľovanou pamäti procesoru a jej veľkosť je štandardne 4KiB. Sprístupnenie fyzickej pamäte sa vykonáva tzv. mapovaním logických adries z LAP na fyzické. Toto mapovanie vykonáva *jednotka správy pamäte* (Memory Management Unit – MMU). Každý proces má svoj vlastný logický adresový priestor rozdelený na stránky. Pri vykonávaní procesu, je nutné pre každý prístup k jeho dátam získať fyzickú adresu miesta, kde sa v RAM fyzicky nachádzajú. V prípade, že proces využíva veľké množstvo pamäte, jej nadmerné rozdelenie na štandardné 4KiB stránky môže spôsobovať

spomalenie pri vyhľadávaní toho, kam je určitá stránka namapovaná. Využívaním spomínaných veľkých stránok (*hugepages*) sa toto spomalenie eliminuje, keďže veľké stránky alokujú násobne väčšie bloky pamäte a to od 2MiB, až po 1GiB a vyššie a tým pádom je ich počet výrazne zmenšuje. Pre prácu s pamäťou je možné využívať knižnicu `rte_malloc`, avšak pre spracovávanie paketov sa používajú iné knižnice, ktorým bude venovaný nasledujúci odsek.

V nasledujúcej časti budú popísané najdôležitejšie štruktúry, resp. knižnice, ktorých vysvetlenie je nevyhnutné pre následné pochopenie implementácie spracovania paketov prostredníctvom frameworku DPDK.

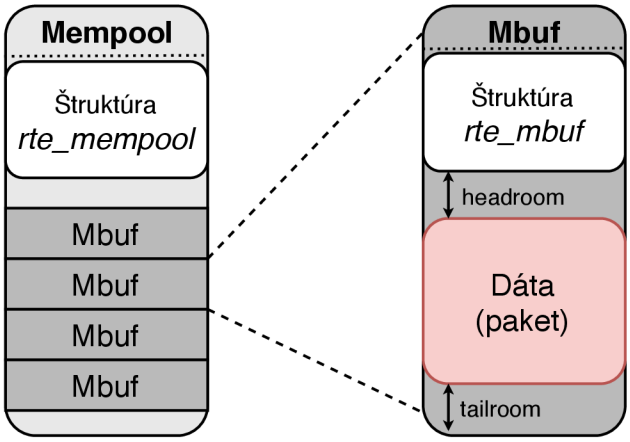
Ring
: Jedná sa o knižnicu DPDK *librte_ring*, ktorá umožňuje správu nad frontami. Jednotlivé kruhové fronty Ring sú fixnej veľkosti a podporujú optimalizované hromadné operácie pre pridávanie a odoberanie objektov (oprarácie `bulk` a `burst`). Typ týchto front je FIFO (angl. First In Frist Out).

Mempool
: Táto knižnica *librte_mempool* zabezpečuje správu štruktúry toho istého názvu, Mempool. Štruktúra Mempool je alokátor objektov fixnej veľkosti. V DPDK je identifikovaný podľa mena a využíva tzv. „mempool handler“ pre správu svojich voľných objektov. Tento „handler“ je založený na štruktúre Ring a teda využíva knižnicu spomínanú v predošlom odseku. Výkonnosť práce so štruktúrou Mempool je výrazne zvýšená zarovnávaním objektov v pamäti. Cieľom toho je, aby začiatok každého objektu začínal v odlišnom kanáli s pozíciou tzv. *rank*, a teda rozloženie objektov bude rovnomerné vzhľadom na pamäťové kanály. Výkonnostné vylepšenie zabezpečuje aj využitie vyrovnávacej pamäte. Zapnutím ladiaceho módu, konfiguráciou možnosti `CONFIG_RTE_LIBRTE_MEMPOOL_DEBUG` je možné získavať štatistiky o aktuálne uložených objektoch v danej štruktúre Mempool. Táto knižnica je taktiež využívaná knižnicou Mbuf, ktorej bude venovaný nasledujúci odsek.

Mbuf
: Knižnica *librte_mbuf* poskytuje rozhranie pre alokáciu a správu bufferov, ktoré sa zvyčajne v DPDK aplikáciách používajú na ukladanie správ, paketov (sekcia 2.1), avšak môžu obsahovať rôzne dáta. Jednotlivé buffery sú uložené v štruktúre Mempool a sú reprezentované štruktúrou `rte_mbuf`. Táto štruktúra je zobrazená v nasledujúcom obrázku 2.10, aj so štruktúrou Mempool.

  V obrázku 2.10 je naznačený vzťah štruktúr Mempool a Mbuf. Tak ako bolo zmienené v predošlom odstavci, Mempool obsahuje objekty fixnej veľkosti a vo väčšine DPDK aplikácii je to objekt štruktúry Mbuf. Štruktúra Mbuf pozostáva z metadát a oblasti fixnej veľkosti pre úschovu prijatého paketu. Túto oblasť ohraničujú dve voľné miesta pevnej veľkosti a to tzv. `headroom` a `tailroom`. Metadáta obsahujú rôzne kontrolné informácie, ako napríklad odkaz na štruktúru Mempool v ktorej je daný Mbuf uložený, typ správy, ofset, dĺžku paketu, či ukazateľ na začiatok dát, resp. paketu. Taktiež môže obsahovať ukazataľ na ďalšiu Mbuf štuktúru v prípade zreťazenia Mbufov. K zreťazeniu dochádza vtedy, ak je pre uschovanie príliš veľkého paketu nutné využiť niekoľko mbufov.

  Mbuf je možné rozdeliť na dva druhy a to priamy a nepriamy. Priamy mbuf je kompletne nezávislý, samostatný a ukazateľ na dáta (uložený paket) odkazuje na vlastné



Obr. 2.10: Štruktúrý Mempool a Mbuf.

dáta Mbufu. Naproti tomu nepriamy Mbuf sa správa rovnako ako priamy, avšak ukazateľ na dáta odkazuje na dáta iného Mbufu. Tákáto konfigurácia sa môže využiť, ak je potrebná duplikácia niektorých Mbufov.

Pre spracovávanie dát, je okrem vyššie spomínaných štruktúr potrebný popis princípov Ethernet portu a jeho špeciálneho ovládača, ktorým v prípade DPDK aplikácii musí Ethernet port disponovať. Jedná sa o tzv. *Poll Mode Driver - PMD* ovládač. Štandardné ovládače v Linuxovom jadre používajú pri príjímaní dát prerušenia, k oznámeniu novo prichádzajúcich paketov. Pri každom prerušení však nastáva bezodkladná obsluha tohto prerušenia a tým aj určité spomalenie systémového behu. Ovládače PMD využívajú techniku dotazovania – *pooling*. Tak ako je znázornené aj na obrázku 2.8, tieto ovládače bežia v užívateľskej vrstve systému a pristupujú k Rx a Tx deskriptorom daného portu priamo, bez prerušení. Táto technika dotazovania zabezpečí rýchlejšie prijatie, spracovania a dodanie paketov do užívateľskej aplikácie. PMD poskytuje taktiež API pre prácu s daným portom, na základe ktorého sú implementované ďalšie funkcie v DPDK API.

V tejto kapitole bol uvedený teoretický základ počítačových sietí, podrobnejší popis konkrétneho IDS/IPS systému Suricata, a taktiež technológie DPDK. Nasledujúca kapitola sa venuje konceptuálnemu návrhu integrácie systému Suricata so systémom DCPro, za pomoci technológie DPDK. Touto technológiou je však nutné systém Suricata rozšíriť a aj tomu sa bude venovať nasledujúca kapitola.

Kapitola 3

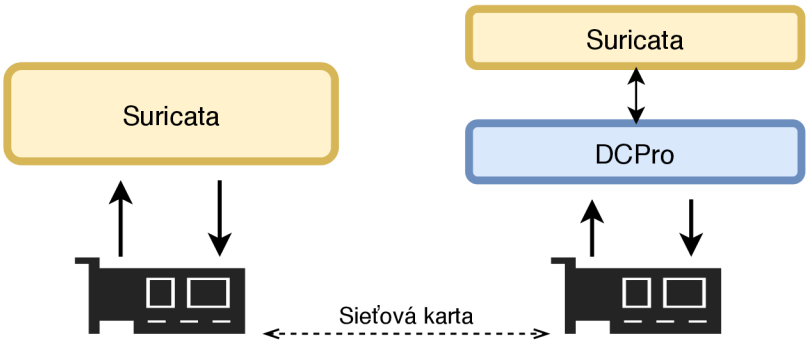
Konceptuálny návrh integrácie

Využitie IDS/IPS systému (sekcia 2.4) v projekte DDoS Protector – DCPro (sekcia 2.5) nachádza svoje opodstatnenie hlavne z dôvodu jeho pokročilých možností analýzy sieťovej prevádzky vo všetkých úrovniach OSI modelu (obrázok 2.1). Tým sa bezpečnostné schopnosti systému DCPro môžu výrazne rozšíriť. Predošlá kapitola 2 popisuje okrem iného aj IDS/IPS systém Suricata (sekcia 2.5), ktorý som vybral spomedzi konkurenčných riešení kvôli jeho výkonnostným výhodám. Taktiež sa v predošlej kapitole popisuje aj technológia DPDK (sekcia 2.6), ktorá zabezpečuje zvýšenie výkonnosti spracovávania sieťovej prevádzky. Keďže systém DCPro technológiu DPDK v svojej novej verzii už využíva, využil som ju aj ja a to rozšírením systému Suricata o jej podporu. Následne bola možná jednoduchšia integrácia systémov DCPro a Suricata prostredníctvom tejto technológie.

Ešte pred samotným popisom návrhu je nutné si zhrnúť ciele a požiadavky, ktorými by mala výsledná práca disponovať:

- podporou technológie DPDK v systéme Suricata,
- možnosťou zahodenia paketu na základe signatúr,
- multivláknovým behom systému Suricata,
- intuitívnou konfiguráciou systému,
- samostatným behom systému Suricata nad DPDK rozhraním,
- komunkáciou systémov Suricata a DCPro a ich spoločným behom.

Vzhľadom na posledné dve uvedené požiadavky som definoval dva návrhové režimy, na základe ktorých sa následne odvíjala aj implementácia. Jedná sa o režim samostatného behu systému Suricata a o režim integrovaného behu systémov Suricata a DCPro. Základné porovnanie tých dvoch režimov je zobrazené na nasledujúcom obrázku 3.1, za ktorým nasleduje základný popis oboch režimov.



Obr. 3.1: Porovnanie samostatného a integrovaného režimu.

Samostatný režim
Tento režim je zobrazený na ľavej strane obrázku 3.1. Jedná sa o režim, kedy systém Suricata beží ako jediná DPDK aplikácia a prijíma pakety priamo zo sieťovej karty. Po ich analýze odosiela späť na sieťovú kartu legitímne pakety.

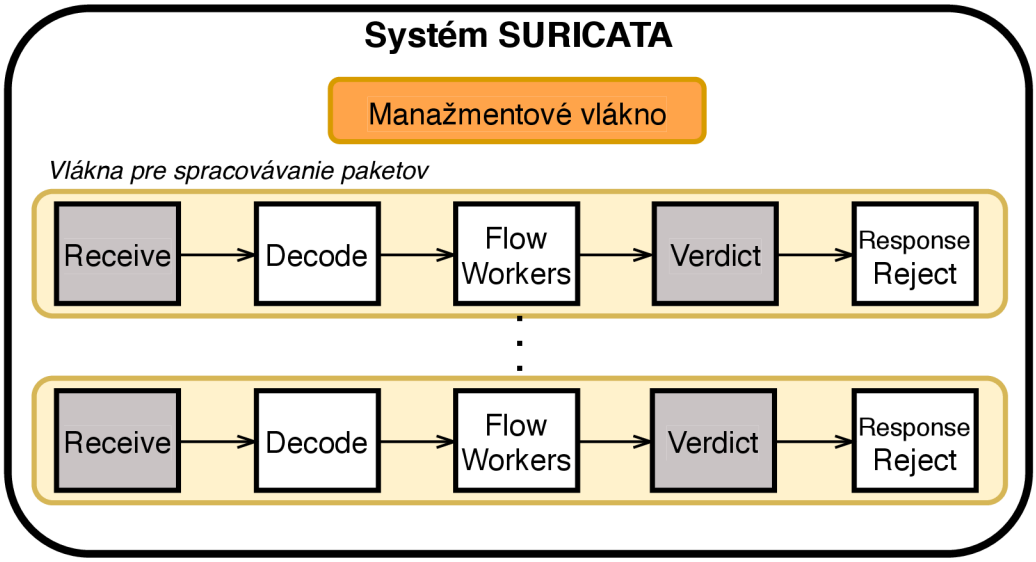
Integrovaný režim
Druhým režimom je režim kedy sú systémy Suricata a DCPro integrované prostredníctvom DPDK. Systém Suricata však na rozdiel od samostatného režimu beží paralelne popri behu systému DCPro. Príjem a aj odosielanie paketov zaisťuje systém DCPro, ktorý predáva systému Suricata len určité pakety na analýzu. Suricata po ich spracovaní odosiela legitímne pakety späť do systému DCPro, ktorý ich následne odosiela na sieťovú kartu.

Táto kapitola je rozdelená do troch hlavných sekcií. Toto rozdelenie súvisí aj s mojim postupom pri vytváraní návrhu a implementácii tejto práce. Prvá sekcia je venovaná popisu štruktúry systému Suricata. V ďalších dvoch sekciách je obsiahnutý podrobnejší popis spomínaných dvoch režimov.

3.1 Štruktúra systému Suricata

Sekcia 2.5 v predošlej kapitole okrem iného taktiež opisuje moduly systému Suricata pre spracovanie sieťovej prevádzky, avšak len z návrhového a teoretického hľadiska. Pre potreby vysvetlenia návrhu mojej práce a spôsobu pridania podpory DPDK do tohto softwaru je nutné zmieniť a vysvetliť jednotlivé moduly z hľadiska implementačného. Moduly, ktoré budú spomínané v nasledujúcej časti sú reálne moduly, ktoré je možné nájsť v zdrojových kódoch systému Suricata. Obrázok 3.2 znázorňuje okrem jednotlivých modulov, taktiež runmód `workers` (sekcia 2.5), ktorý som vyžíval v mojej práci.

Nedôveryhodná sieťová prevádzka smerujúca na zariadenie so systémom Suricata sa rovnomerne rozdeľuje do jednotlivých vlákien na jej spracovanie (Packet Processing Threads – PPT). Tak, ako je naznačené v diagrame 3.2, každé toto vlákno (bloky vyznačené žltou farbou) v runmóde `workers` disponuje celým reťazcom modulov pri spracovávaní paketov. Avšak, pri porovnávaní tohto reťazca s obrázkom uvedeným v predošlej kapitole (obrázok 2.7) je vidieť mierne odlišnosti. Pôvodný obrázok modulov vychádza z oficiálnej dokumentácie systému Suricata, zatiaľ čo novo vytvorený reťazec modulov (obrázok 3.2) vychádza priamo z implementácie tohto systému. Tieto odlišnosti sú hlavne v názvosloví,



Obr. 3.2: Runmód `workers` s implementovanými modulmi Suricaty.

keďže diagram vyššie 3.2 uvádza také moduly a s takými názvami, aké sú používané priamo v implementácii.

Pre prijatie prevádzky sa využíva modul `Receive`, ktorý následne predáva spracovávanie paketu do modulu `Decode`. V tomto module sa paket dekóduje a vyplní sa interná štruktúra Suricaty, ktorá reprezentuje daný paket. Po dekódovaní nasleduje modul `Flow Workers` v ktorom sa vytvára tok v prípade TCP (sekcia 2.1) komunikácie a určuje akcia, čo s daným paketom učiniť na základe konfigurovaných signatúr. Keďže mnou implementované rozšírenie bude využívať režim IPS, je nutné zabezpečiť aj odosielanie legitímnych paketov. Práve preto po detekcii v module `Flow Workers` nasleduje modul `Verdict`, ktorý odosiala, prípadne zahadzuje aktuálne spracovávaný paket na základe detekovanej akcie predošlým modulom. Posledným modulom v spracovávaní paketu je modul `Response Reject`, ktorý vykoná dodatočné akcie v prípade, že je akcia pre daný paket nastavená na `reject` (sekcia 2.5).

V mojej práci bola pre pridanie podpory DPDK do systému Suricata nutná kompletná úprava modulov `Receive` a `Verdict`, ktoré sú v obrázku 3.2 znázornené sivou farbou. Čo sa modulov týka, taktiež bol upravovaný z časti aj modul `Decode`, avšak tieto úpravy sa nelíšia v závislosti od daného režimu. Samostatný aj integrovaný režim DPDK verzie systému Suricata má spoločné aj to, že sa jedná o DPDK aplikáciu a teda je nutná inicializácia EAL vrstvy, príprava systému, portov a ďalších štruktúr pred spustením vlákien na spracovávanie paketov (PPT). Všetky tieto prípravy a inicializácie sa vykonávajú v manažmentovom vlákne (oranžový blok v obrázku 3.2), ešte pred vytvorením PPT vlákien. Nasledujúce dve sekcie sa venujú popisu samostatného a integrovaného režimu DPDK verzie systému Suricata.

3.2 Samostatný režim

Ako už bolo uvedené v úvode tejto kapitoly, výsledný systém by mal disponovať možnosťou výberu medzi samostatným behom systému Suricaty a behom, kedy je závislý na systéme DCPro. To akým spôsobom sa tento výber vykonáva a ako aj ďalšie implementačné detaily budú opísané v nasledujúcej kapitole 4. Táto sekcia sa venuje popisu prvej varianty a teda pridaniu podpory DPDK do systému Suricaty, kedy sa pakety prijímajú a odosielajú priamo cez fyzický Ethernet port a systém funguje samostatne. Pred prijímaním a odosielaním paketov cez Ethernet port je nutná jeho inicializácia.

Príjem a analýza dát Následne je možné vykonávať príjem paketov modulom `Receive`. Návrh tohto modulu pri samostatnom režime spočíva v prijímaní paketov priamo z fyzického portu prostredníctvom knižnice Ethdev. Prijaté pakety sú uložené v jednotlivých štruktúrach Mbuf, pre ktoré je v pamäti alokované miesto štruktúrou Mempool (obrázok 2.10). Keďže úprava nasledujúcich dvoch modulov `Decode` a `Flow Workers` nieje pre zaintegrovanie podpory DPDK nutná, je potrebné aby sa jednotlivé štruktúry Mbuf, obsahujúce prijaté pakety, preformátovali na formát paketu, ktorý je systémom Suricata štandardne využívaný. Po dekódovaní a porovnaní daného paketu naprieč signatúrami je možné daný paket odoslať, ak je legitímny.

Odosielanie dát Odoslanie paketu je úlohou modulu `Verdict`, ako aj aktualizácia štatistík o legitímnych, či neprepustených paketov. Keďže formát paketu, ktorý bol predošlými dvoma modulmi využívaný, je nevyhovujúci pre odosielanie cez DPDK, bolo nutné navrhnúť riešenie komunikácie medzi modulmi `Receive` a `Verdict`. Modul `Verdict` potreboval prístup k miestu uloženia daného paketu modulom `Receive` a to vo formáte Mbuf. To sa zabezpečilo pridaním položky do štandardnej štruktúry paketu využívanej naprieč modulmi a tým pádom modul `Verdict` dokázal pristupovať k miestu uloženia práve spracovávaného paketu, resp. Mbufu. Keďže sa jedná o multivláknové spracovávanie, každé vlákno má vytvorenú samostatnú štruktúru spoločných informácií. Okrem štatistík počtu prijatých a zahodených paketov sa v nej pre jednotlivé vlákna nachádza aj identifikačné číslo vlákna, resp. číslo fronty s ktorou dané vlákno na porte pracuje. Toto číslo sa využíva ako pri prijímaní paketov modulom `Receive`, tak aj pri odosielaní modulom `Verdict`.

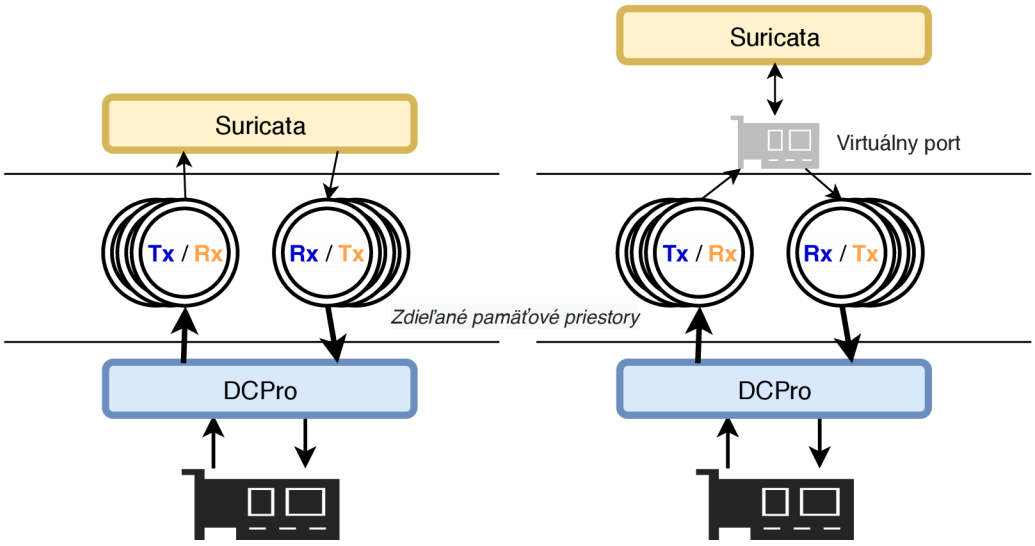
Využitie samostatného behu Suricaty nad technológiou DPDK bolo prvým krokom pri započatí praktickej časti mojej práce. Avšak, bolo nutné pokračovať integráciou celého systému Suricata so systémom DCPro. Aj keď sa návrh módu DPDK v systéme Suricata z tohto dôvodu mierne pozmenil, implementovaná varianta samostatného behu sa ponechala. Nasledujúca sekcia sa bude venovať rozšíreniu možností behu Suricaty nad technológiou DPDK, kedy Suricata bude potrebovať pre svoj úspešný beh druhú, a to primárnu DPDK aplikáciu.

3.3 Integrovaný režim

Hlavným cieľom tejto bakalárskej práce je integrácia systému Suricata do projektu DCPro. Ako už bolo zmienené v úvode tejto kapitoly, integrácia tohto IDS/IPS systému rozšíri schopnosti systému DCPro o detekciu a filtráciu ďalších útokov všetkých vrstiev OSI modelu. Vzhľadom na vysokorýchlostné počítačové siete, kde sa predpokladá nasadenie zariadenia DDoS Protector, je nutné využiť všetky možnosti akcelerácie celého procesu spracovania sieťovej prevádzky. Z tohto dôvodu sa implementácia softwaru DCPro menila a to tak, aby bolo možné využitie technológie DPDK (sekcia 2.6). Predošlá sekcia sa venovala pridaniu podpory technológie DPDK do systému Suricata. Avšak pre integráciu tohto systému so systémom DCPro bolo nutné navrhnúť dodatočné zmeny, ako už v samotnom systéme Suricata, tak aj v systéme DCPro. Táto sekcia sa venuje práve tejto časti návrhu.

Multi-procesová komunikácia Oba spomínané systémy sú samostatne bežiace procesy. Vzhľadom na fakt, že systém DCPro využíva technológiu DPDK a do systému Suricata bola táto podpora už integrovaná, bolo výhodné uvažovať nad možnou komunikáciou medzi týmito procesmi prostredníctvom nej. Takáto podpora medzi-procesovej komunikácie sa v technológii DPDK nachádza a v integrácií týchto systémov bola plne využitá. Pre túto podporu bola upravená vrstva EAL (sekcia 2.6) takým spôsobom, že umožňuje vytváranie dvoch typov procesov, ktorých hlavný rozdiel spočíva v ich oprávneniach v prístupe k `hugepage` pamäti (sekcia 2.6). DPDK aplikácia teda môže byť spustená pod procesom *prímárnym*, alebo *sekundárnym*. Kým primárny proces má plné oprávnenia pre prácu s pamäťou, sekundárny proces môže pristupovať iba k primárnym procesom inicializovaným oblastiam pamäte. Samotnú inicializáciu pamäte vykonávať nesmie. Z toho vyplýva, že sekundárny proces nedokáže plnohodnotne fungovať bez paralelného behu primárneho procesu. Avšak, beh multi-procesových DPDK aplikácií má aj svoje obmedzenia. Medzi tie napríklad patrí nutnosť zabezpečiť to, aby primárny process a sekundárny proces nepoužívali rovnaké logické jadrá. Nedodržanie tejto podmienky môže vytvárať problémy s pamäťovými priestormi a mnohé ďalšie. Z tohto dôvodu bol spôsob vytvárania PPT vlákien v systéme Suricata kompletne upravený pre beh DPDK módov a to s využitím DPDK API pre správu vlákien a jadier.

Návrh integrácie systému Suricata so systémom DCPro spočíva v úprave oboch systémov takým spôsobom, aby sa systém DCPro javil ako proces primárny a systém Suricata ako proces sekundárny. To sa zabezpečuje dodatočným DPDK EAL parametrom pri spúšťaní oboch systémov. Komunikáciu takýchto dvoch na sebe závislých systémov je nutné zabezpečiť zdieľaním pamäťových priestorov. Návrh integrovaného režimu pozostáva z dvoch variant, ktoré sa líšia len mierne. V obrázku 3.3 je naznačený návrh a porovnanie oboch variant integrovaného režimu.



Obr. 3.3: Návrh dvoch variant integrovaného režimu DPDK verzie systému Suricata.

Ako je možné vidieť z obrázku 3.3 komunikácia medzi systémami Suricata a DCPro spočíva v prístupe k zdieľaným pamäťovým priestorom. Keďže systém DCPro je spúšťaný ako prímárny proces, je nutnosť tieto pamäťové priestory inicializovať práve z tohto systému. Inicializácia pamäťového priestoru pozostáva z alokácie Mempoolu a kruhových front, resp. štruktúr typu Ring (sekcia 2.6). Na druhej strane, systém Suricata k takto alokovaným priestorom pristupuje a to tak, že pri jeho spustení sa dotazuje na existenciu konkrétneho Mempoolu, alebo na existenciu Ringov. Dotazovanie prebieha na základe mena daných štruktúr. Obrázok 3.3 zobrazuje dve varianty integrovaného režimu DPDK verzie systému Suricata. Tieto varianty sa líšia v spôsobe prístupu k zieľadným pamäťovým priestorom. Kým v prvej variante sa k nim pristupuje priamo, v druhej variante sa k nim pristupuje cez virtuálny port.

Princíp fungovania tohto návrhu je v tom, že jedine systém DCPro priamo komunikuje so sieťou ku ktorej je zariadenie DDoS Protector pripojené. Na základe internej konfigurácie systému DCPro, prípade procesov v spracovávaní sieťovej prevádzky sa určí ktoré pakety by sa mali spracovať systémom Suricata a ktoré priamo systémom DCPro. Pakety určené systému Suricata sa následne presmerujú do zdieľanej pamäte a to konkrétne zápisom do Tx kruhových front (štruktúra `Ring`). Keďže pred spracovávaním paketov je v manažmentovom vlákne Suricaty vykonané mapovanie interných ukazateľov na systémom DCPro vytvorené pamäťové miesta, je možné pristupovať k Tx ringom aj z tohto systému. Avšak v tomto prípade sa pre systém Suricata javia tieto ringy ako Rx, čiže ringy určené na príjem dát. Takýmto spôsobom sa vybrané pakety pre Suricatu dostanú až k ich spracovávaniu v tomto systéme. Tak ako bolo uvedené v predošlej sekcii, po spracovaní jednotlivých paketov sa následne odosielajú tie, ktoré boli na základe signatúr vyhodnotené ako validné. Odosielanie paketov spočíva v ich zápise do Tx ringov (z pohľadu systému Suricata). K takto zapísaným paketom je následne možné pristupovať zo systému DCPro, ktorému sa dané ringy javia ako Rx, čiže ringy určené na príjem paketov. Odosielanie systémom Suricata spracovaných paketov naspať do počítačovej siete zabezpečuje až samotný systém DCPro.

Multivláknovosť Multivláknové spracovávanie bolo taktiež jednou z požiadaviek funkcionality celkového systému, ako už bolo spomenuté v úvode tejto kapitoly. Zo strany DCPro

systému je multivláknové spracovávanie zabezpečené mechanizmom RSS (Receive Side Scaling). Tento mechanizmus spočíva v počítaní hodnoty s hašovacou funkciou nad každým paketom, čím sa následne získava index jadra/vlákna, ktorým sa má daný paket spracovávať [11]. Ku každému vláknu pripadá jedna dvojica kruhových front Rx/Tx inicializovaná primárnym procesom DCPro. Z obrázku 3.3 je vidieť kruhové fronty, resp. ringy vo viacerých pároch. To predstavuje jednotlivé dvojice front pre jednotlivé vlákna spracovávajúce pakety. Multivláknovosť cieľového systému je aj dôvod výberu `workers` runmódu v systéme Suricata. Úlohou správcu systému je, aby spustenie systémov DCPro a Suricata prebehlo nad rovnakým počtom uvedených logikých jadier, resp. vlákien a aby sa ani jedno vlákno nevyužívalo oboma systémami naraz. Ďalšou úlohou správcu systému je taktiež výber tzv. DPDK módu behu Suricaty.

Predošlá sekcia 3.2 sa venovala samostatnému režimu, pri ktorom funguje systém Suricata samostatne a vykonáva svoj beh ako primárny proces. V tomto režime Suricata pracuje priamo s fyzickým Ethernet portom, tak ako je aj naznačené v ľavej časti obrázku 3.1. Ďalším návrhovým režimom je režim integrovaného behu. Tento režim však má dve varianty, ktoré sa líšia v podstate len implementačne. Obe varianty tohto režimu sú naznačené v obrázku 3.3. Ako už bolo rozpísané v odsekoch tejto kapitoly vyššie, v tomto režime sa systém Suricata spúšťa ako sekundárny proces a pre príjem a odosielanie paketov využíva priamo kruhové fornty, resp. štruktúry typu Ring. Priame využívanie tých front je v prvej variante, avšak v druhej variante sa využívajú nepriamo, a to cez virtuálny port. Podrobnejší popis implementačných odlišností týchto dvoch variant integrovaného režimu, ako aj samostatného režimu bude obsiahnutý v nasledovnej kapitole 4.

Kapitola 4

Popis implementácie

Táto kapitola je venovaná popisu implementácie rozšírenia systému Suricata o technológiu DPDK a taktiež úpravám systému DCPro, pre následnú integráciu týchto dvoch systémov. Jednotlivé sekcie tejto kapitoly sa odvíjajú od predošlej kapitoly návrhu (kapitola 3), na základe ktorého bol celý systém implementovaný. V predošlej kapitole sa v posledných dvoch sekciách opisujú dva DPDK režimy, v ktorých môže systém Suricata fungovať a to režim samostatný 3.2 a integrovaný 3.3. Jedná sa o návrhové režimy na základe ktorých, boli v implementácii vytvorené tri DPDK módy. Prvým módom je mód `ethdev`, ktorý implementuje samostatný režim. Následne boli vytvorené ďalšie dva módy `ethdev-ring` a `native-ring`, ktoré implementuju dve varianty integrovaného režimu (obrázok 3.3). Postupy implementácie týchto troch módov budú rozpísané v posledných dvoch sekciách tejto kapitoly 4.2 a 4.3. Prvou sekciou 4.1 tejto kapitoly však bude popis štruktúry modulov systému Suricata a komunikácie medzi nimi.

Avšak, ešte pred spomínanými sekciami sa nasledujúci text venuje implementačným krokom, ktoré sú spoločné pre všetky implementované módy. Tak, ako pre každú DPDK aplikáciu, je nutné zabezpečiť inicializáciu DPDK EAL vrstvy (sekcia 2.6) a to práve funkciou `rte_eal_init()`. Parametrami tejto funkcie sú vstupné parametre programu Suricata. Funkcia spracúva všetky parametre programu, až po oddeľovač „`--`“, ktorý oddeľuje parametre pre DPDK EAL vrstvu od štandardných parametrov programu. Konkrétne EAL parametre využívané v mojej práci sú popísané v jednotlivých sekciách tejto kapitoly.

Keďže mnou upravený systém Suricata disponuje troma DPDK módmi, je nutné zabezpečiť možnosť konfigurácie, ktorý mód sa má vykonávať. Táto konfigurácia je zabezpečená prostredníctvom YAML konfiguračného súboru, v ktorom som vytvoril sekciu „`dpdk`“. Jedná sa o konfiguračný súbor, ktorý Suricata štandardne využíva pre svoj beh a o ktorom som sa už zmieňoval v sekcii 2.5. Jednou z konfiguračných možností mnou pridanej sekcie „`dpdk`“, je práve možnosť výberu DPDK módu. Pre získavanie hodnôt z YAML konfiguračného súboru som využil knižnicu `Conf` implementovanú priamo v systéme Suricata. To, čo majú ešte všetky tri DPDK módy spoločné, je štruktúra jednotlivých modulov Suricaty, ich vzájomná komunikácia a zabezpečenie multivláknovosti. Práve tomu bude venovaná nasledujúca sekcia.

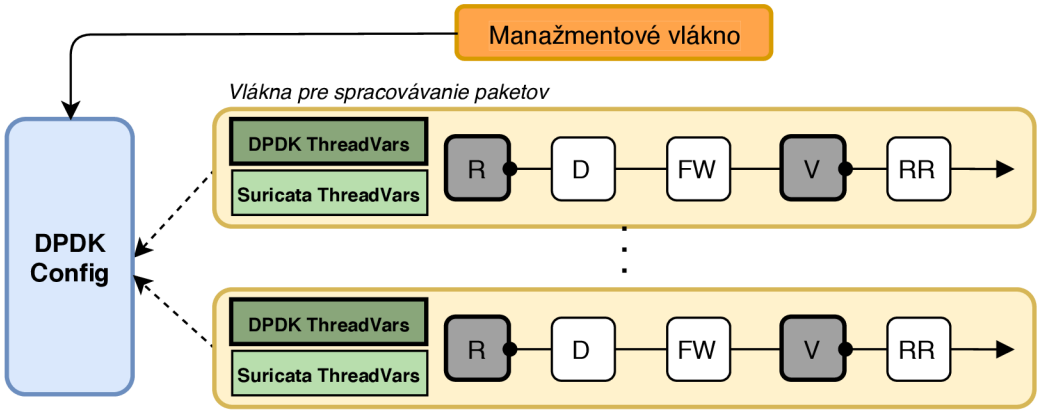
4.1 Štruktúra modulov systému Suricata

Všetky tri spomínané DPDK módy sú implementované nad runmódom `workers` (sekcia 2.5). Vzhľadom na to, že tento runmód podporuje multivláknové spracovanie sieťo-

vej prevádzky, zabezpečenie nekonfliktného behu jednotlivých vlákien je nevyhnutné. Táto sekcia popisuje implementačné detaily vytvárania vlákien na sebe nezávislých, štruktúry modulov a komunikácie medzi nimi.

Pre integrovanie podpory ľubovolného módu v systéme Suricata (AF-PACKET, DPDK, NFQ (odsek 2.5), ...), je nevyhnutná registrácia tohto módu v určitom type.
Mnou vyvíjaný mód DPDK je typu `workers` a tak ako aj ostatné, je registrovaný funkciou `RunModeRegisterNewRunMode()`. Okrem iného, sa ako parameter tejto funkcie zadáva ukazateľ na inicializačnú funkciu nového módu. V rámci tejto inicializačnej funkcie som využil volanie funkcie `RunModeSetIPSWorker()` pre vytvorenie jednotlivých vlákien na spracovanie paketov. Ako už názov tejto funkcie naznačuje, vytvorené vlákna budú pre režim IPS a pre mód typu `workers`. Z toho vyplýva, že každé jedno vlákno disponuje všetkými modulmi systému Suricata a keďže sa jedná o režim IPS, tak aj modulom `Verdict`. To, koľko vlákien sa vytvorí, určuje v prípade mnou implementovaného DPDK módu EAL parameter „`--lcores`“. V prípade, že tento parameter nieje zadaný, vytvoria sa vlákna v počte fyzických jadier procesoru. Pre vytváranie vlákien sa v štandardnej implementácii systému Suricata využíva knižnica POSIX thread (`pthread`). Avšak, pre zabezpečenie nekonfliktného vytvárania vlákien nad tými správnymi jadrami procesoru bolo nutné prerobiť systém vytvárania vlákien prostredníctvom DPDK knižnice `rte_lcore`.

Metadáta vlákien. Každé takto vytvorené vlákno disponuje všetkými modulmi a keďže jednotlivé moduly sú na sebe závislé, je potrebné zabezpečiť komunikáciu medzi nimi. Táto komunikácia je naznačená v nasledujúcom obrázku 4.1.



Obr. 4.1: Schéma štruktúry modulov Suricaty a ich dát.

Bloky vyznačené žltou farbou v obrázku 4.1 reprezentujú jednotlivé vlákna pre spracovávanie paketov, tak ako aj pri obrázku v návrhovej kapitole (obrázok 3.2). Okrem reťazca všetkých modulov, označených ich iniciálami sa ku každému vlánku alokujú dve štruktúry, štruktúra `DpdkThreadVars` a štuktúra `ThreadVars`. Druhá zmienená štruktúra `ThreadVars` je interná štruktúra pôvodnej implementácie systému Suricata a obsahuje položky nutné pre beh daného vlákna, nezávisle na zvolenom móde. Táto štruktúra obsahuje napríklad názov vlákna, referenciu samotného vlákna v položke typu `pthread_t`, zámky, atď..

Avšak, z hľadiska módu DPDK, je nutné zaviesť druhú dátovú štruktúru a to `DpdkThreadVars`. V dôsledku toho nie je nutné zasahovať do pôvodnej štruktúry `ThreadVards` a novo implementovaná štruktúra sa využíva len v prípade behu DPDK

módu. Táto štruktúra obsahuje metadáta DPDK módu, ktoré využívajú moduly `Receive` a `Verdict`. Medzi položky tejto štruktúry patrí napríklad:

- referencia na hlavnú štruktúra daného vlákna – `ThreadVars`,
- číslo Rx/Tx fronty pre dané vlákno,
- počítadlá prijatých, zahodených paketov.

Okrem spomínaných štruktúr alokovaných pre každé jedno vlákno, bola implementovaná aj jedna štruktúra spoločná pre všetky vlákna. Touto štruktúrou je `DpdkConfig`, v obrázku 4.1 vyznačená blokom modrej farby. Táto štruktúra obsahuje položky ku ktorým sa pristupuje v závislosti od zvoleného DPDK módu. Jednotlivé položky predstavujú napríklad:

- zvolený DPDK mód (`ethdev`, `ethdev-ring` alebo `native-ring`),
- číslo a názov portu,
- odkazy na Rx/Tx Ring štruktúry,
- celkový počet štruktúr Ring Rx/Tx.

Inicializácia položiek tejto štruktúry je vykonávaná manažmentovým vláknom. To, ktoré položky a akým spôsobom sa inicializujú bude popísané v nasledujúcich dvoch sekciách, ktoré popisujú implementáciu všetkých troch DPDK módov. Ešte pred tým však bude popísaná spoločná štruktúra všetkých typov modulov systému Suricata a to v nasledujúcej časti tejto sekcie.

Štruktúra modulu. Každý modul pozostáva zo štuktúry niekoľkých ukazovateľov na funkcie. Každá z daných funkcií zabezpečuje výkon jedného z uvedených cieľov:

- **Inicializácia modulu**. Inicializácia modulu spočíva v alokácií a inicializácií metadát, s ktorým bude následne daný modul pracovať. V prípade mnou implementovaného módu DPDK sú metadáta modulov `Receive` a `Verdict` spoločné a reprezentované štruktúrou `DpdkTheadVars`.
- **Výkon jadra modulu**. Jedná sa o funkciu, ktorej implementácia zabezpečuje hlavný beh daného modulu. Táto funkcia sa cyklicky vykonáva až do ukončenia behu systému Suricata.
- **Výpis štatistík**. Po požiadavke ukončiť beh systému Suricata, sa postupne zastavuje beh jednotlivých modulov a v prípade modulov `Receive` a `Verdict` dochádza k výpisu štatistík o spracovaných paketoch.
- **Deinicializácia modulu**. Beh jednotlivých modulov sa definitívne končí po vykonaní funkcie, ktorej úlohou je deinicializácia daného modulu, čím sa dealokujú pamäťové miesta využívané modulom, zastaví sa beh Ethernet portu, atď..

Všetky mnou implementované moduly systému Suricata (sekcia 3.2) sú implementované v zdrojovom súbore `/suricata/src/source-dpdk.c`.

Zameranie tejto sekcie bolo hlavne v oboznámení čitateľa s implementačnými detailami zabezpečenia multivláknovosti v systéme Suricata, jeho štruktúre modulov a komunikácie

medzi nimi prostredníctvom metadát, resp. dátových štruktúr. Nasledujúce dve sekcie sú zamerané na popis implementácie troch DPDK módov. Obe nasledujúce sekcie majú podobnú štruktúru. V prvej časti sekcie sa opisujú základné informácie o danom móde za ktorými nasleduje časť popisujúca prípravu daného módu na príjem a odosielanie dát, resp. paketov. Práve príjem a odosielanie dát bude predmetom posledných častí oboch sekcii. Nasledujúca sekcia popisuje implementačné detaily módu `ethdev`.

4.2 Mód `ethdev`

Mód `ethdev` implementuje samostatný režim (sekcia 3.2). Jedná sa o predvolený DPDK mód behu systému Suricata. Mód, kedy systém Suricata funguje samostatne a jeho beh je nutný v primárnom procese. Primárny proces je implicitný pri spustení DPDK aplikácie, avšak môže byť aj explicitne nastavený EAL parametrom `--proc-type=primary`. Ďalším parametrom nutným pre beh tohto módu je parameter `-w`, ktorého hodnotou sa nastavuje identifikácia portu s DPDK kompatibilným ovládačom. Jedná sa o fyzický port, ktorý bude systém využívať na príjem a odosielanie dát. Ešte pred tým je však nutné tento port nastaviť a inicializovať. Implementácia inicializácie takéhoto portu je popísaná v nasledujúcom odseku.

Inicializácia portu. Inicializácia portu spočíva v prvom rade vo vytvorení pamäťového priestoru pre ukladanie Mbuf štruktúr, resp. paketov. Tento pamäťový priestor, resp. Mempool sa vytvára funkciou `rte_pktmbuf_pool_create()`. Následne sa vytvorený Mempool využije pri nastavovaní Rx Ringov, resp. kruhových front daného portu. Jednotlivé funkcie využívané pri implementácii inicializácie a nastavovania daného portu sú popísané nižšie. Tieto funkcie sú použité z DPDK knižnice `rte_ethdev` a teda sa jedná o Ethernet API.

- `rte_eth_dev_configure()` Funkcia, ktorá zabezpečuje konfiguráciu Ethernet portu na základe jej predanej konfiguračnej štruktúry `rte_eth_conf`.
- `rte_eth_dev_adjust_nb_rx_tx_desc()` Nastavuje počet Rx a Tx deskriptorov daného portu.
- `rte_eth_rx_queue_setup()` / `rte_eth_tx_queue_setup()` Tieto fukcie zabezpečujú alokáciu a inicializáciu kruhobých front Rx a Tx. V prípade Rx smeru sa využíva aj parameter, ktorý odkazuje na už vytvorený Mempool, v ktorom sa budú nachádzať prijaté pakety, resp. štruktúry Mbuf.

Následne po spustení portu je ešte nutné zabezpečit jednu konfiguráciu. Cieľový systém bude slúžit pre analýzu sieťového toku, ktorý nebude adresovaný priamo naň, ale na ľubovolné zariadenie na danej sieti. Jedná sa teda o systém typu NIDS (obrázok 2.6). Vzhľadom na to, je nutné docieliť príjem aj tých paketov, ktoré nie sú adresované priamo tomuto systému, resp. portu. To sa zabezpečí funkciou `rte_eth_promiscuous_enable()`, ktorá nastaví port do tzv. promiskuitného módu. Tento mód portu docieli príjem aj tých paketov, ktoré nie sú adresované systému Suricata tým, že ignoruje destinačnú MAC adresu Ethernet rámca 2.1 a tým pádom prijíma úplne všetky pakety.

Po úspešnej inicializácii Ethernet portu sa pridá referencia naň do jednej z položiek v štruktúre `DpdkConfig`. Tým pádom sa umožní prístup k takto nastavenému portu zo všetkých vlákien. Následne je ním už možné prijímať a odosielať pakety a práve tomu sa bude venovať nasledujúca časť textu.

Príjem dát Príjem dát je zabezpečený modulom `Receive`. Keďže sa jedná o príjem dát priamo z portu, využíva sa k tomu funkcia z DPDK Ethernet API a to `rte_eth_rx_burst()`. Argumentami funkcie sú okrem identifikačného čísla portu, aj identifikačné číslo fronty daného portu s poľom ukazateľov na štruktúry Mbuf v ktorých sú prijaté pakety uložené. Identifikačné číslo danej fronty je uložené v metadátach konkrétneho vlákna, v štruktúre `DpdkTheadVars`. Prístup k metadátam je nutný aj pri aktualizácii počítadiel prijatých paketov, bytov.

Ako už však bolo spomenuté v predošlej kapitole, okrem prijatia paketov do štruktúr Mbuf, je nutné vytvorenie štruktúry `Packet`, s ktorou nasledujúce moduly Suricaty (`Decode` a `Flow Workers`) dokážu pracovať. To sa dosiahne v rámci modulu `Receive` tým, že do alokovanej štruktúry `Packet` sa funkciou `PacketCopyData()` skopírujú dáta reálne prijatého paketu zo štruktúry Mbuf, v ktorej je uložený.

Po prijatí paketu nasleduje jeho dekódovanie. Keďže typ siete na ktorom sa bude zariadenie DDoS Protector so systémom Suricata nachádzať je Ethernet, v module `Decode` bolo postačujúce nad prijatým paketom zavolať jedine funkciu `DecodeEthernet()`. Tým sa docieli dekódovanie paketu prijatého práve na tomto type sieti. Implementácia modulu `Decode` sa vôbec nelíši medzi jednotlivými DPDK módmi. Po dekódovaní dát a následnom vyhodnotení toku dát v module `FlowWorkers` dochádza k vykonaniu akcie určenej týmto modulom nad daným paketom. To sa zabezpečuje v poradí ďalším modulom `Verdict`, ktorý zabezpečuje odosielanie legitímnych paketov naspať do siete. Práve tomu je venovaná posledná časť tejto sekcie.

Odosielanie dát. Keďže tak, ako aj pri prijímaní dát, sa pracuje priamo s Ethernet portom, využíva sa funkcia `rte_eth_tx_burst()`. Táto funkcia sa používa v module `Verdict`, ktorý taktiež pristupuje k matadátam `DpdkTheadVars`. Prostredníctvom nich získava konkrétne číslo Tx fronty portu s ktorou dané vlánko pracuje. Číslo portu, ktoré sa taktiež využíva vo volaní funkcie pre odosielanie paketov sa získava z globálnej štruktúry `DpdkConfig`.

Táto sekcia sa venovala opisu implementačných detailov DPDK módu `ethdev`, pri ktorom systém Suricata prijíma a odosiela pakety priamo z fyzického portu a beží ako samostatný systém bez závislosti na inom systéme. Tento mód bol implementovaný ako prvý a to z dôvodu prvotnej integrácie DPDK technológie do systému Suricata. Avšak, pre potreby mojej bakalárskej práce bolo nutné túto implementáciu rozšíriť o módy, ktoré by spĺňali požiadavku na celkový systém a to takú, kedy systém Suricata kooperuje so systémom DCPro. Nasledujúca sekcia sa venuje práve tým implementovaným DPDK módom, ktoré túto požiadavku spĺňajú.

4.3 Módy `ethdev-ring` a `native-ring`

Hlavným cieľom tejto bakalárskej práce je integrácia systému Suricata so systémom DCPro. K dosiahnutiu tohto cieľu bolo nutné navrhnutie integrovaného režimu (sekcia 3.3). DPDK módy, ktoré implementujú tento režim sú `ethdev-ring` a `native-ring` a práve im sa bude venovať nasledujúca sekcia. Mód `ethdev-ring` implementuje variantu s prístupom cez virtuálny port k zdieľaným pamäťovým priestorom a mód `native-ring` variantu s priamym prístupom (obrázok 3.3).

Oba módy majú za úlohu spolupracovať so systémom DCPro, ktorý po spustení pracuje v primárnom procese. Tak, ako bolo uvedené aj v predošlej kapitole návrhu (viz. sekcia o integrovanom režime 3.3), systém Suricata a teda oba tieto módy musia byť spustené v sekundárnom DPDK procese. To sa zabezpečí EAL parametrom `--proc-type=secondary` pri spustení systému Suricata. Narozdiel od predošlého módu `ethdev`, EAL parameter `-w` nieje nutný, keďže tieto módy nebudú pristupovať k Ethernet portu priamo.

Štruktúra tejto sekcie je velmi podobná tej predošlej, až na to, že sa venuje dvom DPDK módom. Nasledujúca časť bude venovaná príprave systému Suricata na príjem a odosielanie dát. Po nej bude opísaný samotný príjem a odosielanie dát. V každej z nasledujúcich častí budú taktiež zdôraznené implementačné rozdiely módov `ethdev-ring` a `native-ring`.

Príprava systému. V predošlej sekcii, pri dpdk móde `ethdev` bolo postačujúce opísať jedine implementáciu inicializácie portu, čím sa zabezpečila posledná príprava systému pred prijímaním a odosielaním dát. V tejto sekcii je však opis implementácie prípravy systému rozsiahlejší. Okrem inicializácie portu v prípade `ethdev-ring`, je nutné zabezpečiť vyhľadanie štruktúr Ring, poprípade štruktúry Mempool. Taktiež je v tejto časti spomenutá implementácia druhého systému, systému DCPro. V tomto systéme, ako systéme pracujúcom pod primárnym procesom bolo nutné vytvorenie štruktúr následne vyhľadávaných systémom Suricata.

V prvom rade je teda nutné zmieniť implementačné rozširenia v systéme DCPro. Keďže tento systém podľa návrhu beží ako primárny, jedine on môže obsluhovať pamäťové priestory. Pre komunikáciu so sekundárnym procesom a teda systémom Suricata bolo nutné vytvorenie štruktúr Ring pre Rx aj Tx smer funkciou `rte_ring_create()`. Vytvorené Ringy sa využívajú v móde `ethdev-ring` aj `native-ring`. Avšak, pre beh módu `ethdev-ring` bolo nevyhnutné vytvorenie štruktúry Mempool a to funkciou `rte_pktmbuf_pool_create()`. K takto vytvoreným štruktúram Ring aj Mempool sa pri ich vytváraní priraďuje meno, na základe ktorých ich je možné v systéme so sekundárnym procesom, a teda Suricatou, vyhľadať. Po behu celého systému sa vytvorené štruktúry dealokujú taktiež systémom DCPro. Systém Suricata k nim len pristupuje a to, akým spôsobom, je opísane v nasledujúcom texte.

Prístup k systémom DCPro vytvoreným štruktúram sa zabezpečuje funkciami `rte_ring_lookup()` a `rte_mempool_lookup()`. Obe tieto funkcie vyžadujú jeden parameter, ktorým je názov hladanej štruktúry. Názov hľadaných štruktúr Ring sa skladá z troch častí. Prvou je spoločná časť názvu, ktorá je konfigurovaná administrátorom systému v YAML konfiguračnom súbore položkou `port-extern` v sekcii `dpdk`. Druhou časťou názvu je označenie smeru danej štruktúry Ring a to Rx, alebo Tx. Poslednou časťou je index, ktorý označuje taktiež index vlákna, ktoré bude s danou dvojicou Rx/Tx Ringov pracovať.

Rozdiel medzi módmi opisovanými v tejto sekcii je ten, že pre mód `native-ring` sa vyhľadaním všetkých Rx/Tx Ringov, implementácia posledných príprav pred prijímaním a odosielaním dát končí. To z dôvodu, že tento mód využíva štruktúry Ring priamo pre prijímanie a odosielanie dát a nie cez port, ako mód `ethdev-ring`.

Keďže mód `ethdev-ring` pristupuje k Ringom cez port a s fyzickým portom nie je možné pri týchto módoch pracovať, je nutné si daný port vyrvoriť. Pre vytvorenie takéhoto virtuálneho portu sa využila funkcia `rte_eth_from_rings()`. Parametrami tejto funkcie sú práve nájdené štruktúry Ring pre Rx, aj Tx smer. Takto vytvoreným portom sa teda nepriamo môže pristupovať k systémom DCPro vytvoreým Ringom. Keďže mód `ethdev-ring` bude pracovať s portom, aj keď virtuálnym, je nutné tento port taktiež, ako reálny fyzický

port, inicializovať. Inicializácia portu je popísaná v predošlej sekcii 4.2 venovanej módu `ethdev`.

Referencia na takto inicializovaný port, či aj na konkrétne štruktúry Ringov sa ukladá v tomto bode implementácie do globálnej štruktúry `DpdkConfig`, ku ktorej sa v priebehu spracovávania paketov pristupuje. Nasledovná časť sa venuje prijímaniu dát.

Príjem dát. Príjem dát je pre módy `ethdev-ring` a `native-ring` rozdielny, avšak len DPDK API funkciou ktorá tento príjem vykonáva. Implementácia príjmu dát je pre všetky módy realizovaná v module `Receive`. V prvom rade sa táto časť venuje módu `ethdev-ring`. Tento mód prijíma dáta rovnakou implementáciou, ako mód `ethdev`. Podrobnejší popis tejto implementácie je uvedený v predošlej sekcii 4.2.

Rozdiel v implementáciách príjmu dát dochádza až v prípade módu `native-ring`. Keďže tento mód nepracuje s fyzickým Ethernet portom, ani tým virtuálnym, ale priamo so štruktúrami Ring, bolo nutné využiť funkciu z DPDK RTE Ring knižnice. Tou funkciou je `rte_ring_dequeue_burst()`, ktorá zabezpečuje príjem paketov. Avšak, tak ako aj pri predošlých dvoch módoch, aj pri tejto funkcii sa k prijatým paketom pristupuje cez pole ukazateľov na štruktúry Mbuf, v ktorých sú dané pakety uložené. Hlavným rozdielom, čo sa parametrov týka, je parameter odkazujúci na konkrétnu štruktúru Ring namiesto identifikačného čísla Ethernet portu, ako je tomu pri predošlých dvoch módoch.

V nasledujúcej časti bude opísaná implelementácia odosielania dát, ale zmeny medzi jednotlivými módmi sú totožného charakteru, ako aj pri ich príjme.

Odosielanie dát. Tak, ako aj pri príjme dát, mód `ethdev-ring` využíva implementáciu rovnakú, ako mód `ethdev`. Podrobnejšie informácie sa nachádzajú v predošlej sekcii 4.2.

Mód `native-ring` aj pri tejto operácii pristupuje k štruktúram Ring priamo a to funkciou taktiež z DPDK RTE Ring knižnice, `rte_ring_enqueue_burst()`.

V systéme DCPro, sa presun paketov do štruktúr Ring, pre dalšie spracovávanie systémom Suritata, zabezpečuje taktiež funkciami `rte_ring_dequeue_burst()` a `rte_ring_enqueue_burst()`. Rozdiel je len v tom, že systém DCPro posiela pakety do tých Ring štruktúr, z ktorých ich systém Suricata prijíma a naopak.

V tejto kapitole bola opísaná implementácia podpory prijímania a odosielania dát systémom Suricata prostredníctvom technológie DPDK. Taktiež sa kapitola venovala popisu implementácie integrovania systému Suricata so systémom DCPro prostredníctvom tejto technológie. Pre prehľadnú správu systému Suricata a využitie jej možností v rôznych situáciách boli implementované tri módy a to `ethdev`, `ethdev-ring` a `native-ring`. Posledné dva zmienené módy sa využívajú v integrovanom režime so systémom DCPro. Nasledujúca kapitola sa bude venovať výsledkom testovania, kde budú porovnané nielen jednotlivé implementované DPDK režimy, ale aj výkonnostné zlepšenia v spracovávaní sieťovej prevádzky po zavedení podpory technológie DPDK do systému Suricata.

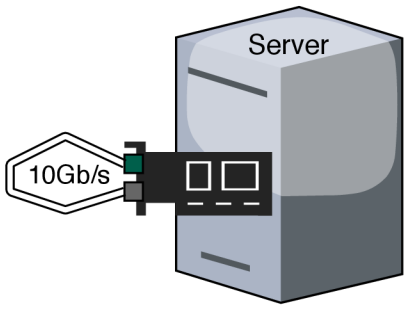
Kapitola 5

Testovanie a meranie výkonnosti

Táto kapitola je zameraná na popis výsledkov testovania. Keďže sa celkový systém rozdeľuje na dva režimy a to na samostatný a integrovaný režim (viď kapitolu 3), postup testovania sa odvíjal na základe týchto režimov. V prvom rade sú v sekcii 5.2 predstavené dosiahnuté výsledky po pridaní podpory technológie DPDK (sekcia 2.6) do systému Suricata (sekcia 2.5) a to testy nad DPDK módom `ethdev`. Následne sú v sekcii 5.3 popísané výsledky testovania po integrácii systému Suricata so systémom DCPro (sekcia 2.5) módmi `ethdev-ring` a `native-ring`. Hlavným ukazateľom správnosti dosahovaných výsledkov je priepustnosť daného systému. Tá sa meria na základe porovnania počtu vygenerovaných paketov a počtu úspešne prijatých paketov. Vzhľadom na fakt, že na počet úspešne prijatých paketov má vplyv aj počet konfigurovaných signatúr systému Suricata, ktoré sú určené na porovnávanie s aktuálnou prevádzkou, aj tento typ testu bol vykonaný. Ďalším faktorom výrazne vplyvným na celkovú výkonnosť systému je podpora multivláknovosti. Aj tento test zobrazujúci zvýšenie výkonnosti pri zvýšení počtu vlákien bol vykonaný tiež. V sekcii venovanej meraniu výkonnosti integrovaného režimu budú taktiež zobrazené aj výsledky porovnania dvoch módov implementujúcich tento režim a to módov `ethdev-ring` a `native-ring` (viď sekciu 4.3). Avšak, ešte pred samotným popisom dosiahnutých výsledkov sa bude nasledujúca sekcia 5.1 venovať popisu prostredia, v ktorom sa testy vykonávali.

5.1 Testovacie prostredie

V tejto sekcii je predstavené prostredie, ktoré slúžilo na vykonávanie jednotlivých testov, ako aj pri samotnom vývoji systému. Taktiež je v tejto sekcii opísaný spôsob vykonávania testov. Nasledujúci obrázok 5.1 zobrazuje hardvérové zapojenie testovacieho prostredia.



Obr. 5.1: Zapojenie testovacieho servera.

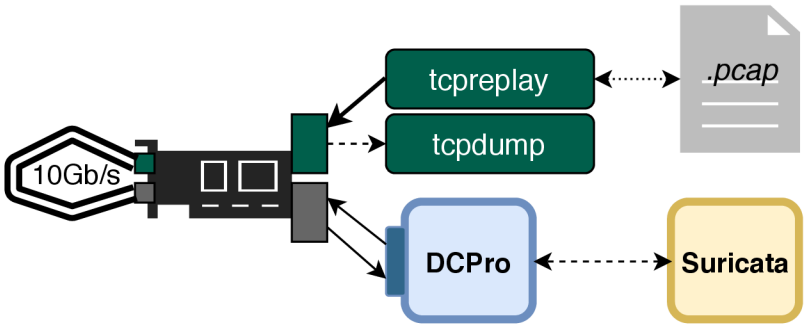
Obrázok 5.1 zobrazuje server, na ktorom boli testy vykonávané. Server disponuje okrem iného aj sieťovou kartou od spoločnosti Intel[1], ktorá obsahuje dva porty (zelený a sivý na obrázku 5.1). Tie sú navzájom prepojené optickou linkou o rýchlosti 10Gb/s. V nasledujúcej tabuľke 5.1 sú zobrazené špecifiká daného servera z pohľadu hardvéru a aj softvéru.

Hardvérové parametre	
Typ servera	DELL R720
Procesor – CPU	Xeon(R) CPU E5-2620 0 @ 2.00GHz
Počet jadier CPU	12
RAM	64GB DDR4
Základná doska – MB	Dell Inc. 0VWT90
Sieťová karta – NIC	Intel X520-SR2 10GbE

Softvérové parametre	
Operačný systém	Scientific Linux
Verzia jadra	3.10.0-1062.12.1.el7.netx.x86_64
Verzia DPDK	v19.08.2
Verzia DCPro	v0.9.2.
Verzia Suricata	v5.0.2

Tabuľka 5.1: Špecifiká testovacieho servera

Okrem hardvérových špecifík testovacieho stroja, vyššie uvedená druhá tabuľka uvádza aj verzie softvérov, ktoré boli v mojej práci využívané a na danom stroji inštalované. Avšak, z hardvérových parametrov je nutné sa ešte podrobnejšie zmieniť o jednom komponente a to o sieťovej karte, resp. jej portoch. Pre úspešný chod cieľového systému a síce ktorejkoľvek aplikácie vyvíjanej nad technológiou DPDK je nutné, aby ovládače využívaného portu boli nastavené korektne. Pre sprístupnenie priamej komunikácie užívateľskej aplikácie cez PMD ovládač bolo nutné nastaviť na port, využívaný priamo DPDK aplikáciou, kernel ovládač IGB_UIO[2]. Týmto ovládačom sa sprístupnili hardvérové registre a umožnila sa komunikácia so sieťovou kartou z užívateľského priestoru systému prostredníctvom PMD ovládaču. Na obázku 5.2 je takto nastavený port sivej farby. Druhý port, zelenej farby, disponuje kernel ovládačom IXGBE, štandarne využívaným pre Intel sieťové karty rýchlosti 10Gb.



Obr. 5.2: Schéma spôsobu testovania.

[1] https://ark.intel.com/content/www/us/en/ark/products/39774/intel-ethernet-converged-network-adapter-x520-sr2.html

[2] https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html

Aj napriek tomu, že bolo vykonaných niekoľko typov testov, spoločným menovateľom je spôsob ich vykonávania. Princíp testovania je znázornený na obrázku 5.2 a celé testovanie prebiehalo na jednom serveri. V prvom rade testovanie spočíva v generovaní sieťovej prevádzky uloženej na SSD disku servera vo formáte PCAP. Toto generovanie bolo vykonávané pomocou nástroja `tcpreplay`. Po takto generovanej sieťovej prevádzke na port, ktorý je prepojený priamo na druhý port sieťovej karty toho istého serveru, bolo možné túto prevádzku prijímať s využitím systému DCPro [3]. Po analýze a spracovaní danej prevádzky aj systémom Suricata, bola sieťová prevádzka odposlaná naspäť systémom DCPro, prípadne priamo systémom Suricata (samostatný režim). Na porte znázornenom zelenou farbou v obrázku 5.2 sa v prípade funkčných testov v priebehu vývoja taktiež zachytávala sieťová prevádzka a to nástrojom `tcpdump`, čím sa kontroloval aj počet úspešne spätne preposlaných paketov.

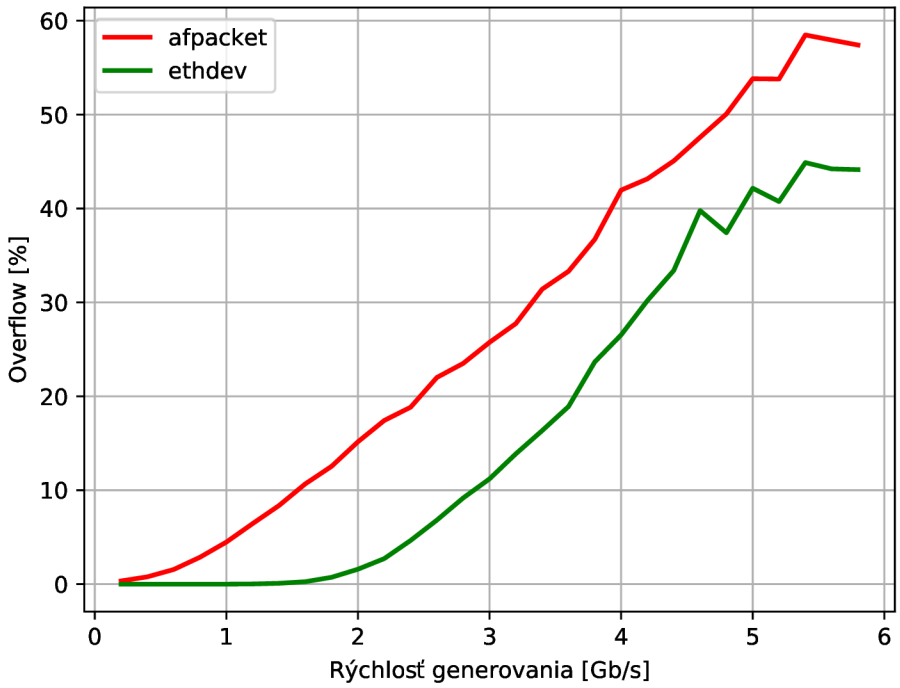
Táto sekcia sa venovala popisu testovacieho prostredia, avšak nasledujúce sekcie sú už zamerané priamo na popis dosiahnutých výsledkov. V nasledujúcej sekcii 5.2 budú znázornené výsledky po zavedení podpory technológie DPDK do systému Suricata a ich porovnanie s iným, už štandardne využívamým módom tohto softvéru.

[3] V prípade samostatného režimu sa prijímali pakety systémom Suricata priamo z portu (obrázok 3.1)

5.2 Samostatný beh systému Suricata

V prvom rade bolo nutné overenie úspešnosti pridania podpory technológie DPDK do systému Suricata. Táto podpora sa implementovala módom `ethdev` (sekcia 4.2). Testovaniu funkčnosti a výkonnosti tohto DPDK módu sa bude venovať práve táto sekcia. Okrem toho bude v jednotlivých grafoch zobrazené aj porovnanie tohto DPDK módu so štandardne využívaným módom systému Suricata, ktorým je mód AF-PACKET. Oba tieto módy boli v priebehu testovania spúšťané nad jedným PPT vláknom, čomu zodpovedajú aj hodnoty uvedné v nasledujúcich grafoch.

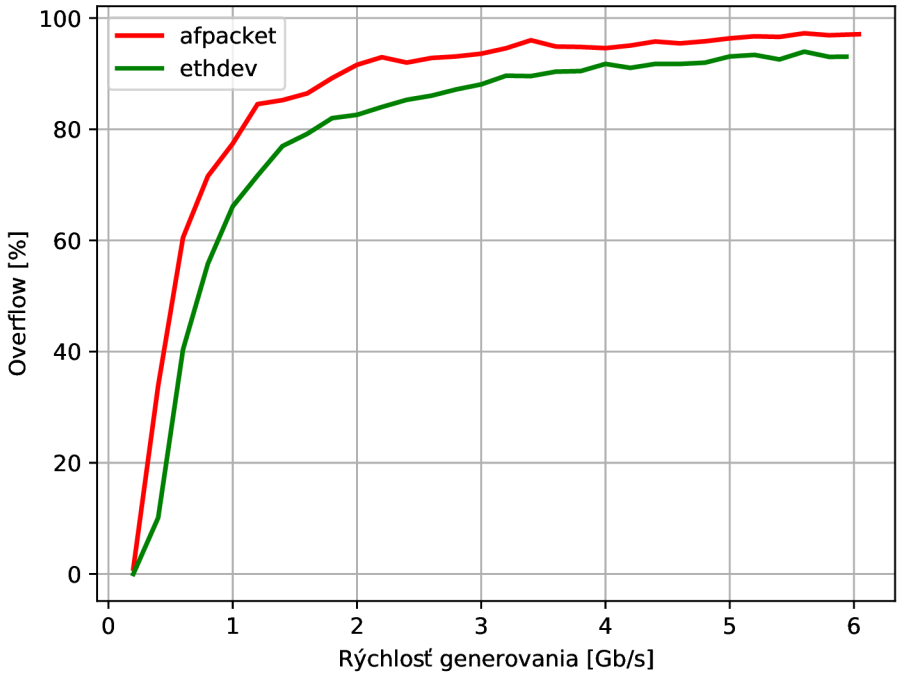
Nad samostatným režimom DPDK verzie systému Suricata sa vykonal len jeden typ testu. Jedná sa o test zameraný na overenie množstva zahodenej sieťovej prevádzky, ktoré sa ekvivalentne zvyšuje so zvyšovaním rýchlosti generovania vtupnej sieťovej prevádzky. Tento typ testu má následne dve varianty a to, variantu spúšťania systému Suricata bez prítomnosti signatúr a naopak, za ich prítomnosti. Nasledujúce grafy 5.3 a 5.4 zobrazujú porovnanie módu AF-PACKET a DPDK `ethdev` z hľadiska výkonnosti spracovávania paketov. Prvý graf 5.3 zobrazuje výkonnosť módov bez nahratých signatúr. X-ová os nasledujúceho grafu zobrazuje rýchlosť generovania sieťovej prevádzky nástrojom `tcpreplay`. Y-ová os grafu zobrazuje tzv. *overflow* hodnotu, ktorá označuje podiel zahodených paketov voči vygenerovaným paketom. Ako je vidieť z daného grafu, DPDK mód `ethdev` (zelená krivka) je v priemere o 10% až 15% výkonnejší a to pri každej rýchlosti generovania.



Obr. 5.3: Test priepustnosti samostatného režimu bez signatúr.

Nasledujúci test, ktorého výsledky zobrazuje graf 5.4 sa líši od toho predošlého tým, že systém Suricata je spúšťaný s výrazným množstvom konfigurovaných signatúr, konkrétne sa jedná o 20245 signatúr. Jedná sa o signatúry Emerging Threats[4], ktoré sú pre systém Suricata štandardne už v inšatalácii dostupné.

[4] https://rules.emergingthreats.net/open/suricata/rules/



Obr. 5.4: Test priepustnosti samostatného režimu so signatúrami.

Ako je možné vidieť z grafu 5.4, takéto množstvo signatúr výrazne vplýva na celkovú výkonnosť systému. Taktiež je z grafu vidieť zmenšenie výkonnostného rozdielu módov `afpacket` a `ethdev`. Dôvod tohto poklesu spočíva v tom, že pre analýzu jednotlivých paketov naprieč signatúrami sa pre oba módy používajú rovnaké implementácie modulov `Decode`, aj `Flow Workers` (obrázok 3.2). A vzhľadom na veľké množstvo signatúr, systém Suricata využíva tieto moduly vo vačšej miere a na celkový výkon systému majú väčší dopad práve tieto moduly. Moduly určené pre príjem a odosielanie paketov (`Receive` a `Verdict`), ktorých implementácia sa pre jednotlivé módy líši, majú v tejto situácií menší vplyv na výkonnosť systému a tým pádom sa výkonnostný rozdiel medzi módmi `afpacket` a `ethdev` zmenšuje. Taktiež ako aj pri predošlom grafe 5.3, jedná sa o výsledky merania nad jedným vláknom, čomu odpovedajú aj namerané hodnoty.

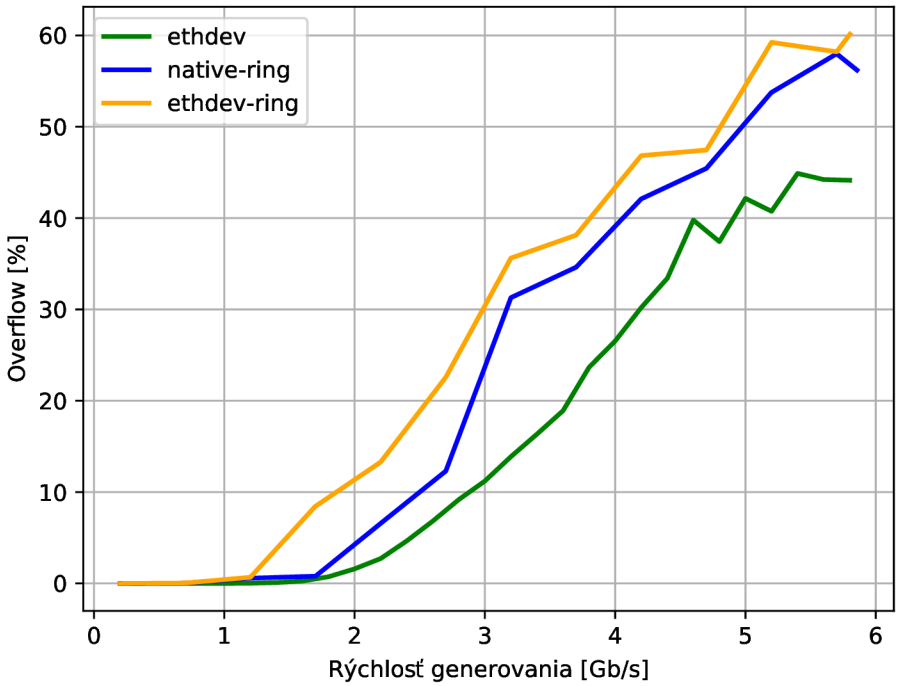
Táto sekcia sa venovala testom zameraným na overenie výkonnosti systému Suricata po implementácii podpory technológie DPDK. Nasledujúca sekcia sa venuje testom zameraných na overenie výkonnosti a funkčnosti integrovaných DPDK módov `ethdev-ring` a `native-ring`.

5.3 Beh integrovaného systému Suricata so systémom DCPro

Druhá sada testov bola vykonávaná nad integrovaným režimom (sekcia 3.3) systému Suricata. Pri integrovanom režime sa testy delili na dva typy a to testy zamerané na výkonnosť spracovania paketov a testy zamerané na schopnosť detekcie útokov, resp. udalostí za rôznych rýchlostí generovania vstupnej sieťovej prevádzky. Testy sa vykonávali za behu primárnej DPDK aplikácie, ktorou bol systém DCPro. Tento systém taktiež umožňoval distribúciu sieťovej prevádzky do piatich Ringov (sekcia 2.6). Každý Ring je spracovaný

jedným vláknom, resp. jadrom, čím dochádza k paralelnému multivláknovému spracovaniu. Výkonnosť celkového chodu integrovaného režimu sa tým zvyšuje.

Nasledujúci graf 5.5 však zobrazuje výkonnostné porovnanie módov integrovaného režimu s módom `ethdev` samostatného režimu. Jedná sa o výkonnostný test bez prítomnosti signatúr.

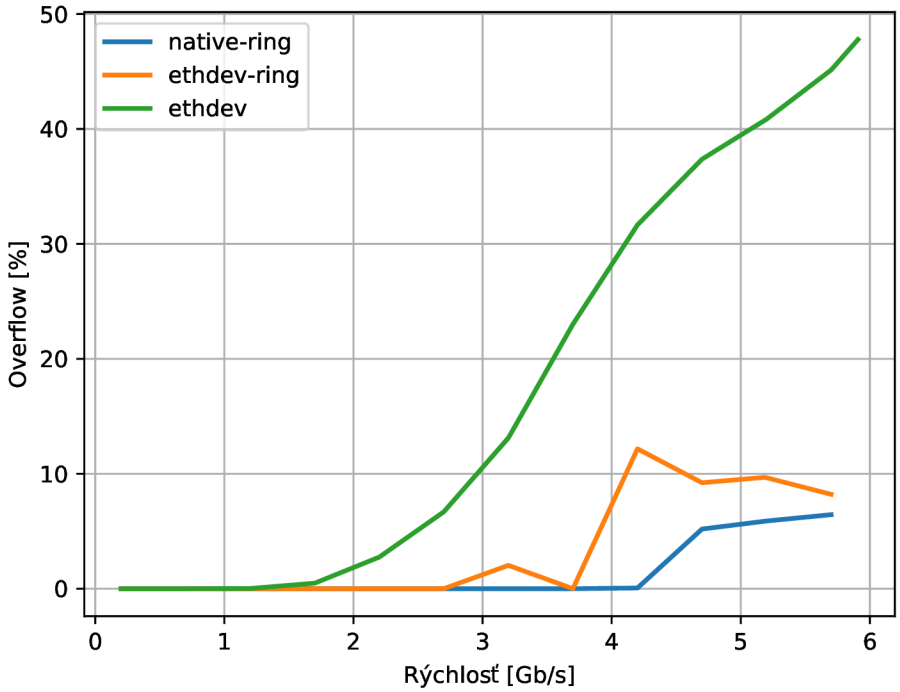


Obr. 5.5: Porovnanie výkonnosti DPDK módov systému Suricata.

Pre zabezpečenie korektnosi porovnávania boli pri tomto porovnávaní, tak ako aj mód `ethdev`, aj módy `native-ring` a `ethdev-ring` spustené nad jedným vláknom. Graf 5.5 zobrazuje mierny výkonnostný pokles módov integrovaného režimu. To je práve z dôvodu ich nepriamej komunikácii s portom sieťovej karty. Tým, že sa jednotlivé pakety preposielajú cez systém DCPro do systému Suricata, dochádza k miernemu spomaleniu. Mierne rozdiely možno vidieť aj medzi jednotlivými módmi integrovaného režimu. Mód `ethdev-ring` pristupuje k zdieľaným pamäťovým priestorom, kde sú uložené pakety primárnou aplikáciou, prostredníctvom virtuálneho portu. Mód `native-ring` však pristupuje priamo, ako je to zobrazené aj na obrázku 3.3. V dôsledku tohto priameho prístupu k zdieľaným pamäťovým priestorom má mód `native-ring` mierne lepšiu výkonnosť.

V ďalšej časti tejto sekcie budú zobrazované grafy pre módy `native-ring` a `ethdev-ring` obsahovať výkonnostné hodnoty získané nad behom systému Suricata v piatich PPT vláknach.

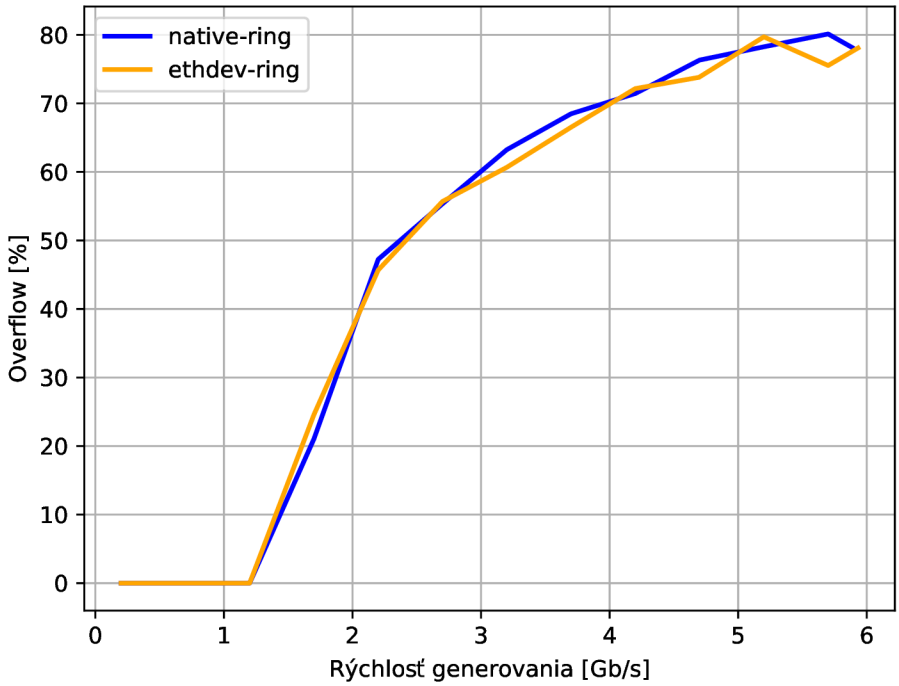
Na nasledujúcom grafe 5.6 je znázorený výsledok testu priepustnosti jednotlivých DPDK módov bez prítomnosti signatúr.



Obr. 5.6: Test priepustnosti integrovaného režimu bez signatúr.

Zelená krivka znázorňuje DPDK mód `ethdev`. Ako je vidieť, z grafu 5.6, tento mód je v tomto prípade výrazne memej výkonný, než DPDK módy integrovaného režimu. To je práve z dôvodou jeho behu len nad jedným PPT vláknom, resp. jadrom procesoru. Zvyšné dva módy, ktoré implementujú integrovaný režim poskytujú výrazne vyššiu výkonnosť v spracovávaní sieťovej prevádzky. Oba tieto módy pracujú nad piatimi PPT vláknami a každé toho vlákno beží na samostatnom jadre procesoru. To je práve dôvod ich vysokej výkonnosti. Mierne odlišnosti možno badať aj v porovnaní módov `native-ring` a `ethdev-ring`. Ako bolo uvedné v návrhovej kapitole (obrázok 3.3), tieto dva módy sa odlišujú v ich prístupe k zdieľaným pamäťovým priestorom, ktoré sa vytvárajú primárnou DPDK aplikáciou (DCPro). Vzhľadom na fakt, že mód `ethdev-ring` pristupuje k týmto priestorom prostredníctvom tzv. virtuálneho portu a nie priamo, tak ako `native-ring`, môže dochádať k miernemu omeškaniu. Následkom toho môže byť strata paketov. Tento jav taktiež vidieť z grafu 5.5.

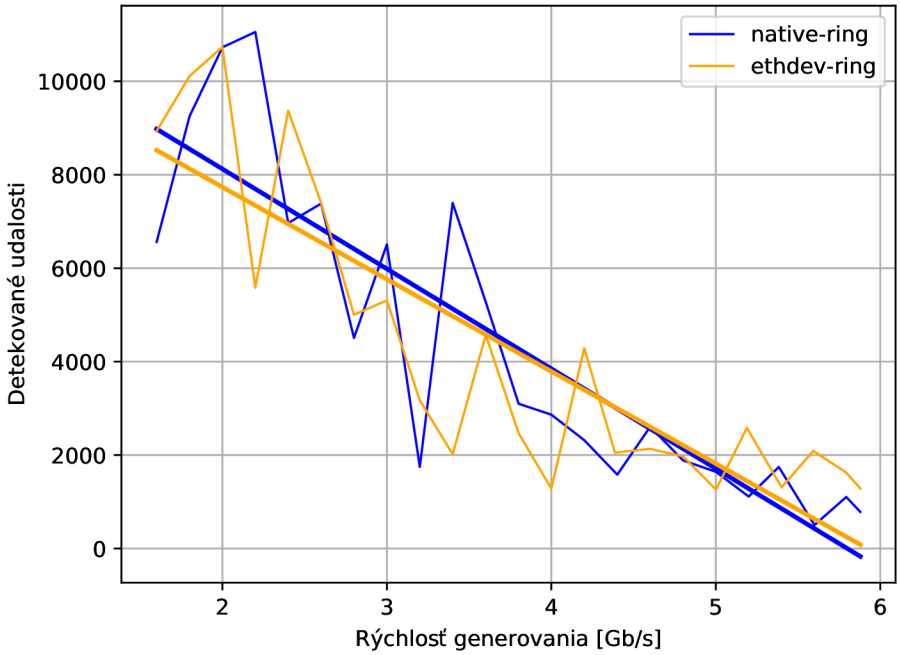
Avšak, celková priepustnosť systému sa opäť rapídne znižuje v prípade, ak sa do systému Suricata pridajú signatúry, ktorými sa spracovávaná sieťová prevádzka následne analyzuje. Tento rozdiel v priepustnosti systému je zobrazený na nasledujúcom grafe 5.7.



Obr. 5.7: Test priepustnosti integrovaného režimu so signatúrami.

V priebehu testu, ktorého výsledky sú zobrazené grafom 5.7, boli aktívne v systéme Suricata aj signatúry. Počet aktívnych signatúr bol 20245. Tento vysoký počet signatúr, s ktorými sa spracovávané sieťové toky porovnávali, výrazne ovplyvnil priepustnosť celého systému, tak ako aj pri samostatnom režime v predošlej sekcii (graf 5.4). Avšak, rozdiel medzi módmi integrovaného režimu (`native-ring` a `ethdev-ring`) nie je tentokrát taký výrazný. Keďže hlavná záťaž v spracovávaní jednotlivých paketov je v moduloch `Decode` a `Flow Workers` (obrázok 3.2), vplyv spôsobu prijímania a odosielania jednotlivých paketov na celkový výkon systému nie je až taký významný.

Posledným testom vykonávaným nad DPDK módmi integrovaného režimu je test schopnosti detekcie udalostí v závislosti od rýchlosti prijímanej sieťovej prevádzky. Výsledky tohto testu sú zobrazené v grafe 5.8. Avšak, bolo nutné do výsledného grafu pridať polynómy pre oba módy, ktoré jasne zobrazujú tendenciu správania systému pri zvyšovaní rýchlosti generovanej sieťovej prevádzky. Dôvod nepresnosti a vysokej nestability detekovaných udalostí pri rôznych rýchlostiach generovania spočíva v spôsobe ich generovania. Sieťová prevádzka, ktorá je prehrávaná nástrojom `tcpreplay`, bola zachytená z reálnej siete v určitej prenosovej rýchlosti. Prehrávanie takto zachytenej sieťovej prevádzky v rôznych rýchlostiach má za následok jej časovú deformáciu, čím dochádza k nestabilite v rámci analýzy celých sieťových tokov systémom Suricata. Tento test však, aj vzhľadom na citlivú analýzu sieťových tokov naprieč signatúrami, poskytuje podstatný ukazateľ správnosti výsledného systému. Z grafu 5.8 je možné vyčítať tendenciu znižovania počtu úspešne detekovaných udalostí v závislosti od zvyšovania rýchlosti generovanej sieťovej prevádzky smerujúcej do systému Suricata. Táto tendencia je zobrazená polynomom preloženým cez body nameraných hodnôt pre oba módy.



Obr. 5.8: Test schopnosti detekcie udalostí integrovaným režimom.

5.4 Zhrnutie dosiahnutých výsledkov

Táto kapitola bola venovaná popisu dosiahnutých výsledkov. Tak, ako aj implementácia celkového systému bola rozdelená na dve časti, resp. dva režimy, tak aj testovanie v záverečnej fáze mojej práce bolo rozdelené na dve časti. Predchádzajúce dve sekcie 5.2 a 5.3 zobrazujú výsledky dosiahnuté nad samostatným režimom (sekcia 3.2) systému Suricata a taktiež nad integrovaným režimom (sekcia 3.3). Na celkové výsledky má samozrejme zásadný vplyv aj prostredie, v ktorom boli testy vykonávané a preto sa úplne prvá sekcia 5.1 tejto kapitoly venovala jeho popisu.

Výsledky druhej sekcie 5.2 zobrazovali namerané hodnoty pre samostatný režim, resp. DPDK mód `ethdev` a mód AF-PACKET. Výsledky oboch módov sú merané pri ich behu nad jedným vláknom a to z dôvodu, že mód `ethdev` nepodporuje multivláknové spracovanie. Práve pridaním multivláknového spracovania je možné túto prácu ďalej rozšíriť. Napriek tomu je možné z grafov 5.3 a 5.4 usúdiť výkonnostné výhody mnou implementovaného DPDK módu `ethdev`.

Tretia sekcia 5.3 sa venovala výsledkom testov nad integrovaným režimom systému Suricata a to konkrétne DPDK módom `ethdev-ring` a `native-ring`. Z prvých dvoch grafov 5.6 a 5.7 uvedených v tejto sekcii je možné porovnávaním s grafmi sekcie 5.2 o samostatnom režime vidieť výrazné výkonnostné zlepšenie. Toto zlepšenie je z dôvodu spracovávania sieťovej prevádzky nad piatimi PPT vláknami a zároveň piatimi rôznymi jadrami procesoru.

Všetky tieto grafy sú zamerané na vyhodnotenie výkonnosti systému. Výkonnosť systému sa meria rozdielom počtu generovaných a úspešne prijatých paketov systémom Suricata v rôznych rýchlostiach generovania. Tento typ testu sa rozdeľuje na dve varianty a to variantu, kedy sa výkonnosť systému meria bez prítomnosti signatúr (grafy 5.3 a 5.6) a na variantu kedy je výkonnosť obmedzená vysokým počtom konfigurovaných signatúr (grafy 5.4 a 5.7).

Avšak, integrovaný režim bol overený aj dalším typom testu a tým je test na overenie schopnosti detekcie udalostí za rôznych rýchlostí generovania sieťovej prevádzky. Z grafu 5.8 je možné vidieť výsledky tohto testu. Tak ako je uvedené v sekcii 5.3, dôvod vysokej nestability tohto testu je zapríčinený spôsobom generovania zachytenej sieťovej prevádzky. Tá je zachytená z reálnej siete v určitej rýchlosti a jej generovanie v rôznych iných rýchlostiach má za následok jej časovú deformáciu a nestabilnú analýzu v systéme Suricata. Avšak, pre potreby základného overenia správnosti systému boli nameranými hodnotami preložené polynómy. Tie naznačujú správnu tendenciu správania systému pri zvyšovaní rýchlosti generovania. Rýchlejšie generovanie má za následok nižší počet detekovaných udalostí. Problém s generovaním a následnou deformáciou zachytenej sieťovej prevázky sa dá riešiť nasledovným spôsobom. Rýchlosť generovania bude v konštantnej rýchlosti, ale sieťová prevádzka sa bude znásobovať, čím sa vo výsledku dosiahne vyššia rýchlosť pri prijímaní prevádzky systémom Suricata. V tomto prípade bude nutné dodatočné nastavenie aj samotného systému, vzhľadom na príjem duplikátnych paketov. To však bude už predmetom rozšírenia mojej práce.

Suricata je velmi zložitý systém so širokou škálou volieb, ktoré umožňujú výkonnosť tohto systému ďalej optimalizovať. Pre jednoduchosť som však využíval štandardnú konfiguráciu. Ďalším pokračovaním práce teda môže byť posúvanie týchto výkonnostných limitov a optimalizácia výkonu, ako už samotnou konfiguráciou systému Suricata, tak aj profilovaním a úpravami môjho kódu. Taktiež po zoptimalizovaní sa následné testy môžu vykonávať na karte so 100Gb/s priepustnosťou namiesto 10Gb/s, ktorá bola využívaná v tejto práci.

Ďalšou možnosťou rozšírenia je pridanie konfiguračného rozhrania spoločného pre systém Suricata aj DCPro. Vzhľadom na to, že systém DCPro už využíva prostredie NetX Netc[5], vhodnou variantou by bolo rozšírenie tohto prostredia pre konfiguráciu aj priamo systému Suricata. Tým by sa správa celého systému zjednodušila. Z dlhodobého hľadiska je zaujímavá aj možnosť zavedenia tzv. *Bypass* režimu. Tento režim je štandardne podporovaný systémom Suricata a pri detekcii umožňuje určenie tokov, ktoré už nieje nutné analyzovať a je možné ich priamo filtrovať. Táto funkcia je riešená akýmsi predfiltrom. Tento softvérový filter je následne možné akcelerovať priamo na sieťovej karte. Využitie tejto funkcie Suricaty by bolo možné prostredníctvom návrhu komunikačného rozhrania, s ktorým by systém Suricata mohol posielať správy predfiltru priamo systému DCPro. Systém DCPro by následne samotnú predflitráciu vykonal už samostatne, prípadne by sa pravidlá pre predfiltráciu nahrávali priamo do sieťovej karty.

[5] https://docs.netx.as/docs/cli/netc-cli.html

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bola integrácia systému IDS do už existujúceho zariadenia DDoS Protector – DCPro. DCPro v svojej aktuálnej verzii využíva technológiu DPDK pre spracovávanie vysokorýchlostnej sieťovej prevádzky. Z tohto dôvodu bolo nutné mnou vybraný IDS systém Suricata upraviť a rozšíriť o podporu technológie DPDK. Po pridaní podpory tejto technológie bolo možné zavedenie integrácie týchto dvoch systémov.

Začiatok riešenia môjho zadania spočíval v štúdiu systémov IDS a ich konceptu pri zabezpečení počítačových sietí. Po porovnaní rôznych open-source IDS systémov som vybral systém Suricata. Tento systém som vybral z dôvodu jeho výkonnostných výhod spomedzi konkurenčných riešení. Zoznamovanie sa s týmto systémom prebiehalo spočiatku len na užívateľskej a teoretickej úrovni. Následne však bolo nutné sa zorientovať aj v samotnej implementácii systému pre budúci návrh riešenia integrácie. Po štúdiu systému Suricata nasledoval systém DCPro, ktorého koncept, zameranie a z časti aj implementáciu bolo nutné naštudovať. Tento systém v svojej najnovšej verzii využíva technológiu DPDK, ktorá umožňuje vysokorýchlostné spracovávanie sieťovej prevádzky. Vzhľadom na to bolo nutné zoznámenie sa aj s touto technológiou a jej možnosťami využitia pri integrácii.

Po štúdiu všetkých podstatných súčastí budúceho integrovaného systému som vytvoril konceptuálny návrh riešenia integrácie systému Suricata a DCPro. Návrh som rozdelil do dvoch režimov, do samostatného a integrovaného režimu. Samostatný režim slúži na prvotné pridanie podpory technológie DPDK do systému Suricata. V prípade samostatného režimu systém Suricata funguje bez závislosti na inom systéme a pakety prijíma priamo z fyzického portu. Integrovaný režim slúži na integrovanie systému Suricata so systémom DCPro. Tento režim funguje v závislosti na druhom, primarnom systéme – DCPro, od ktorého následne aj prijíma sieťovú prevádzku.

Pre oba spomínané režimy boli v implementácii systému Suricata vytvorené tzv. módy. Samostatný režim implementuje mód `ethdev`. Implementácia integrovaného režimu sa rozdeľuje do dvoch módov – `ethdev-ring` a `native-ring`. Tieto dva módy majú rovnaký účel, ale ich implementácia sa mierne líši. Pre úspešnú implementáciu integrovaného režimu bolo taktiež nutné pridanie úprav do systému DCPro.

Po takto implementovaných módoch a overení ich funkčnosti nasledovalo výkonnostné testovanie. Testovanie spočívalo v porovnávaní výkonnosti medzi jednotlivými módmi. Výkonnosť sa merala na základe počtu vygenerovaných a počtu úspešne prijatých paketov. Výslednými grafmi sa zobrazuje závislosť vplyvu rýchlosti generovanej sieťovej prevádzky na počet úspešne prijatých paketov. Tieto výkonnostné testy boli rozdelené na dve varianty. Prvou z nich bola varianta testov bez prítomonosti signatúr systému Suricata a druhou bola varianta, kedy sa testy vykonávali za prítomnosti veľkého množstva signatúr. Posledným

testom bol test zameraný na schopnosť detekcie útokov systémom Suricata a jej závislosť na rýchlosti generovanej sieťovej prevádzky.

V rámci tejto bakalárskej práce sa mi úspešne podarilo pridanie podpory DPDK do systému Suricata a aj jeho následná integrácia so systémom DCPro. Výsledky mojej práce majú pritom praktické využitie pri rozvoji zariadenia DDoS Protector. Dosiahnutá výkonnosť však pre priame praktické využitie nieje ideálna. Určitý vplyv na výkonnosť mohlo mať aj samotné testovacie prostredie. Ďalšou oblasťou, kde sú určite výkonnostné rezervy je konfigurácia systému Suricata, ktorú je možné ďalej optimalizovať. Ďalšou možnosťou rozšírenia je pridanie konfiguračného rozhrania spoločného pre oba systémy DCPro a Suricata, prípadne podpora tzv. *Bypass* režimu, ktorý umožní akceleráciu celého systému s využitím predfiltrácie priamo na sieťovej karte. Tieto a dalsie možnosti rozširovania tejto práce budú predmetom ďalšej naväzujúcej spolupráce so združením CESNET.

Literatúra

[1] *DPDK* [online]. [cit. 2019-02-21]. Dostupné z: https://www.dpdk.org.

[2] *DPDK Documentation* [online]. [cit. 2019-02-21]. Dostupné z: https://doc.dpdk.org/guides-19.08/.

[3] *Suricata* [online]. [cit. 2019-12-12]. Dostupné z: https://suricata-ids.org.

[4] *Suricata-Update* [online]. [cit. 2020-02-05]. Dostupné z: https://suricata.readthedocs.io/en/suricata-5.0.2/rule-management/suricata-update.html.

[5] Bogdanoski, M., Suminoski, T. a Risteski, A. Analysis of the SYN flood DoS attack. *International Journal of Computer Network and Information Security (IJCNIS)*. MECS Publisher. 2013, roč. 5, č. 8, s. 1–11.

[6] CESNET. *DDoS Protector* [online]. [cit. 2019-11-23]. Dostupné z: https://www.liberouter.org/technologies/ddos-protector/.

[7] CESNET. *The NIX.CZ National Interconnection Node will Test DDoS Protection Developed by CESNET* [online]. [cit. 2020-05-09]. Dostupné z: https://www.cesnet.cz/cesnet/reports/press-releases/the-nix-cz-national-interconnection-node-will-test-ddos-protection-developed-by-cesnet/?lang=en.

[8] Chakraborty, N. Intrusion detection system and intrusion prevention system: A comparative study. *International Journal of Computing and Business Research (IJCBR)*. 2013, roč. 4, č. 2, s. 1–8.

[9] Day, D. J. a Burns, B. M. A performance analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines. In: *ICDS 2011*. 2011.

[10] Hank, A. *Detekce narušení počítačové sítě*. Brno, CZ, 2007. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Dostupné z: https://www.fit.vut.cz/study/thesis/5828/.

[11] Herbert, T. a Bruijn, W. de. *Scaling in the Linux Networking Stack* [online]. [cit. 2019-04-13]. Dostupné z: https://www.kernel.org/doc/Documentation/networking/scaling.txt.

[12] IMPERVA. *HTTP Flood* [online]. [cit. 2019-11-14]. Dostupné z: https://www.imperva.com/learn/application-security/http-flood/.

[13] Kupreev, O., Badovskaya, E. a Gutnikov, A. *DDoS attacks in Q3 2019* [online]. [cit. 2019-11-02]. Dostupné z: https://securelist.com/ddos-report-q3-2019/94958/.

[14] Kučera, J. *Hardwarová akcelerace pro IDS systémy.* Brno, CZ, 2019. Diplomová práca. Brno University of Technology, Faculty of Information Technology.

[15] Matoušek, P. *Síťové služby a jejich architektura.* Publishing house of Brno University of Technology VUTIUM, 2014. 396 s. Dostupné z: https://www.fit.vut.cz/research/publication/10567. ISBN 978-80-214-3766-1.

[16] OSSEC, P. T. *Ossec Documentation* [online]. [cit. 2019-12-5]. Dostupné z: https://www.ossec.net/docs/.

[17] Paxson, V. Bro: a system for detecting network intruders in real-time. *The International Journal of Computer and Telecommunications Networking.* Elsevier. 1999, roč. 31, 23-24, s. 2435–2463.

[18] PROOFPOINT. *Emerging Threats Open Ruleset* [online]. [cit. 2020-05-12]. Dostupné z: https://rules.emergingthreats.net/.

[19] Roesch, M. a Green, C. *SNORT Users Manual* [online]. [cit. 2019-12-5]. Dostupné z: http://manual-snort-org.s3-website-us-east-1.amazonaws.com.

[20] Černý, M. *Analýza nedostatků IDS systémů.* Brno, CZ, 2009. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Dostupné z: https://www.fit.vut.cz/study/thesis/7839/.

[21] Šimunek, J. *Bezpečný upgrade firmware embedded platformy.* 2018. Bachelor's thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum.

[22] ZEEK. *Zeek Manual* [online]. [cit. 2019-12-5]. Dostupné z: https://docs.zeek.org/en/master/.

Príloha A

Obsah priloženého pamäťového média

```
/
├── doc/
├── suricata/
├── testing/
│   ├── automatedTests/
│   ├── csvs/
│   ├── graphs/
│   └── parsingScripts/
├── manual.md
├── README.md
└── xmjaso00_bp.pdf
```