FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Adaptivní mitigace DDoS útoků na základě online analýzy
Student:	Bc. Pavel Šiška
Vedoucí:	Ing. Tomáš Čejka, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Seznamte se s problematikou paketové analýzy síťového provozu a mitigací DDoS útoků [1,2,3].
Navrhněte algoritmus a vhodné datové struktury pro online zpracování síťového provozu s cílem odhadnout strukturu provozu, který útok tvoří, a vytvoření optimálních mitigačních pravidel pro eliminaci vybraného typu DDoS útoků.
Implementujte softwarový prototyp s použitím vhodných existujících efektivních metod (jako jsou např. sketch nebo bloom filter).
Vyhodnoťte výkonnost a nároky na zdroje u vytvořeného prototypu.
Zhodnoťte vhodnost prototypu pro nasazení v reálném prostředí lokálních nebo páteřních síťových infrastruktur a případně navrhněte, jakým způsobem tento prototyp využít k obraně komplexních síťových infrastruktur.

Seznam odborné literatury

[1] T. Jánský, T. Čejka, M. Žádník, V. Bartoš, "Augmented DDoS Mitigation with Reputation Scores," in Proceedings of the 13th International Conference on Availability, Reliability and Security, New York, NY, USA, 2018, pp. 54:1–54:7.
[2] F. Křesťan: Automatický odhad parametrů pro mitigaci DDoS útoků, Master thesis, Faculty of information technology, CTU in Prague, 2019.
[3] M. Kuka, K. Vojanec, J. Kučera, P. Benáček: Accelerated DDoS Attacks Mitigation using Programmable Data Plane. In: ANCS '19: Symposium on Architectures for Networking and Communications Systems, Cambridge, United Kingdom, 2019. ISBN 978-1-72814-387-3.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

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

V Praze dne 19. června 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Adaptivní mitigace DDoS útoků na základě online analýzy

Bc. Pavel Šiška

Katedra informační bezpečnosti
Vedoucí práce: Ing. Tomáš Čejka, Ph.D.

7. ledna 2021

Poděkování

Rád bych tímto poděkoval svému vedoucímu Ing. Tomáši Čejkovi, Ph.D. za vedení a podporu během celé doby vzniku této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 7. ledna 2021 . . . . . .

Odkaz na tuto práci

Šiška, Pavel. *Adaptivní mitigace DDoS útoků na základě online analýzy.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá návrhem a implementací nástroje pro online paketovou analýzu síťového provozu. Cílem práce je poskytnout síťovému administrátorovi informace, na základě kterých dokáže nastavit obranné mechanismy pro mitigaci DDoS útoků. Nástroj poskytuje přehled o aktuální struktuře síťového provozu a na základě charakteristik volumetrických DDoS útoků dokáže probíhající útok v síťovém provozu identifikovat a doporučit taková mitigační pravidla, která útok potlačí. Nástroj pro ukládání dat k analýze využívá speciální pravděpodobnostní datové struktury zvané sketche, které dokáží efektivně uchovávat velké množství dat s nízkou paměťovou náročností. Výkonnost a funkčnost nástroje byla otestována v laboratorních podmínkách nad zkušebními daty při rychlostech až 100 Gb/s.

Klíčová slova DDoS, analýza síťového provozu, mitigace, sketch, heavy hitters

Abstract

This thesis deals with design and implementation of the tool for online packet analysis of network traffic. Main goal is to provide necessarily informations for administrator to ensure, that he can set defence mechanisms for mitigation

of DDoS attacks. Tool provides overview of actual structure of the network traffic. It can also identify and recommend mitigation rules to suppress DDoS attack, based on characteristics of volumetric DDoS attacks. Tool for saving data for analysis is using special probability data structures, called sketch, which can effectively store great amount of data with low memory requirements. Performance and functionality of the tool was tested in lab over test data with speed reaching up to 100 Gb/s.

Keywords DDoS, network traffic analysis, mitigation, sketch, heavy hitters

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Monitorování a analýza síťového provozu je nezbytnou součástí každé dobře fungující infrastruktury. Sledování provozu může vést ke zlepšování výkonu, stability a především bezpečnosti sítě. Jednou z mnoha aktuálních bezpečnostních hrozeb jsou útoky typu DoS, případně jejich distribuovaná a také častější varianta DDoS (Distributed Denial of Service). Cílem těchto útoků je snaha o znepřístupnění určitého systému legitimním uživatelům. Útoků typu DDoS existuje celá řada a požadovaného cíle dosahují různými přístupy. Například TCP SYN flood útoky cílí na vyčerpání zdrojů cílového serveru, zatímco volumetrické amplifikační DDoS útoky cílí na zahlcení kapacity síťové linky připojené koncové sítě.

Volumetrické DDoS útoky představují pro síťovou infrastrukturu poměrně závažný problém tím, že zahltí kapacitu hraničních síťových linek, u kterých je v dnešní době běžné, že pracují na vysokých rychlostech - desítky až stovky Gb/s. V situaci probíhajícího útoku tak přichází i související problém s přetížením nástrojů monitorovací infrastruktury, které jsou typicky založené na bázi analýzy flow záznamů. V takovém stavu je obtížné získávat informace pro správné nastavení obranných mechanismů jako jsou například filtry pro zahazování škodlivého provozu.

Na základě tohoto pozorování a identifikovaných problémů byl naplánován ve spolupráci se specialisty ze sdružení CESNET vývoj nástroje, který bude pracovat na bázi online analýzy síťového provozu na paketové úrovni a jeho cílem bude umožnit administrátorovi sítě identifikovat strukturu DDoS útoku a doporučit taková mitigační pravidla, která útok potlačí, a to i při vysokých rychlostech dosahujících 100 Gb/s, kde ostatní nástroje mohou selhávat.

Navržený algoritmus je založený na existujících datových strukturách, tzv. sketches. Tento algoritmus sleduje zdroje a cíle provozu podle informací v hlavičkách paketů a pomocí vhodné reprezentace se snaží identifikovat původce útoku. Algoritmus byl testován pomocí generovaného síťového provozu při různých rychlostech, tzn. vytížení linky. Experimenty se zaměřovaly

na použitelnost tohoto přístupu na vybraných extrémních případech struktury síťového provozu. K provedení těchto experimentů vznikl prototyp algoritmu v jazyce C++.

Práce je rozdělena do několika částí, logicky strukturovaných podle jednotlivých etap řešení zadaného problému. Kapitola 2 blíže představuje problematiku útoků typu odepření služby (DoS), včetně popisu vybraných typů útoků a aktuálních trendů. Kapitola 3 představuje zařízení nazvané DDoS Protector, které je vyvíjené sdružením CESNET a slouží pro mitigaci především volumetrických DDoS útoků, a možná spolupráce s tímto zařízením je pro vytvářený nástroj důležitá. Kapitola 4 popisuje odlišné přístupy k analýze síťových dat a kapitola 5 pak představuje pravděpodobnostní datovou strukturu zvanou sketch a problematiku hledání nejvíce zastoupených prvků. Kapitola 6 již spadá do praktické části práce a detailně popisuje návrh řešení vytvářeného nástroje. V kapitole 7 jsou popsány implementační detaily nástroje, včetně řešení nalezených problémů. Vyhodnocení výsledků testování, které zahrnují měření datové propustnosti, zhodnocení paměťových nároků a ověření funkčnosti jsou popsány v kapitole 8. Závěrečná kapitola se zabývá celkovým zhodnocením práce a možností budoucího rozšíření nástroje.

Kapitola 1

Cíl práce

Práce si klade za cíl vytvořit prototyp nástroje pracujícího na bázi online analýzy paketů, který poskytne administrátorovi sítě přehled o aktuálního struktuře síťového provozu a v případě aktivního volumetrického DDoS útoku dokáže doporučit mitigační pravidla, která tento útok potlačí.

1.1 Motivace

Motivací ke vzniku této práce byl bezpečnostní incident, který se stal v síti spravované sdružením CESNET[1]. Přesný popis události není veřejný a tak zde bude událost popsána bez bližších detailů. Jednalo se o amplifikační DDoS útok, který využíval tehdy nově objevené chyby v implementaci OpenVPN[2], která umožnila dosáhnout DDoS útoku amplifikačního faktoru 1:60. Tento útok byl tak silný, že dokázal vyřadit nejen cíl útoku, ale i monitorovací infrastrukturu sítě, která je založena na bázi analýzy síťových toků. I přestože po zpětné analýze bylo zjištěno, že pravidlo pro zablokování DDoS útoku bylo vcelku primitivní, v době útoku nebyly administrátoři sítě toto pravidlo schopni zjistit, protože bez dat z monitoringu sítě byli slepí. Proto vznikl požadavek na vytvoření prototypu nástroje, který by pracoval na bázi online analýzy paketů, a který by dokázal v případě potřeby dočasně nahradit monitorování sítě založené na bázi síťových toků a to s cílem určit charakter DDoS útoku a případně vytvořit takové mitigační pravidlo, které by útok potlačilo.

[1] https://www.cesnet.cz/
[2] https://openvpn.net/

Kapitola 2

Útoky typu odepření služby

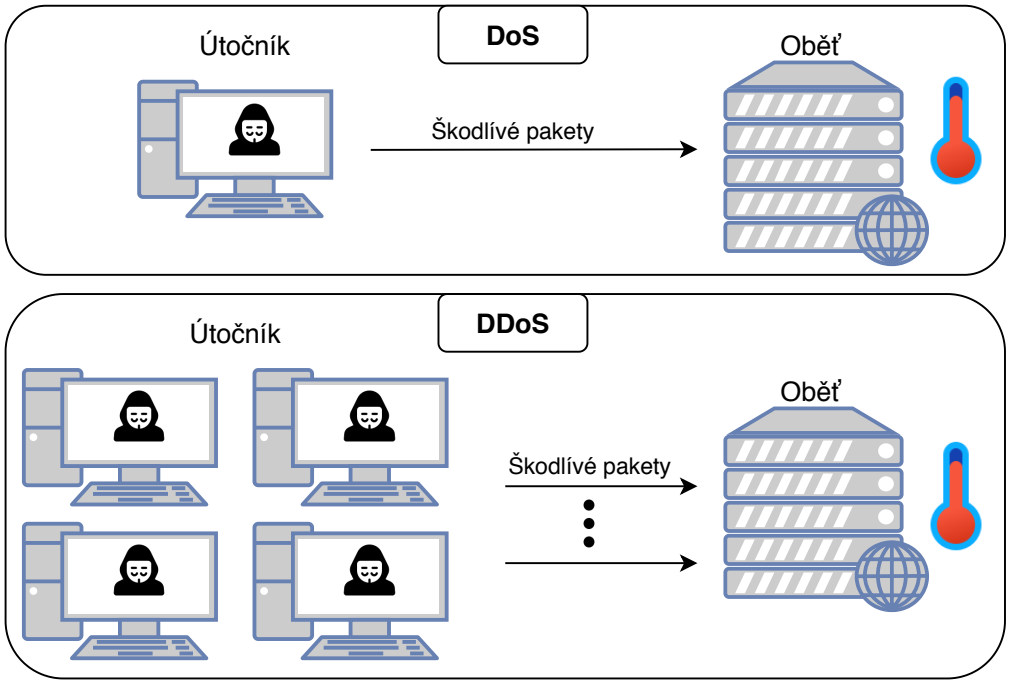
Cílem této kapitoly je seznámit čtenáře s definicí, historií a typy Denial of Service (DoS) útoků.

2.1 Definice

Útoky typu odepření služby, neboli Denial of Service (DoS), spadají do skupiny kybernetických útoků, které mají za cíl zabránit legitimnímu uživateli v používání specifického síťového zdroje, jako je např. webová stránka, síťová služba nebo počítačový systém [1]. Útoky se zaměřují pouze na zamezení přístupu legitimních uživatelů k těmto zdrojům, nikoli na získávání či poškozování dat. Efektu odepření služby lze dosáhnout dvěma způsoby [2]. Prvním způsobem je odeslání jednoho nebo více specificky vytvořených paketů, které zneužívají vlastností jednotlivých síťových protokolů či chyby v programu běžícím na cílové oběti. Druhým způsobem je odesílání velkého množství paketů útočníkem, tzv. záplavy, která má za cíl, stejně jako první metoda, vyčerpání některého z omezených zdrojů oběti, jako je kapacita přenosového pásma, paměť, CPU, kapacita disku, atd. [3]. Služba typicky po vyčerpání svých zdrojů není schopna vyřizovat příchozí zprávy a stane se pro legitimní uživatele nedostupnou.

2.2 Distribuovaný útok na odepření služby

Útočníci často při svých útocích využívají distribuovanou variantu DoS útoku, tzv. Distributed Denial of Service (DDoS). Rozdíl mezi DoS a DDoS útokem je v počtu útočících zařízení, ze kterých je útok veden. U DoS útoků je útočící zařízení pouze jedno, zatímco u DDoS útoků jsou útočící zařízení minimálně dvě. Jelikož je DoS útok veden jen z jednoho zařízení, typicky není příliš silný a jeho odhalení není příliš složité. DDoS útok poskytuje útočníkovi oproti DoS útoku množství výhod, nejenom že s více útočícími zařízeními je útočník



Obrázek 2.1: Porovnání DoS a DDoS útoku.

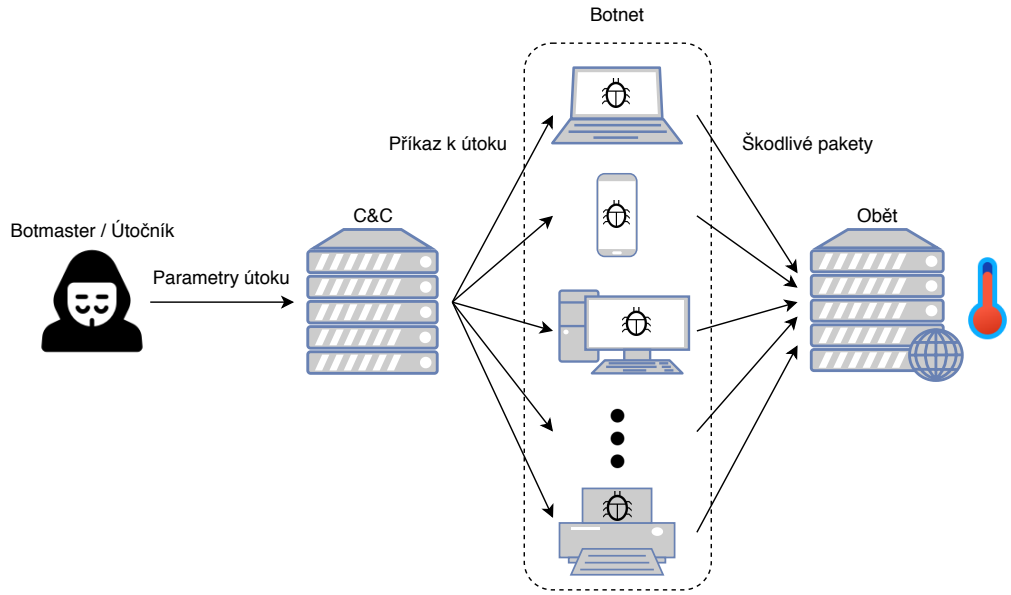
schopen znásobit objem a intenzitu odesílaných paketů, ale také zkomplikuje oběti možnosti, jak se takovému útoku účinně bránit, a to hlavně z nutnosti identifikace velkého množství zdrojů útoku, které je nutné blokovat k potlačení útoku. Mnohdy může být při velkém množství zdrojových zařízení odlišení legitimního uživatele od útočníka velice obtížné. Rozdíl mezi DoS a DDoS útokem ukazuje obrázek 2.1.

2.3 Botnet

K provedení velkých DDoS útoků se používají stovky až tisíce zařízení. Pořizování a údržba takového to počtu zařízení by byla pro útočníky příliš nákladná a kvůli centralizaci i neefektivní. Proto útočníci k provedení velkých DDoS útoků používají skupinu zařízení, která se nazývá botnet.

Botnet označuje skupinu zařízení, která jsou infikována malwarem (škodlivý kód), který zajišťuje, že jsou zařízení pod kontrolou botmastera (útočníka) [4]. Botnet pak pod kontrolou útočníka provádí nežádoucí činnost, jako je těžba kryptoměn, rozesílání spamu, DDoS útoky a podobně.

Zařízení mohou být infikovaná prostřednictvím mnoha různých kanálů. Mezi způsoby infekce patří využití zranitelnosti webových stránek či aplikací, trojské koně nebo také prolomení slabé autentizace za účelem získání vzdáleného přístupu [5]. Infikovaná zařízení se nazývají zombie nebo také boti.



Obrázek 2.2: Organizace DDoS útoku s pomocí botnetu.

Botnet má celou řadu topologií, které blíže popisuje text [5], pro ukázku zde bude popsána topologie Klient-server. V této topologii existuje jeden (nebo malé množství) centralizovaný server, který se nazývá Command and Control server (C&C). Každé nově infikované zařízení se u takového to C&C serveru zaregistruje a následně čeká na příkaz, který mu určí, co má dělat. Než ale bot příkaz obdrží, přepne se do tzv. vyčkávacího režimu, ve kterém se snaží nevzbuzovat zbytečnou pozornost. Pro komunikace mezi C&C serverem a boty se používají většinou běžné komunikační kanály jako je Internet Relay Chat (IRC), ale i složitější komunikační protokoly, které podporují vyšší formu zabezpečení. Topologie Klient-server poskytuje několik výhod, stejně tak i nevýhod. Mezi výhody patří její jednoduchost a nízká latence při propagování příkazů. Hlavní nevýhodou této topologie je možné selhání nebo zablokování C&C serveru, který způsobí, že se boti nemohou zaregistrovat, ani obdržet nové příkazy. Ukázka této topologie při provádění DDoS útoku je zobrazena na obrázku 2.2.

2.4 Motivace

I přesto, že se obětí (D)DoS útoku může stát kdokoli s připojením k internetu, útočníci si obvykle vybírají oběti z následujících důvodů [6][7]:

- **Finanční nebo ekonomické zisky:** Jedním ze způsobů, jak DoS útoky zpeněžit, nebo se na nich obohatit je vydírání. Útočník si vybere oběť,

na kterou následně provede útok a za jeho zastavení požaduje po oběti výkupné. Dalším způsobem je, když je útočníkovi zaplaceno třetí stranou za provedení útoku na vybraný cíl.

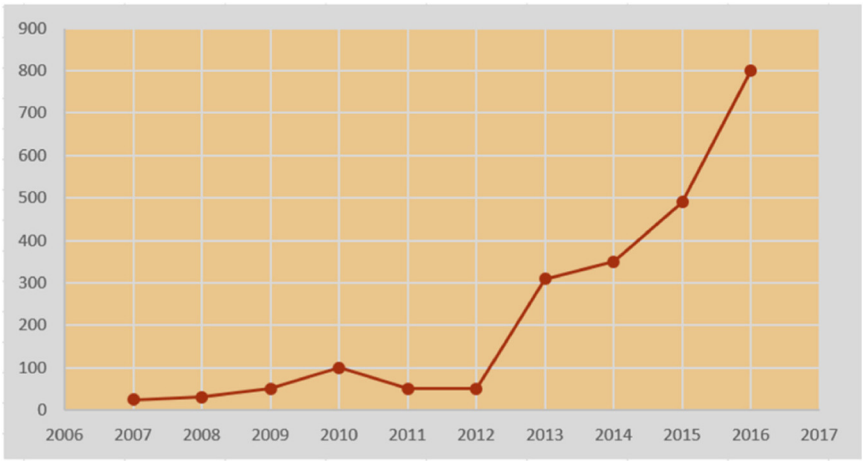
- **Pomsta:** Útočník se snaží získat odplatu na jednotlivci, organizaci nebo firmě. Typicky útok cílí na vysoké školy, soudy, orgány činné v trestním řízení či novináře, s cílem způsobit škodu za vnímanou křivdu.
- **Ideologické neshody:** Do této kategorie spadají útoky, kdy se útočník snaží DoS útokem vyjádřit jistou formu nesouhlasu. Většinou se jedná o politicky motivované útoky na weby vlády nebo politických stran. Těmto politicky motivovaným útočníkům se říká „hactivisté“ a mezi nejznámější hactivistickou skupinu patří Anonymous.
- **Seberealizace:** Často může být hnacím motorem DoS útoků potřeba někomu z komunity ukázat, že na to daný jedinec má a zvýšit si tak společenský kredit, případně sám si dokázat, že je toho schopen.
- **Kybernetická válka:** Typicky organizované státní nebo teroristické skupiny, které se snaží směrovat útoky na systémy kritické pro fungování státu a jeho infrastrukturu.

2.5 Historie a současné trendy

První provedený DDoS útok se stal podle [8] v roce 1996. Útok byl tehdy směrován na společnost Panix, nejstaršího internetového poskytovatele (ISP) v New Yorku. Do většího povědomí se ale DDoS útoky dostaly v roce 2000, kdy teprve 15letý hacker s přezdívkou *Mafiaboy* dokázal znepřístupnit webové stránky několika velkým společnostem, konkrétně se jednalo o Amazon, eBay, Dell, CNN a FIFA [9]. Tento útok měl zničující následky a způsobil chaos na akciovém trhu. Mafiaboy, který byl později odhalen a odsouzen na 8 měsíců v detenčním ústavu pro mladistvé [10] využil ke svému DDoS útoku zkompromitované servery několika univerzit. Na základě tohoto útoku vzniklo několik dodnes platných zákonů o počítačové kriminalitě.

O dalším zvýraznění DDoS útoků ve veřejném prostoru se postarala hacktivistická skupina *Anonymous* v roce 2010, kdy provedla operaci nazvanou *Payback* jako podporu Julianu Assangovi v kauze *WikiLeaks*. V rámci této operace se skupině podařilo znepřístupnit webové stránky firmám jako jsou PayPal, MasterCard, Visa a mnoho dalších a to z důvodu, že blokovali platby, které směřovali na podporu webu WikiLeaks [11]. Skupina Anonymous působí také v České republice, kde podnikla již celou řadu DDoS útoků, např. na webové stránky OSA (Ochranný svaz autorský) za vybírání autorských poplatků [12], nebo na webové stránky politických stran [13].

Síla a počet DDoS útoků se v posledních letech stále zvětšuje. Tento trend potvrzuje i článek [14], který popisuje trend růstu maximální síly DDoS



Obrázek 2.3: Trend růstu maximální síly DDoS útoku v Gb/s v letech 2007–2016 na sítích spravovaných firmou Arbor. Zdroj: [14]

útoků v letech 2007–2016 na datech z výroční zprávy společnosti Arbor za rok 2017 [15]. Tento trend je zobrazen na obrázku 2.3. V říjnu roku 2016 byly ale na sítích, které nespadají pod tuto společnost provedeny ještě silnější útoky. Útoky byly provedeny pomocí botnetu *Mirai*, který se skládá z několika set tisíc Internet of Things (IoT) zařízení jako jsou kamery, chytré televize, tiskárny a další věci chytré domácnosti připojené k internetu. Tyto útoky dosahovaly síly více než 1 Tbps a postihly webové stránky jako jsou Netflix, Airbnb, Redit, Github a další [9]. Největší dosud zaznamenaný DDoS útok se stal v únoru roku 2020 a jeho síla dosahovala hranice 2.3 Tbps [16]. Útok směřoval na Amazon Web Service a využíval techniky odrazu ke znásobení své síly. Technika odrazu bude popsána dále v práci.

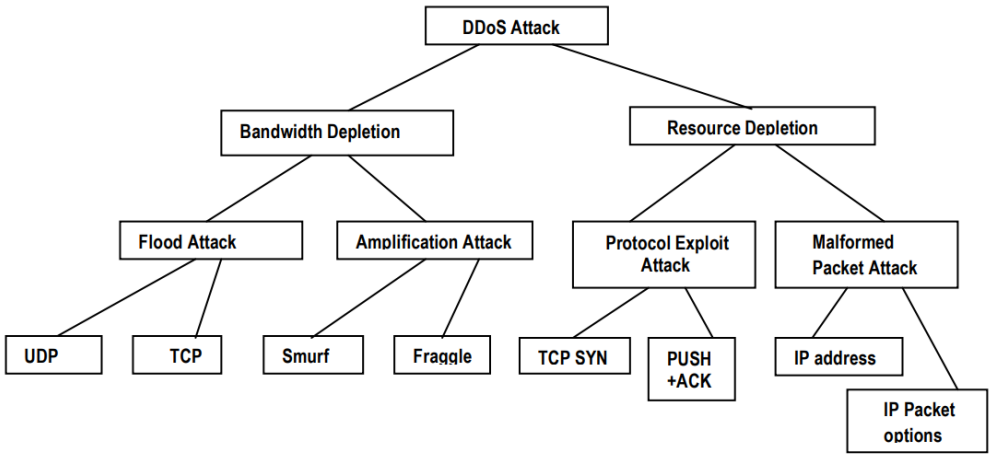
Rostoucí popularitě DDoS útoků přispívá i to, že si je lze velice snadno objednat na internetu. Vlastníci botnetů nabízejí svá zkompromitovaná zařízení k pronájmu, případně přímo poskytují provedení DDoS útoku na vybraný cíl. Stačí tedy jen zadat cíl útoku, parametry jako je síla, doba trvání či typ útoku, následně zaplatit a o zbytek se postará již někdo jiný. Ceník pro vybrané parametry útoku uvedený na stránce, která tyto služby nabízí zobrazuje tabulka 2.1.

2.6 Dělení DDoS útoků

Existuje mnoho odlišných rozdělení DDoS útoků. Podle [3] (obrázek 2.4) lze DoS útoky rozděli do dvou základních kategorií:

Cena [$]	Doba trvání [min]	Síla [Gb/s]
9.99	5	15
25.99	30	15
34.99	60	15
179.99	10	60
649.99	60	60
1999.99	60	225

Tabulka 2.1: Ceník DDoS útoků uvedený na stránkách [17].

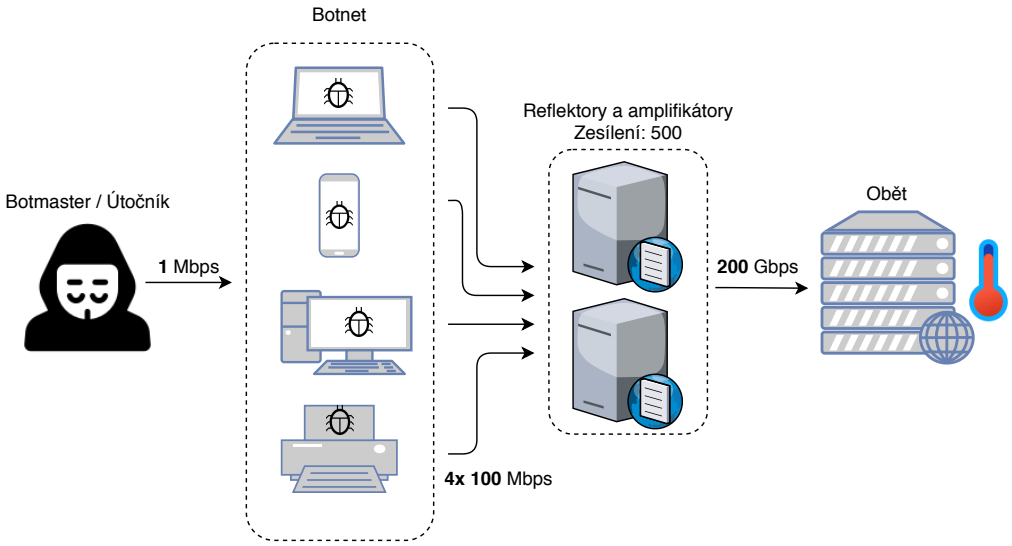


Obrázek 2.4: Klasifikace DoS útoků. Zdroj: [3]

Útoky cílené na vyčerpání konektivity

Útoky, které cílí na konektivitu se snaží o zahlcení oběti nebo jiného síťového zařízení, přes které je oběť připojena, a to zasíláním velkého množství provozu. Tyto útoky se nazývají také jako volumetrické útoky. Pokud je síť dobře dimenzovaná a velké množství paketů zvládne zpracovat, stává se úzkým hrdlem samotné síťové zařízení oběti, které je jako jediné zasaženo. Je-li ale útok dostatečně silný, může se stát, že jsou zahlceny i směrovače po cestě k oběti, což vede k ovlivnění celé podsítě, kterou směrovač spravuje. Tyto útoky lze dále rozdělit na záplavové (flood) a amplifikační útoky. Tato práce se zaměřuje právě na tuto kategorii útoků.

- *Záplavové útoky* posílají velké množství typicky nesmyslných paketů k vyčerpání konektivity oběti. Pakety legitimních uživatelů v tomto velkém množství příchozích paketů tvoří menšinu a proto jsou s vysokou pravděpodobností na zařízení, které se stalo úzkým hrdlem zahozeny. U těchto útoků je typicky použit botnet, který posílá pakety přímo



Obrázek 2.5: Ukázka amplifikačního DDoS útoku.

k oběti. Provedení tohoto typu útoku je velmi primitivní, ale i přesto dokáže být velice účinný. Nejznámějšími zástupci záplavových útoků jsou UDP, TCP a ICMP flood, kteří jsou popsány v sekci 2.7 odkaz.

- *Amplifikační útoky* jsou specifické tím, že zneužívají jiná (nezkompromitovaná) síťová zařízení k provedení útoku. Útočník u tohoto typu útoku využívá tzv. reflektorů k amplifikaci (zesílení) útoku. Reflektor je síťové zařízení, které odpovídá na příchozí pakety podle pravidel použitého protokolu. Útočník posílá zprávy s takovými dotazy, na které reflektor dokáže odpovědět s mnohonásobně větší zprávou, než byla velikost původního dotazu a tím dojde k amplifikaci. Jako příklad používaného reflektoru lze uvést otevřený DNS resolver. DDoS útoky, které využívají odrazu se nazývají Distributed Reflected Denial of Service (DRDoS). Ukázku takového útoku zobrazuje obrázek 2.5, na kterém je vidět, že i útočník s relativně slabým připojením dokáže vygenerovat obrovský DDoS útok.

  Reflektor posílá odpověď na zdrojovou adresu příchozího paketu, aby se pakety nevracely zpět k útočníkovi, ale směřovali k oběti útoků se používá technika podvržení zdrojové IP adresy (*IP address spoofing*). Útočník vloží do odesílaného dotazu jako zdrojovou adresu IP adresu oběti, reflektor pak odpoví na útočníkem zfalšovanou IP adresu. Pro trasování útočníků využívajících podvržené IP adresy a reflektoru je nutná spolupráce mezi správcem reflektoru a správcem oběti, která celý proces značně ztěžuje. Další možností jak lze amplifikační útok provést je zaslat zprávu jako broadcast s podvrženou zdrojovou IP adresou. Router

broadcast zprávu rozešle na všechna zařízení ve své síti a tyto zařízení pak odešlou odpověď na podvrženou adresu, tedy na adresu oběti. Typickým příkladem je *Smurf* útok, který využívá protokol ICMP. Odpověď u tohoto útoku sice není větší než dotaz, dochází ale k amplifikaci počtu zpráv, protože na jeden dotaz odpovídá více zařízení v síti.

Útoky cílící na vyčerpání zdrojů

Útoky, které cílí na vyčerpání zdrojů mají typicky menší objem provozu než útoky cílící na konektivitu. Tento typ útoků se zaměřuje na hardwarové a softwarové prostředky jako výpočetní výkon, paměť nebo různé limity, ať už systémové či uživatelem definované. Útoky lze rozdělit do dvou podkategorií, a to protokolové útoky a útoky využívající poškozeného či speciálně modifikovaného paketu.

- *Protokolové útoky* využívají specifické funkcionality nebo slabin v návrhu jednotlivých protokolů. Protokol TCP je pro tyto útoky obzvlášť náchylný a je základem pro nejčastější typ útoku, a to TCP SYN flood, který zneužívá procesu navazování spojení u TCP protokolu. TCP SYN flood útok je blíže popsán dále v textu.

- *Útok poškozenými či upravenými pakety* cílí na chyby v implementaci protokolů a aplikací. Útočník posílá nesprávně vytvořené pakety do systému oběti, aby způsobil selhání systému nebo vyčerpání jeho zdrojů. Při jednom z těchto útoků útočník posílá pakety, které mají stejnou zdrojovou a cílovou IP adresu. To může zmást operační systém a vést až k jeho selhání. Existuje celá řada dalších útoků tohoto typu, ale tato práce se těmito útoky dále nezabývá.

2.7 Vybrané typy útoků

Tato podkapitola detailněji popisuje vybrané typy DoS útoků.

- **UDP flood útok** využívá protokolu UDP, který má na starosti komunikaci v síti, podobně jako TCP. Protokol UDP je bezstavový a ke komunikaci nepotřebuje navazovat spojení jak je tomu u protokolu TCP. Díky tomu je oproti TCP rychlejší, ale na druhou stranu náchylný k útoku UDP flood. Tento útok funguje na jednoduchém principu a je snadný na provedení. Útočník posílá velké množství UDP paketů na náhodné cílové porty systému oběti. Systém po přijetí paketu nejprve zkontroluje, zda je k danému cílovému portu připojena nějaká z jeho aplikací. Pokud zjistí, že žádná aplikace na daném portu nenaslouchá, musí systém podle pravidel komunikace odpovědět na každý takovýto paket informací, že je cílová destinace nedostupná. Tato odpověď se posílá ICMP paketem se zprávou *Destination Unreachable*. Snahou útočníka je tedy

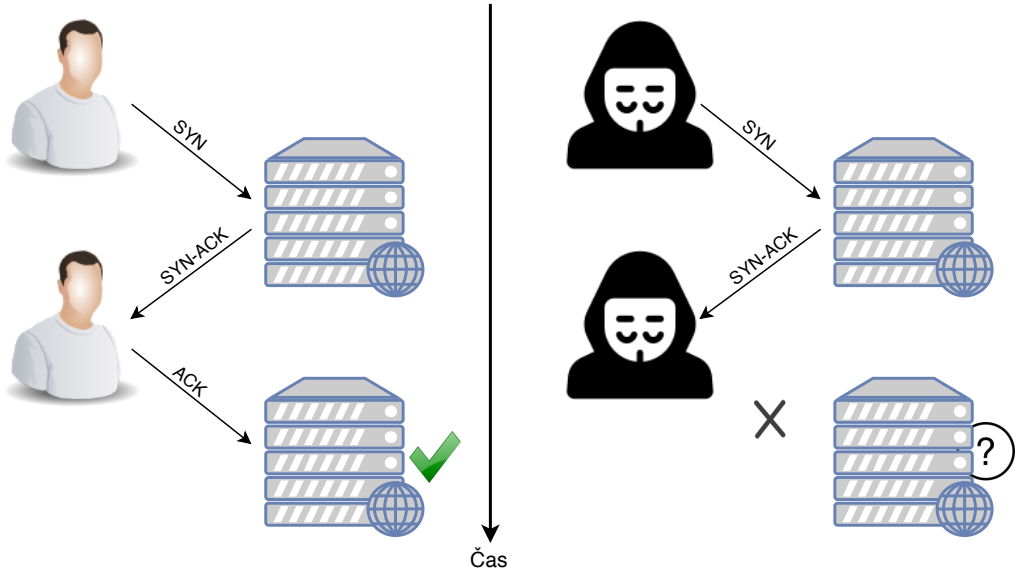
zahltit systém oběti velkým množstvím UDP paketů s nesmyslnými cílovými porty, aby byl systém nucen na každou takovou zprávu odpovědět ICMP zprávou o nedostupnosti. Pokud je počet UDP paketů dostatečný, dojde k zahlcení systému oběti, nebo kapacity linky, přes kterou je oběť připojena a jakákoli jiná komunikace se stane nemožnou. Útočník k takovémuto útoku využívá typicky botnet, který posílá UDP pakety s podvrženou zdrojovou IP adresou. Ztíží tak odhalení zdrojů útoku, ale zároveň zabrání tomu, aby se ICMP zprávy o nedostupnosti nevraceli k útočícím zařízením a nedošlo k jejich přetížení [18].

- **ICMP flood útok** využívá protokolu ICMP, který je jedním z hlavních internetových protokolů a slouží k odesílání chybových hlášení či diagnostik. Útočník využívá zprávy ICMP *Echo Request*, která se běžně používá pro účely zjištění dostupnosti cílového zařízení. Princip útoků spočívá v tom, že útočník posílá velké množství ICMP paketů typu *Echo Request* do systému oběti, na které dle pravidel komunikace oběť odpovídá ICMP zprávou *Echo Reply* o stejné velikosti jako byla přijatá zpráva. Cílem útoku je zahltit linku oběti, a to jak zprávou *Echo Request* tak i zprávou *Echo Reply*. Při tomto útoku je ale zahlcována i linka útočníka, a proto pro úspěšný útok je nezbytné, aby byla linka útočníka rychlejší než linka oběti. Možností jak tuto podmínku obejít je podvržení zdrojové IP adresy u zprávy *Echo Request*, tím se útočník vyhne příjmu zpráv *Echo Reply* a jeho zařízení se tak nezahltí [19].

- **TCP SYN flood útok** využívá protokolu TCP, který má na starosti komunikaci v síti, podobně jako protokol UDP. TCP je stavový protokol, který před zahájením samotné komunikace potřebuje navázat spojení. Spojení se navazuje pomocí tzv. *three-way handshake* procesu, který se snaží ověřit, zda obě strany doopravdy stojí o spojení. Princip útoku spočívá v tom, že útočník zneužije proces navazování spojení k vyčerpání zdrojů systému oběti. Ukázka jak legitimní navazování spojení vypadá je ukázáno v levé části obrázku 2.6.

  Legitimní proces o navázání spojení funguje tak, že klient, jakožto strana zahajující spojení, odešle na server paket s nastaveným příznakem SYN. Server si tento požadavek uloží do tabulky tzv. polootevřených spojení a odpoví klientovi paketem s příznakem SYN-ACK. Klient po přijetí SYN-ACK paketu zašle na server závěrečné potvrzení, a to paket s příznakem ACK. Server podle informací v ACK paketu vyhledá v polootevřených spojení původní požadavek a přesune toto spojení do otevřených, což dokončí celý proces a spojení je navázáno.

  Útočník při TCP SYN flood útoku posílá velké množství paketů s příznakem SYN, jako by chtěl navázat spoustu nových spojení, ale už serveru neodpovídá závěrečným paketem s příznakem ACK, jak je zobrazeno v pravé části obrázku 2.6. Server je tak nucen udržovat si velké množství



Obrázek 2.6: Porovnání procesu navazování spojení legitimního uživatele a útočníka.

spojení v polootevřeném režimu, dokud jim nevyprší časovač. Počet takto polootevřených spojení je omezen limitem a po jeho překročení server začne všechny nově příchozí požadavky o spojení odmítat. Pro legitimního uživatele se tak stane nedostupným [20]. Útočník při tomto útoku často podvrhne zdrojovou IP adresu paketu s příznakem SYN, aby se mu nevraceli odpovědi od serveru. Server tedy odpoví na podvrženou zdrojovou IP adresu, která ale neočekává navázání spojení a tak nemá potřebu odpovědět.

- **DNS amplifikační útok** zneužívá služeb systému doménových jmen DNS, konkrétně pak otevřených rekurzivních DNS name serverů, které poskytují své služby i mimo svou síť. Útočník posílá na tyto DNS servery dotaz *ANY*, což je požadavek na kompletní záznamy k určitému doménovému jménu. Zároveň útočník podvrhne zdrojovou IP adresu dotazu, aby DNS server svou odpověď směřoval na zařízení oběti. Velikost odpovědi, kterou může DNS server na dotaz *ANY* poskytnou, může být až několikanásobně větší, než dotaz samotný. Útočník se obvykle dotazuje na takovou doménu, která poskytuje co největší faktor zesílení.

- **NTP amplifikační útok** využívá serverů systému pro synchronizaci času NTP (Network Time Protocol). Útočník posílá na NTP server požadavky s podvrženou zdrojovou adresou, která je zároveň adresou cílové oběti. K amplifikaci se zneužívá příkaz *monlist*, který slouží k monitorování provozu na serveru. Tímto příkazem je možné získat až po-

sledních 600 adres, které se k danému NTP serveru připojily. Tímto příkazem je možné amplifikovat původní dotaz v rozmezí 20:1 až 200:1 [21], což je pro potřeby amplifikačního útoku ideální. K útoku lze využít pouze linuxové NTP servery, které jsou verze nižší než 4.2.8 a mají povolený příkaz *monlist* v konfiguraci. Od vyšších verzí NTP serverů byl příkaz *monlist* zcela odstraněn.

- **SNMP amplifikační útok** využívá protokolu SNMP (Simple Network Management Protocol), který se používá na sběr statistik, testování stavu zařízení a případnému managementu síťových zařízení a serverů. Na jedné straně komunikace je monitorovací zařízení, které se nazývá SNMP manager, na druhé straně jsou monitorovaná zařízení, kterým se říká SNMP agenti. Útok je veden tak, že útočník posílá SNMP agentům dotazy typu *SNMP GET* s podrženou zdrojovou IP adresou, která je zároveň adresou oběti. Odpověď agentů na dotaz může být podle [22] až 1700x větší než samotný dotaz. Agenti amplifikovanou odpověď odešlou na podvrženou adresu, tedy adresu oběti. Tento typ útoku je možné provádět pouze u SNMP verze jedna a dva. Od verze tři již tento útok není možné provádět, protože komunikace vyžaduje ověření prostřednictvím jména a hesla a zároveň se používá šifrované spojení.

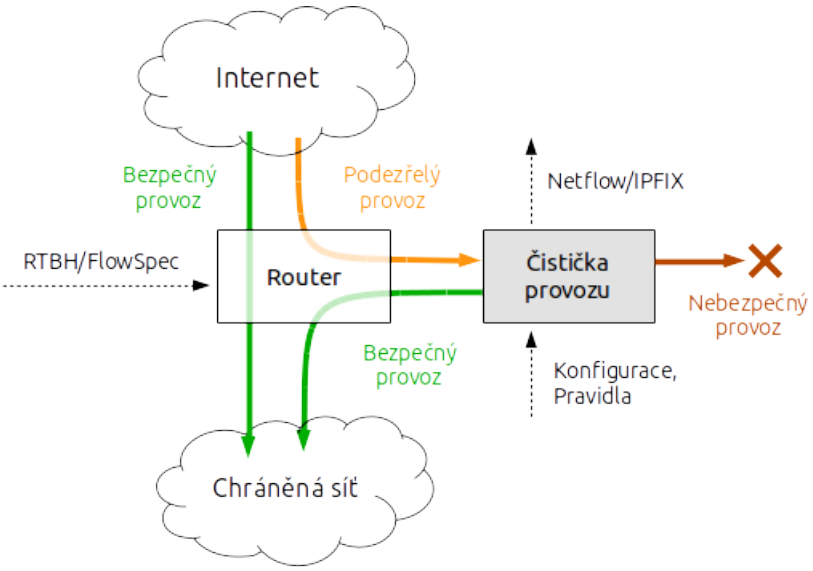
Kapitola 3

DDoS Protector

Cílem této kapitoly je seznámit čtenáře se zařízením pro ochranu před DDoS útoky, se kterým je tato práce úzce provázána. Tato kapitola vychází především z uživatelského manuálu zařízení DDoS Protectoru [23], není-li uvedeno jinak.

DDoS Protector je aktivní síťové zařízení, které chrání vybrané sítě před příchozími DDoS útoky. Toto zařízení je vyvíjeno v rámci sdružení právnických osob CESNET. Zařízení je zaměřeno na mitigaci volumetrických DDoS útoků, ale i na protokolový TCP SYN flood útok. Amplifikačním útokům se typicky oběť nemůže sama bránit, protože tyto útoky cílí na vyčerpání kapacity linky, kterou je oběť připojena, a proto ochranu musí zajistit poskytoval připojení, jehož kapacity jsou většinou dostatečné, aby útok zvládnul zpracovat. Proto je předpokládané zapojení DDoS Protectoru do páteřní sítě poskytovatele připojení, kde bude chránit oběti DDoS útoků v připojených podsítích. Ukázka zapojení DDoS Protectoru v síti je zobrazena na obrázku 3.1. Na tomto obrázku je DDoS Protector zapojen v kombinaci se směrovačem, který přeposílá provoz chráněných sítí do DDoS Protectoru. DDoS Protector příchozí provoz analyzuje a pokud není detekován aktivní DDoS útok, tak je provoz zaslán zpět na směrovač, který jej následně směruje do cílové chráněné sítě. Pokud je ale při analýze příchozích dat zjištěn aktivní DDoS útok, jsou pakety útočících zařízení zahozeny. DDoS Protector pak ke směrovači pošle odfiltrovaný provoz o menším objemu, se kterým si cílová chráněná síť poradí. Efekt DDoS útoku je tím pádem eliminován.

DDoS Protector dokáže pracovat na rychlosti 100 Gb/s (148 milionů paketů za sekundu) a skládá se z hardwarové a softwarové části. Hardwarová část je založena na hardwarově akcelerované síťové kartě, která využívá technologii FPGA (Field Programmable Gate Array) ke zpracování vysokorychlostních síťových dat. Hlavním úkolem karty je přeposílání paketů do softwarové části a na výstupní rozhraní. Karta také dokáže na základě instrukcí od softwarové části blokovat část provozu podle specifických pravidel. Softwarová část DDoS protectoru zajišťuje ovládání chování celého zařízení, jako je analýzu paketů,



Obrázek 3.1: Ukázka zapojení DDoS Protectoru v síti. Zdroj: [23]

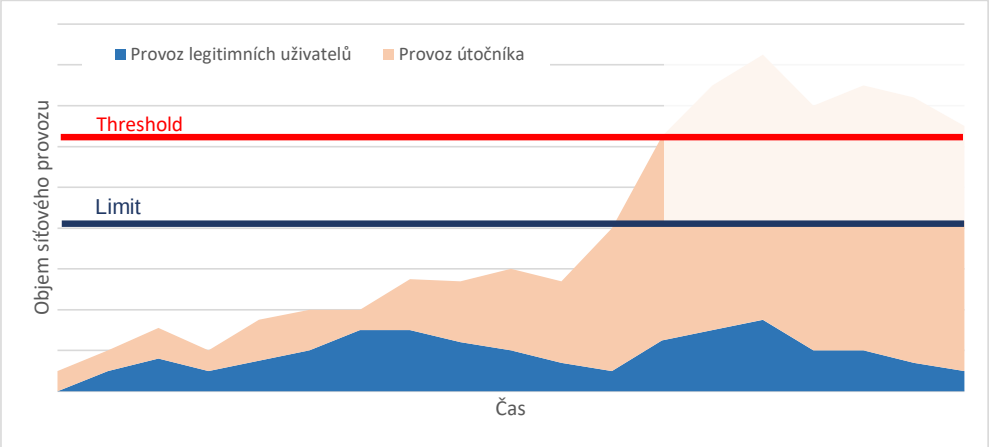
mitigaci útoku, logování atd.

Detekci DDoS útoku DDoS Protector vyhodnocuje na základě pravidel, které nastaví administrátor sítě. Pravidlo je vždy zadáno pro konkrétní cílovou síť a udává jaký typ a objem provozu může do dané sítě směřovat. Každý paket je pak vůči takovýmto pravidlům porovnáván a pokud paket odpovídá danému pravidlu je započítán k danému pravidlu do statistik. Detekce probíhá v periodických časových intervalech označených jako pozorovací okno (observation window).

3.1 Amplifikační útoky

Pro mitigaci amplifikačních DDoS útoků se používá *Amplifikační modul*. Hlavním cílem tohoto modulu je při detekovaném DDoS útoku snížit objem provozu směřující do chráněné sítě na přijatelnou mez. Hranici toho, kdy se jedná o DDoS útok a co je přijatelná mez vyjadřuje proměnná *threshold* a *limit*, která se zadává v bajtech nebo paketech za vteřinu. Překročí-li objem provozu směřující do chráněné sítě hodnotu *threshold*, spustí se mitigační proces, který sníží objem provozu na hranici *limit*. Tento proces je ukázán na obrázku 3.2.

Mitigační proces probíhá tak, že algoritmus seřadí zdrojové IP adresy, které viděl během posledního pozorovacího okna, a to podle toho, kolik objemu provozu poslaly. Následně vybere ty adresy, které svou aktivitou přispěly nejvíce a začne je blokovat. Adres je vybráno k zablokování právě tolik, dokud se nepodaří snížit objem provozu na hranici *limit*. V současné verzi DDoS Protectoru je možno blokovat až 32768 unikátních zdrojových IP adres. Hodnota



Obrázek 3.2: Princip amplifikačního modulu. Zdroj: [24]

threshold a *limit* se vztahují k pravidlům, které zadává administrátor a mají následující formát.

```
<RULE> <DST-NET> <PROTOCOL> <SRC-PORT> <DST-PORT> <FRAGMENTATION>

<TCPFLAGS> <LENGTH> <THRESHOLD> <LIMIT> <WHITELIST>
```

Význam jednotlivých položek je následující:

- `RULE:` Nepovinný textový popis pravidla.
- `DST-NET:` Udává chráněnou cílovou síť ve formátu adresy a prefixu (např. 105.185.42.0/24). Může být zadána jak IPv4 tak i IPv6 adresa. Tato položka je povinná.
- `PROTOCOL:` Udává protokol paketu podle hodnoty položky *Protocol* v IP hlavičce (TCP, UDP, ICMP, ...). Nepovinná položka.
- `SRC-PORT:` Udává hodnotu zdrojového portu paketu. Může být zadán jako menší než, větší než, rozsah nebo přesná hodnota. Nepovinná položka, která je validní jen pro pakety s protokolem obsahujícím porty.
- `DST-PORT:` Udává hodnotu cílového portu paketu. Může být zadán jako menší než, větší než, rozsah nebo přesná hodnota. Nepovinná položka, která je validní jen pro pakety s protokolem obsahujícím porty.
- `FRAGMENTATION:` Nepovinná položka, která určuje fragmentaci paketu a může nabývat následujících hodnot:

  - `YES:` Pouze fragmentované pakety včetně první a poslední části paketu.
  - `NO:` Pouze nefragmentované pakety.
  - `FIRST:` Pouze pakety obsahující první část fragmentu.
  - `LAST:` Pouze pakety obsahující poslední část fragmentu.
  - `MID:` Pouze fragmentované pakety, které nejsou první nebo poslední částí paketu.
  - `NOFIRST:` Všechny fragmentované pakety s výjimkou první části paketu.

- `TCPFLAGS:` Udává TCP příznaky obsažené v paketu. Nepovinná položka, která je validní pouze s protokolem TCP. Může nabývat následujících hodnot:
  - `S:` Příznak SYN.
  - `A:` Příznak ACK.
  - `F:` Příznak FIN.
  - `P:` Příznak PUSH.
  - `R:` Příznak RESET.

- `LENGTH:` Udává délku paketu. Může být zadána jako menší než, větší než, rozsah nebo přesná hodnota. Nepovinná položka.

- `THRESHOLD:` Udává mezní hodnotu v bajtech nebo paketech za vteřinu, jejíž překročení je signálem DDoS Protectoru, aby začal mitigovat provoz spadající do tohoto pravidla. Povinná položka.

- `LIMIT:` Udává mezní hodnotu v bajtech nebo paketech za vteřinu (musí se shodovat s položkou `THRESHOLD`), na kterou se objem provozu sníží v případě, že byla překročena hodnota položky `THRESHOLD`. Povinná položka.

- `WHITELIST:` Udává seznam bezpečných IP adres, které budou z procesu mitigace vyloučeny. Nepovinná položka.

3.2 TCP SYN flood útoky

Pro mitigaci TCP SYN flood útoků se používá *synflood* modul. Tento modul dokáže pomocí tří odlišných strategií blokovat příchozí SYN pakety, které se jeví jako neligitmní. Popis fungování jednotlivých strategií není pro tuto práci důležitý a nebude zde popsán. Stejně jako u *amplifikačního* modulu zde figuruje proměnná *threshold*, která je zadána v paketech za vteřinu. Pokud

počet SYN paketů směřujících do chráněné sítě překročí tuto hranici, spustí se jedna z mitigačních metod, dokud počet SYN paketů směřující do chráněné sítě není zase v povolených mezích. Popis pravidel je zobrazen v následující ukázce a to bez položek, které jsou specifické pro jednotlivé strategie.

```
<RULE> <DST-NET> <SRC-PORT> <DST-PORT> <FRAGMENTATION>

<THRESHOLD> <WHITELIST>
```

Význam jednotlivých položek je následující:

- `RULE:` Nepovinný textový popis pravidla.
- `DST-NET:` Udává chráněnou cílovou síť ve formátu adresy a prefixu (např. 105.185.42.0/24). Může být zadána jak IPv4 tak i IPv6 adresa. Tato položka je povinná.
- `SRC-PORT:` Udává hodnotu zdrojového portu paketu. Může být zadán jako menší než, větší než, rozsah nebo přesná hodnota. Nepovinná položka.
- `DST-PORT:` Udává hodnotu cílového portu paketu. Může být zadán jako menší než, větší než, rozsah nebo přesná hodnota. Nepovinná položka.
- `THRESHOLD:` Udává mezní hodnotu v počtu SYN paketů za vteřinu, jejíž překročení je signálem DDoS Protectoru, aby začal mitigovat provoz spadající do tohoto pravidla. Povinná položka.
- `WHITELIST:` Udává seznam bezpečných IP adres, které budou z procesu mitigace vyloučeny. Nepovinná položka.

Kapitola 4

Analýza síťového provozu

V zájmu každého provozovatele (správce) síťové infrastruktury je udržovat svou síť v kontrolovaném a nenarušeném stavu. K dosažení těchto cílů je zapotřebí, aby měl správce přehled o tom co se v jeho síti aktuálně děje, jaký provoz a kam jeho sítí prochází, aby například dokázal určit cíl DDoS útoku a nastavit taková mitigační pravidla, která útok potlačí. Proto je zapotřebí mít takové nástroje, které jsou taková data schopna poskytnout. Tyto nástroje, dále nazývány jako analyzátory síťového provozu, pracují se dvěma základními typy dat, a to s pakety nebo síťovými toky. Oba přístupy jsou popsány dále v práci. Analyzátory síťového provozu lze ale také dělit podle času analýzy dat a to následovně:

- **Online analýza** zpracovává aktuální data procházející nějakým bodem síti v reálném čase. Tato data jsou vyčtena ze vstupního rozhraní a rovnou i zpracována analyzátorem síťového provozu. Při dnešních rychlostech páteřních sítí a množství dat, které prochází jedním bodem, je velmi obtížné a náročné na výkon provádět analýzu (zejména paketů) přímo online. Mezi nástroje, které využívají online analýzu, patří již zmíněný *DDoS Protector* (pracující s pakety), nebo nástroj graficky znázorňují procházející data *NfSen*[3] (pracující se síťovými toky).

- **Offline analýza** využívá toho, že je možné data nejprve uložit do souboru a teprve později takto uložená data zpracovat analyzátory síťového provozu. Výhodou práce s uloženými daty je možnost provedení sofistikovanější nebo výpočetně složitější analýzy, která by při online analýze nebyla s ohledem na výpočetní náročnost možná, dále je také možné tato data analyzovat opakovaně. Tento způsob analýzy také umožňuje vytvářet dlouhodobé statistiky o provozu v síti. Nevýhodou tohoto přístupu jsou nároky na dostupné místo na disku, které mohou být na

[3] http://nfsen.sourceforge.net

páteřních sítích obrovské. Jako příklad lze uvést paketový analyzátor *Wireshark*[4].

Vzhledem k velkému objemu dat proudících na páteřních sítích podléhají vstupní data před zpracováním obvykle filtrování dle nastavených pravidel, aby se snížila výpočetní náročnost analýzy a zpracovávala se jen ta data, která jsou potřebná. V případě nemožnosti zpracovávat veškerá síťová data nebo pro snížení zátěže je možné použít vzorkování dat (sampling), při kterém se zpracovává jen zlomek dat (například každý *n-tý* paket). Tato metoda je ale závislá na konkrétním účelu pořizování dat, protože ne každý algoritmus v analyzátoru síťových dat je schopen s takto upravenými daty pracovat.

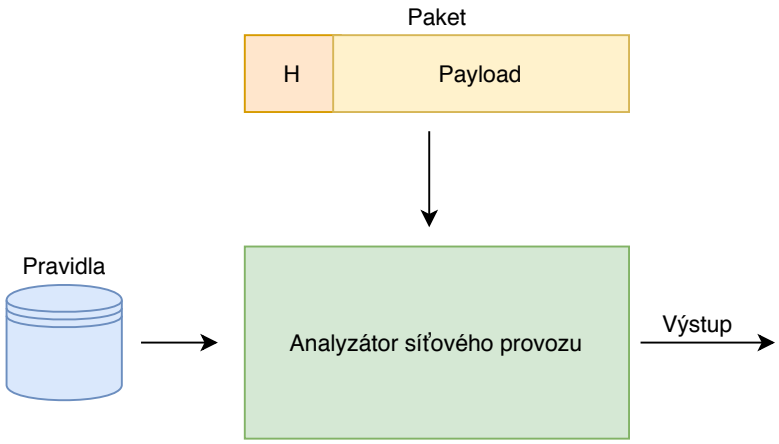
4.1 Analýza na základě paketů

Jak již název napovídá zdrojem informací jsou v tomto případě jednotlivé pakety. Jedná se o zpracování *Raw* dat, tedy dat zachycených během komunikace v jejich původní formě. Výhodou paketové analýzy je, že dokáže vidět celá neupravená data, oproti analýze síťových toků, kde jsou data agregována. Této výhody využívá několik analyzačních algoritmů, které dokáží při analýze paketů poskytovat mnohem lepší výsledky, než by tomu bylo u analýzy síťových toků. Například *DDoS Protector* dokáže provádět mnohem jemnější analýzu a blokaci dat. Jednou z metod, která se při analýze paketů používá je tzv. hloubková analýza paketu (*Deep Packet Insepction, DPI)*. DPI provádí analýzu payloadu paketu, která značně rozšiřuje možnosti, co mohou analyzátory síťového provozu detekovat za události. Nástroje využívající techniku DPI jsou například *Suricata*[5] nebo *Snort*[6]. Ukázka high-level schématu jak takový analýzator síťového provozu funguje ukazuje obrázek 4.1. Datovým vstupem analyzátoru jsou pakety, obsahující hlavičky a payload. S těmito pakety analyzátor v závislosti na zadaných pravidlech provede potřebné operace a typicky vytvoří určitou formu výstupu. Může se jednat o detekci incidentu, grafické zobrazení dat, informace k zablokování dat apod.

4.2 Analýza na základě síťových toků

Vstupem analyzátoru je v tomto případě záznam o síťovém toku, nazývaný také jako *flow* záznam. Síťový tok je definován jako jednosměrná posloupnost paketů se stejnými vlastnostmi procházející pozorovacím bodem v určitém časovém období. Stejnými vlastnostmi se rozumí zdrojová a cílová IP adresa, zdrojový a cílový port a protokol transportní vrstvy. Tyto toky jsou

[4] https://www.wireshark.org
[5] https://suricata-ids.org
[6] https://www.snort.org



Obrázek 4.1: High-level schéma paketového analyzátoru síťového provozu.

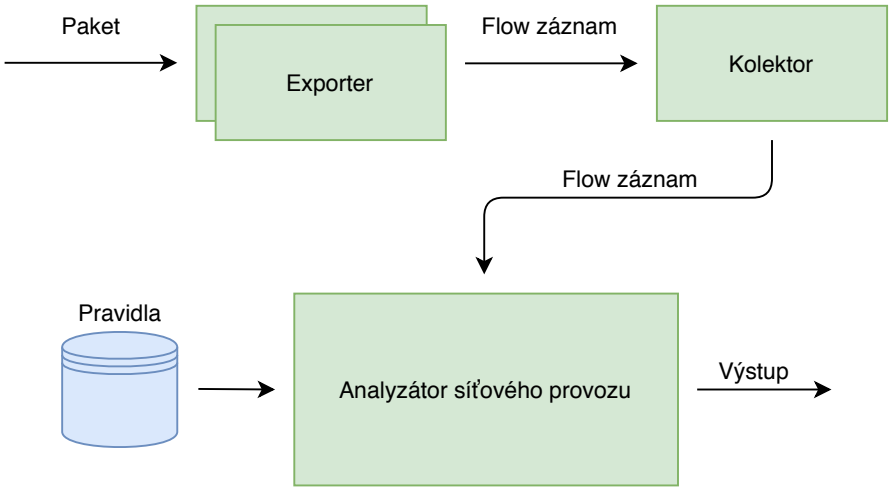
sbírány typicky na několika místech v sítí, a to na zařízeních nazývaných *Exportéry*. Exportér může být např. router, switch nebo jiné specializované síťové zařízení. Exportér zachytává příchozí pakety, ze kterých získá požadovaná data a provede agregaci do jednotlivých síťových toků. Po určité době exportér přestane další pakety zachytávat a tok ve formě záznamu exportuje na *kolektor*. Základní záznam o síťovém toku typicky obsahuje další informace jako je začátek a konec toku, počet paketů nebo součet velikosti paketů. Tento základní záznam může být exportérem rozšířen o další položky z aplikačních protokolů. Těmto záznamům se říká rozšířené. Exporter typicky odesílá záznamy ve formátu *NetFlow*[7] nebo *IPFIX*[8]. Kolektor má na starost sběr flow záznamů od jednodnotlivých exporterů. Exportér může záznamy ukládat pro další zpracování, nebo je přímo předat analyzátorům síťového provozu. High-level schéma, jak tento proces probíhá ukazuje obrázek 4.2. Flow záznamy jsou analyzátorem opět zpracovány podle zadaných pravidel a následně je obvykle vytvořena nějaká forma výstupu.

Výhodou analýzy založené na bázi síťových toků oproti paketové analýze je menší objem zpracovávaných dat, protože jsou typicky exportována jen ta políčka, která jsou pro analýzu potřeba a navíc je agregováno více paketů do jednoho záznamu. To také vede k menším nárokům na diskové místo v případě ukládání dat. Tento typ analýzy se používá především pro monitorování provozu v síti, ale i k detekci bezpečnostních incidentů nebo k účtování za množství přenesených dat. Příkladem nástroje využívající datových toků je analyzátor síťového provozu *NEMEA*[9].

[7] https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod$_{w}hite_{p}aper0900aecd80406232.html$

[8] https://en.wikipedia.org/wiki/IP$_{F}low_{I}nformation_{E}xport$

[9] https://github.com/CESNET/Nemea



Obrázek 4.2: High-level schéma analyzátoru síťového provozu založeného na bázi síťových toků.

Kapitola 5

Sketche

Tato kapitola si klade za cíl seznámit čtenáře s pravděpodobnostní datovou strukturou nazývanou sketch, jejími typy a možnostmi využití se zaměřením na sketche pro odhad frekvence prvků v množině, jejichž použití při návrhu a implementaci nástroje pro analýzu je základní myšlenkou této práce.

5.1 Popis

Sketche jsou pravděpodobnostní datové struktury, které díky svým vlastnostem dokáží efektivně uchovávat velké množství dat v poměru s jejich malou paměťovou náročností. Toho dosahují tím, že pro ukládání dat využívají různé kompresní metody, které ale mohou vést k určité míře nepřesnosti uložených dat. Přesnost uložených dat je silně závislá na velikosti sketche a je tak možné ji předem ovlivnit. Struktura sketche také umožňuje efektivní provádění operací, potřebných k vyhodnocení uložených dat.

Existuje několik variant sketchů s různými typy použití. Nejznámějším sketchem je Bloomův filtr [25], který se používá k ověřování přítomnosti prvku v množině. Dalším známým sketchem je HyperLogLog [26], který se používá k výpočtu kardinality množiny. Tato práce je zaměřená na sketche, které slouží k odhadnutí frekvence jednotlivých prvků v množině. Konkrétně se jedná o Count-Min sketch a Count-Median sketch, které jsou detailněji popsány níže v této sekci.

5.2 Odhad frekvence

Problém hledání frekvence prvků v zadané množině, nazývaný také jako *Count Tracking problem* [27] je jednou ze základních věcí, která se typicky řeší při zpracování dat. Každé řešení tohoto problému musí poskytovat dvě základní funkce: `Update`(i, c), která aktualizuje frekvenci prvku i přičtením hodnoty c a `Estimate`(i), která poskytne aktuální odhad frekvence prvku i.

Tento problém lze řešit jednoduše a exaktně pomocí tradičních datových struktur jako je například hash tabulka, která v sobě uloží čítač pro každý unikátní prvek ze vstupní množiny. Operace `Update` pak tento čítač aktualizuje přičtením dané hodnoty a `Estimate` vrátí hodnotu čítače. Obě tyto funkce mohou být implementované pomocí standardních funkcí, které hash tabulka poskytuje.

Popsaný způsob řešení ale přináší do řešení problému celou řadu nevýhod. Množství paměti použité datovou strukturou může být velmi velké s postupným přidáváním nových prvků. Exaktní metody řešení problému potřebují $O(m)$ čítačů pro uložení m rozdílných prvků. V některých případech se může stát, že prostor pro uložení m rozdílných prvků přesahuje velikost dostupné paměti a nutnost využití přístupu do externí paměti celý proces značně zpomalí. Zpomalení ale může nastat i při relativně malém m, kdy se paměť datové struktury nevejde do cache paměti procesoru a zhorší se tak lokalita dat, což vede na častější výpadek cache a nutnost přístupu do hlavní paměti.

Datová struktura sketche poskytuje řešení těchto problému a to za cenu nahrazení přesného výsledku v odpovědi vysoce kvalitní aproximací. Některé případy použití, jako je třeba monitorování síťového provozu jsou schopny pracovat s určitou mírou chybovosti, která se nepovažuje za významnou. Pro odhad frekvence lze použít Count-Min nebo Count-Median sketche, které pracují s pevnou velikostí paměti, která se velmi často vejde do cache paměti, což zajistí rychlou aktualizaci nebo zpracování dotazu. Sketche ve své datové struktuře navíc neukládají klíče uložených prvků, což může vést v případech, kdy není třeba si pamatovat všechny uložené klíče ke značnému snížení paměťových nároků.

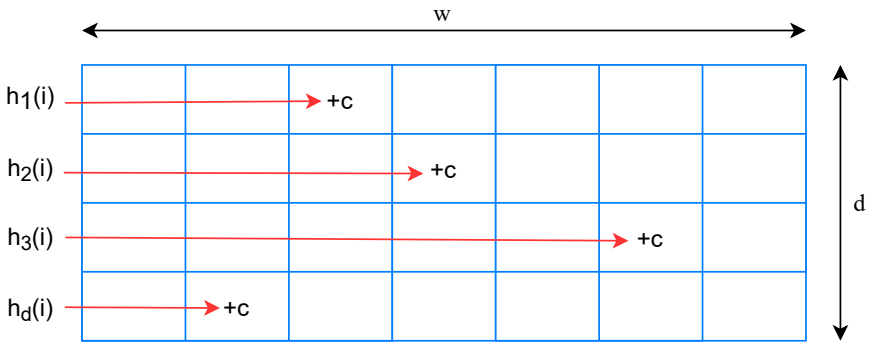
Count-Min sketch (CM sketch) poskytuje řešení problémů výpočtu přibližného počtu prvků v zadané množině dat a je pojmenovaný podle dvou základních operací: Počítání prvků a hledání minima. CM sketch byl navržen Grahamem Cormodem a S. Muthukrishnanem a tento text vychází z jejich článku [27]. CM sketch ke své práci využívá předem alokovanou paměť o pevné velikosti, která svou velikost s rostoucím počtem uložených prvků nemění. Přesto je CM sketch schopen poskytnou užitečné odhady frekvence prvků a to s předem definovanou přesností.

Datová struktura CM sketche je reprezentována dvourozměrným polem čítačů o šířce w a výšce d: $count[1,1]...count[d,w]$. Při inicializaci jsou všechny čítače nastaveny na nulu a každý řádek pole má přiřazenou rozdílnou hashovací funkci $\{h_1, h_2, ...h_d\}$, která mapuje vstupní prvky rovnoměrně v rozmezí $\{1, 2, ...w\}$ v přiřazeném řádku.

Princip fungování základních funkcí *Update* a *Estimate* je popsán společně s funkcí *Merge* zde:

- **Update(i,c)**: Funkce v každém řádku spočítá hash prvku i pomocí hashovací funkce přiřazené k danému řádku. Na základě hashe se

určí, jaký čítač v daném řádku se má použít a k tomuto čítači je následně přičtena hodnota c. Formální zápis: $count[j, h_j(i)] += c$ pro všechny $j \in [d]$. Ukázku principu této funkce zobrazuje obrázek 5.1.



Obrázek 5.1: Princip funkce Update u Count-Min sketche.

- **Estimate(i)**: Tato funkce je podobná funkci Update. Opět se v každém řádku vypočítá hash prvku i a určí se pozice jeho čítače v daném řádku. Následně je ze všech řádků jako výsledek vybrán čítač s nejmenší hodnotou. Formálně: $\hat{a}_i = \min_j count[j, h_j(i)]$ pro všechny $j \in [d]$. Pokud a_i je skutečná frekvence prvku a $\hat{a}_i$ je hodnota vrácená ze sketche vždy platí: $a_i \leq \hat{a}_i$. Výsledná hodnota tedy nemůže být nikdy podhodnocená skutečné frekvenci.
- **Merge**: Funkce Merge lze použít pouze nad sketchi o stejné velikosti a stejných hashovacích funckích. Merge se provádí sčítáním čítačů, které leží na stejných indexech. Tato funkce lze využít například při vícevláknovém zpracování, kde každé vlákno má svou vlastní instanci sketche a před dotazováním se sketche jednotlivých vláken sloučí do jednoho sketche, nad kterým se dotazování provede.

CM sketch je parametrizován dvěma parametry (w, d), které určují velikost dvojrozměrného pole, ale i z ní vycházející požadovanou přesnost výsledku. Přesnost výsledku se vztahuje k aktuální sumě všech čítačů v daném řádku a tato suma bude dále značena jako N. Kvalita výsledku je garantována několika statistickými analýzami blíže popsanými v [27].

Tyto analýzy říkají, že pokud je celková suma všech čítačů v daném řádku N, tak hodnota dotazovaného čítače bude nanejvýš $2N/w$ větší než skutečná frekvence a to s pravděpodobností nejméně $\frac{1}{2}$. Tento samý proces se provádí u každého řádku s rozdílnou hashovací funkcí. Protože jsou hashovací funkce rozdílné, je v každém řádku rozdílné mapování

prvků na čítače a rozdílné prvky spolu kolidují. Pokaždé je zde ale nanejvýš 50 % šance na to, že chyba čítače bude větší než $2N/w$ a alespoň 50 % šance na to, že chyba bude menší. Protože CM sketch jako výsledek vrací hodnotu čítače z řádku, ve kterém měl dotazovaný prvek nejmenší hodnotu, pravděpodobnost že bude výsledek od skutečné frekvence větší než $2N/w$ nastane s pravděpodobností nanejvýš $\left(\frac{1}{2}\right)^d$.

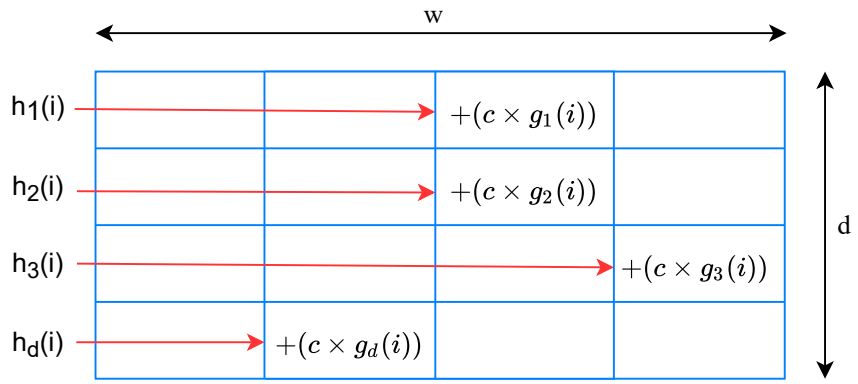
Na základě analýzy zde bude uveden příklad jak takové nastavení parametrů u CM sketche může vypadat. Předpokládá se chyba nanejvýš 0.1% (z celkové sumy všech frekvencí), s mírou jistoty 99.9 %, že nenastane chyba větší než $2N/w$. Potom musí platit $2/w = 1/1000$, tedy $w = 2000$ a $\left(\frac{1}{2}\right)^d = 0.001$, tedy $d = 10$.

Count-Median sketch často nazývaný jen Count sketch je v mnoha ohledech velmi podobný Count-Min sketchi, a proto zde budou popsány především odlišnosti mezi těmito dvěma sketchi. Následující text vychází ze článku [28]. Datová struktura Count sketche je stejná jako u Count-Min sketche. Opět je tedy reprezentována dvourozměrným polem čítačů o šířce w, výšce d a množinou hashovacích funkcí $\{h_1, h_2, ..., h_d\}$. Count sketch navíc přidává druhou sadu hashovacích funkcí $\{g_1, g_2, ..., g_d\}$, které mapují vstupní prvky na hodnotu +1 nebo -1.

Princip fungování základních funkcí *Update* a *Estimate* je popsán zde:

- **Update(i,c)**: Funkce v každém řádku spočítá hash prvku i pomocí hashovací funkce $h_j(i)$ přiřazené k danému řádku. Dále se spočítá druhá hash $g_j(i)$, která namapuje prvek i na hodnotu +1 nebo -1. Na základě hashe $h_j(i)$ se určí, jaký čítač v daném řádku se má použít a k tomuto čítači je následně přičtena hodnota $+(c \cdot g_j(i))$. Formální zápis: $count[j, h_j(i)] += c \cdot g_j(i)$ pro všechny $j \in [d]$. Ukázku principu této funkce zobrazuje obrázek 5.2.
- **Estimate(i)**: Tato funkce je podobná funkci Update. Opět se v každém řádku vypočítají dvě hash funkce $h_j(i)$ a $g_j(i)$. Na základě funkce $h_j(i)$ se určí pozice čítače v řádku a jako výsledek z daného řádku se vezme hodnota čítače vynásobená hodnotou +1 nebo −1, kterou určuje funkce $g_j(i)$. Výsledná hodnota je spočítána jako medián hodnot ze všech řádků. Formálně: $\hat{a}_i = median_j count[j, h_j(i) \cdot g_j(i)]$ pro všechny $j \in [d]$. Výsledná hodnota $\hat{a}_i$ může být oproti skutečné frekvenci a_i podhodnocená, ale i nadhodnocená.

Count-Median sketch je stejně jako Count-Min sketch parametrizován dvěma parametry (w, d), které určují velikost dvojrozměrného pole, ale i z ní vycházející požadovanou přesnost výsledku. Popsané vzorce vycházejí z textu [29], kde jsou uvedené matematické důkazy. Šířka w se určuje jako $w = \left\lceil \frac{k}{\epsilon^2} \right\rceil$, kde k je konstanta > 2, která ovlivňuje



Obrázek 5.2: Princip funkce Update u Count sketche.

pravděpodobnost chyby čítače (a také poměr mezi šířkou a hloubkou sketche) a ϵ určuje velikost absolutní chyby, která zaručuje, že hodnota čítače dotazovaného prvku bude od skutečné frekvence vzdálená nejvíce o $\epsilon \|v\|_2$ a to s pravděpodobností $1 - \delta$. $\|v\|_2 = \sqrt{\sum_{i=1}^{m} v_i^2}$ je $L2$-norma vstupních dat. Pravděpodobnost, že chyba bude v zadaných mezích ovlivňuje hloubka sketche d a to následovně: $d = \left\lceil \frac{\ln(\delta^{-1})}{\frac{1}{6} - \frac{1}{3k}} \right\rceil$.

Na základě analýzy zde bude opět uveden příklad, jak takové nastavení parametrů u Count-Median sketche může vypadat. Předpokládá se chyba nanejvýš 0,1 %, s mírou jistoty 99,9 %, že nenastane chyba větší než $\epsilon \|v\|_2$, pro zvolené $k = 4$ Potom musí platit $w = \left\lceil \frac{4}{0{,}01^2} \right\rceil$, tedy $w = 4000$ a $d = \left\lceil \frac{\ln(0{,}001^{-1})}{\frac{1}{6} - \frac{1}{12}} \right\rceil$, tedy $d = 83$.

5.3 Heavy hitters problém

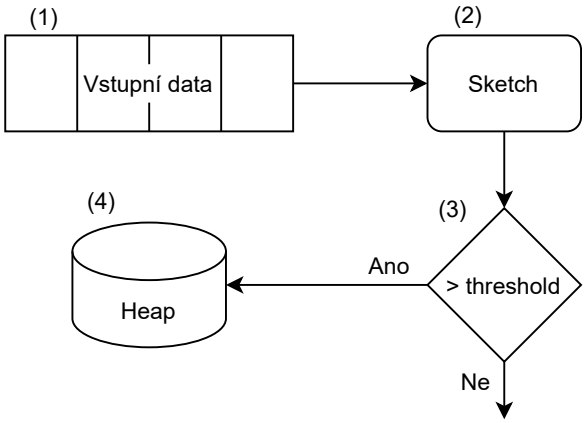
Heavy hitters problém známý také jako problém hledání nejfrekventovanějších nebo nejpopulárnějších prvků má za cíl nalézt v zadané množině dat prvky, které jsou v množině zastoupeny nejvícekrát.

Heavy hitter (HH) problém se typicky skládá ze vstupní množiny o délce (počtu prvků) n a parametru k ($n >> k$). Cílem je během jednoho průchodu zadané množiny nalézt prvky, které se v množině objevují alespoň $\frac{n}{k}$ krát.

Problém lze stejně jako při odhadu frekvence řešit jednoduchým a exaktním způsobem řešení, který zde ovšem naráží na stejné problémy, které již byly popsány v sekci 5.2. Proto lze pro řešení HH problému využít struktury Count-Min nebo Count-Median sketche a řešit tento problém přibližně se zadanou přesností, kterou sketch poskytuje. Ukázka, jak takové řešení HH problému s pomocí sketche může vypadat zobrazuje obrázek 5.3. Popis jednotlivých

kroků v obrázku vychází ze článku [30] a počítá s předem neznámým počtem prvků m vstupní množiny.

1. Postupně bereme položky x_i ze vstupní množiny dat.
2. Ke každému položce zavoláme nad sketchem funkci Update(x_i, c), která aktualizuje čítače dané položky ve sketchi.
3. Následně zavoláme funkci Estimate(x_i). Pokud platí Estimate(x_i) $\geq \frac{m}{k}$, přejdeme na krok číslo 4, jinak ověříme, že se zadaná položka není uložená na hromadě a pokud je, tak ji odstraníme. Hodnota m je počet dosud zpracovaných prvků.
4. Uložíme klíč položky x_i na hromadu, pokud tam již není.



Obrázek 5.3: Princip detekce Heavy hitters s využitím sketche.

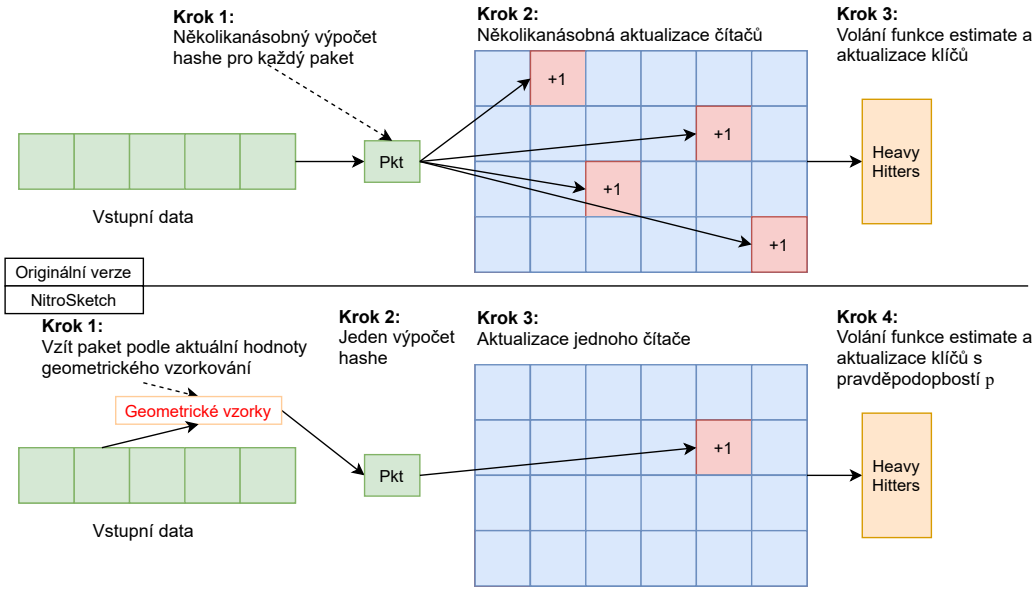
Na konci jsou tak na hromadě uloženy všechny klíče, splňující podmínku $\frac{n}{k}$, protože zde ale mohou být uložené i klíče, které během zvětšujícího se m přestaly podmínku splňovat, je nutné všechny uložené položky projít a odstranit ty, které podmínku nesplňují.

5.4 NitroSketch

NitroSketch představuje postup, jakým lze zvýšit výkonnost struktury sketche a Heavy hitteru na základě vzorkování vstupních dat. Tento postup byl popsán v článku [31], ze kterého vychází následující text.

Porovnání NitroSketche oproti originální variantě je ukázáno na obrázku 5.4. Hlavní myšlenkou NitroSketche je snížení počtu prováděných výpočetních operací a snížení počtu zpracovaných dat. NitroSketch ze vstupní množiny vybírá pouze ta data, u kterých je potřeba aktualizovat čítač. Data jsou

vybírána na základě hodnoty proměnné z geometrické řady, která slouží k vzorkování vstupních dat, ale i pole čítačů. Na základě této hodnoty a hloubky sketche je určeno, kolik čítačů (a tedy i paketů) se má přeskočit, než bude potřeba znovu aktualizovat čítač. Pokud bude nastaveno vzorkování na 1/32, bude aktualizován každý 32 čítač, pokud by sketch měl hloubku 5 (tedy 5 čítačů na jeden paket), byl by aktualizován jeden čítač v přibližně každém 5-6 paketu. NitroSketch oproti originální verzi neprovádí aktualizaci struktury Heavy hitteru s každým zpracovaným paketem, ale provedení funkce `Estimate` nad sketchem a aktualizace Heavy hitteru je provedena s pravděpodopností p.



Obrázek 5.4: Porovnání NitroSketche s originální variantou sketche a Heavy hitters.

NitroSketch může být implementován jak s Count-Min tak i s Count-Median sketchem. Podle matematických důkazů popsaných v článku [31], NitroSketch konverguje k zadané chybovosti použitého sketche se zvětšujícím se množstvím zpracovaných dat.

Kapitola 6

Návrh

Tato kapitola se zabývá návrhem nástroje pro online paketovou analýzu síťového provozu. V první části jsou blíže popsány požadavky, které by měl nástroj splňovat. Druhá část je zaměřena na popis architektury nástroje včetně detailního popisu jednotlivých částí a možnosti paralelizace.

6.1 Požadavky na software

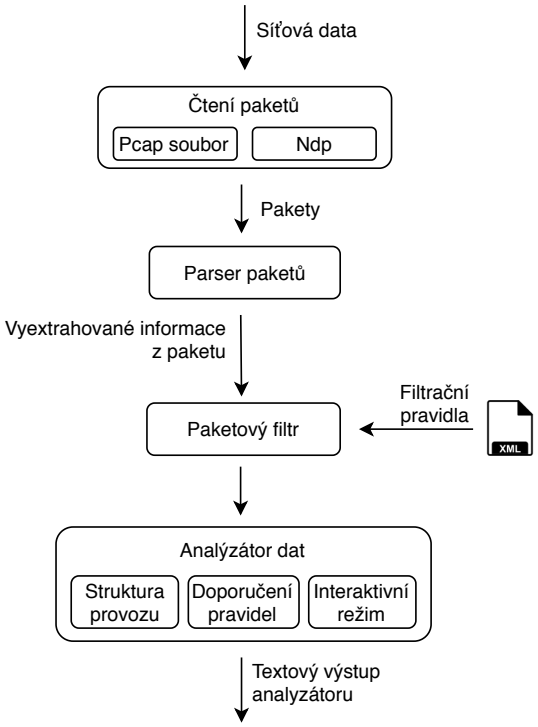
Před samotným návrhem programu je potřeba identifikovat požadavky, které vyplývají z účelu programu a způsobu místa zapojení v síti. Je tedy potřeba vzít v úvahu následující požadavky:

- Program by měl poskytovat administrátorovi sítě přehled o aktuální struktuře provozu. To je důležité především pro odhalení typu a cíle DDoS útoku.
- Program by měl být schopen na základě charakteristik volumetrických DDoS útoků určit možný cíl útoku a doporučit taková mitigační pravidla, která útok potlačí. Doporučená mitigační pravidla by měla být kompatibilní s pravidly *DDoS Protectoru*.
- Program by měl být schopen filtrovat vstupní pakety, které mají být analyzovány a to na základě administrátorem definovaných pravidel.
- Analýza provozu musí být prováděna nad pakety a v reálném čase.
- Návrh počítá s nasazením programu v nejvyšší úrovni síťové infrastruktury, které jsou dimenzovány na vysoké přenosové rychlosti. Infrastruktura sítě CESNET, pro kterou je návrh programu primárně zaměřen pracuje na rychlosti 100 Gb/s. Z toho důvodu je důležité, aby program dokázal na takto vysoké rychlosti pracovat.

- Program by měl být paměťově efektivní a úsporný, bez nutnosti dynamické alokace paměti za běhu analyzačního algoritmu. To je důležité i s ohledem na možnou budoucí implementaci do FPGA čipu síťové karty.

6.2 Architektura

Architektura softwaru je ukázána na obrázku 6.1. Návrh počítá s rozdělením do čtyř základních částí, které jsou detailně popsány dále v textu.



Obrázek 6.1: Vysokoúrovňová architektura nástroje.

6.3 Čtení paketů

Tato část bude řešit vyčítání paketů (síťových dat) ze vstupního rozhraní. Program bude pracovat se dvěma následujícími druhy rozhraní:

PCAP rozhraní bude využívat knihovny *libpcap* pro práci s *PCAP* soubory. Vyčítání paketů ze souborů skrz knihovnu *libpcap* není příliš výkonné, rychlost čtení se pohybuje maximálně okolo 1 Gb/s, což pro potřeby

vytvářeného softwaru nebude dostatečné. Nicméně toto rozhraní je vhodné pro testovací účely, případně pro provádění offline analýzy ze zachycených PCAP souborů.

NDP (Netcope Data Plane) je vysokorychlostní rozhraní, které zajišťuje DMA přenosy dat mezi hardwarově akcelerovanou síťovou kartou *nfb* (Netcope FPGA Board) a cílovou aplikací. Program bude využívat knihovny *libnfb* k vysokorychlostnímu čtení paketů skrz NDP rozhraní ze síťové karty. Toto rozhraní a speciální síťovou FPGA kartu ke své činnosti využívá také *DDoS Protector*.

6.4 Parser paketů

Z přijatých paketů bude potřeba získat potřebné informace o jejich obsahu pro další zpracování. Tyto informace budou vyčteny z hlaviček jednotlivých protokolů, ve kterých jsou zapouzdřena přenášená data. Informace, které budou z paketů získávány popisuje tabulka 6.1. Velikost jednotlivých položek odpovídá skutečné velikosti položek v hlavičkách paketu, až na zdrojovou a cílovou IP adresu a Vlan ID. Vzhledem k tomu, že bude podporována IP adresa verze 4 i 6, je velikost těchto položek přizpůsobena velikosti IPv6 adresy. IPv4 adresa využije z této velikosti pouze 4 B, zbytek bude nevyužitý. Velikost alokovaná pro Vlan ID je v tomto případě rozšířena na 2 B, resp. 16 bitů z originální velikosti položky 12 bitů. Celkově tak každý paket bude reprezentován 42 B dat.

Položka	Velikost (Byte)
Source IP	16 B
Destination IP	16 B
Protocol	1 B
Source port	2 B
Destination port	2 B
Length	2 B
TCP flags	1 B
Vlan ID	2 B
Celkem	42 B

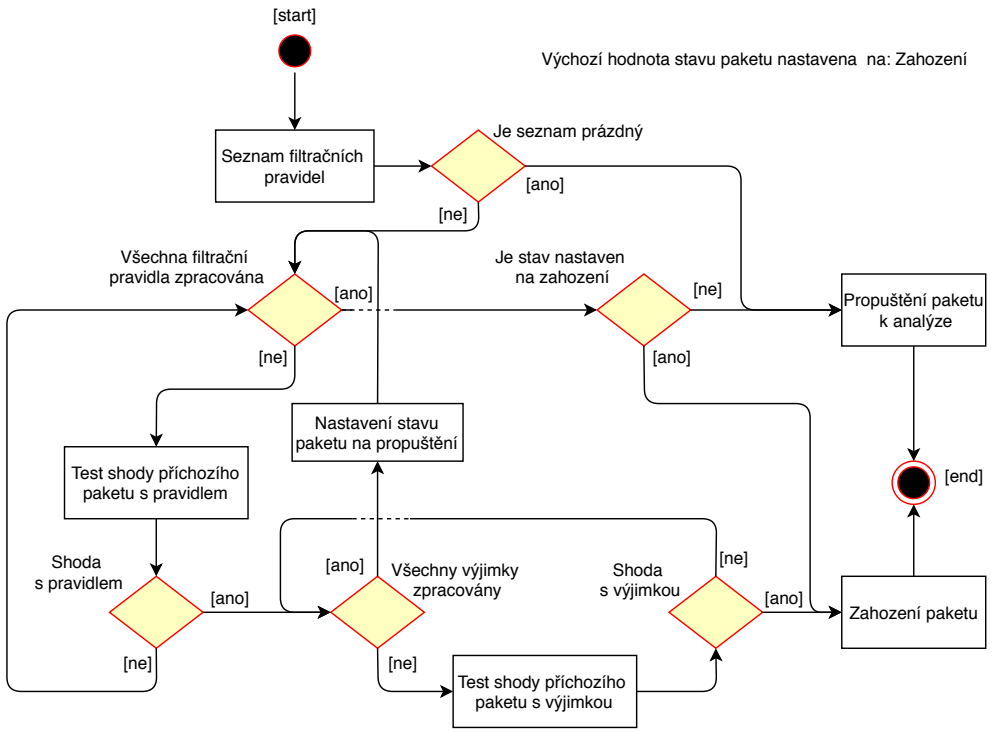
Tabulka 6.1: Položky a jejich velikost získávané z paketů

6.5 Paketový filter

Paketový filtr bude mít na starost filtrování paketů na základě administrátorem zadaných filtračních pravidel. Filtr bude logicky zapojen za parserem paketů, který mu bude předávat vyextrahované informace (viz tabulka 6.1)

z příchozích paketů. Každý nově příchozí paket bude porovnáván vůči množině filtračních pravidel, které zadá administrátor skrz konfigurační soubor ve formátu *XML*. Každé filtrační pravidlo také bude moci obsahovat seznam výjimek (whitelist), ve kterém bude moci administrátor blíže specifikovat, jaký typ paketů chce z analýzy vyloučit v případě, že se paket bude shodovat s daným filtračním pravidlem.

Proces rozhodování paketového filtru popisuje vývojový diagram na obrázku 6.2. Stav příchozí paketu bude ve výchozím stavu nastaven na jeho zahození, ale v případě, že nebudou nastavena žádná filtrační pravidla, budou všechny pakety filtrem propouštěny dále k analýze. Příchozí paket bude postupně porovnáván se všemi příchozími filtračními pravidly, pokud se paket s filtračním pravidlem nebude shodovat, automaticky se přejde na následující pravidlo. V případě shody paketu s pravidlem se bude paket dále porovnávat se zadaným seznamem výjimek pro dané pravidlo. Pokud bude mít paket shodu s nějakou s výjimek, rozhodne se o přímém zahození paketu a další pravidla se již nebudou porovnávat. Pokud nebude mít paket shodu s žádnou z výjimek nebo seznam výjimek bude prázdný, změní se rozhodnutí o stavu paketu na propuštění a přejde se na další pravidlo.



Obrázek 6.2: Vývojový diagram popisující rozhodovací proces paketového filtru.

Návrh v jakém formátu budou zadávána pravidla skrz konfigurační soubor je zobrazen na následujícím příkladu:

```
<rule>
    <src_address>IP address/prefix</src_address>
    <dst_address>IP address/prefix</dst_address>
    <protocol>value</protocol>
    <src_port>value</src_port>
    <dst_port>value</dst_port>
    <length>value</length>
    <vlan_id>value</vlan_id>

    <whitelist>
        <src_address>IP address/prefix</src_address>
        <dst_address>IP address/prefix</dst_address>
        <protocol>value</protocol>
        <src_port>value</src_port>
        <dst_port>value</dst_port>
        <length>value</length>
        <vlan_id>value</vlan_id>
    </whitelist>

    <whitelist>
        .
        .
        .
    </whitelist>
</rule>
<rule>
    .
    .
    .
</rule>
```

Filtrační pravidla vychází z pravidel, které používá *DDoS Protector*. Filtrační pravidlo se bude značit tagem `<rule>` a konfigurační soubor těchto pravidel bude moci obsahovat více. Každé pravidlo bude následně specifikováno minimálně jednou z následujících položek:

- `<src_address>`: Udává zdrojovou IP adresy a prefix, do které musí zdrojová IP adresa příchozího paketu spadat. Formát zadání je např. 105.185.42.0/24. Může být zadána jak IPv4 tak i IPv6 adresa.
- `<dst_address>`: Udává cílovou IP adresy a prefix, do které musí cílová IP adresa příchozího paketu spadat. Formát zádání je shodný s položkou `<src_address>`.

- `<protocol>`: Udává protokol paketu podle hodnoty položky *Protocol* v IP hlavičce. Hodnota může být zadána jako menší než, větší než, rozsah nebo přesná hodnota. Příklad možného zadání je např. `<53`, `>17`, `6-1000` nebo `1024`. V případě zadání hodnoty menší nebo větší než, je nutné porovnávací znak v XML souboru psát pomocí ekvivalentní escape sekvence a to `lt;` pro znak `<` a `gt;` pro znak `>`.

- `<src_port>`: Udává hodnotu zdrojového portu paketu. Hodnota může být zadána jako menší než, větší než, rozsah nebo přesná hodnota.

- `<dst_port>`: Udává hodnotu cílového portu paketu. Hodnota může být zadána jako menší než, větší než, rozsah nebo přesná hodnota.

- `<length>`: Udává délku paketu. Hodnota může být zadána jako menší než, větší než, rozsah nebo přesná hodnota.

- `<vlan_id>`: Udává Vlan ID, do které musí paket spadat. Hodnota musí být zadána jako přesná hodnota. Tato položka není v pravidlech aktuální verze *DDoS Protectoru* podporována, ale je plánováno její přidání a proto se zde s touto položkou počítá.

- `<tcp_flags>`: Udává TCP příznaky obsažené v paketu. Položka je validní pouze s protokolem TCP a může nabývat následujících hodnot:

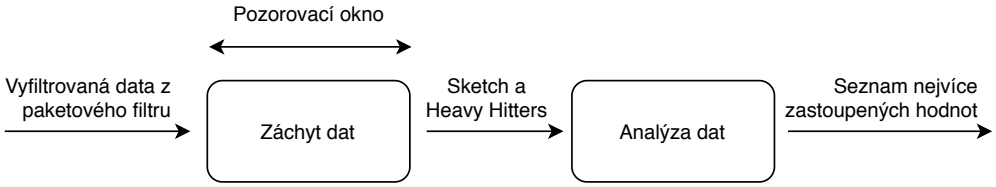
  - `S`: Příznak SYN.
  - `A`: Příznak ACK.
  - `F`: Příznak FIN.
  - `P`: Příznak PUSH.
  - `R`: Příznak RESET.

- `<whitelist>`: Popisuje výjimku se zadaného pravidla. Pravidlo může obsahovat více výjimek a každá výjimka může obsahovat položky, které jsou popsány výše.

6.6 Režimy analýzy dat

Návrh předpokládá s implementací tří režimů analýzy dat, které administrátorovi poskytnou potřebnou funkcionalitu pro identifikování DDoS útoku. Část programu, která bude implementovat tyto režimy bude logicky zapojená za paketový filtr a vstupem tak budou přefiltrované informace o paketech. Režimy budou ke své práci využívat struktury Sketchů a Heavy Hitters, které byly popsány v kapitole 5. Nástroj bude pracovat vždy jen s jedním vybraným režimem.

Struktura provozu. Tento režim bude poskytovat administrátorovi sítě přehled o tom, jakou strukturu má vstupní síťový provoz, podobně jako tomu je u jiných analyzátorů síťového provozu. Výstupem algoritmu bude seřazený seznam nejvíce zastoupených hodnot a to pro každou položku z tabulky 6.1. Administrátor tak například získá přehled o tom, jaké je aktuální zastoupení protokolů, portů, délek paketů atd. Tyto informace mohou administrátorovi usnadnit identifikaci typu DDoS útoku a to například zvýšeným výskytem určitého protokolu v síťovém provozu. Počet položek v zobrazovaném seznamu bude omezen pomocí administrátorem definované proměnné *TOP-N*, která bude vyjadřovat kolik nejvíce zastoupených hodnot může být zobrazeno na výstupu. Návrh schématu fungování tohoto režimu ukazuje obrázek 6.3. Informace o paketech, které projdou paketovým filtrem budou ukládány do sketche a ke každé sledované položce paketu bude udržována struktura nejvíce frekventovaných položek (Heavy Hitters) o velikosti *TOP-N* hodnot. Data budou zachytávána po administrátorem definovaném časovém období (pozorovací okno). Po skončení tohoto pozorovacího okna se zastaví příjem paketů a proběhne vyhodnocení zpracovaných dat. Na základě nejvíce frekventovaných položek uložených ve strukturách Heavy Hitterru se vyhledají přidružená data ve sketchi a výsledky se odešlou na výstup.



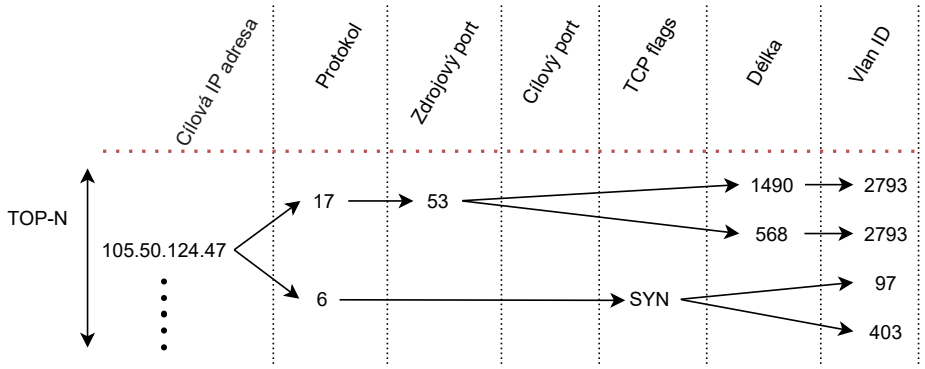
Obrázek 6.3: Návrh schématu fungování režimu pro zobrazení struktury provozu.

Doporučení pravidel. Cílem tohoto režimu je určit potenciální oběť DDoS útoku a doporučit administrátorovi sítě mitigační pravidla, která dokáží útok potlačit. Cíl DDoS útoku se bude určovat podle jedné z typických charakteristik amplifikačních a záplavových DDoS útoků a to podle velkého objemu provozu směřujícího do cílové sítě. Vstupem tohoto módu bude opět hodnota *TOP-N*, která bude udávat, pro kolik nejvíce zastoupených cílových zařízení se má algoritmus pokusit doporučit mitigační pravidla. Druhým vstupním parametrem pak bude hodnota *threshold*, která bude určovat minimální procentuální zastoupení mitigačního pravidla na objemu provozu směrujícím do cílové sítě, které by dané pravidlo zablokovalo.

Návrh počítá s implementací adaptivních filtračních pravidel, které se budou automaticky rozšiřovat a zpřesňovat na základě aktuálně získaných dat. Tato pravidla budou specifikovat, jaký typ provozu splňuje podmínku minimálního procentuálního zastoupení objemu provozu, který je zadán hodnotou *threshold*. Cílem režimu doporučení pravidel není nalezení všech možných

kombinací položek paketu, které splňují podmínku minimálního zastoupení provozu, ale nalezení co možná nejspecifičtějších pravidel v závislosti na pořadí analyzovaných položek. Tento přístup byl zvolen jak s ohledem na požadavek vysoké datové propustnosti, kde nalezení všech možných kombinací jednotlivých položek paketu by bylo výpočetně velmi náročné a při plné rychlosti by nebylo možné takovou analýzu provést, ale i s ohledem na snahu o nezahlcení administrátora sítě velkým množstvím překrývajících se doporučených mitigačních pravidel.

Návrh počítá s postupnou analýzou položek paketu, a to v následujícím pořadí: Cílová IP adresa, protokol, zdrojový port, cílový port, TCP příznaky, délka a vlan ID. Ukázku návrhu odvozování mitigačních pravidel zobrazuje obrázek 6.4. Algoritmus v prvním kroku nejprve určí podle objemu provozu



Obrázek 6.4: Ukázka fungování režimu doporučení mitigačních pravidel.

(počtu paketů nebo bajtů) nejvíce zastoupené cílové IP adresy, které následně začne postupně analyzovat. V prvním kroku bude pouze jedno filtrační pravidlo, které bude specifikovat aktuálně analyzovanou cílovou IP adresu. V druhém kroku byla zvolena analýza protokolu, a to z důvodu, že drtivá většina DDoS útoku je protokolově závislá, což znamená, že všechny útočící pakety obsahují stejný protokol, z toho důvodu určení protokolu útoku přinese lepší vyfiltrování dat pro další analýzu. Jak je vidět z obrázku, protokol číslo 17 a 6 překročily hranici *threshold* a nově tak bude adaptivní filtr obsahovat dvě pravidla (Cílová IP adresa + protokol [17 nebo 6]). V dalším kroku byla zvolena analýza zdrojového portu, který také u většiny útoků bývá pevně daný (např. DNS amplifikační útok, zdrojový port 53). V opačném případě, kdy je zdrojový port útoku náhodný, se předpokládá, že žádný z portů nepřekročí hranici *threshold* a adaptivní filtrační pravidlo tak zůstane beze změny, jak je ukázáno na obrázku u pravidla s protokolem číslo 6. Další analyzovanou položkou je cílový port, kde je situace obdobná, jako u portu zdrojového. Po cílovém portu bude následovat analýza příznaků TCP protokolu, které jsou důležité při identifi-

kaci TCP SYN flood útoku. Další analyzovanou položkou bude délka paketu, která při uměle generovaném provozu útočníkem skrz nástroj k tomu určený může mít typicky stejnou velikost. Poslední analyzovanou položkou pak bude identifikátor Vlany.

Interaktivní režim. Tento režim bude poskytovat administrátorovi možnost ovlivnění dvou předchozích režimů za běhu algoritmu. Zatímco předchozí režimy budou fungovat zcela automaticky a budou se dát ovlivnit jen skrz konfiguraci nástroje, tento režim bude pomocí vstupu od administrátora skrz standardní rozhraní stdin určovat směr, jakým se má algoritmus vydat. Na základě vstupu pak administrátor dokáže za běhu algoritmu ovlivnit celou řadu věcí, a to v závislosti na zvoleném režimu, jehož výběr bude proveden hned při startu nástroje.

Při zvoleném režimu struktury provozu bude mít administrátor možnost výběru položky paketu, která má být analyzována, a to i opakovaně. Zároveň bude možnost měnit počet zobrazovaných položek, tedy za běhu měnit hodnotu TOP-N.

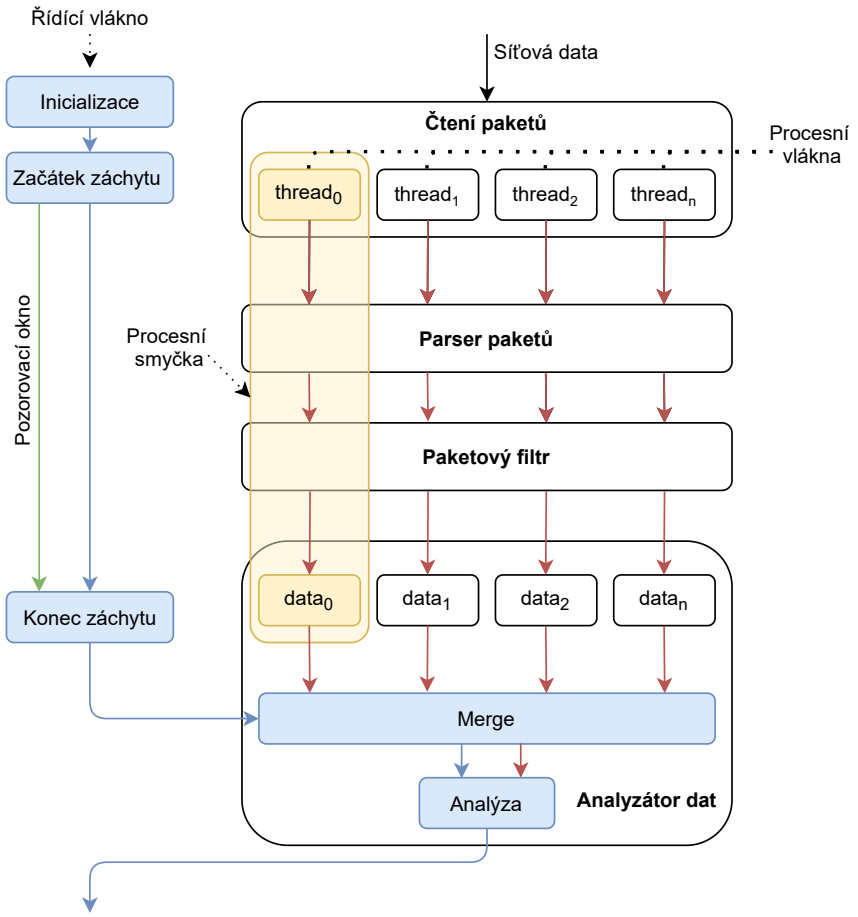
U režimu doporučení mitigačních pravidel bude možné vybrat jednu z TOP-N cílových IP adres, která bude následně analyzována a nebude tak prováděna analýza všech TOP-N adres, jako u původního režimu. Zároveň bude mít administrátor možnost zvolit pořadí analyzovaných položek paketu, které je u původního režimu pevně dáno.

6.7 Možnost paralelizace

Vzhledem k požadavku zpracování velkého množství paketů a objemu dat, bude nutné využít možnost paralelizace pomocí vláken. Návrh počítá s paralelním zpracováním dat na všech úrovních, které zpracovávají příchozí pakety. Návrh paralelizace nástroje je ukázán na obrázku 6.5. Návrh paralelizace rozděluje nástroj do dvou základních částí: řídící a jednu nebo více procesních.

Řidící vlákno (modré části) bude mít na starost načtení konfigurace a inicializaci jednotlivých částí nástroje, dále bude řídit začátek a konec zachytávání dat v procesních vláknech.

Procesní část bude mít na starost zpracování příchozích paketů, neboli datovou cestu (červené šipky). Pro zpracování paketů více jak jedním vláknem bude vyžadována podpora na straně vstupního rozhraní. Pcap rozhraní takovou podporu nemá, ale vzhledem k výkonnosti tohoto rozhraní není paralelní zpracování potřeba a bude tak vždy vytvořeno pouze jedno procesní vlákno. NDP rozhraní podporu vícevláknového zpracování paketů nabízí, a to pomocí rozdělování příchozích paketů do určeného počtu DMA kanálů, návrh počítá s rozdělováním paketů do jednotlivých kanálů pomocí Round-robin algoritmu. Každé procesní vlákno pak bude pracovat se svým přiřazeným DMA kanálem a díky rovnoměrnému rozhazování paketů bude výpočetní zátěž rovnoměrně rozdělena mezi procesní vlákna. Vytvoření procesních vláken bude



Obrázek 6.5: Schéma fungování režimu pro zobrazení struktury provozu.

mít na starost modul pro čtení paketů, kde každé vytvořené vlákno bude pracovat se svou vlastní instancí dat (obdélníky $thread_0 - thread_n$). Každá tato instance již paralelně vyčte paket ze svého přiřazeného vstupního rozhraní a předá jej do Parseru paketů. Parser paketů bude sdílen mezi všemi procesními vlákny, protože zde při paralelním zpracování paketů nebude potřeba žádných synchronizačních mechanismů, které by tomu bránily. Vyparsovaná data budou v rámci procesního vlákna dále předána Paketovému filtru, který bude opět sdílen všemi procesními vlákny, neboť zde bude probíhat pouze čtení dat z filtru, které nevyžaduje synchronizaci. Vyfiltrovaná data budou pokračovat do Analyzátoru dat, kde se provedou nad daty potřebné operace a výsledek se uloží každému procesnímu vláknu do vlastní instance dat (obdélníky $data_0 - data_n$). Tento přístup byl zvolen z důvodu, aby se proces zpracování paketů obešel bez nutnosti synchronizace, která je časově velmi drahá operace. V opačném případě by zde docházelo k paralelnímu čtení

a zápisu dat, které si některou z forem synchronizace vyžaduje. Po uložení dat skončí jedna iterace procesního vlákna a přejde se na další, tedy opět na začátek k vyčtení paketu ze vstupního rozhraní. Jedna iterace se nazývá jako procesní smyčka (oranžový obdélník) a je opakována tak dlouho, dokud ji procesní vlákno nezastaví koncem záchytu dat.

Po ukončení záchytu dat proběhne další zpracování dat sekvenčně, a to v řídícím vlákně. Řídící vlákno se spojí s datovou cestou a proběhne sloučení zachycených dat (Merge) z jednotlivých procesních vláken ($data_0 - data_n$). Nad sloučenými daty se jako poslední krok provede analýza podle zvoleného režimu a proběhne zobrazení výsledků.

Kapitola 7

Realizace

V této kapitole je detailně popsána implementace jednotlivých částí analyzačního nástroje. Pozornost je také věnována vyskytnutým problémům a jejich způsobu řešení.

Implementace nástroje je pojata jako konzolová aplikace vytvářená v jazyce C++ pod operačním systémem GNU/Linux. Jazyk C++ byl pro implementaci zvolen z výkonnostních důvodů a program je napsán s nejnovějším standardem jazyka C++20. Pro překlad je použit překladový systém CMake.

7.1 Implementace čtení paketů

Rozhraní pro čtení paketů z podporovaných vstupních rozhraní poskytuje třída `Input_module`. Tato třída obsahuje obecné ukazatele na funkce, jejichž protytyp je vypsán v následující ukázce kódu. Každé vstupní rozhraní pak musí tyto funkce implementovat. Tento přístup byl zvolen z důvodu snadné rozšiřitelnosti o další možná vstupní rozhraní jako je například DPDK.

```
using thread_init =
int (*)(unsigned thread_id, void *ifc_config, void **ifc_data);

using thread_deinit =
void (*)(void *ifc_data);

using packet_read =
int (*)(void *ifc_data, Raw_packet& in_packet);
```

Funkce `thread_init` slouží k inicializaci vstupního rozhraní v procesním vláknu a jejím cílem je vytvořit instanci vstupního rozhraní specifickou pro dané procesní vlákno. Pomocí parametru `ifc_config` se předá potřebná konfigurace pro vstupní rozhraní, tento parametr je zobecněn *void* ukazatelem a může tak obsahovat libovolnou strukturu, kterou si vstupní rozhraní definuje. Funkce skrz parametr `ifc_data` vrací inicializovanou instanci vstupního

rozhraní, která se předává jako obdoba parametru *this* ostatním obecným funkcím. Pro ukončení práce se vstupním rozhraním je zde pak funkce `thread_deinit`.

Funkce `packet_read` slouží k vyčtení paketu ze síťového rozhraní. Funkce v návratovém kódu vrací informaci o tom, zda se podařilo paket úspěšně vyčíst, nebo zda na vstupním rozhraním nejsou data. Pokud se paket podaří úspěšně vyčíst, předá se paket dále ke zpracování skrz parametr `in_packet`. Tento parametr reprezentuje struktura, která obsahuje ukazatel na začátek dat paketu a velikost paketu.

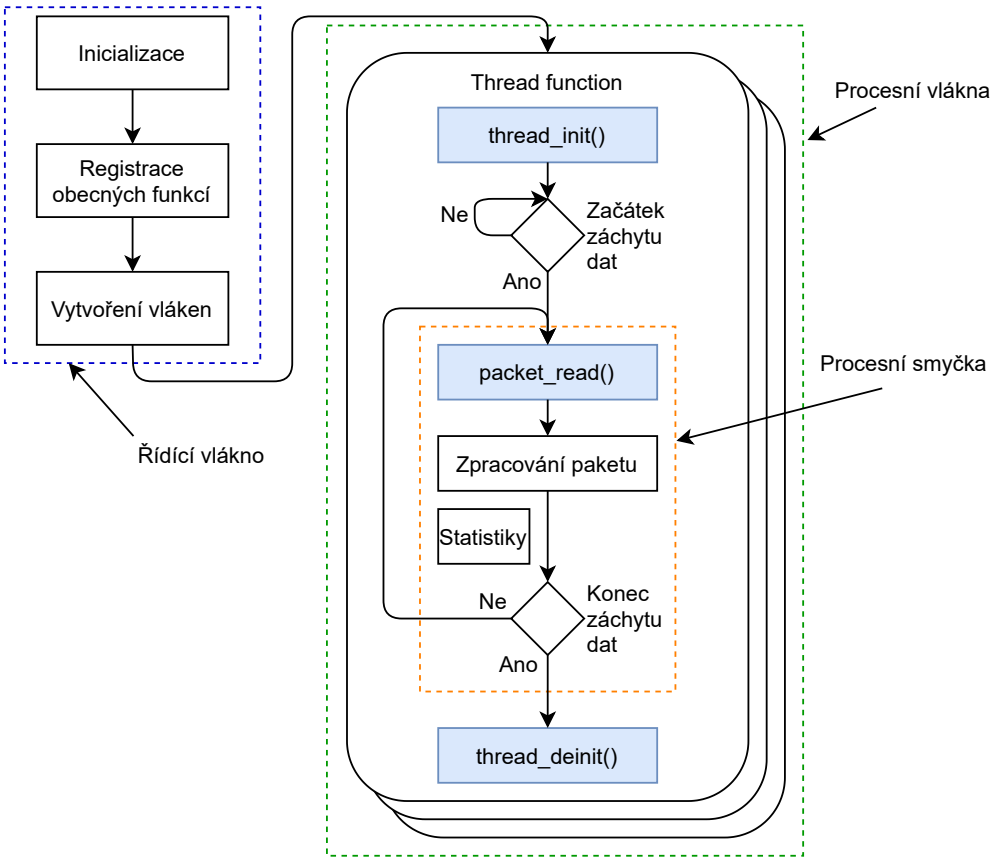
Zjednodušené schéma fungování třídy pro čtení paketů zobrazuje obrázek 7.1. Řídící vlákno provede inicializaci třídy, kde se na základě administrátorem zadané konfigurace zaregistrují obecné funkce zvoleného vstupního rozhraní. V dalším kroku se vytvoří zvolený počet procesních vláken, kde si každé vytvořené vlákno zavolá funkci `thread_init` a vytvoří si vlastní instanci vstupního rozhraní. Po inicializaci vstupního rozhraní proběhne synchronizace všech procesních vláken s řídícím vláknem, které po ověření, že všechna procesní vlákna jsou úspěšně inicializována, spustí zachytávání dat. Každé procesní vlákno se dostane to tzv. procesní smyčky, kde probíhá vyčítání paketů skrz obecnou funkci `packet_read` a jejich následné zpracování. Na konci každé iterace procesní smyčky je proveden test, zda řídící vlákno neukončilo záchyt dat. Pokud byl záchyt ukončen, provede se volání funkce `thread_deinit` a ukončí se práce se vstupním rozhraním a procesní vlákno se ukončí, v opačném případě se provede další iterace procesní smyčky.

Každé procesní vlákno si také udržuje statistiky o počtu zpracovaných paketů a bajtů, aby měl administrátor v případě potřeby přehled o tom, kolik dat jednotlivá procesní vlákna zpracovala. Při implementaci byl kladen důraz na efektivní alokaci potřebné paměti pro jednotlivé procesní vlákna, a to především proto, aby nedocházelo k vyplavování cache paměti procesoru mezi vlákny (tzv. *false sharing*). Dále byl kladen důraz také na to, aby alokované adresy začínaly na násobcích velikosti cache line procesoru, které zajistí minimální možný počet přístupů do hlavní paměti.

7.2 Implementace parseru paketů

Parser paketů implementuje třída `Packet_parser`, která poskytuje funkci `parse` pro extrakci dat z hlaviček příchozích paketů. Prototyp této funkce je vypsán v následujícím kódu. Vstupním parametrem je struktura `Raw_packet` obsahující ukazatel na začátek příchozího paketu a velikost paketu v bajtech. Výstupním parametrem je struktura `Packet`, která obsahuje vyextrahované informace z hlaviček paketu. Funkce skrz datový typ `Parser_code` vrací výsledek zda se parsování paketu podařilo či nikoliv.

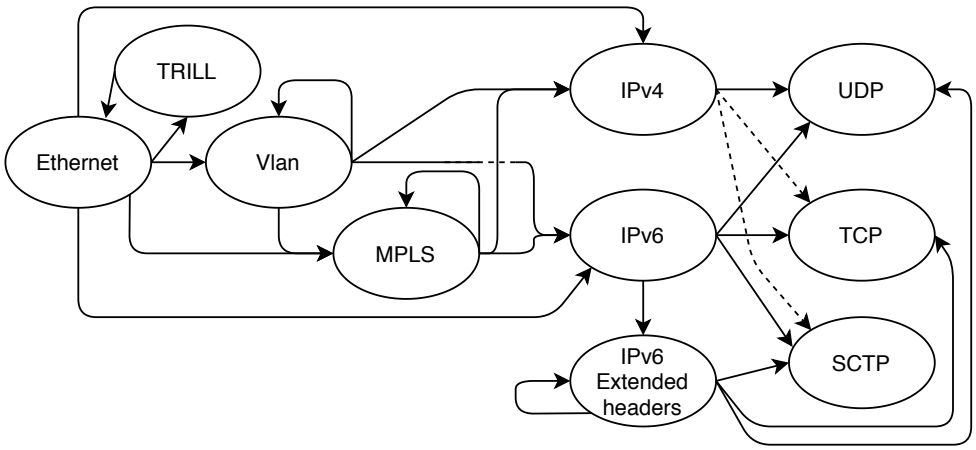
```
Parser_code parse(const Raw_packet& in_packet, Packet& packet);
```



Obrázek 7.1: Zjednodušené schéma fungování třídy pro čtení paketů.

Proces parsování paketu je zobrazen na obrázku 7.2. Začátek parsování je vždy v Ethernetové hlavičce, po které může následovat některý z následujících protokolů (TRILL, Vlan, MPLS, IPv4 nebo IPv6). V případě, že protokol následující za ethernetovou hlavičkou nespadá do jmenovaného výčtu, skončí parsování paketu neúspěchem. Další možností jak parsování paketu může skončit neúspěchem, je překročení limitu počtu hlaviček daného typu. Konkrétně se jedná o limity pro Vlan (4), MPLS (4), TRILL (1) a rozšiřující IPv6 (2) hlavičky, hodnoty uvedené v závorkách značí maximální počet parsovaných hlaviček daného typu. Toto omezení je vytvořeno s ohledem na požadavek vysoké datové propustnosti a větší než povolené množství parsovaných hlaviček by vyžadovalo více přístupů do paměti pro načtení dat paketu, což je časově velmi náročná operace. Poslední možností jak může parsování skončit neúspěchem je poškozený paket, konkrétně snaha o parsování dat mimo velikost paketu.

Extrahovaná data se ukládají do struktury `Packet`, jejíž prototyp je vypsán v následujícím kódu. První extrahovanou hodnotou je Vlan identikátor. V pří-



Obrázek 7.2: Diagram popisující parsování paketu.

padě, že paket obsahuje více Vlan hlaviček, je extrahován identifikátor z první, tedy nejvnějšejší Vlan hlavičky. Další extrahované informace jsou z IPv4 nebo IPv6 hlavičky, jedná se o zdrojovou a cílovou IP adresu, číslo protokolu následující hlavičky a délku payloadu. Položka zdrojového a cílového portu je extrahována v případě, že se za IP hlavičkou nachází některý z protokolů obsahující porty, tedy UDP, SCMP nebo TCP. Pokud se v paketu nachází poslední jmenovaný protokol, extrahuje se navíc ještě informace o tcp příznacích. V případě, že příchozí paket některou z extrahovaných položek neobsahuje, je daná položka nulována.

```
struct Packet {
    ip_addr_t src_ip;
    ip_addr_t dst_ip;
    uint8_t protocol;
    uint8_t tcp_flags;
    uint16_t src_port;
    uint16_t dst_port;
    uint16_t length;
    uint16_t vlan_id;
};
```

7.3 Implementace paketového filtru

Implementaci paketového filtru zajišťuje třída `Packet_filter`, která poskytuje dvě veřejné funkce. První funkcí je `parse_filter_file`, která se stará o nahrání administátorem zadaných pravidel z XML souboru. Pro čtení XML souboru se používá C++ knihovna *rapidxml* a výsledná pravidla jsou ukládána

do struktury *std::vector*, která zajistí, že všechna data jsou uložena fyzicky v jednom bloku paměti, čímž se eliminuje počet přístupů do hlavní paměti.

Druhou veřejnou funkcí je funkce `match`, jejíž prototyp je zobrazen v následující ukázce kódu. Tato funkce slouží ke zjištění toho, zda příchozí paket odpovídá některému z administrátorem zadaných pravidel. Vstupním parametrem této funkce je struktura `Packet` s vyextrahovanými informacemi z paketu. Paket je ve funkci postupně porovnáván s filtračními pravidly podle algoritmu 6.2, který byl popsán v návrhu. Návratová hodnota funkce udává výsledek porovnání, která je v případě shody paketu s filtračním pravidlem nastavena na hodnotu *true* a značí propuštění paketu dále k analýze. V opačném případě je rozhodnuto o zahození paketu.

```
bool match(Packet& packet);
```

7.4 Implementace sketchů

Sketche jsou implementovány pomocí šablonované třídy `Sketch<T>`, kde šablona T udává typ použitého sketche (Count-Min nebo Count-Median), který tato třída dědí. Skrz tuto třídu tak lze přistupovat k funkcím implementovaných sketchů bez ohledu na jejich typ. Při profilování výpočetní náročnosti sketche bylo zjištěno, že nejvíce výpočetně náročnou operací, která se nad sketchem provádí, je několikanásobný výpočet hashe, který mapuje vstupní položku na index čítače v každém řádku. Proto byla pro výpočet hashe zvolena vysoce výkonná funkce XXH3 [32].

XXH3 je nekryptografická 64 bitová hash funkce. Zvolení právě této funkce vychází z benchmarku popsaném v tabulce 7.1. Benchmark používá nástroj SMHasher a testuje hash funkce a kontrolní součty na distribuci hodnot, množství kolizí a výkonnost. Sloupec s kvalitou určuje bodové hodnocení v prvních dvou testech, kde hodnota 10 je nejvyšší možná a udává nejlepší výsledek.

Tabulka 7.1: Výsledky testů funkcí v SMHasher nástroji. Tabulka je převzatá z [32].

Jméno	Rychlost	Kvalita	Šířka dat
XXH3	31,5 GB/s	10	64 b
XXH128	29,6 GB/s	10	128 b
City64	22,0 GB/s	10	64 b
T1ha2	22,0 GB/s	9	64 b
City128	21,7 GB/s	10	128 b
XXH64	19,4 GB/s	10	64 b
XXH32	9,7 GB/s	10	32 b

Při měření výkonnosti nástroje, které je blíže popsáno v kapitole Výsledky se ukázalo, že základní varianty sketchů jsou pro zpracování maximálního možného počtu paketů (148 milionů paketů) výkonnostně nedostatečné, a to i přes zvolenou vysoce rychlou hashovací funkci. Z toho důvodu byla na základě článku [31] implementována do sketchí rychlejší funkce `update`.

Hlavní datovou strukturou obou implementovaných sketchů je dvourozměrné pole čítačů, které bylo implementováno jako jednorozměrné pole s mapováním indexů do 2D prostoru. Tento přístup byl zvolen proto, aby se zlepšila lokalita dat a snížil se počet přístupů do paměti. Velikost čítačů byla nastavena na 64 bitů, protože 32 bitový čítač by mohl při velkém množství provozu přetéct. Pro zvýšení výkonnosti bylo povoleno nastavení šířky řádku sketche pouze na mocniny čísla dvě, a to z důvodu nahrazení výpočetně náročné operace modulu při mapování hashe na index čítače logickou operací AND, která lze provést pouze za podmínky, že je šířka řádku mocnina dvou. Výpočet indexu čítače v řádku je ukázán v následující ukázce kódu.

```
uint32_t line_index = hashval & (width - 1);
```

Jednotlivé sketche implementují funkce, jejichž prototyp je vypsán v následující ukázce kódu.

```
uint64_t update(const void *key, const uint32_t count);

bool update_nitro(const void *key, const uint32_t count);

uint64_t estimate(const void *key);
```

Funkce `update` má dva vstupní parametry, a to klíč (vstupní prvek) a hodnotu, která má být přičtena k čítačům daného klíče. Vzhledem k tomu, že se po aktualizaci sketche provádí volání funkce update nad strukturou Heavy hitteru, která vyžaduje znalost aktuální frekvence prvku, byla funkce `update` upravena tak, že skrz návratovou hodnotu vrací aktuální frekvenci prvku, tedy funkce provede zároveň operaci update i estimate. Toto řešení ušetří volání funkce `estimate`, což má pozitivní vliv na výkonnost, neboť se nebude muset počítat znovu několikanásobně hash.

Funkce `update_nitro` implementuje variantu funkce update s využitím myšlenky NitroSketche. Tato funkce je volána nad všemi pakety, které byly propuštěny k analýze a rozhodnutí o tom, zda se paket na základě nastaveného vzorkování přeskočí či nikoliv, je učiněno v této funkci. Struktura sketche si při využití varianty NitroSketche udržuje dva čítače, které určují kolik paketů má být přeskočeno před další aktualizací a index příštího řádku v poli čítačů, který bude aktualizován. Výpočet hodnot ukazuje následující ukázka kódu:

```
next_counter_index += samp_rate;
packets_to_skip = next_counter_index / depth;
next_counter_index = next_counter_index % depth;
```

Funkce skrz návratovou hodnotu vrací informaci o tom, zda vstupní paket způsobil aktualizaci čítače nebo byl přeskočen. To je důležité pro následné volání funkce update nad strukturou Heavy hitteru, u které není nutné, aby byla volána v případě, kdy ve sketchi nenastala žádná změna.

Poslední důležitou implementovanou funkcí je funkce `estimate`, která vrací odhad aktuální frekvence předaného klíče.

7.5 Implementace heavy hitters

Implementaci heavy hitters problému poskytuje třída `Heavy_hitters`, která v sobě dokáže efektivně uložit až *TOP-N* nejvíce frekventovaných položek. K ukládání dat je použita hash mapa *ska::flat_hash_map*[10], která funguje na bázi *Robin Hood*[11] hashování. Jako hashovací funkce je zvolena rychlá XXH3, která byla popsána v sekci 7.4. Hlavní důvod pro použití hash mapy při implementaci je vyhledávání, vkládání a mazání položek v $\mathcal{O}(1)$ konstantním čase.

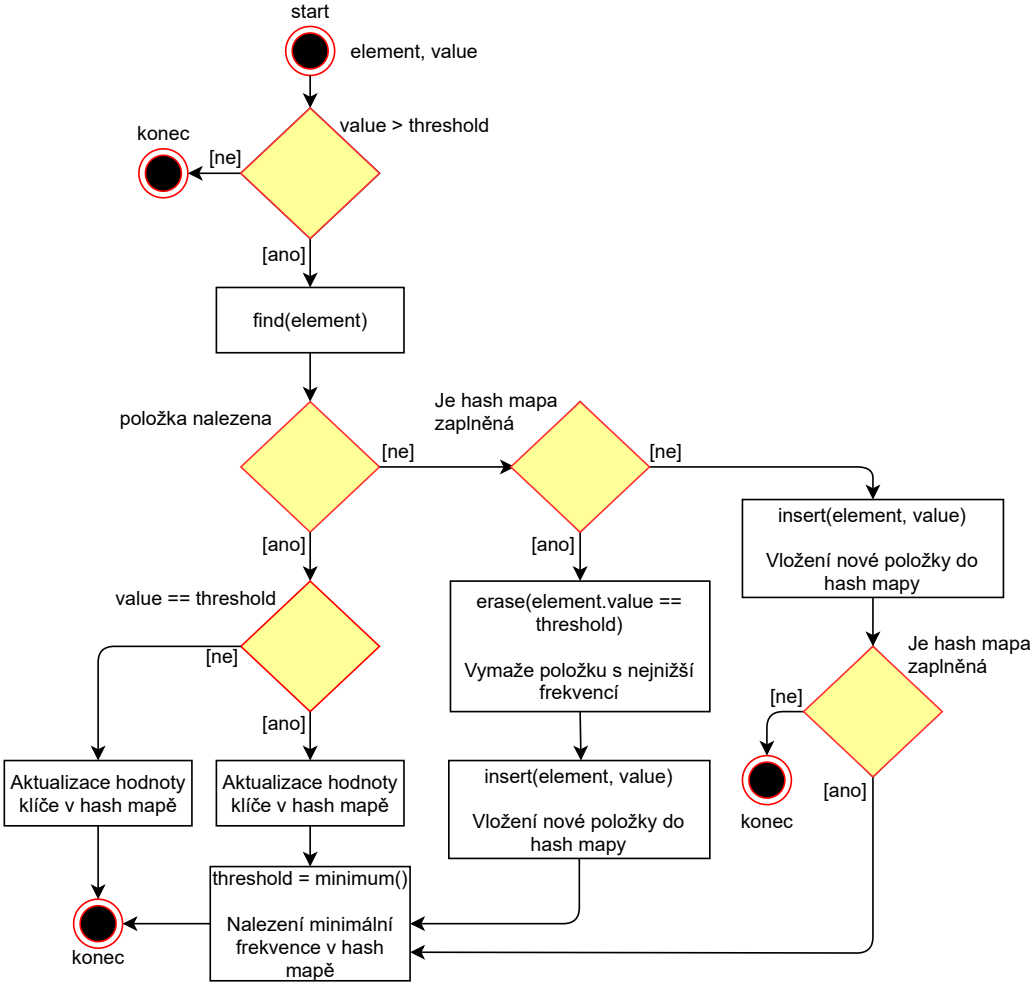
Třída `Heavy_hitters` poskytuje funkce, jejichž prototyp je zobrazen v následující ukázce kódu.

```
template<typename T>
void update(T element, uint64_t value);

template<typename T>
std::vector<std::pair<T, uint64_t>> sort();
```

Funkce `update` slouží k aktualizaci vnitřní datové struktury (hash mapy). Klíčem do hash mapy je šablonovaný parametr *element* a k němu přidružená hodnota *value* udávající aktuální frekvenci klíče. Vývojový diagram funkce ukazuje obrázek 7.3. Frekvence vstupní položky je hned na začátku funkce porovnávána vůči proměnné *threshold*, jejíž hodnota je nastavena při plném zaplnění hash mapy (TOP-N položek) a udává jakou nejnižší možnou frekvenci musí vstupní položka mít, aby provedla v uložených datech změnu. Tímto způsobem lze odfiltrovat velké množství dat s nízkou frekvencí, což má pozitivní vliv na výkonnost. Výchozí hodnota položky *threshold* je nastavena na nulu, a proto než dojde k zaplnění hash mapy, je vždy podmínka splněna. Po splnění podmínky se pokusí vyhledat vstupní položka v hash mapě. Pokud je položka nalezena, provede se aktualizace k ní přidružené frekvence a pokud původní frekvence položky odpovídala hranici *threshold*, provede se vyhledání nového minima. Hledání minima se provádí sekvenčním procházením položek uložených v hash mapě. V případě, že vstupní položka v hash mapě nalezena nebyla, je provedena kontrola zaplněnosti hash mapy. Pokud je hash mapa zcela zaplněná, provede se odstranění položky s nejnižší frekvencí, která je

[10] https://github.com/skarupke/flat_hash_map
[11] https://programming.guide/robin-hood-hashing.html

následně v hash mapě nahrazena vstupní položkou a k ní přidruženou frekvencí. Po vložení nového prvku je opět provedena aktualizace hodnoty *threshold*. Pokud hash mapa zcela zaplněná není, provede se vložení nové položky do hash mapy a v případě, že tímto vložením došlo k zaplnění tabulky, dojde k prvotnímu nastavení hodnoty hranice *threshold*.



Obrázek 7.3: Diagram popisující funkci update u heavy hitters problému.

Druhou funkcí, kterou třída poskytuje je funkce `sort`, která pomocí datové struktury std::vector vrací seřazené položky a k nim přiřazené frekvence. Položky jsou seřazeny sestupně a to podle hodnoty frekvence.

7.6 Implementace režimu struktury provozu

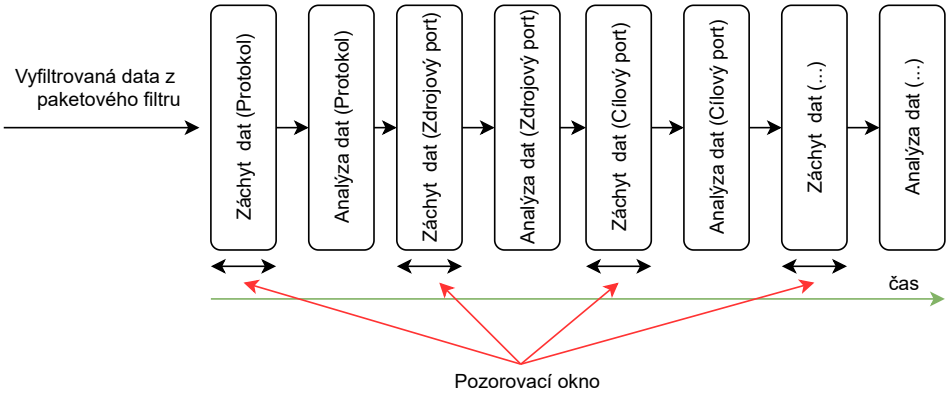
Režim analýzy struktury provozu je implementován šablonovanou třídou `Traffic_structure<Sketch_type>`, kde šablona Sketch_type udává typ použitého sketche pro ukládání dat.

Při testování výkonnosti nástroje bylo zjištěno, že provádění výpočtu pro všechny vstupní položky paketu (tabulka 6.1) ve stejném pozorovacím okně, je výpočetně velmi náročné a s ohledem na požadavek vysoké datové propustnosti a přijatelné míře chybovosti i nemožné. Pro příklad, pokud by paket obsahoval všechny vstupní položky, kterých je celkem 8 a hloubka sketche by byla zvolena na hodnotu 5 (míra jistoty 99.6 % u CM sketche, že nenastane chyba větší než zvolená mez), znamenalo by to, že analýza jednoho paketu by vyžadovala čtyřicetinásobný výpočet hashe ve sketch struktuře (*počet položek* $\times$ *hloubka sketche*) a až osminásobnou aktualizaci heavy hitters struktury pro každou analyzovanou položku zvlášť. Při takovéto výpočetní zátěži není možné dosáhnout vysoké datové propustnosti.

Pro řešení tohoto problému byl zvolen přístup postupné analýzy jednotlivých položek, který je zobrazen na obrázku 7.4. Analýza je rozdělena do více pozorovacích oken a v každém pozorovacím oknu je analyzována jiná položka paketu. Takto jsou iterace záchytu a analýzy dat opakovány, dokud nejsou zpracovány všechny vstupní položky. Tento přístup sebou přináší několik výhod. Jednou z výhod je snížení paměťové náročnosti, protože je potřeba současně udržovat Heavy hitters strukturu pouze pro jednu, zrovna analyzovanou položku. Datové struktury jsou pro všechny analyzované položky sdílené a není potřeba vytvářet pro každou iteraci nové. Další výhodou je snížení množství kombinací hodnot, kterých mohou analyzované položky nabývat a tím i potenciální snížení počtu kolizí ve sketchi. Nevýhodou tohoto řešení je, že záchyt dat neprobíhá ve stejný časový okamžik, ale v krátkých časových intervalech jdoucích za sebou. To způsobí, že je vždy zobrazena aktuální struktura provozu pro zrovna analyzovanou položku, ale vzhledem k tomu, že podle vzoru z DDoS Protectoru se předpokládá s velikostí pozorovacího okna v řádu jednotek vteřin, lze tuto nevýhodu akceptovat.

Třída `Traffic_structure` v sobě obsahuje množinu instancí obecné třídy `Mode_data<Sketch_type>`, kde každá instance reprezentuje data jednoho procesního vlákna. Každé procesní vlákno v sobě obsahuje ukazatel na funkci `process_packet`, kterou třída `Mode_data` implementuje. Tento ukazatel je skrz funkci `std::bind()` svázán s danou instancí třídy `Mode_data`, nad kterou je funkce `process_packet` volána. Funkce je volána v případě, že paket prošel skrz paketový filtr dále k analýze. Ukazatel na funkci a prototyp funkce je ukázán v následující ukázce kódu.

```
std::function<void(Packet&)> process_packet;

void process_packet(Packet& packet);
```



Obrázek 7.4: Postupný záchyt a analýza položek u režimu struktury provozu.

Funkce `process_packet` ze vstupní struktury `packet` vyjme aktuálně zpracovávanou položku a zavolá nad její hodnotou funkci `update` nebo `update_nitro` (NitroSketch), podle zvolené konfigurace. Zdrojový kód zmíněných dvou funkcí je zobrazen v následující ukázce. Vstupní parametr `value` je reprezentován datovým typem uint128_t, který v sobě dokáža pojmou jakoukoli analyzovanou položku, včetně IPv6 adres. Druhý parametr count pak udává hodnotu jaká má být přičtena k čítači frekvence ve sketchi. Mód struktury provozu podporuje zobrazení zastoupení provozu pro analyzovanou hodnotu položky podle počtu paketů nebo počtu bajtů. V případě zobrazování podle počtu paketů je hodnota vstupního parametru `count` vždy nastavena na hodnotu 1, v případě zobrazení podle bajtů nabývá parametr `count` délky paketu uvedené v IP hlavičce. Funkce následně zavolá nad vstupními parametry funkci `update` nad strukturou sketche, následovanou funkcí `update` nad strukturou Heavy hitters s aktuální frekvencí. V případě volání varianty `update_nitro` je aktualizace sketche a Heavy hitters podmíněna podle principu popsaného v sekci 5.4.

```
void update(uint128_t value, uint32_t count)
{
        uint64_t counter;
        counter = sketch.update(&value, count);
        hh.update(value, counter);
        hh.total_sum += count;
}

void update_nitro(uint128_t value, uint32_t count)
{
        uint64_t counter;
        bool updated = sketch.update_nitro(&value, count);
        if (updated) {
                if (get_random_number() < 1) {
```

```
                    counter = sketch.estimate(&value);
                    hh.update(value, counter);
                }
                hh.total_sum += count;
        }
}
```

Po ukončení pozorovacího okna a zastavení záchytu dat je nad třídou `Traffic_structure` zavolána z řídícího vlákna funkce `analyze`, která vyhodnotí zachycená data a zobrazí výsledek analýzy aktuální položky z paketu na výstup. Funkce nejprve nad množinou instancí třídy `Mode_data`, které reprezentují data z procesních vláken provede funkci merge, která spojí data ze všech procesních vláken do jedné instance. Nejprve se provede merge nad strukturou sketche a následně i nad strukturou Heavy hitters. Položky uložené ve sloučené struktuře Heavy hitters jsou následně seřazeny podle jejich frekvence a následně jsou postupně vypisovány na standardní výstup ve formátu následující ukázky výpisu.

```
#########################################################
Src port    [Packets]       [Approx. Packets]   [Total %]
1025         43041604           43041604          14.47%
1026         43039994           43039994          14.47%
1024         43033381           43033381          14.47%
1029         10009161           10009161           3.37%
1032         10007551           10007551           3.37%
```

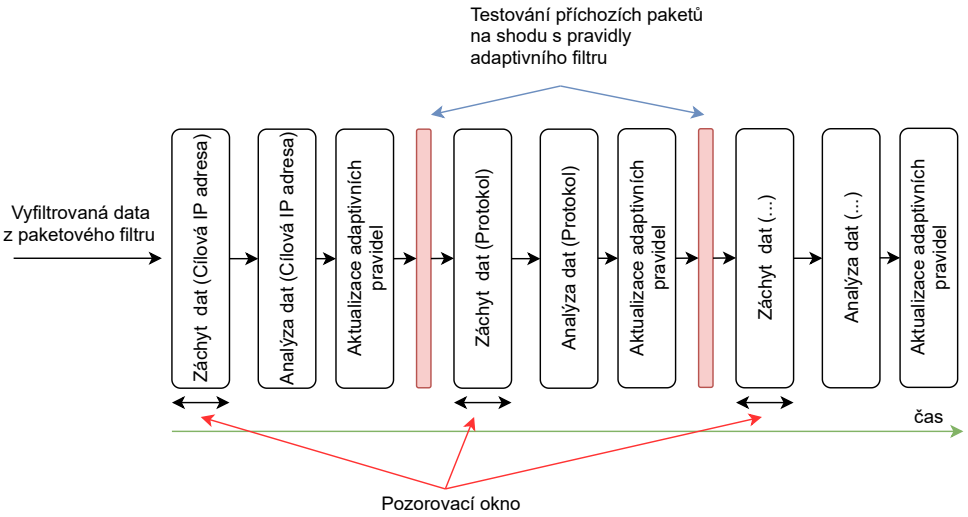
Obrázek 7.5: Ukázka výpisu z módu struktury provozu.

První sloupec udává typ aktuálně analyzované položky. Druhý sloupec je pojmenovaný podle toho, zda jsou počítány pakety či bajty a vyjadřuje frekvenci položky získanou ze sketche, ta může být v případě využití varianty NitroSketche zkreslená, proto je zde třetí sloupec, který na základě hodnoty zvoleného samplování vstupních dat odhadne reálnou hodnotu položky. Poslední sloupec vyjadřuje zastoupení dané položky v celkovém objemu analyzovaných dat.

Po zobrazení výsledku je řídícím vláknem volána funkce `prepare_next_element`, která připraví prostředí pro analýzu další položky paketu. To obnáší uvedení datových struktur procesních vláken do výchozího stavu a nastavení položky, která má být analyzována. Po dokončení funkce se začne provádět nová iterace analýzy a je opět spuštěn záchyt dat.

7.7 Implementace režimu doporučení mitigačních pravidel

Režim doporučení mitigačních pravidel je implementován šablonovanou třídou `Rules_recommendation<Sketch_type>`, kde šablona Sketch_type udává typ použitého sketche pro ukládání dat. Schéma zpracování a analýzy dat v tomto režimu je ukázáno na obrázku 7.6.



Obrázek 7.6: Schéma záchytu a zpracování dat v režimu doporučení pravidel.

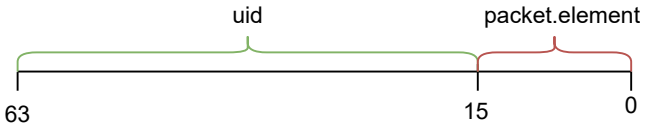
Schéma je podobné jako u režimu struktury provozu, záchyt a analýza dat opět probíhá v několika iteracích, kde v každé je zpracována jiná položka paketu. Hlavním rozdílem, oproti předchozímu režimu je využití Adaptivního paketového filtru, který propouští k uložení do sketche jen ty pakety, které měly shodu s některým z adaptivních pravidel. Tato pravidla vyjadřují, jakou strukturu má provoz splňující minimální míru zastoupení provozu směřujícího do cílové sítě.

Adaptivní paketový filtr je implementován třídou Addaptive_rules, která poskytuje následující funkce:

```
void add_element(Element_type type, ip_addr_t ip);

void add_element(Element_type type, uint64_t sketch_key);

bool match(Packet& packet, uint64_t& rule_uid, bool& match_dst_net);

void show_rule(Element_type type, uint64_t rule_uid);
```

Funkce `add_element` slouží k přidání položky paketu, která je specifikována parametrem `type`, do filtračního pravidla. Jako první je vždy uložena cílová

IP adresa, která je výchozím filtračním pravidlem. Jak bylo popsáno v sekci návrhu, algoritmus nejprve vyhodnotí nejvíce zastoupené cílové IP adresy, které pak postupně zpracovává. Funkce si pro každou uloženou položku (a tedy i nové pravidlo) vytvoří mapování na unikátní 48 bitovou hodnotou, která reprezentuje všechny položky pravidla. Parametr funkce `sketch_key`, který je reprezentován 64 bitovou hodnotou pak v sobě obsahuje jak identifikátor pravidla, tak hodnotu položky paketu, která má být do pravidla přidána. Všechny položky paketu, které jsou pro cílovou síť analyzovány jsou reprezentovány maximálně 16 bitovou hodnotou, která dává v součtu s identifikátorem pravidla 64 bitů. Ukázku rozložení parametru `sketch_key` zobrazuje obrázek 7.7.



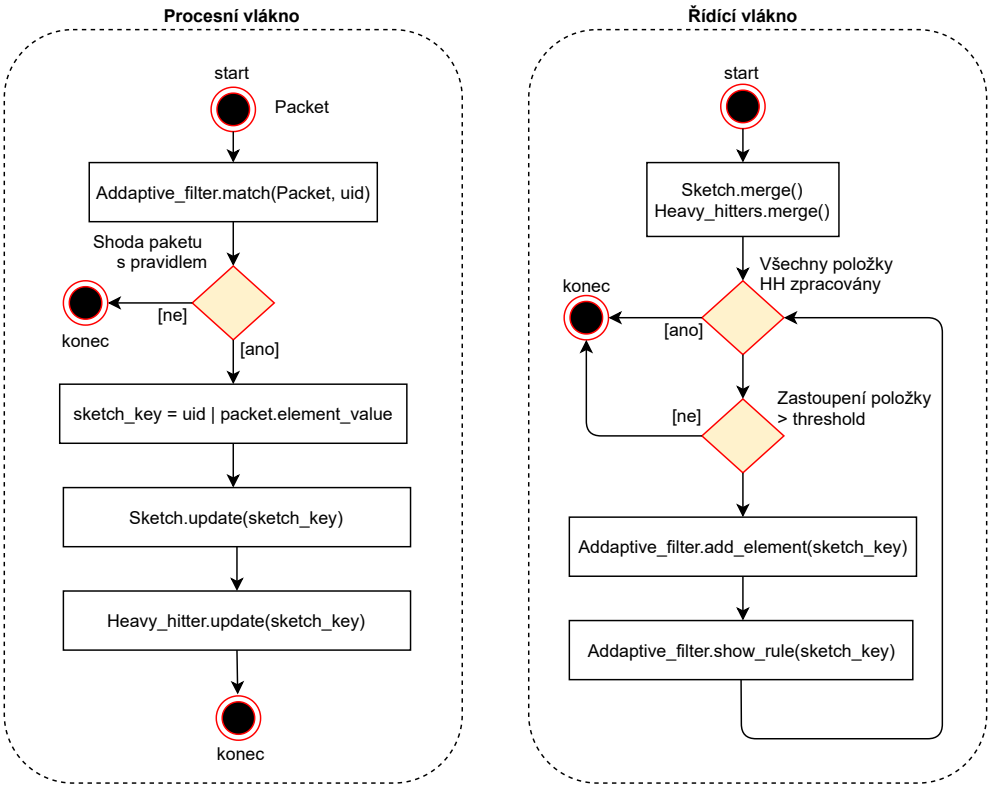
Obrázek 7.7: Reprezentace klíče do sketche.

Identifikátor pravidla a aktuální hodnota analyzované položky paketu také tvoří klíč pro uložení do struktury sketche, a to z důvodu jednoznačné identifikace, ke kterému pravidlu se hodnota analyzované položky vztahuje. V případě, kdy by adaptivní filtr obsahoval dvě následující pravidla a na vstupu by byly pakety např. se stejným zdrojovým portem, které se vztahují k některému z pravidel, nebylo by možné v případě ukládání pouze hodnoty zdrojového portu tato dvě pravidla rozlišit.

```
dst net 145.50.41.12/32 and protocol 17
dst net 145.50.41.12/32 and protocol 6
```

Funkce `match` se používá ke zjištění, zda vstupní paket odpovídá některému z pravidel adaptivního filtru. Funkce v případě nalezení shody paketu s pravidlem předá skrz parametr `rule_uid` identifikátor daného pravidla, který je následně použit při vytváření klíče do sketche. Parametr `match_dst_net` pak předává informaci, zda paket měl jako cílovou IP adresu nastavenou aktuálně analyzovanou cílovou IP adresu, aby bylo možné udržovat čítače o tom, kolik provozu na zadanou IP adresu směřuje a následně se mohla určit míra zastoupení analyzovaných položek paketu.

Třída `Rules_recommendation` v sobě obsahuje, stejně jako u předchozího režimu, množinu instancí obecné třídy `Mode_data<Sketch_type>`, kde každá instance reprezentuje data jednoho procesního vlákna. Instance jsou s procesními vlákny provázané stejným způsobem, jaký byl popsán u předchozího režimu. Jedinou změnou u tohoto režimu je využití adaptivního paketového filtru ve funkci `process_packet`, kterou třída `Mode_data` implementuje. Zpracování paketu ve funkci `process_packet` a analýzu dat v řídícím vlákně popisuje vývojový diagram 7.8.



Obrázek 7.8: Vývojový diagram popisující zpracování dat v procesním a řídícím vláknu.

Levá část vývojového diagramu popisuje zpracování jednoho paketu ve funkci `process_packet` v procesním vláknu. Paket je v prvním kroku porovnán s pravidly adaptivního filtru a pokud je nalezena shoda, vytvoří se klíč, který se použije pro přístup do sketche. Klíč je vytvořen operací OR nad identifikátorem pravidla a hodnotou aktuálně zpracovávané položky paketu. Pomocí tohoto klíče jsou následně aktualizovány čítače ve sketchi a i struktura Heavy hitteru. Tento proces je opakován dokud nedojde k pozastavení záchytu dat řídícím vláknem a nepřejde se na jejich analýzu.

Pravá část diagramu ukazuje proces analýzy dat řídícím vláknem. Tento proces začíná ve chvíli, kdy řídící vlákno pozastaví záchyt dat procesním vláknům. V prvním kroku se provede funkce `merge` nad datovými strukturami ze všech procesních vláken. Poté jsou postupně zpracovávány nejvíce zastoupené hodnoty aktuálně analyzované položky paketu. Každá položka je testována, zda její hodnota (frekvence) splňuje hranici minimálního procentuálního zastoupení nad objemem provozu směřujícího na cílovou adresu. V případě, že je podmínka splněna, je pravidlo, ke kterému se položka paketu vztahovala o tuto položku rozšířeno. Posledním krokem je vypsání do-

poručeného mitigačního pravidla na výstup. Formát pravidla je ukázán v následujícím výpisu.

```
Recommended rule:
"dst net 192.0.0.1/32 protocol 6 src port 1024 length 110".
This rule should block 47.63% of traffic heading to destination
network.
```

Obrázek 7.9: Ukázka výpisu doporučeného mitigačního pravidla z módu doporučení pravidel.

Výpis obsahuje pravidlo ve formátu vstupních pravidel DDoS Protectoru (bez položky threshold a limit) a také informaci o tom, kolik procent z celkového objemu provozu směřujícího na cílovou adresu by toto pravidlo mělo pokrývat. Těchto pravidel může být vypsáno více, v závislosti na zvolené hranici zastoupení a struktuře provozu.

7.8 Implementace interaktivního režimu

Tento režim je implementován třídou `Interactive_mode`, která v sobě obsahuje ukazatele na třídu režimu struktury provozu a doporučení pravidel. Na základě administrátorova vstupu je vybrán jeden z režimů, jehož ukazatel je následně inicializován a je tak vytvořena instance požadovaného režimu.

Režim rozšiřuje třídu `Traffic_structure` a `Rules_recommendation` o funkci, jejíž prototyp je zobrazen v následující ukázce kódu.

```
bool prepare_next_element_interactive_mode();
```

Tato funkce v sobě oproti původní funkci `prepare_next_element` obsahuje rozšíření o interakci s administrátorem, která určí následují krok algoritmu a tedy i přípravu prostředí na zvolený typ analyzované položky paketu. Funkce není volána přímo, ale je volána skrz třídu `Interactive_mode` nad instancí zvoleného režimu. Veškerá zbylá funkcionalita je prováděna ve třídě `Traffic_structure` a `Rules_recommendation` a tento režim ji nijak dál nemění.

Kapitola 8

Vyhodnocení

Tato kapitola obsahuje výsledky provedených testů, které byly zaměřeny především na vyhodnocení funkčnosti nástroje, propustnosti a množství použitých zdrojů.

8.1 Testovací prostředí

Veškerá měření probíhala na testovacím serveru, který byl osazen dvěma 10–jádrovými procesory Intel(R) Xeon(R) Silver 4114 CPU s pracovní frekvencí 2.20 GHz. Server disponoval 96 GB operační paměti typu DDR4 (2666 MHz) a FPGA síťovou kartou NFB-200G2QL. Karta byla v konfiguraci síťových portů 2x100 Gb/s.

Všechen testovací síťový provoz byl generován hardwarovým testerem Spirent TestCenter[12], který generuje libovolný síťový provoz, a to při rychlosti až 100 Gb/s.

Program byl zkompilován pomocí g++ (9.3.1) s parametry -Wall -Wextra -pedantic -O3 -flto -msse4.2 -std=c++2a.

8.2 Sketche

Porovnání implementovaných sketchů mezi sebou při stejném nastavení maximální chyby a míry jistoty není možné, a to proto, že u každého sketche tyto hodnoty mají jiný význam. U Count-Median sketche se chybovost vztahuje k L2 normě, zatímco u Count-Min sketche se vztahuje k L1 normě.

Sketche tak byly porovnávány, jak si při stejné velikosti pole čítačů dokáží poradit s různými variantami testovacího provozu, a to vzhledem k počtu nesprávných hodnot zobrazených na výstupu. Testování sketchů probíhalo v režimu struktury provozu. Veškeré testy probíhaly nad položkou zdrojové

[12] https://www.spirent.com/products/testcenter-platforms-software

IP adresy paketu, protože svým rozsahem dokáže pokrýt nejvíce vstupních hodnot.

První testovací případ sloužil ke zjištění počtu nesprávně zobrazených hodnot mezi nejfrekventovanějšími položkami, a to vzhledem k počtu vstupních hodnot (počet unikátních IP adres) a typu použitého sketche. Jako nesprávná hodnota se považovala IP adresa legitimního uživatele, která byla zobrazena mezi frekventovanými IP adresami. Struktura generovaného provozu je ukázána v tabulce 8.1. Pro jednoduchost je ukázána pouze položka, která byla analyzována.

Tabulka 8.1: Struktura testovacího provozu pro testování sketche.

Typ položky	**Nejvíce frekventované položky**	**Ostatní provoz**
Zdrojová IP adresa	192.0.0.0 - 192.0.1.255	10.0.0.1 - 10.0.63.254 (16384) 10.0.0.1 - 10.0.127.254 (32768) 10.0.0.1 - 10.0.255.254 (65536)
Zastoupení	50 %	50 %

Struktura provozu je rozdělena na dvě části. První část zastupuje 512 nejvíce frekventovaných zdrojových IP adres, reprezentující IP adresy útočníka. Tyto adresy zastupují 50 % z celkového objemu provozu. Zbylých 50 % je zastoupeno provozem legitimních uživatelů, který je tvořen celkem 16384, 32768 a 65536 různými zdrojovými IP adresami. Test byl spuštěn s výchozí šířkou sketche 2048 a hloubkou sketche 5. Výsledek testu zobrazuje tabulka 8.2.

Tabulka 8.2: Počet nesprávně uložených hodnot ovlivňující výstup sketche v závislosti na množství vstupních unikátních hodnot.

Typ sketche	**Počet nesprávných výsledků**					
Count-Min	2	1	1	0	0	0
Count-Median	24	30	27	0	0	0
NitroSketch (Cmin, 16, 8)	3	1	3	0	0	0
NitroSketch (Cmin, 32, 16)	2	3	5	0	0	0
NitroSketch (Cmedian, 16, 8)	24	30	19	0	0	0
NitroSketch (Cmedian, 32, 16)	17	31	30	2	0	0
Počet IP adres legitimních uživatelů	8K	16K	65K	8K	16K	65K
TOP-N	512			20		

Z tabulky 8.2 je vidět, že varianty používající Count-Min sketch poskytují při stejné velikosti dvojrozměrného pole čítačů přesnější výsledky tím, že obsahují méně falešně zobrazených frekventovaných hodnot. Hlavní rozdíl je viditelný při zobrazení všech 512 nejvíce frekventovaných hodnot, kdy varianty používající Count-Median sketch měly několikanásobně vyšší počet falešně zobrazených hodnot. Varianty NitroSKetche, jejichž hodnoty uvedené v závorce vyjadřují vzorkování dat ve sketchi a vzorkování dat v Heavy hitteru, poskytují přibližně stejné výsledky, jako základní varianty sketchů. Při zobrazení pouze 20 nejvíce frekventovaných hodnot dokázaly všechny měřené varianty kromě *NitroSketch(Cmedian, 32, 16)* zobrazit správné výsledky bez falešně zastoupených hodnot.

Druhý testovací případ se zabýval vlivem velikosti sketche na počtu falešně zobrazených hodnot. Struktura testovacích (tabulka 8.1) dat byla stejná jako v předchozím testovacím případu. Test byl spuštěn s hodnotou TOP-N 512 a počet IP adres legitimních uživatelů byl 65536. Výsledek testu zobrazuje tabulka 8.9.

Tabulka 8.3: Počet nesprávně uložených hodnot ovlivňující výstup sketche v závislosti na velikosti sketche.

Typ sketche	**Počet nesprávných výsledků**					
Count-Min	13	3	1	0	0	0
Count-Median	21	22	15	2	2	0
NitroSketch (Cmin, 16, 8)	22	5	2	0	0	0
NitroSketch (Cmin, 32, 16)	31	6	4	0	0	0
NitroSketch (Cmedian, 16, 8)	46	22	20	5	4	3
NitroSketch (Cmedian, 32, 16)	84	26	28	6	6	5
Šířka, výška	1024,5	2048,4	2048,5	2048,6	4096,4	4096,5

Tabulka 8.9 ukazuje, že se zvětšujícím se počtem čítačů klesá počet uložených falešně frekventovaných položek. Varianty s Count-Min sketchem opět vychází v porovnání s variantami Count-Median sketche mnohonásobně lepé a již od velikosti sketche (2048, 6) zobrazují výsledky bez falešně frekventovaných hodnot.

8.3 Paměťová náročnost

Paměťová náročnost implementovaného algoritmu je důležitá především s ohledem na efektivní využití cache paměti. Vysoká paměťová náročnost má za

následky časté výpadky cache paměti a nucený přístup do hlavní paměti, který je výpočetně velmi náročný a celý algoritmus tak zpomaluje.

Paměťová náročnost implementovaného řešení zahrnuje strukturu sketche a strukturu Heavy hitteru, jejichž paměťová náročnost je popsána v následujících odrážkách:

- **Count-Min sketch** je parametrizován šířkou w a hloubkou d udávající velikost dvourozměrného pole čítačů, kde každý čítač má velikosti 64bitů. Dále si sketch udržuje 64 bitové pole o velikosti d položek, ve kterém jsou uloženy seedy hashovacích funkcí. Celkově tak Count-Min sketch ke své práci potřebuje $(w \times d + d) * 8$ bajtů paměti. Paměťová náročnost u varianty NitroSketche je stejná.

- **Count-Median sketch** je také parametrizován šířkou w a hloubkou d udávající velikost dvourozměrného pole čítačů, kde každý čítač má velikost 64 bitů. Sketch si v sobě udržuje dvě 64 bitové pole o délce d, kde každé ukládá seed jedné ze dvou hashovacích funkcí daného řádku. Navíc ještě ke své práci potřebuje další 64 bitové pole o velikosti d, které slouží k dočasnému výsledku při výpočtu mediánu. Celkově tak Count-Min sketch ke své práci potřebuje $(w \times d + 3 \times d) * 8$ bajtů paměti. Paměťová náročnost u varianty NitroSketche je opět stejná.

- **Heavy hitters** je parametrizován hodnotou n, udávající maximální počet udržovaných nejvíce frekventovaných položek a hodnotou k, udávající velikost klíče pro uložení dat. Velikost klíče je pro všechny položky paketů shodných 128 bitů a to z důvodu, aby nemuselo během analýzy jednotlivých položek paketu docházet k realokaci paměti. Struktura v sobě obsahuje hashovací mapu `flat_hash_map`, jejíž paměťová náročnost je závislá na velikosti klíče k, velikosti dat c, a počtu uložených položek. Velikost klíče je tedy 128 bitů a velikost dat je nastavena na 64 bitů reprezentující hodnotu čítače ze sketche. V hashovací mapě je zarezervováno místo pro n položek. Pro uložení n položek hashovací mapa vyžaduje $2 \times N \times (k + c)$ paměti, kde N je následující mocnina dvou od hodnoty parametru n. Celkem tak `flat_hash_map`, ale i celá struktura Heavy hitteru vyžaduje ke své práci $48 \times N$ bajtů paměti.

Při výchozích parametrech nástroje, to je šířce sketche 2048, hloubce sketche 5 a počtu udržovaných hodnot v Heavy hitteru 10, je celková paměťová náročnost jednoho procesního vlákna následující:

- Count-Min sketch = $(2048 \times 5 + 5) * 8 = 81960$ bajtů.
- Count-Median sketch = $(2048 \times 5 + 3 \times 5) * 8 = 82040$ bajtů.
- Heavy hitters = $(48 \times 16) = 768$ bajtů.

Jak ukazuje tabulka 8.2 obě varianty sketchů v sobě při této velikosti dokáží efektivně uložit alespoň 66048 IP adres (útočník + legitimní uživatelé), a to bez zobrazení falešně frekventovaných položek pro hodnotu TOP-N 10. Následující tabulka 8.4 ukazuje porovnání s exaktní metodou řešení tohoto problému.

Tabulka 8.4: Porovnání paměťové náročnosti Count-Min a Count-Median sketche s exaktní metodou řešení.

Typ algoritmu	**Paměť**
Count-Min + Heavy hitters	82 728 B ∼ 83 kB
Count-Median + Heavy hitters	82 808 B ∼ 83 kB
Exaktní metoda řešení	1 585 152 B ∼ 1,59 MB

Velikost paměťové náročnosti metody exaktního řešení je minimálně *počet unikátních vstupních položek* × *velikost klíče* × *velikost čítače*. To při použití největší analyzované položky paketu, tedy IPv6 adresy dává velikost klíče 16 B a velikosti čítače 8 B. Paměťová náročnost je pak následující:

- Exaktní metoda = 66 048 × (16 + 8) = 1 585 152 bajtů.

Paměťová náročnost implementovaného řešení poskytuje oproti metodě exaktního řešení výrazně nižší paměťové nároky a efektivněji tak využije vyšší úrovně hierarchie cache paměti.

8.4 Propustnost

Měření datové propustnosti probíhalo zvlášť pro režim struktury provozu a pro režim doporučení mitigačních pravidel. Způsob provedení a výsledky měření provedených testů je popsán v následujícím textu.

Režim struktury provozu. Výsledkem měření datové propustnosti u tohoto režimu byla zvolena datová propustnost při zpracování položky zdrojové IP adresy. Toto rozhodnutí bylo učiněno vzhledem k tomu, že na nejkratších paketech (64 B), které byly při měření datové propustnosti generovány, nelze dostatečně reprezentovat všechny analyzované položky paketu a tak by byly výsledky měření zkresleny. Proto byla pro měření datové propustnosti vybrána zdrojová IP adresa, která svou složitostí zpracování společně s cílovou IP adresou pokrývá nejhorší možný případ výpočetní náročnosti. Nejhorší možný případ reprezentuje proto, že je se svou velikostí položky 16 B nejnáročnější pro výpočet hashe. Ostatní položky jsou reprezentovány nanejvýš 2 B dat a výpočet hashe je tak mnohem rychlejší. Platnost tohoto tvrzení byla ověřena na testovacích datech.

Všechny testy byly měřeny nad stejnou sadou testovacích dat s rozdílným nastavením parametrů nástroje. Struktura testovacího provozu je ukázána

v tabulce 8.5. Veškerý provoz byl generován rovnoměrně ze zadaných rozsahů hodnot. Datová sada se dá rozdělit na dvě části, a to frekventované položky a ostatní provoz. U zdrojové IP adresy tvořily frekventované položky 50 % z celkového objemu provozu a bylo jich celkem 100. Zbylých 50 % tvořilo 16384 IP adres ze zadaného rozsahu. Tento provoz svou strukturou reprezentuje situaci probíhajícího DDoS útoku, kde určité procento zdrojových IP adres tvoří svou aktivitou velké procento celkového objemu provozu.

Tabulka 8.5: Struktura testovacího provozu při měření datové propustnost režimu struktury provozu.

Typ položky	**Nejvíce frekventované položky**	**Ostatní provoz**
Zdrojová IP adresa	192.85.1.1 - 192.85.1.100	100.50.1.1 - 100.50.63.255
Cílová IP adresa	192.0.1.1 - 192.0.1.100	192.50.1.1 - 192.50.63.255
Protokol	UDP	UDP
Zdrojový port	512-612	1-65535
Cílový port	1024-1124	1-65535
Délka	64	64
Zastoupení	50 %	50 %

První test spočíval v měření datové propustnosti v závislosti na počtu procesních vláken a typu použitého sketche. NDP rozhraní na testovacím serveru poskytovalo zpracovaní až 16 procesními vlákny a byla tedy testována datová propustnost pro 1, 8 a 16 procesních vláken. Nástroj byl spuštěn s výchozími parametry, tedy šířkou sketche 2048, hloubkou sketche 5, TOP-N hodnot 10. Výsledky měření zobrazuje tabulka 8.6.

Tabulka 8.6: Datová propustnost režimu struktury provozu v závislosti na počtu procesních vláken a typu použitého sketche.

Typ sketche	**Propustnost Mp/s**			**Proustnost %**		
Count-Min	15,44	102,65	144,05	10,37	68,99	96,81
Count-Median	7,09	54,92	111,03	4,76	36,91	74,61
NitroSketch (Cmin, 16, 8)	23,95	148,8	148,8	16,09	100	100
NitroSketch (Cmin, 32, 16)	25,52	148,8	148,8	17,15	100	100
NitroSketch (Cmedian, 16, 8)	24,11	148,8	148,8	16,20	100	100
NitroSketch (Cmedian, 32, 16)	24,21	148,8	148,8	16,27	100	100
Počet procesních vláken	1	8	16	1	8	16

Z tabulky 8.6 je vidět, že pro základní varianty sketchů se nepodařilo dosáhnou plné datové propustnosti při nejkratších paketech (148,8 Mp/s). Plné propustnosti se při měření povedlo dosáhnout až ve variantě NitroSketche s 8 a 16 procesními vlákny. Hodnoty uvedené v závorce u NitroSketche vyjadřují vzorkování ve sketchi a vzorkování v Heavy hitteru. Tabulka také ukazuje, že varianta Count-Min sketche je výkonnější, než varianta Count-Median sketche, ale na druhou stranu neškáluje tak dobře se zvětšujícím se počtem procesních vláken. Škálování u Count-Median sketch dosahuje lineárního zrychlení. Datová propustnost při použití NitroSketche a jednoho procesního vlákna naráží na horní limit počtu zpracovaných paketů jedním vláknem a jde vidět, že ani se zvětšujícím se vzorkováním dat datová propustnost dále neškáluje.

Druhý testovací případ zahrnoval vliv počtu udržovaných položek v Heavy hitteru (TOP-N) na datovou propustnost. Nástroj byl spuštěn s následujícími parametry: šířka sketche 2048, hloubka sketche 5 a počet procesních vláken 16. Výsledky měření jsou ukázány v tabulce 8.7.

Tabulka 8.7: Datová propustnost režimu struktury provozu v závislosti na hodnotě TOP-N a typu použitého sketche při 16 procesních vláknech.

Typ sketche	**Propustnost Mp/s**			**Proustnost %**		
Count-Min	125,98	121,89	75,79	84,66	81,92	50,93
Count-Median	101,07	95,98	75,49	67,92	64,50	50,73
NitroSketch (Cmin, 16, 8)	148,8	148,8	144,12	100	100	96,85
NitroSketch (Cmin, 32, 16)	148,8	148,8	148,8	100	100	100
NitroSketch (Cmedian, 16, 8)	148,8	148,8	140,27	100	100	94,27
NitroSketch (Cmedian, 32, 16)	148,8	148,8	148,8	100	100	100
TOP-N	30	100	500	30	100	500

Z tabulky 8.7 je vidět, že se zvětšující se hodnotou TOP-N a tedy i počtem zobrazených hodnot na výstupu snižuje výkonnost. Nicméně NitroSketche poskytují stále dostatečný výkon i při vysokých hodnotách proměnné TOP-N.

Dalším testovacím případem byl vliv počtu pravidel v paketovém filtru na datovou propustnost. Jako výchozí pravidlo bylo zvolené pravidlo z následující ukázky, které bylo nastaveno tak, aby zahrnulo veškerý provoz a žádné pakety nebyly paketovým filtrem zahozeny. Nástroj byl spuštěn s následujícími parametry: šířka sketche 2048, hloubka sketche 5, TOP-N 10 a počet procesních vláken 16. Výsledky měření zobrazuje tabulka 8.8.

```
<network>
        <src_address>192.85.1.2/0</src_address>
        <dst_address>192.0.1.1/0</dst_address>
        <dst_port>1-65535</dst_port>
        <src_port>1-65535</src_port>
        <length>1-65535</length>
        <protocol>17</protocol>
</network>
```

Tabulka 8.8: Datová propustnost režimu struktury provozu v závislosti na počtu pravidel v paketovém filtru při 16 procesních vláknech.

Typ sketche	**Propustnost Mp/s**			**Proustnost %**		
Count-Min	114,64	98,49	70,74	77,04	66,18	47,54
Count-Median	75,01	65,02	54,59	50,41	43,70	36,68
NitroSketch (Cmin, 16, 8)	148,8	146,66	112,76	100	98,56	75,78
NitroSketch (Cmin, 32, 16)	148,8	148,8	126,72	100	100	85,16
NitroSketch (Cmedian, 16, 8)	148,8	140,44	114,11	100	94,38	76,68
NitroSketch (Cmedian, 32, 16)	148,8	146,61	125,71	100	98,53	84,48
Počet pravidel	1	5	10	1	5	10

Z tabulky 8.8 vyplývá že počet pravidel porovnávaných v paketovém filtru má významný vliv na propustnost a při 10 pravidlech již plná propustnost nebyla dosažena ani v jednom testovacím případu. Nicméně předpokladem paketového filtru je část provozu zahodit a tím i snížit výpočetní náročnost, což se u tohoto testu nedělo, neboť byl analyzován veškerý provoz.

Poslední provedený test zjišťoval vliv velikosti sketche (šířka a hloubka) na datovou propustnost. Nástroj byl spuštěn s následujícími parametry: TOP-N 10 a počet procesních vláken 16. Výsledky měření zobrazuje tabulka 8.9.

Tabulka 8.9: Datová propustnost režimu struktury provozu v závislosti na velikosti sketche při 16 procesních vláknech.

Typ sketche	**Propustnost Mp/s**			**Proustnost %**		
Count-Min	146,58	132,02	133,97	98,51	88,72	90,03
Count-Median	113,09	91,18	99,93	76,00	61,72	67,15
Šířka, hloubka	(1024,5)	(2048,6)	(4096,5)	(1024,5)	(2048,6)	(4096,5)

Tabulka 8.9 neobsahuje informace o NitroSketchi, protože u všech testovaných případů byla dosažena plná propustnost. Nicméně na základních variantách sketchů lze vidět, že zdvojnásobení velikosti řádku sketche má na propustnost menší vliv než zvýšení hloubky sketche o jedničku. Při snížení počtu řádků na 1024 oproti výchozí hodnotě 2048 byla v porovnání s výsledky z tabulky 8.6 zvýšena propustnost jen nepatrně.

Režim doporučení mitigačních pravidel byl testován jedním testovacím případem, který měřil datovou propustnost vzhledem k počtu procesních vláken a typu použitého sketche. Testovací provoz simuloval DNS amplifikační DDoS útok a jeho strukturu zobrazuje tabulka 8.10.

Tabulka 8.10: Struktura testovacího provozu pro testování propustnosti režimu doporučení mitigačních pravidel.

Typ položky	**Rozsah hodnot**
Zdrojová IP adresa	192.85.1.1 - 192.85.31.255
Cílová IP adresa	192.0.0.1
Protokol	UDP
Zdrojový port	53
Cílový port	32720
Délka	64

Testovací provoz zahrnuje jen provoz útočníka, a to z důvodu, aby propustnost nebyla ovlivněna zahozením části provozu adaptivním paketovým filtrem. Reálný útok by obsahoval pakety s mnohem větší velikostí, ale pro potřeby otestování propustnosti na nejkratší délce paketů byla velikost paketů snížena na 64 B. Za výslednou hodnotu určující datovou propustnost se vybrala nejnižší hodnota datové propustnosti mezi všemi pozorovacími okny za běhu algoritmu. Nástroj byl spuštěn s následující konfigurací: šířka sketche 2048, hloubka sketche 5, TOP-N 10 a threshold 20 %. Výsledek měření ukazuje tabulka 8.11.

Z tabulky 8.11 je vidět, že pro základní variantu Count-Min sketche se podařilo dosáhnout při 16 procesních vláknech plné datové propustnosti a Count-Median sketch pak dosáhl jen těsně tuto hranici. Plné datové propustnosti při zpracování dat 16 procesními vlákny dosáhly i všechny varianty NitroSketche, u kterého hodnoty uvedené v závorce vyjadřují vzorkování dat ve sketchi a vzorkování v Heavy hitteru. Při zpracování dat jedním procesním vláknem a s použitím NitroSketche se naráží na horní limit počtu zpracovaných paketů jednoho procesního vlákna, protože ani zvětšující se vzorkování dat nepřináší podobně jako v tabulce 8.6 výraznější nárůst propustnosti. Při celkovém pohledu na data lze říci, že použití varianty používající Count-Min sketch dosahují vyšší datové propustnosti oproti variantám využívajících Count-Median sketch, což lze vzhledem k počítání jedné hash funkce navíc u Count-Median sketche očekávat.

Tabulka 8.11: Datová propustnost režimu doporučení mitigačních pravidel v závislosti na počtu procesních vláken a typu sketche.

Typ sketche	**Propustnost Mp/s**			**Proustnost %**		
Count-Min	15,09	101,29	148,8	10,14	68,07	100
Count-Median	11,36	77,99	146,13	7,63	52,41	98,21
NitroSketch (Cmin, 16, 8)	21,90	142,61	148,8	14,71	95,84	100
NitroSketch (Cmin, 32, 16)	23,89	148,8	148,8	16,06	100	100
NitroSketch (Cmedian, 16, 8)	20,46	134,86	148,8	13,75	90,63	100
NitroSketch (Cmedian, 32, 16)	23,83	143,59	148,8	16,01	96,49	100
Počet procesních vláken	1	8	16	1	8	16

8.5 Funkčnost

Proces ověřování správné funkčnosti nástroje zahrnoval tyto testy:

Test parseru paketů sloužil k ověření správného rozpoznání podporovaných hlaviček protokolů (podle diagramu 7.2) a správné extrakci získávaných položek z hlaviček protokolů do vnitřní datové struktury. Test spočíval v generování několika možných podporovaných kombinací hlaviček paketu, které prošly parserem paketů a následně byla ověřena jejich správnost. Testovány byly i mezní hodnoty podporovaného počtu hlaviček, které byly popsány v sekci 7.2.

Test paketové filtru spočíval v ověření správného propouštění nebo zahazování paketů na základě zadaného konfiguračního souboru. Byla ověřena správná funkčnost všech podporovaných položek paketového filtru, včetně překrývajících se pravidel nebo zadaných whitelistů.

Test režimu struktury provozu sloužil k ověření správné identifikace nejvíce zastoupených hodnot v generovaném provozu a k ověření jejich správného pořadí na výstupu. Byl vytvořen jeden testovací případ, který svou strukturou pokrýval většinu položek paketu. Strukturu generovaného provozu ukazuje tabulka 8.12.

Struktura generovaného provozu simulovala situaci, která by v případě aktivního volumetrického DDoS útoku mohla nastat, a to zvýšené zastoupení některých hodnot u položek paketu. Tento testovací případ vytvářel tyto více frekventované hodnoty u všech analyzovaných položek vyjma protokolu a příznaků TCP. Provoz nejvíce frekventovaných položek zastupoval celkem 50 % z celkového objemu provozu a byl tvořen celkem 10 hodnotami u všech položek mimo ty výše zmíněné. Pro ověření správného pořadí položek na

Tabulka 8.12: Struktura provozu při testování režimu struktury provozu.

Typ položky	**Nejvíce frekventované položky**	**Ostatní provoz**
Zdrojová IP adresa	192.85.1.1 - 192.85.1.10	100.50.1.1 - 100.50.63.255
Cílová IP adresa	192.0.0.1 - 192.0.0.10	192.0.0.11 - 192.0.0.255
Protokol	UDP	UDP, TCP
Zdrojový port	1-10	11-65535
Cílový port	1-10	11-65535
Délka	100-110	111-1400
Vlan ID	1-10	11-4095
TCP flags		SYN, ACK
Zastoupení	50 %	50 %

výstupu algoritmu bylo zastoupení nejvíce frekventovaných položek vytvořeno tak, že položky s nižší hodnotou mají zastoupení vyšší než položky s vyšší hodnotou. Ostatní provoz byl generován rovnoměrně v zadaných mezích.

Test probíhal s výchozím nastavením nástroje a zobrazoval 10 nejvíce zastoupených hodnot. Test byl spuštěn opakovaně pro variantu s Count-Min sketchem, Count-Median sketchem a NitroSketchem. Všechny testované režimy správně identifikovaly nejvíce frekventované hodnoty a zobrazily je ve správném pořadí. Vzhledem k velikosti výpisu zde bude ukázána jen část zobrazující strukturu provozu u zdrojového portu s využitím Count-Min sketche.

```
Src port   [Packets]   [Approx. Packets]   [Total %]
1           1241484         1241484         14.67%
2            818272          818272          9.67%
3            606069          606069          7.16%
4            465713          465713          5.50%
5            359691          359691          4.25%
6            274317          274317          3.24%
7            204324          204324          2.41%
8            143955          143955          1.70%
9             91049           91049          1.08%
10            44103           44103          0.52
```

Výpis ukazuje, že nástroj správně identifikoval nejvíce frekventované položky a zobrazil je ve správném pořadí. Zastoupení jednotlivých položek odpovídá tomu, jak byly generovány a jejich suma odpovídá 50 % celkového provozu.

Test režimu doporučení mitigačních pravidel spočíval v ověření správné identifikace probíhajícího DDoS útoku v síťovém provozu. Celkem byly vytvořeny dva testovací případy, kdy v každém byl testován jiný typ DDoS útoku. Generovaný síťový provoz v sobě obsahoval provoz legitimních uživatelů

a DDoS útok. Provoz legitimních uživatelů byl generován tak, aby se přiblížil provozu na reálné síti.

První testovací případ zahrnoval ověření správné identifikace DNS amplifikačního útoku. Struktura generovaného provozu je ukázána v tabulce 8.13. Provoz legitimních uživatelů tvořil 30 % z celkového objemu generovaných paketů a byl generován rovnoměrně na 3 cílové IP adresy. Jedna z těchto adres byla oběť útoku, na kterou tedy směřovala třetina provozu legitimních uživatelů. Třetinu legitimního provozu zastupoval protokol UDP a zbytek byl zastoupen protokolem TCP. Zdrojové a cílové porty byly generovány náhodně v zadaném rozsahu, stejně jako délka paketů. Provoz útočníka zastupoval 70 % z celkového objemu provozu a byl generován ze 100 zdrojových IP adres, které reprezentují zneužité rekurzivní DNS servery. Cílem útoku byla IP adresa `192.0.0.1`, na kterou tak i s legitimním provozem směřovalo celkem 80 % celkového provozu. Jelikož se jednalo o DNS amplifikační útok, byl protokolem transportní vrstvy nastaven protokol UDP se zdrojovým portem 53. Útok cílil na dva různé cílové porty z nichž jeden zastupoval 30 % a druhý 40 % z celkového objemu provozu. Délka útočících paketů byla generována náhodně v zadaném rozsahu.

Test probíhal s výchozím nastavením nástroje a byl spuštěn opakovaně pro variantu s Count-Min sketchem, Count-Median sketchem a NitroSketchem. Všechny režimy ukázaly stejný výsledek. Výsledek s použitím Count-Min sketche zobrazuje následující výpis.

Tabulka 8.13: Struktura provozu při testování DNS amplifikačního útoku.

Typ položky	**Rozsah hodnot (legitimní uživatelé)**	**Rozsah hodnot (útočník)**
Zdrojová IP adresa	192.85.1.1 - 192.85.31.255	104.45.5.1 - 104.45.5.100
Cílová IP adresa	192.0.0.1 - 192.0.0.0.3	192.0.0.1
Protokol	UDP, TCP	UDP
Zdrojový port	1-65535	53
Cílový port	1-65535	32720, 42350
Délka	64-1400	800-1400
Vlan ID		
TCP flags	SYN, ACK	
Zastoupení	30 %	70 %

```
Dst IP     [ Packets ]  [Approx. Packets ]  [Total %]
192.0.0.1     4295602         4295602         79.992%

############################################################
```

```
Recommended rule: "dst net 192.0.0.1/32 protocol 17".
This rule should block 91.67% of traffic heading to
    ↪ destination network.

Recommended rule: "dst net 192.0.0.1/32 protocol 17 src port
    ↪ 53".
This rule should block 87.50% of traffic heading to
    ↪ destination network.

Recommended rule: "dst net 192.0.0.1/32 protocol 17 src port
    ↪ 53 dst port 42350".
This rule should block 50.00% of traffic heading to
    ↪ destination network.

Recommended rule: "dst net 192.0.0.1/32 protocol 17 src port
    ↪ 53 dst port 32720".
This rule should block 37.50% of traffic heading to
    ↪ destination network.
```

Z výpisu lze vidět, že nástroj správně detekoval nejvíce zastoupenou cílovou IP adresu, kterou určil jako oběť útoku. První doporučené pravidlo je velmi obecné a zahrnuje v sobě i legitimní provoz, nicméně algoritmus správně určil protokol útoku. Druhé doporučené pravidlo správně určilo zdrojový port, kterým pokrylo nejen celý útočníkův provoz, ale hlavně dalo administrátorovi sítě jasnou informaci, že se jedná právě o DNS amplifikační DDoS útok. Poslední dvě pravidla pak ještě zpřesnila druhé pravidlo o cílové porty a tím poskytla informaci k přesnějšímu blokování útoku.

Druhý testovací případ v sobě obsahoval TCP SYN flood útok a strukturu generovaného provozu ukazuje tabulka 8.14. V tomto případě samotný DDoS útok tvořil pouze 20 % z celkového objemu provozu a na provozu cílové IP adresy (oběti) se podílel jen 40%. Zbylý provoz byl od legitimních uživatelů a tvořil jej ze 44 % UDP provoz a ze zbylých 16 % TCP provoz. Provoz legitimních uživatelů byl generován z rozsahu hodnot zobrazených v tabulce. Cílem útoku byla IPv6 adresa *2000::1*, na kterou útočník posílal TCP pakety s nastaveným příznakem SYN. Rozsah obou portů byl generován náhodně v celém rozsahu. Velikost útočících paketů byla nastavena na 120 B, což simuluje situaci, kdy jsou pakety útočníkem uměle generovány podle nějakého vzoru.

Testování probíhalo stejným způsobem, jako u předchozího příkladu. Všechny režimy opět ukázaly stejný výsledek a výsledek s použitím Count-Median sketche zobrazuje následující výpis.

Tabulka 8.14: Struktura provozu při testování TCP SYN flood útoku.

Typ položky	**Rozsah hodnot (legitimní uživatelé)**	**Rozsah hodnot (útočník)**
Zdrojová IP adresa	2005::1 - 2005::2fff	2001::1 - 2001::1ffff
Cílová IP adresa	2000::1 - 2000::5	2000::1
Protokol	UDP, TCP	TCP
Zdrojový port	1-65535	1-65535
Cílový port	1-65535	1-65535
Délka	100-1400	120
Vlan ID		
TCP flags	SYN, ACK	SYN
Zastoupení	80%	20%

```
Dst IP     [ Packets ]  [Approx. Packets ]  [Total %]
2000::1     7806498         7806498         49.99%

############################################################

Recommended rule: "dst net 2000::1/128 protocol 6".
This rule should block 56.00% of traffic heading to
    ↪ destination network.

Recommended rule: "dst net 2000::1/128 protocol 17".
This rule should block 44.00% of traffic heading to
    ↪ destination network.

Recommended rule: "dst net 2000::1/128 protocol 6 tcpflags
    ↪ S".
This rule should block 44.00% of traffic heading to
    ↪ destination network.

Recommended rule: "dst net 2000::1/128 protocol 6 length 120
    ↪ tcpflags S".
This rule should block 40.03% of traffic heading to
    ↪ destination network.
```

Výpis ukazuje, že oběť útoku byla nástrojem správně určena. V prvním kroku byla doporučena dvě velmi obecná pravidla, která z velké části obsahují provoz legitimních uživatelů. Bylo vypsáno i pravidlo s protokolem UDP, které se na útoku nepodílí, ale tvoří velkou část provozu, a proto se na něj algoritmus zaměřil. Třetí pravidlo rozšiřuje předchozí obecné TCP pravidlo o informaci,

že pakety s příznakem SYN tvoří velkou část provozu. Toto pravidlo již specifikuje značnou část útočícího provozu a také poskytuje administrátorovi sítě informaci o typu DDoS útoku. Poslední pravidlo ještě více specifikuje provoz útočníka, a to rozšířením pravidla o použitou délku paketů.

Závěr

Tato diplomová práce se věnovala návrhu a implementaci nástroje pro analýzu síťového provozu, který je založen na bázi online analýzy paketů. Cílem tohoto nástroje je získat informace o struktuře provozu a umožnit administrátorovi co nejrychleji identifikovat zdroje DDoS útoků. Díky tomu je možné vytvořit účinná mitigační pravidla, kterými mohou být útoky blokovány.

Motivací pro vznik tohoto nástroje byl reálný případ útoků zaznamenaných v národní akademické infrastruktuře. Absence takového nástroje, který by dokázal v případě potřeby identifikovat strukturu provozu a potenciální zdroje útoků a zároveň i navrhnout vhodná mitigační pravidla zásadně ztěžuje a zpomaluje obranu infrastruktury. Kritickým případem je stav, ve kterém standardní monitorovací infrastruktura přestane zvládat množství provozu a správci sítě pak nemají k dispozici potřebné informace.

Nástroj vytvořený v rámci této diplomové práce je navržen pro autonomní zpracování síťového provozu v online režimu na vysokorychlostních linkách. Na základě rešerše existujicích vhodných datových struktur nástroj využívá technologii Sketch. Výsledkem je aplikace napsaná v jazyce C++, která může být integrována do systému DDoS mitigace vyvíjeného ve sdružení CESNET. Experimentální vyhodnocení popsané v samostatné kapitole této práci ukazuje výkonnostní i paměťové vlastnosti, ze kterých vyplývá použitelnost do produkčního nasazení na 100 Gb/s linky. Při této rychlosti dokáže nástroj zpracovat celý provoz i pro nejkratší pakety.

Vedle vytvořené aplikace, jež je hlavním přínosem této práce, se text diplomové práce podrobně věnoval analýze problematiky DDoS útoků a vhodných datových struktur pro ukládání dat o provozu.

Jako potenciální možné pokračování této práce by bylo vhodné prozkoumat další typy síťových útoků, na které by mohl mitigační systém reagovat a rozšířit systém návrhu mitigačních pravidel i na tento provoz.

Literatura

[1] Tripathi, N.; Mehtre, B.: DoS and DDoS attacks: Impact, analysis and countermeasures. In *National Conference on Advances in Computing, Networking and Security, Nanded, India. Retrieved from https://docs. google. com/viewer*, 2013, str. 1.

[2] Mirkovic, J.; Reiher, P.: A taxonomy of DDoS attack and DDoS Defense mechanisms. *ACM SIGCOMM Computer Communication Review*, ročník 34, 05 2004, doi:10.1145/997150.997156.

[3] Prasad, K. M.; Reddy, A. R. M.; Rao, K. V.: DoS and DDoS attacks: defense, detection and traceback mechanisms-a survey. *Global Journal of Computer Science and Technology*, 2014: str. 2.

[4] Hoque, N.; Bhattacharyya, D. K.; Kalita, J.: Botnet in DDoS Attacks: Trends and Challenges. *IEEE Communications Surveys amp Tutorials*, ročník 17, 06 2015: str. 2, doi:10.1109/COMST.2015.2457491.

[5] What is a Botnet? Cloudfare, [Online; navštíveno 16.11.2020]. Dostupné z: `https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/`

[6] Abhishta, A.; Heeswijk, W.; Junger, M.; aj.: Why would we get attacked? An analysis of attacker's aims behind DDoS attacks. *J. Wireless Mobile Netw., Ubiquitous Comput., Dependable Appl.*, ročník 11, č. 2, 2020: str. 6.

[7] Brodsky, Z.: The Psychology Behind DDoS: Motivations and Methods. 2020. Dostupné z: `https://www.perimeter81.com/blog/network/the-psychology-behind-ddos-attacks/`

[8] Balaban, D.: The History and Evolution of DDoS Attacks. 2020. Dostupné z: `https://www.embedded-computing.com/guest-blogs/the-history-and-evolution-of-ddos-attacks`

[9] Famous DDoS attacks — The largest DDoS attacks of all time. Cloudfare, [Online; navštíveno 19.11.2020]. Dostupné z: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/

[10] McCarthy, K.: Mafiaboy given eight months. 2001. Dostupné z: https://www.theregister.com/2001/09/13/mafiaboy_given_eight_months

[11] NGAK, C.: 13 members of hacking group Anonymous indicted over "Operation Payback". cbsnews.com, 2013. Dostupné z: https://www.cbsnews.com/news/13-members-of-hacking-group-anonymous-indicted-over-operation-payback/

[12] PolesnýK, D.: Čeští a slovenští Anonymous zaútočili na weby OSA a vlády. Živě.cz, 2010. Dostupné z: https://www.e15.cz/magazin/cesti-a-slovensti-anonymous-zautocili-na-weby-osa-a-vlady-737826

[13] Dočekal, D.: Weby českých politických stran jsou mimo provoz, Anonymous splnili hrozbu. Lupa.cz, [Online; navštíveno 19.11.2020]. Dostupné z: https://www.lupa.cz/clanky/weby-ceskych-politickych-stran-jsou-mimo-provoz-anonymous-splnili-hrozbu/

[14] Mahjabin, T.; Xiao, Y.; Sun, G.; aj.: A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*, ročník 13, č. 12, 2017: str. 1550147717741463, doi:10.1177/1550147717741463.

[15] Arbor 12th Annual World Infrastructure Security Report. 2017. Dostupné z: https://www.netscout.com/news/press-release/worldwide-infrastructure-security-report

[16] Beatrice, A.: THE LARGEST AND FAMOUS DDOS ATTACKS OF ALL TIME. analyticsinsight.net, 2020. Dostupné z: https://www.analyticsinsight.net/largest-famous-ddos-attacks-time/

[17] STRESSTHEM. Dostupné z: https://www.stressthem.to/#pricing

[18] UDP Flood. imperva.com. Dostupné z: https://www.imperva.com/learn/ddos/udp-flood/

[19] ICMP Flood Attacks. netscout.com. Dostupné z: https://www.netscout.com/what-is-ddos/icmp-flood

[20] TCP SYN Flood. imperva.com. Dostupné z: https://www.imperva.com/learn/ddos/syn-flood/

[21] NTP Amplification. imperva.com. Dostupné z: https://www.imperva.com/learn/ddos/ntp-amplification/

[22] SNMP Reflection / Amplification. imperva.com. Dostupné z: `https://www.imperva.com/learn/ddos/snmp-reflection/`

[23] Šiška P; Kuka M, K. J.: DDoS Protector User Manual. LIBEROUTER / CESNET TMC group, 2017.

[24] ŠIŠKA, P.: *Systém pro ochranu před DoS útoky*. Bakalářská práce, 2018.

[25] Lu, Y.; Prabhakar, B.; Bonomi, F.: Bloom filters: Design innovations and novel applications. 01 2005.

[26] Heule, S.; Nunkesser, M.; Hall, A.: HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm. In *Proceedings of the EDBT 2013 Conference*, Genoa, Italy, 2013.

[27] Cormode, G.; Muthukrishnan, S.: Approximating Data with the Count-Min Data Structure. 05 2012.

[28] Charikar, M.; Chen, K.; Farach-Colton, M.: Finding Frequent Items in Data Streams. ICALP '02, Berlin, Heidelberg: Springer-Verlag, 2002, ISBN 3540438645, str. 693–703.

[29] Hovmand, J. N.: *Estimating Frequencies and Finding Heavy Hitters*. Diplomová práce, 2016.

[30] Roughgarden, T.; Valiant, G.: CS168: The Modern Algorithmic Toolbox Lecture 2: Approximate Heavy Hitters and the Count-Min Sketch. 2016.

[31] Liu, Z.; Ben-Basat, R.; Einziger, G.; aj.: Nitrosketch: robust and general sketch-based monitoring in software switches. 08 2019, ISBN 978-1-4503-5956-6, s. 334–350, doi:10.1145/3341302.3342076.

[32] xxHash - Extremely fast hash algorithm. github.com. Dostupné z: `https://github.com/Cyan4973/xxHash`

Příloha A

Seznam použitých zkratek

C&C Command and Control

CPU Central processing unit

DDoS Distributed Denial of Service

DMA Direct memory access

DRDoS Distributed Reflected Denial of Service

DoS Denial of Service

DPI Deep Packet Inspection

DNS Domain Name System

FPGA Field Programmable Gate Array

HH Heavy hitters

IoT Internet of Things

ISP Internet Service Provider

IRC Internet Relay Chat

NDP Netcope Data Plane

NFB Netcope FPGA Board

NTP Network Time Protocol

SNMP Simple Network Management Protocol

Gbps Gigabits per second

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

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