
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**
**DEPARTMENT OF COMPUTER SYSTEMS**


# TESTOVÁNÍ ZAŘÍZENÍ PRO OCHRANU PŘED DOS ÚTOKY

**TESTING OF DEVICE FOR DOS ATTACK PROTECTION**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **MATÚŠ BURZALA**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. JAN KUČERA**
**SUPERVISOR**



**BRNO 2019**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

# Zadání bakalářské práce


Student: **Burzala Matúš**
Program: Informační technologie
Název: **Testování zařízení pro ochranu před DoS útoky**
**Testing of Device for DoS Attack Protection**
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se s problematikou útoků typu odepření služby (DoS) a zařízením vyvíjeným v rámci CESNET pro ochranu před těmito útoky.
2. Nastudujte možnosti generátoru síťového provozu Spirent TestCenter a dále se seznamte s CI (Continuous Integration) systémem Jenkins.
3. Navrhněte rozšiřitelné prostředí, které umožní automatizované testování tohoto specifického síťového zařízení pro ochranu před DoS útoky.
4. Prostředí dle návrhu implementujte. V rámci implementace vytvořte také nezbytnou sadu testů ověřující funkční i výkonnostní parametry zařízení dle specifikace.
5. V závěru diskutujte vlastnosti vytvořeného řešení, dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kučera Jan, Ing.**
Konzultant: Dražil Jan, Ing., UPSY FIT VUT
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 26. října 2018

# Abstrakt

Táto práca sa zaoberá testovaním zariadenia pre ochranu pred (D)DoS útokmi DCPRO, vyvíjaným v rámci združenia CESNET. Cieľom práce bolo navrhnúť a implementovať rozšíriteľný testovací systém, ktorý by umožňoval automatizované testovanie zariadenia DCPRO. Okrem samotného systému bola v rámci práce vytvorená kolekcia testov overujúcich funkčné a výkonnostné parametre zariadenia. Vytvorený systém bol následne integrovaný do systému kontinuálnej integrácie Jenkins. Konkrétne bolo v rámci práce vytvorených 109 špecifických testovacích scenárov zameraných na testovanie firmvérových modulov zariadenia, 7 testovacích scenárov zameraných na overenie priepustnosti zariadenia, 10 testovacích scenárov overujúcich správnu funkčnosť softvérových modulov na ochranu pred SYN Flood a amplifikačnými útokmi a jeden test overujúci schopnosť zariadenia smerovať sieťovú prevádzku. Vytvorený testovací systém bol navrhutý a implementovaný tak, aby bol jednoducho rozšíriteľný. Za účelom zjednodušenia budúceho rozširovania systému je v rámci zdrojových súborov vytvorená šablóna pre vytváranie testov a text práce obsahuje návod ako pri vytváraní nových testov postupovať.



# Abstract

This thesis deals with testing of a device for (D)DoS protection DCPRO, that is developed within the CESNET association. The aim of the thesis was to design and implement an extendable testing system, which would allow automated testing of DCPRO device. In addition to the testing system, there was created a collection of tests for verification of functional and performance parameters of the device within the thesis. Afterwards, the developed system was integrated into a continuous integration system Jenkins. Particularly within the thesis there were created 109 specific test scenarios to test device firmware modules, 7 throughput test scenarios, 10 test scenarios to verify proper functionality of software modules dedicated to SYN Flood and amplification attacks protection, and one test for verification of device network routing ability. The developed testing system is easily extensible. In order to simplify a future extension of the system, there is a created template encapsulated in source files for new test creation and text part of the thesis contains guide how to create new tests.



# Kľúčové slová

CESNET, DCPRO, (D)DoS Protector, (D)DoS útoky, Jenkins, kontinuálna integrácia, Spirent TestCenter, testovanie, sieťové rozhranie, spracovanie sieťových dát, filtrácia sieťovej prevádzky

# Keywords

CESNET, DCPRO, (D)DoS Protector, (D)DoS attacks, Jenkins, continuous integration, Spirent TestCenter, testing, network interface, network data processing, network traffic filtration


# Citácia

BURZALA, Matúš. *Testování zařízení pro ochranu před DoS útoky.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

# Testování zařízení pro ochranu před DoS útoky

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jana Kučeru. Ďalšie informácie mi poskytol Ing. Jan Dražil. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....
Matúš Burzala
15. mája 2019

## Poďakovanie

Rád by som poďakoval vedúcemu práce pánovi Ing. Janovi Kučerovi za odborné vedenie práce, priateľský prístup a množstvo užitočných rád a pripomienok poskytnutých počas tvorby tejto práce. Taktiež by som sa rád poďakoval odbornému konzultantovi pánovi Ing. Janovi Dražilovi.

# Obsah

# Kapitola 1

# Úvod

Proces testovania je neoddeliteľnou súčasťou vývoja každého produktu, či už sa jedná o softvér, alebo hardvér. Potreba testovania vzniká hlavne za účelom udržania kvality, zamedzenia regresií a overenia funkčnosti produktu ako celku. U testovania je požadované, aby boli jeho výsledky jednoducho interpretovateľné a aby bolo možné ho vykonávať opakovane, vždy v prípade potreby. Cieľom tejto práce bolo testovanie zariadenia pre ochranu pred (D)DoS útokmi DCPRO, vyvíjaného v rámci združenia CESNET. Konkrétne bolo potrebné navrhnúť a implementovať rozšíriteľný systém umožňujúci automatizované testovanie zariadenia ako celku. Súčasťou vytvoreného systému je aj kolekcia testov overujúcich funkčné a výkonnostné parametre zariadenia. Výsledný systém je integrovaný do systému kontinuálnej integrácie Jenkins. Pri tvorbe systému boli zohľadnené vlastnosti zariadenia DCPRO a možnosti testovacieho prostredia dostupného v rámci združenia CESNET.

Práca je rozdelená do viacerých kapitol, z ktorých každá sa zaoberá riešením inej časti zadaného problému. Kapitola 2 obsahuje teoretické informácie o problematike (D)DoS útokov a teoretické informácie a technické detaily zariadenia pre ochranu pred (D)DoS útokmi DCPRO. Ďalej sú v tejto kapitole zhromaždené informácie o generátore sieťovej prevádzky Spirent TestCenter a systéme kontinuálnej integrácie Jenkins. V závere kapitoly sú zhrnuté informácie o niektorých najpoužívanejších testovacích frameworkov jazyka Python a analýza ich použiteľnosti v rámci tejto práce. V kapitole 3 sa nachádza návrh systému, ktorý obsahuje popis testovacieho prostredia a súhrn požiadaviek na testovací systém a štruktúru testov. V závere kapitoly sa nachádza návrh štruktúry systému a testov a návrh výstupu testov. V kapitole 4 je stručný popis implementácie. Konkrétne je tu uvedený zoznam a účel použitých technológií, popis adresárovej štruktúry systému a popis štruktúry a priebehu testov. V ďalšej časti tejto kapitoly je popis implementácie výstupu testov a testovacích výsledkov a popis toho ako bola implementovaná komunikácia so systémom Spirent TestCenter a so zariadením DCPRO. Na záver je v kapitole popísaný implementovaný proces integrácie a spúšťania testov v systéme Jenkins. V kapitole 5 sa nachádzajú podrobnosti o implementovaných testoch firmvérových a softvérových modulov a testov priepustnosti. V kapitole 6 je uvedený súhrn dosiahnutých výsledkov a možnosti rozšíriteľnosti vytvoreného systému spolu s návodom ako pri budúcom rozširovaní systému postupovať. Posledná kapitola 7 obsahuje celkový súhrn vykonanej práce a jej výsledkov.

# Kapitola 2

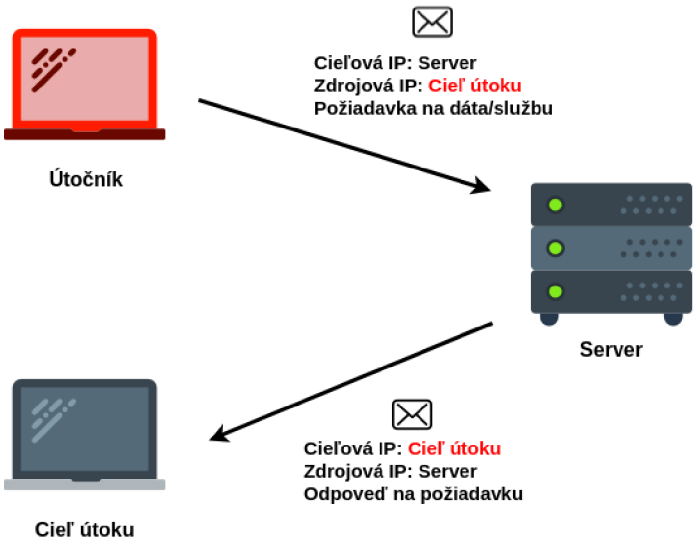
# Teoretický rozbor

V tejto kapitole sú uvedené teoretické informácie, ktorých znalosť je nutná pre pochopenie významu a spôsobu prevedenia praktickej časti práce. Sekcia 2.1 obsahuje úvod do problematiky útokov typu odoprenie služby, popis princípov, ktoré využívajú a zoznam najčastejších typov útokov a ich stručný popis. V sekcii 2.2 sú uvedené charakteristické vlastnosti zariadenia pre ochranu pred DoS útokmi DCPRO, popis jeho architektúry, firmvéru a jednotlivých firmvérových modulov. V nasledujúcej sekcii 2.3 je uvedený popis Generátoru sieťovej prevádzky Spirent TestCenter a základné princípy testovania pomocou tejto platformy. V závere tejto kapitoly sa nachádza sekcia 2.4, obsahujúca elementárne informácie o systéme Jenkins a jeho použití pri metóde kontinuálnej integrácie. Kapitolu uzatvára sekcia 2.5 analyzujúca aktuálne dostupné testovacie frameworky pre jazyk Python a možnosť ich použitia pri testovaní zariadenia DCPRO.

## 2.1 Útoky typu odopretie služby

Základom kybernetickej bezpečnosti je podľa [14] zabezpečenie ochrany pred hrozbami úniku informácií, narušenia integrity dát, odoprenia služby a nelegitímnym použitím informácií. Útoky typu Denial of Service (ďalej len DoS), teda odopretie služby, majú za cieľ obmedziť, alebo úplne zneprístupniť určitú sieťovú službu legitímnym užívateľom. Najčastejším cieľom sú emailové služby, elektronické bankovníctvo, alebo webové stránky. Docielenie stavu, v ktorom sa bude určitá služba užívateľom javiť ako spomalená, alebo nedostupná, je možné dvoma spôsobmi. Pri prvom spôsobe sa útočník snaží pomocou špeciálne upravených správ vyvolať chybu programu, alebo protokolu na cieľovom systéme a tým daný systém znefunkčniť. Druhým spôsobom je zahltenie sieťovej infraštruktúry cieľového systému obrovským množstvom zdanlivo legitímnych správ a tým spotrebovanie dostupných zdrojov cieľového systému, ako napríklad šírka pásma alebo pamäť. Tento typ útokov nesie názov volumetrické útoky, alebo tiež aj útoky hrubou silou. Dôsledkom takýchto útokov je to, že cieľový systém je natoľko zahltený, že nemá potrebné zdroje na uspokojovanie požiadaviek legitímnych užívateľov, a tak sa požadovaná služba javí ako nedostupná.

Pri väčšine DoS útokov, je využívaná technika podvrhávania falošnej IP adresy nazývaná IP Spoofing [7] viď obrázok 2.1. Pri tejto technike útočník (ľavá horná časť obrázku) podvrháva falošnú IP adresu zdroja do odosielaných IP paketov, prostredníctvom ktorých realizuje útok. Podvrhnutá adresa zvyčajne býva IP adresa cieľa útoku. Server (pravá časť obrázku), bez toho, aby o tom vedel, následné odpovede nesmeruje na útočníka, ale na cieľ útoku (ľavá spodná časť obrázku). Vďaka tejto technike teda môže útočník utajiť svoju iden-

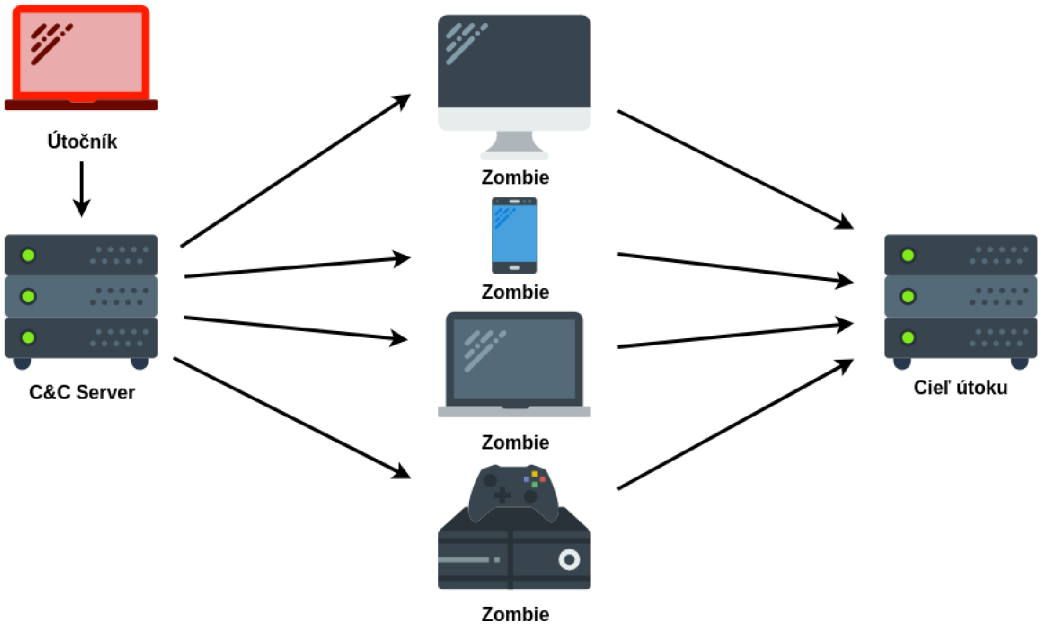



Obr. 2.1: Grafické znázornenie techniky podvrhávania falošnej IP adresy nazývanej IP Spoofing.

titu, ale aj presmerovať odpovede na iné zariadenie v sieti, ktoré môže byť cieľom útoku. IP spoofingu sa dá zamedziť napríklad pomocou filtrovania odchádzajúcich paketov popísaného v štandarde BCP 38 [24], podľa ktorého by mali poskytovatelia internetových služieb overovať, či zdrojové IP adresy u odchádzajúcich paketov patria do rozsahu siete, z ktorej boli odoslané.

### Distribuované útoky typu odopretie služby

Distribuované DoS útoky v angličtine nazývané Distributed Denial of Service (ďalej len DDoS), ktorých princíp je ilustrovaný na obrázku 2.2, používajú rovnaké mechanizmy ako DoS útoky s tým rozdielom, že pri DDoS útokoch útočník využíva na generovanie škodlivých správ viac zariadení súčasne. S počtom zariadení využitých pri útoku rastie aj efektivita a sila útoku a znižuje sa pravdepodobnosť vypátrania útočníka. Zoskupenie, alebo sieť týchto zariadení sa nazýva botnet [7]. Zariadenia tvoriace botnet (znázornené uprostred obrázku) nazývane bot, alebo zombie. Botnet je zvyčajne tvorený tisícmi až miliónmi zariadení. Konkrétne počítačmi, mobilnými zariadeniami, ale napríklad aj IoT zariadeniami, ktoré boli infikované škodlivým softvérom nazývaným malware [7]. Rôzne druhy malware majú rôzne stupne viditeľnosti. Môžu napríklad prevziať úplnú kontrolu nad zariadením a tým znemožniť majiteľovi používanie zariadenia, alebo naopak, môžu bežať na pozadí a čakať na inštrukcie útočníka, pričom majiteľ ani nespozoruje, že jeho zariadenie bolo infikované a stalo sa súčasťou botnetu. Obeť môže byť infikovaná prostredníctvom webových stránok, narušenia autentifikácie pre vzdialený prístup, alebo zneužitím prílohy v elektronickej pošte. Po nainštalovaní malware sa bot pripojí ku command and control serveru (C&C) (na obrázku vľavo) a čaká na inštrukcie. Pripojenie k C&C je realizované ako pripojenie k bežnej webovej doméne, alebo napríklad prostredníctvom aplikačného protokolu IRC (Internet Relay Chat) [23]. Akonáhle útočník (ľavý horný roh obrázku) iniciuje útok prostredníctvom C&C serveru, server rozošle potrebné inštrukcie zotročeným zariadeniam, ktoré následne začnú generovať samotný útok.

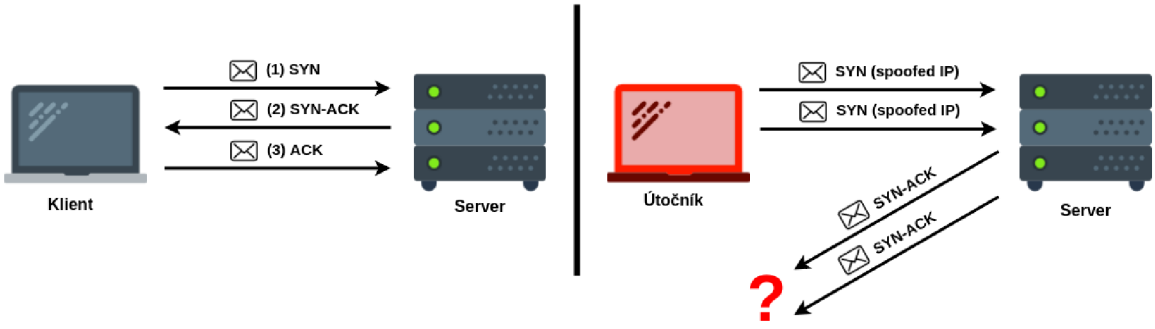



Obr. 2.2: Realizácia DDoS útoku prostredníctvom siete Botnet, tvorenou Zombie zariadeniami, ktorú útočník riadi pomocou Command and Control serveru (C&C).

O tom, že táto téma je stále aktuálna, svedčí aj nedávny útok uskutočnený 28. februára 2018, ktorého cieľom bola webstránka Github [20]. Ide o doposiaľ najväčší DDoS útok v histórií Internetu, ktorý dosiahol hodnotu 1,35 Tb/s. Môžeme však takmer s istotou predpokladať, že aj tento útok bude časom prekonaný, nakoľko sila (D)DoS útokov neustále narastá. Pre porovnanie môže byť spomenutý útok, ktorý bol zaznamenaný začiatkom roku 2013 a ktorého cieľom boli webové stránky holandskej antispamovej organizácie Spamhaus [16]. Útok vtedy dosiahol rekordnú hodnotu 300 Gb/s. Útok z februára 2018 teda viac ako štvornásobne prevýšil hodnotu rekordného útoku z pred piatich rokov. Je nutné podotknúť, že okrem sily útokov sa enormne zvyšuje aj počet útokov. Jeden z poskytovateľov systému na ochranu pred (D)DoS útokmi, na svojich stránkach uviedol skutočnosť, že napríklad počet Amplifikačných DNS útokov (popísaných nižšie v sekcii 2.1) v prvom štvrťroku roku 2018 vzrástol medzikvartálne o 201,42 % a dokonca až o 660,92 % medziročne [21].

### TCP SYN Flood

TCP protokol [12], je protokol transportnej vrstvy modelu OSI [1], ktorý zaručuje spoľahlivé doručenie paketov od odosielateľa k príjemcovi. Spoľahlivosť je zaručená tým, že pred posielaním samotných dát, TCP protokol nadviaže spojenie medzi koncovými zariadeniami. Spojenie je nadviazané mechanizmom three way handshake (graficky znázornený v ľavej časti obrázku 2.3). Pri tomto mechanizme iniciuje nadviazanie spojenia klient tým, že pošle na server paket s príznakom SYN. Server v prípade dostupnosti pridá do frontu neúplných spojení záznam pre danú žiadosť o nadviazanie spojenia, čím na určitý čas alokuje zdroje pre dané spojenie a odpovedá klientovi zaslaním paketu s príznakmi SYN-ACK. Po prijatí paketu od serveru klient odpovie serveru paketom s príznakom ACK. Server po prijatí ACK paketu vyberie odpovedajúci záznam z frontu neúplných spojení, čím je spojenie nadviazané a môže začať preposielanie samotných dát. Pri TCP SYN Flood útoku [7] (ilustrovaný




Obr. 2.3: Porovnanie použitia mechanizmu three way handshake pre nadviazanie TCP spojenia medzi klientom a serverom a zneužitia tohto mechanizmu pri TCP SYN Flood útoku.

v pravej časti obrázku 2.3), útočník zneužíva na zneprístupnenie služby práve spomínaný mechanizmus three way handshake a to tak, že opakovane zasiela SYN pakety na port cieľového zariadenia, ktorý používa služba, ktorá má byť zneprístupnená. Server pre každý prijatý paket alokuje zdroje a odpovedá paketom SYN-ACK. Útočník, ktorý vo väčšine prípadov používa pri zasielaní SYN paketov falošnú IP adresu, pakety od severu vôbec neprijme, alebo serveru neodpovedá. Na strane servera teda zostávajú otvorené neúplné spojenia až po kým neuplynie doba ich exspirácie. Počet možných neúplných spojení pre každý port je limitovaný a po určitom množstve prijatých SYN paketov sa front neúplných spojení zaplní a server nemá možnosť nadviazať ďalšie spojenia či už s útočníkom, alebo s legitímnymi užívateľmi a služba sa stane nedostupná. Proti tomuto typu útokov sa dá brániť napríklad technikou recyklovania najstarších neúplných spojení vo fronte po zaplnení jeho kapacity, alebo využívaním SYN cookies [7], pri použití ktorých nedochádza k zaplňovaniu frontu neúplných spojení.

### UDP Flood

Útočník pri útoku typu UDP flood [7] zasiela na cieľový systém nadmerné množstvo UDP [26] paketov za účelom zahltenia systému. Počas bežnej prevádzky server po prijatí UDP paketu na niektorý zo svojich portov najprv skontroluje, či na danom porte nejaká aplikácia očakáva prijímanie paketov. Ak systém vyhodnotí, že paket na danom porte neočakáva žiadna z aplikácií, snaží sa o tom informovať odosielateľa paketu prostredníctvom ICMP [27] správy o nedostupnosti cieľa (Destination Unreachable). V dôsledku toho, že pri prijatí každého paketu systém spotrebuje určitú časť zdrojov na overenie toho, či je paket žiadúci sa pri dostatočnom množstve paketov môžu vyčerpať zdroje systému. Druhým možným dôsledkom tohoto útoku je vyčerpanie kapacity prenosového pásma dostatočne veľkým množstvom správ odoslaných útočníkom ako aj ICMP správ generovaných serverom. Vo väčšine prípadov pri tomto útoku útočník požíva falošné IP adresy, podvrhnuté pomocou techniky IP Spoofingu [7] a tým zabraňuje prijímaniu ICMP správ od servera. Služba zasiahnutá týmto typom útoku sa môže javiť ako veľmi pomalá, prípadne môže byť nedostupná. V súčasnosti sa tomuto typu útokov zabraňuje napríklad limitovaním počtu generovaných ICMP odpovedí operačným systémom, alebo firewallom.

### ICMP Flood

Tento útok býva tiež nazývaný Ping Flood [7], pretože využíva rovnaké ICMP [27] správy ako nástroj `ping` [10], ICMP `ECHO_REQUEST` a ICMP `ECHO_RESPONSE`. Každé sieťové zariadenie, prijímajúce správu `ECHO_REQUEST` na túto správu odpovedá jej odosielateľovi správou `ECHO_RESPONSE`. Útok teda cieli buď na vyčerpanie zdrojov systému, ktoré sú spotrebovávané pri obsluhe prichádzajúcich správ, ale aj na vyčerpanie prenosovej kapacity v dôsledku veľkého množstva prijímaných, ale aj odosielaných správ. Tomuto typu útok sa dá brániť podobne ako UDP Flood útokom, a to limitovaním počtu ICMP správ.

### HTTP Flood

Pri tomto type DDoS útokov sa útočník snaží zahltiť systém obete obrovským množstvom HTTP [9] správ. Protokol HTTP sa bežne používa na načítanie obsahu web stránok, alebo zasielanie klientskych dát načítaných prostredníctvom HTML formulárov. Pri tomto type útoku útočník zvyčajne využíva botnet pre dosiahnutie čo najväčšej efektivity.

Existujú dva varianty tohoto útoku a prvý z nich sa nazýva HTTP GET Flood [7]. Ako aj z názvu vyplýva, pri tomto variante budú za účelom zahltenia systému použité HTTP správy typu `GET`. Útočiace zariadenia pomocou týchto správ koordinovane žiadajú od serveru obrázky, súbory, alebo iné dáta. Cieľom je, aby na pomerne malú požiadavku `GET` server reagoval veľkou odpoveďou. Systém je v dôsledku toho zahltený obrovským množstvom správ od útočiacich zariadení a odpovedí na tieto správy a nie je schopný reagovať na požiadavky legitímnych užívateľov.

Pri druhom variante, nazývanom HTTP POST Flood [7], útočník posiela na cieľový systém veľké množstvo HTTP správ typu `POST`. Server pri spracúvaní týchto správ musí zvyčajne vyhodnotiť prijaté dáta a napríklad uskutočniť zmenu v databáze. Spracovanie `POST` správy na strane servera je teda v porovnaní s vytvorením a odoslaním tejto správy na strane klienta omnoho náročnejšie. Druhý variant útoku využíva práve túto vlastnosť, prostredníctvom ktorej sa snaží o spotrebovanie zdrojov systému a tým jeho zneprístupnenie.

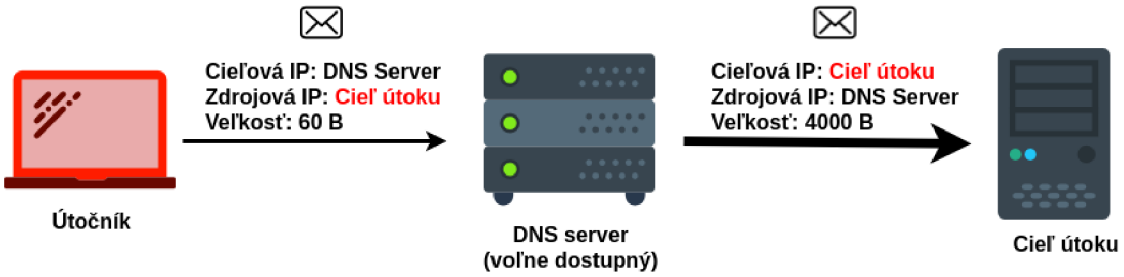
Proti tomuto typu útokov sa možno brániť napríklad používaním Web Application Firewallu, ktorý slúži na monitorovanie a filtrovanie HTTP prevádzky medzi webovou aplikáciou a zariadeniam v sieti Internet.

### Amplifikačné DDoS útoky

Informácie uvedené v nasledujúcej sekcii boli vytvorené na základe týchto zdrojov: [31, 6, 22]. Charakteristikou amplifikačných útokov je to, že využívajú efekty zosilnenia a zrkadlenia, respektíve odrazu, ktoré sú znázornené na obrázku 2.4.

Zrkadlenie, alebo odraz, je mechanizmus, pri ktorom útočiace zariadenie neposiela pakety priamo na zariadenie, ktoré je cieľom útoku, ale na takzvaný reflektor. Reflektor je server, ktorý je voľne dostupný v sieti Internet a útočník ho nijak nevlastní, ani nemá nad ním kontrolu. Na tomto serveri zvyčajne bežia služby ako DNS [19], NTP [18], alebo SNMP [3]. Útočiace zariadenia na tento server posielajú UDP pakety, ktorých zdrojová IP adresa sa zhoduje s IP adresou cieľa útoku. Server, ktorý vníma UDP pakety ako legitímne, na základe obsahu paketu generuje odpoveď, ktorú vďaka podvrhnutej IP adrese posiela na cieľové zariadenie.

Efekt zosilnenia je založený na zrkadlení a jeho podstatou je to, aby útočiace zariadenie pomocou pomerne malého množstva vynaložených zdrojov dokázalo cieľové zariadenie zahl-




Obr. 2.4: Efekt zrkadlenia a zosilnenia, využívaný pri amplifikačných DDoS útokoch, pri ktorom je ako reflektor použitý verejne dostupný DNS server.

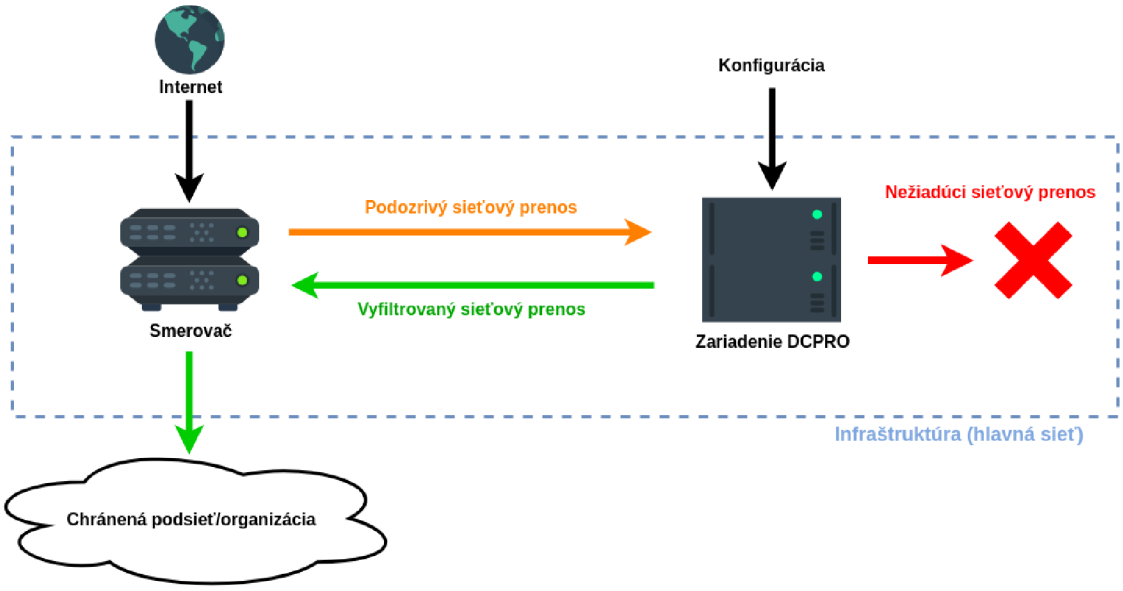
tiť veľkým množstvom dát a na strane cieľového zariadenia tak spotrebovať niekoľkonásobne väčšie množstvo zdrojov.

Príkladom takéhoto typu útoku môže byť DNS aplifikačný útok [11], ktorý pre zosilnenie zneužíva systém doménových mien DNS a DNS servery. DNS server teda slúži ako reflektor, na ktorý zariadenia ovládané útočníkom zasielajú požiadavky na kompletné záznamy určitej doménovej adresy. Konkrétna doménová adresa, ktorej záznamy budú požadované, je vybraná tak, aby obsahovala čo naviac záznamov, a teda aby bola následná odpoveď čo najväčšia. Zdroj [11] uvádza, že odpoveď od DNS serveru môže byť v niektorých prípadoch až 100 násobne väčšia ako požiadavka, na základe ktorej bola vygenerovaná.

Ďalším zástupcom amplifikačných útokov je NTP amplifikačný útok [7], zneužívajúci NTP protokol [18], ktorý slúži na synchronizáciu času medzi zariadeniami v sieti. Útočiace zariadenia posielajú na verejne dostupný NTP server UDP pakety obsahujúce príkaz monlist. Odpoveďou na tento príkaz je správa, ktorá obsahuje IP adresy posledných 600 zaradení, ktoré zaslali požiadavky na daný NTP server. Server, ktorý ma v pamäti zaznamenaných 600 adries, vygeneruje odpoveď, ktorá bude 206 krát väčšia ako požiadavka, ktorá ju iniciovala. Takýto efekt zosilnenia je však možné dosiahnuť iba na serveroch s verziou systému NTP nižšou ako 4.2.7, nakoľko od tejto verzie je príkaz monlist zablokovaný.

## 2.2 Zariadenie pre ochranu pred DoS útokmi DCPRO

Text nasledujúcej sekcie vychádza z informácií uvedených v diplomových prácach [15, 32], informácií uvedených na wiki stránkach projektu DCPRO [5] a užívateľskej príručke zariadenia DCPRO [4]. Zariadenie vyvíjané v rámci združenia CESNET s názvom DCPRO je určené predovšetkým na detekciu a ochranu pred volumetrickými DoS útokmi. Z pohľadu koncovej siete je ochrana pred volumetrickými útokmi problematická, nakoľko tieto útoky typicky zahltia vstupnú linku koncovej siete, proti čomu sa daná sieť nemá možnosť brániť. Z toho dôvodu je jednoduchšie riešiť detekciu a filtrovanie týchto útokov ešte pred vstupom do koncovej siete, a to na vyššej úrovni sieťovej infraštruktúry v sieťach s vysokými prenosovými rýchlosťami typicky 100 Gb/s. Vzhľadom k nutnosti vysokej rýchlosti spracovania a vysokej dátovej priepustnosti je využívaná hardvérová akcelerácia, realizovaná pomocou technológie FPGA. Obrázok 2.5 zobrazuje spôsob nasadenia zariadenia do siete. Zariadenie v sieti figuruje ako L3 zariadenie, teda má IP adresu, na ktoré je prostredníctvom smerovačov presmerovaná podozrivá sieťová prevádzka, ktorú smerovače pomocou vlastných pravidiel nedokážu dostatočne vyfiltrovať. Zariadenie túto komunikáciu zanalyzuje a ak vyhodnotí niektoré pakety ako nebezpečné, budú dané pakety z komunikácie vyfiltrované. V opačnom prípade zariadenie prepošle pakety naspäť do chránenej infraštruktúry. O zaho-




Obr. 2.5: Spôsob nasadenia zariadenia na ochranu pred (D)DoS útokmi DCPRO do siete.

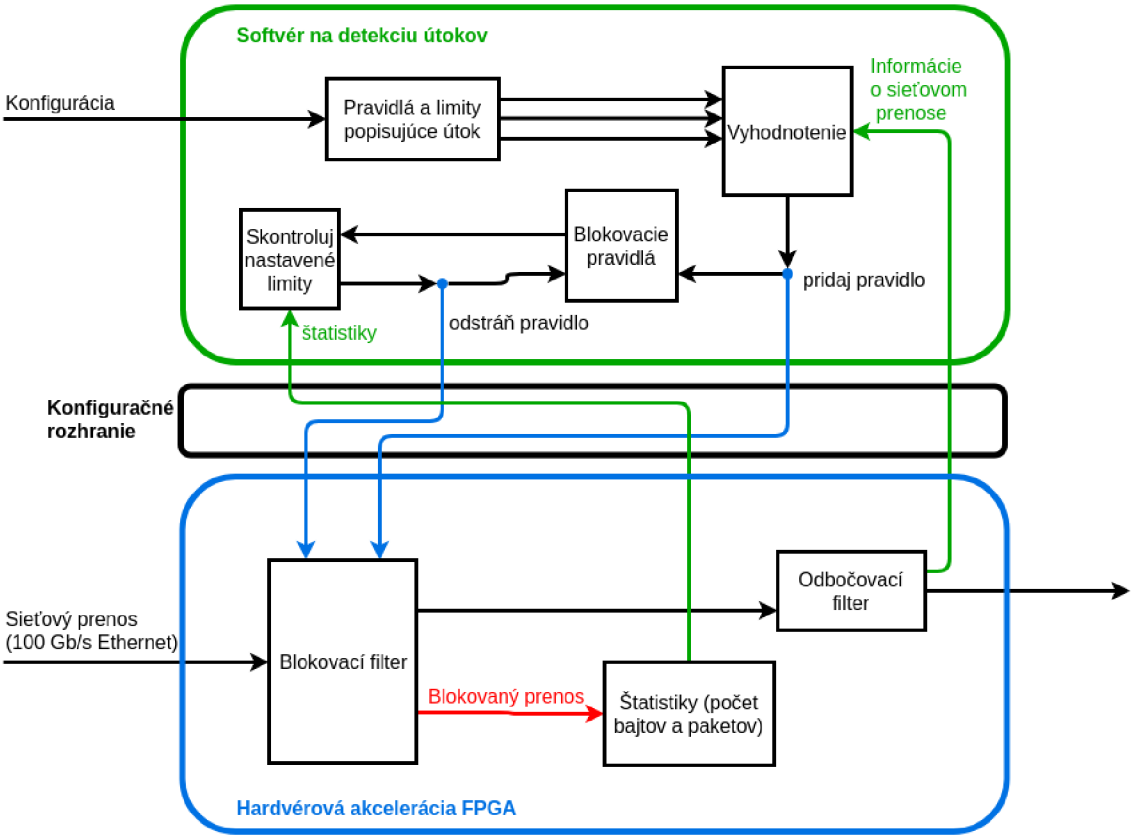
denej sieťovej prevádzke sú ďalej zaznamenávané štatistiky, ako počet zahodených bajtov, paketov a podrobnejšie štatistiky na báze sieťových tokov.

### Architektúra zariadenia

Zariadenie DCPRO sa skladá z dvoch hlavných častí, ktoré medzi sebou komunikujú prostredníctvom konfiguračného rozhrania, a to hardvérového akcelerátoru FPGA (spodná, modrá časť obrázku 2.6), ktorý spracováva pakety tvoriace sieťovú prevádzku a softvérovej časti (vrchná zelená časť obrázku 2.6), ktorá uskutočňuje detekciu útokov a riadi hardvérovú časť zariadenia.

K tomu, aby bola dosiahnutá prenosová rýchlosť 100 Gb/s, je uspôsobená činnosť hardvérového akcelerátora, ktorého úlohou je predspracovanie a predanie prijatých paketov softvérovej časti a následné filtrovanie paketov na základe konfigurácie vytvorenej softvérovou časťou. Predspracovanie je realizované pomocou odbočovacieho filtra a spočíva v transformácii paketov na takzvané UH hlavičky, s ktorými ďalej pracuje softvérová časť zariadenia. UH hlavičky majú jasne definovaný jednotný formát a obsahujú IP adresy, čísla portov, protokol, dĺžku paketov a ďalšie informácie potrebné pri vyhodnocovaní v softvérovej časti. Vďaka tomu je softvérová časť odbremenená od spracovania celých paketov a tým je dosahovaná požadovaná rýchlosť.

Vyhodnocovanie toho, či sa jedná o útok, prebieha v softvérovej časti, kde sú jednotlivé UH hlavičky kontrolované na základe definovaných pravidiel. Pravidlá sa skladajú z konkrétne stanovených hodnôt, ako sú rozsahy zdrojových a cieľových IP adries, čísla zdrojových a cieľových portov, transportných protokolov, dĺžky a fragmentácie paketov a hodnôt protokolu TCP. Pravidlá okrem toho obsahujú hodnoty limitov maximálneho povoleného množstva prenesených dát daného prenosu, uvádzané v počte paketov, alebo bajtov. Po prekročení týchto limitov bude konkrétny prenos vyhodnotený ako útok a na základe pravidiel, ktorých limity boli prekročené, softvér sformuluje nové pravidlá obsahujúce zdrojové IP adresy a popis konkrétneho prenosu zahŕňajúci čísla portov, protokol, TCP flagy, fragmentáciu, veľkosť a limity. Tieto pravidlá sú pridané k pravidlám používaným v softvérovej časti



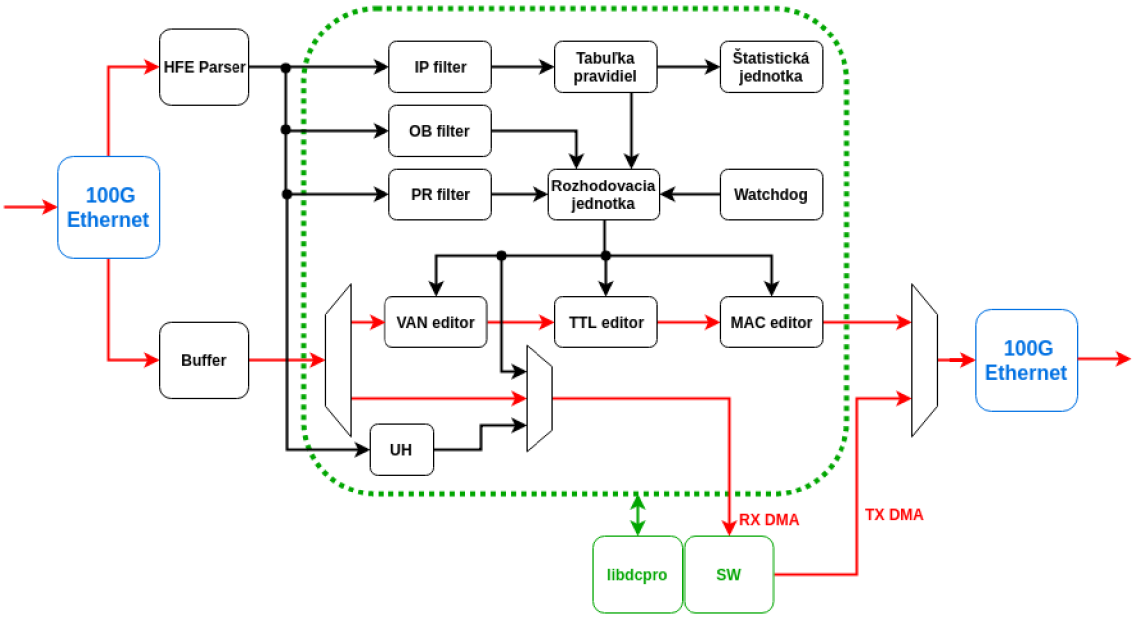

Obr. 2.6: Bloková schéma zariadenia pre ochranu pred DoS útokmi, obsahujúca dve hlavné časti zariadenia, softvérovú časť a hardvérovú akceleráciu FPGA.

a tiež je vygenerovaná nová konfigurácia, ktorá je nahraná do blokovacieho filtru nachádzajúceho sa v hardvérovej časti zariadenia, ktorý vykonáva filtrovanie paketov. V hardvérovej časti sú o zablokovanom prenose zaznamenávané štatistiky obsahujúce informácie o počte zablokovaných paketov a bajtov, viazané k jednotlivým blokovacím pravidlám. Softvérová časť na základe týchto štatistík periodicky overuje, či u blokovaných prenosov dochádza k prekračovaniu limitov. Ak dané limity prekračované nie sú, konkrétne pravidlá sú odstránené z množiny pravidiel používanej softvérom a tiež je aktualizovaná konfigurácia blokovacieho filtru.

### 2.2.1 Firmvér zariadenia

V nasledujúcej časti bude popísaná architektúra firmvéru zariadenia, ktorú je nutné uviesť pre správne pochopenie významu a zamerania jednotlivých testov, ktoré budú vytvorené. Schéma firmvéru je zobrazená na obrázku 2.7). Štruktúra firmvéru sa skladá z troch častí, dátovej cesty (znázornená červenou farbou), riadiacej cesty (znázornená čiernou farbou) a cesty používanej na konfiguráciu firmvéru prostredníctvom softvérového rozhrania (znázornená zelenou farbou).

Pomocou dátovej cesty sú pakety prenášané medzi jednotlivými modulmi firmvéru. Moduly využívajúce tieto dáta možno rozdeliť do dvoch skupín. Do prvej skupiny patria moduly HFE (Header Field Extractor) a vyrovnávacia pamäť (Buffer). Tieto moduly spracovávané pakety neupravujú, iba z nich extrahujú dáta potrebné pre ďalšie spracovanie. Do druhej




Obr. 2.7: Zjednodušená bloková schéma firmvéru hardvérového akcelerátoru zariadenia na ochranu pred DoS útokmi.

skupiny možno zaradiť moduly VLAN editor, TTL editor a MAC editor. Tieto moduly na základe inštrukcií z riadiacej cesty modifikujú spracovávané pakety. Poradie, v akom prebieha spracovanie jednotlivých modulov, je vyobrazené na obrázku 2.7. Na základe rozhodovacej jednotky budú pakety ďalej spracovávané a upravované.

Riadiacu cestu tvoria moduly HFE, odbočovací filter, IP filter, PR filter, modul s tabuľkou špecifických podmienok, štatistická jednotka a modul na vyhodnocovanie priorít jednotlivých operácií. Pomocou riadiacej cesty sú na základe získaných informácií z prijatých paketov a nastavených riadiacich registrov určené akcie, ktoré majú byť vykonané. Na základe toho sú potom modulom posielané príkazy, ktoré majú byť vykonané nad spracovávanými paketmi.

Knižnica libdcpro poskytuje softvérové rozhranie, prostredníctvom ktorého je možné konfigurovať firmvérové moduly. Toto rozhranie je využívané softvérovým démonom, ktorý na základe výsledkov vyhodnotenia sieťovej prevádzky v softvérovej časti riadi firmvérové moduly. Nad rozhraním knižnice libdcpro sú implementované aj CLI nástroje pre jednotlivé firmvérové moduly, pomocou ktorých možno moduly tiež ovládať napríklad za účelom testovania, alebo ladenia. CLI nástroje budú v rámci testovania zaradenia DCPRO použité pri izolovanom testovaní samostatných firmvérových komponentov.

**Popis hlavných modulov firmvéru:**

**Parser HFE (Header Field Extractor)** je modul vykonávajúci extrahovanie potrebných dát z hlavičiek jednotlivých sieťových protokolov. Extrahované dáta ako IP adresy, MAC adresy, protokoly, čísla portov, dĺžky rámcov a IP paketov sú ďalej poskytované ostatným modulom. Okrem týchto informácií sú modulom poskytované aj presné polohy konkrétnych položiek v paketoch pomocou takzvaného ofsetu. Ofset je potom vo firmvéri využívaný napríklad pri modifikácií dát.

**Vyrovnávacia pamäť (Buffer)** implementovaná ako pamäť typu FIFO, v ktorej sú ukladané pakety po dobu ich vyhodnocovania. Po vyhodnotení je daný paket preposlaný nasledujúcemu modulu na spracovanie.

**VLAN editor** modul slúžiaci na výmenu položky VLAN ID v spracovávanom pakete. Výmena sa vykonáva na základe pravidiel, v ktorých je definovaná pôvodná hodnota VLAN ID a nová hodnota VLAN ID, ktorá má na základe zhody s pôvodnou hodnotu, túto hodnotu nahradiť.

**Modul pre kontrolu TTL** realizuje kontrolu TTL (Time to live) hodnoty. V prípade, že hodnota je jedna, rozhodovacej jednotke je predaný príkaz na zahodenie paketu. V opačnom prípade je hodnota TTL znížená o jedna a taktiež je v hlavičke IP paketu nahradená hodnota kontrolného súčtu za aktualizovanú hodnotu.

**MAC editor** umožňuje vzájomnú výmenu zdrojovej a cieľovej MAC adresy, alebo ich nahradenie za užívateľom definované hodnoty.

**Odbočovací filter (OB filter)** je modul, pomocou ktorého je možné špecifikovať aký sieťový prenos chceme preposielať do operačnej pamäte počítača. Ďalej je možné nastavovať zahadzovanie paketov, určovať, či do operačnej pamäte budú preposielané UH hlavičky, celé pakety, alebo či majú byť pakety skracované.

**Blokovací filter** je tvorených z dvoch modulov IP filtru a tabuľky pravidiel popisujúcich konkrétne sieťové prenosy, ktoré sú vnímané ako nebezpečné. Vyhodnocovanie prebieha tak, že IP filter na základne zdrojovej IP adresy a konfigurácie poskytnutej softvérovou časťou vyberie popis konkrétneho sieťového prenosu. Na základe vybraného popisu je z tabuľky vybrané špecifické pravidlo, ktoré slúži na porovnanie s dátami získanými od HFE modulu. Výsledok porovnania je následne poslaný štatistickej jednotke a rozhodovacej jednotke.

**Generátor UH hlavičiek** na základe informácií z modulu HFE vytvára UH hlavičky, ktoré obsahujú informácie o spracovávaných paketoch ako IP adresy, čísla portov, čísla protokolov a fragmentáciu. Vygenerované hlavičky sú preposielané do softvérovej časti zariadenia.

**Štatistická jednotka** zaznamenáva štatistiky o blokovaných sieťových prenosov, ako počet vyfiltrovaných bajtov a paketov. Štatistiky sú periodicky vyčítané softvérovou časťou, kde sú ďalej vyhodnocované.

**Prefixový filter (PR filter)** je modul, ktorý umožňuje na základe prefixu cieľovej IP adresy rozhodovať o tom, či bude paket preposlaný do operačnej pamäte, na sieťové rozhranie, do oboch uvedených častí, alebo bude zahodený. Filter ďalej na základe prefixu cieľovej IP adresy umožňuje v paketoch nahradzovať cieľovú MAC adresu za MAC adresu, ktorú pre konkrétny prefix definuje užívateľ.

**Watchdog** vykoná predvolenú akciu v prípade, že interný čítač dosiahne maximálnej prednastavenej hodnoty. Povolené akcie sú vypnutie ethernetových liniek (a poslanie príslušnej chybovej správy), zahodenie všetkých paketov, alebo preposlanie všetkých paketov naspäť na rozhrania.

**Rozhodovacia jednotka** na základe vyhodnotenia paketu v IP filtri, obecnom filtri, PR filtri a kontroly TTL určí výslednú akciu. Vyhodnocovanie prebieha v poradí TTL

kontrola, IP filter, PR filter a nakoniec odbočovací filter, ktorý rozhoduje o tom, či bude paket preposlaný do operačnej pamäte, na sieťové rozhranie, alebo do oboch častí.

### 2.2.2 Softvér zariadenia

Pre potreby vývoja, ladenia a testovania existuje rozhranie, pomocou ktorého je možné konfigurovať, prípadne vyčítavať informácie z firmvéru zariadenia. Toto rozhranie je poskytované formou knižnice jazyka C, alebo formou CLI nástrojov. Podobne ako firmvér zariadenia aj rozhranie je rozdelené na jednotlivé moduly a teda jednotlivé firmvérové moduly popísané vyššie možno konfigurovať prostredníctvom príslušného rozhrania. Okrem týchto nástrojov na konfiguráciu modulov existujú aj ďalšie nástroje:

**nfb-eth** – zobrazuje informácie o vstupných a výstupných sieťových rozhraniach, ako napríklad celkový stav rozhraní, počet bajtov, počet prijatých paketov, počet zahodených paketov atď.

**ndp-tool** – tento nástroj môže fungovať vo viacerých módoch. V móde read vykonáva viacvláknové čítanie dát z RX DMA kanálov. V móde generate umožňuje viacvláknové odosielanie paketov na TX DMA kanály.

**ddpd** – jedná sa o softvérového daemona (program bežiaci na pozadí bez priamej kontroly užívateľa), ktorý vykonáva riadenie firmvérových modulov, príjem paketov odoslaných softvérovou cestou a ich spracovanie a odoslanie na výstup. Okrem rozhrania pre komunikáciu s firmvérom, softvér zariadenia zahrňuje aj dva moduly, prostredníctvom ktorých možno vykonávať špecifické filtrovanie podozrivej sieťovej prevádzky a to amplifikačný modul a SYN flood modul.

#### Amplifikačný modul

Prvý modul slúži na detekciu a ochranu pred amplifikačnými DDoS útokmi a jeho správanie je definované zoznamom pravidiel uložených v súbore `amprules.conf`.

Pravidlá popisujúce amplifikačné DDoS útoky majú nasledujúci formát:

```
<RULE> <DST-NET> <PROTOCOL> <SRC-PORT> <DST-PORT> <FRAGMENTATION>
            <TCP-FLAGS> <LENGTH> <THRESHOLD> <LIMIT>
```

Popis jednotlivých položiek tvoriacich pravidlá popisujúce amplifikačné DDoS útoky:

`<RULE>` je voliteľná položka a slúži na uloženie užívateľom definovaného názvu daného pravidla.

`<DST-NET>` je povinná položka a slúži na definovanie chránených IP adries v tvare `dst net` IP/prefix. Za kľúčovými slovami `dst net` môže nasledovať aj viac dvojíc IP/prefix oddelených medzerami, pričom je podporovaná IPv4 aj IPv6.

`<PROTOCOL>` je voliteľná položka a pri filtrovaní je porovnávaná s číslom protokolu v hlavičke IP paketu. Zápis tejto položky je uvedený kľúčovým slovom `protocol` a ďalej môže nasledovať jeden, alebo viac protokolov ako napríklad `TCP`, `UDP`, `ICMP` atď.

`<SRC-PORT>` a `<DST-PORT>` sú voliteľné a slúžia na definovanie zdrojového a cieľového portu. Zapisujú sa v tvare `src port`, alebo `dst port` a následne buď číslo portu, znak porovnávania (`<` a `>`) a číslo portu, alebo rozsah čísel od-do.

`<FRAGMENTATION>` jedná sa o voliteľnú položku, ktorá sa zapisuje v tvare kľúčového slova `fragmentation` nasledovaného jednou z hodnôt `YES` (iba všetky fragmentované pakety), `NO` (iba pakety bez fragmentácie), `FIRST` (iba fragmentované pakety a zároveň iba prvá časť fragmentovaného paketu, okrem fragmentov idúcich po sebe), `LAST` (iba fragmentované pakety a zároveň iba posledná časť fragmentovaného paketu, okrem všetkých predchádzajúcich fragmentov), `MID` (iba fragmentované pakety a zároveň tá časť fragmentovaného paketu, ktorá nie je ani prvá ani posledná) a `NOFIRST` (iba fragmentované pakety a zároveň všetky časti fragmentovaného paketu okrem prvej).

`<TCP-FLAGS>` voliteľná položka určujúca TCP flagy zapisovaná v tvare `tcpflags` a hodnotami `S` (SYN), `A` (ACK), `F` (FIN), `P` (PUSH), `R` (RESET), prípadne znakom negácie (`!`) nasledovaným hodnotou.

`<LENGTH>` je voliteľná položka slúži na špecifikáciu veľkosti Ethernetového rámca a zapisuje sa v tvare `length` medzera číslo, znak porovnávania (`<` a `>`) a číslo, alebo rozsah čísel od-do.

`<THRESHOLD>` je povinná položka a definuje hranicu maximálnej rýchlosti v paketoch za sekundu, alebo bitoch za sekundu, ktorá keď je prekročená, tak sa sieťová prevádzka redukuje na hodnotu nastavenú v položke `<LIMIT>`. Položka `<THRESHOLD>` sa zapisuje v tvare `threshold <VALUE> <UNIT>`, kde `<VALUE>` je hodnota vyjadrená číslom a `<UNIT>` je jednotka. Podporované formáty jednotiek sú `pps`, `kpps`, `Mpps` pre rýchlosť v paketoch za sekundu a `Mbps` a `Gbps` pre rýchlosť v bitoch za sekundu.

`<LIMIT>` je povinná položka a zapisuje sa v tvare `limit <VALUE> <UNIT>`, kde formát položiek `<VALUE> <UNIT>` odpovedá formátu popísanému u položky `<THRESHOLD>`. Po prekročení hodnoty definovanej v položke `<THRESHOLD>` bude sieťová prevádzka redukovaná na rýchlosť definovanú v tejto položke.

### SYN Flood modul

Tento modul slúži na detekciu a ochranu pred SYN Flood útokmi. Pravidlá popisujúce podozrivú sieťovú prevádzku a tým aj správanie tohto modulu sú uložené v konfiguračnom súbore s názvom `synrules.conf`. Formát pravidiel je nasledovný:

`<RULE> <DST-NET> <THRESHOLD> <STRATEGY> <M> <N>`

Význam a syntax jednotlivých položiek pravidla je nasledovná:

`<RULE>` je voliteľná položka a slúži na uloženie užívateľom definovaného názvu daného pravidla.

`<DST-NET>` je povinná položka a slúži na definovanie chránených IP adries v tvare `dst net` IP/prefix. Za kľúčovými slovami `dst net` môže nasledovať aj viac dvojíc IP/prefix oddelených medzerami, pričom je podporovaná IPv4 aj IPv6.

`<THRESHOLD>` je povinná položka a definuje maximálny počet prijatých SYN paketov smerujúcich do chránenej siete. Ak je počet prijatých paketov prekročený zariadenie DCPRO začne filtrovať podozrivú prevádzku smerujúcu do danej siete. Položka `<THRESHOLD>` sa zapisuje v tvare `threshold <VALUE> <UNIT>`, kde `<VALUE>` je hodnota vyjadrená číslom a `<UNIT>` je jednotka. Podporované formáty jednotiek sú `pps`, `kpps`, `Mpps` pre počet SYN paketov za sekundu.

Tabuľka 2.1: Rozhodovacia tabuľka využívaná SYN Flood modulom, v ktorej sú pri rozhodovaní či bude paket prijatý (ACCEPT), alebo zahodený (DROP) zohľadnené hodnoty spodného limitu `M`, vrchného limitu `N` a počet prijatých SYN a ACK paketov.

| **Rozsah** | | **SYN** | | | |
|---|---|---|---|---|---|
| | | **1** | **[2,M]** | **(M,N]** | **N<** |
| **ACK** | **0** | DROP | ACCEPT | ACCEPT | DROP |
| | **0<** | ACCEPT | ACCEPT | ACCEPT | DROP |

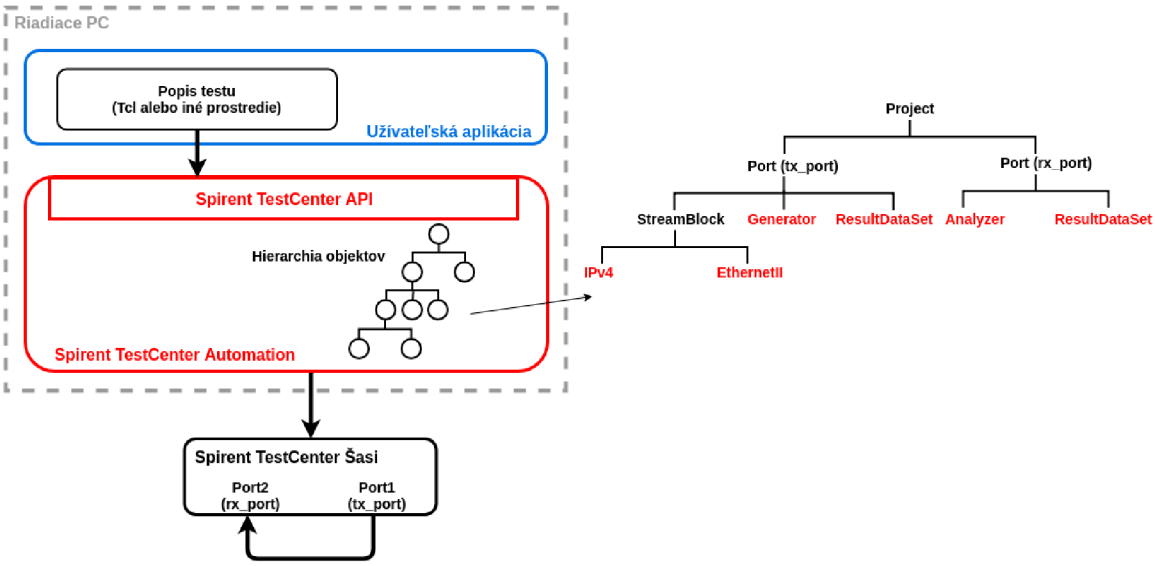
`<STRATEGY>` je povinná položka a zapisuje sa v tvare kľúčového slova `strategy` a hodnotou špecifikujúcou stratégiu ktorá bude použitá na potlačenie prebiehajúceho útoku. Povolené hodnoty tejto položky sú `syn-drop`, `ack-spoof`, `rst-cookie`.

`<M>` je voliteľná položka, ktorá sa ale definuje iba v prípade že bola v položke `<STRATEGY>` nastavená hodnota `syn-drop`. Táto položka špecifikuje spodný limit označovaný písmenom `M` pre počet SYN paketov. Filtrovanie sieťovej prevádzky na základe tohto parametru a parametru `<N>` v závislosti na počte prijatých SYN a ACK paketov je popísaný v tabuľke 2.1. Predvolená hodnota toho parametru je `SYN_PARAM_M=10`.

`<N>` je voliteľná položka, ktorá sa ale definuje iba v prípade že bola v položke `<STRATEGY>` nastavená hodnota `syn-drop`. Táto položka špecifikuje horný limit označovaný písmenom `N` pre počet SYN paketov. Filtrovanie sieťovej prevádzky na základe tohto parametru a parametru `<M>` v závislosti na počte prijatých SYN a ACK paketov je popísaný v tabuľke 2.1. Predvolená hodnota toho parametru je `SYN_PARAM_N=10000`.

## 2.3 Generátor sieťovej prevádzky Spirent TestCenter

Informácie uvedené v tejto sekcii vychádzajú zo zdrojov [34] a [33]. Spirent TestCenter je platforma určená k funkčnému a výkonovému testovaniu na sieťových vrstvách L2 až L7. Tento systém je schopný generovať, ako aj analyzovať sieťovú prevádzku pri prenosových rýchlostiach od 10 Mb/s do 100 Gb/s. Umožňuje detailné nastavenie vlastností generovanej sieťovej prevádzky, čím je možné docieliť emuláciu variabilných typov sieťových infraštruktúr a špecifických scenárov, ako napríklad zahltenie siete obrovským množstvom paketov generovaným veľkým počtom sieťových zariadení. Spirent TestCenter okrem hardvérového šasi zahrňuje aj softvérové balíčky, ako napríklad balíček Spirent TestCenter Automation, ktorý poskytuje rozhranie pre programovanie aplikácií (ďalej len API), pomocou ktorého je možné vytvárať a spúšťať automatizované testy. Tento systém tiež poskytuje grafické užívateľské rozhranie, umožňujúce vytváranie testov, spúšťanie testov a analýzu výsledkov. Analýza výsledkov je možná aj prostredníctvom API a výsledky možno analyzovať počas generovania prevádzky, aj po jej skončení. Vďaka možnosti ukladania a znovu použitia sieťových konfigurácií a možnosti automatizovaného spúšťania generovania sieťovej prevádzky a následnej analýzy je vhodné tento nástroj použiť pri testovaní zariadenia DCPRO. Testy, ktoré sú implementované vo frameworku, ktorý je predmetom tejto práce, sú konceptuálne odlišné, ako testy vytvorené pomocou nástroja Spirent TestCenter. Hlavný rozdiel v konceptoch je ten, že testy nástroja Spirent TestCenter sú vo frameworku použité




Obr. 2.8: Princíp testovania pomocou generátoru sieťovej prevádzky Spirent TestCenter, konfigurovaného užívateľskou aplikáciou prostredníctvom Spirent TestCenter Automation API.

na emuláciu sieťovej infraštruktúry a jej prevádzky a čiastočne k analýze výsledkov, čo tvorí len určitú podmnožinu požiadaviek, ktoré musí test v rámci frameworku spĺnať. Pri tvorbe testov je teda nutné rozumieť princípom testovania pomocou platformy Spirent TestCenter, ktoré sú popisované v tejto kapitole.

### Princíp testovania pomocou Spirent TestCenter

Softvérový balíček Spirent TestCenter Automation poskytuje API, prostredníctvom ktorého možno vytvárať a spúšťať testy a získavať detailné výstupy testov. Základom je API používajúce jazyk Tcl, ale existujú aj ďalše API používajúce napríklad jazyk Python. V koncepte nástroja Spiret TestCenter, graficky znázorneného na obrázku 2.8, je test definovaný pomocou:

- softvérovej konfigurácie skladajúcej sa z množiny príkazov popisujúcich emulovanú sieť a sieťovú prevádzku, ktorá má byť sieťou generovaná
- hardvérovej konfigurácie, ktorá zahrňuje šasi nástroja Spirent TestCenter prepojené pomocou vstupných a výstupných portov s testovaným zariadením, alebo s testovanou sieťovou infraštruktúrou

Obrázok 2.8 zobrazuje testovacie prostredie, v ktorom sa nachádza riadiaci počítač (ľavá horná čas obrázku, ohraničená šedou farbou) a Spirent TestCenter šasi (znázornené čiernou farbou v ľavej spodnej časti obrázku), ktoré pre zjednodušenie ukážky využíva dva navzájom prepojené porty. Jeden port slúži na posielanie prevádzky a druhý na prijímanie. Pri nasadení zariadenia v reálnych podmienkach, môžu byť porty pripojené napríklad k hraničnému smerovaču testovanej sieťovej infraštruktúry.

Na riadiacom počítači beží užívateľská aplikácia (znázornená modrou farbou) a softvérová časť zariadenia Spirent TestCenter (znázornená červenou farbou) poskytujúca užívateľovi API prostredníctvom, ktorého možno konfigurovať a riadiť zariadenie a analyzovať

Tabuľka 2.2: Najčastejšie používané typy objektov, ich popis a príklad atribútov.

| **Typ objektu (príklady)** | **Význam** | **Atribúty (príklady)** |
|---|---|---|
| Project | Koreňový objekt Hierarchie, ktorý musí obsahovať každá testová konfigurácia. | Active<br>Name |
| Port | Objekt typu Port je potomok objektu Project | Active<br>Location Name |
| Host | Objekt typu Host je potomok objektu Project. Tento typ objektov sa používa na emuláciu sieťových hosťov. | Active<br>DevideCount |
| StreamBlock | Definuje vlastnosti toku sieťovej prevádzky. Prostredníctvom jeho potomkov môže definovať rôzne typy a vlastnosti sieťovej prevádzky. Konkrétne vlastnosti toku potom dosiahneme aktiváciou konkrétnych StreamBlockov. | Active<br>FixedFrameLength<br>InterFrameGap |
| EthernetII | Tento typ objektu je príkladom hlavičkového objektu. Pomocou neho možno definovať dáta obsiahnuté v Ethernetových rámcoch sieťovej prevádzky. | destMac<br>etherType<br>preamble<br>srcMac |
| Router | Objekt, pomocou ktorého možno emulovať Router. | Active<br>DeviceCount<br>RouterId |

výsledky. Užívateľskou aplikáciou môže byť napríklad skript napísaný v jazyku Python, alebo Tcl, tvoriaci softvérovú konfiguráciu testu, ktorý má byť vykonaný. Aplikácia prostredníctvom API predáva konfiguráciu do softvérovej časti zariadenia. Softvér na základe konfigurácie testu získanej prostredníctvom API vytvorí hierarchiu objektov, ktorá predstavuje internú reprezentáciu konfigurácie testu. Pravá časť obrázku zobrazuje príklad hierarchie objektov obsahujúcu dva porty, jeden na generovanie sieťovej prevádzky (tx_port) a druhý na prijímanie prevádzky (rx_port). Hierarchia tiež obsahuje objekty potrebné na generovanie a špecifikáciu prevádzky pozostávajúce z rámcov typu Ethernet a objekty na analýzu prevádzky. Každá hierarchia je tvorená objektami, atribútmi špecifikujúcimi vlastnosti objektov a vzťahmi medzi objektami. Objekty reprezentujú rozličné komponenty testovej konfigurácie a tabuľka 2.2 obsahuje príklad často používaných typov objektov, ich popis a atribúty. Objekty, ktorých názvy sú v pravej časti obrázku 2.8 zobrazené červenou farbou, softvér vytvoril automaticky. Napríklad, keď vytvoríte objekt StreamBlock, systém automaticky vytvorí objekty typu EthernetII a IPv4, ktoré sú potomkami objektu StreamBlock.

Pred spustením testu uskutoční softvérová časť validáciu danej hierarchie. Ak je konfigurácia korektná, softvérová časť vykoná test prostredníctvom komunikácie so šasi. Konfigurácia prepojenia softvérovej časti a šasi nie je súčasťou testovej konfigurácie poskytovanou užívateľom, alebo jeho aplikáciou. Softvér počas behu testu udržuje hierarchiu objektov a taktiež zhromažďuje výsledky a ukladá ich do hierarchie. Základné činnosti uskutočňované počas behu testu sú generovanie a analýza sieťovej prevádzky. Tieto operácie vykonávajú objekty typu Generator a Analyzer. Užívateľská aplikácia môže vyčítať výsledky a manipulovať s objektami počas behu testu. Výsledky testu sú dostupné užívateľskej aplikácií prostredníctvom API, alebo v súbore s výstupom testu. Pri testovaní systém generuje viaceré výstupy zahrňujúce výsledky testov, ale aj záznamy logov.

Všetky uvedené funkcie a operácie možno vykonávať aj prostredníctvom grafického užívateľského rozhrania. Spôsoby, akými je možné zariadenie Spirent TestCenter ovládať a konfigurovať, sa dajú aj kombinovať, napríklad za účelom zjednodušenia práce. Pomocou prístupu cez grafické rozhranie môžeme napríklad vytvoriť súbor s testovacou konfiguráciou obsahujúcou objekty a ich atribúty. Následne môžeme danú konfiguráciu načítať a načítané objekty upravovať a používať pri spúšťaní testov automatizovane pomocou API.

## 2.4 Systém kontinuálnej integrácie Jenkins

Ak nebude uvedené inak, všetky informácie v tejto sekcii vychádzajú z dokumetácie systému Jenkins dostupnej na stránkach [13]. Jenkins je open-source systém, ktorý primárne slúži k automatizácií rozličných operácií spojených s kompiláciou, testovaním a dodávaním softvéru. Systém Jenkins môže byť tiež označený ako nástroj na podporu a automatizáciu techniky kontinuálnej integrácie (angl. Continuous Integration). Kontinuálna integrácia je podľa [17] technika vývoja softvéru, pri ktorej členovia vývojového tímu integrujú svoju časť práce do výsledného systému na pravidelnej báze. Každý proces integrácie v sebe zahŕňa kompiláciu zdrojových súborov a automatizované testovanie, čo má za následok detekciu implementačných chýb vo fáze vývoja.

Za účelom automatizácie uvedených operácií systém Jenkins poskytuje rozšíriteľnú sadu pluginov nazývanú Jenkins Pipeline (alebo iba Pipeline). Pipeline umožňuje modelovanie jednoduchých, ale aj zložitých procesov integrácie a dodávania softvéru na úrovni zdrojového kódu. Definícia Pipeline je typicky zapísaná v textovom súbore nazývanom `Jenkinsfile`, ktorý býva väčšinou umiestnený v koreňovom adresári projektu (s podporou systému na správu verzií), ktorý má byť integrovaný. Po vytvorení Pipeline systém Jenkins automaticky reaguje na zmeny v úložisku daného projektu, ako je vytváranie vetiev, alebo zmena obsahu a vykonáva procesy integrácie definované v súbore `Jenkinsfile`.

Súbory `Jenkinsfile` majú definovanú syntax a proces integrácie sa v nich delí do viacerých etáp, ktoré sa syntakticky označujú názvom `stage`. Tri základné etapy, ktoré by mal obsahovať každý proces integrácie sú kompilácia, testovanie a dodanie (nasadenie). Každá etapa je tvorená špecifickými krokmi zapisovanými ako `step`. Každý krok je samostatná operácia, alebo príkaz, ktorý má byť vykonaný. Ako príklad kroku môže byť uvedené spustenie príkazu `make`, pomocou shellu. Pomocou krokov teda možno nadefinovať aké operácie a v akom poradí sa budú vykonávať v jednotlivých etapách.

Pipeline ďalej umožňuje tvoriť zložitejšie syntaktické konštrukcie, ako napríklad zaobalenie kroku do `retry` bloku, ktorý umožní opakované vykonanie zaobaleného kroku v prípade neúspechu. Blok `timeout` umožňuje nastaviť maximálnu dobu vykonávania zaobaleného kroku, po ktorej dosiahnutí bude výsledkom Pipeline zlyhanie. Blok post slúži na definíciu akcií, ktoré majú byť uskutočnené po vykonaní etáp, z ktorých pozostáva Pipeline.

K rôznym výsledkom behu možno nastaviť rôzne akcie, napríklad v prípade neúspechu môže byť vytvorená notifikácia, ktorá bude užívateľovi zaslaná v podobe emailu. Pomocou povinného bloku `agent` sa definuje, ako a v akom prostredí bude vykonanie Pipeline uskutočnené. Táto vlastnosť umožňuje napríklad použitie dopredu vytvorených Docker [8] kontajnerov, čo oslobodzuje Pipeline od konfigurácie potrebných systémových nástrojov a závislostí. Syntaktické pravidlá ďalej umožňujú použitie premenných, ktoré môžu byť používane napríklad pri autentifikácii prístupov. Syntax ďalej umožňuje prijímať vstupy od užívateľa, čím napríklad možno implementovať mechanizmus, v ktorom je nutná interakcia užívateľa pred vykonaním nadchádzajúcej etapy.

V kontexte tejto práce bude systém Jenkins využitý na automatizované spúšťanie testov softvéru aj firmvéru zariadenia DCPRO, ktoré budú overovať funkcionalitu v testovacom prostredí so skutočnou sieťovou prevádzkou generovanou systémom Spirent TestCenter. Toto automatizované testovanie rozšíri už existujúci proces kontinuálnej integrácie zariadenia DCPRO, ktoré doposiaľ zahrňuje kompiláciu a verifikáciu softvéru aj firmvéru zariadenia.

## 2.5 Testovacie frameworky pre jazyk Python

Pre implementáciu prostredia a samotných testov bol vybraný jazyk Python a to hlavne z toho dôvodu, že systém Spirent TestCenter poskytuje API pre jazyk Python. Pre tento jazyk existuje niekoľko frameworkov, ktoré sú určené na uľahčenie, alebo automatizáciu testovania, a preto je vhodné zvážiť ich aspoň čiastočné použitie pri testovaní zariadenia DCPRO.

### Unittest framework

Testovací framework tiež nazývaný PyUnit [30], je to štandardný framework pre tvorbu unit testov v jazyku Python. Poskytuje podporu pre automatizáciu testov, zdieľanie kódu prostredníctvom `setUpk` a `tearDownk` metód, agregáciu testov do kolekcií. Jeho najväčšou výhodou v kontexte automatizácie testovania zariadenia DCPRO je to, že poskytuje rôzne rozšírenia, ako napr. HtmlTestRunner [28]. Toto rozšírenie generuje z výstupu testov HTML [2] dokument, ktorý je pre užívateľa čitateľnejšou a prehľadnejšou alternatívou ako textový výstup v konzole.

### Nose framework

Framework nose [25] rozširuje funkcionalitu načítavania a spúšťania testov napísaných vo frameworku unittest a uľahčuje tak tvorbu testov, ich automatické vyhľadávanie a spúšťanie. Umožňuje agregáciu testov do skupín automaticky, bez nutnosti vytvárania kolekcií, ak sú dodržané konvencie pre tvorbu testov v tomto jazyku, ako napr. správna štruktúra zdrojových súborov a správne názvoslovie. Výstup testov je podobný ako u unittest, ale zahŕňa aj štandardný výstup testov, ktoré skončili neúspechom, pre jednoduchšie debugovanie. Tento framework taktiež poskytuje rozšírenie nose-html-reporting [29], ktoré umožňuje transformáciu výstupu testov do podoby HTML [2] dokumentu.

### Použiteľnosť frameworkou pri testovaní zariadena DCPRO

Nevýhodu oboch frameworkov, vzhľadom k použitiu pri automatizácii testovania zariadenia DCPRO je to, že sú primárne určené na tvorbu unit testov. Testovanie zariadenia DCPRO

je však zamerané na overenie funkcionality prostredníctvom funkčných testov, ktoré zahŕňajú opakované spúšťanie komplexnejších testovacích scenárov s použitím rôznych testovacích dát, konfiguráciu testovaného zariadenia DCPRO a riadenie systému Spirent TestCenter. Teda použitie konceptu uvedených testovacích frameworkov by mohlo znamenať nutnosť prispôsobenia štruktúry testov pravidlám frameworku a tým obmedzenie z hľadiska tvorby zložitejších testovacích scenárov a z hľadiska ďalšej rozšíriteľnosti testov. Vzhľadom na nevýhody uvedených frameworkov a za účelom jednoduchého vytvárania a rozširovania testov pre zariadenie DCPRO bol vytvorený nový framework, ktorý je predmetom tejto práce. Návrh a implementácia tohto frameworku je popísaná v nasledujúcich kapitolách.

# Kapitola 3

# Návrh systému

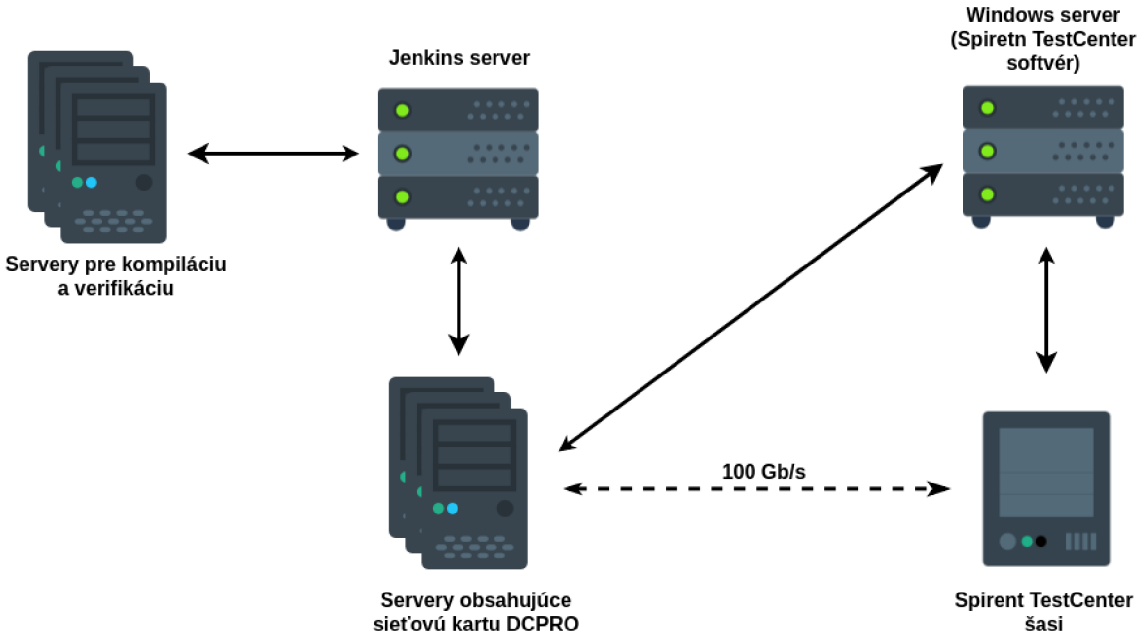
V tejto kapitole je popísaný návrh systému pre testovanie zariadenia DCPRO, ktorý obsahuje popis prostredia a zoznam komponentov, ktoré sú pri testovaní použité. Ďalej je u každého komponentu stručne vysvetlený jeho význam v kontexte tohto systému, jeho úlohy a spôsob interakcie s ostatnými popisovanými komponentami. V ďalšej časti tejto kapitoly sa nachádza stručný súhrn požiadaviek na testovací systém a štruktúru testov, ktoré musí výsledná implementácia spĺňať. V závere kapitoly je uvedený návrh štruktúry systému a testov a návrh výstupu testov.

## 3.1 Prostredie pre testovanie zariadenia DCPRO

Návrh testovacieho prostredia určeného na funkčné testovanie zariadenia DCPRO znázorneného na obrázku 3.1 sa skladá zo serveru, na ktorom beží systém Jenkins, množiny testovacích serverov, ktoré obsahujú sieťovú kartu DCPRO, riadiaceho počítača, na ktorom beží softvérová časť systému Spirent TestCenter (ďalej len STC) a hardvérového šasi systému STC.

Jenkins server (zobrazený v strede vrchnej polovice obrázku 3.1) je server, na ktorom beží systém Jenkins, ktorého úlohou je iniciovať spúšťanie automatizovaných testov zariadenia DCPRO. V sieťovej infraštruktúre združenia CESNET sa nachádza niekoľko fyzických serverov, ktoré obsahujú sieťovú kartu DCPRO. Samotné testovanie prebieha tak, že systém Jenkins vyberie jeden z množiny dostupných serverov obsahujúcich zariadenie DCPRO a na ňom spustí automatizované testy. Systém Jenkins má v sieti CESNET navyše k dispozícii aj skupinu serverov (ilustrované v ľavom hornom rohu obrázku), na ktorých uskutočňuje kompiláciu a verifikáciu firmvéru.

Na každom z množiny serverov obsahujúcich sieťovú kartu DCPRO (stred spodnej polovice obrázku 3.1) je možné spustiť testovacie skripty, ktoré predstavujú jednotlivé testovacie scenáre. Zariadenie DCPRO, ktoré na serveroch figuruje ako sieťové rozhranie je prostredníctvom vstupno-výstupných portov prepojené s portom hardvérového šasi systému STC. Pomocou tohto spojenia budú na zariadenie posielané testovacie pakety predstavujúce podozrivú sieťovú prevádzku. Dôvod behu testov na serveri obsahujúcom zariadenie DCPRO je ten, že počas testovania je nutná komunikácia priamo so zariadením DCPRO, napríklad prostredníctvom CLI nástrojov, pre potreby konfigurácie zariadenia, alebo za účelom vyčítavania zaznamenaných hodnôt. Každý server obsahujúci sieťovú kartu DCPRO je teda prostredníctvom zariadenia DCPRO prepojený so šasi systému STC a ďalej je pomocou bežného sieťového rozhrania prepojený s Jenkins serverom a s Windows serverom, na ktorom




Obr. 3.1: Prostredie pre testovanie zariadenia DCPRO pozostávajúce z Jenkins serveru, serverov na kompiláciu a verifikáciu firmvéru, testovacích serverov obsahujúcich sieťovú kartu DCPRO, Windows serveru obsahujúceho softvérovú časť systému Spirent TestCenter a hardvérového šasi systému Spirent TestCenter.

beží softvérová časť systému STC. Úlohou testovacieho systému po tom, ako sú spustené systémom Jenkins testy, je riadenie systému STC a načítavanie zaznamenaných hodnôt z tohto systému, ako aj konfigurácia a načítavanie hodnôt zo zariadenia DCPRO. Na záver testovací systém agregované výsledky vyhodnotí a odovzdá spolu s riadením systému Jenkins.

Tak, ako je uvedené v teoretickej časti venovanej systému STC (kapitola 2.3), tento systém sa skladá z dvoch častí - hardvérového šasi (pravý spodný roh obrázku 3.1) a softvérovej časti, ktorá je v rámci popisovaného testovacieho prostredia umiestnená na Windows serveri (pravý horný roh obrázku). Softvérová časť poskytuje API, ktoré testovací skript využíva na rezerváciu portov STC, konfiguráciu šasi a vyčítanie zaznamenaných dát zo systému STC. Hardvérové šasi obsahuje niekoľko portov, ktoré sú prepojené s testovanými zariadeniami DCPRO. Skript počas testovania prostredníctvom API poskytovaného softvérom najprv rezervuje port šasi a nahrá potrebnú konfiguráciu. Šasi potom generuje sieťovú prevádzku, ktorú posiela z rezervovaného portu na testované zariadenie a taktiež príma naspäť preposlané pakety od testovaného zariadenia.

## 3.2 Požiadavky na testovací systém a štruktúru testov

Vzhľadom na to, že táto práca sa zameriava na testovanie fyzického sieťového zariadenia. Taktiež vzhľadom na to, že funkčné testovanie tohto zariadenia vyžaduje jeho zapojenie do sieťovej infraštruktúry, v ktorej je sieťová prevádzka generovaná systémom STC. A v neposlednom rade vďaka tomu, že testovanie tohto zariadenia vyžaduje komunikáciu testovacieho systému so samotným zariadením DCPRO a vzhľadom na potreby testovania

vyplývajúce z architektúry zariadenia a jeho funkcionality, musí testovací systém spĺňať nasledujúce požiadavky:

- Testovací systém musí umožňovať spúšťanie kompletnej preddefinovanej sady testov pre celé zariadenie, ktoré bude využívané napríklad v automatizovanom procese integrácie systémom Jenkins, ako aj spúšťanie testov jednotlivých firmvérových či softvérových častí, ktorých samostatné testovanie môže byť využité napríklad pre debugovacie účely.
- Systém musí zabezpečiť komunikáciu so zariadením DCPRO, za účelom konfigurácie zariadenia a jeho komponent, ako aj za účelom vyčítania hodnôt zaznamenaných zariadením. Taktiež je nutné, aby systém umožňoval komunikovať so systémom STC, ktorého softvérová časť beží na vzdialenom Windows serveri.
- Štruktúra testov musí byť rozdelená tak, aby testy, ktoré spolu sémanticky súvisia, alebo sa zameriavajú na testovanie rovnakej funkcionality, či časti zariadenia, boli implementované na rovnakom mieste. Ďalej, aby bolo možné sa v testoch jednoducho orientovať za účelom úprav, alebo pridávania testovacích scenárov. A taktiež, aby bolo možné izolovane spúšťať iba vybrané testy, napríklad za účelom debugovania.
- Priebeh testov musí byť rozdelený do viacerých logických častí, alebo fáz, ktoré sa pri testovaní budú vykonávať v pevne stanovenom poradí. Počet fáz, poradie ich vykonávania a ich samotný obsah musí byť koncipovaný čo najuniverzálnejšie, teda aby bolo možné pridať nový testovací scenár, alebo testovaciu triedu bez nutnosti opakovaného implementovania elementárnych úkonov, ako napríklad pripojenie k STC serveru, alebo spracovanie výstupu z jednotlivých CLI nástrojov zariadenia DCPRO.
- Výstup testovania by mal poskytovať základné informácie o priebehu testovania, použitých testovacích konfiguráciách, jednotlivých krokoch testovania a hlavne výsledky testovania, ktoré v prípade odhalenia chyby budú jednoznačne identifikovať konkrétnu chybu, prípadne poskytovať informáciu o očakávanom správnom výsledku, ktorý v danom prípade nebol dosiahnutý. Forma výstupu by mala byť čo najjednoduchšia a najprehľadnejšia z dôvodu rýchlej identifikácie chyby akýmkoľvek používateľom.

## 3.3 Návrh štruktúry systému a testov

### Návrh štruktúry systému

S ohľadom na požiadavky uvedené v predchádzajúcej sekcii bude štruktúra testovacieho systému rozdelená do viacerých logických celkov. Konkrétne zdrojové súbory budú podľa ich obsahu a významu v rámci systému rozdelené do dvoch kategórií:

- súbory obsahujúce statické konfiguračné dáta
- súbory obsahujúce implementáciu funkcionality

Toto rozdelenie bude uplatnené na globálnej úrovni, teda v rámci koreňovej štruktúry systému, ako aj v rámci implementácie testov, z ktorých každý jeden bude zameraný na testovanie iného, konkrétneho firmvérového modulu, alebo jedného konkrétneho logického celku.

### Návrh štruktúry testov

Za účelom vytvoriť čo najobecnejšiu štruktúru testov, ktorá bude aplikovateľná pri testovaní akéhokoľvek modulu, alebo vlastnosti zariadenia DCPRO, bez ohľadu na konkrétny obsah testu bude priebeh testov rozdelený do nasledujúcich troch fáz:

**Fáza pred začatím testovania** – obsahom tejto fázy je príprava testovacieho prostredia, ako napríklad pripojenie testovacieho systému k systému STC a jeho počiatočná konfigurácia, počiatočná konfigurácia a reboot zariadenia DCPRO atď.

**Fáza testovania** – obsahom tejto fázy je vykonanie jednotlivých testovacích scenárov, a tým overenie požadovanej funkcionality či vlastnosti zariadenia DCPRO. Priebeh každého testovacieho scenára bude pozostávať z troch fáz. Prvá fáza bude slúžiť na konfiguráciu zariadenia DCPRO a systému STC hodnotami odpovedajúcimi konkrétnemu scenáru. Obsahom druhej fázy bude generovanie sieťovej prevádzky, zaznamenanie a vyhodnotenie nameraných hodnôt a výsledku testu. V tretej fáze budú vynulované počítadlá slúžiace na zaznamenávanie nameraných hodnôt.

**Fáza po skončení testovania** – obsahom tejto fázy je uvoľnenie alokovaných zdrojov a ukončenie spojenia medzi testovacím systémom a systémom STC.

## 3.4 Návrh výstupu testov

Vzhľadom na požiadavky uvedené v predchádzajúcej časti tejto kapitoly bude testovací systém generovať výstup o priebehu testov v textovej forme v podobe logov a zároveň vo forme HTML dokumentu. Logovacie záznamy budú obsahovať všetky podrobné informácie z priebehu testovania a vykonaných operáciách, ako informácie o konfigurácii jednotlivých CLI nástrojov, informácie o začatí a ukončení konkrétnych testovacích scenárov, výsledky daných scenárov informácie o očakávaných a reálne nameraných hodnotách, detailné informácie o vyskytnutých chybách, atď.

Za účelom lepšej čitateľnosti a prehľadnosti výsledkov pre akéhokoľvek užívateľa, bude okrem logovacích záznamov systém generovať aj HTML dokument s výsledkami testov. Tento dokument bude obsahovať základné informácie o behu testov, ako napríklad názov serveru, na ktorom boli testy vykonané, čas spustenia testov, dobu trvania testov, počet spustených, úspešných a chybových testov. V prípade detekcie chyby bude vo výstupe uvedená konkrétna chyba, ako napríklad očakávaný počet paketov preposlaných na výstupné rozhranie a reálne nameraný počet paketov. V prípade testov priepustnosti bude dokument obsahovať grafy zobrazujúce namerané hodnoty.

# Kapitola 4

# Implementácia

V tejto kapitole je uvedený stručný popis implementácie testovacieho systému, ktorý je predmetom tejto práce. Konkrétne je v tejto kapitole uvedený zoznam a účel použitia jednotlivých technológií, popis adresárovej štruktúry implementovaného systému. Popis štruktúry testov a priebehu testov, ktorý je rozdelený do troch fáz. Ďalej je popísaná forma a spôsob poskytovania informácií o behu testov a výsledkov testov užívateľovi a spôsob komunikácie testovacieho systém so systémom Spirent TestCenter a so zariadením DCPRO. Na konci kapitoly je v krátkosti popísaný proces integrácie a spúšťania testov v systéme Jenkins implementovaný v rámci tejto práce.

## 4.1 Použité technológie

Implementácia testovacieho systému bola realizovaná v jazyku Python, pričom niektoré sekvencie príkazov, ako napríklad počiatočná konfigurácia firmvérových komponentov pomocou CLI nástrojov, boli implementované vo forme skriptov v jazyku Bash. Automatizované spúšťanie testov a integrácia testov do už existujúceho systému bola realizovaná pomocou syntaktických konštrukcií Jenkins Pipeline a jazyka Groovy. Pre vytváranie konfigurácií systému Spirent TestCenter (ďalej len STC) bolo využívané GUI tohto systému, pre vzdialenú komunikáciu so systémom STC a jeho konfiguráciu počas testovania bola použitá knižnica spirentlib implementovaná v rámci združenia CESNET. Pre ukladanie konfigurácií vytvorených prostredníctvom GUI bol použitý formát súborov XML. Súčasťou výstupu testov je aj HTML dokument zahrňujúci prehľad uskutočnených testov, ich výsledky prípadne grafy s výsledkami testov priepustnosti. Tento dokument obsahuje formátovanie definované jazykom CSS a funkcionalitu implementovanú v jazyku JavaScript.

## 4.2 Implementácia štruktúry systému a testov

Popis implementácie štruktúry systému a testov je kľúčový z hľadiska porozumenia významu jednotlivých častí kódu a jednotlivých zdrojových súborov, a najmä je nutné ho zdokumentovať za účelom zjednodušenia budúceho upravovania či rozširovania testovacieho systému, alebo upravovania už implementovaných a pridávania nových testovacích tried a scenárov.

### Adresárová štruktúra systému

Adresárová štruktúra zobrazená na obrázku 4.1, vyplýva z návrhu uvedeného v predchádzajúcej kapitole v sekcii 3.3 a obsahuje tri podadresáre. Prvý z nich `config` slúži na ukladanie

súborov obsahujúcich konfiguračné dáta, ako napríklad zoznam testovacích tried, ktoré budú spustené skriptom `runtests.py`.

```
dcpro/
├── config/
├── src/
├── tests/
└── runtests.py
```

Obr. 4.1: Adresárová štruktúra testovacieho systému obsahujúca skript na automatizované spúšťanie testov a podadresáre určené na ukladanie konfiguračných súborov, skriptov, tried potrebných pri testovaní a testovacích tried.

Podadresár `src` obsahuje všetky súbory, ktoré sú z hľadiska implementovaného systému potrebné na globálnej úrovni, ako napríklad implementáciu triedy `StcTest`, od ktorej dedia všetky testovacie triedy a ktorá obsahuje implementáciu základnej štruktúry testov a metód využívaných pri testovaní. Podadresár `src` ďalej obsahuje triedy na obsluhu logovania, spracovania vstupných parametrov, generovania HTML dokumentu s výsledkami testovania, skript používaný na bootovanie karty a ďalšie zdrojové súbory. Posledný z trojice podadresárov ilustrovaných na obrázku 4.1 s názvom `tests` obsahuje testovacie triedy, z ktorých každá jedna je zameraná na testovanie iného, konkrétneho firmvérového komponentu, alebo jedného konkrétneho logického celku. Každá testovacia trieda je umiestená v adresári, ktorý nesie rovnaké meno ako daná trieda a ktorý je umiestený v adresári `tests`.

```
tests/
└── maceditor/
    ├── config/
    ├── src/
    └── maceditor.py
```

Obr. 4.2: Štruktúra adresára testovacej triedy `maceditor`, obsahujúca podadresáre `src` a `config` a zdrojový súbor `maceditor.py` obsahujúci implementáciu testovacej triedy.

Ako príklad slúži testovacia trieda `maceditor.py` uložená v adresári `maceditor` vyobrazená na obrázku 4.2. V adresári každej testovacej triedy sa ďalej nachádza podadresár `config`, ktorý obsahuje XML súbor s konfiguráciou pre systém Spirent TestCenter prípadne konfiguračné súbory pre jednotlivé firmérové komponenty využívané na konfiguráciu zariadenia počas priebehu konkrétnych testovacích scenárov. Podadresár `src` obsahuje skripty `setup.sh`, ktoré slúžia na počiatočnú konfiguráciu zariadenia DCPRO a jeho komponent vykonávanú pred spustením konkrétnych testovacích scenárov.

### Štruktúra testov

Štruktúra testov implementovaná podľa návrhu popísaného v sekcii 3.3 je definovaná v triede `StcTest`, od ktorej dedia všetky ostatné testovacie triedy, teda napríklad testovacie triedy, ktorých obsahom je otestovanie kompletnej funkcionality konkrétneho firmvérového modulu. Priebeh testov je rozdelený do troch hlavných fáz, ktoré sú v triede `StcTest` implementované prostredníctvom troch metód.

### Fáza pred začatím testovania

Táto fáza je implementovaná v metóde `_prologue()` a vykonáva sa vždy na začiatku testovania pre každú konkrétnu testovaciu triedu. Jej obsah pozostáva z vytvorenia TCP spojenia medzi testovacím systémom a softvérovou časťou systému STC. Ďalším krokom je načítanie konfigurácie systému STC z predpripraveného XML súboru. Následne sa vykoná pripojene k šasi STC a rezervácia odpovedajúceho portu STC. Potom nasleduje rebootovanie zariadenia DCPRO. Tu je nutné podotknúť, že v tomto kroku je využívaný rýchly reboot zariadenia, ktorý nezahrňuje načítavanie novej konfigurácie, ale iba opätovné spustenie zariadenia s už predtým nahranou konfiguráciou. Nahranie požadovanej konfigurácie je teda potrebné zabezpečiť ešte pred spustením samotných testov. V procese kontinuálnej integrácie vytvorenom v rámci tejto práce je načítavanie požadovanej konfigurácie automatizované prostredníctvom skriptu. Posledným krokom popisovanej fázy je spustenie skriptu zabezpečujúceho počiatočnú konfiguráciu zariadenia DCPRO a jeho komponent, ktorého umiestenie v rámci adresárovej štruktúry bolo popísané vyššie v tejto kapitole.

### Fáza testovania

Je implementovaná v metóde `_testing()` a jej obsahom je otestovanie jednotlivých testovacích scenárov. Každý testovací scenár je v rámci implementovaného systému definovaný hodnotami testovacích dát a operáciami, ktoré sa majú nad definovanými dátami vykonať. Štruktúra a hodnoty testovacích dát a operácie, ktoré majú byť vykonané, sa pre každú testovaciu triedu líšia a vyplývajú z konkrétneho zámeru, za ktorým bola daná testovacia trieda vytvorená. Implementácia týchto operácií a hodnoty testovacích dát sú teda umiestnené v každej testovacej triede, ktorá je potomkom triedy `StcTest`.

Testovacie dáta reprezentujúce konkrétne testovacie scenáre obvykle obsahujú názov, respektíve krátky popis testovacieho scenáru, zoznam názvov StreamBlockov (tabuľka 2.2) použitých na emuláciu požadovanej sieťovej prevádzky, príkazy na konfiguráciu komponentov zariadenia DCPRO, názvy konfiguračných súborov určených na konfiguráciu zariadenia DCPRO a očakávané hodnoty nameraných veličín, ktoré budú použité pri vyhodnocovaní výsledku daného scenára. Zjednodušením príkladom testovacieho objektu pozostávajúceho z testovacích dát môže byť objekt používaný na overenie funkcionality TTL editoru. Takýto zjednodušený objekt by obsahoval štyri atribúty:

- popis scenáru: `TTL editor aktívny, IPv4 pakety s TTL hodnotou 1`
- názov StreamBlocku, ktorý bude použitý: `ipv4-ttl-1`
- príznak, či bude TTL editor aktivovaný, alebo nie: `ENABLE`
- očakávaný počet paketov preposlaných na výstupné rozhranie: `1000000`

Nad každým jedným objektom testovacích dát, ktorý reprezentuje jeden testovací scenár, sa postupne v uvedenom poradí vykonajú definované operácie, ktoré určujú priebeh daného testovacieho scenára. Operácie sú logicky rozdelené do troch fáz vyplývajúcich z návrhu:

- Prvá fáza je imlementovaná v metóde `_pre_test()` a jej účelom je konfigurácia zariadenia DCPRO a jeho komponentov, konfigurácia systému STC (napr. nastavenie vlastností emulovanej sieťovej prevádzky) prípadne spustenie softvérových nástrojov

zariadenia DCPRO (napr. za účelom zaznamenávania paketov preposlaných do softvérovej časti zariadenia).

- Počas druhej fázy implementovanej v metóde `_test()` sa obvykle iniciuje spustenie a následne zastavenie generovanie sieťovej prevádzky systémom STC, načítanie nameraných hodnôt a hodnôt odpovedajúcich vygenerovanej sieťovej prevádzke zo systému STC, ako aj zo zariadenia DCPRO. Nakoniec sa namerané hodnoty porovnávajú s očakávanými hodnotami, alebo hodnotami vygenerovanej sieťovej prevádzky. Na základe porovnania týchto hodnôt je test vyhodnotený ako úspešný, alebo v prípade chyby je do výsledkov zaznamenaná chyba vo forme popisu danej chyby spolu s očakávanými a nameranými hodnotami.
- Obsahom tretej fázy implementovanej v metóde `_post_test()` je vynulovanie počítadiel systému STC a počítadiel zariadenia DCPRO, prípadne zastavenie softvérových nástrojov zariadenia DCPRO, ak tak nebolo uskutočnené počas vyčítavania zaznamenaných hodnôt.

#### Fáza po skončení testovania

Táto je implementovaná v metóde `_epilogue()` a vykonáva sa po skončení testovania. V tejto fáze sa ukončuje spojenie so systémom STC zahrňujúce uvoľnenie rezervovaných portov.

## 4.3 Výstup priebehu testov a testovacích výsledkov

Testovací systém počas testovania poskytuje detailné informácie o priebehu testu na štandardný výstup a taktiež tieto informácie zaznamenáva do logovacieho súboru. Na tieto účely využíva systém knižnicu s názvom logging. Ak sú testy spúšťané pomocou testovacieho skriptu `runtest.py`, systém generuje okrem logov aj HTML dokument. Generovanie HTML dokumentu bolo implementované pomocou knižnice yattag, formátovanie dokumentu bolo prevzaté z nástroja HtmlTestRunner [28] a následne prispôsobené potrebám implementovaného systému.

## 4.4 Komunikácia so systémom Spirent TestCenter

Ako je uvedené v sekcii popisujúcej testovacie prostredie zariadenia DCPRO 3.1, softvérová časť systému STC beží na rozdiel od testovacieho systému na separátnom Windows servery. Implementovaný systém teda využíva na komunikáciu so systémom STC knižnicu spirentlib. Táto knižnica poskytuje identické rozhranie pre ovládanie systému STC, ako rozhranie softvérového balíčka Spirent TestCenter Automation popísaného v sekcii 2.3 s tým rozdielom, že systém možno ovládať z iného počítača, ako ten, na ktorom beží softvérová časť systému STC. Knižnica spirentlib vytvára TCP spojenie medzi serverovou časťou, ktorá beží na windows servery spolu so softvérom systému STC a klientskou časťou, ktorá je implementovaným systémom počas testovania využívaná na komunikáciu so systémom STC.

## 4.5 Komunikácia so zariadením DCPRO

Komunikácia so zariadením DCPRO je realizovaná pomocou CLI nástrojov popísaných v sekcii 2.2. Spúšťanie CLI nástrojov a následné načítanie štandardného výstupu a štandardného chybového výstupu je realizované knižnicou subprocess. Na extrahovanie potrebných hodnôt z výstupu niektorých CLI nástrojov sú implementované potrebné metódy, ktoré výstup spracovávajú za pomoci regulárnych výrazov a knižnice re.

V prípade nástroja `ndp-tool`, ktorý je pri testovaní využívaný v móde read za účelom zaznamenávania počtu paketov preposlaných do softvérovej časti zariadenia DCPRO a v móde generate za účelom generovania sieťovej prevádzky systémom DCPRO, bolo nutné implementovať paralelný beh tohto nástroja a testovacieho skriptu. To je dosiahnuté taktiež použitím knižnice subrocess, pomocou ktorej je spustený samostatný proces, ktorého identifikátor PID si skript ukladá a v prípade potreby toto PID použije pre prístup k procesu.

## 4.6 Integrácia testov do systému Jenkins

V rámci tejto práce bolo nutné prostredníctvom Jenkins Pipeline implementovať proces automatizovaného spúšťania testov systémom Jenkins, za účelom následného pridania tohto procesu do už existujúceho procesu využívaného pri vývoji zariadenia DCPRO. Vzhľadom na to, že existuje viac typov zariadenia DCPRO a softvér pre každý z týchto typov je implementovaný v samostatnej vetve, implementovaná Pipeline obsahuje jeden vstupný parameter, a to typ zariadenia DCPRO. Pipeline na základe predaného typu zariadenia najprv overí či v rámci testovacieho prostredia existuje server obsahujúci daný typ zariadenia, následne či je daný server prostredníctvom zariadenia DCPRO prepojený s nejakým portom šasi STC. Toto overovanie sa uskutočňuje na základe aktuálneho popisu fyzického zapojenia serverov v rámci testovacieho prostredia, umiestneného v zdrojovom kóde Pipeline. Ak existuje jeden, alebo viac takýchto serverov, Pipeline sa pokúsi rezervovať niektorý zo serverov.

Po úspešnej rezervácii sa vykonávanie operácií presunie na rezervovaný server, kde Pipeline naklonuje potrebné repozitáre. Ďalej sa vytvorí adresár s číslom prebiehajúceho buildu, do ktorého budú umiestnené výstupy testovania, ako súbor s logmi atď. Následne sa nainštalujú potrebné softvérové balíčky zariadenia DCPRO a knižnice jazyka Python využívané pri testovaní.

Ďalším krokom je nabootovanie odpovedajúceho typu zariadenia DCPRO pomocou skriptu `bootcard.py`. Po úspešnom nabootovaní zariadenia je uskutočnené samotné testovanie prostredníctvom spustenia skriptu `runtests.py`. Archivovanie testovacích artefaktov, ktorými sú výstupy testovania nasleduje po ukončení skriptu `runtests.py`. Na záver sa systém Jenkins odhlási z rezervovaného serveru a odstráni jeho rezerváciu z rezervačného systému.

# Kapitola 5

# Implementované testy

V tejto kapitole sa nachádza popis testov implementovaných v rámci tejto práce. Je tu uvedený popis testov firmvérových modulov, testov priepustnosti a testov softvérových modulov na ochranu pred SYN Flood a amplifikačnými útokmi. U každého z uvedených typov testov je stručne popísaný proces testovania, spôsob použitia systému STC a zoznam a spôsob vyhodnocovania vlastností sieťovej prevádzky, ktoré sa u daných testov zaznamenávajú.

## 5.1 Testy firmvérových modulov

U väčšiny testovacích tried zameraných na overovanie správnej funkčnosti jednotlivých firmvérových modulov je postup testovania veľmi podobný. Konkrétne testovacie triedy sa líšia hlavne nastavením firmvérových modulov zariadenia DCPRO, či parametrami sieťovej prevádzky generovanej systémom STC (napríklad hodnoty zdrojových a cieľových IP adries, portov, MAC adries, hodnoty VLAN ID, použité protokoly atď.) Testovacie triedy sa ďalej môžu odlišovať zaznamenávanými vlastnosťami sieťovej prevádzky spätne preposlanej do systému STC (napr. hodnoty zdrojových a cieľových IP adries, MAC adries, hodnoty VLAN ID) a parametrami využívanými pri overovaní správnej funkčnosti daného modulu (počet paketov preposlaných do softvérovej časti zariadenia DCPRO, počet paketov vygenerovaných systémom STC a preposlaných na výstupné rozhranie zariadenia DCPRO a tým spätne do systému STC).

Pred začatím testovania špecifického modulu je uskutočnená konfigurácia ostatných firmvérových modulov zariadenia tak, aby tieto moduly nemali vplyv na spracovávanú sieťovú prevádzku, a teda aby bolo možné sledovať izolované správanie práve testovaného modulu. Následne sa začnú vykonávať jednotlivé testovacie prípady. Väčšinou je prvý testovací prípad zameraný na overenie predpokladaného predvoleného správania daného modulu po nabootovaní zariadenia DCPRO, teda bez ďalšej explicitnej konfigurácie testovaného modulu. Následne sa vykonávajú ďalšie testovacie prípady, ktoré už zahrňujú konfiguráciu testovaného modulu. Na záver každého testovacieho prípadu sa porovnávajú očakávané a reálne namerané hodnoty a to, či boli všetky pakety prijaté na sieťové rozhranie zariadenia spracované.

Jednotlivé testovacie prípady pre konkrétny modul (teda v rámci jednej testovacej triedy) sa od seba odlišujú nastavením daného modulu, hodnotami parametrov generovanej sieťovej prevádzky a konkrétnymi hodnotami parametrov využívaných pri overovaní správnej funkčnosti modulu. Každá testovacia trieda má väčšinou preddefinovaný istý počet rozdielnych StreamBlockov (tabuľka 2.2), výsledkom použitia ktorých sú dosiahnuté

požadované parametre generovanej sieťovej prevádzky. Pri jednotlivých testovacích prípadoch sa teda používajú rozličné kombinácie StreamBlockov. Spôsob konfigurácie firmvérových modulov závisí od špecifických vlastností daného modulu. Niektoré moduly umožňujú napríklad iba aktiváciu a deaktiváciu filtrovania, alebo upravovania špecifickej položky v spracovávanom pakete, iné moduly umožňujú konfiguráciu prostredníctvom väčšieho počtu komplexných pravidiel, ktoré sú popísané bezkontextovou gramatikou. U modulov, ktorých konfigurácia je popísaná gramatikou, je spravidla vytvorených viac testovacích prípadov a to z toho dôvodu, aby boli otestované všetky možné syntaktické konštrukcie od tých najjednoduchších, pozostávajúcich z jedného pravidla zameraného vždy na správne vyhodnocovanie určitej jednej vlastnosti spracovávaného paketu, až po komplexné konfigurácie zložené z viacerých pravidiel.

Stručný popis testov jednotlivých modulov:

- Test modulu MAC editor sa zameriava na testovanie toho, či modul dokáže správne prepisovať zdrojovú alebo cieľovú MAC adresu v prijatých paketoch, alebo či dokáže tieto adresy vzájomne vymieňať. Teda počas testovania sú kontrolované správne hodnoty MAC adries u paketov preposlaných do STC.

- Test modulu s názvom odbočovací filter (OB filter) overuje, či na základe konkrétnej konfigurácie bol do softvéru zariadenia poslaný správny počet paketov a zároveň, či bol správny počet paketov preposlaný prostredníctvom výstupného sieťového rozhrania zariadenia DCPRO do cieľovej siete.

- U prefixového filtru sa taktiež overuje počet paketov preposlaných do softvéru zariadenia a na výstupné rozhranie, a u paketov preposlaných na výstupné rozhranie sa ďalej overuje hodnota cieľovej MAC adresy, ktorú je možne pomocou daného modulu modifikovať.

- Blokovací filter, ktorý je tvorený dvomi modulmi štatistickým IP filtrom a štatistickou tabuľkou pravidiel je testovaný tak, že na základe konfigurácie uvedených dvoch modulov sú porovnávané počty preposlaných paketov a zároveň počty paketov zaznamenaných modulom s názvom štatistická jednotka.

- Modul pre kontrolu TTL je testovaný tak, že na základe hodnoty TTL u paketov poslaných na rozhranie zariadenia DCPRO overuje, či bola hodnota TTL správne zmenšená o jedna, prípadne či boli pakety z hodnotou TTL jedna zahodené.

- Test modulu VLAN editor zahrňuje nahranie mapovania jednotlivých VLAN ID z konfiguračného súboru a následne overovanie správnej modifikácie hodnôt VLAN ID u preposlaných paketov.

- Modul watchdog je testovaný tak, že jeho interný čítač je nastavený na určitú hodnotu v sekundách a následne je nakonfigurovaná jedna z troch možných akcií, ktorú dokáže daný modul vykonávať. Ďalej je na zaradenie poslaná sieťová prevádzka pozostávajúca zo špecifikovaného počtu paketov a následne sú vyčítané počty paketov preposlaných do softvéru zariadenia a na výstupné rozhranie. Potom systém čaká na dosiahnutie maximálnej hodnoty čítača modulu a následne je na zariadenie opäť poslaná sieťová prevádzka pozostávajúca zo špecifikovaného počtu paketov a sú vyčítané počty paketov preposlaných do softvéru zariadenia a na výstupné rozhranie. Zo zaznamenaných hodnôt je vyhodnotené, či modul správne preposielal pakety pred naplnením čítača a či po naplnení čítača uskutočnil nakonfigurovanú akciu.

## 5.2 Testy priepustnosti

Testy priepustnosti zariadenia DCPRO realizujú meranie priepustnosti v dvoch smeroch, a to priepustnosť v smere prijímania paketov RX a priepustnosť v smere odosielania paketov TX. RX priepustnosť sa meria tak, že na vstupné rozhranie zariadenia DCPRO je zo systému STC posielaná sieťová prevádzka rýchlosťou 100 Gb/s. Keďže pomocou nástroja `nfb-eth` je možné vyčítať koľko paketov bolo prijatých na vstupné rozhranie zariadenia DCPRO a koľko paketov zariadenie reálne spracovalo, RX priepustnosť je jednoducho vypočítateľná ako podiel týchto dvoch hodnôt. TX priepustnosť sa meria tak, že sú určitý čas (10s) pakety posielané na šasi STC. Po uplynutí daného času je zo systému STC vyčítaný počet prijatých bitov a výsledná TX priepustnosť je vypočítaná, ako podiel počtu prijatých bitov a času generovania.

Bez ohľadu na to, či je konkrétna testovacia trieda zameraná na testovanie priepustnosti RX, TX, alebo oboch naraz sa priepustnosť meria vždy pre dĺžky rámcov v rozsahu od 64 B do 1526 B. Meranie však z časových dôvodov nie je vykonávané pre každú dĺžku v danom rozsahu, teda od veľkosti 64 po veľkosť 1526 s krokom 1, ale iba pre niektoré hodnoty z daného rozsahu zvolené s určitým krokom. Systém namerané hodnoty priepustnosti vykresľuje vo forme grafu, a to v jednotkách gigabitov za sekundu a v počte paketov za sekundu. Každý testovací prípad má stanovenú minimálnu možnú priepustnosť v percentách, ktorá ak počas testovania nie je aspoň pri jednej dĺžke rámca dosiahnutá, daný testovací prípad je vyhodnotený ako neúspešný. Jednotlivé testovacie prípady sa od seba odlišujú nastavením firmvérových modulov a spustením nástrojov `ndp-read` a `ndp-generate`, od čoho sa odvíjajú aj stanovené minimálne limity priepustnosti. V rámci tejto práce boli implementované tri testovacie triedy zamerané na testovanie nasledujúcich priepustností RX priepustnosť, TX priepustnosť a RX súčasne s TX priepustnosťou.

## 5.3 Testy softvérových modulov

Pri testovaní softvérových modulov na ochranu pred SYN Flood a amplifikačnými útokmi musí vytvorená konfigurácia pre STC obsahovať emulovaného sieťového hosťa, ktorý má ako default gateway nastavenú IP adresu, ktorá bude nastavená ako IP adresa sieťového rozhrania zariadenia DCPRO. Testovanie potom prebieha tak, že sa najprv do konfiguračných súborov jednotlivých modulov nahrajú pravidlá popisujúce podozrivú sieťovú prevádzku. Potom je spustený softvérový daemon a ďalej sa nastaví IP adresa (totožná s tou, ktorú má nastavený emulovaný hosť ako default gateway) na sieťové rozhranie zariadenia DCPRO a rozhranie sa uvedie do stavu `UP`. Potom sa adresa emulovaného hosťa nastaví ako next hop adresa pre všetky siete, do ktorých bude cielená sieťová prevádzka zo STC. To bude mať za následok to, že keď zariadenie DCPRO príjme prevádzku zo STC bude ju preposielať naspať na STC, kde bude možné ju analyzovať. Nasledujúci priebeh testovania je obdobný ako u testov firmvérových modulov. Podľa preddefinovaných testovacích dát sa aktivujú požadované StreamBlocky a spustí sa generovanie prevádzky. Generovanie je spustené niekoľko sekúnd a po jeho ukončení nasleduje načítanie nameraných hodnôt a vyhodnotenie úspešnosti testu. V implementovaných scenároch sa generuje rovnaká sieťová prevádzka smerujúca do dvoch rozdielnych cieľových sietí. Pravidlá sú nastavené tak, aby prevádzka smerovaná do jednej zo sietí bola vyhodnotená ako podozrivá a prevádzka smerovaná do druhej siete ako legitímna. Pakety preposlané späť na STC sa potom spočítavajú do skupín na základe cieľovej IP adresy. Pomocou zaznamenaných počtov paketov je potom overené, či bola podozrivá prevádzka vyfiltrovaná a legitímna prevádzka preposlaná ďalej.

Okrem testov modulov na ochranu pred SYN Flood a amplifikačnými útokmi bol implementovaný aj jednoduchý test na overenie toho, či po spustení softvérového daemona zariadenie dokáže správne smerovať pakety. Tento test prebieha tak, že po spustení daemona sa na sieťové rozhranie zariadenia DCPRO nastaví IP adresa a rozhranie sa uvedie do stavu `UP`. Následne sa na servery, ktorý obsahuje zariadenie DCPRO, spustí nástroj `ping` [10], ktorého úlohou je overiť sieťové spojenie s hosťom emulovaným na STC (hosť musí mať ako default gateway nastavenú IP adresu rozhrania). Keďže tento server je so STC prepojený iba pomocou zariadenia DCPRO, pakety generované nástrojom `ping` budú posielané cez toto zariadenie. Ak teda overenie dostupnosti emulovaného hosťa nástrojom `ping` dopade úspešne, test je vyhodnotený ako úspešný, nakoľko zariadenie dokáže správne smerovať pakety.

# Kapitola 6

# Dosiahnuté výsledky

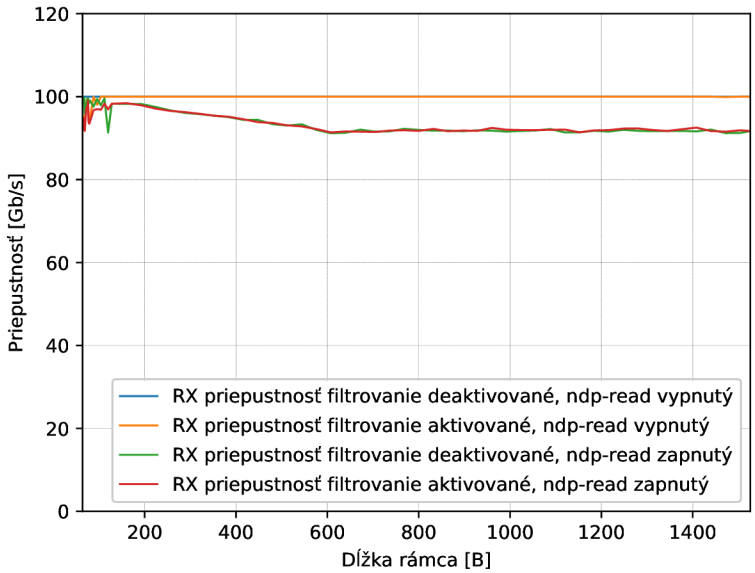
Táto kapitola obsahuje prehľad testovacích tried a počet testovacích prípadov implementovaných v rámci tejto práce. Ďalej sa v tejto kapitole nachádzajú výsledky testov priepustnosti vo forme grafov a na záver sú zhrnuté možnosti rozšíriteľnosti implementovaného systému a návod ako pri rozširovaní postupovať.

## 6.1 Prehľad implementovaných testov

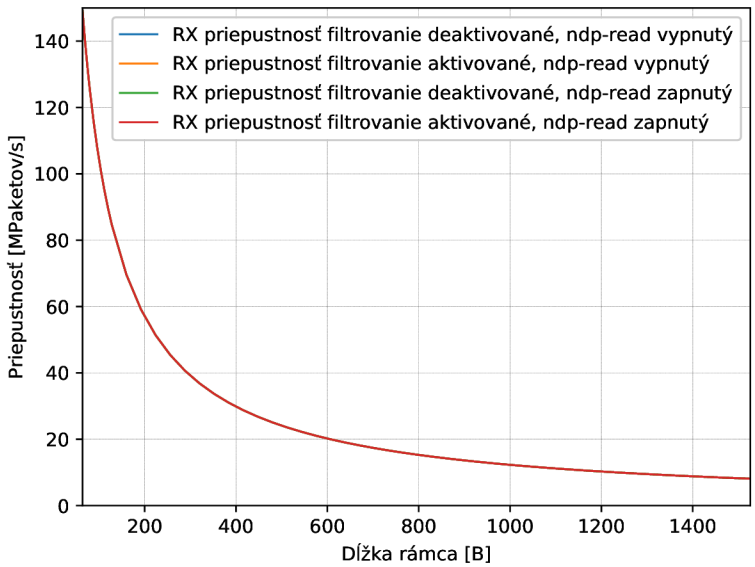
V rámci tejto práce bola okrem samotného testovacieho systému implementovaná aj kolekcia testov overujúcich funkčné a výkonnostné parametre zariadenia DCPRO. Konkrétne bolo implementovaných sedem testovacích tried zameraných na funkčné testovanie jednotlivých firmvérových modulov, ktoré zahrňujú 109 špecifických testovacích scenárov. Ďalej boli implementované tri testovacie triedy zamerané na výkonnostné testovanie zariadenia, ktoré obsahujú 7 testovacích prípadov zameraných na overenie priepustnosti zariadenia DCPRO. Za účelom overenia správnej funkcie smerovania sieťovej prevádzky zariadením DCPRO bol implementovaný jeden test zameraný na túto funkcionalitu. Okrem toho boli implementované dve testovacie triedy zamerané na overenie funkčnosti softvérových modulov na ochranu pred SYN Flood a amplifikačnými útokmi, ktoré obsahujú 10 testovacích prípadov.

## 6.2 Výsledky testov priepustnosti

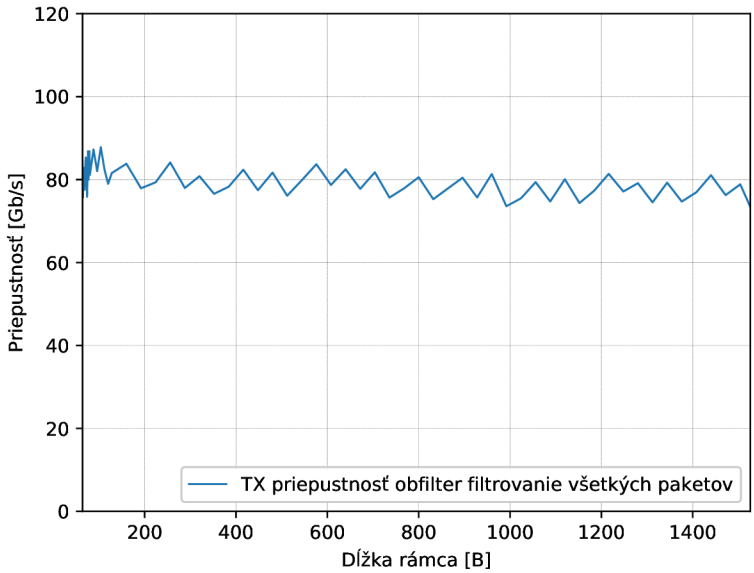
Proces testovania priepustnosti zariadenia DCPRO implementovaný v rámci tejto práce je detailne popísaný v predchádzajúcej kapitole v sekcii 5.2. V tejto kapitole sa nachádzajú namerané výsledky jednotlivých testovacích prípadov zameraných na testovanie priepustnosti v smere RX (grafy 6.1 a 6.2), TX (grafy 6.3 a 6.4) a RX súčasne s TX (grafy 6.5 a 6.6). Meranie ktorého výsledky sú uvedené nižšie, bolo uskutočnené na servery amarone, ktorý sa nachádza v sieti CESNET a je prepojený s generátorom sieťovej prevádzky STC prostredníctvom zariadenia DCPRO typu 100G2Q-LR4 a linkou s prenosovou kapacitou 100 Gb/s.



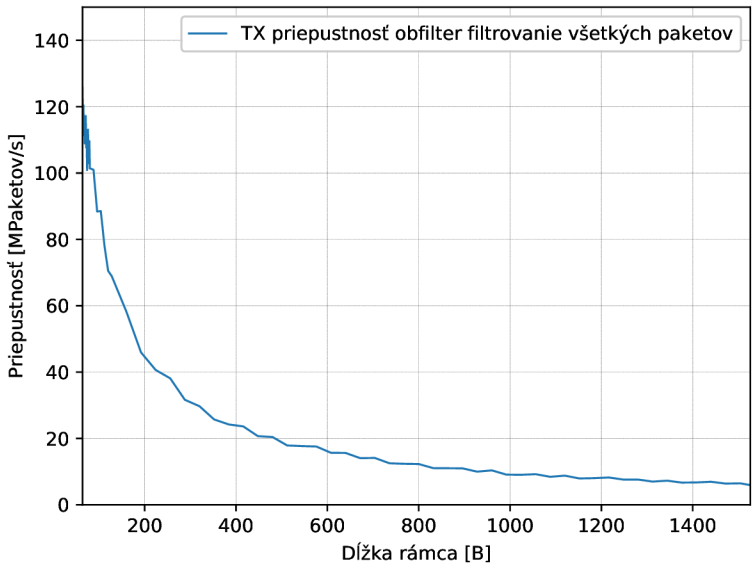

Obr. 6.1: Namerané hodnoty priepustnosti RX v gigabitoch za sekundu pre rôzne dĺžky rámcov s rôznym nastavením filtrovania a použitím nástroja `ndp-read`. Graf zobrazuje akou rýchlosťou dokáže zariadenie prijímať pakety, prijaté na vstupné rozhranie a aký vplyv má na túto rýchlosť nastavenie filtrovania a nástroj `ndp-read`.



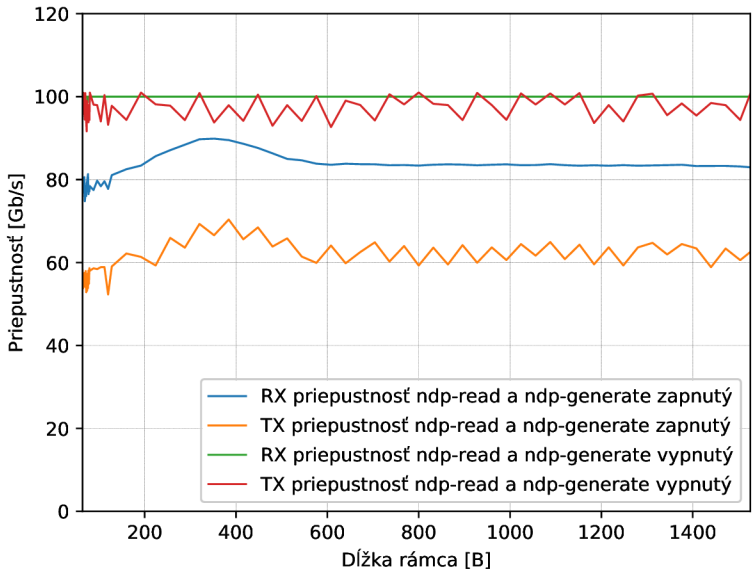

Obr. 6.2: Namerané hodnoty priepustnosti RX v miliónoch paketov za sekundu pre rôzne dĺžky rámcov s rôznym nastavením filtrovania a použitím nástroja `ndp-read`. Graf zobrazuje akou rýchlosťou dokáže zariadenie prijímať pakety, prijaté na vstupné rozhranie a aký vplyv má na túto rýchlosť nastavenie filtrovania a nástroj `ndp-read`.



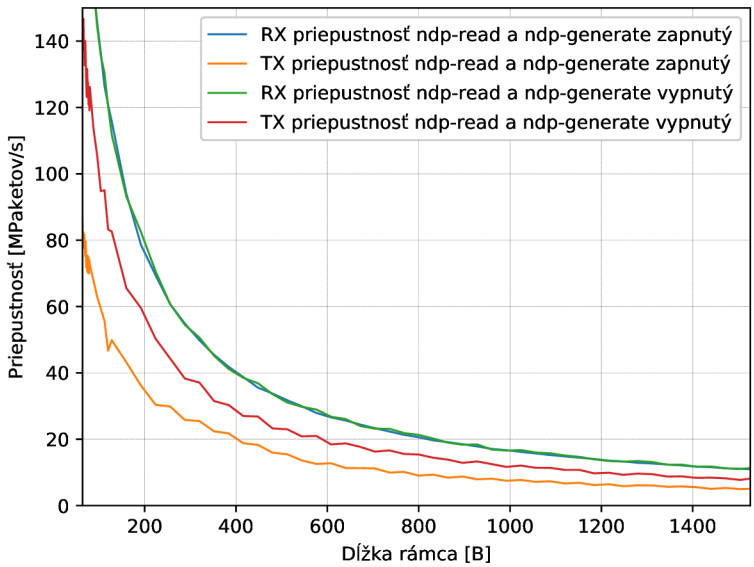

Obr. 6.3: Priepustnosť v smere TX v gigabitoch za sekundu pre rôzne dĺžky rámcov, pri nastavení modulu OB filter tak, aby boli všetky prijaté pakety zahodené a pakety posielané na výstup boli generované nástrojom `ndp-generate`. Graf zobrazuje akou rýchlosťou dokáže zaradenie generovať a preposielať pakety na výstupné rozhranie.




Obr. 6.4: Priepustnosť v smere TX v miliónoch paketov za sekundu pre rôzne dĺžky rámcov, pri nastavení modulu OB filter tak, aby boli všetky prijaté pakety zahodené a pakety posielané na výstup boli generované nástrojom `ndp-generate`. Graf zobrazuje akou rýchlosťou dokáže zaradenie generovať a preposielať pakety na výstupné rozhranie.




Obr. 6.5: Namerané hodnoty priepustnosti v smere RX a súčasne v smere TX v gigabitoch za sekundu pre rôzne dĺžky rámcov, pri spustených a vypnutých nástrojoch `ndp-read` a `ndp-generate`. Graf zaznamenáva, akou rýchlosťou dokáže zariadenie prijímať a zároveň preposielať pakety a aký vplyv na to majú nástroje `ndp-read` a `ndp-generate`.




Obr. 6.6: Namerané hodnoty priepustnosti v smere RX a súčasne v smere TX v miliónoch paketov za sekundu pre rôzne dĺžky rámcov, pri spustených a vypnutých nástrojoch `ndp-read` a `ndp-generate`. Graf zaznamenáva, akou rýchlosťou dokáže zariadenie prijímať a zároveň preposielať pakety a aký vplyv na to majú nástroje `ndp-read` a `ndp-generate`.

## 6.3 Rozšíriteľnosť systému

Cieľom práce bolo vytvoriť systém, ktorý bude jednoducho rozšíriteľný, či už sa jedná o rozšírenie pridaním nových testovacích prípadov v rámci už implementovaných testovacích tried, alebo rozšírenie prostredníctvom vytvorenia úplne nových testovacích tried zameraných napríklad na testovanie novo vytvoreného softvérového, alebo firmvérového modulu. Vo všetkých možných prípadoch budúceho rozširovania systému, je nutné aby bol programátor oboznámený s návrhom systému (kapitola 3), implementáciou systému (kapitola 4) a spôsobom implementácie už existujúcich testov (kapitola 5). Pre zjednodušenie budúceho rozširovania systému je vhodné v tejto kapitole uviesť stručný popis ako postupovať pri rozširovaní systému.

### Postup pri vytváraní nového testovacieho prípadu

Pri vytváraní nového testovacieho prípadu je postup veľmi jednoduchý a skladá sa z nasledujúcich krokov:

- Oboznámenie sa s implementáciou testovacej triedy, do ktorej bude nový prípad pridaný.
- Oboznámenia sa s atribútmi triedy `TestData` (vnorená trieda testovacej triedy), ktorej inštancie predstavujú konkrétne testovacie prípady.
- Oboznámenie sa s konfiguráciou systému STC používanou pre danú testovaciu triedu, prostredníctvom GUI systému STC. V prípade potreby rozšírenie existujúcej konfigurácie vytvorením nových StreamBlockov (tabuľka 2.2) s požadovanými vlastnosťami.
- Pridanie nového objektu testovacích dát triedy `TestData`, ktorý bude reprezentovať novovytvorený testovací prípad do metódy `_set_data()`.
- Vloženie hodnôt do atribútov vytvoreného objektu testovacích dát, ktoré budú definovať testovací prípad, ako napríklad názov testovacieho prípadu, názvy StreamBlockov, ktoré budú použité pre generovanie prevádzky, očakávané hodnoty, ktoré budú slúžiť na overenie úspešnosti daného prípadu, názvy konfiguračných súborov pre jednotlivé moduly, atď.

### Postup pri vytváraní novej testovacej triedy

Pre zjednodušenie vytvárania nových testovacích tried bola vytvorená šablóna testovacej triedy, ktorá sa nachádza v podadresári `src`, koreňového adresára systému (obrázok 4.1) v súbore s názvom `test_class_template.py`. Postup vytvárania novej triedy je nasledovný:

- Vytvorenie podadresára v adresári `tests`, ktorého názov sa bude zhodovať s názvom zdrojového súboru novovytvorenej testovacej triedy, ktorý bude umiestený v tomto adresári. Napríklad, ak bude trieda testovať modulx, vytvoríme adresár s názvom `modulx` a v ňom súbor `modulx.py`. Ďalej vo vzniknutom adresári vytvoríme podadresáre `src` a `config`. Výsledná štruktúra musí odpovedať štruktúre uvedenej na obrázku 4.2.
- Pre novovytvorenú triedu vytvoríme prostredníctvom GUI systému STC konfiguráciu systému STC. Obsah konfigurácie bude odpovedať testom, ktoré chceme v rámci

vytváranej triedy vykonávať. Vzniknutú konfiguráciu uložíme vo formáte XML do podadresára `config` a názov konfiguračného súbor pre systém uložíme v testovacej triede do premennej `SPIRENT_INPUT_FILE`.

- V podadresári `src` vytvoríme súbor `setup.sh`. Tento súbor je určený na vloženie príkazov, ktoré sa majú vykonať vo fáze pred testovaním a ktoré sú špecifické iba pre túto testovaciu triedu. Napríklad pri testovaní špecifických firmvérových modulov sa v tomto súbore nachádzajú príkazy na deaktiváciu ostatných modulov tak, aby nemali vplyv na spracovávanú prevádzku. V prípade že nie je nutná žiadna dodatočná konfiguráciu súbor môže ostať prázdny.

- Posledným krokom je implementácia obsahu testovacej triedy a vytvorenie testovacích dát reprezentujúcich jednotlivé testovacie scenáre. Po vykonaní predchádzajúcich bodov postupu, testovacia trieda obsahuje mierne upravený zdrojový kód prevzatý zo šablóny. Tento kód obsahuje vzor testovacích dát a metódy, ktoré majú byť v triede implementované. U každej takejto metódy je v šablóne uvedený popis toho, aké operácie v nej majú byť vykonané a je v nej uvedený príklad implementácie vybranej operácie.

# Kapitola 7

# Záver

Cieľom tejto práce bolo navrhnúť a implementovať rozšíriteľný systém na automatizované testovanie zariadenia pre ochranu pred (D)DoS útokmi DCPRO, ktoré je vyvíjané v rámci združenia CESNET. Ďalšími požiadavkami bolo vytvorenie kolekcie testov overujúcich funkčné a výkonnostné parametre zariadenia DCPRO a taktiež integrácia vytvoreného testovacieho systému do systému kontinuálnej integrácie Jenkins.

Počas riešenia tejto práce som sa vo fáze teoretického rozboru najprv oboznámil s najčastejšími typmi (D)DoS útokov a princípmi ich fungovania. Ďalej som sa zaoberal zariadením pre ochranu pred (D)DoS útokmi DCPRO, vyvíjaným v rámci združenia CESNET. Konkrétne som sa oboznámil s architektúrou tohto zariadenia a spôsobom jeho nasadenia a fungovania v sieti. Ďalej som sa zameral na firmvérovú a softvérovú časť tohto zariadenia. Zozbieral som detailné informácie o princípe a význame jednotlivých firmvérových a softvérových modulov a základné informácie o dostupných softvérových nástrojoch, ktoré sú súčasťou zariadenia DCPRO. Vo fáze teoretického rozboru som si ďalej naštudoval informácie o generátore sieťovej prevádzky Spirent TestCenter a podrobnosti o fungovaní a možnostiach použitia systému kontinuálnej integrácie Jenkins. V závere teoretického rozboru som analyzoval možnosť použitia najpoužívanejších dostupných testovacích frameworkov pre jazyk Python v rámci tejto práce.

Po získaní uvedených teoretických znalostí som vo fáze návrhu popísal testovacie prostredie, ktoré je dostupné v rámci združenia CESNET. Ďalej som zhrnul požiadavky na testovací systém a štruktúru testov, ktoré vyplývali z charakteristiky zariadenia DCPRO a z možností testovacieho prostredia. V poslednej časti návrhu systému som s ohľadom na získané teoretické znalosti a zaznamenané požiadavky vytvoril návrh štruktúru systému a testov a návrh výstupu testov.

Podľa vytvoreného návrhu som následne realizoval implementáciu testovacieho systému a kolekcie testov. Dôležité informácie o implementácii som zaznamenal do technickej správy. Vzhľadom na to, že jednou z požiadaviek na systém bola rozšíriteľnosť, som do zdrojových súborov systému pridal šablónu pre vytváranie nových testovacích tried. Účelom šablóny je uľahčenie práce pri budúcom rozširovaní systému. Okrem šablóny som do technickej správy uviedol návod ako postupovať pri budúcom rozširovaní.

Spolu s testovacím systémom bola vytvorená aj kolekcia testov overujúcich funkčné a výkonnostné parametre zariadenia DCPRO. Celkovo bolo vytvorených deväť testovacích tried zameraných na testovanie firmvérovných a softvérových modulov zahrňujúcich spolu 119 špecifických testovacích scenárov. Ďalej bol vytvorený jeden test na overovanie správnej schopnosti zariadenia DCPRO smerovať pakety. Okrem toho boli vytvorené tri triedy zamerané na testovanie priepustnosti systému obsahujúce 7 rozdielnych testovacích prípadov.

Testovací systém bol už počas neskoršej fázy vývoja integrovaný do systému Jenkins, čo umožnilo automatizované spúšťanie testov nad najnovšími verziami softvéru zariadenia DCPRO. Vďaka tomu testovací systém už počas fázy vývoja dokázal odhaliť chyby zariadenia DCPRO, ako napríklad chybu pri bootovaní nástrojom `nfb-boot`, nemožnosť nahrávať garantovaný počet pravidiel do blokovacieho filtru, chybne fungujúce načítavanie pravidiel do prefixového filtru a iné chyby.

Možným pokračovaním práce je väčšie začlenenie vytvorených testov do procesu vývoja zariadenia DCPRO v rámci systému Jenkins. Kde sa existujúci proces, ktorý zahrňuje kompiláciu firmvéru a vytvorenie inštalačných balíčkov rozšíri o fázu testovania implementovanú v rámci tejto práce. V budúcnosti bude potom nutné rozširovanie testov napríklad za účelom otestovania novovytvorenej funkcionality zariadenia. Prípadne bude nutná ďalšia adaptácia existujúcich testov, napríklad v prípade uskutočnenia určitej zmeny vo funkcionalite zariadenia.

# Literatúra


[1] Beal, V.: *The 7 Layers of the OSI Model*. [Online; navštívené 26.11.2018].
URL https://www.webopedia.com/quick_ref/OSI_Layers.asp

[2] Berners-Lee, T.; Connolly, D.: Hypertext Markup Language - 2.0. RFC 1866, November 1995.
URL https://tools.ietf.org/html/rfc1866

[3] Case, J.; Fedor, M.: A Simple Network Management Protocol (SNMP). RFC 1157, Máj 1990.
URL https://tools.ietf.org/html/rfc1157

[4] CESNET: *DDoS Protector User Guide*. [Online; navštívené 8.4.2019].
URL https://jenkins.liberouter.org/job/DCPRO/job/DOCS/job/master/lastSuccessfulBuild/artifact/docs/userguide-2.1.12-1+20190314125841+451dc10.pdf

[5] CESNET: *Wiki projektu DCPro*. [Online; navštívené 10.12.2018].
URL https://homeproj.cesnet.cz/projects/dcpro/wiki

[6] Cloudbric Corp.: *Reflection Attacks and Amplification Attacks*. [Online; navštívené 29.11.2018].
URL https://www.cloudbric.com/blog/2015/03/reflection-attacks-and-amplification-atttacks/

[7] Cloudflare: *Learning Center - Learn About DDoS Attacks*. [Online; navštívené 25.11.2018].
URL https://www.cloudflare.com/learning/

[8] Docker Inc.: *Build and Ship any Application Anywhere*. [Online; navštívené 18.12.2018].
URL https://hub.docker.com

[9] Fielding, R.; Gettys, J.; Mogul, J.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Jún 1999.
URL https://tools.ietf.org/html/rfc2616

[10] Free Software Foundation: *ping(8) - Linux man page*. [Online; navštívené 26.11.2018].
URL https://linux.die.net/man/8/ping

[11] Imperva: *DNS AMPLIFICATION*. [Online; navštívené 1.12.2018].
URL
https://www.incapsula.com/ddos/attack-glossary/dns-amplification.html

[12] Information Sciences Institute: Transmission Control Protocol. RFC 793, September 1981.
URL https://tools.ietf.org/html/rfc793

[13] Jenkins: *Jenkins User Documentation*. [Online; navštívené 18.12.2018].
URL https://jenkins.io/doc/

[14] Jirovský, V.: *Kybernetická kriminalita : nejen o hackingu, crackingu, virech a trojských koních bez tajemství*. Praha: Grada, prvé vydanie, 2007, ISBN 978-80-247-1561-2.

[15] Kuka, M.: *Hardwarově akcelerované zařízení pro ochranu před DoS útoky*. Diplomová práca, Vysoké učení technické v Brně. Fakulta informačních technologií, 2016.
URL https://primo.lib.vutbr.cz/primo-explore/fulldisplay?docid=420BUT_Aleph000141831&context=L&vid=420BUT&search_scope=Everything&tab=default_tab&lang=cs_CZ

[16] Leyden, J.: *BIGGEST DDoS ATTACK IN HISTORY hammers Spamhaus*. [Online; navštívené 26.11.2018].
URL https://www.theregister.co.uk/2013/03/27/spamhaus_ddos_megaflood/

[17] Martin Fowler: *Continuous Integration*. [Online; navštívené 18.12.2018].
URL https://www.martinfowler.com/articles/continuousIntegration.html

[18] Mills, D. L.: Network Time Protocol (Version 3). RFC 1305, Marec 1992.
URL https://tools.ietf.org/html/rfc1305

[19] Mockapetris, P.: Domain Names - Implementation and specification. RFC 1035, November 1987.
URL https://www.ietf.org/rfc/rfc1035.txt

[20] Mohit, K.: *Biggest-Ever DDoS Attack (1.35 Tbs) Hits Github Website*. [Online; navštívené 26.11.2018].
URL https://thehackernews.com/2018/03/biggest-ddos-attack-github.html

[21] Nexusguard: *DDoS Threat Report 2018 Q1*. [Online; navštívené 26.11.2018].
URL https://www.nexusguard.com/threat-report-q1-2018

[22] Noction: *DDoS Amplification Attacks*. [Online; navštívené 29.11.2018].
URL https://www.noction.com/blog/ddos-amplification-attacks

[23] Oikarinen, J.; Reed, D.: Internet Relay Chat Protocol. RFC 1459, Máj 1993.
URL https://www.ietf.org/rfc/rfc1459.txt

[24] P. Ferguson, D. S.: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, Máj 2000.
URL https://tools.ietf.org/html/rfc2827.html

[25] Pellerin, J.: *nose*. [Online; navštívené 19.11.2018].
URL https://nose.readthedocs.io/en/latest/

[26] Postel, J.: User Datagram Protocol. RFC 768, August 1980.
URL https://tools.ietf.org/html/rfc768

[27] Postel, J.: Internet Control Message Protocol. RFC 792, August 1981.
URL http://www.networksorcery.com/enp/rfc/rfc792.txt

[28] Python Software Foundation: *html-testRunner 1.1.2*. [Online; navštívené 19.11.2018].
URL https://pypi.org/project/html-testRunner/

[29] Python Software Foundation: *nose-html-reporting 0.2.3*. [Online; navštívené 19.11.2018].
URL https://pypi.org/project/nose-html-reporting/

[30] Python Software Foundation: *unittest – Unit testing framework*. [Online; navštívené 19.11.2018].
URL https://docs.python.org/2/library/unittest.html

[31] Radware: *Amplification Attack*. [Online; navštívené 29.11.2018].
URL https://security.radware.com/ddos-knowledge-center/ddospedia/amplification-attack/

[32] Šiška, P.: *Systém pro ochranu před DoS útoky*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=21101

[33] Spirent Communications: *Spirent TestCenter Automation Overview Manual*.
URL http://support.spirent.com

[34] TR instruments spol. s r.o.: *Modulární zátěžový generátor/analyzátor Spirent TestCenter*. [Online; navštívené 16.12.2018].
URL http://www.trinstruments.cz/spirent-testcenter

# Prílohy

# Príloha A

# Obsah pamäťového média

```
/
├── dcpro/
├── README.txt
├── text/
└── xburza00_BP.pdf
```

**Adresár `dcpro`** obsahuje zdrojové kódy vytvoreného testovacieho systému a vytvorenej kolekcie testov.

**Súbor `README.txt`** obsahuje návod na spustenie.

**Adresár `text`** obsahuje zdrojové súbory textu práce pre možnosť úpravy a opätovného prekladu systémom LaTeX, vrátane zdrojových súborov použitých obrázkov.

**Súbor `xburza00_BP.pdf`** obsahuje text práce vo formáte PDF.