SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Zvýšenie odolnosti webových aplikácií proti útokom typu DDoS, za pomoci horizontálneho škálovania

Správa z riešenia projektu

Predmet: Princípy informačnej bezpečnosti
Vyučujúci: Ing. Michal Valiček
Vypracoval: Miroslav Hájek
Akademický rok: 2020 / 2021
Semester: Letný

Obsah

1 Dostupnosť ako bezpečnostný atribút

Zabezpečenie nepretržitého prístupu k webovým službám je očakávanou a takmer nevyhnutnou požiadavkou pre akýkoľvek významnejší informačný systém. Predstavuje neoddeliteľnú súčasť obrazu o spoľahlivosti ich prevádzkovateľov pôsobiacich vo virtuálnom priestore internetu. Aspekt dostupnosti sa prejavuje tým, že údaje sú k dispozícii pre autorizovaných používateľov okamžite a bez nečakaných obmedzení [1]. Odlišná interpretácia definuje pojem dostupnosti ako ochranu proti zlomyseľnému zatajovaniu informácií. Spoločným menovateľom pre oba tieto pohľady je dôraz na všadeprítomnosť služby v ľubovoľnom čase potreby. Taký stav je vskutku ideálny, ale je možné sa mu aspoň priblížiť predovšetkým identifikáciou bodov zlyhania alebo miest prieniku a ich následným systematickým eliminovaním.

Pre bezpečné nakladanie s informáciami nestačí samotná dostupnosť, ale zároveň je potrebné pri návrhu a prevádzke systémov myslieť aj na dôvernosť a integritu údajov. Spolu tvoria tradičný model informačnej bezpečnosti označovaný ako tzv. *CIA triáda* (Confidentiality, Integrity, Availablity), ktorý sa často spolieha na vyváženosť a rovnocennosť týchto troch prvkov [1]. Nutno poznamenať, že to úplne neplatí, pretože nežiadaným znemožnením prístupu k zdrojom sa ich neporušenosť a zabezpečenosť proti neoprávnenému prezeraniu, či úprave, stáva bezpredmetná. O dostupnosť sa teda opierajú všetky ďalšie bezpečnostné predpoklady, ktoré má systém napĺňať.

1.1 Kľúčový zabezpečovatelia dostupnosti

Komunikačné technológie často tvoria chrbtovú kosť väčšiny moderných biznisov pričom ich hlavnou úlohou je sprostredkovanie informácií naprieč organizáciou a podieľajú sa na riadení podnikových procesov. Okrem ľudského kapitálu sa spoliehajú na tri kľúčové prvky: *softvér, hardvér a počítačovú sieť* [1]:

Softvér: Softvér je najkritickejším komponentom spomedzi vymenovaných, pretože na základe príkazov v kóde programov je ovládaný hardvér a sieťové zariadenia. Všetky potenciálne útoky a ich dopady musia byť riešené primárne na úrovni softvéru. Na napadnutie sú využívané zraniteľnosti systému, sprístupnené prelomením nedostatočne zabezpečeného verejného rozhrania služieb. Najčastejším cieľom útočníkov je dosiahnutie kontroly nad zariadením alebo vyvolaním chaosu vo fungovaní prevádzky.

Zlyhanie programového vybavenia nemusí byť iba v dôsledku nepriaznivých vonkajších vplyvov, ale tiež sa prejavujú chyby spôsobené nekorektným návrhom alebo implementáciou systému s odchýlkami od požadovaného správania. Tieto chyby sú vnesené neúmyselne najčastejšie programátorom. Počas behu aplikácie môžu nastať zlyhania operačného prostredia zapríčinené nedostatkom pamäte pri alokácii, zaplneniu diskového úložiska alebo uviaznutím systému.

Hardvér: Poruchy hardvéru bývajú zriedkavejšie, ale o to podstatne fatálnejšie pre celkový chod, keď výpadok nie je adresovaný redundanciou komponentov. Prinavrátenie do funkčného stavu znamená výmenu zariadenia za iné prevádzkyschopné, či už dočasnou úpravou fyzickej infraštruktúry alebo neodkladnou montážou náhrady. Duplikáciou napr. diskov cez RAID dosiahneme síce vyššiu dostupnosť ale za cenu integrity z dôvodu zdvojením dát [1], preto je vždy potrebné mať na pamäti vyváženosť bezpečnostných vlastností navzájom.

Sieť: Obmedzením počítačových sietí je ich priepustnosť podmienená šírkou prenosového pásma a réžiou spotrebovanou na obsluhu zvolených komunikačných protokolov. Problémy nastávajú v situáciach, kedy sa zahltí sieťová linka. Za normálnych okolností rešpektujú uzly v sieti signalizáciu zaznačenú do paketov a prispôsobia rýchlosť vysielania, čím sa po istom čase uľaví náporu. Útočník, ktorý chce saturovať serverové pripojenie to pochopiteľne nerešpektuje a preto by sa nadmerná premávka mala presmerovať a filtrovať. Pokiaľ bude smerovač vystavený niekoľkonásobnej záťaži než je schopný podporovať, určite vyvolá straty paketov a zvýšenú latenciu.

1.2 Dôvody viktimizácie prevádzkovateľov

Nedostupnosť služieb máva za následok, buď priame škody v podobe finančných strát alebo vplýva na pošramotenie nadobudnutej reputácie u klientov, ktorí si zlyhanie môžu spájať so stratou spoľahlivosti a dôveryhodnosti služby. Poškodenie reputácie sa radí so 47% medzi najväčšie obavy firiem počas kybernetického útoku [2]. Ďalšími zreteľmi starostí sú potenciálna strata ziskov s 21%, obmedzená dostupnosť s 12% a zníženie produktivity počas útoku na 7%. Čím väčší poskytovateľ a používanejšia webová stránka, tým rozsiahlejšie sú potenciálne dopady pri neočakávanom vyradení z prevádzky. Zároveň dochádza k adekvátnemu navýšeniu zriadenej odolnosti investovanej do predchádzania a aktívnej defenzívy proti útokom.

Motivácie a dôvody stojace za aktivitami úmyselne narúšajúcimi dostupnosť zvolených obetí v podobe webových aplikácií sa líšia od prípadu k prípadu, ale sú zhrnuteľné do nasledujúcich okruhov [3] [4]:

- *Kapitálové zisky* - nadobudnutie finančnej odplaty od objednávateľa alebo nekalé snahy o potopenie konkurencie v ekonomickej súťaži predstavujú významný hnací faktor pre útočníkov. Hlavným zámerom je lepšie peňažné zabezpečenie sa nehľadiac na taktiky vynaložené na tento účel. Úspešná realizácia vyžaduje značné technické zručnosti, pretože firmám, ktoré sú hlavných cieľom, ide o veľa. Zvykne sa dotýkať komerčných webstránok alebo služieb finančných inštitúcií (zaznamenané útoky na HSBC, BTC a Ethereum burzy), serverov a sieťových zariadení poskytovateľov webhostingu alebo internetového pripojenia (Deutsche Telekom, OVH, Dyn) alebo serverov herných spoločností (Steam, Blizzard, EA Sports) [3]

- *Pomsta* - prevažne frustrovaný jednotlivci snažiaci sa o odplatu za vnímanú nespravodlivosť, ktorú podľa nich prevádzkovateľ pácha. Počas operácie Payback z roku 2010 bol odplatou hackerskej skupiny Anonymous, za blokovanie stránok s torentami a pirátskym softvérom, útok odoprenia služieb na organizácie chrániace autorské práva. V decembri zhodili stránky MasterCard, Visa, Paypal a iných, ktoré vydržiavali donácie organizácii Wikileaks, pretože publikovala prísne tajné informácie americkej vlády.
- *Ideologické a politické presvedčenie* - útočník sa snaží dať hlasno najavo svoj nesúhlas s ideovo protichodnými názormi a postojmi znefunkčnením alebo poškodením platformy prostredníctvom ktorej oponent pôsobí alebo šíri svoj svetonázor. Takáto forma útokov sa označuje tiež za hacktivizmus, ktorého prívrženci hájia slobodu slova a právo na súkromie proti nadmernému sledovaniu. Pre predstavu napádaných webstránok boli prominentné útoky roku 2016 cielené na skupiny ako Black Lives Matter, Ku Klux Klan, Wikileaks, oboch prezidentských kandidátov v USA, taliansku a írsku vládu alebo európsku komisiu. [3]
- *Demonštrácia schopností* - ide o experimentálne útoky so zámerom hackera vyskúšať si nové techniky alebo predviesť svoje kompetencie.
- *Kybernetický terorizmus* - útočník je súčasťou vojenskej alebo teroristickej operácie s cieľom poškodenia nepriateľovi. Kritická infraštruktúra štátu prestavuje najčastejšie zasahovaný cieľ.

1.2.1 Teória rutinných aktivít

Výber primeranej obete má taktiež svoj podiel na úspešnosti škodlivého zásahu útočníka do prevádzky, pretože ten sa výrazne nelíši od konvenčného zločinu. Medzi obvykle spomínané zdôvodnenia páchania kriminality zo sociologickej perspektívy patrí kriminologická teória známa pod názvom *teória rutinných aktivít*. Vysvetľuje predpoklady, aké musí daný subjekt spĺňať nato, aby bol zasiahnutý. Vyslovuje, že zločin sa udeje vtedy, keď motivovaný útočník, so sklonmi na páchanie trestnej činnosti, príde do stretu s objektom ponechaného bez prítomnosti schopného strážcu [5].

Pokiaľ sú vytvorené priaznivé okolnosti na prelomenie do systému je veľká šanca, že sa motivovaný útočník bude snažiť o zneužitie bezpečnostnej diery. Na druhej stane pri absencii jediného kritéria, sa buď znižujú šance na viktimizáciu alebo sa úplne vylúčia. Podľa *teórie racionálneho jednania* koná útočník pri zvažovaní realizácie svojho činu racionálne, hoc sa jedná o obmedzenú rozumnosť a síce z uhľa pohľadu delikventa ide o cieľavedomé rozhodnutie, kde pozitívny obnos prevažuje nad možnými rizikami uskutočnenia.

Vhodnosť zamerania sa na zvolenú infraštruktúru pre potenciálneho páchateľa je zachytiteľné kritériami *VIVA* (Value, Inetria, Visibility, Accesibility) [3]. V kontexte útokov odmietnutia služby sa pod *hodnotou* rozumie dôležitosť rozbitia cieľa pre útočníka, teda či

daný internetový portál alebo herný server dosahuje dostatočné zisky, aby ich odstavením bola spôsobená dostatočná škoda. *Zotrvačnosťou* sa myslí odpor, ktorý kladie infraštruktúra voči útoku rozličnými bezpečnostnými mechanizmami. Medzi priamočiare praktiky patrí udržiavanie aktualizovanému systému so zaplátanými zraniteľnosťami alebo obmedzením počtu dopytov z jednej adresy. Služby s veľkou zotrvačnosťou sú schopné ustáť väčší nápor v prípade napadnutia. *Viditeľnosť* predstavuje rozsah verejnej prístupnosti a známosti webstránky. *Prístupnosť* značí jednoduchosť v dosiahnutí vytýčených sieťových uzlov, ktoré majú predstavovať obeť, použitou taktikou útoku bez povšimnutia. Rovnako sa spája so schopnosťou nezanechať stopy na mieste činu následkom neprítomnosti mechanizmu monitorovania a detekcie narušenia.

Relatívne vysokou hodnotou, viditeľnosťou a prístupnosťou a nízkou zotrvačnosťou sa cieľ stáva exponovanejší a tým žiadanejší pre útočníka. Oproti klasickému zločinu, kedy býva nutná prítomnosť páchateľa a obete na jednom mieste, kyberzločin dovoľuje útočníkovi pôsobiť cez internet takmer od hocikiaľ a maskovať sa proti odhaleniu.

1.2.2 Psychologická predeterminácia útočníka

Pokiaľ by nejestvovali indivíduá so zámerom druhému spôsobiť ujmu vyplývajúcu z nastolených motivačných faktorov nebolo by ani potrebné sa výrazne zaoberať zvyšovaním odolnosti informačných služieb, či brániť voči kriminalite ako takej. Určité vzorce ľudského správania naznačujú, že je prakticky nemožné sa pred týmito spoločenskými javmi vyhraniť.

Teória diferenciálnej asociácie tvrdí, že v spoločnosti existujú paralelne tak prosociálne, ako aj asociálne normy, postoje a spôsoby správania [6], čím sa vysvetľujú trestné činy dostatočne zabezpečených jedincov strednej vrstvy. Dochádza u nich k stotožňovaniu sa s antisociálnymi prístupmi na ceste za osobným úspechom. Zároveň páchatelia podvedome zľahčujú následky sociálneho zlyhania v spojitosti s stanovenými antikriminálnymi normami. V snahu neutralizovať svoje konanie popierajú zodpovednosti skrývaním sa za bezvýchodiskovosť situácie, neuznávajú význam obete prenášajúc naň vinu a ohraďujú sa konaním v záujme vyššieho princípu. Na základe teórie etiketovania je delikvencia len momentálny stav osobnosti, ktorý pramálo súvisí s psychickými vlastnosťami a správaním.

Predpokladom na spáchanie kybernetického zločinu sú okrem úvodných pohnútok aj požadované technické zručnosti útočníka. Ak pracuje človek radšej sám na seba, a hoc aj nedopatrením má záznam v registri trestov, je niekedy motivovaný vlastnou otáznejšou minulosťou voči bežným klientom sa uchýliť k predávaniu alebo prenajímaniu kompromitovaných strojov. Hacker si časom vybuduje ilegálny biznis, ktorým si dokáže zarobiť uspokojivý obnos [7]. Využívané programy sú vymieňané alebo predávané sprostredkovane cez fóra, sú reklamované a tamojšími administrátormi overované podobne ako bežný komerčný softvér. Na fórach prebiehajú diskusie a objavujú sa návody k schodným spôsobom ako si dostupný malvér sprevádzkovať a upraviť podľa potreby.

Vykonanie útoku so zámerom obmedziť dostupnosť cudzieho systému je podľa platnej legislatívnej úpravy na Slovensku trestným činom podľa *§247a Trestného zákona* a obdoby sú zavedené takisto v iných právnych poriadkoch (napr. v USA - Computer Fraud and Abuse Act a 18 U.S.C. § 1030, v Nemecku - Strafgesetzbuch: §303b Computersabotage). Činnosť neoprávneného zásahu do počítačového systému spadá v podstate do rovnakej oblasti ako poškodenie cudzieho majetku s trestami odňatia slobody pri preukázaní, od šiestich mesiacov až po 10 rokov podľa závažnosti [8].

2 Anatómia útokov Denial of Service

Internet predstavuje prostredie sprístupňujúce na jednej strane ohromnú kvantitu služieb, ale zároveň sprostredkúva útočníkom širokú paletu nástrojov umožňujúcich ich odstavenie. Útoky odoprenia služby - *Denial of Service (DoS)* - spôsobujú nežiaduci zásah do schopnosti legitímneho používateľa na prístup k zdrojom dostupných v počítačovej sieti. Zneprístupnenie sa realizuje vyčerpaním šírky pásma linky alebo systémových prostriedkov obete, či už CPU, operačnej pamäti alebo priepustnosti vstupno-výstupných operácií [4]. Pokiaľ sa na útoku podieľa značný počet zariadení označuje sa ako distribuovaný DoS skrátene DDoS.

Útoky typu DDoS sú na internete obrovským problémom, napriek snahe vyvíjať neustále lepšie metódy obrany v reakcii na stále sofistikovanejšie modifikácie techník útokov. V architektúre internetu prevládajú prvky so zameraním skôr na efektivitu a spoľahlivosť prenosu paketov medzi koncovými uzlami, než na silné zabezpečenie detailnou kontrolou prenášaného toku. Z distribuovanej povahy a autonómie administrácie nezávislých samostatných sietí, z ktorých je Internet zložený, a ktoré si riadi do veľkej miery každý poskytovateľ pripojenia zvlášť, by bola na systematické celoplošné politiky nevyhnutná ťažko dosiahnuteľná širšia dohoda. Zároveň platí, že akokoľvek je cieľový systém ochránený stále závisí od úrovne zabezpečenia ostatných uzlov v sieti. Okrem rôznorodého vynútenia pravidiel sú ďalšími predpokladmi na degradáciu služieb obmedzené výpočtové zdroje každej entity na trase, hlavne limitované kapacity vyrovnávacích pamätí.

Myšlienka uskutočnenia DoS útoku je pomerne priamočiara. Pokiaľ útočník disponuje väčšou celkovou rýchlosťou pripojenia, je schopný preťažiť linku obete a tým spomaliť spracovanie oprávnených požiadaviek. Prenesene sa uplatňuje zásada, že silnejší pri súboji vyhráva. Strana disponujúca lepším pripojením spravidla predurčí stav dostupnosti služby v kritických momentoch. Spustenie útoku iba z jedného akokoľvek výkonného stroja je pre útočníkov nevýhodné, pretože zmarenie zlovoľnej činnosti spočíva jednoducho vo vyčítaní adresy pôvodcu odchytávaním premávky, jeho následne zablokovanie a pridanie na čierne zoznamy.

2.1 Botnet

Centralizované prevedenie útoku odoprenia služby je z dnešného pohľadu nepriechodné. Najčastejšie sa preto na znefunkčnenie služby uplatňuje taktika ovládnutia rozsiahlej skupiny zraniteľných počítačov, ktoré dokáže útočník ovládať na diaľku a nasadiť do želanej ofenzívy. Napadnuté zariadenia ani ich užívatelia častokrát netušia, že sa stali súčasťou takéhoto zoskupenia, ktoré sa označuje ako *botnet*. Počítač slúžiaci útočníkovi na naplnenie nekalých úmyslov vzdialeným vykonaných povelov je tzv. *bot*, *zombie*, či *dron*. Boti sa správajú ako hybrid viacerých kybernetických hrozieb s pridanou hodnotou komunikačného kanála so schopnosťou koordinácie cez ovládacie miesta. Šíria sa podobne červom,

skrývajú sa pred detekciou ako vírusy a obsahujú útočne metódy toolkitov [9]. Vlastník armády botov je tzv. *botmaster*. Neprávom nadobudnuté výpočtové prostriedky riadi prostredníctvom command and control (C&C) infraštruktúry.

2.1.1 Komunikácia s botmi

DDoS útočné siete používajú spravidla tri typy architektúry: *Agent-Handler*, *Internet Relay Chat (IRC)* a *webovú architektúru* [4] [10]. Model Agent-Handler pozostáva z klientov - útočníkov, ktorí sa pripájajú na tzv. handler so zaneseným softvérovým vybavení na zisťovanie stavu a koordináciu agentov. Umiestňuje sa spravidla do zariadení s veľkým objemom sieťovej premávky a ich strategický výber umožňuje výrazne kamuflovať podozrivú komunikáciu. Terminológia *handler* a *agent* sa zvykne zamieňať s *master* a *démon*.

Medzičlánkom preposielania povelov od botmasterov sa rovnako môže stať verejný IRC server. Vtedy sa jedná o architektúru založenú na protokole Internet Relay Chat. Pôvodný účel využitia botov spočíval pri asistencii moderovania rušných četových miestností IRC kanálov [9]. Jedným z prvých bol bot Eggdrop napísaný už v roku 1993. V tom čase začali vznikať boti so zámerom útočiť na ostatných používateľov a IRC servery. Dovoľovali útočníkovi ukrytie sa za aktivity bota alebo dokonca za botov na viacerých počítačoch, ktorý neboli k útočníkom priamo vystopovateľný. Tým bolo umožnené napádať čím ďalej väčšie ciele.

Agenti sa po pridaní k botnetu ohlásia na dezignovaný IRC kanál a ďalej prijímajú a posielajú správy cezeň. Tento spôsob je lákavý pre jednoduchosť komunikácie v podobe krátkych textových správ príkazov a dostatočnú anonymitu bez silnej autentifikácie. Útočník nemusí udržiavať zoznam dostupných agentov, keďže po prihlásení sa na server vie zobraziť všetkých podriadených botov. Pre zložitejšie odhalenie napomáha využitie známych portov pre IRC (6667/TCP), pomerne veľká prevádzka na známych IRC serveroch a technika „preskakovania medzi kanálmi“ (channel hoping), kedy botmaster využíva zvolený IRC kanál iba na krátke obdobie.

Najpoužívanejším modelom je síce pre svoju flexibilitu IRC forma komunikácie, ale v posledných rokoch sa objavujú botnety založené na webových aplikáciach. Boty posielajú webovému serveru pravidelne informácie o svojom stave. Ovládané sú cez komplexné PHP skripty a komunikácia s agentami dokáže byť šifrovaná cez TLS a skrývať sa za bežnú webovú prevádzku na portoch 80/TCP, 443/TCP a tým odolávať tým bežným sieťovým filtrom. Narozdiel od IRC spočíva ich nesporná výhoda v nemožnosti únosu botnetu od svojho pôvodného tvorcu únosom četovej miestnosti.

2.1.2 Šírenie replikáciou škodlivého kódu

Nehľadiac na výber spôsobu komunikácie agentov so svojim command and control uzlom, musí ich sieť byť dostatočne rozsiahla na to, aby spôsobila znateľnejší dopadu na webové služby. Zároveň by mala disponovať metódami vlastnej replikácie sa na priľahlé na-

padnuteľné počítače. Priebeh rozširovania vplyvu botnetu nad zväčšujúcou sa skupinou hostiteľov sa odohráva v postupných fázach.

V prvom rade musí dôjsť k objaveniu zraniteľných hostov, potenciálnych budúcich botov. Útočník si môže vytipovať vhodnú známu obeť a pokúsiť sa o prevzatie kontroly manuálne, systematickým skúšaním prelomenia známych zraniteľností konkrétneho systému. Automatizované skripty, ktoré sú umiestňované do už nakazených počítačov sa nepotrebujú vopred špecificky zacieliť, ale dokážu si poskladať zoznam IP adries, ktoré bude postupne navštevovať a preverovať preddefinované nezaplátané bezpečnostné diery.

Skenovanie môže prebiehať **náhodne** [11], kedy každý kompromitovaný uzol v sieti generuje postupnosť ľubovoľných IP adries. Technika je použiteľná iba pri IPv4, pretože pri hustote rozloženia obsadených IPv6 adries by bol tento postup výrazne neefektívny. Náhodné skúšanie hostov vytvára veľký objem podozrivej sieťovej premávky smerovanej akiste medzi vzdialenými sieťami, ktoré normálne nekomunikujú, čím sa zvyšuje šanca na odhalenie takejto aktivity. Keďže nedochádza pri skenovaní k synchronizácii medzi infikovanými počítačmi rastie množstvo duplicitných dopytov na rovnaký už preverený koncový uzol s ich zväčšujúcim sa počtom.

Obdržaním zoznamu počítačov (**hitlist**) s ľahko prelomiteľnou obranou dokáže botnet usmerniť svoje šírenie. Známym vyhľadávačom verejných adries IoT zariadení s konektivitou k Internetu a prehľadom známych bezpečnostných dier je `shodan.io`. Kolíziam sondovania sa zabraňuje prerozdelením celého hitlistu na menšie časti, čím sa zabezpečí, že každý agent overí stroje z presne určeného rozsahu. Nevýhoda spočíva v nutnom zostavení celého zoznamu predtým než dôjde k samotnému rozširovaniu útočnej siete. Dôležité je zvolenie vhodnej veľkosti jeho dielov na preposielanie. Ak je zoznam rozsiahly tvorí sa značná sieťová premávka, krátky zoznam zapríčiní malú finálnu populáciu agentov.

Topologické skenovanie nasleduje prirodzene vznikajúce komunikácie objavujúce v sieti, aby sa dosiahlo presnejšie splynutie s bežným tokom paketov. Nakazený hostiteľ v podobe webového servera pošle do prehliadača klientov škodlivý kód a za správnych okolností sa ten dokáže dostať na iné webové servere, ktoré klient prezerá. Spoliehaním sa na správanie používateľov sa výrazne znižuje rýchlosť a úplnosť ovládnutia vyhovujúcich obetí a útočník nedokáže šírenie počítačového červa regulovať.

Predošlé varianty skenovania je užitočné upraviť na prehľadávanie cieľov **v lokálnej podsieti**, čím sa dajú nakaziť náchylné počítače za firewallom a agent pritom neprezrádza svoju lokáciu využívaním nadmernej intersieťovej výmeny správ.

Nachádzanie vektorov prieniku počas prechádzania zoznamom adries je uskutočňované, buď horizontálne, napríklad preverovaním rovnakého otvoreného portu, či mierenej zraniteľnosti naprieč všetkými cieľmi, alebo sa koná vertikálne a síce testovaním širokého spektra malvérom pribalených utilít snažiac sa vniknúť dnu hocako. S vykonávaním vybranej metódy pomalým tempom má útočník príležitosť zostať nebadaný po dlhšiu dobu a ponúka sa mu čas na preverenie možností získania kontroly nad systémom. Po fázach

náboru a vykorisení nového bota sa naň prenáša škodlivý kód pochádzajúci z centrálneho úložiska (červ 1i0n) alebo sa siahne zo zariadenia, ktoré bol pôvodcom nákazy v predošlom kroku, tzv. „back-chaining“ (červy Morris, Ramen) [11].

2.2 Klasifikácia typov DDoS útokov

Útok odoprenia služby závisí od schopnosti čo najväčšej alebo špeciálne zameranej sieťovej premávky, aby informačná služba neakceptovala požiadavky legitímnych žiadateľov. Odohráva sa privlastnenie si celej vyhradenej linky alebo výpočtového výkonu prevádzkovateľa útočníkom. Ak je vyústením prevalcovanie serverovej infraštruktúry obete nedokonalosťou zabezpečovacích mechanizmov dochádza k zrušeniu dostupnosti s následkami už uvedenými. Aby sme porozumeli metódam efektívnej obrany je nevyhnutné zatriediť a kategorizovať objavujúce sa hrozby, s ktorými sa ciele DDoS útoku vedia stretnúť. V literatúre existujú rozličné taxonómie separujúce problematiku z rôznych uhlov pohľadu [4] [10] [11] [12].

Základné rozdelenie DDoS útokov spočíva v identifikácii ich primárneho vektora. Webová aplikácia býva zneprístupnená, buď vyčerpaním šírky prenosového pásma hrubou silou záplavy paketov, alebo vyplytvaním systémových prostriedkov sémantickým útokom na komunikačný protokol.

Volumetrické útoky (veľkoobjemové útoky) saturujú kapacitu linky rozmanitou plejádou nálože. Spoločným menovateľom je technika flooding (záplava). Populárnou formou útoku je posielanie UDP datagramov na náhodné porty s úmyslom zapríčiniť overovanie, či sú porty otvorené a spôsobiť reakciu servera signalizačnými správami ICMP Destination port unreachable. So snahou donútiť systém, aby sa venoval predovšetkým záškodníckym spávam útočníka, pracuje tiež záplava paketmi ICMP Echo Request (Ping) s následnou odpoveďou ICMP Echo Reply. Na VoIP služby je účinným SIP Flood, ktorý zaplaví SIP proxy s falošnými správami pre začatie hovoru SIP INVITE [10]. Webový aplikačný server je možné zahltiť záplavou HTTP(S) požiadaviek GET alebo POST na náhodné alebo existujúce URI webstránky. Dopytovanie sa na neexistujúcu cestu okamžite vráti stavový kód rádu 400 pre chybu klienta, ale rovnako sa bude server musieť zaoberať spracovaním takejto požiadavky, len sa stáva jednoduchšie pozorovateľnou z prístupových logov.

Protokolové útoky sa prižívujú na zraniteľnosti v návrhu komunikačného protokolu na transportnej až aplikačnej vrstve OSI, ktoré spoliehajú na priebežné ukladanie stavových informácii o naviazaných reláciach. Rozšírené sú taktiky na zneužitie časovačov stavového automatu protokolu TCP a príznakov v TCP segmentoch, ktorými sa odosielateľ a prijímateľ dohadujú na priebehu výmeny správ.

Počas TCP SYN Flood je doručené také množstvo podnetov na otvorenie spojenia segmentami s príznakom SYN, ktoré vyústi v zaplnenie pamäte vyhradenie na uchováva-

nie aktívnych relácii. Server je povinný pri zahajovaní TCP spojenia cez 3-way handshake a obdržaní SYN odoslať SYN+ACK a počkať stanovenú dobu. Timeout býva dostatočný na to, aby dokázal útočník ponechať tabuľku relácií zaplnenú iba svojimi podvratnými požiadavkami. Schodnou ochranou je zavedenie tzv. TCP Cookie. Systém po prijatí TCP SYN odošle TCP SYN+ACK a nevytvorí v pamäti žiadnu reláciu [12]. Po prijatí právoplatnej odpovede TCP ACK sa spätne dopočíta TCP sekvencia paketov a až vtedy sa zaháji spojenie.

TCP RST útok sa zameriava na rušenie nadviazaných spojení medzi serverom a klientmi, kedy však je nutné poznať zdrojovú IP adresu klienta, pretože útočník háda začiatkom konverzácie náhodne započaté sekvenčné čísla. Ak uspeje preberie reláciu a zruší ju. Za bežných okolností je také niečo ťažko spáchateľné, lebo TCP spojenia zvyknú mať krátke trvanie a vznikajú ad-hoc.

Nastavením príznaku PSH je serveru nanútené okamžité vyprázdnenie vyrovnávanie pamäte klientovi a odoslanie potvrdzujúcej správy ACK. Pri enormnej hromade takýchto výziev nebude schopný server vybavovať ďalšie požiadavky, čím dôjde k zrušeniu dostupnosti webových a podobných služieb poskytovaných z daného bodu.

Zlomyseľným zásahom do riadenia toku TCP spojenia predchádzajúceho zahlteniam, presnejšie vyžiadaním a udržiavaním nulovej veľkosti okna príjemcu, sa vie útočník obsadiť všetky dostupné spojenia v tabuľke spojení a tým znemožniť nadviazanie komunikácie so serverom ostatným. Určenie veľmi malej nenulovej veľkosti okna spôsobí rozdrobenie odpovedí na veľmi malé fragmenty. Prevenciou býva zapojenie Nagelovho algoritmu (RFC 896) do TCP implementácie, ktorého úlohou je zamedziť veľkej réžii pri posielaní miniatúrneho payloadu.

Na relačnej vrstve modelu OSI je vďačným protokolom na útoky spotrebujúce značný výpočtový výkon Secure Sockets Layer (SSL/TLS). Keďže majorita webových aplikácií v súčastnosti používa HTTPS je dôležité si uvedomiť, že proces šifrovania spolu s réžiou pri výmene kľúčov v SSL handshaku predstavuje pre server násobnú náročnosť oproti klientovi. Znovu sa naskýta prostý útok záplavou iniciácií TLS spojenia alebo opätovné dohodnutie SSL komunikácie (renegotiation), ktorá zvykne zahŕňať zmenu parametrov šifrovania alebo vyžiadanie certifikátu servera. Riešením je blokovanie takýchto požiadaviek alebo „SSL offloading“ do špecializovaného hardvéru [12].

Spotrebovanie všetkých ponúkaných spojení HTTP protokolu aplikáciou webového servera sú preferované tzv. „low and slow“ útokmi. Agent sa maskuje akoby za veľmi pomalú rýchlosť pripojenia, no v skutočnosti zámerne rozdrobuje svoj dopyt na krátke fragmenty a posiela ich s významným oneskorením, aby držal spojenie otvorené čo najdlhšie. Dôvody existujúcich obmedzení tkvejú v maximálnom počte súborových deskriptorov procesu alebo únosnej hladine bežiacich procesov. Zástupcom tejto skupiny útokov je Slowloris a R-U-Dead-Yet? (RUDY). Z dôvodu malej generovanej premávky, prechádza pomerne ľahko bez povšimnutia, pretože nemá dopad na iné atribúty systému.

Rovnako ako botnety slúžia na zitenzívnenie devastačného prúdu paketov, tak môžu nepriamo zapájať do útoku aj malvérom nenakazené počítače technikami odrazu a zosilnenia. Útoky s odrazom (RDoS a DRDoS) zapríčiňujú poslanie paketov s podvrhnutou zdrojovou adresou cieľa útoku záchytným bodom (pivotom). V domnienke správnosti pôvodcu správy sa odpoveď doručí v konečnom dôsledku na obeť. Samo o sebe to nemá až taký význam, okrem odklonenia nevyhnutného prúdu odpovedí od skutočných spúšťačov požiadaviek na tretie strany. Nastavením cieľovej IP adresy na broadcastovú adresu lokálnej podsiete (L2 alebo L3 OSI) sa útok zosilní, pričom zasiahne všetky počítače v spoločnom broadcastovom segmente siete. Tiež je priechodné odrazenie útoku od viacerých reflektorov. Na týchto princípoch fungujú útoky Smurf a Fraggle.

Domain Name System (DNS) amplifikačné útoky využívajú podstatu odrazeného útoku, ale obsahujú obohatenie zaručeného nárastu veľkosti DNS odpovede voči dopytu. Faktor zväčšenia DNS query response sa pohybuje od 1,1 pri jednom A zázname (example.net), 2,75 v prípade troch AAAA záznamoch (youtube.com) alebo dokonca 3,5 (yahoo.com) pri siedmich štvor-áčkových záznamoch. Protokol DNSSEC ponúka cez otvorené rekurzívne resolvery až 30-násobnú amplifikáciu [12], z dôvodu početnosti vrátených NS záznamov a digitálnych podpisov v DS a RRSIG záznamoch. Nástrojom `dig` sme objavili odpoveď s 24-násobným zväčšením a podobne iné domény dosahovali amplifikácie bežne v rozsahu 13 - 18-krát:

```
dig +dnssec +trace opendns.com
dig +dnssec @b.root-servers.net opendns.com
```

Každoročný prehľad v trendoch kybernetických hrozieb publikovaných európskou inštitúciou ENISA konštatuje, že takmer 80% všetkých DDoS útokov v treťom kvartáli 2019 boli TCP SYN záplavy [13], stávajúc sa najpopulárnejším typom útoku spolu s DNS odrazenou amplifikáciou. V apríli 2019 bol zaznamenaný SYN Flood útok s prietokom až 580 miliónov paketov za sekundu. Vyskytujú sa hlavne multivektorové útoky, čím je ich zdolanie komplexnejšie. Zároveň dominovali útoky kratšie ako 10 minút, ktorých bolo 84% zo zaznamenaných. Celkovo došlo k nárastu v počte nahlásených útokov o 241% oproti rovnakému obdobiu predošlého roku. [13].

2.3 Ochrana spevnením sieťovej ochrany

Obranné mechanizmy na úspešné zvládnutie útokov odoprenia služby rozlišujeme primárne podľa úrovne pripravenosti reakcie na **proaktívne** a **reaktívne** stratégie [11].

Prevenciou sa zabezpečuje systém proti prieniku priebežným monitorovaním a pravidelným sťahovaním a inštaláciou bezpečnostných záplat. Súčasne sa budujú bezpečnostné politiky organizácie, ktoré rátajú s klasickými vektormi DDoS útokov a premýšľa sa nad adekvátnym minimalizovaním dopadu hrozieb s nimi spojenými. Prakticky osvedčenými riešeniami sú „resource accounting“, čiže účtovanie a limitovanie počtu vyhradených spojení pre každú IP adresu pristupujúcu k službe pri maximálnej frekvencii odpovedí od

servera, alebo sa využíva „resource muliplication“, kedy sú zdroje systému duplikované na viaceré zariadenia a prichádzajúca záťaž je vyvažovaná medzi nimi.

Reaktívne spôsoby sa usilujú o zmiernenie útoku počas jeho konania. Pre automatizované riadenie defenzívy by mali byť obranné prvky schopné detekcie útoku odlíšením od typickej sieťovej premávky s ohľadom na odstránenie vlastnej chybovosti pri identifikácii falošných poplachov a prehliadnutí škodlivých činností. Zisťovanie prítomnosti pokusov na odoprenie služieb sa sleduje rozpoznávaním podozrivých vzorov podľa analýzy predošlých útokov alebo pozorovanie anomálií v komunikácií. Strážny komponent je sústredený na preddefinované scenáre alebo sa natrénuje na bežnej premávke a spustí varovanie po prekročení prahových hodnôt.

2.3.1 Remotely Triggered Black Hole

Nežiadúcu premávku je vhodné zlikvidovať na okraji autonómneho systému vzdialene spustiteľnou čiernou dierou (RTBH). Nahlásením zdrojov alebo častejšie cieľa útoku vie správca siete vložiť do spúšťacieho smerovača statickú cestu na virtuálne rozhranie *Null*, ktoré spôsobí zahodenie paketov [14]. Postihnutá služba bude síce odrezaná od Internetu, čiže útočník odoprie dostupnosť pre ostatných používateľov, ale zmiernia sa negatívne dopady na zvyšnú infraštruktúru. Na edge routeroch sa musí vopred nakonfigurovať statická cesta pre ďalší skok stanovený za black hole a zakázať preň odpovedanie o nedosiahnutí cieľa cez ICMP:

```
ip route 192.0.2.1 255.255.255.255 Null0
interface Null0: no ip unreachables
```

Na trigger smerovači sa musí aktivovať BGP politika na presmerovanie cesty k obeti po obdržaní vloženej statickej cesty so spúšťacou značkou (tagom) a povolenie jej distribúcie medzi iBGP peerov a zároveň nepropagovanie mimo autonómneho systému [15]:

```
route-map blackhole
    match tag 66
    set ip next-hop 192.0.2.1
    set origin igp
    set community no-export
router bgp 65535
    redistribute static route-map blackhole
```

Poskytovateľ pripojenia môže na podnet zablokovať premávku na obeť známej IP adresy cez pravidlo na spúšťacom routeri. Uvedená statická cesta musí byť po skončení odobratá zo spúšťacieho routera, ktorý rozpošle BGP route withdrawal svojim iBGP peerom:

```
ip route 172.5.23.1 255.255.255.255 Null0 tag 66
```

2.3.2 IP Spoofing

Na zabránenie spätnej väzby paketov vracajúcej sa naspäť botom útočníka, sa zvykne pri záplavových útokoch vložiť do paketu sfalšovaná zdrojová adresa. Vychádza z potreby útočníka zostať v anonymite a prípadne zmiasť bezpečnostnú obranu vzbudením dojmu, že nadmerná premávka pochádza od rozptýlených podnecovateľov. Spoofing sa hodí u reflektorových útokov alebo v situáciach, kedy si útočník praje, aby bol vinený za iniciátora určitý počítač. Prostredníctvom zapojenia Ingress a Egress filtrovania sú poskytovatelia pripojenia na trase schopný zabrzdiť škodlivú premávky takmer v zárodku. Ukradnuté zdrojové adresy sa pre navodenie dôveryhodnosti vyberajú z rozsahu verejne smerovateľných IP adries (RFC 1918, RFC 3330). Privátne adresy sa totiž štandardne zahadzujú na smerovačoch registrovaním príslušných ACL pravidiel ako realizácia **Egress filtrovania** Uvedená konfigurácia odhodí pakety zo súkromného rozsahu a prepustí ostatné [16]:

```
access-list 110 deny ip 192.168.0.0 0.0.255.255 any
access-list 110 permit ip any any
```

Podvrhnuté IP adresy sa rozlišujú podľa techniky ich selekcie. Najmenej sofistikovaný postup spočíva vo vygenerovaní náhodného 32-bitového čísla, ktoré predstavuje spoofovanú zdrojovú adresu. Vymyslená IP adresa nesmie byť úplne svojvoľná, ale pochádzať z platných podsieti, pre ktoré router má záznam v smerovacej tabuľke alebo sa môže nachádzať niekde po ceste k obeti. **Ingress filtrovanie** (RFC 2827) totiž povoľuje v striktnom režime smerovať iba pakety, pre ktoré existuje vo FIB (Forwarding information base) tabuľke mapovanie reverznej cesty (Reverse Path Forwarding) cez rovnaké rozhranie. Ak nedokáže byť zabezpečené symetrické smerovanie potom „loose“ mód dovoľuje akceptovať spätnú cestu cez ľubovoľné rozhranie smerovača:

```
ip verify unicast reverse-path list            # Strict mode
ip verify unicast source reachable-via any     # Loose mode
```

Naproti utkvelej predstave prevláda snaha o zúžitkovanie validných zdrojových adries agentov útočníka, tam kde je to uskutočniteľné [11].

2.3.3 Linux Netfilter

Zníženie záťaže na systémové zdroje pridelené na tvorbu paketov odpovede je dosiahnuteľné cez *rate limiting*. Linux disponuje v rámci *procfs* (`/proc/sys/net/ipv4/`) premennými na obmedzenie rýchlosti pri odpovedaní maskou zvolenými ICMP správami `icmp_ratelimit`, `icmp_ratemask` a `icmp_echo_ignore_broadcasts`[1]. Predvolene je frekvencia odpovedí na ICMP Destination Unreachable 1 sekunda a ignorujú sa broadcastové správy. Proti SYN záplave sa systém zabezpečí cez `/etc/sysctl.conf` povolením TCP cookies a limitovaním počtu otvorených, klientom nepotvrdených TCP spojení [17]:

[1] `https://man7.org/linux/man-pages/man7/icmp.7.html`

```
net.ipv4.tcp_syncookies = 1          # Aktivácia TCP SYN cookie
net.ipv4.tcp_max_syn_backlog = 1024  # Maximálny počet nepotvrdených spojení
net.ipv4.conf.all.rp_filter = 1      # Ingress filter (RPF) proti IP spoofingu
```

Z bezpečnostného hľadiska sa tiež odporúča ponechať pravidlá firewallu na najmenšej úrovni priepustnosti smerom dnu. Ak webová aplikácia beží na porte HTTPS/443 a administrácia vstupuje na server cez SSH/22 zo stáleho rozsahu adries intranetu, môže jednoduchá konfigurácia firewallu vyzerať nasledovne [18]:

Kód 1: Pravidlá pre povolenie prichádzajúcej komunikácie cez SSH a HTTPS

```
iptables --append INPUT --protocol tcp --dport 443 --jump ACCEPT
iptables --append INPUT --protocol tcp --dport 22 \
         --source 192.168.0.0/24 --jump ACCEPT
```

Kód 2: Politiky reťazí pravidiel sú zahodiť všetko okrem odchádzajúcich paketov

```
iptables --policy INPUT DROP
iptables --policy FORWARD DROP
iptables --policy OUTPUT ACCEPT
```

3 Škálovanie webových aplikácií

Rozvrhnutie architektúry na nasadenie webovej aplikácie, ktorá ustojí legitímnu premávku, ale tiež škodlivo nadmerný prúd vykonštruovaných požiadaviek, vyžaduje balans medzi sústredením prostriedkov na jednom mieste do homogénneho monolitu a ich parcelovaním znásobením počtu inštancií. Návrhár musí nájsť kompromis medzi obstaraním silnejšieho počítača alebo viacerých počítačov. Priklonenie sa k vybranej alternatíve predurčí budúce komplikácie pre rozšírenie platformy hosťujúcej webovú službu.

Vertikálne škálovanie spočíva v navyšovaní výpočtovej sily stroja pridaním, urýchlením alebo zväčšením kapacity hardvérových komponentov - procesora, pamätí, diskového poľa, či sieťovej karty - utesňujúc vzájomnú väzbu dielcov a očakávajúc ich nevyhnutnú vzájomnú kompatibilitu. Uľahčuje sa tým údržba a kontrola systému, zachovanie konzistentnosti dát bez nutnosti navyše častí na zabezpečenie ich integrity. Zároveň sa to odráža na nižšej energetickej spotrebe voči mnohým spoločne rovnako výkonným duplikátom. Všetko za cenu o poznanie vyšších obstarávacích nákladov tohto kompaktného balíka s výhľadom na obmedzený rozsah pre upgrade, vytvárajúc jediný bod zlyhania pre prípadný celkový výpadok poskytovanej aplikácie.

Horizontálne škálovanie je oproti vertikálnemu oveľa odolnejšie proti celkovému zlyhaniu služby, keďže sa spolieha na viacero paralelne bežiace kópii aplikácie na separátnych zariadeniach. Komponenty sú lacnejšie a jednoduchšie na upgrade. Nevyhnutnosťou je spravidla zosieťovanie a distribúcia záťaže úloh pomedzi uzly. Zavedením prostredníka takouto reverznou proxy sa zvyšuje réžia pri nadväzovaní spojení medzi klientom a serverom. Okrem toho rastie závislosť na externých prepájacích prvkoch a ich zabezpečení.

3.1 Algoritmy vyvažovania záťaže

Load balancing spočíva v rovnomernom optimálnom prerozdelení prichádzajúcich úloh na spracovanie medzi všetky účastne entity [19]. Výsledkom je simultánne vykonávanie viacerých požiadaviek, každá vykonávaná pôvodnou rýchlosťou limitovanou špecifikami každého komponentu osve [20]. Ideálnou symetrizáciou záťaže sa zvyšuje priepustnosť a spoľahlivosť, naplno sa využívajú dostupné výpočtové zdroje, zvyšuje sa odolnosť proti chybám a rastie stabilita služby. Úspešnosť nasadeného riešenia závisí od zapojenia vhodného algoritmu pre konkrétny typ očakávaných pripojení. Najčastejšie sa vyplatí aplikovať dynamické algoritmy so za behu upraviteľným váhovaním a riadením od centrálneho uzla - load balancer. Tie berú do úvahy aj momentálne informácie o usmerňovanej sieťovej premávke s dostatočne nízkou réžiou, čím dosahujú dobrý pomer medzi férovosťou a efektívnosťou.

Round-Robin prerozdeľuje objavujúce sa požiadavky postupne každému serveru z rotujúceho zoznamu započnúc v náhodne zvolenom. Ak sa zariadenia líšia výkonnosťou

je vážením dosiahnuteľné priradenie väčšej záťaže výkonnejším strojom. Napríklad pre priradenie váh trojici serverov $A = 4, B = 3, C = 2$ je ich dobrou permutáciou počas plánovacieho cyklu postupnosť $AABABCABC$ [21]. Na server pripadne množstvo spojení podľa určenej váhy podielom zo sumy všetkých váh: $W_k / \sum_i W_i$. Rôznym počtom pridelených požiadaviek sa môže prejavovať dynamická nerovnováha medzi servermi, keď je väčšina dopytov s rozsiahlou odpoveďou presmerovaná na rovnaký uzol. Časová zložitosť výberu uzla na spracovanie je $\mathcal{O}(1)$, pretože sa vyberie ďalší v poradí zo zoznamu.

Least Connections priradí požiadavku serveru s najmenším počtom aktívnych spojení. Medzi servermi s rovnakou záťažou sa môže aplikovať Round-Robin. Odporúča sa využiť v prípadoch, kedy bývajú spojenia rozlične dlho otvorené, pretože postupné prideľovanie neberie do úvahy rozdielne trvanie takýchto relácií a tým sa môže dôjsť k nahromadeniu klientov na málo hostoch. Pri váženej verzii algoritmu, kde počet aktívnych spojení uzla je C_i a a váha W_i, smeruje príchodzie spojenie na taký server j, že platí [21]:

$$\frac{C_j}{W_j} = \min_{1 \leq i \leq n} \left\{ \frac{C_i}{W_i} \right\}$$

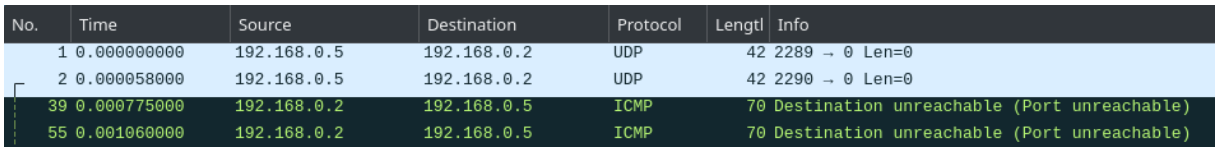
Least Connections má časovú zložitosť $\mathcal{O}(n)$ od počtu serverov, pretože hľadá minimálnu dĺžku fronty spracúvaných požiadaviek od každého servera.

Least Time je menej častá dynamická stratégia, kedy sa vyberá server podľa najmenšieho počtu aktívnych spojení a súčasne na základe najnižšej priemernej latencie vyrátanej z času trvania po prijatie prvého bajtu alebo kompletnej, či nekompletnej odpovede [22].

Random vylosuje na spracovanie požiadavky náhodný zo skupiny dostupných serverov, čo môže viesť ku krátkodobému navýšeniu zaneprázdnenosti jediného servera, zatiaľčo ostatné sú nečinné. Priemerne sa však každej destinácii dostane rovnaký podiel klientov [19]. Stratégia sa oplatí využívať v prípade rozsiahlejších serverových fariem, kedy znižuje kladivový efekt na frekventovane pridávané a odoberané servery oproti odlišným metódam. Účinnou modifikáciou je náhodný výber dvoch serverov, z ktorý sa vyberie ten s menším počtom aktívnych spojení s Least Connections alebo Least Time. Namiesto priklonenia sa k absolútne najlepšej možnej voľbe pri nekompletnej plánovacej informácii sa algoritmus „Power of Two Choices“ vyhýba horšej variante výberom lepšej možnosti z dvoch. S väčším počtom náhodných výberov konverguje práve k Least Connections [23] za cenu výkonnosti vyvažovania, pretože sa blíži k vyhľadávaniu globálne najkratšej fronty.

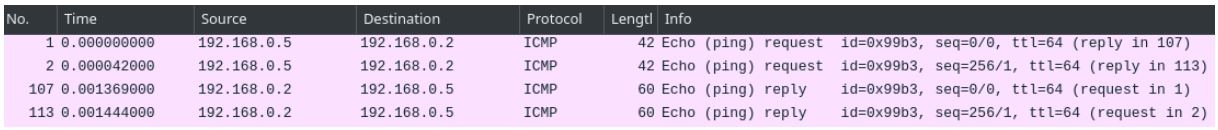
Hashing sa uplatňuje pri voľbe servera na základe vlastností spojenia alebo hlavičky HTTP požiadavky. Konzistentné hašovanie dokáže zaručiť, že pri zmene bežiacich serverov sa minimalizuje a zrovnomerňuje počet spojení, ktoré budú zaradené na spracovanie inému serveru. Pri prerozdeľovaní zahašovaním zdrojovej IP adresy sa klientovi dostáva

odpovede zakaždým od rovnakého uzla pokým je v prevádzke, čiže dokáže byť uchovaná perzistencia HTTP relácie s cookies. Tento prístup sa nazýva best-effort *sticky session* [20], pretože sa klient akoby „prilepí“ na server z farmy a pokiaľ je to možné výhradne s ním komunikuje. Na druhej strane sa dá uplatniť hašovanie podľa URI cieľa požiadavky alebo iného atribútu s dostatočnou variabilitou.



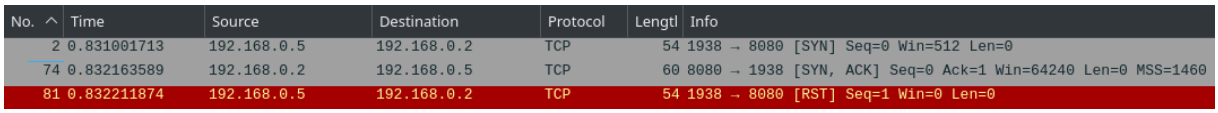
No.	Time	Source	Destination	Protocol	Lengtl	Info
1	0.000000000	192.168.0.5	192.168.0.2	UDP	42	2289 → 0 Len=0
2	0.000058000	192.168.0.5	192.168.0.2	UDP	42	2290 → 0 Len=0
39	0.000775000	192.168.0.2	192.168.0.5	ICMP	70	Destination unreachable (Port unreachable)
55	0.001060000	192.168.0.2	192.168.0.5	ICMP	70	Destination unreachable (Port unreachable)

(a) UDP záplava



No.	Time	Source	Destination	Protocol	Lengtl	Info
1	0.000000000	192.168.0.5	192.168.0.2	ICMP	42	Echo (ping) request id=0x99b3, seq=0/0, ttl=64 (reply in 107)
2	0.000042000	192.168.0.5	192.168.0.2	ICMP	42	Echo (ping) request id=0x99b3, seq=256/1, ttl=64 (reply in 113)
107	0.001369000	192.168.0.2	192.168.0.5	ICMP	60	Echo (ping) reply id=0x99b3, seq=0/0, ttl=64 (request in 1)
113	0.001444000	192.168.0.2	192.168.0.5	ICMP	60	Echo (ping) reply id=0x99b3, seq=256/1, ttl=64 (request in 2)

(b) ICMP záplava



No.	Time	Source	Destination	Protocol	Lengtl	Info
2	0.831001713	192.168.0.5	192.168.0.2	TCP	54	1938 → 8080 [SYN] Seq=0 Win=512 Len=0
74	0.832163589	192.168.0.2	192.168.0.5	TCP	60	8080 → 1938 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
81	0.832211874	192.168.0.5	192.168.0.2	TCP	54	1938 → 8080 [RST] Seq=1 Win=0 Len=0

(c) TCP SYN záplava

Obr. 1: Priebeh toku paketov záplavových DoS útokov z pcap záznamu vo Wireshark

3.2 Redundancia nižších vrstiev RM OSI

Odolnosť sieťovej infraštruktúry proti poruchám a výpadkom aktívnych sieťových prvkov, s vedľajším rovnako dôležitým efektom navýšenia dátového prietoku komunikačných spojov, sa utužuje vyvažovaním záťaže na nižších vrstvách modelu RM OSI rôznymi metódami. Agregáciou liniek na spojovej vrstve sa vyberá sieťový prepoj zo skupiny, ktorým sa paket pošle. Viaccestným smerovaním na sieťovej vrstve sa rozhoduje medzi trasami cez ktoré sériu paketov presmerovať. Virtuálny smerovač (IPVS) na transportnej vrstve určí aktívy uzol spomedzi klastera pod spoločnou IP adresou.

Agregácia liniek sa uplatňuje pokiaľ nestačí maximálna prenosová rýchlosť samotného jedného sieťového rozhrania a je nevýhodné, či dokonca nemožné vymeniť sieťovú kartu. Vtedy sa oplatí zoskupiť niekoľko fyzických rozhraní do jedného logického linku, spôsob nazývaný ako „trunking“ alebo „bonding“. Navyšovanie prenosových rýchlostí medzi generáciami technológii na fyzickej vrstve môže takto prebiehať lineárne. Ak je vyžadovaný GigabitEthernet (1 Gbit/s), ale nie je k dispozícii, dokáže Trunked Fast Ethernet zabezpečiť rýchlosti 200 - 800 Mbit/s, v porovnaní s obyčajným 100 Mbit/s Fast Ethernet. Rámce sú striedavo posielané aktívnymi redundantnými linkami vrámci zhluku, čím sa zabezpečí zvýšená dostupnosť a symetrizácia záťaže na participujúcich portoch.

Link Aggregation Control Protocol (LACP), ktorý je súčasťou štandardu IEEE 802.3ad, dovoľuje sieťovému zariadeniu vyjednanie automatického združenia liniek výmenou LACP

paketov medzi partnermi. Priebežnými keepalive správami kontroluje LACP priechodnosť spoju pre zamedzenie straty paketov v nefunkčnej linke a overuje chyby spôsobené nesprávnym fyzickým zapojením vznikajúcim prekrížením kabeláže: „loopback links“ alebo „split-trunk“. Prakticky sa na smerovači nastaví kanál do ktorého sú pridané porty[2]:

```
interface port-channel 1
   ip address 192.168.0.1 255.255.255.0
interface range g2/0/0-1
   no ip address
   channel-group 1 mode active
```

Viaccestné smerovanie pridáva spoľahlivosť v miestach, kde vedú do cieľovej destinácie aspoň dve trasy. *Equal-cost multi-path routing* (ECMP) je stratégia smerovania, kedy sa striedavo posielajú pakety cez viaceré ďalšie skoky s rovnako dobrými metrikami cesty. V praxi je tento prístup málo uplatňovaný pre komplikácie viažuce sa k dynamickému výberu spomedzi dostupných smerov na rovnomerné rozloženie záťaže [24]. Rozličné spojenia tiež zvyknú mať rôzne veľkosti MTU (maximum transmission unit) a variabilné oneskorenia vedúce k zbytočnému preusporiadaniu paketov správy mimo poradia. Pri použití mnohých alternatívnych preskakujúcich ciest môže dochádzať k strate paketov.

Zmiernenie popísaných negatív výberu pre next-hop, ale vyžaduje udržovanie si stavu prebiehajúcich tokov alebo zvýšené výpočtové nároky pre voľbu next-hop. *Modulo-N Hash* presmeruje paket na cestu podľa identifikátora toku z hlavičky paketu (najčastejšie zdrojová a cieľová adresa) modulo počet dostupných skokov. Ak dôjde k zmene musí sa upraviť $(N-1)/N$ tokov [24]. *Hash-Threshold* rovnomerne mapuje uzly do výstupu hašovacej funkcie a podľa porovnania hašu identifikátoru toku s hranicami oblasti je zvolený next-hop. Pri zmenách sa upravuje cesta štvrtine až polovici tokov. *Highest Random Weight* počíta hash zakaždým zároveň z hlavičky paketu a kľúča pre next-hop. Zvolí sa ďalší skok s najvyšším výsledným číslom. Za väčšej časovej náročnosti sa mení pri pridaní alebo odobratí cesty, smer už len $1/N$ tokov. Cisco router využívajúci na smerovanie protokol OSPFv2 uplatní ECMP jednoduchým nastavením[3]:

```
router ospf 1
maximum-paths 2
```

Virtuálny smerovač zamedzuje oknám v obsluhe IP adresného priestoru udržiavaním viruálnej IP adresy, pre ktorú eviduje redundantné aplikačné servery na zabezpečenie vysokej dostupnosti služby. Narozdiel od konceptu základnej dostupnosti, kedy sa vyvinie systém spĺňajúci len nevyhnutné funkčné požiadavky, prináša vysoká dostupnosť znásobenie počtu komponentov, aby v prípade údržby, chyby alebo zlyhania komponentu mohol zaujať miesto obsluhy úloh náhradný prvok.

[2] https://www.cisco.com/c/en/us/td/docs/ios/12_2sb/feature/guide/gigeth.html

[3] https://www.techrepublic.com/article/how-to-configure-equal-cost-multi-path-in-ospf/

Odstránenie jediného bodu zlyhania nadobúda dve podoby konfiguráciami *Active/Standby* a *Active/Active* [1]. Odlišujú sa v móde prevádzky záložného riešenia, ktoré je buď nečinnou kópiou (Standby) v „Cold“ alebo „Hot“ režime, preberajúce zodpovednosť za spracovanie požiadaviek, iba ak toho hlavný člen nie je schopný, alebo sa neplytvá žiadnymi zdrojmi a záloha je operatívna (Active) zároveň s primárnou infraštruktúrou. Kontinuálna dostupnosť zachádza ešte ďalej so snahou zamaskovať akékoľvek časové prestoje spojené s poruchou, či už zámernou krátkodobou vznikajúcou pri údržbe alebo vážnejšou zapríčinenou útokom.

Virtual Router Redundancy Protocol (VRRP) umožňuje súhru viacerých uzlov pôsobiacich ako VRRP smerovače tvoriace jeden virtuálny router. Spoločne si volia zariadenie zodpovedné za zdieľanú vysoko dostupnú IP adresu. Zariadenie vrámci klastera v roli master má najvyššiu prioritu a v pravidelných intervaloch (najčastejšie 1 - 5 sekúnd) vysiela obežníkové správy *VRRP Advertisiment* na multicastovú adresu 224.0.0.18 [25], na ktorú sú prihlásené cez protokol IGMP všetky uzly v klasteri. Po inicializácií je hlavný uzol povinný oznámiť všetkým zariadeniam v LAN cez Gratious ARP správu zmena vlastníka IP adresy na svoju MAC adresu. Po výpadku, kedy nebol zachytený Advertisiment približne viac ako trojnásobok intervalu, nastupuje backup uzol s najbližšou nižšou prioritou a cez Gratious ARP prehodí IP adresu na seba. Keepalived je softvérový balík využívajúci Linux Virtual Server kernel modul umožňujúc nastaviť dynamický failover virtuálneho smerovača s VRRP [26]:

```
vrrp_instance malina {        # VRRP inštancia
    state MASTER              # štandardný stav inštancie MASTER / BACKUP
    interface eth0            # sieťové rozhranie
    virtual_router_id 1       # VRRP router id pre inštanciu
    priority 100              # priorita VRRP routera 0 - 255
    advert_int 1              # advertisement interval v sekundách
    virtual_ipaddress { 192.168.0.50 } # VRRP virtuálna IP adresa
}
```

DNS load balancing je staršou technikou prerozdelenia záťaže, pričom obľúbenosť tkvie vo výraznej jednoduchosti. Doménové meno služby je asociované s viacerými DNS A záznamami pre IPv4 adresy alebo AAAA záznamami pre IPv6 adresy. Následne sa vykonáva statický algoritmus round-robin, ktorý klientom pri DNS dopyte preusporiada zoznam vrátených serverov a klient si vyberá prevažne prvý z nich. Nevýhody sa prejavujú v nedostatočnej flexibilite v porovnaní s rovnomernosťou distribúcie záťaže, aká je dosiahnuteľná serverovým load balancerom [27].

DNS vyvažovanie záťaže sa vyznačuje značnou nepredvídateľnosťou plánovania smerovania sieťovej premávky z distribuovanej podstaty činenia rozhodnutí klientmi. Značným problémom je tiež DNS caching, ktorého zámer je obmedzenie vyťažovania autoritatív-

nych DNS serverov resolvermi. Čas zneplatnenie záznamu z cache býva výrazne dlhší ako je potrebné na okamžité zachytenie výpadku alebo rozširovanie kapacity serverovej farmy. DNS nerealizuje health-checks serverov a porucha spôsobí, že značné množstvo požiadaviek sa stratí, kým sa DNS záznam stihne všade propagovať.

Kód 3: BIND9: nastavenie DNS zóny .home

```
zone "home" {
    type master;
    file "/etc/bind/db.home";
}
```

Kód 4: Priradenie troch DNS A záznamov pre doménu website.home

```
website  IN  A  192.168.0.2
         IN  A  192.168.0.3
         IN  A  192.168.0.4
```

3.3 Reverzné proxy HAProxy a NGINX

Horizontálne škálovanie zabezpečuje spolu s vyvažovaním záťaže reverzný proxy server, ktorý je transparentným pre klientov využívajúcich webovú službu. Nasadenie reverznej proxy ponúka množstvo výhod týkajúcich sa schopnosti zrýchliť čas obratu prerozdelením požiadaviek, ich komprimovaním (gzip), dešifrovaním (TLS) a cachovaním. Dokáže do istej miery ochrániť serverovú farmu pred DDoS útokmi s ACL rate limiting. Netreba zabúdať, že proxy tvorí úzke hrdlo pre premávku a predstavuje centrálny bod zlyhania, ak nie je duplikované.

Obľúbenými open-source riešeniami pre load balancing na transportnej a aplikačnej vrstve sú softvérové balíky `haproxy` a `nginx`, ktorých konfiguráciu a výkonnosť porovnáme. *HAProxy* počúva na spojenia od klientov na ľubovoľnom počte `frontend`, ktoré sú prepojené na zvolený `backend`, kde sú umiestnené definície serverov, ktoré majú byť k dispozícii. Medzi základné sekcie patria tiež `global` a `defaults` určujúce správanie HAProxy daemona a nastavenia spoločné pre všetky frontend a backend sekcie.

NGINX konfigurácia pozostáva z dvoch hlavných blokov: `events`, ktorý podmieňuje vlastnosti bežiaceho NGINX ako celku, a kontext `http` obsahujúci direktívy podmieňujúce spracovanie HTTP(S) požiadaviek. Rozhranie pre pripájanie klientov tvoria virtuálne servery blokmi `server`, ktoré okrem poskytovania statického a dynamického webového obsahu cez vnorené kontexty `location`, dokážu slúžiť ako proxy pre skupinu serverov zhromaždených v `upstream` bloku.

3.3.1 Vyvažovanie záťaže

V konfiguračných súboroch oboch softvérových balíkov je najprv nutné nastaviť mód prevádzky na L7, respektíve L4 load balancing, pre HAProxy s `mode http`, respektíve `mode tcp` na NGINX, umiestnením serverov do kontextu `http {}`, resp. `stream {}`. V prevádzke sa nezaobídeme bez schopnosti *logging*, ktorá nám umožní vyhodnocovať stav load balancera v reálnom čase a zároveň záznamy pre spätnú analýzu odohraných javov po incidente.

Zvýšenie bezpečnosti systému s HAProxy daemonom sa dosiahne zmenou koreňového adresáru cez príkaz `chroot`, ktorý predvolene vytvorí symbolický odkaz z interného `/dev/log` na systémový súbor `/var/log/haproxy.log`, kam s `log` direktívou nastavíme preposielanie logov z lokálneho zariadenia `local0`. Destináciu pre logy na súborovom systému je zameniteľný za IP adresu alebo socket *syslog* servera štandardne na porte UDP/514. Pre NGINX podobne určíme cieľový súbor prístupových `access_log` a chybových záznamov `error_log`. Možnosť využitia *syslog* servera je samozrejmosťou: `access_log syslog:server=192.168.1.1`.

Kód 5: HAProxy mód prevádzky a logging

```
global
   log /dev/log local0
   chroot /var/lib/haproxy

defaults
  log global
  option httplog   # tcplog
  mode http        # tcp
```

Kód 6: NGINX mód prevádzky a logging

```
worker_processes auto;
events {
   worker_connections 1024;
}
http {  # stream
  access_log /var/log/access.log;
  error_log /var/log/error.log;
}
```

Aby proxy server reagoval na žiadosti klientov o otvorenie HTTP spojenia musí počúvať na porte 80 s direktívami `bind *:80`, resp. `listen 80;`. Následne sú všetky na port prichádzajúce požiadavky presmerované na skupinu webových serverov, cez `default_backend`, resp. `proxy_pass`, pričom aktivujeme vypĺňanie HTTP hlavičky *Forwarded*, čím bude adresa pôvodného klienta sprostredkovaná upstream serverom pre účely loggovania.

Kód 7: HAProxy: preposielanie požiadavok na pool serverov za reverznou proxy

```
frontend web
    bind *:80
    option forwardfor
    default_backend website
```

Kód 8: NGINX: preposielanie požiadavok na pool serverov za reverznou proxy

```
server {
   listen 80;
   server_name _;
   location / {
     proxy_pass http://website;
     proxy_set_header Forwarded
           $proxy_add_forwarded;
   }
}
```

Algoritmus rozdelenia záťaže medzi webové servery určíme práve v pomenovanej sekcii `backend`, resp. `upstream`. Najčastejšie aplikované stratégie ako sú Round Robin, Least Connection a Hešovanie, podľa zdroja alebo cieľa, sú podporované na HAProxy aj na NGINX. V HAProxy sa nachádzajú pod voľbami pre príkaz `balance` s pomenovaniami: `roundrobin`, `leastconn`, `source`, `uri`. [20] NGINX predvolenie volí stratégiu Round Robin, ktorá sa dá ekvivalentne zmeniť na: `least_conn;` `ip_hash;` alebo na `hash $request_uri consistent;` [22].

Serverom dokážeme obmedziť maximálny počet aktívnych spojení cez `maxconn n`, kedy zvyšné budú čakať vo fronte čím sa zníži sa nápor na servery. Príznakom `check` povolíme aktívne *health checks*, alebo je umožnené dočasne vyradiť server z prevádzky priradením stavu `backup` alebo `disabled`. Dokonca časovačom `slow_start` zamedzíme nárazovej záťaži na uzol dopadajúcej v predurčenom intervale po jeho spustení. Odlišné váhy pre rôzne výkonné servery určíme s prepínačom `weight n` [20].

Kód 9: HAProxy: vyvažovanie záťaže najmenej spojení na trojicu HTTP serverov

```
backend website
    balance leastconn
    server A 192.168.0.2:80
    server B 192.168.0.3:80
    server C 192.168.0.4:80
```

Kód 10: NGINX: vyvažovanie záťaže najmenej spojení na trojicu HTTP serverov

```
upstream website {
    least_conn;
    proxy_http_version 1.1;
    server 192.168.0.2:80;
    server 192.168.0.3:80;
    server 192.168.0.4:80;
}
```

3.3.2 Ochrana proti DDoS

Účinnou obranou proti škodlivo nadmernej sieťovej premávke, nielen voči útokom odoprenia služby, je zúčtovanie počtu spojení, rýchlosti súsledných dopytov alebo podobné sledovanie vyhradenia proporcie dostupných zdrojov alokovaných na IP adresu, ako primárneho identifikátora klienta, aby sa zamedzilo ich vyčerpanie. Do zaznamenanie užívania prostriedkov sa zapájajú pravidlá na zámerné riadenia prístupu cez *ACL*.

Najjednoduchšou technikou obrany proti zneužívateľom držiacim úmyselne dlho otvorené spojenia s nekompletnou správou je časovač, ktorého vypršaním sa spojenie automaticky uzavrie so stavovým kódom 408 - *Request Timeout* alebo 504 - *Gateway Timeout*, ale to len vtedy ak nie je nekompletná správa vložená do vyrovnávacej pamäte a rovno je odoslaná serveru za proxy.

Kód 11: HAProxy: timeout časovače

```
timeout connect 10s
timeout client 30s
timeout server 30s
timeout http-request 5s
option http-buffer-request
```

Kód 12: NGINX: timeout časovače

```
proxy_connect_timeout 10s;
proxy_read_timeout 30s;
proxy_send_timeout 30s;
client_body_timeout 5s;
client_header_timeout 5s;
```

Pokiaľ poznáme IP adresy známych narušovateľov môžeme z nich prístup zakázať, alebo naopak dovolíme pristupovať iba vyhradenému schválenému prefixu[4]. Riadiť prístup dokážeme taktiež špecificky pre zastaralú HTTP verziu 1.0, nepovolenú metódu PATCH, DELETE, či HTTP hlavičku, najčastejšie podľa *User Agent* s neprípustnou alebo neznámou hodnotou.

[4] `https://www.nginx.com/blog/mitigating-ddos-attacks-with-nginx-and-nginx-plus/`

Kód 13: Haproxy: ACL pre zdrojové adresy

```
http-request deny if {
  src -f deny_list.lst
}
tcp-request connection accept if {
  src 192.168.1.0/24
}
```

Kód 14: NGINX: ACL pre zdrojové adresy

```
location / {    # Deny list IP
  deny 123.123.123.0/28;
}
location / {    # Allow list IP
  allow 192.168.1.0/24; deny all;
}
```

Kód 15: HAProxy: Zablokovanie HTTP požiadavky podľa jej atribútov

```
http-request deny if HTTP_1.0                    # HTTP verzia 1.0
acl valid_method method GET POST OPTION HEAD
http-request deny if !valid_method               # Nepovolené HTTP metódy
http-request deny if { req.hdr(user-agent) -i -m sub curl }   # User agent
http-request deny unless { req.hdr(user-agent) -m found }
```

Aktívnou metódou je využitie vstavaných počítadiel s asociatívnymi tabuľkami, ktoré priraďujú ku kľúču, IP adrese, štatistiku počtu alebo frekvencie TCP spojení, resp. HTTP požiadaviek. HAProxy má na tento účel „stick tables“, ktoré sú nazvateľné po umiestnenie pod separátnu sekci `backend`. Pri vytvorení určíme, čo do tabuľky bude ukladať, v našom prípade IPv4 adresy `type ip` a vyhradíme miesto nad maximálny počet záznamov `size 1m`, tomto prípade 1 milión. O čase po akom sa má odstrániť nemodifikovaná položka určuje `expire 10s`, čiže podrží najstarší záznam na 10 sekúnd. Argument za `store` určuje ako hodnotu ku kľúču uložíme: počet HTTP požiadaviek v posuvnom časovom okne `http_req_rate(10s)`, počet spojení pre kľúč v tabuľke `conn_cur`, `conn_rate` [28]. Počítadlá sú obnovené automaticky cez `track-src0 src`.

Kód 16: HAProxy: obmedzenie počtu otvorených spojení na IP adresu na 10 zároveň

```
stick-table type ip size 1m expire 10s store conn_cur
tcp-request connection track-sc0 src
tcp-request connection reject if { src_conn_cur ge 10 }
```

Kód 17: HAProxy: obmedzenie frekvencie pripojení na používateľa

```
stick-table type ip size 1m expire 30s store conn_rate(3s)
tcp-request connection track-sc0 src
tcp-request connection reject if { src_conn_rate ge 10 }
```

NGINX používa direktívu `limit_req_zone`, ktorou sa pre zónu zdieľanej pamäte nastavia parametre limitovania požiadaviek, a príkazom `limit_req` sa určí kontext, kde sa má obmedzovanie HTTP požiadaviek algoritmom *leaky bucket* aplikovať [29]. Analogicky s `limit_conn_zone` sa vyhradí priestor pre tabuľku pripojení a pre konkrétnu URI cestu limitujeme počet TCP spojení, napríklad na 10 spojení na IP adresu s `limit_conn zone 10`. Premennou `$binary_remote_addr` určíme, že bude za kľúč považovať zdrojové IP adresy v binárnom tvare, čím ušetríme úložisko oproti obyčajnej `$remote_addr`. Zónou zvolíme

miesto vyhradené pre tabuľku, pričom stavové informácie pre približne 16 tisíc IP adries sa zmestia do megabajtu. Ak obmedzíme príchodzie spojenia na jedno za 100 ms s `rate=10r/s` a príde ďalšie v kratšom časovom intervale bude odmietnuté s kódom 503 - *Service Unavailable*, čo je nežiaduce v prípade, že sa aplikácia dopytuje nárazovo.

Atribútom `burst=20` sa vytvorí front s 20 miestami pre spojenia prichádzajúce v rýchlejšom slede ako umožňuje obmedzovač [29]. Nastanú situácie, kedy si nemôže dovoliť na odpoveď pre legitímnu požiadavku čakať, len preto že bola umiestnená do fronty. Parameter `nodelay` presmeruje napríklad 20 simultánnych požiadaviek z IP adresy na server okamžite, ale označí front za zaplnený a uvoľní sloty postupne podľa `rate` časovača.

```
limit_req_zone $binary_remote_addr zone=one:10m rate=10r/s;
limit_conn_zone $binary_remote_addr zone=addr:10m;
location / {
    limit_req zone=one burst=20 nodelay;   # 10 požiadaviek za sekundu
    limit_conn addr 10;                    # 10 aktívnych spojení naraz
}
```

3.3.3 SSL Termination

Výmena nezašifrovaných HTTP správ sa na webe stáva ojedinelejšia, pretože zvýšené bezpečnostné nároky komunikácie a vynucovanie zabezpečeného TLS/SSL spojenia zo strany webových prehliadačov[5] spôsobili rapídny rozmach HTTPS protokolu v posledných rokoch. Dešifrovanie na strane servera prináša však so sebou zvýšenú záťaž na spracovanie požiadavky, preto reverzné proxy umožňujú popri symetrizácii záťaže SSL premávku dešifrovať pred dosiahnutím webového servera a zašifrovať spätnú odpoveď klientovi - *TLS Offloading* [30], čím je SSL spojenie ukončené už na proxy. Hlavnou výhodou z pohľadu administrácie je centralizovaný a konzistentný manažment TLS konfigurácie a súvisiacich mechanizmov. Zjednodušuje sa predovšetkým redistribúcia certifikátov, keďže nemusia byť rozšírené na všetky uzly farmy. Okrem ukončenia SSL môže load balancer figurovať v režime *TLS bridging*, čiže správu k serveru opätovne zašifruje po medzikontrole. Z pozície *TLS pass-through* nenaruší šifrovaný kanál, ale k vyvažovaniu záťaže musí dochádzať na transportnej vrstve.

Certifikát verejného kľúča by sme na produkčnú prevádzku nadobudnúť od certifikačnej autority (napr. Let's Encrypt). Na účely testovanie alebo na použitie v uzavretej komunite môžeme vytvoriť certifikát podpísaný samým sebou - *self-signed certificate*

Kód 18: OpenSSL požiadavka (CSR) na samo popísanie certifikátu X.509 so súkromným kľúčom RSA o dĺžke 4096 bitov a ročnou platnosťou

```
openssl req -x509 -nodes -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
```

[5] `https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html`

HAProxy[6] a NGINX[7] umožňujú označiť komunikáciu na otvorenom porte za `ssl` a zahrnúť certifikát a cestu na disku k súkromný kľúču určeného na dešifrovanie. U skupiny upstream serverov stačí, aby podporovali nezabezpečený HTTP prenos.

Kód 19: HAProxy: TLS termination

```
frontend:
    bind *:443 ssl crt website.pem
backend:
    server A 192.168.1.1:80
```

Kód 20: NGINX: TLS termination

```
server {
    listen 443 ssl;
    ssl_certificate cert.pem;
    ssl_certificate_key key.pem; }
```

[6] https://www.haproxy.com/blog/haproxy-ssl-termination/
[7] https://docs.nginx.com/nginx/admin-guide/security-controls/terminating-ssl-http/

4 Monitorovanie webovej aplikácie

Bezpečnostná politika ochrany pred útokmi odoprenia služby musí ideálne okrem prevencie konfiguráciou zariadení a serverov, schopnou ustať nápor požiadaviek ich rozložením v čase alebo zahadzovaním, zahrňovať aj reaktívne stratégie. Bez existencie monitorovacieho riešenia je takmer nemožné zabezpečiť vhodnú včasnú odpoveď na kybernetickú hrozbu alebo rýchlo odhaliť závažné prevádzkové poruchy.

4.1 Štatistiky v reálnom čase

Obe porovnávané reverzné proxy na symetrizáciu záťaže, HAProxy a NGINX, disponujú modulmi, ktoré po aktivovaní sprístupňujú pohľad na momentálnu a agregovanú sieťovú prevádzku prostredníctvom prehľadnej webstránky na nastaviteľnom URL koncovom bode. Cez ACL sa štandardne obmedzuje prístup k štatistikám iba vopred schváleným správcovským IP adresám alebo pre *localhost*, odkiaľ sa údaje sprostredkujú ďalej.

Kód 21: HAProxy: štatistiky

```
frontend stats
    bind *:8404
    stats enable
    stats uri /stats
    stats refresh 10s
    stats admin if LOCALHOST
```

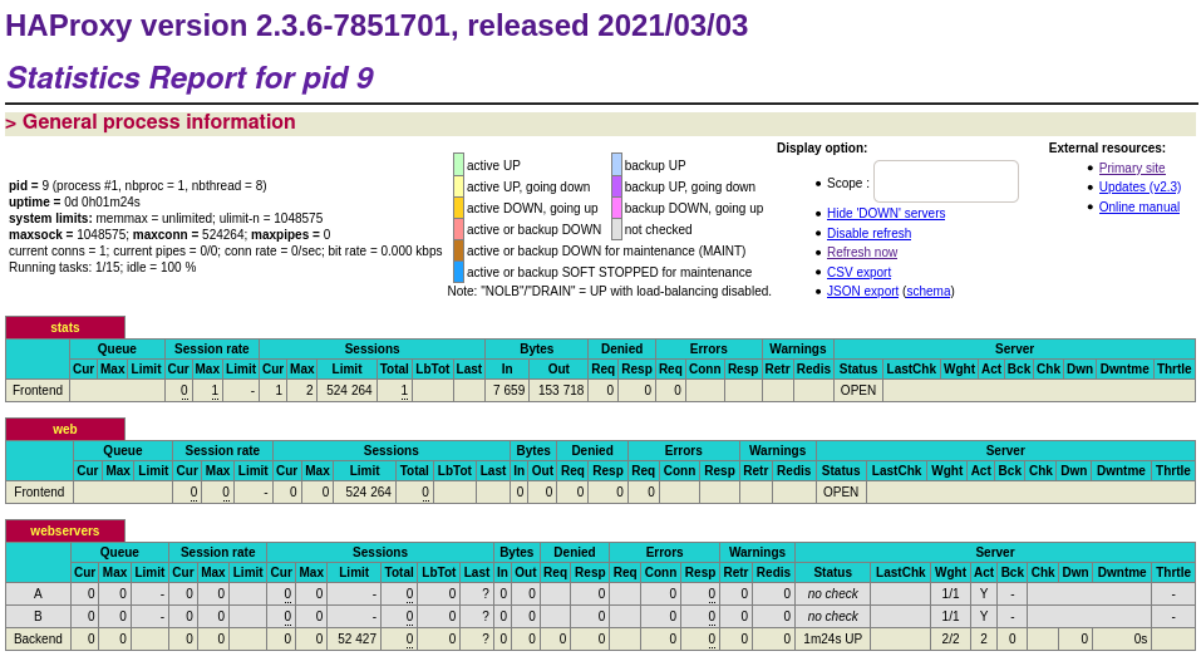
Kód 22: NGINX: štatistiky

```
location = /stats  {
    stub_status;
    access_log off;
    allow 127.0.0.1;
    deny all;
}
```

HAProxy poskytuje pomerne súhrnný náhľad na aktivitu cez vstavanú palubnú dosku[8]. Počítadlá sú rozdelené podľa rozhrania, respektíve pomenovaného bloku konfigurácie, pričom sa mierne odlišujú dostupné štatistiky pre `frontend` a `backend`. Zozbieravajú sa metriky ohľadom tempa nadväzovania spojení, relácií a frekvencie obdržaných požiadaviek (*Session rate*), a o kumulatívnom množstve zahájených a nadviazaných relácii, či počte HTTP požiadaviek s ich rozčlením podľa skupín stavových kódov (*Sessions*). Okrem toho sa meria kvantita prichádzajúcich a odchádzajúcich bajtov (Bytes), počet odmietnutých požiadaviek alebo odpovedí zamedzením prístupu cez explicitné ACL pravidlá (*Denied*) a počet požiadaviek s chybovým výsledkom (*Errors*).

Sekcia *Status* zhromažďuje rozvrhnutie a zdravia zaradených serverov, čiže terajší stav uzla, kedy naposledy bol skontrolovaný a s akým výsledkom (*LastChk*). Pre úplnosť sa dozvieme aplikované váhy (*Wght*), zaradenie medzi aktívne alebo záložné servery, počet neúspešných health checks (*Chk*), čas strávený vyradením z prevádzky (*Dwntime*) a počet zmien stavu servera medzi *up* a *down*. V časti Warnings pre `backend` je odsledovateľný počet opakovaných pokusov o vybavenie požiadavky a počet preplánovaní požiadavky po zlyhaní nadviazania spojenia so obslužným serverom, ak boli také možnosť povolené.

[8] `https://www.haproxy.com/blog/exploring-the-haproxy-stats-page/`



HAProxy version 2.3.6-7851701, released 2021/03/03

Statistics Report for pid 9

> General process information

pid = 9 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h01m24s
system limits: memmax = unlimited; ulimit-n = 1048575
maxsock = 1048575; maxconn = 524264; maxpipes = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.000 kbps
Running tasks: 1/15; idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
Scope :
Hide 'DOWN' servers
Disable refresh
Refresh now
CSV export
JSON export (schema)

External resources:
Primary site
Updates (v2.3)
Online manual

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	1	2	524 264	1			7 659	153 718	0	0	0					OPEN								

web

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	524 264	0			0	0	0	0	0					OPEN								

webservers

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
A	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	no check		1/1	Y	-				-
B	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	no check		1/1	Y	-				-
Backend	0	0		0	0		0	0	52 427	0	0	?	0	0	0	0		0	0	0	0	1m24s UP		2/2	2	0		0	0s	

Obr. 2: HAProxy palubná doska prevádzkových štatistík

V stĺpci *Queues* sa zobrazujú štatistiky o čakacích dobách požiadaviek klientov, kým sa server sa uvoľní.

NGINX, vo voľne prístupnej verzii, zobrazuje iba najzákladnejšie metriky o prijatých požiadavkách s modulom *HTTP Stub Status*[9]. Dozvieme sa aktuálny celkový počet otvorených spojení s klientmi a u koľkých z nich sú čítané hlavičky (*reading*), zapisované odpovede naspäť klientom (*writing*), alebo ktoré spojenia nečinne čakajú na požiadavky (*waiting*). Zároveň sa akumuluje počet iniciovaných (*requests*), akceptovaných (*accepted*) a vybavených (*handled*) požiadaviek. Počas bezporuchovej prevádzky by si mal zodpovedať počet prijatých a odbavených žiadostí:

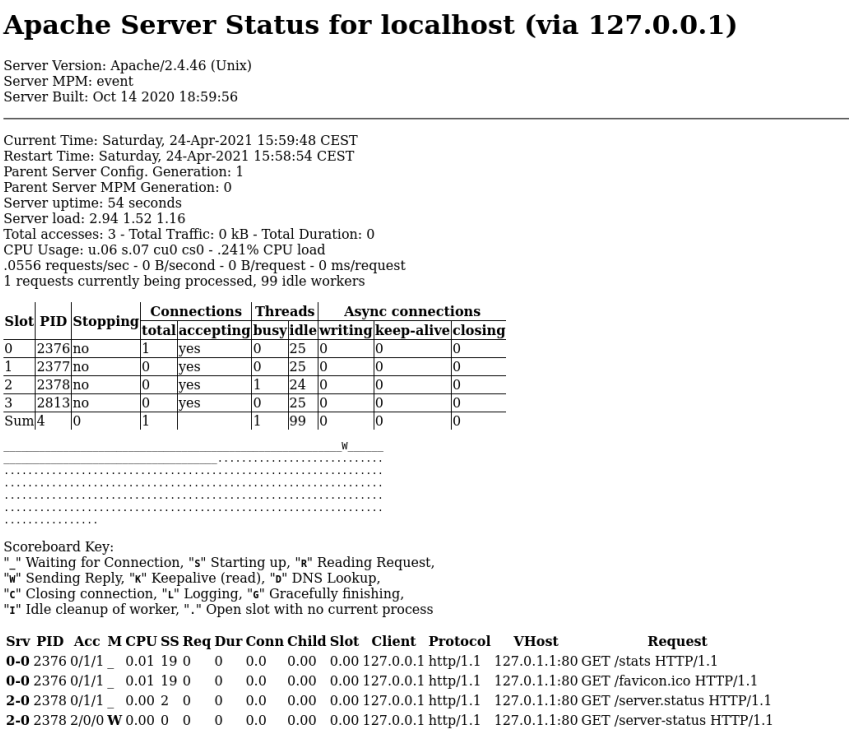
```
Active Connections: 2
server accepts handled requests
 3 3 5
Reading: 0 Writing: 1 Waiting: 1
```

Pre webovú aplikáciu umiestnenú na webovom serveri *Apache*, s ktorým môže vyvažovač záťaže nadväzovať spojenia, dokážeme získať detailné stavové informácie o rozložení požiadaviek medzi spracovateľské procesy a zobraziť využitie systémových zdrojov. HTML stránka, generovaná modulom `mod_status` pre akciu `server-status`[10], organizovane poskytuje detaily o *Worker* vláknach, konkrétne report početnosti podľa ich aktívnosti, typu spracúvanej úlohy naviazanej na príslušnú percentuálnu záťaž CPU. Modul dokáže zostaviť spriemerovaný počet požiadaviek za sekundu a prechádzajúci prietok bajtov. Súčasne

[9] `http://nginx.org/en/docs/http/ngx_http_stub_status_module.html`
[10] `https://httpd.apache.org/docs/2.4/mod/mod_status.html`

je dostupná varianta stavovej stránky so strojovo-čitateľným zobrazím pre automatizovaný zber metrík. Prevedenie smerovania stavových informácií na cestu `/server-status`:

```
<Location "/server-status">
    SetHandler server-status
    Require ip 127.0.0.1
</Location>
```



Apache Server Status for localhost (via 127.0.0.1)

Server Version: Apache/2.4.46 (Unix)
Server MPM: event
Server Built: Oct 14 2020 18:59:56

Current Time: Saturday, 24-Apr-2021 15:59:48 CEST
Restart Time: Saturday, 24-Apr-2021 15:58:54 CEST
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 54 seconds
Server load: 2.94 1.52 1.16
Total accesses: 3 - Total Traffic: 0 kB - Total Duration: 0
CPU Usage: u.06 s.07 cu0 cs0 - .241% CPU load
.0556 requests/sec - 0 B/second - 0 B/request - 0 ms/request
1 requests currently being processed, 99 idle workers

Slot	PID	Stopping	Connections		Threads		Async connections		
			total	accepting	busy	idle	writing	keep-alive	closing
0	2376	no	1	yes	0	25	0	0	0
1	2377	no	0	yes	0	25	0	0	0
2	2378	no	0	yes	1	24	0	0	0
3	2813	no	0	yes	0	25	0	0	0
Sum	4	0	1		1	99	0	0	0

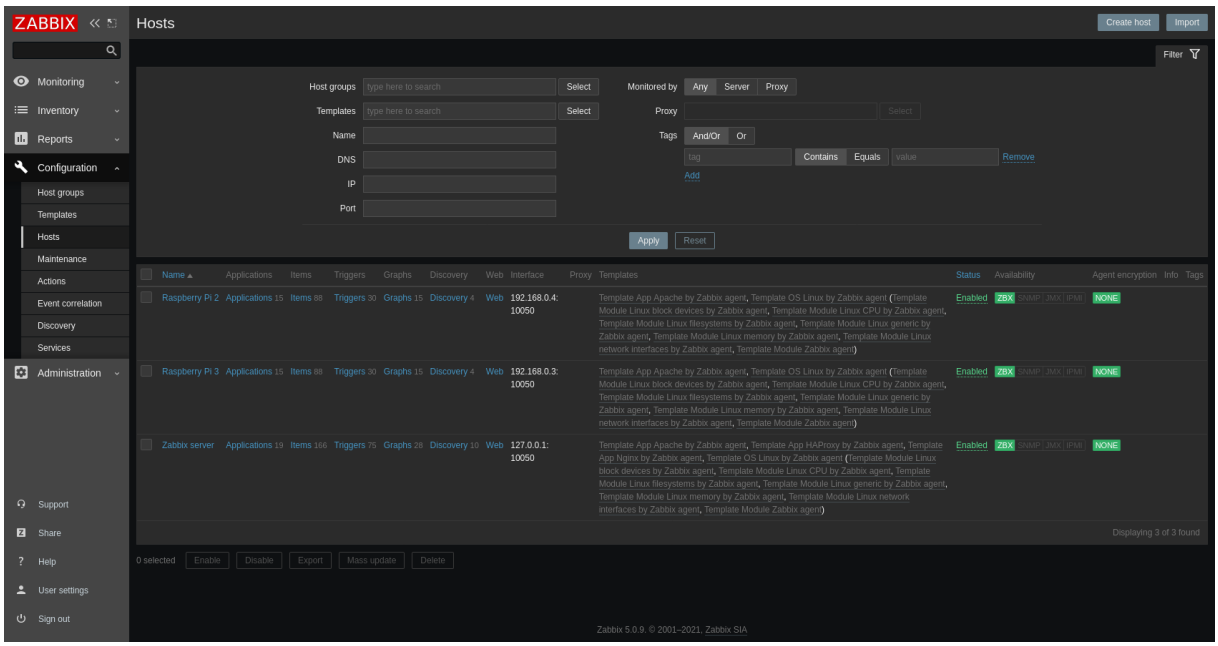
Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	CPU	SS	Req	Dur	Conn	Child	Slot	Client	Protocol	VHost	Request
0-0	2376	0/1/1	_	0.01	19	0	0	0.0	0.00	0.00	127.0.0.1	http/1.1	127.0.1.1:80	GET /stats HTTP/1.1
0-0	2376	0/1/1	_	0.01	19	0	0	0.0	0.00	0.00	127.0.0.1	http/1.1	127.0.1.1:80	GET /favicon.ico HTTP/1.1
2-0	2378	0/1/1	_	0.00	2	0	0	0.0	0.00	0.00	127.0.0.1	http/1.1	127.0.1.1:80	GET /server.status HTTP/1.1
2-0	2378	2/0/0	W	0.00	0	0	0	0.0	0.00	0.00	127.0.0.1	http/1.1	127.0.1.1:80	GET /server-status HTTP/1.1

Obr. 3: Stránka so stavovými metrikami webového servera Apache

Monitorovacie riešenie Zabbix umožňuje kolokáciu uvedených zozbieraných metrík z load balancera a webových serverov obohatenú o získané aktuálne prevádzkové vlastnosti prostredia uzlov v sieti a vyťaženie prostriedkov systému podľa priradených šablón. Obsahuje nástroje na organizáciu dát zo sledovanej infraštruktúry, nepretržitú vizualizáciu zvolených veličín, za predpokladu dostupnosti konkrétneho servera, a prispôsobiteľné úrovne varovania s výstrahami pre administrátorov cez emailové notifikácie v prípade dosiahnutia nežiaducich alebo kritických hodnôt.

Zabbix pozostáva z niekoľkých hlavných komponentov s podelenými zodpovednosťami [31]. Centrálny prvok je zosobnený Zabbix serverom sústreďujúcim všetku konfiguráciu, štatistické a operatívne dáta, ktoré si ukladá primárne do relačnej databázy MySQL alebo PostgreSQL. Súčasťou kompletného balíka je webové rozhranie pre priamočiary prístup k nastaveniam a nameraným hodnotám a spravidla, ale nie nutne, sa nasadzuje na rovnaké fyzické zariadenie ako Zabbix server. Na výber pre rozhranie sú webové servery Apache a NGINX. Zabbix server môže byť pri zbere metrík podporený so Zabbix proxy pre zníženie náporu na rozsah zhromažďovacích dopytov.

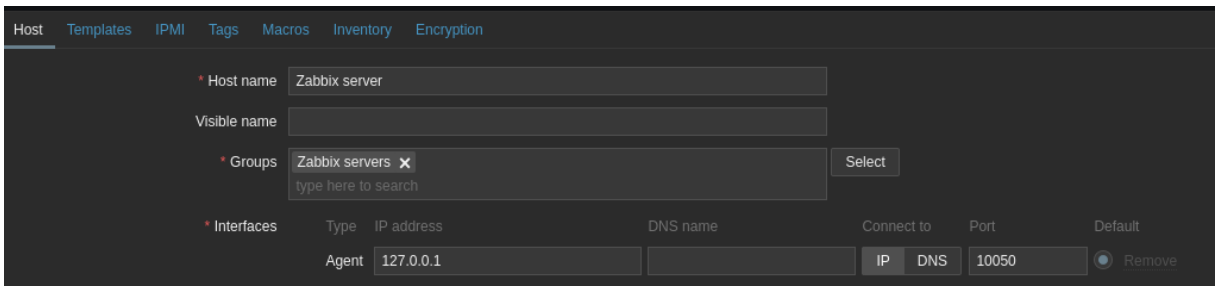


Obr. 4: Webové rozhranie Zabbix servera so zoznamom monitorovaných zariadení

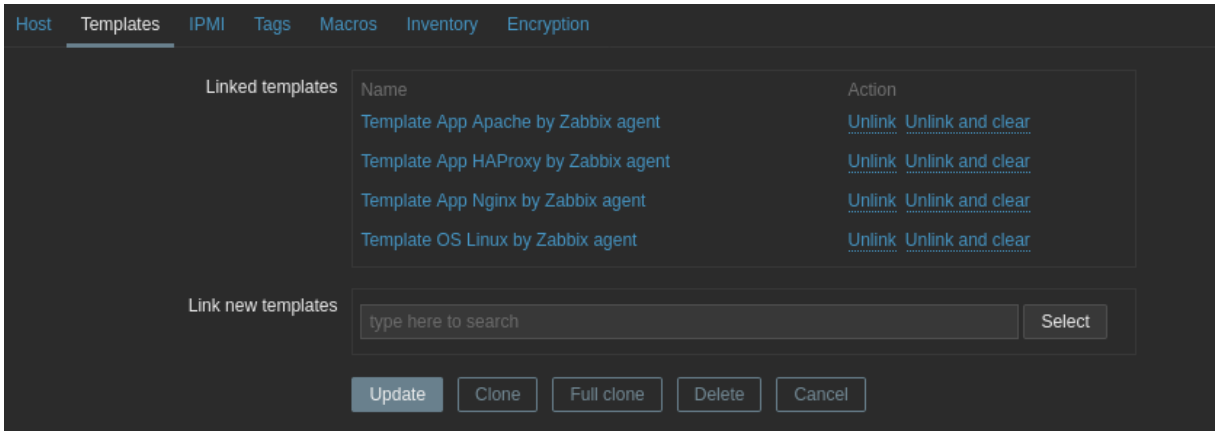
Zabbix agenti sú nasadené na cieľové monitorované uzly, aby mohli efektívne sledovať prostriedky cez systémové volania a odovzdávajú nameraný stav zdrojov Zabbix serveru alebo proxy podľa `zabbix_agent.conf`. Operácia dokáže prebiehať v dvoch odlišných režimoch: pri *passive checks* si server periodicky sťahuje údajov z agentov. Náročnejšie na spracovanie sú *active checks*, kedy si agent vypýta od servera zoznam sledovaných položiek a samostatne pravidelne posiela tieto údaje. Výmena správ sa odohráva vo formáte JSON.

Keď nastane potreba monitorovať veličinu dôležitú pre zabezpečenie fungovania služby, tak po nainštalovaní Zabbix servera s databázou a webovým rozhraním, nasadíme na záujmový hostiteľský systém Zabbix agent (medzi ďalšie možnosti patrí využitie protokolov a štandardov HTTP, SNMP, JMX, IMPI), kde pre pasívne kontroly nastavíme adresu Zabbix servera. Na serveri pridáme vytvoríme nový *Host* s ľubovoľným názvom, ale presnou IP adresou alebo doménovým menom agenta. V záložke *Templates* podľa monitorovaného softvérového balíka zvolíme zo širokej ponuky žiaducu šablónu podľa zamýšľaného módu zberu údajov. Na korektné fungovanie skriptu šablóny je nutné nastaviť softvérový balík podľa pokynov v priloženej dokumentácii k šablóne a v záložke *Macros* upraviť nastavenia od nášho prípadu použitia. U HAProxy, NGINX a Apache využíva Zabbix práve lokálne dostupnú URL pre štatistiky. Koncový uzol je nakoniec pridaný do zoznamu hostov a odteraz sa dostáva pod drobnohľadom jeho stav, ktorého výchylky sa zobrazujú v sekcii problémov.

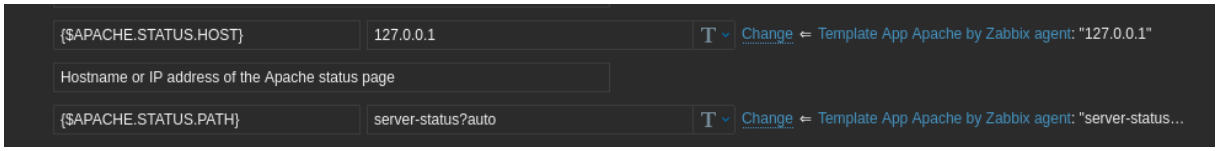
Skontrolovanie úspešnosti zhromaždenia hodnôt pre položky zo zariadenie môžeme uskutočniť na podstránke *Latest Data*, kde sa vyskytujú údaje pre všetky monitorované uzly z daného Zabbix servera zoskupené podľa použitých šablón. Zvolené metriky sa dajú prehľadne v prispôsobiteľných grafoch na viacerých *Dashboard* s meniteľným rozsahom zahrnutého historického vývoja.



Obr. 5: Základné nastavenia pri pridávaní nového hosta



(a) Pridávanie šablón vybraných softvérových balíkov agentom



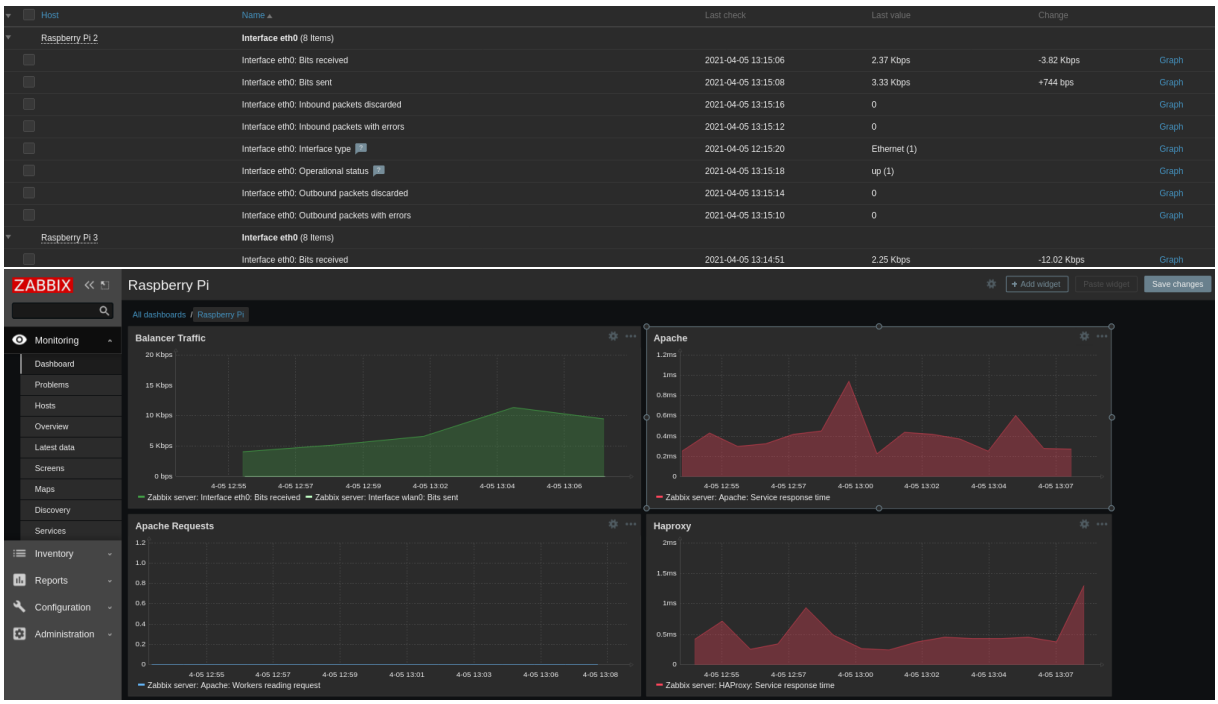
(b) Modifikácia hodnôt makier šablóny

Obr. 6: Postup pridávania nového hosta na monitorovanie

4.2 Logging

Zaznamenávanie každej požiadavky webovému serveru s časovou pečiatkou je cennou pomôckou pri diagnostike problémov v prevádzke poskytovanej služby, na odhalenie bezpečných prienikov, alebo na účely auditu. Podrobnosť výpisov sa obmedzuje najmenšou úrovňou závažnosti, ktorú je žiaduce zapísať. Základné stupne sú *debug, info, warning, error*. Ustálená textová podoba záznamov naprieč rozličnými webservermi zjednodušuje ich analýzu a sumarizáciu existujúcimi všeobecne použiteľnými nástrojmi a posilňuje zachovanie kompatibility pri softvérovej zmene v infraštruktúre.

Zaužívaným východzím formátom logov je **Common Log Format**, uplatňujúci sa už u prvých webových serverov NCSA HTTPd, ktorého koncepty napomohol rozšíriť do povedomia Apache projekt. Záznam o prístupe (*Access Log*) je uložený v súbore ako jeden riadok oddelený medzerami pozostávajúc zo siedmych atribútov: doménové meno alebo adresa klienta, identifikácia vzdialeného užívateľa podľa RFC 1413 - najčastejšie „-“, autentizované používateľské meno, dátum a čas požiadavky s časovou zónou celé v



Obr. 7: Načítané a vizualizované metriky zozbierané od agentov

hranatých zátvorkách, prvý riadok HTTP požiadavky v úvodzovkách, HTTP stavový kód odpovede, dĺžka tela prenášaného dokumentu [32].

Kód 23: Ukážka záznamu v Common Log Format

```
192.168.0.2 - - [05/Apr/2021 13:45:20 +0200] "GET␣/␣HTTP/1.1" 200 1254
```

Štruktúra záznamov sa zvykne podľa potreby rozširovať vlastnými definíciami cez formátovacie špecifikátory a premenné poskytované v implementácii webového servera. Bežne sa Common Log zvykne dopĺňať o pole *Referer*, čiže stránku, z ktorej klient hlási, že bol odkázaný; *User agent* predstavuje webový prehliadač a platformu návštevníka webstránky a na koniec sa pridávajú v úvodzovkách *Cookies* vo forme „kľúč = hodnota“ oddelené bodkočiarkami. Takto rozšírený formát sa nazýva **Combined Log Format**.

Kód 24: Vlastný formát pre access log na webovom serveri Apache

```
LogFormat "%h␣%l␣%u␣%t␣\"%r\"␣%>s␣%I␣%O␣\"%{Referer}i\"␣\"%{User-Agent}i\"" custom
CustomLog ${APACHE_LOG_DIR}/access.log custom
```

Na odhalenie slabých článkov a úzkych hrdiel sa hodí softvérom vyvažovania záťaže sledovať metriky o sieťovej premávke týkajúce sa trvania a rozsahu správ, charakteristiky požiadaviek a odpovedí v podobe hlavičiek, stavových kódov a ich obsahu so zdôvodnením chýb vzniknutých pri odbavovaní [33]. Zaujímavé sú taktiež plánovacie rozhodnutia load balancera ohľadom výberu serverov určených za spracovateľov. NGINX dokáže pre spojenie vyvažované na transportnej vrstve zaznamenať okrem stálych parametrov, ako sú adresa pôvodcu a časová pečiatka, trvanie nadviazanej relácie, v sekundách a milisekundovým rozlíšením, spolu s objemom prenesených bytov oboma smermi. Vyvažovanie

na aplikačnej vrstve vie zaznamenať adresu cieľového servera a časy v sekundách strávené pripájaním, čítaním hlavičiek upstream serverom a príjmom návratovej odpovede.

Kód 25: NGINX: vlastná štruktúra TCP a HTTP Log

```
log_format fmt '$remote_addr␣[$time_local]␣$protocol␣$status␣$bytes_sent␣$bytes_received␣
    $session_time';
log_format fmt '$remote_addr␣[$time_local]␣"$request"␣$status␣$upstream_addr␣
    $upstream_bytes_received␣$upstream_connect_time␣$upstream_header_time␣
    $upstream_response_time';
access_log /var/log/nginx/access.log fmt;
```

Kód 26: NGINX záznamy z TCP a HTTP logov s vynechanou časovou pečiatkou

```
192.168.0.5 TCP 200 465 308 20.960
192.168.0.5 "GET␣/␣HTTP/1.1" 200 192.168.0.3:80 1294 0.000 0.040 0.040
```

HAProxy predvolene zapisuje všetky údaje potrebné na spätné trasovanie požiadaviek. S milisekundovou precíznosťou vidíme, z akého frontendu na ktorý server z backendu bolo spojenie smerované. Časovače pre HTTP mód vo formáte `TR/Tw/Tc/Tr/Ta` sledujú postupne celkový čas obratu, čakanie vo fronte, nadväzovanie TCP spojenia, čas odozvy servera a aktívny čas HTTP požiadavky [33]. Nasledujú polia pre HTTP stavový kód, prenesené bajty a skratky pre neštandardne skončenú reláciu. Päť počítadiel oddelených lomkami hovorí o súbežných spojeniach spracúvaných HAProxy, keď bola relácia zaznamenaná. Dozvieme sa celkový počet spojení, z toho aktuálne smerované cez frontend, cez backend, aktívne na serveri alebo znovu opakovaných pokusov o vytvorenie relácie. Pred typom HTTP požiadavky sa nachádzajú dĺžky front čakajúcich požiadaviek.

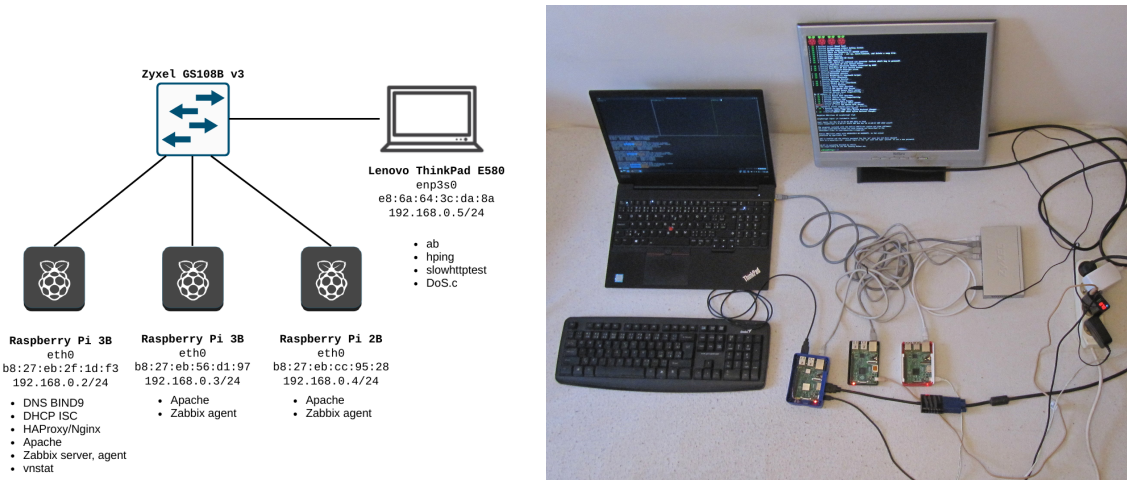
Kód 27: HAProxy: predvolený formát záznamov v logoch

```
haproxy[2550]: 192.168.0.5:45152 [28/Mar/2021:18:07:58.798]
    web webservers/A 0/0/5/23/29 200 2862 - - ---- 9/9/8/8/0 0/0 "GET␣/␣HTTP/1.1"
```

5 Simulácie útokov a záťažové testy

Úlohou softvérových riešení na symetrizáciu sieťovej premávky ako sú *HAProxy* a *NGINX* je zvýšiť priepustnosť spracovania dopytov na webovú aplikáciu a zaistiť zlepšenie dostupnosti pod náporom oproti samostatne stojacemu webovému serveru akým je *Apache*.

Experimentálne porovnáme silu útokov odoprenia služby dosiahnuteľnú z bežného osobného počítača nástrojom `hping` a zanalyzujeme vlastnú implementáciu škodlivého kódu v jazyku C (`DoS.c`), ktorý dokáže realizovať záplavové útoky s UDP a TCP SYN paketmi alebo útoky aplikačnej úrovne Slowloris a podvrhnúť zdrojovú IP adresu. Záťažovými testami *ApacheBench* (`ab`) na vlastnej HTML stránke preskúmame účinnosť a plánovanie vyvažovania záťaže s rastúcim počtom fyzických uzlov pri výbere algoritmu Round Robin alebo Least Connections naprieč riešeniami HAProxy a NGINX, na sieťových vrstvách L4 aj L7. Ďalej sa pozrieme na priebeh útoku Slowloris s nástrojom `slowhttptest` na webový server nezabezpečený technikou obmedzovania rýchlosti relácií. Zámerom je získať ucelený pohľad na prevedenie najbežnejších typov DoS útokov z pohľadu útočníka a poskytnúť ukážky správania sa load balancera pod nadmernou intenzitou požiadaviek, čím zdôvodníme potrebu vyvažovania záťaže aspoň pre kritické aplikácie.



(a) Logická topológia (b) Fyzická topológia

Obr. 8: Topológia izolovanej siete na experimenty

Na priblíženie sa reálnemu prostrediu, kde býva load balancer dedikovaným zariadením, a zároveň pre čo najvyššiu mieru hermetickosti, vytvoríme na účely pokusov izolovanú počítačovú sieť. Nutno poznamenať, že každý použitý počítač disponuje dvoma sieťovými rozhraniami, a síce pre Ethernet, ale tiež Wifi adaptérom s pripojením na domácu sieť s internetovým pripojením pre doplnkový softvérový manažment. Spúšťanie útokov a testov uskutočníme z notebooku *Lenovo ThinkPad E580* s Manjaro Linux spojeného cez 8-portový gigabitový prepínač *Zyxel* ku 3 počítačom *Raspberry Pi* s distribúciou odvodenou od Debian Buster verzie z 11.1.2021. Medzi útočníkom a serverovou farmou dochádza priamo ku komunikácii iba cez konkrétne *Raspberry Pi 3B* figurujúce ako vyvažovač zá-

ťaže a hlavný uzol skupiny. Konfigurácia prebieha cez protokol SSH. Na centralizovanú distribúciu IP adries podľa Obr. 8a slúži *ISC DHCP server* a lokálne doménové meno pre webovú stránku sprostredkúva *BIND9 DNS server*, ktorým sme overovali techniku „DNS load balancing“. Na Obr. 8b sa ešte objavuje predlžovacia šnúra na elektrické napájanie a externý monitor na odlaďovanie a vykonávanie meraní na serveroch počas zahltenia siete.

Pôvodne sa na extenzívne porovnanie algoritmov vyvažovania záťaže pripravovala aj skupina premenlivého množstva *Docker* kontainerov webových serverov orchestrovaných cez *Docker Compose*. Neskôr popísané záťažové testy sú uskutočniteľné pre väčší počet dostupných inštancii uzlov cez takto definovanú virtuálnu sieť, pričom sa nevezmú do úvahy okolnosti spojené s fyzickými spojmi:

Kód 28: Vyvažovanie Apache inštancii cez HAProxy v docker-compose.yml

```
version: '3'
services:
    balancer:
        image: 'haproxy:latest'
        ports:
          - 80:80
        volumes:
          - ${PWD}/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
    web-1:
        image: 'php:7.3-apache'
        volumes:
            - ./website/:/var/www/html/
```

5.1 Analýza prevedenia DoS útokov

V skonštruovanej počítačovej sieti sme nástrojom `hping` odmerali prietok paketov spôsobený bežným laptopom meraný cez monitor sieťovej premávky `vnstat` spustením na serveri s adresou 192.168.0.2. HPing umožňuje prepínačmi príkazového riadka zostaviť pakety so žiadaným obsahom hlavičky pre protokoly ICMP, UDP a TCP a prepustiť ich z rozhranie útočníka v čo najväčšom množstve vyvolajúc záplavu.

Kód 29: Príkazy na spustenie záplavových útokov a meranie

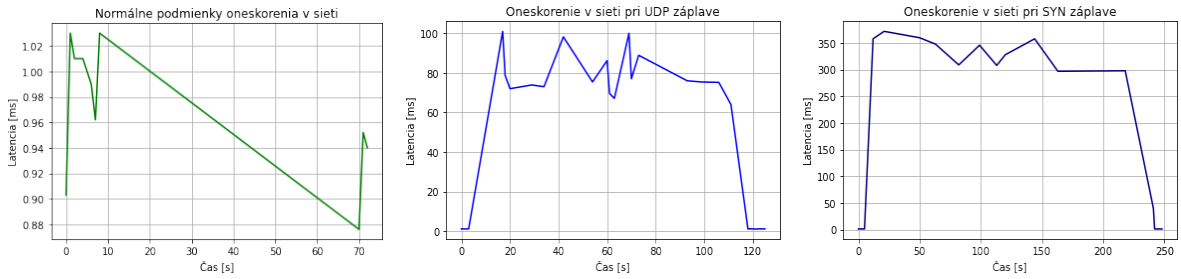
```
hping 192.168.0.2 --udp --flood                    # UDP záplava na port 0
hping 192.168.0.2 --icmp --flood                   # ICMP Echo request záplava
hping 192.168.0.2 --syn --flood --destport 80   # TCP SYN záplava na port 80
hping 192.168.0.2 --syn --flood --spoof 192.168.0.10 # IP Spoofing
vnstat --traffic                                   # Meranie priemernej premávky
```

Z tabuľky č.1 je patrné, že sa podarilo z celkovej teoretickej kapacity linky 100 Mbit/s, ohraničenej rýchlosťou sieťovej karty na Raspberry Pi, zahltiť najviac polovicu dátovým prietokom do 55 Mbit/s spôsobeným 147 tisíc UDP paketmi za sekundu bez payloadu, každého v rámci dĺžky 42 bajtov. Navzdory najväčšiemu prítoku spôsobil UDP flood iba zníženie dostupnosti, kedy do 15 sekúnd sme sa zaručene dočkali odpovede od webového servera. Správy sa posielali vždy na rovnaký port a Linux Netfilter limitoval premennou `icmp_ratelimit` rozostup odpovedí ICMP Destination unreachable na minimálne 1 sekundu, preto nepozorujeme pomalšiu odozvu. Úplné zneprístupnenie služby v priebehu

trvania útoku bolo dosiahnuté cez SYN Flood, kedy na server prišlo okolo 122 tisíc TCP segmentov za sekundu s dĺžkou 54 bajtov zaberajúc 45 Mbit/s, ale na druhej stane bola pri nadviazaní relácií vyvolaný spätný prúd SYN+ACK segmentov s objemom 2,9 Mbit/s, ktorý zamestnal časť systémových prostriedkov a dočasne sa zabrali voľné slotov v tabuľke spojení. Dôkaz, že hrubá sila nepredstavuje jedinú použiteľnú metódu odoprenia služby uvádzame prenosové rýchlosti útoku Slowloris (popísaného neskôr), ktorý je z pohľadu útočníka rovnako schopný ako SYN flood za nižšej generovanej obojsmernej komunikácie činiacej 140 kbit/s, čim sa výrazne znižujú šance na rýchle detegovanie a rozpoznanie tejto škodlivej aktivity. Záplavy zvýšili oneskorenie ICMP Echo správ posielaných od klientom na server z 1 ms, na priemerne 80 ms pri UDP flood a až na 320 ms pri SYN Flood, ako je čitateľné z grafov na Obr.9.

	RX [Mbit/s]	TX [Mbit/s]	RX [pakety/s]	TX [pakety/s]
UDP Flood	54,22	0,001	147345	1
ICMP Flood	45,07	2,21	122483	5534
SYN Flood	45,09	2,87	122539	5118
Slowloris	0,14	0,13	175	170

Tabuľka 1: Dosiahnutá sila útokov s hping odmerané spriemerovaním 5 sekúnd premávky s vnstat na sieťovej linke s reálnou šírkou pásma 94.2 Mbit/s odmeranou cez iperf



Obr. 9: Latencia v sieti meraná príkazom: ping -D -U 192.168.0.2

Na overenie technickej priamočiarosti spáchania DoS útokov, a relatívnej jednoduchosti implementácie malvéru takéto druhu, sme sa pokúsili replikovať na základe dostupného zdrojového kódu pre `hping`[11] namerané charakteristiky generovanej sieťovej premávky s vlastným skriptom. Keďže sme potrebovali vlastnoručne naplniť obsah hlavičiek paketov na sieťovej a transportnej vrstve bolo potrebné siahnuť po RAW socketoch, ktorými dokážeme posielať v podstate ľubovoľné zoskupenie bajtov. Následne cez `setsockopt` vyžiadame príznakom `IP_HDRINCL`, aby IP hlavička nebola doplnená automaticky operačným systémom.

Kód 30: Otvorenie RAW Socket bez dopĺňania IP hlavičky

[11] `https://github.com/antirez/hping`

```
#include <sys/socket.h>
int s = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
setsockopt(s, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one));
```

Samotný UDP datagram pre UDP Flood alebo TCP SYN segment pre SYN Flood je reprezentovaný ako pole bajtov, ktoré pretypovaním na ukazovateľ na knižničnú štruktúru `iphdr` pre IP hlavičku, `udphdr` pre UDP hlavičku alebo `tcphdr` pre TCP hlavičku naplníme požadovanými hodnotami ako sú náhodný zdrojový, cieľový port a určené IP adresy a tiež vložíme do povinných polí prípustné hodnoty. Zhotovený paket v premennej `buffer` pošleme obeti cez vyhradený súborový deskriptor socketu funkciou `sendto()`.

Kód 31: Poslanie UDP datagramov cez RAW Socket na Linuxe

```
#include <netinet/ip.h>
#include <netinet/udp.h>

char buffer[64];
struct iphdr *ip = buffer;
struct udphdr *udp = buffer + sizeof(struct iphdr);
// Naplnenie štruktúr ...
sendto(s, buffer, ip->tot_len, 0, (struct sockaddr *)&sin, sizeof(sin));
```

Dôvodom nezúžitkovania možnosti `SOCK_DGRAM` pri vytváraní UDP socketu a tým pádom nutnosť manuálneho dopĺňania protokolových hlavičiek je vo flexibilite dovoľujúcej podvrhnutie zdrojovej IP adresy, čím presmerujeme spätnú väzbu odpovedí servera mimo útočníka. Ukážeme realizáciu techniky IP Spoofing s generovaním náhodných adries z podsiete, ktorú určíme argumentom `net` pre funkciu `ip_random()`. Vyžaduje sa 32-bitové Big-endian číslo predstavujúce IPv4 adresu získateľné s `inet_addr()`. Argument `cidr` označuje CIDR prefix, z ktorého sa vytvorí sieťová maska. Invertovaná maska sa použije pri náhodnom generovaní bitov hosta a nakoniec sa celá adresa poskladá dohromady cez bitové OR operátory.

Kód 32: Vytvorenie náhodnej IP adresy obmedzenej podsieťou

```
in_addr_t ip_random(in_addr_t net, int cidr)
{
    in_addr_t net_mask = (~0 << (32 - cidr));
    in_addr_t host_mask = ~net_mask;
    in_addr_t target_host = 0;
    if (host_mask != 0) target_host = rand() % host_mask;
    return htonl((ntohl(net) & net_mask) | target_host);
}
```

Keďže po uvedenom zásahu do zdrojovej IP adresy nedokáže systémový protokolový zásobník priradiť MAC adresu rozhrania pôvodcu, pretože nemusí ísť dokonca ani o adresu dohľadateľnú cez protokol ARP, je potrebné Ethernet rozhranie explicitne pomenovať. Po

skopírovaní názvu do štruktúry `ifreq` vyhľadáme index cez systémové volanie `ioctl` a cez `setsockopt` a vlajku `SO_BINDTODEVICE` napojíme socket na príslušné linkové rozhranie.

```
strcpy(ifr.ifr_name, "enp3s0");
ioctl(s, SIOCGIFINDEX, &ifr);
setsockopt(s, SOL_SOCKET, SO_BINDTODEVICE, &ifr);
```

Narozdiel od záplavových útokov, kedy je myšlienkou zapratať sieť obete, spočíva útok Slowloris v obsadení všetkých nadviazateľných HTTP relácii s webovým serverom neukončením žiadnej požiadavky dvojicou nových riadkov ako ukladá protokol HTTP, ale pomalým posielaním stále ďalších podvodných hlavičiek. Vypršaním časovača na strane webservera sú spojenia pomaly uzatvárané s chybovým stavovým kódov, preto je nutné ich pravidelne obnovovať. Tentoraz nevieme podsunúť neplatnú adresu zdroja, pretože agent musí kompletne nadviazať TCP spojenie.

Kód 33: Odoslanie HTTP GET požiadavky s náhodným URL query parametrom

```
int s = socket(AF_INET, SOCK_STREAM, 0);
connect(s, (struct sockaddr *)&victim, sizeof(victim))
snprintf(buffer, LEN, "GET␣/?%d␣HTTP/1.1\r\n", rand() % 1000);
send(s, buffer, strlen(buffer), 0);
```

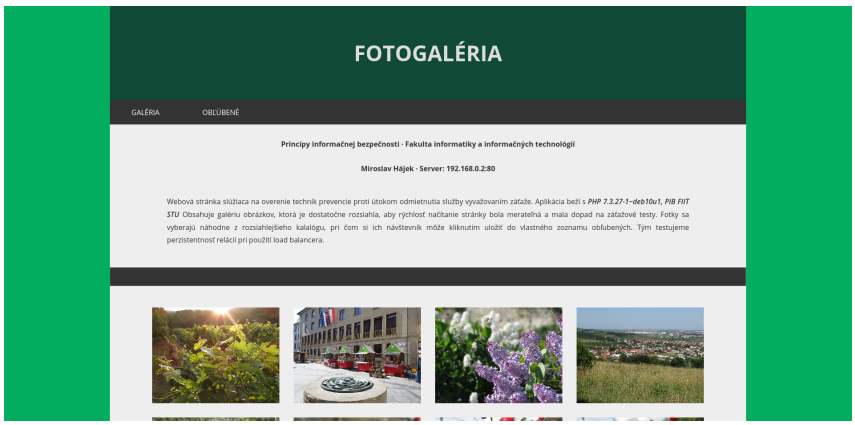
Kód 34: Udržiavanie živosti relácie pravidelným posielaním nových dát

```
for (int i = 0; i < CNT; i++) {
    snprintf(buffer, BUF_LEN, "X-a:␣%d\r\n", randint(1, 50000));
    send(sockets[i], buffer, strlen(buffer), 0); }
sleep(10);
```

5.2 Slowhttptest Slowloris útok

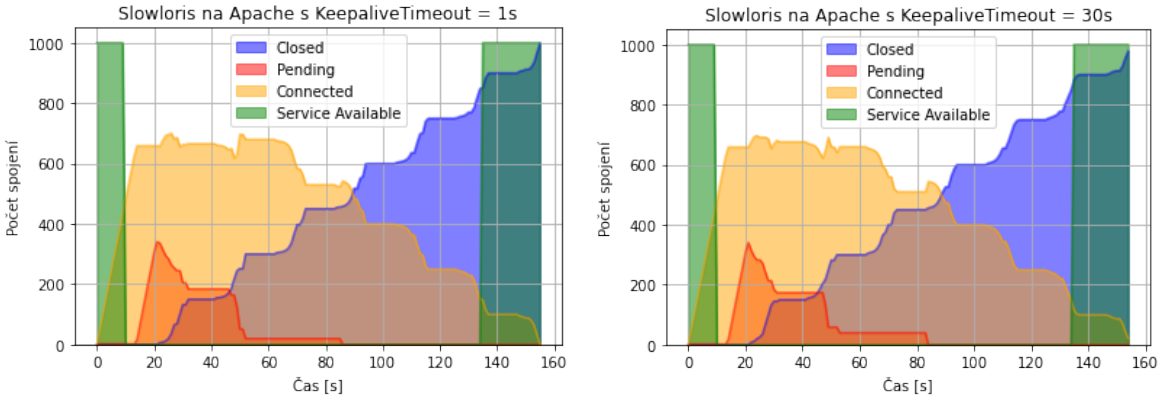
Dosiaľ sme sa zaoberali len vplyvom priepustnosti sieťového spojenia a obmedzeniami prostriedkov jedného servera. Teraz budeme zisťovať účinnosť horizotálneho škálovania inštancii statickej webovej stránky fotogalérie (Obr.10 na webserveri Apache prístupnej cez vyvažovače záťaže HAProxy a NGINX, ktoré vzájomne porovnáme. Sústredíme sa na priebehy útoku Slowloris pre infraštruktúru v rozdielnych konfiguráciach. Ako základný test spustíme nástroj `slowhttptest` bez akéhokoľvek load balancingu a pripojíme sa priamo na Apache, ktorý nám umožní meniť hodnoty keepalive časovačov a počtu spojení so zdanlivo potenciálnymi vplyvom dostupnosti služby.

Slowloris sme spustili pre cieľových 1000 spojení pridávajúc sa postupne tempom 50 nových relácií za sekundu a následné údaje sa posielali s 10 sekundovými prestávkami. Predpoklad, že zmenou premenných `Timeout`, `KeepAlive`, `MaxKeepAliveRequests` alebo `KeepAliveTimeout` výrazne ovplyvníme dostupnosť webového servera boli mylné. Ilustrácia priebehu experimentu je vynesená v grafoch na Obr.11 pre keepalive timeout 1



Obr. 10: HTML stránka fotogalérie na testovanie útokov

a 30 sekúnd. Nad 480 súbežných pripojeniach sa stal server nedostupný, tým že nedokázal vrátiť klientovi obsah webstránky. Najviac Apache zvládol nadviazanie 700 HTTP relácií zároveň, ostatné museli čakať na uvoľnenie slotov. Po 20 sekundách od začiatku testu začal server postupne zatvárať spojenia a k obnoveniu dostupnosti prišlo až pri 150 zvyšných aktívnych klientoch.

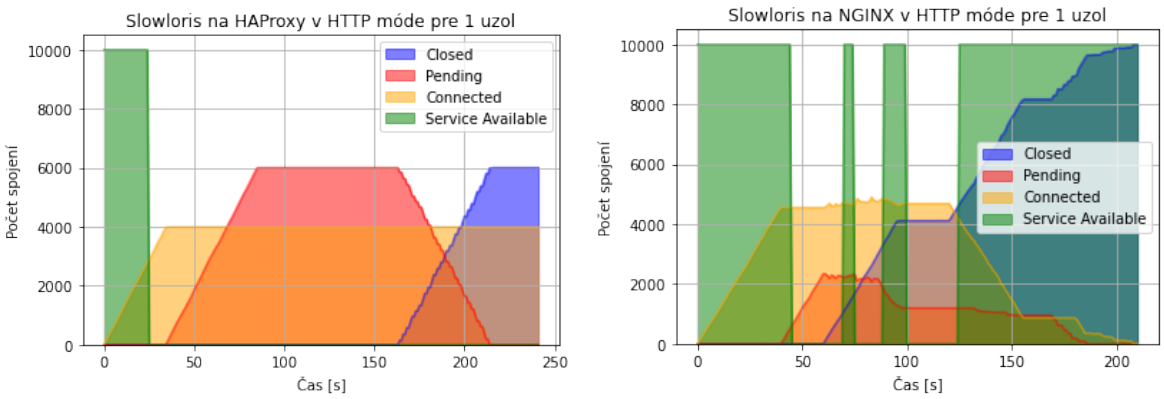


Obr. 11: Reakcia webového servera Apache na útok SlowLoris pri zmenách Keepalive timeout na spojenie. MaxKeepAliveRequests=100, Timeout=300s

```
slowhttptest -H -c 1000 -i 10 -r 50 -l 180 -g -o stat -u http://192.168.0.2/
```

Predradením softvérového vyvažovača záťaže na aplikačnej vrstve webovému serveru sa môže z grafov na Obr.12 zdať, že pri cieľovom počte 10000 spojení sa podarilo rapídne zlepšiť množstvo spojení dopadajúce na zníženie dostupnosti už len na jedinom uzle. HAProxy zvláda 2800 súčasných relácii a NGINX až 4500 spojení. Pri bližšom pohľade do logov však zisťujeme, že všetky spojenia skončili na load balanceri a nikdy neboli prepojené na vzdialený webový server, pretože skončili so stavovým kódom 400 (Bad Request) alebo 408 (Request Timeout). HAProxy signalizuje značkou CR, že klient prerušil reláciu pred odoslaním úplnej požiadavky. Vnútorná infraštruktúra by bola týmto predsa len ochránená pred náporom bezvýznamných HTTP správ.

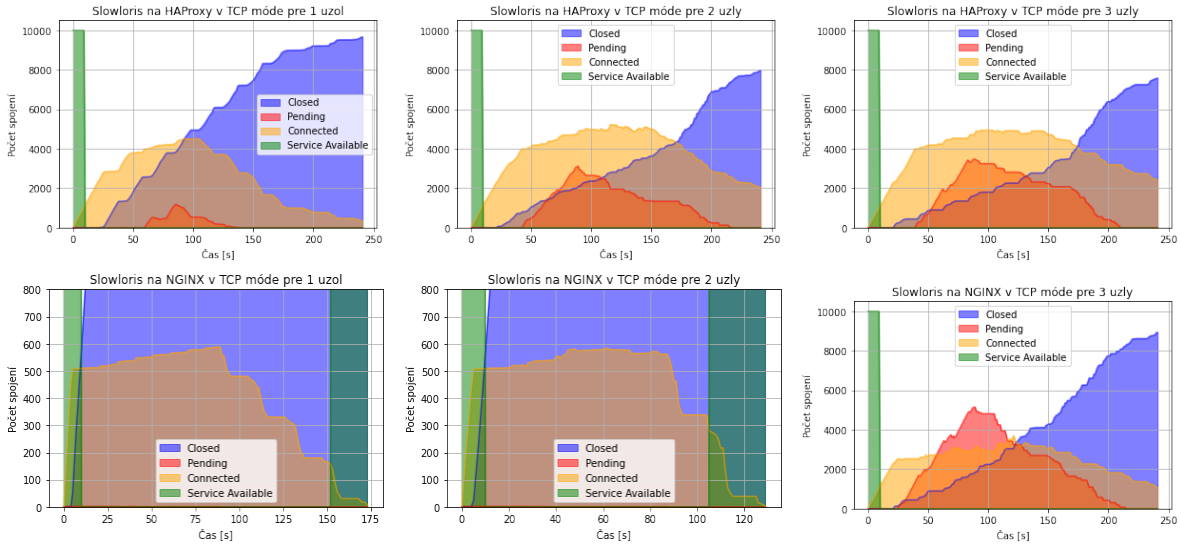
Kód 35: Skrátené záznamy z logov počas Slowloris pre HAProxy a NGINX v HTTP móde



Obr. 12: Porovnanie účinkov útoku Slowloris na počet aktívnych spojení a dostupnosť služby pri L7 load balancingu na HAProxy a NGINX:
`slowhttptest -H -c 10000 -i 10 -r 200 -l 240 -g -o stat -u http://192.168.0.2/`

```
web/<NOSRV> -1/-1/-1/-1/64642 400 187 CR-- 2000/2000/0/0/0 "<BADREQ>"
192.168.0.5  "GET␣/␣HTTP/1.1" 408 - - - - -
```

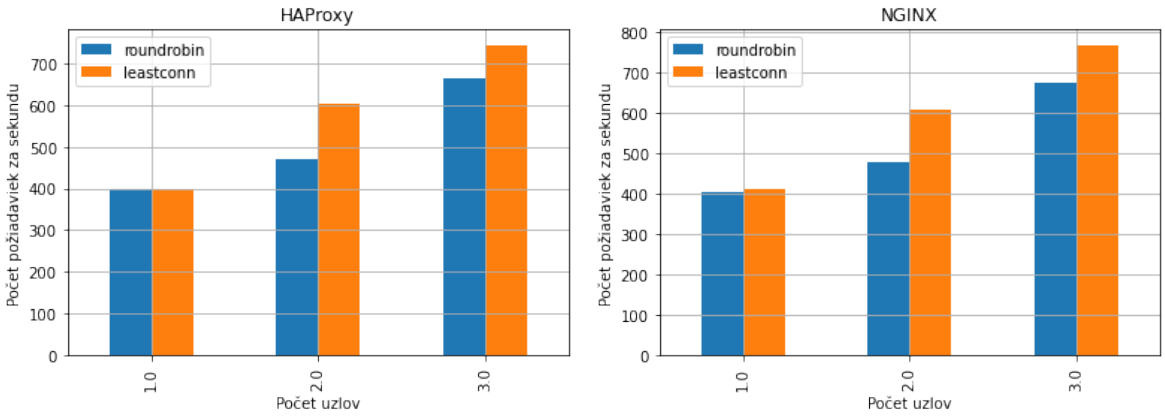
Prepnutím do TCP režimu vyvažovania záťaže nie sú spojenia rušené load balancerom, pretože nemá poňatie čo predstavuje úplnú požiadavku a očakávateľne dosahujeme pri jednom uzle farmy podobných výsledkov ako samostatným webovým serverom s pridanou hodnotou radenia čakajúcich požiadaviek do front. HAProxy klesá kuriózne s rastúcim počtom uzlov rýchlosť terminácie spojení, čo sa dá vysvetliť zvýšením réžie presmerovaním na podradené servery. Horná hranica dostupnosti sa u oboch riešení nachádza v rozmedzí 1033 - 1145 simultánne aktívnych spojení, pretože nastavený `ulimit` obmedzoval počet otvorených súborových deskriptorov vrátane socketov na 1024.



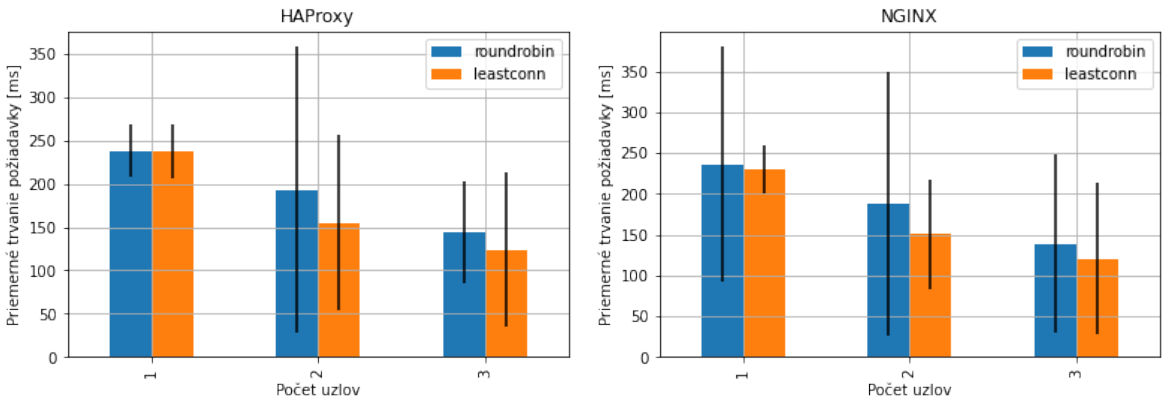
Obr. 13: Porovnanie účinkov Slowloris pri L4 load balancingu na HAProxy a NGINX

5.3 ApacheBench záťažové testy

Na demonštrovaných výsledkoch sa zatiaľ nepreukázal pravý význam vyvažovania záťaže a síce zvýšenie rýchlosti spracovania požiadaviek a pokles časov odozvy stránky pre klientov skrátením doby obratu. V zostavenej izolovanej sieti sme zvyšovali počet backend, resp. upstream uzlov v rozmedzí 1 až 3, z ktorých má HAProxy, resp. NGINX na výber pri vyvažovaní a zároveň sme sledovali dopad aplikovaného algoritmu podporovaných oboma riešeniami: Round Robin a Least Connections. Od notebooku testera (192.168.0.5) sme realizovali viacero obmien záťažových testov ApacheBench (`ab`) v kombináciach celkový počet spojení verzus paralelné relácie: 10/1, 100/10, 1000/100, 10000/1000. Dohromady 48 reportov vypísaných nástrojom `ab` (pre 2 load balancery, 3 uzly, 2 algoritmy, 4 intenzity spojení) sa spracovalo a výsledky agregovali v Jupyter Notebook. Regulárnymi výrazmi sme vyfiltrovali z textových súborov hodnoty na relevantných riadkoch: „Requests per second“, v tabuľke „Connection Times - Total“ stĺpce `mean` a `[+/-sd]`.



Obr. 14: Počet spracovaných požiadaviek za sekundu pre algoritmy vyvažovania záťaže podľa ApacheBench: `ab -n 1000 -c 100 http://192.168.0.2/`



Obr. 15: Priemerné trvanie požiadaviek pre algoritmy vyvažovania záťaže podľa ApacheBench: `ab -n 1000 -c 100 http://192.168.0.2/`

Intencie zlepšenia výkonnosti webovej aplikácie od počtu horizontálne škálovaných uzlov ilustrujú grafy pre 1000/100 spojení na Obr.14 a Obr.15. Počiatočných 400 spojení

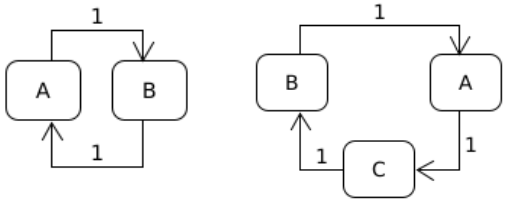
zaznamenalo u oboch softvérových riešení nárast obsluhy. Algoritmom Round Robin (RR) o 18% pridaním druhého uzla a o 67% s tretím uzlom v porovnaní so samostatne stojacim serverom. Least Connections (LC) dosahoval mierne lepšie štatistiky, kde prišlo k navýšeniu počtu spracovaných spojení o 48% s dvoma uzlami a o 86% s troma, čo sa pri vyššom vyťažení služby dá považovať za významné zlepšenie. Ak berieme do úvahy 100000 spojení za podobných podmienok, tak s RR sú pri vyvažovaní medzi tri servere odbavené za 150 sekúnd, ale s LC len za 132 sekúnd. Obe reverzné proxy sa za zrealizovaných meraní prejavujú navzájom veľmi podobne. Rastom počtu spracovaných požiadaviek za sekundu klesá priemerné trvanie na odpovede klientom obdobným spôsobom. Z približne 235 ms v priemere na požiadavku sa dostávame pri troch spolupracujúcich uzloch na 122 - 144 ms podľa použitého algoritmu, čím sa skracuje trvanie úlohy až o 38% - 48%. Vysoká smerodajná odchýlka môže byť zapríčinená variabilným zahltením siete súbežnými požiadavkami počas experimentu. Vyjadrené trendy v trvaní požiadaviek sa preto dajú brať nanajvýš orientačne.

Diferenciu efektívnosti algoritmov RR a LC prisudzujeme väčšej informovanosti LC ohľadom momentálnych trvaní požiadaviek pri návrhu plánovacích rozhodnutí a nízky počet dostupných serverov, čím sa stáva nájdenie najkratšej fronty takmer konštantnou operáciou. Súdiac z prednesenej domnienky musí byť prechodový diagram následnosti alokovaných uzlov pre LC nepredvídateľnejší, avšak permutácia vychádzajúca z RR by mala mať zreteľný trend v poradí usporiadania uzlov.

Po záťažových testoch `ab` sme akumulované záznamy v `haproxy.log` pomocným skriptom rozkúskovali do viacerých súborov s prahovým rozdielom medzi následnými časovými značkami činiacim 1 sekundu. Manuálne sme odstránili nadbytočné oddelené čiastky logov, ktoré obsahovali prevažne stavové správy o reštarte HAProxy medzi zmenami konfigurácie. Súbory s korektným počtom riadkov podľa nastavenú počtu požiadaviek daného zátažového testu sme roztriedili do priečinkov podľa počtu uzlov za proxy a uplatneného algoritmu.

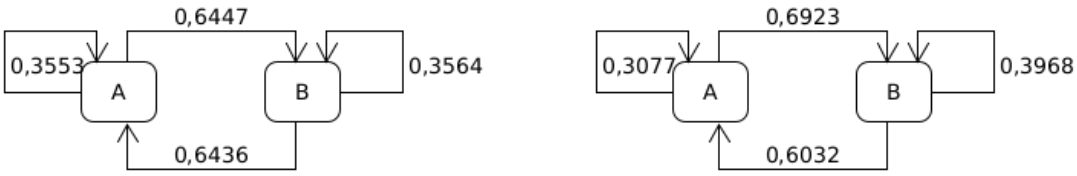
Ďalším Python skriptom sme z medzerou oddelených atribútov loggov vytiahli 8. stĺpec s údajom o obslužnom servery pre konkrétnu požiadavku a zo zachytenej skupiny regulárneho výrazu `^\w+/([A-C])$` sme zostavili permutáciu presmerovaní v rámci vyvažovania záťaže. Do štvorcovej prechodovej matice o rozmeroch $n \times n$, kde n je počet load balancových uzlov, sme umiestnili absolútne početnosti plánovacích prechodov $S_k \to S_{k+1}$, kde S označuje zvolený server v poradí. Na získanie aproximácie pravdepodobností stavových prechodov sme zúžitkovali vlastnosti stochastických matíc, že súčet pravdepodobností v riadku sa rovná jednej. Pozíciu i, j v matici preto z absolútnych početností získame: $S_{i,j} / \sum_{j=1}^{n} S_{i,j}$. Výsledné matice reprezentujeme stavovým diagramom zodpovedajúceho Markovovho reťazca.

Automat na Obr.16 ukazuje, že 10 za sebou príchodzích relácií je striktne po porade vyvažovaných na všetky dostupné serveri u RR aj LC. S väčším počtom súbežných



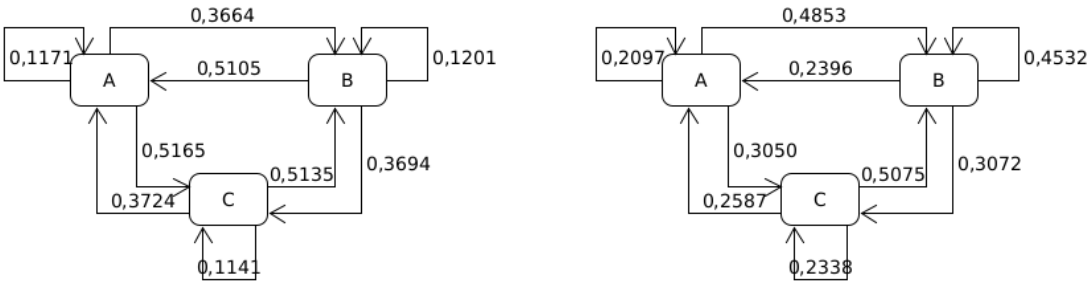
Obr. 16: Markovove reťazce plánovania pridelenia uzlov na obsluhu požiadaviek pri Round Robin a Least Connections: `ab -n 10 -c 1 http://192.168.0.2/`

spojení sa ukazuje odlišná situácia. Dve zariadenia (Obr.17) ešte nezapríčiňujú výrazné vzájomné rozdiely medzi pravdepodobnostnými ohodnoteniami prechodov. Stále majorita požiadaviek (60 - 70%) sa rozdistribuuje prísne sekvenčne, pričom pre zvyšných 30 - 40% následných spojení sa plánovaním vyhradí ten istý server. Pri troch uzloch (Obr.18) sa v reťazci naplno prejavuje asymetria plánovania algoritmom LC. V diagrame pre RR zostáva patrný cyklus $A \rightarrow C \rightarrow B$ s ohodnoteniami hrán 51%. V opačnom chode sa uskutoční z každého uzla okolo 37% požiadaviek na server a najmenej z nich (12%) sa pridelí tam isto. LC sa neprejavuje podobným vzorom správania sa, ani jednoznačnou postupnosťou pri rozložení spojení.



(a) Round Robin (b) Least Connections

Obr. 17: Markovove reťazce plánovania pridelenia uzlov na obsluhu požiadaviek pri 2 koncových uzloch: `ab -n 10000 -c 1000 http://192.168.0.2/`



(a) Round Robin (b) Least Connections

Obr. 18: Markovove reťazce plánovania pridelenia uzlov na obsluhu požiadaviek pri 3 koncových uzloch: `ab -n 10000 -c 1000 http://192.168.0.2/`

6 Záver

Dostupnosť webových aplikácií a generálne informačných systémov sa radí medzi dôležité bezpečnostné aspekty produkčných riešení. Opísali sme rozličné metódy na zabezpečenie dostupnosti nielen vyvažovaním záťaže naprieč vrstvami sieťového modelu RM OSI, vzhľadom na najbežnejšie sa objavujúce úkazy v oblasti útokov odoprenia služby - DDoS. Pozreli sme sa efekt horizontálneho škálovania, ako formu proaktívnej obrany. Venovali sme sa či už enumerácii algoritmov plánovania load balancingu, ale aj praktickým porovnaním konfigurácie a zmeraním výkonnosti L4 a L7 load balancerov HAProxy a NGINX predradených webovému serveru Apache záťažovými testami ApacheBench a SlowHTTPTest. Neopomenuli sme ani monitorovanie (Zabbix) a loggovanie ako stavebné piliere reaktívnych stratégií obrany.

V špecifikovanej predmetnej doméne sa podarilo významne pokryť hlavnú trojicu okruhov: útoky odmietnutia služby, ochrana redundanciou aplikačných serverov a riadenie prístupu na stroje za reverznou proxy. Z projektu sa oproti zámeru upustilo od preskúmania možností obmedzenia inventarizácie infraštruktúry obmedzením veľavravných HTTP hlavičiek. Detailnejšie sme sa nevenovali stavovým spojeniam s cookies cez load balancer. Pre finálny rozsah experimentov sme netestovali nástroje *thc-ssl-dos* a *High Orbit Ion Cannon*. Druhý spomenutý však obsahuje funkcionalitu podobné mnohostrannejšiemu HPing.

Pokusy prebehli výlučne na zostavenej fyzickej počítačovej sieti pozostávajúcej len z troch uzlov. Zistenie limitu množstva serverov, z ktorých by mala farma sprostredkujúca web pozostávať, aby sa ešte dosiahol výhodný pomer náklady verzus výkon považujeme za najzaujímavejšie potenciálne rozšírenie práce.

Literatúra

1. MIR, Suhail; QUADRI, Syed. Information Availability: An Insight into the Most Important Attribute of Information Security. *Journal of Information Security*. 2016, roč. 07, s. 185–194. Dostupné z DOI: 10.4236/jis.2016.73014.
2. *DDoS Handbook: The Ultimate Guide to Everything You Need to Know about DDoS Attacks*. Radware, 2016. https://www.radware.com/getattachment/Security/Research/702/Radware_DDoS_Handbook_2015.pdf.aspx.
3. ABHISHTA, Abhishta; HEESWIJK, Wouter van; JUNGER, Marianne; NIEUWENHUIS, Bart; JOOSTEN, Reinoud. Why would we get attacked? An analysis of attacker's aims behind DDoS attacks. *Journal of Wireless Mobile Networks*. 2020, roč. 11, s. 3–22. Dostupné z DOI: 10.22667/JOWUA.2020.06.30.003.
4. PRASAD, K.Munivara; REDDY, A.Rama Mohan; RAO, K.Venugopal. DoS and DDoS Attacks: Defense, Detection and Traceback Mechanisms -A Survey. *Global Journal of Computer Science and Technology (GJCST)*. 2014, roč. 14. ISSN 0975-4172. https://core.ac.uk/reader/231149430.
5. COHEN, Lawrence; FELSON, Marcus. Social Change and Crime Rate Trends: A Routine Activity Approach. *American Sociological Review*. 1979, roč. 44. Dostupné z DOI: 10.2307/2094589.
6. HERETIK, Anton. Forenzná psychológia. In: 2. vyd. Sasinkova 5, 815 19 Bratislava: Slovenské Pegagogické Nakladateľstvo, 2004, kap. 4, s. 64–71. ISBN 80-10-00341-7.
7. *Infiltrating a Botnet*. Cisco, 2020. https://tools.cisco.com/security/center/resources/infiltrating_botnet.
8. Zákon 300/2005 Z.z. Trestný zákon. 2021. https://www.slov-lex.sk/pravne-predpisy/SK/ZZ/2005/300/.
9. COOKE, Evan; JAHANIAN, Farnam; MCPHERSON, Danny. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. 2005.
10. ALOMARI, Esraa; MANICKAM, Selvakumar; B. GUPTA, B.; KARUPPAYAH, Shankar; ALFARIS, Rafeef. Botnet-based Distributed Denial of Service (DDoS) Attacks on Web Servers: Classification and Art. *International Journal of Computer Applications*. 2012, roč. 49, č. 7, s. 24–32. Dostupné z DOI: 10.5120/7640-0724.
11. MIRKOVIC, Jelena; REIHER, Peter. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*. 2004, roč. 34, č. 2, s. 39–53. Dostupné z DOI: 10.1145/997150.997156.
12. *Ochrana pred útokmi DDoS (Príručka administrátora)*. CSIRT.SK, 2013. https://www.csirt.gov.sk/doc/DDoS_CSIRT.pdf.
13. ENISA Threat Landscape 2020 - Distributed denial of service. *ENISA*. 2020. https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-distributed-denial-of-service.
14. KUMARI, W.; MCPHERSON, D. *Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF)* [Internet Requests for Comments]. RFC Editor, 2009-08. RFC, 5635. RFC Editor. ISSN 2070-1721.
15. *Remotely Triggered Black Hole Filtering - Destination Based and Source Based*. Cisco, 2005. https://www.cisco.com/c/dam/en_us/about/security/intelligence/blackhole.pdf.
16. *Protecting Your Core: Infrastructure Protection Access Control Lists*. Cisco, 2008. https://www.cisco.com/c/en/us/support/docs/ip/access-lists/43920-iacl.html.

17. ASSMANN, Baptiste. Use a Load Balancer as a First Row of Defense Against DDOS. *HAProxy Blog*. 2012. https://www.haproxy.com/blog/use-a-load-balancer-as-a-first-row-of-defense-against-ddos/.

18. *Linux Hardening (Príručka administrátora)*. CSIRT.SK, 2013. https://www.csirt.gov.sk/doc/Hardened_v1.pdf.

19. SOUNDARABAI, Paulsingh; SANDHYA, Rani; SAHAI, Ritesh; K R, Venugopal; PATNAIK, Lalit. COMPARATIVE STUDY ON LOAD BALANCING TECHNIQUES IN DISTRIBUTED SYSTEMS. 2012, roč. 6.

20. *HAProxy documentation*. 2021. http://cbonte.github.io/haproxy-dconv/.

21. *Job Scheduling Algorithms in Linux Virtual Server*. 2021. http://www.linuxvirtualserver.org/docs/scheduling.html.

22. *NGINX Load Balancer Guide*. 2021. https://docs.nginx.com/nginx/admin-guide/load-balancer/.

23. RICHA, Andréa; MITZENMACHER, Michael; SITARAMAN, Ramesh. The Power of Two Random Choices: A Survey of Techniques and Results. 2000. Dostupné z DOI: 10.1007/978-1-4615-0013-1_9.

24. THALER, D.; HOPPS, C. *Multipath Issues in Unicast and Multicast Next-Hop Selection* [Internet Requests for Comments]. RFC Editor, 2000-11. RFC, 2991. RFC Editor. ISSN 2070-1721. http://www.rfc-editor.org/rfc/rfc2991.txt.

25. KNIGHT, S.; WEAVER, D.; WHIPPLE, D.; HINDEN, R.; MITZEL, D.; HUNT, P.; HIGGINSON, P.; SHAND, M.; LINDEM, A. *Virtual Router Redundancy Protocol* [Internet Requests for Comments]. RFC Editor, 1998-04. RFC, 2338. RFC Editor. ISSN 2070-1721. Dostupné tiež z: http://www.rfc-editor.org/rfc/rfc2338.txt.

26. *Keepalived User Guide*. 2021. https://keepalived.readthedocs.io/en/latest/.

27. BOURKE, Tony. Server load balancing. In: O'Reilly, 2001, s. 3–7.

28. LAVOLE, Chad. *Introduction to HAProxy Stick Tables*. 2018. https://www.haproxy.com/blog/introduction-to-haproxy-stick-tables/.

29. RAWDAT, Amir. *Rate Limiting with NGINX and NGINX Plus*. 2017. https://www.nginx.com/blog/rate-limiting-nginx/.

30. *Infrastructure Layouts Involving TLS*. HAProxy, 2021. https://www.haproxy.com/documentation/aloha/12-5/deployment-guides/tls-infrastructure/.

31. *Zabbix Manual*. 2021. https://www.zabbix.com/documentation/current/manual.

32. *Log Files*. 2021. https://httpd.apache.org/docs/trunk/logs.html.

33. MHEDHBI, Moemen. *Introduction to HAProxy Logging*. 2019. https://www.haproxy.com/blog/introduction-to-haproxy-logging/.