VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

MONITOROVACÍ SYSTÉM SÍŤOVÝCH PRVKŮ PRO ZÁTĚŽOVÉ TESTOVÁNÍ

MONITORING SYSTEM OF NETWORK NODES FOR STRESS TESTING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE **Jiří Svozil**

AUTHOR

VEDOUCÍ PRÁCE **Ing. Marek Sikora**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Jiří Svozil ***ID:*** 230676

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Monitorovací systém síťových prvků pro zátěžové testování

POKYNY PRO VYPRACOVÁNÍ:

Cílem této práce je vytvořit systém pro zaznamenávání a vizualizaci vytížení hardwaru a stavu systému síťových prvků, počítačů a serverů. Úkolem práce je navrhnout a vytvořit software pro sběr monitorovaných dat. Dalším úkolem je propojit semestrální úkol s již existující webovou aplikací pro vizualizaci nasbíraných dat v reálném čase, inovovat/rozšířit její funkce a vytvořit metodiku pro spolehlivý přenos. Velký důraz bude kladen na přívětivost, přehlednost, funkčnost a spolehlivost výsledného systému během zátěžového testování.

DOPORUČENÁ LITERATURA:

[1] CAMBIASO, Enrico, Gianluca PAPALEO a Maurizio AIELLO. Taxonomy of Slow DoS Attacks to Web Applications. In: Recent Trends in Computer Networks and Distributed Systems Security. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 2012, s. 195-204. Communications in Computer and Information Science. ISBN 978-3-642-34134-2. Dostupné z: doi:10.1007/978-3-642-34135-9_20

[2] ASSZONYI, Jakub. Webová aplikace pro monitoring stanic v počítačové síti. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Marek Sikora

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Marek Sikora

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

UPOZORNĚNÍ:

ABSTRAKT

Tato práce je zaměřená na návrh, implementaci a testování systému určeného ke sběru, uchovávání a následného přenosu dat o stavu serveru do aplikace pro jejich vizualizaci. Tento systém umožňuje jednoduchým a nenáročným způsobem monitorovat a ukládat základní informace o webovém serveru a dále je posílat pro další využití těchto dat. Práce se věnuje návrhu a tvorbě software pro sběr, ukládání měřených dat a následnému přenesení těchto dat do již existující webové aplikace pro vizualizaci.

KLÍČOVÁ SLOVA

webový server, monitoring, model TCP/IP, webová aplikace, HTTP protokol, databáze, Denial of Service útoky

ABSTRACT

This thesis is focused on the design, implementation and testing of a system specifically designed for the collection, storage and transmission of server data. The aim of this system is to provide users with a simple and efficient means of monitoring and storing essential server information, which can later be transmitted for further use. The work deals with the design and creation of software for collection, transmission and storage of the measured data.

KEYWORDS

web server, server monitoring, model TCP/IP, web application, HTTP protocol, database, Denial of Service attacks

SVOZIL, Jiří. *Monitorovací systém síťových prvků pro zátěžové testování*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 54 s. Bakalářská práce. Vedoucí práce: Ing. Marek Sikora

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Jiří Svozil
VUT ID autora:	230676
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Monitorovací systém síťových prvků pro zátěžové testování

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Marku Sikorovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

S rozvíjejícím vývojem informačních technologií se spousta průmyslových i jiných odvětví přenáší do kyberprostoru. Tím vzniká nová sféra, ve které je potřeba chránit informace a služby. S tímto rostoucím odvětvím vznikají čím dál větší nároky na servery, na které jsou tyto služby přenášeny a je nutné sledovat chování těchto serverů. V dnešní době dochází k výraznému vývoji monitorovacích nástrojů a softwaru, který správcům serverů a sítí umožňuje sledovat chování a správu těchto zařízení. Na základě naměřených dat mohou správci lépe pochopit různé nestandardní situace, které na serverech mohou nastat, a dokonce za použití analytických nástrojů jím do určité míry předcházet. Za nestandardní situace můžeme považovat i kybernetické útoky a právě u jejich odhalování je nutné pracovat s informacemi o chování serverů.

Cílem této práce je vytvořit jednoduchý nástroj, který bude schopný sbírat informace o serverech. Uložená data bude dále zpracovávat. Nástroj bude sloužit ke zaznamenávání vytížení hardwaru a stavu systému síťových prvků, počítačů a serverů. Při tvorbě systému bude kladen velký nárok na jednoduchost a přehlednost.

V první části této práce je popsán model počítačových sítí a princip přenosu dat. Ve druhé části této práce je vysvětlen princip funkčnosti webových serverů a vývoj webových aplikací. Třetí část práce se zaměřuje na monitorování serveru, charakteristiku a základní pojmy. Dále se zabývá popisem již existujícího softwaru určeného pro monitoring sítě a serverů, základním metrikám a způsobu jejich ukládání. Čtvrtá část se věnuje Denial of Service útokům, jejich charakteristikou, dělením a motivací útočníků. Pátá část práce obsahuje samotný návrh, implementaci a testování tvořeného systému.

1 Počítačové sítě a přenos dat

Pojem počítačová síť vznikl v padesátých letech minulého století. Tento pojem slouží jako označení pro počítače, které jsou mezi sebou propojeny a umožňují vzájemnou komunikaci. V dnešní době je nejznámější počítačovou sítí síť Internet, která zažila veliký nástup v devadesátých letech minulého století a dnešní svět je velice složité si bez ní představit. [1]

Aby počítačové sítě mohly správně fungovat a zajišťovat komunikaci mezi stanicemi, bylo nezbytné založit pravidla, pomocí kterých komunikace probíhá. Tato pravidla se nazývají protokoly. Protokoly definují např. syntaxi a synchronizaci komunikace a mohou být hardwarové, softwarové nebo kombinací obou. [2]

Ke zjednodušení vývoje protokolů i síťových zařízení se problematika počítačových sítí rozděluje do vrstev, ve kterých se řeší dílčí problémy. Nejnižší vrstva pracuje s fyzickým hardwarem. Samotné uživatelské aplikace obvykle pracují s vrstvou nejvyšší. Síť internet je založena na sadě protokolů (modelu) s názvem TCP/IP (Transmision Control Protocol/Internet Protocol).[3]

1.1 Model TCP/IP

Transmision Control Protocol/Internet Protocol (dále jen TCP/IP) je sada komunikačních protokolů zajišťujících přenos dat mezi zařízeními v síti internet. Název je odvozen z dvou hlavních protokolů. Transmision Control Protocol (protokol na 3. vrstvě) a Internet Protocol (protokol na 2. vrstvě). Sada se skládá ze čtyř vrstev. Model vyplynul z výzkumu a vývoje Defense Advanced Research Project Agency na konci 60. let minulého století v USA. [4]

Každá vrstva má tzv. datagram. Datagram je složený z hlavičky (header), těla (body) a v některých případech i patičky (footer). Hlavička obsahuje informace jako je odesílatel, příjemce, případně další řídící informace. Tělo obsahuje přenášená data a obvykle nese datagram nadcházející vrstvy. [5]

Vrstva síťového rozhraní

Vrstva síťového rozhraní (také nazývána jako linková vrstva nebo fyzická vrstva) zpracovává fyzické části odesílání a přijímání dat např. pomocí bezdrátové sítě nebo ethernetového či optického kabelu. Datagram vrstvy síťového rozhraní se nazývá rámec. Typickým příkladem pro tuto vrstvu je standart Ethernet. Mezi síťová zařízení na této vrstvě patří např. switch (přepínač), repeater (opakovač) a hub (rozbočovač).

Síťová vrstva

Síťová vrstva (network layer) se používá u účastníků sítě, mezi kterými neexistuje přímé spojení (nejsou si navzájem sousedy). Datagramem internetové vrstvy je tzv. packet. Tyto packety se pohybují po síti od odesílatele k příjemci přes tzv. uzly (stanice, pro které packet není určený), bez ohledu na přenosové médium díky předchozí vrstvě. Protokol použitý na síťové vrstvě u modelu TCP/IP se nazývá Internet Protocol (IP). V dnešní době se používá IPv4 a IPv6. Mezi hlavní informace v hlavičce IP paketu patří zdrojová (source) a cílová (destination) adresa. Díky IP adresám uzly ví, kam packety směrovat. IPv4 adresa je 32 bitové číslo, specifikující dané zařízení. Zařízení operující na této vrstvě se nazývají routery (směrovače). [6, 7]

Transportní vrstva

Transportní vrstva (transport layer) je určena k doručení dat specifickému aplikačnímu procesu na koncové stanici. Jinými slovy vrstva umožňuje adresovat aplikace. Transportní vrstva se nachází nad síťovou vrstvou, která se stará o doručení dat přes síť ke konkrétní stanici. Komunikace je koncová (end-to-end) a některé protokoly poskytují např. detekci a opravu chyb. Adresace probíhá pomocí tzn. portů, které identifikují jaký datagram patří jakému procesu na koncové stanici. Do hlavičky datagramu transportní vrstvy patří cílová a zdrojová adresa a čísla portů.

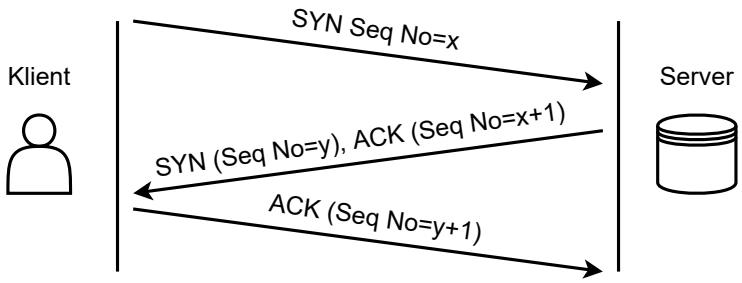
Mezi používané protokoly na transportní vrstvě patří TCP (Transmision Control Protocol) a UDP (User Datagram Protocol). [8]

UDP je protokolem bez zajištění záruky doručení na rozdíl od TCP. Protokol UDP je tedy vhodnější pro použití u aplikací, které vyžadují jednoduchost, rychlou odezvu nebo tam, kde občasná ztráta dat může být akceptována. UDP se také využívá u aplikací se systémem challenge-response (výzva-odpověď). Mezi použití UDP patří např. přenos multimédií, digitální televize (IPTV), VoIP (Voice over Internet Protocol) nebo u některeých online her. [9]

TCP se řadí mezi protokoly se zajištěním spojení. Protokol zajišťuje navázání spojení a k přenosu dat dochází až po navázání spojení. Po správném a bezchybném přenosu dochází k potvrzení úspěšnosti. V případě chyb se přenos opakuje. Díky těmto a dalším operacím dochází k tzv. řízení přenosu, který je typický pro TCP protokol [10]. Navazování spojení u TCP (Three way handshake [11]) je nastíněný na obrázku 1.1 a probíhá následujícím způsobem:

a) Navázání spojení mezi klientem a serverem pomocí SYN paketu.
b) Server příjme SYN a odpoví klientovi SYN/ACK paketem.
c) Klient příjme SYN/ACK paket a odešle třetí ACK paket pro potvrzení, že je zahájeno TCP spojení.

Tohoto procesu lze využít jako zranitelnost pro Denial of Service útoky.



Obr. 1.1: Princip Three way handshake

Aplikační vrstva

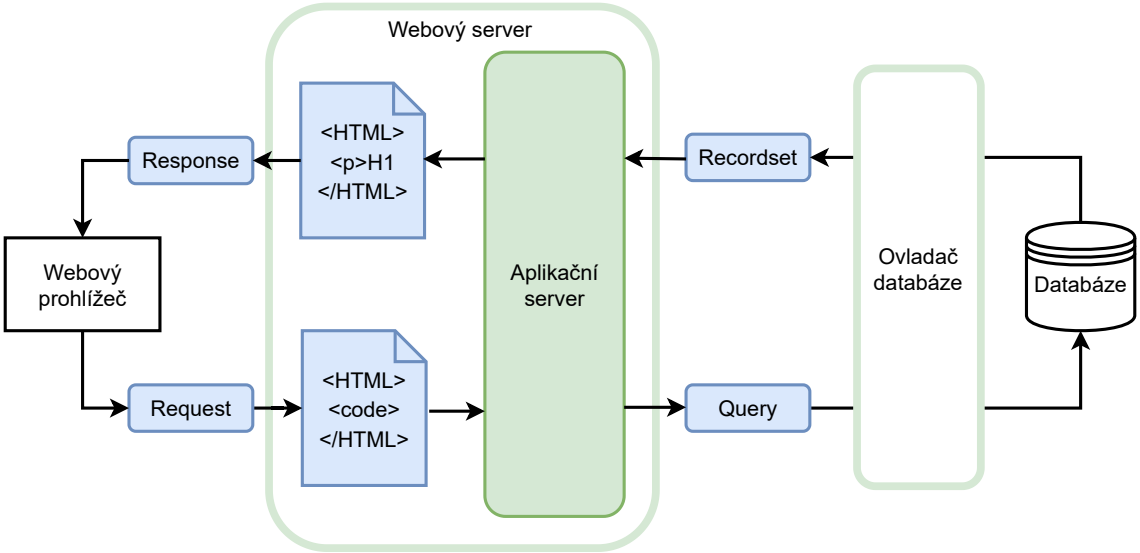
Aplikační vrstva zahrnuje funkce konkrétních aplikačních procesů, proto se funkce této vrstvy liší podle aplikace. Vrstva specifikuje protokoly použité v nižších vrstvách, v transportní vrstvě využívá TCP nebo UDP. Existují i protokoly, které využívají kombinaci těchto protokolů (DNS). Mezi nejvíce používané protokoly na této vrstvě patří: Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP) a systém doménových jmen (DNS). [12]

2 Webové služby a vývoj webových aplikací

Tato kapitola je věnována vymezení pojmu webový server, charakteristice HTTP protokolu, základnímu popisu programu Apache. Dále jsou zde popsány metody a nástroje pro vytváření webových aplikací.

2.1 Webové servery

Termín webový server může představovat software, hardware nebo obojí společně. Webový server je po hardwarové stránce počítač, na kterém běží specifický software (webového serveru) a na kterém jsou uloženy statické části webového serveru jako např. obrázky, texty, HTML (HyperText Markup Language) a CSS (Cascading Style Sheets) soubory nebo jiné. Po softwarové stránce je webový server počítačový program, který komunikuje s klienty prostřednictvím počítačové sítě, nejčastěji pomocí HTTP protokolu. Program na vyžádání zasílá klientovi požadovaný obsah (text, obrázky atd.). Stanice klienta za pomocí internetového prohlížeče tento obsah zobrazí uživateli. Webový server může být buď statický nebo dynamický. Statický webový server zasílá klientovi soubory tak jak jsou (bez-úpravy). Oproti tomu dynamický webový server obsahuje navíc aplikační server (aplikaci) a databázi. Tento aplikační server přidává ke statickým souborům i dynamickou část. V dnešní době se statické webové servery už téměř nevyskytují a to z důvodu dynamičtější a efektivnější práci s daty. Ukázka schématu, jak může vypadat webový server je na obrázku 2.1. [13, 14]



Obr. 2.1: Ukázka schématu webového a aplikačního serveru s databází

Protokol HTTP a HTTPS

Hypertext Transfer Protocol (HTTP) je aplikační protokol sloužící pro komunikaci mezi klientem a webovým serverem. HTTP využívá protokol TCP většinou na portu 80 (na straně serveru). Protokol přenáší hypertextové dokumenty (např. HTML). HTTP podporuje různé způsoby (dotazovací metody), podle kterých klient specifikuje svoje požadavky na webový server. Metod existuje celá řada. Nejpoužívanější jsou: GET, POST, HEAD, PUT, OPTIONS. [15]

Datagram protokolu HTTP se skládá z hlavičky (header) a těla (body). Hlavička může být buď s žádostí (request) nebo s odpovědí (response). V hlavičce odesílatele mohou být informace jako: verze protokolu, použitá metoda (GET), Host (většinou adresa služby), informace o internetovém prohlížeči a operačním systému, jazyk, znaková sada a další (viz obrázek 2.2). V hlavičce odpovědi můžeme najít např. čas, kdy byla odpověď odeslána, typ webového serveru, znaková sada atd. Součástí hlavičky odpovědi je i tzv. stavový kód, který klientovi poskytne bližší informaci o tom, jak byl požadavek zpracován. Jestli např. kladně, záporně popřípadě jestli nedošlo k chybě. [16, 17]

HTTP stavové kódy se dělí do pěti skupin:

- 1XX Information – Informační stavové kódy jsou určeny např. k informování klienta o změněně protokolu (101 Switching protocol)
- 2XX Success – Úspěšné vykonání HTTP požadavku (např. 200 OK)
- 3XX Redirect
- 4XX Client Error – Stavové kódy začínající na 4XX značí chyby na straně klienta (typicky kód 404 Not Found - Požadovaný dokument nebyl nalezen)
- 5XX Server Error – Chyby na straně serveru. Např. verze HTTP protokolu uvedená v požadavku není serverem podporována (505).

```
Hlavička žádosti

GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;…) Firefox/3.5.5
Accept: text/html,…,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
```

```
Hlavička odpovědi

HTTP/1.1 403 Forbidden
Server: Apache
Content-Type: text/html; charset=UTF-8
Date: Mon, 07 Nov 2022 04:36:25 GMT
Keep-Alive: timeout=5, max=1000
Connection: Keep-Alive
Age: 3722
Content-Length: 210

<!DOCTYPE html PUBLIC „-//IETF//DTD HTML 2.0 //EN">
```

Obr. 2.2: HTTP Hlavičky

Od svého vzniku protokol HTTP prošel mnohými úpravami a bylo publikováno několik verzí (HTTP/1.0, HTTP/1.1, HTTP/2 a HTTP/3). Mezi hlavní problémy verze 1.0 patří potřeba navazovat a ukončovat TCP spojení s každou HTTP žádostí

(a tím provádět Three way handshake). Tento způsob je časově náročný a nepraktický pro moderní webové aplikace. Ve verzi 1.1 vývojáři přišli s řešením tohoto problému v podobě využití jednoho TCP spojení pro více žádostí. V HTTP hlavičce se tato funkce definuje parametrem "Keep-Alive". Další funkcionalitou verze 1.1 je tzv. HTTP pipelining, který umožňuje zasílání více žádostí bez čekání na odpovědi předchozích žádostí (viz obrázek HTTP/1.0 and HTTP/1.1). [17, 18]

Důraz na rychlost a efektivnost přenosu dat zapříčinil vznik HTTP/2, který přichází s tzv. Streamy. Každý dotaz má svůj vlastní stream. Tento stream vzniká společně s položením dotazu a zaniká s odpovědí. Rozdíl oproti TCP spojení však spočívá v založení a ukončení proudu, který nezahrnuje skoro žádnou režii ani zpoždění. Klient položí dotaz s novým identifikačním číslem proudu, a tím je proud založen. Server pak poslednímu datagramu odpovědi nastaví příznak END_STREAM, čímž jej ukončí. Proudy jsou na sobě nezávislé a datagramy se mohou zaměnit. Pokud dojde ke zpoždění nějakého z datagramů, neovlivní to jiné proudy. Ideálním využitím této funkcionality je přenos menších souborů. Každý datagram obsahuje identifikační číslo (31 bitů), které patří danému proudu. Proudy založené serverem mají sudá identifikační čísla, zatímco proudy zakládané klientem lichá. Proudy mají svoji prioritu, která určuje důležitost přenášených dat. Mezi další funkcionality 2. verze patří např. komprese hlaviček a ochrana proti zahlcení. [19, 20]

Další verzí je HTTP/3, který je založený na experimentálním protokolu QUIC, využívající protokol UDP.

Zabezpečená verze HTTP má označení HTTPS (HTTP Secure) a společně s HTTP využívá také protokoly SSL (Secure Sockets Layer) a TLS (Transport Layer Security). HTTPS umožňuje zabezpečenou komunikaci v podobě šifrování, autentizace, důvěryhodnosti přenášených dat a zajištění jejich integrity. [21]

Webový server Apache

Apache je volně dostupný softwarový webový server. Software je vyvíjen otevřenou komunitou vývojářů (Apache Software Foundation). Mezi jeho hlavní výhody patří: jednoduchost a rozsáhlé možnosti konfigurace, podpora řady operačních systémů a podpora velkého množství funkcí a programovacích jazyků (PHP, Perl atd.).

Dalšími charakteristickými funkcemi jsou: restrikce dle adresářů, domén a hesel, individuální omezení uživatelských stránek jednotlivými uživateli, možnost oddělení obsahu serveru od ostatních částí systému, zavádění tzv. aliasů (přezdívky adresářů), integrovaná proxy a využití virtuálních serverů tzv. Virtual Hostů a moduly rozšiřují jádro o další funkcionality (např. `mod_auth`, `mod_proxy`). [22, 23]

Apache je jeden z nejpopulárnějších a nejpoužívanějších webových serverů na světě (okolo 23%). Mezi další používané webové servery patří např. program NGINX.

2.2 Vývoj webových aplikací

Webová aplikace je souhrnný termín označující software zprostředkovaný internetovým prohlížečem. Tento software se v dnešní době dělí na dvě části. První částí je aplikace ze strany klienta tzv. frontend. Druhou částí aplikace je tzv. backend neboli serverová část aplikace. [24]

2.2.1 Frontend

Klientská část aplikace se zabývá obsahem, vzhledem a interakcí s uživatelem. Zjednodušeně řečeno zabývá se tím, co uživatel vidí a co může v aplikaci dělat. Klientská strana využívá především značkovacího jazyka HTML. Dříve webové stránky využívaly pro svoji funkčnost a vzhled jen HTML. S rozšířením webových stránek se objevil požadavek po vzhledu stánek. To dalo za vznik Kaskádovým stylům (CSS). Dalším požadavkem bylo dynamicky měnit obsah popř. vzhled, bez nutnosti znovu načtení stránek při každé změně. O tuto funkcionalitu se stará skriptovací jazyk JavaScript. [25]

HTML

Hyper Text Markup Language (HTML) je hypertextový značkovací jazyk určený k vytváření webových stránek. Vznikl na přelomu 80. a 90. let minulého století, odkud se vyvinul až do své aktuální podoby. Aktuální verzí HTML je verze 5 z roku 2014. Jak už z názvu vyplývá HTML používá značky, také nazývané tagy. Tagům se přiřazují atributy a hodnoty, které jednotlivým prvkům stránky přikládají určitou roli. Prostřednictvím HTML tagů se například definuje, kde budou odkazy a kam budou odkazovat nebo kde bude obrázek a odkud je bude prohlížeč načítat. [26]

Značky (tagy) se zapisují do špičatých závorek následujícím způsobem: `<tag>`. Tagy se dělí na párové a nepárové. Párové tagy musejí být ukončeny lomítkem a původním tagem (`</tag>`). Většina HTML tagů je párových, existují i výjimky, jako například zalomení řádku (`<br>`). Struktura HTML je dána několika hlavními značkami, které by měly být součásti každého HTML dokumentu. [27]

Struktura by měla obsahovat následující tagy:

- `<!DOCTYPE html>` – deklarace typu dokumentu
- `<html>` – kořenový element, který zastřešuje celý dokument
- `<head>` – hlavička určující metadata pro dokument typicky obsahuje scripty, kusy JavaScript kódu, odkaz na CSS či titulek
- `<meta>` – specifikace kódování (UTF-8)
- `<title>` – titulek stránky, který se zobrazuje v záložce prohlížeče
- `<body>` – obsah dokumentu

Kaskádové styly

Kaskádové styly (Cascading Style Sheets - CSS) se používají k formátování, rozložení webové stránky. Lze je použít k definování barev, fontů písma, stylů textu, velikostí tabulek, ořezání obrázků a spousty dalšího, které dříve bylo možné definovat pouze na stránce HTML.

CSS pomáhá vytvořit jednotný vzhled webu. Místo definování stylu každého nadpisu nebo tabulky na HTML stránce je možné v dokumentu CSS definovat pouze jednou. Jakmile je styl definován v souboru CSS, může být použit jakoukoli stránkou, která odkazuje na tento soubor. CSS navíc usnadňuje změnu stylů na několika stránkách najednou. Například webový vývojář může chtít zvýšit výchozí velikost textu pro padesát stránek webu. Pokud všechny stránky odkazují na stejnou šablonu stylů, je třeba změnit pouze velikost textu na šabloně stylů a na všech stránkách se zobrazí změněný text. [28]

Kaskádové styly poskytují přesnější kontrolu nad tím, jak budou vypadat webové stránky, než HTML. V dnešní době se můžeme setkat i se skriptovacími jazyky jako je např. Syntactically Awesome Style Sheets (SASS), které přináší dynamičnost a usnadňují stylování webových aplikací. [29]

JavaScript

JavaScript (zkráceně JS) je objektově orientovaný skriptovací jazyk učený pro tvorbu moderních dynamických webů. Byl představen v devadesátých letech 20. století, jako doplněk ke klasickým statickým webovým stránkám. JavaScript se dá použít pro interaktivní webové aplikace, příjemnější uživatelské rozhraní doplněné animacemi a grafikou. Prioritou JavaScriptu je zhotovování klientských částí aplikací, ovšem s příchodem Node.js a podobných technologií dokáže plnohodnotně pracovat i na straně serveru. Syntaxe JavaScriptu vychází z programovacích jazyků C/C++. Funkčně je ale od těchto jazyků zcela odlišný. S jazykem Java má společný pouze název a to z marketingových důvodů doby jeho vzniku. Programy psané v JavaScriptu se nazývají skripty a mohou se zapisovat přímo do HTML kódu. [30]

V minulosti JavaScript nebyl dostatečně kompatibilní a u některých prohlížečů, to mohlo způsobovat problémy v podobě špatné funkčnosti. V dnešní době je situace zcela odlišná. Popularita JavaScriptu stoupá a dělá z něj jeden z nejžádanějších programovacích jazyků. JS si svoji popularitu získal i díky frameworkům, jako je React nebo Angular. S postupem času se JavaScript používá i na vývoj mobilních aplikací prostřednictvím technologií, jako je React Native. [31]

Výhodami JavaScriptu je jeho rozšířenost, nabídka frameworků, které usnadňují tvorbu opakovaných částí webů i webových aplikací. Mezi nevýhody JavaScriptu je

možné zařadit potenciální hrozbu zneužití kódu, který se běžně zobrazuje v prohlížeči vedle HTML. Proti tomuto již existuje řada technik, kterými lze zneužití zabránit.

2.2.2 Backend

Backend je v modelu klient-server aplikací ta část aplikace, která se nachází na webovém serveru v podobě aplikačního serveru (aplikace). Hlavními funkcemi serverové části aplikace je výpočetní logika, komunikace s databází, zpracování a generování dat. V jednoduchosti to např. znamená, že server po přijetí uživatelského jména a hesla pomocí předem dané logiky zpracuje a porovná přijaté údaje v databázi a zašle výsledek zpět na frontend.

Rozšíření webových aplikací je v dnešní době tak rozsáhlé, že tvorbu serverové části podporuje (nějakým způsobem) téměř každý programovací jazyk. Mezi typické jazyky pro tvorbu webových aplikací patří PHP nebo Perl. Ovšem díky frameworkům lze pro vývoj použít např. Python, Java, ASP, JavaScript a další. Užití frameworků se může snížit počet chyb v aplikaci, především díky větší jednoduchosti a přehlednosti kódu, také možností koncentrovat se na důležitější části kódu. Součásti backednu mohou být i nejrůznější API (Application Programming Interface), které přes HTTP komunikují s jinou aplikací. [32]

Node.js

Node.js je framework, který umožňujě spuštění JavaScriptu na straně serveru. Node.js má tzv. single-thread procesy. To znamená, že pracuje v jednom vlákně, což podmiňuje využívání principů asynchronního programování, kdy není řešení jednoho požadavku závislé na dokončení požadavku druhého. To má za příčinu minimalizaci režie a maximalizaci výkonu. [33, 34]

Klientské požadavky jsou ukládány do zásobníku, ze kterého jsou vyjímány ve stejném pořadí, jedná se tedy o systém FIFO.Jádrem technologie je V8 engine od společnosti Google, který má na starosti kompilaci do strojového kódu. Architektura Node.js stojí na tzv. Event Loopu (Smyšcce událostí). Smyčka přijímá požadavky a ve formě událostí je přiděluje jednotlivým zdrojům. Ze smyčky a využití single-thread vychází potřeba asynchronní komunikace a princip fronty.V praxi to funguje tak, že vývojář vytvoří webový server, jenž bude na portu čekat na uživatelské požadavky. Jakmile nějaký požadavek zachytí, pošle ho do smyčky, která jej zadá příslušnému controlleru. V mezičase zpracovává další požadavky.Jakmile je požadavek zpracován, dojde k zavolání (callback) výsledku a následnému dokončení procesu. Složitější požadavky tak neblokují vlákno těm, které mohou být provedeny rychleji. [35]

3 Monitorování serverů

Následující kapitola se zabývá charakteristikou pojmu monitoring serverů a pojmy s ní spojené. Dále je zde shrnutý průzkum trhu s dostupným software pro monitorování serverů. Nakonec jsou zde rozebrány metriky, které je vhodné v rámci práce sledovat a ukládat.

3.1 Charakteristika a základní pojmy

Monitorování serverů je proces zahrnující získávání informací o aktivitě a událostech týkajících se síťových zařízení. Vzhledem k nárůstu kyberprostoru a ke stálému zvyšování kvality a dostupnosti služeb poskytovaných přes internet, je velice důležité získávat data o stavu síťových prvků. Získání a analyzování dat je nezbytné pro zajištění správného chodu zařízení, ale i pro bezpečnost a kvalitu služeb. Servery mohou být jedny z nejkritičtějších součástí společností. Zvláště společnosti podnikající v IT (jako jsou například e-shopy, komunikační služby atd.) přímo závisí na správném fungování jejich serverů a síťové infrastruktury. Z tohoto důvodu je logické, že získávání informací o stavech serverů a sítě jsou pro tyto společnosti velice důležité.

Monitorování poskytuje správcům serverů a infrastruktury řadu klíčových metrik, které slouží ke správnému běhu systému, služby atd. Mezi tyto metriky mohou patřit např. vytížení procesorových jednotek, využití vyrovnávací paměti nebo provoz na síťových kartách. Dále se může sledovat počet navázaných TCP spojení, doba odezvy a celá řada jiných metrik. Monitoring serveru se nejčastěji používá pro zpracování dat v reálném čase, ale má své uplatnění i při vyhodnocování a analýze historických dat. Při pohledu do historie může analytik určit nejrůznější informace o stavu infrastruktury, jako je například snížení výkonu serveru v závislosti na jeho životnosti. Díky sledování a analýze serverů může analytik dokonce predikovat stavy, které nastanou v budoucnu. [36]

Server management

S monitoringem serverů souvisí taktéž pojem správa serverů (Server management). Server management je nepřetržitý proces, kterým se zajišťuje provozuschopnost, spolehlivost a bezchybnost provozu a funkčnost serverů. Server management může zahrnovat širokou škálu specifických činností v závislosti na společnosti a její IT infrastruktuře (typy a počet zařízení). Ve většině společností správa serverů zahrnuje každodenní monitorování, instalaci aktualizací softwaru, instalaci a nastavení nového vybavení, řešení problémů a incidentů.

Správci serverů se také mohou setkávat s plánováním potřebné kapacity, aby byl zajištěný dostatek zdrojů pro splnění potřeb společnosti. Například vytížení e-shopů před vánocemi bude větší než po zbytek roku. Právě s tímto musejí správci počítat a připravit se na to. Server management zahrnuje také hardwarovou část správy. Správci se musejí potýkat i s fyzickými problémy, jako například spotřeba elektrické energie, záložní energetické zdroje, přenosová kapacita sítě nebo chlazení a optimální teplota pro správný provoz zařízení. [36]

Virtuální servery

S postupem času a vývoje hardwarového vybavení serverů společnosti došly do stavu, kdy nevyužijí celý potenciál zařízení nebo z ekonomického hlediska není výhodné vlastnit fyzické zařízení. To dalo za vznik tzv. Virtuálním serverům. Virtuální server je server, který využívá virtualizovaný hardware a běží na stejném stroji jako jiné virtuální servery, které jsou na sobě nezávislé. Tuto službu zpravidla poskytují společnosti zabývající se tzv. Hostingem. Ty se zabývají pronajímáním výpočetních prostředků a virtualizovaného hardware společnostem, které nechtějí vlastnit fyzický hardware. Virtualizace serverů zapříčinila vznik několika úrovní poskytování služeb týkajících se IT infrastruktury společností. Společnost tak může na třetí stranu přenést část provozu IT infrastruktury. V dnešní době se služby rozdělují na IaaS (Infrastructure as a Service), PasS (Platform as a Service) a SaaS (Software as a Service). [37]

Server management system

Systém správy serverů (Server management system) je softwarový nástroj, který umožňuje správcům serveru spravovat jeden nebo více serverů. Systém umožňuje shromažďovat provozní data, jako jsou využití procesoru, paměť, místo na disku, typy přijatých protokolů, informace o uživatelském přístupu a další. Tato data systém zobrazuje v reálném čase na řídicím panelu. Systém je také schopen shromažďovat historická data, což IT manažerům umožňuje sledovat tyto metriky v průběhu času.

3.2 Software pro monitorování serverů

V dnešní době existuje mnoho schopných programů a nástrojů pro širokou škálu monitoringu serverů a sítí. Mezi nejpoužívanější software patří Sematext, Datadog Prometheus a Grafana. Většina dostupných programů má velice propracované nástroje a funkce, které jsou pro svoji nadbytečnost a komplikovanost nežádoucí ve vyvíjené aplikaci v rámci této práce. Další nevýhodou je, že většina monitorovacího software je neustále v provozu a provádí mnoho procesů, které mohou zatěžovat méně výkonná zařízení, na kterém je spuštěn software. [38]

Mezi jednoduchý nástroj pro zjištění informací o využití zdrojů a o stavu aktuálně spuštěných procesech patří nástroj top. Program v záhlaví zobrazuje tzv. Load averages, souhrn procesů, stavy CPU, využití paměti. Tento nástroj je dostupný na operačních systémech vycházejících z UNIXu a i když se jeví jako jednoduchý, tak u méně výkonných zařízení výrazně zatěžuje výpočetní zdroje zdroje. Z programu vychází další nástroje pro monitoring stanice, jedním z nich je nástroj ntop, který rozšiřuje funkcionalitu top o webové rozhraní.

3.3 Základní metriky

Následující metriky slouží správcům serveru k vyhodnocení a predikci správného fungování serveru.

CPU

Procento zatížení procesoru je základní měřenou veličinou na serveru. Procento by v ideálním případě mělo dosahovat maxima jen zřídka a špičky by měly být krátké. Pokud je CPU často na maximu nebo se často blíží maximu i mimo špičku, je to známka toho, že systém není v pořádku. [39]

RAM

Paměť RAM se používá jako operační paměť počítačů. V této paměti jsou uloženy všechny běžící programy (včetně operačního systému) a jejich data. Tento parametr je velice důležitý pro sledování stavu serveru. Sledováním toho parametru se dají predikovat některé nežádoucí události, jako je například nadbytečné využití RAM některé z aplikací. Dále se dá predikovat životnost operační paměti. Pokud je vše v pořádku a přesto je procento využití RAM vysoké, může to znamenat blížící se konec životnosti paměti nebo nutnost navýšit operační paměť. Tento parametr se většinou udává v MB. Typicky 2678 MB z 4096 MB. Můžeme se ovšem setkat i s procentuálním vyjádřením. [40]

Bit rate

Bit rate neboli počet bitů za sekundu udává rychlost, kterou jsou bity přenášeny přes síťovou kartu a následně přes přenosové médium. Tento parametr se nejčastěji udává v bitech za sekundu (b/s), v kilobitech za sekundu (kb/s) nebo v megabitech za sekundu (Mb/s). Síťové karty mají ve své specifikaci uvedenou maximální možnou přenosovou rychlost. Sledováním tohoto parametru můžeme zjistit aktuální zatížení přenosové linky nebo síťové karty. Pokud je provoz vysoký a neměl by být, může to znamenat nežádoucí chování nebo zatěžování sítě nebo serveru. Parametr se dělí na *bit rate in* - počet příchozích bitů za sekundu a bit rate out - počet odchozích bitů za sekundu. [41, 42]

Packet rate

Dalším parametrem je *packet rate* tedy počet paketů za sekundu. Stejně jako u předchozího parametru se packet rate dělí na příchozí a odchozí. Parametr pomáhá sledovat dění na síťových kartách.

Počet navázaných TCP spojení

Tento parametr souvisí s protokolem TCP a udává počet navázaných TCP spojení. Po provedení TCP 3-Way Handshake je spojení navázáno a klient využívá nějakou službu aplikační vrstvy, po tuto dobu je stále navázáno TCP spojení a to udává tento parametr. Stavy protokolu TCP jsou nastíněny na obrázku 3.1.

Počet přijatých TCP-SYN žádostí

Dalším parametrem, který souvisí s protokolem TCP je počet přijatých SYN žádostí (SYN-RECEIVED viz obrázek 3.1). Tento parametr souvisí s TCP 3-Way Handshake a reprezentuje počet přijatých packetů s příznakem SYN. Dle TCP 3-Way Handshake by server měl odpovědět klientovy s příznaky SYN, ACK.

Počet odeslaných TCP-SYN žádostí

Počet odeslaných TCP-SYN žádostí (SYN-SENT viz orbázek 3.1) udává počet odeslaných TCP packetu s příznakem SYN. S tímto parametrem je možné se setkat na klientské straně TCP spojení.

Počet TCP spojení v stavu FIN-WAIT

Tento parametr udává počet TCP spojení ve stavu čekání na potvrzení ukončení TCP spojení. Tato situace nastává u tzv. aktivního ukončení spojení (na straně serveru).

Počet TCP spojení v stavu TIME-WAIT

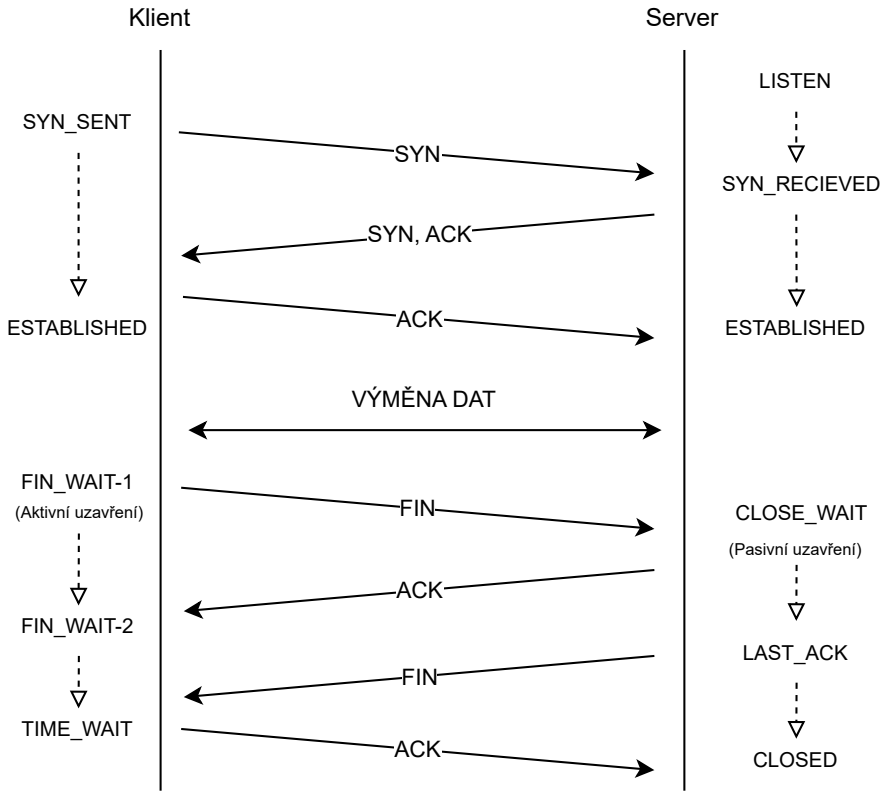
Dalším parametrem souvisejícím s aktivním ukončením TCP spojení je stav TIME-WAIT. Tento stav představuje spojení, které je ještě po nějakou dobu otevřené a čeká, než protistrana (klient) obdrží potvrzení požadavku na ukončení spojení.

Počet TCP spojení v stavu CLOSE-WAIT

CLOSE-WAIT parametr souvisí s tzv. pasivním ukončením TCP spojení (strana serveru). Po tom co server obdrží od klienta FIN příznak pro ukončení spojení, přechází stav spojení z ESTABLSHED do CLOSE-WAIT, server tedy čeká na uzavření.

Počet TCP spojení v stavu CLOSING

Tento parametr udává počet spojení ve stavu, kdy klient čeká na potvrzení ukončení spojení (FIN) od serveru. Stav CLOSING je dočasným stavem, který existuje po obdržení žádosti o ukončení (FIN) od druhé strany a před přechodem do stavu TIME-WAIT.



Obr. 3.1: Stavy TCP u navázání a ukončení spojení

Počet přijatých HTTP žádostí

Tento paramtr souvisí s protokolem HTTP a reprezentuje počet přijatých HTTP žádostí na webový server. Sledováním tohoto parametru je možné zjistit počet přiojených klientů k webovému serveru. Pokud se aktuální hodnota výrazně liší od dlouhodobějších hodnot, může se jednat o nestandardní situaci, kterou může být např. DoS útok.

Průměrná délka HTTP žádosti a odpovědi

Průměrná délka HTTP žádosti a odpovědi reprezentují průměrnou velikost přijatých a odeslaných HTTP segmentu za jednu sekundu. Jednotku tohoto parametru jsou byty za sekundu (B/s).

Průměrná čas vyřízení HTTP požadavku

Průměrná čas vyřízení HTTP požadavku udává průměrný čas (v sekundách), který server stráví vyřízením jednoho požadavku.

3.4 Ukládání a uchování metrik

Naměřená a získaná data je potřeba uchovávat pro další použití. K tomuto účelu se používají databáze.

Databáze je softwarový systém pro ukládání dat a jejich následné zpracování. Databáze se používají při programování webových aplikací. Právě v databázi jsou uložena data, se kterými dále operuje aplikace. Data v databázi jsou ve strukturovaném formátu, s nimiž lze pomocí nějakého databázového systému manipulovat. Typickými operacemi jsou vyhledávání, vkládání, editování, třídění a porovnávání.

Software, který zajišťuje komunikaci mezi databází a aplikačními programy se nazývá *Database Management System*. Tento systém musí být schopen efektivně pracovat s velkým množstvím dat, ale také musí řídit (vkládat, modifikovat, mazat) a definovat strukturu těchto perzistentních dat. Touto funkcí se liší od prostého souborového systému. [43]

Databáze se dělí podle modelů a typu ukládaných dat do několika skupin. Na tři z nich se zaměřím v následujících podkapitolách.

Relační databáze

Nejpoužívanějším typem je v současnosti relační databáze. Základem jsou tzv. relace, což jsou databázové tabulky. Ty se skládají ze sloupců (atributů) a z řádků (záznamů). Jednotlivé atributy pak mohou plnit například funkci kandidátního, primárního, alternativního nebo cizího klíče. Všechny atributy, které jsou označovány jako kandidátní klíče, se teoreticky mohou stát primárním klíčem. Ty, které se jím nestanou, se pak nazývají alternativní klíče. Primární klíč musí, kromě své unikátnosti, vždy obsahovat nenulovou hodnotu a měl by být neměnný. Pokud neexistuje přirozený primární klíč (unikátní identifikátor), použije se uměle vytvořený identifikátor. Ten se automaticky přiřadí každému novému záznamu.

Relační databáze se tvoří pomocí jazyka SQL (Structured Query Language). Díky strukturovanosti a vysoké konzistentnosti je možné záznamy v tabulce snadno filtrovat, třídit a provádět s nimi výpočty. Jednotlivé relace lze propojovat mezi sebou pomocí tzv. cizích klíčů, jenž se shodují s primárním klíčem jiné relace. V praxi se v různých systémech propojují například záznamy o zákaznících, objednávkách a produktech, které jsou uvedeny v oddělených relacích. [44]

V dnešní době mezi nejpoužívanější SQL databázové systémy patří MySQL, PostgreSQL, MSQL a spousty dalších. Zvláštní pozornost bych chtěl věnovat SQLite databázi.

SQLite je relační databázový systém, který klade důraz na jednoduchost a malé zatížení systému. Databáze není založená na principu klient–server, kde je databázový server spuštěn jako samostatný proces, SQLite je pouze nevelká knihovna, která po přilinkování k aplikaci, je k dispozici pomocí jednoduchého rozhraní. Každá databáze je uložena v samostatném souboru .dbm (Database Manager). Tento formát databázových souborů je přitom nezávislý na operačním systému a SQLite tak představuje jednoduchý nástroj pro přenos strukturovaných dat.

Nerelační databáze

Nerelační (NoSQL) databáze jsou databáze, které se od relačních zásadně liší. Neukládá totiž data v pevně definovaných tabulkách. Datový model je tak velmi flexibilní a dovoluje kombinovat odlišné struktury dat bez nutnosti zasahovat do schématu databáze. Název NoSQL však může být trochu zavádějící, protože nerelační databáze umí pracovat rovněž s SQL dotazy. Přesnější název by tedy mohl být *„NotOnlySQL“* nebo *„Nejen relační databáze“*.

Existuje několik typů NoSQL databází, které se odlišují tím, jak pracují s jednotlivými záznamy. Tou nejjednodušší variantou je *„Key-value“* databáze. Ta přiděluje každé hodnotě při ukládání unikátní klíč. Hodnota může mít podobu čísla, řetězce nebo klidně odkazu na další pár *„klíč–hodnota“*. Na rozdíl od relačních databází zde

není předdefinovaná přesná struktura, což umožňuje vysokou flexibilitu a snadné propojování záznamů. Dále existují NoSQL databáze založené na Grafech, či modelu *JSON*. Nerelačních databázových systémů existuje mnoho. Mezi ty nejpoužívanější patří například MongoDB, Apache Cassandra, Elasticsearch nebo Redis. [44]

TSDB databáze

Time series database (TSDB) - databáze časových řad, jsou databáze optimalizované pro ukládání času a měřených veličin (např. teplota). TSBD jsou vhodné pro pravidelné měření stejné veličiny v průběhu času. Typicky se jedná o metriky jako je například teplota nebo zátěž procesoru. Dále je tento typ databází vhodný i pro zaznamenávání tzv. *eventů* - nepravidelné měření stejné veličiny (např. změna stavu atd.). TSBD systémy umí efektivně vyhledávat hodnoty podle času nebo časového rozmezí. Mezi hlavní přednosti TSDB systému patří jednotnost ukládaných dat. Díky jednotnosti se data dají lépe komprimovat. Oproti jiným typům databází jsou TSDB méně náročné na úložiště a výkon. TSDB databáze lze nakonfigurovat tak, aby pravidelně odstraňovaly stará data, na rozdíl od běžných databází, které jsou navrženy k ukládání dat na dobu neurčitou. Mezi nejpoužívanější databázové systémy tohoto typu patří InfluxDB a TimescaleDB. [45]

4 Denial of service útoky

V dnešní době lze monitoringem serverů a sítě zjišťovat nejrůznější informace, které mohou mít dopad na celou organizaci popř. společnost. Jedním z důvodů proč monitorovat server a síť je schopnost detekovat (a následně předejít) tzv. DoS útokům. Právě tomuto druhu útoků se chci věnovat v následující kapitole.

4.1 Charakteristika

Pojmem Denial of service (DoS) - odepření služby vyjadřuje typ kybernetického útoku většinou cíleného na internetové služby nebo stránky. Cílem útočníka je znepřístupnit nebo znefunkčnit cílovou službu ostatním, legitimním uživatelům. K tomu lze dojít dvěma způsoby. Prvním způsobem útočník zahltí cílovou službu obrovským množstvím požadavků. Tím dojede k vyčerpání zdrojů systému (CPU, RAM, šířka pásma na síťových kartách) a legitimnímu uživateli se nepodaří ke službě přistoupit nebo útok výrazně omezí fungování služby.

Druhým způsobem je využití některé chyby, která útočníkovi neumožní službu nebo část služby ovládnout, ale umožní ji znefunkčnit. Chybou se mohou rozumět i vlastnosti a funkčnost síťových protokolů. [48]

Mezi hlavní projevy DoS patří:

- Celková nedostupnost části nebo celé služby
- Neobvyklé zpomalení služby
- Nemožnost připojení legitimního uživatele ke službě
- Extrémní nárůst obdrženého spamu

Splnění těchto podmínek nutně nemusí znamenat DoS útok. Může se jednat o výpadek zaviněný hardwarem nebo softwarem samotného serveru bez cizího zavinění. Schopnost rozlišovat DoS útoky od výpadků nám právě pomáhají nástroje pro monitoring serverů a sítí.

Mezi speciální druh DoS patří tzv. DDoS neboli distributed denial-of-service, při kterém je pro zahlcení cílové služby využito více uzlů z různých geografických lokalit, které s hlavním útočníkem spolupracují a napadají cílovou službu v určitém čase a s určitou intenzitou. Odtud slovo *„distribuovaný“*. [46, 47]

4.1.1 Záplavové útoky

Záplavové DoS (DoS Flood) jsou jedny z nejprostších. Jejich cílem je vygenerování co největšího počtu požadavků tak, aby cílová služba nebyla schopna odpovídat legitimním uživatelům. Také se ale může jednat o zahlcení linky oběti. Nebezpečnost těchto útoků spočívá v tom, že je téměř nemožné jim přímo zabránit, zvláště pokud jsou distribuované.

ICMP Flood

Tento útok využívá Internet Control Message Protocol (ICMP), nejčastěji se používají pakety typu ICMP Echo. Jde o packety, které využívají ping a slouží ke zjišťování dostupnosti vzdáleného zařízení. Základním předpokladem je, že útočník má k dispozici rychlejší připojení k internetu. Principem ICMP Flood útoku je zasílat ICMP Echo na cílový server a přitom podvrhnout zdrojovou adresu, aby se ICMP Echo reply pakety nevracely. Je zde také využito toho, že ICMP Echo reply paket je stejně velký jako ICMP Echo. Když tedy útočník zašle dostatečné množství velkých ICMP Echo paketů (s pozměněnou zdrojovou adresou) a má rychlejší připojení k internetu než oběť, tak vyčerpá přenosovou kapacitu linky oběti, kde už nebude prostor pro pakety od legitimních uživatelů. [49]

UDP Flood

U UDP Flood útoku se využívá transportního protokolu UDP a zranitelnosti služeb echo a chargen. Služby echo a chargen bývaly defaultně spuštěny v operačních systémech Linux. Služba echo pracuje tak, že veškerá data, která přijdou na její port, jsou poslána zpět. Služba chargen funguje tak, že příchozí data pošle jako data náhodná. Útok spočívá v zaslání dat na port echo se zfalšovanou zdrojovou adresou a portem. Zfalšované údaje přitom odpovídají na jiný sever se službou echo nebo chargen. Tím útočník docílí toho, že tyto dva servery si budou data posílat stále dokola. Výhodou tohoto útoku je, že oběť může mít daleko vyšší rychlost připojení než útočník. Zmíněné služby se ovšem v dnešní době již zřídka kde používají. [50]

4.1.2 Logické útoky

Logické DoS jsou útoky využívající logickou slabinu, která se nachází v aplikaci, operačním systému oběti nebo síťovém protokolu. Tyto chyby se projevují například tak, že při určitých speciálně upravených packetech program vytíží procesor více, než je obvyklé, nebo program začne obsazovat více paměti. Logické DoS se dají považovat za nejnebezpečnější DoS útoky, protože k jejich provedení jsou minimální

požadavky. Útok většinou spočívá pouze ve vygenerování malého množství speciálních packetů a oběť je nedostupná. Na rozdíl od záplavových útoků zde nehraje roli rychlost připojení útočníka a oběti. Díky tomu, že k provedení útoku stačí několik packetů, je těžké útočníka vysledovat, a také je velice těžké se tomuto útoku bránit nějakými filtrovacími pravidly. Logické útoky se dají dělit na protokolové útoky - zneužívající vlastnosti protokolu nebo aplikační útoky - zneužívají chybnou implementaci aplikace, či služby.

Ping of Death

Jedná se o jeden z nejstarších útoků využívající chyby protokolu, který na své provedení není složitý. Útok používá ICMP Echo request packetu. Specifikace ICMP uvádí, že délka ICMP Echo paketů může být maximálně 65.535 bytů. Když ale útočník porušil specifikaci a velikost změnil na větší, některé operační systémy na to nebyly připraveny a při obdržení velkého packetu zkolabovaly. Tato chyba byla ve většině operačních systémech odstraněna a v dnešní době je útok nepoužitelný. [51]

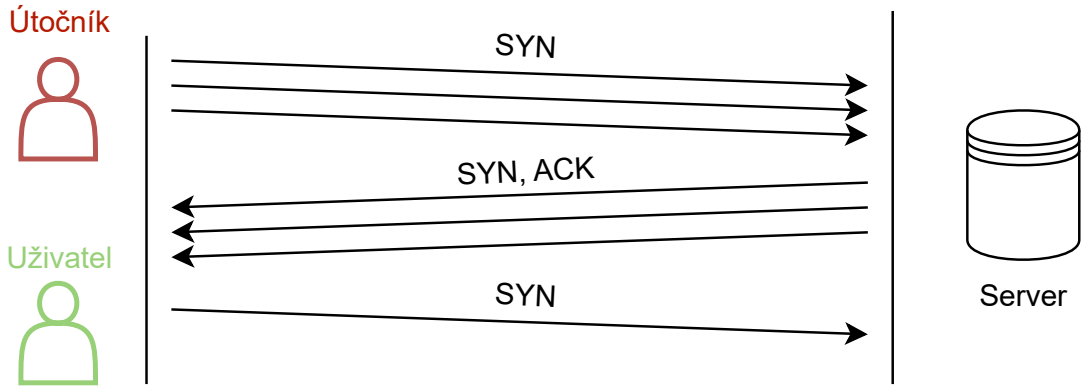
SYN Flood

Útok zneužívá implementace navázání spojení u protokolu TCP (Three way handshake). Útočník (v roli klienta), posílá serveru příznak SYN k navázání komunikace a server mu nazpět odešle SYN + ACK, server poté čeká na ACK od útočníka. Jelikož se pakety v síti často ztrácejí a server čekající na odpověď si „myslí“, že packet nebyl doručen tak pošle znovu příznak SYN + ACK. Po určité době pokud nepřijde serveru příznak ACK, tak vymaže toto spojení. Problém nastává ve chvíli, kdy útočník pošle více příkazů SYN a vyčerpá serveru všechna možná spojení s klienty, což způsobí odstavení serveru pro jiné legitimní uživatele. Příznak SYN má pouhých 42 bytů, a proto nezáleží na rychlosti spojení a je možnost zfalšovat IP adresy. Jako ochrany před tímto útokem může být využito snížení doby čekání na odpověď serveru. Jde o podobný typ útoku jako u útoků využívajících chyb implementace, které jsou způsobené špatným navržením. Tyto chyby mohou způsobit větší zátěž na procesoru nebo na paměti a k útoku je zde potřeba většího datového toku a rychlost připojení zde také hraje roli. Princip je v tom, že útočník pošle velké množství packetů, procesor napadeného zařízení se přetíží a nestíhá reagovat a nastane režim nepoužitelnosti. Zjednodušený princip SYN Flood útoku je demonstrován na obrázku 4.1. [52]

SlowLoris

Jedná se o útok využívající funkčnost webového serveru a HTTP požadavku na něj. Princip SlowLoris útoku spočívá v navázání TCP spojení a jeho udržování po dlou-

hou dobu (klidně i několika hodin). Toho útočník dosáhne zasíláním nedokončeného HTTP požadavku a to stále dokola (pomalým způsobem těsně před vypršením timeoutu). Webový server obvykle otevře spojení a čeká na doručení celého HTTP požadavku, na který bude odpovídat. Útočník ale pomalu posílá jednotlivé řádky nekonečné hlavičky. A server trpělivě čeká a čeká a nikdy se nedočká. Server tedy zůstane zahlcen čekajícími spojeními a přestane odpovídat na ta legitimní. Web server Apache má standardně timeout nastavený na 300 sekund. Pokud mu v tomto intervalu útočný software pošle alespoň jeden další řádek hlavičky, počítadlo se vynuluje a server čeká dál. S naprosto minimální námahou tak útočník dokáže spojení udržovat neomezenou dobu. [53]



Obr. 4.1: Princip SYN Flood

4.2 Motivace útočníků

Motivů pro využívání Denial od Service útoků útočníky může být spousta zde jsou ty hlavní:

- **Vydírání** – Útočník vymáhá finanční prostředky od oběti pod útokem.
- **Konkurence** – Získání konkurenční výhody (např. jeden provozovatel e-shopu si objedná nebo provede útok na svoji konkurenci)
- **Válka** – státem sponzorované útoky (znepřátelené země, opozice atd.)
- **Ideologie** – rozdílné ideologické nebo politické názory
- **Zakrytí hlavního útoku** – např. krádež dat,
- **Script-kiddies** – Útočník provádí činnost pro svoje uspokojení či pro zábavu.

5 Implementace a řešení

V této kapitole je rozebrána praktická část bakalářské práce. Je zde popsaný návrh a implementace nástroje pro monitorování serveru. Cílem práce bylo navrhnout a vytvořit software pro sběr, metodiku pro přenos a ukládání dat a následně vyvinutý nástroj propojit s již existující webovou aplikací pro vizualizaci nasbíraných dat v reálném čase. První část je věnována návrhu software. Ve druhé části je popsána implementace, způsob řešení, dále jsou zde probrány implementační problémy a testování monitorovacího systému.

5.1 Návrh

Nástroj byl vytvořen pro sběr dat v rámci monitoringu serveru. V dnešní době existuje velké množství monitorovacího software, který splňuje veškeré požadavky na tento software a navíc disponují dalšími funkcemi. Mnohdy tyto nástroje bývají komplikované a náročné. Proto vyvíjený nástroj klade velký důraz na lehkost a jednoduchost, jak zatížení serveru, tak možnosti spuštění a následně jeho používání. Velký důraz je zde také kladen na co nejmenší a nejefektivnější přenos naměřených dat po síti a to v jen žádanou dobu. Právě tuto funkcionalitu spousta monitorovacího software nepodporuje a komunikace s ostatními systémy probíhá pravidelně bez možnosti přerušení.

5.1.1 Požadované vlastnosti nástroje

U nástroje je důležité, aby splňoval určité vlastnosti, které jsou nezbytné pro jeho správnou implementaci.

Sběr dat

Nástroj bude schopný zaznamenávat data jako jsou: vytížení CPU a RAM, bitrate (in/out), packetrate (in/out) a TCP established (viz kapitola 3.3).

Ukládání dat

Nasbíraná aktuální data bude v rámci software potřeba ukládat pro další použití (vyžádání od webové aplikace pro vizualizaci). Uložení a celková práce s daty by měla být co nejjednodušší a nejefektivnější s ohledem na vytížení prostředků monitorovaného serveru.

Přenos dat

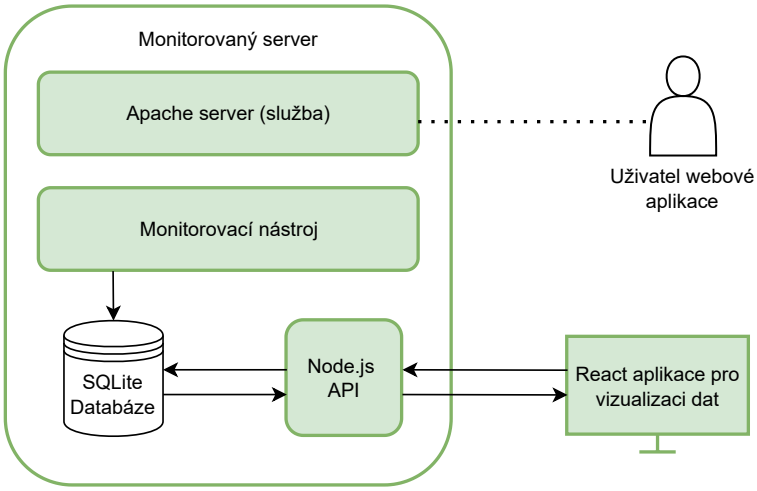
Nástroj bude mít důraz na co nejmenší síťové zatížení tak, aby přenášená data mezi monitorovaným zařízením a webovou aplikací pro vizualizaci byla co nejmenší. Kdy v ideálním případě budou stejná data přenesena pouze jednou.

Omezení komunikace

Nástroj bude s webovou aplikací pro vizualizaci komunikovat pouze na vyžádání a to v případě zaslání požadavku a úspěšného přijetí dat.

5.2 Popis řešení

Vyvíjený nástroj se skládá z několika částí (viz 5.1). První částí je webová aplikace na zobrazení a vizualizaci dat. Implementace této aplikace není součástí této práce, ale je součástí návrhu. Aplikace byla vytvořena v rámci bakalářské práce jiného studenta. Slouží k vizualizaci naměřených dat, sledování aktuálního zatížení monitorovaných stanic a zpětnému prohlížení dat v rámci definovaného časového intervalu. Druhou částí celého systému je software nacházející se přímo na monitorovaném zařízení. Konkrétně se jedná o nástroj na sběr (získávání) metrik a jejich následné ukládání do databáze. Dále se na monitorované stanici nachází nástroj pro čtení dat z databáze a odesílání do webové aplikace pro vizualizaci. Na monitorované stanici se může nacházet jedna či více služeb (jiných aplikací), například se může jednat o webový, mailový či DNS server. V následujících podkapitolách jsou jednotlivé části detailněji popsány.



Obr. 5.1: Diagram návrhu nástroje

5.2.1 Sběr dat

Nástroje, příkazy a postupy pro získání potřebných dat jsou implementované pro použití na operačních systémech Linux. Metriky jsou sbírány a ukládány do databáze v rámci bash skriptu, který je spuštěn v nekonečné smyčce. Zde jsou popsány metody a principy sběru jednotlivých metrik.

Procento vytížení procesorové jednotky

Pro získání procenta vytížení procesoru (CPU) existuje mnoho nástrojů a postupů jak tato data získat. V rámci Linuxu a většiny jeho distribucí je oblíbený a nejvíce využívaný nástroj **top**. Tento nástroj slouží k zobrazení a získání dat o využití procesoru a mimo jiné i o právě spuštěných procesech. Nástroj je jednoduchý a dají se z něj zjistit potřebná data, tedy aktuální stav vytížení procesoru.

Mezi další podobné nástroje patří např. nástroj **htop**, který ovšem nebývá v základní podobě systému nainstalován a je potřeba jej doinstalovat. Tento nástroj rozšiřuje program **top**, ale pro použití v rámci práce je vhodnější právě nástroj *top*, právě kvůli jeho jednoduchosti a dostupnosti.

Výpis 5.1: Příklad použití pžíkazů pro zjištění stavu CPU

```
top -b -n 1 | grep Cpu | cut -d ':' -f2 | awk '{print $1}'
```

Společně s nástrojem **top** byly využity další linuxové nástroje jako jsou **grep**, **cut** a **awk**. Tyto nástroje slouží k očištění a získání dat o CPU z příkazu *top*. Konkrétně příkaz *grep* vrátí pouze řádek na kterém se nachází parametr „Cpu“. Příkazy na zjištění hodnoty CPU jsou nastíněny ve výpisech 5.1 a 5.2.

Výpis 5.2: Popis funkčosti jednotlivých nástrojů na očištění hodnoty CPU

```
$ top -b -n 1 | grep Cpu
%Cpu(s): 52.1 us,  0.0 sy,  0.0 ni, 52.1 id,  0.0 wa,  0.0 hi
```

Dále pomocí příkazu **cut** je odebrán řetězec s „Cpu“, tak aby zůstaly pouze hodnoty. Nakonec je použit příkaz **awk**, který vrátí požadovanou hodnotu nacházející se na prvním místě. Tato hodnota se později ukládá do databáze (viz 5.2.2).

Procento vytížení paměti

Pro zjištění procenta aktuálního využití RAM (random-access memory) opět existuje nepřeberné množství nástrojů a způsobů jak tuto informaci zjistit. Nejednodušším způsobem jak zjistit tuto informaci na operačních systémech Linux je použití nástroje **free**. Tento nástroj slouží k zobrazení velikosti volné a využité paměti v systému, jakož i vyrovnávací paměti používané jádrem. Za pomoci dalších nástrojů lze z příkazu **free** zjistit námi požadovaná data.

Výpis 5.3: Příklad použití příkazů pro zjištění aktuálního využití RAM

```
free -m | grep "Mem" | awk `{print $3/$2*100}`
```

Příkazem **free** je možné zjistit aktuální využití RAM. Nástroj **grep** výstup z předchozího příkazu očistí o všechny řádky kromě požadovaného, začínajícího na „Mem“. Následně za pomoci nástroje **awk** se vypíše výsledek výpočtu procentuálního vytížení paměti. Příkaz v celé podobě je ve výpisu 5.3. Tento příkaz je vytvořený tak, aby co nejméně zatěžoval chod zařízení, na kterém je spuštěný. Zároveň všechny nástroje zpravidla bývají součástí základních nástrojů ve většině distribucích Linuxu.

bit rate a packet rate

Pro získávání těchto parametrů je nejjednodušší využít nízkoúrovňové nástroje obsažené v systému Linux. Celkem jsou použity čtyři příkazy pro získání čtyř metrik (bit rate IN, bit rate OUT, packet rate IN, packet rate OUT). K těmto informacím je nejjednodušší se dostat pomocí příkazu **cat** (viz 5.4). Obdobným způsobem je možné zjistit zbylé tři metriky.

Výpis 5.4: Příklad užití net/statistics

```
cat /sys/class/net/<interface>/statistics/rx_bytes
```

Výstupem těchto příkazů je vždy číslo. Toto číslo reprezentuje počet přijatých bitů nebo packetů od připojení daného síťového rozhraní nebo od startu systému. Proto nejjednodušší způsob pro zjištění bit rate ve formátu bit/sec je následující.

Výpis 5.5: Příklad získání parametru bit rate in

```
1 #!/bin/bash
2 interface=enp0s5
3 interval=1
4 R1=`cat /sys/class/net/$interface/statistics/rx_bytes`
5 sleep $interval
6 R2=`cat /sys/class/net/$interface/statistics/rx_bytes`
7 RBPS=$(($R2-$R1))
8 RbPS=$(($RBPS*8))
9 echo $RbPS bit/s
```

První je nutné uložit aktuální hodnotu např. *rx_bytes* vyčkat nějaký časový interval např. 1 sekundu, znovu uložit hodnotu *rx_bytes* a poté dvě získané hodnoty od sebe odečíst. Odečtená hodnota je v bytech, proto je nutné ji ještě převést na požadované bity. Výsledkem je počet přijatých bitů za časový okamžik. Ukázka skriptu je ve výpisu 5.5. Podobným způsobem lze získat i parametry packet rate.

Opět existuje spousta nástrojů, jak tato data získávat a monitorovat už v rámci nějakého nástroje. Oblíbeným nástrojem je **vnstat**, který tyto informace zobrazuje rovnou. Problémem tohoto řešení je, že **vnstat** není součástí základní programové výbavy v některých distribucích Linuxu a bylo by jej nutné doinstalovat.

Parametry týkající se TCP stavů

Mezi tyto parametry patří: ESTABLISHED, SYN_RECIEVED atd. viz 3.3. Tyto parametry lze jednoduše zjistit za pomoci nástroje **netstat**, který slouží k zobrazení aktuálně navázaných síťových spojení. Společně s přepínači, příkazu **grep** a **wc** lze získat požadované hodnoty, jako je například počet navázaných TCP spojení. Ukázka získání metrik pomocí předešlých příkazů je ve výpisu 5.6.

Výpis 5.6: Příklad získání TCP parametrů

```
tcp_est=`netstat -tan | grep ESTABLISHED| wc -l`
syn_recv=`netstat -an | grep SYN_RECV | wc -l`
syn_sent=`netstat -an | grep SYN_SENT | wc -l`
fin_wait=`netstat -an | grep FIN_WAIT | wc -l`
time_wait=`netstat -an | grep TIME_WAIT | wc -l`
close_wait=`netstat -an | grep CLOSE_WAIT | wc -l`
closing=`netstat -an | grep CLOSING | wc -l`
```

HTTP parametry

Jedná se o parametry, které souvisí s protokolem HTTP a webovým server Apache. Konkrétně se jedná o počet přijatých HTTP žádostí, průměrný čas vyřízení HTTP požadavku a průměrná délka HTTP žádosti a odpovědi. Získání těchto dat probíhá prostřednictvím čtení informací z logu programu Apache. Jelikož základní formát logu programu neobsahuje všechny potřebné hodnoty, je nutné upravit formát logu 5.7. Konkrétně je nutné přidat následující dva řádky do konfiguračního souboru programu Apache. Detailní postup je popsán v návodu na spuštění, který je součástí příloh.

Výpis 5.7: Specifický formát logu programu Apache

```
LogFormat '%h %l %u %t %r %>s %b %I %O %T %D' custom
CustomLog /var/log/apache2/custom.log custom
```

Při uchovávání logů ve správném formátu je možné získávat ze souboru potřebná data opět za pomocí příkazu **grep** a **awk**. Důležitou částí je *sum += $11*, která souvisí s parametrem *%I* ve formátu logu 5.7. Výsledkem celého příkazu je velikost přijatých požadavků v daném časovém okamžiku. Poté je data nutné vydělit počtem

přijatých HTTP požadavků, tedy vypočítat průměr velikosti přijatých požadavků za časový okamžik, typicky za jednu danou sekundu. Obdobným způsobem probíhá sběr a výpočet i ostatních metrik souvisejících s HTTP webserverem Apache.

Výpis 5.8: Příklad získání HTTP parametrů

```
#!/bin/bash
LOG=/var/log/apache2/custom.log
now=`date +%d/%b/%Y:%H:%M:%S`
req_size=$(grep $now $LOG | awk `{sum += $11} END {print sum}`)
avg_request_size=$((http_total_request_size / http_num_requests))
```

Nutné je podotknout, že sběr metrik spojený s HTTP protokolem lze monitorovat jen za předpokladu, že se jedná o webový server s programem Apache, který zapisuje logy ve správném formátu. Nicméně sběr ostatních metrik je možný i na jiných typech zařízení na kterých není webový server Apache. Za tohoto předpokladu mají HTTP metriky nulové hodnoty. Postup pro získání metrik je ve výpisu 5.8.

5.2.2 Ukládání dat

Naměřené a získané metriky je nutné vhodným způsobem ukládat pro další použití, zejména pro odeslání webové aplikaci. Z toho důvodu bylo nutné najít vhodný systém pro ukládání naměřených dat. V rámci práce byly testovány dva databázové systémy.

První databázovou aplikací bylo SQLite. V rámci této aplikace nebylo nutné vytvářet složitou strukturu databáze. Jedná se o model pouze s jednou tabulkou, kde jednotlivé sloupce reprezentují měřené metriky a čas, ve kterém byly naměřeny. Tyto hodnoty je nutné ukládat v rámci skriptu pro sběr a ukládání metrik. Jediným předpokladem je, aby aplikace byla nainstalovaná na monitorovaném zařízení. Ukládat metriky do databáze je možné v rámci skriptovacího jazyku bash za pomoci příkazu **sqlite** (viz 5.9).

Výpis 5.9: Ukládání metrik do SQLite

```
sqlite3 $DB_FILE `INSERT INTO data(cpu,ram,bit_rate_in,...)
    VALUES ($cpu_use,$mem_use,$TbPS,...,strftime('%s'));`
```

Výhodou SQLite je především jednoduchost a skutečnost, že systém funguje na principu klient-server, kde je databázový server spuštěn jako samostatný proces.

Druhým testovaným databázovým systémem byl InfluxDb. Jedná se o TSDB systém, který je vhodný pro ukládání času a měřených veličin, tedy i pro využití v rámci této práce. Další výhodu TSDB je efektivní vyhledávání dat podle času. Hlavní nevýhodu InfluxDB je, že systém funguje na principu klient-server a při velkém zatížení sítě hrozí riziko neukládání a následná ztráta naměřených dat.

5.2.3 Přenos dat

Přenosem dat je myšlen způsob, který zahrnuje čtení dat z databáze a jejich odesílání z monitorované stanice do aplikace pro vizualizaci. Z práce studenta, který vyvíjel aplikaci pro vizualizaci, vychází, že přenos dat po síti má probíhat prostřednictvím protokolu HTTP. Způsobem, kdy se aplikace dotazuje přímo na monitorovanou stanici, která na vyžádání odešle aplikací potřebná data. Z pohledu monitorovaného zařízení se jedná o jakési API, prostřednictvím kterého webová aplikace pro vizualizaci bude získávat naměřená data. Pro tuto funkčnost byl vybrán programovací jazyk JavaScript s frameworkem Node.js, pomocí kterého je možné přistupovat k naměřeným datům v databázi a následně je na vyžádání zasílat aplikaci pro vizualizaci.

Dle aplikace pro vizualizaci dat, je potřeba, aby API odesílalo naměřená data ve čtyřech možných formátech. Tyto formáty definují jeden nebo více časových okamžiků, na základě kterých, mají být naměřené hodnoty přeneseny do aplikace pro vizualizaci. Formáty jsou následující:

- **Range** – Odeslání dat z určitého časového rozsahu (např. od 11:00:00-11:05:00).
- **Update** – Formát vyžadující odeslání dat od určitého časového okamžiku po současnost (např. 11:00:00 - nyní).
- **Times** – Formát určený k odeslání konkrétních časových okamžiků (např. 11:00:10, 11:00:21, 11:00:40)
- **All** – Tento formát slouží k získání všech časových okamžiků v databázi. Pokud je skript na sběr dat spuštěný po delší dobu, tudíž je v databázi hodně dat, stává se tento formát nevhodným, protože trvá dlouho a zatěžuje síť.

Tyto formáty jsou definované ze strany aplikace pro vizualizaci a nacházejí se v těle HTTP žádosti (viz 5.10).

Výpis 5.10: Ukázka HTTP žádosti

```
POST http://<server_address>:<port>/
Content-Type: application/json
{type: range, from:‘2023-04-21 11:00:00 GTM+0200‘,
to:‘2023-04-21 11:02:00 GTM+0200‘}
```

Úkolem API je tedy přijmout žádost, zjistit o jaký se jedná formát, na základě kterého se provede vhodný dotaz do databáze a v poslední řadě je nutné získaná data z databáze v HTTP odpovědi ve shodném formátu vrátit zpět aplikaci pro vizualizaci.

Celý způsob není složitý, výpočetní a časová složitost ovšem může stoupat s většími časovými intervaly. Kdy hodně dat v odpovědi znamená delší dobu přenosu.

Postup zpracování požadavku je následující. Nejdříve je zjištěno o jaký typ formátu se jedná, tuto hodnotu převezme Node.js z HTTP požadavku a nachází se v *request.body["type"]*. Po rozlišení o jaký formát se jedná je nutné převést časové okamžiky na časová razítka, tak aby bylo možné vyhledávat v databázi. Dále je

nutné se správným způsobem dotázat do databáze. Ukázka dotazu je na na řádku 6 ve výpisu 5.11, kdy jsou požadovány naměřené metriky od časového okamžiku *from* do okamžiku *to*. Následně je výsledek dotazu ve funkci *get_rows* upraven a společně s informacemi jako je IP adresa, název serveru atd. odeslán zpět aplikaci pro vizualizaci. Stejně jak požadavek, tak i odpověď jsou ve formátu JSON.

Výpis 5.11: Ukázka zpracování žádosti range v node.js

```
1 if (request.body["type"] === "range") {
2     const date_from=new Date(request.body["from"])
3     const date_to=new Date(request.body["to"])
4     const query;
5     query = `SELECT * FROM data WHERE timestamp BETWEEN ? AND ?`
6     let params=[date_from.getTime()/1000, date_to.getTime()/1000]
7     db.all(query, params, (err, rows) =>
8         get_rows(err, rows, response, main_message, logg_mesage))
9 }
```

V rámci implementace bylo použito několik externích knihoven. První z nich je knihovna *sqlite3*, která slouží ke čtení dat z databáze. Druhou externí knihovnou je knihovna *cors*, která se stará o správu politik a sdílení zdrojů mezi různými zdroji ve webových aplikacích. Tím, že se na API dotazuje rovnou webová aplikace pro vizualizaci je nutné povolit CORS a k tomu slouží tato knihovna. Další použitou knihonou je knihovna *date-format*, která slouží k jednodušší práci s časovými formáty. Poslední použitou knihovnou je *express*. Express je minimalistický webový aplikační framework pro Node.js, který usnadňuje vytváření webových aplikací a API. Tento framework (knihovna) poskytuje jednoduchý způsob, jak vytvořit serverovou stranu aplikace. Jeho hlavním cílem je zjednodušit práci s HTTP protokolem a různými aspekty tvorby webových aplikací, jako jsou routování nebo zpracování požadavků a odpovědí.

Mimo hlavní funkčnost node.js API serveru je i nějakým způsobem uchovávat informace o činnosti API. Tedy ukládat logy do souboru, tak aby v případě problému, či dalšího užití bylo možné zpětně dohledávat informace o chování aplikace. Systém pro ukládání logů do souboru je součástí Node.js a po definování cesty k souboru se logy zapisují pomocí funkce *console.log()*. Ukázka logů v souboru je ve výpisu 5.12.

Výpis 5.12: Ukázka logu

```
[21/04/23 18:10:33]-INFO-`Running Server on http://10.10.1.2:8080`
[21/04/23 18:10:33]-INFO-`Connected to SQLite database`
[21/04/23 17:22:17]-INFO-`POST / {type:update} Returned_values: 1`
[21/04/23 17:23:09]-INFO-`POST / {type:range} Returned_values: 46`
```

5.2.4 Úprava aplikace pro vizualizaci dat

V rámci práce bylo nutné upravit některé části webové aplikace pro vizualizaci. Původní implementace aplikace totiž nepočítala s přímou komunikací mezi monitorovanými stanicemi a samotnou aplikací. Původní návrh počítal s jakýmsi prostředníkem (kolektor), který bude shromažďovat metriky z monitorovaných zařízení a tyto metriky poté poskytovat aplikaci pro vizualizaci. Od tohoto návrhu bylo upuštěno z důvodu jednoduchosti. Původní návrh by totiž požadoval další zařízení v topologii, proto bylo rozhodnuto, že se tento prvek vynechá a aplikace se bude dotazovat přímo na monitorované stanice.

Aplikace pro vizualizaci dat je napsaná v programovacím jazyce JavaScript a je založena na knihovně React. V rámci této aplikace bylo nutné provést několik změn oproti původní podobě. První změnou bylo předělání způsobu dotazování rovnou na monitorované stanice nikoliv na kolektor. Druhou změnou bylo přidání několika metrik. Původní podoba aplikace podporovala pouze základní metriky (CPU, RAM, bit rate in/out, packet rate in/out a počet navázaných TCP spojení). Bylo tedy nutné upravit aplikaci tak, aby byla schopna zpracovávat a zobrazovat všechny měřené metriky (viz 3.3). Další změnou byla agregace metrik do grafů. Z důvodu přehlednosti bylo nutné upravit grafy, tak aby v jednom grafu bylo více křivek reprezentující metriky. Například je zbytečné mít zvlášť graf pro bit rate IN a zvlášť pro bit rate OUT. Tyto metriky mají společnou jednotku a je možné je zobrazovat v rámci jednoho grafu. Zde je přehled grafů a metrik:

- **CPU/RAM** – Tento graf obsahuje procentuální vytížení procesoru (CPU) a vyrovnávací paměti (RAM).
- **Bitrate** – Graf obsahuje metriky biterate IN a OUT. Jednotkou jsou Kbit/s.
- **Packetrate** – Graf zobrazující metriky packetrate IN a OUT.
- **TCP Statements** – Tento graf zahrnuje nejvíce metrik, tedy všechny které souvisí s TCP protokolem.
- **HTTP Requests** – Graf obsahující pouze jednu metriku - počet přijatých HTTP požadavků.
- **HTTP Packet Size** – V rámci tohoto grafu jsou zahrnuty metriky týkající se velikosti přijatých a odeslaných HTTP datagramů. Jednotkou jsou byty za sekundu.
- **HTTP Time** – Posledním grafem je graf zobrazující vývoj průměrného času, který webový server stráví zpracováním požadavků. Jednotkou jsou sekundy.

Mezi další změny týkají se pružného chodu aplikace, řešení různých nestandardních situací, jako je nedostupnost nějaké stanice. V poslední řadě úprava aplikace zahrnovala i řešení a ladění chyb v rámci celé aplikace.

5.2.5 Řešení implementačních problémů

Mezi implementační problémy patřil výběr databázového systému. InfluxDB je vhodnější pro uchovávání metrik a má mnoho výhod jako jsou např. možnost nastavení automatického mazání nasbíraných dat po nějakém čase. Bohužel, tento systém pracuje v režimu klient-server, což může zapříčinit větší náročnost na výpočetní zdroje monitorovaných zařízení. Oproti tomu SQLite databáze funguje na principu, ukládání dat do souboru a zápis či čtení probíhá přes jednoduché rozhraní, které není spuštěné jako samostatný proces. To dělá z SQLite vhodný databázový systém pro zařízení s malým výkonem nebo pro aplikace, kde není potřeba využívat pokročilé funkcionality. Z toho důvodu bylo rozhodnuto upřednostnit SQLite.

Dalším implementačním problémem bylo nadměrné zatěžování monitorovaných stanic nástrojem top. Tento poměrně jednoduchý nástroj ve full-screen režimu zatěžoval monitorované zařízení více než bylo žádoucí, což by mohl být problém zvláště u zařízení s malým výpočetním výkonem. Nicméně tento problém nepřímo souvisí s implementovaným řešením, protože v rámci implementace je nástroj top používán pouze ke zjištění stavu zatížení CPU a není používán nepřetržitě.

Mezi zásadní implementační problémy patřilo napojení aplikace pro vizualizaci dat k API rozhraní. Hlavním problémem byl rozdílný formát času v aplikaci a na straně API. S tím souvisela i implementace API, tak aby nedocházelo k chybám spojeným například se změnou času nebo rozdílností v časových pásmech mezi aplikací pro vizualizaci a monitorovaným serverem. Tento problém zahrnoval změnit formát používaný v React aplikaci na takový, ve kterém je i informace o časovém pásmu *"2023-05-16 14:39:06 GMT+0200"*. Node.js API po přijetí tohoto formátu převede na časové razítko (Unix timestamp), tak aby byl správně vytvořen dotaz do databáze. Alternativou bylo převádět hodnotu na časové razítko rovnou ve webové aplikaci. Pro přehlednost bylo vybrána první zmíněná varianta.

S dalším implementačním problémem souvisel nastavený jazyk na monitorovaných stanicích. Konkrétně rozdílnost znaku používaném v desetinných číslech v českém (znak *','*) a v anglickém (znak *'.'*) jazyce. Tento problém se projevil v ukládání do databáze u hodnot využívající datový typ *float*. Problém bylo nutné ošetřit tak, aby aplikace nebyla závislá na zvoleném jazyku. Řešení tohoto problému nebylo nikterak složité. Výslednou hodnotu z příkazů pro sběr dat stačilo pomocí funkce zkontrolovat, zda li je s *'.'* (tečka) nebo ne. Pokud není, tak nahradit znak *','* (čárka) znakem *'.'* (tečka). Poté se tato hodnota ukládá do databáze jako hodnota float v rámci scriptu, tak jak bylo popsáno v 5.2.2.

Mezi další implementační problém se dá řadit i čtení z logu z Apache serveru. Program totiž v základním logu neukládá informace jako je délka přijatých žádostí apod. Bylo tedy nutné vytvořit vlastní formát logu (viz 5.2.1).

5.2.6 Testování

V rámci testování byl využit virtuální server, na kterém byl spuštěný webový server Apache. V tomto případě Apache simuluje službu, která je dostupná pro legitimní uživatele. Na serveru byl dále spuštěný vytvořený nástroj, databáze SQLite a Node.js API. Systém byl testovaný za třech odlišných situací.

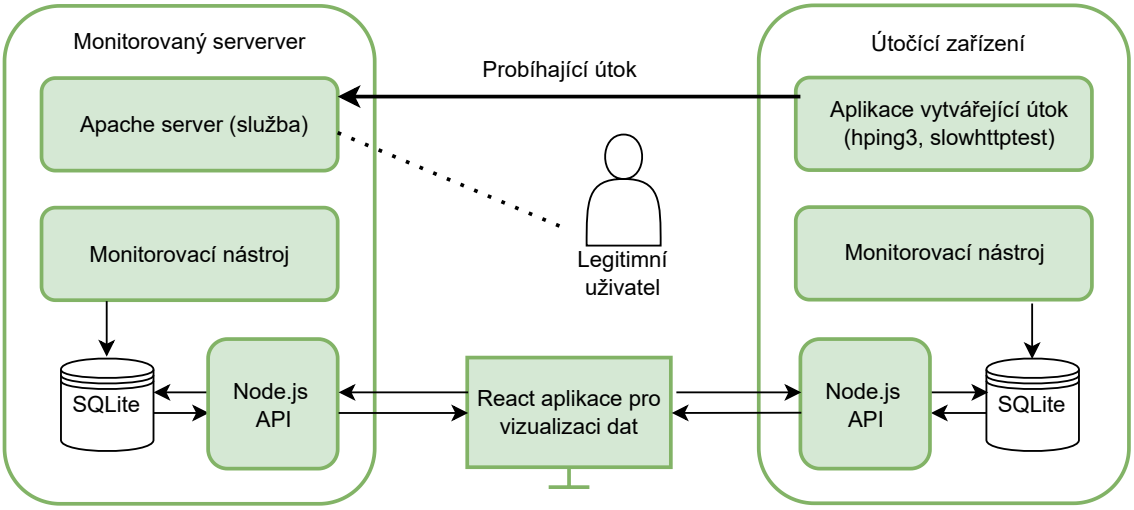
Test 1

V prvním testu bylo monitorováno zařízení za běžného provozu, tedy kdy nedochází k nestandardním situacím nebo k vyčerpání výpočetních prostředků zařízení. Situace je stejná jak na obrázku 5.1.

Během první části testu systém fungoval dle předpokladů, tedy za běžného provozu byly dostupné i aktuálně naměřené hodnoty prostřednictvím modulu *current* v aplikaci pro vizualizaci. Průběh testu je možné prohlédnout na 5.3

Test 2

Při druhém testu (pod DoS útokem) byl využit stejný virtuální server jako u prvního testu a navíc zařízení, ze kterého byl spuštěný záplavový útok (v tomto případě Kali Linux).



Obr. 5.2: Schéma druhého testu

Dále bylo nutné omezit šířku přenosového pásma u monitorovaného serveru tak, aby útočící zařízení mělo rychlejší připojení a byly splněny všechny předpoklady pro záplavový útok. Situace je nastíněna na obrázku 5.2. Typ útoku byl vybrán

záplavový útok SYN Flood. Útok byl konkrétně konstruován pomocí programu **hping3**, příkazem 5.13.

Výpis 5.13: Příkaz pro spuštění SYN Flood útoku.

```
hping3 -S --flood -V -p 80 10.211.55.6
```

Výsledkem druhého testu bylo vyčerpání přenosové kapacity monitorovaného zařízení, tedy velice pomalé načítaní služby legitimním uživatelem, zároveň monitorovaný server pomalu odpovídal i na dotazy týkající monitoringu. Node.js API ze začátku odpovídalo velice pomalu a ve druhé půlce testu nebylo dostupné vůbec. Nicméně i během útoku se metriky ukládaly do SQLite databáze a bylo k nim možné přistupovat po útoku, prostřednictvím modulu *range* v aplikaci pro vizualizaci dat. Konkrétní grafy, tak jak je zachytila aplikace pro vizualizaci jsou na obrázku 5.4. Z grafů je patrné, že během útoku vůbec nedošlo k nějakému většímu zatížení operační paměti a procesoru, přesto služba nebyla dostupná, ani nebylo možné přistupovat k měřeným datům v reálném čase.

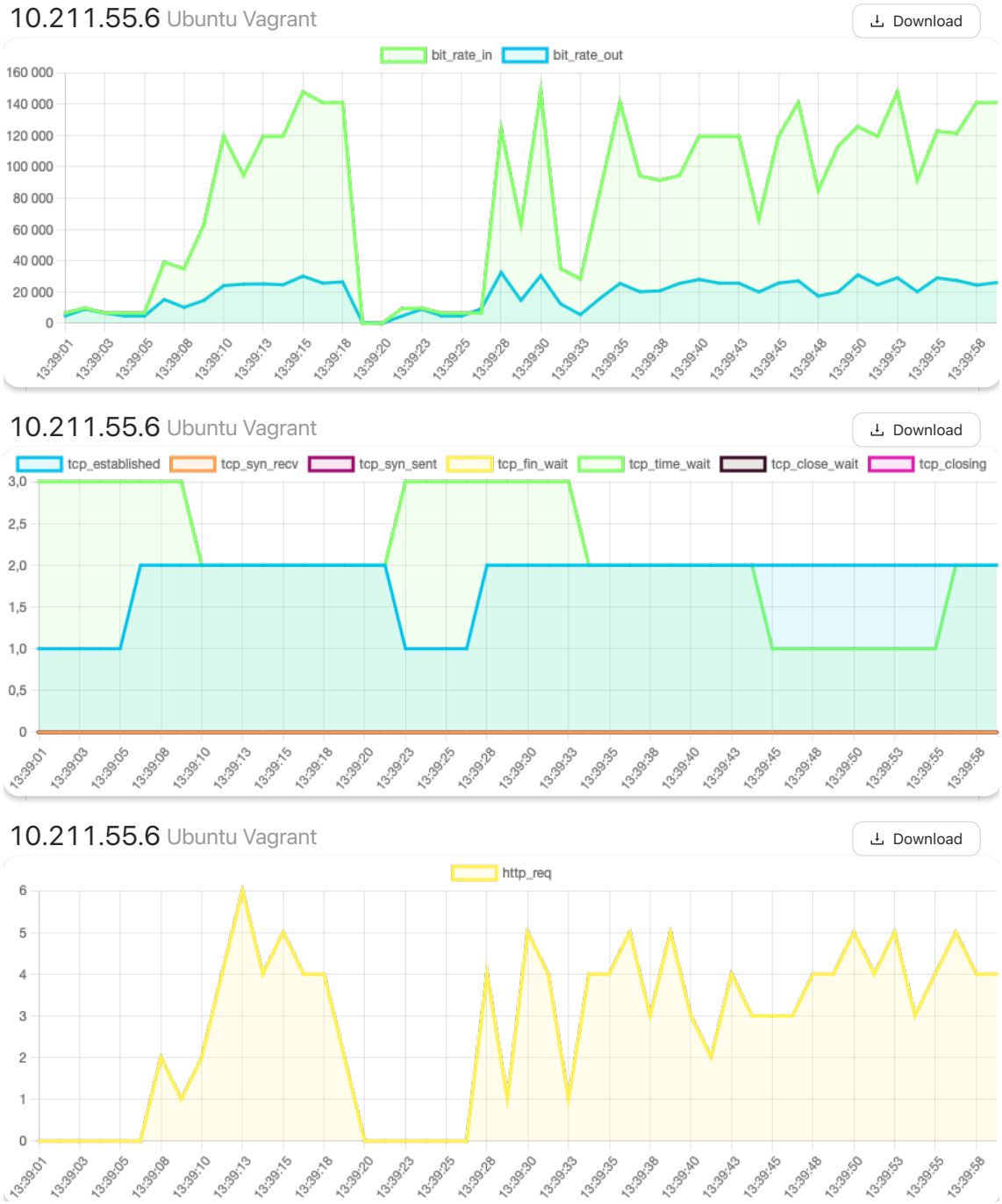
Test 3

V rámci třetího testu je systém testován pod dalším DoS útokem, konkrétně SlowLoris. Situace je velice podobná, jako u druhého testu 5.2. Pro realizaci útoku u tohoto testu byl použit nástroj **slowhttptest**. Spouštěcí příkaz je na výpisu 5.14.

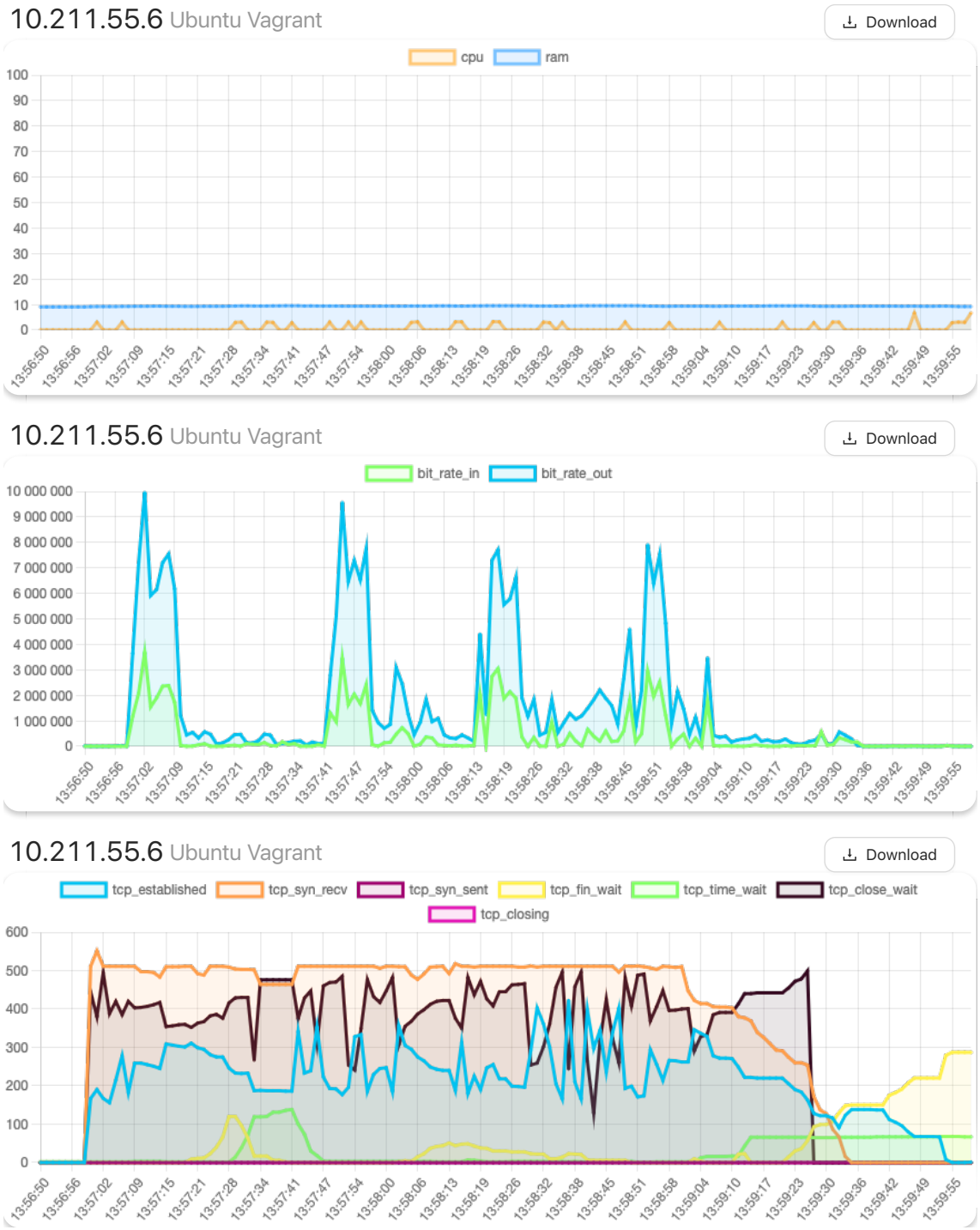
Výpis 5.14: Příkaz pro spuštění SlowLoris útoku.

```
slowhttptest -H -g -o slowloris -u http://10.211.55.6 -c 1500
-i 500 -l 180 -x 1600
```

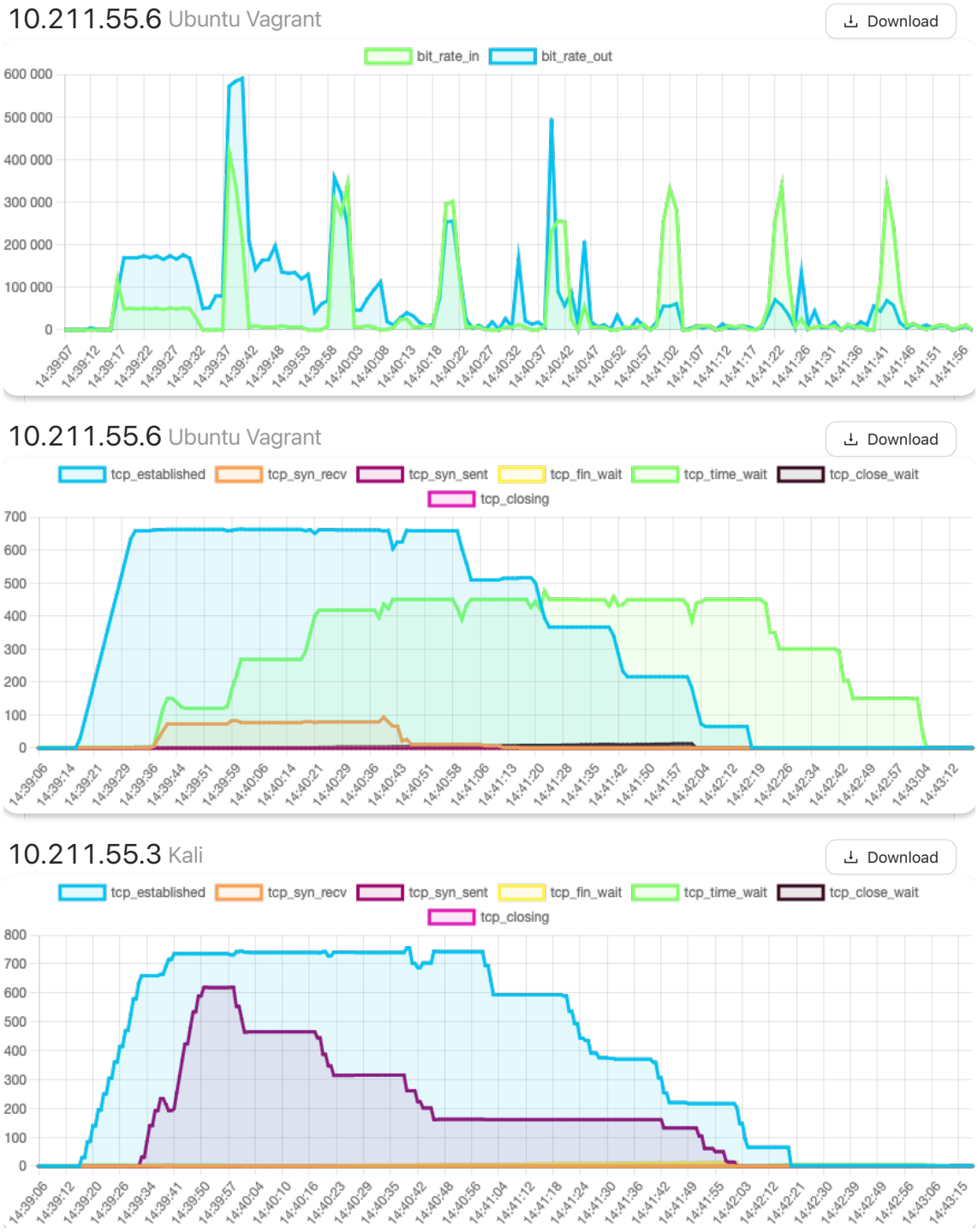
Výsledkem tohoto testu je, že služba byla nedostupná a nebylo nutné dosáhnout ani vytížení CPU a RAM, ani vyčerpat přenosovou kapacitu. Pokud by tedy monitorovací systém spoléhal pouze na tyto metriky, nebyl by SlowLoris útok vůbec detekován, přesto byla služba nedostupná. Tím, že nebyla plně využita kapacita přenosového pásma, bylo možné celou situaci sledovat v reálném čase prostřednictvím modulu *current* v aplikaci pro vizualizaci. Průběh útoku je aplikací zachycen na obrázku 5.5.



Obr. 5.3: Průběh testu 1 - Grafy Bitrate, TCP Statements a HTTP Requests



Obr. 5.4: Průběh testu 2 - Grafy CPU/RAM, Bitrate a TCP Statements



Obr. 5.5: Průběh testu 3 - Grafy Bitrate, TCP Statements u oběti a útočníka

Závěr

Práce se zabývala analýzou dostupného softwaru určeného pro monitoring serverů a návrhem vlastního systému vhodného pro sběr, ukládání a následný přenos měřených dat do již existující aplikace pro vizualizaci těchto dat.

V první kapitole semestrální práce byl popsán model TCP/IP. Byl zde vysvětlen a nastíněn princip a účel každé z vrstvy síťového modelu.

Druhá kapitola se zaobírala principem funkčnosti webových serverů, typy webových serverů a vývojem webových aplikací. Dále se kapitola zaměřovala funkčností protokolu HTTP a jeho verzemi.

Monitorování serveru bylo vysvětleno ve třetí kapitole práce. Byla zde popsána charakteristika a základní pojmy, popis již existujícího softwaru určeného pro monitoring sítě a serverů. Dále zde byly rozebrány měřené metriky, které je vhodné sledovat v rámci monitoringu serverů. Dále zde byly popsány způsoby uchovávání měřených dat. V kapitole jsou také shrnuty důvody proč je dobré monitorovat chování serverů a síťových prvků.

Čtvrtá část se věnovala DoS útokům, jejich charakteristikou, dělením a jednotlivými příklady typů útoku. Následně byla nastíněna úloha monitoringu serverů v detekci DoS útoků. Dále byly probrány důvody a motivace útočníků.

Pátá část se zabývá samotným návrhem, implementaci a testování vytvořeného systému. Ze získaných informací byl vytvořen jednoduchý nástroj pro sběr metrik v systémech Linux. Pro získání metrik byly využity základní a co nejjednodušší linuxové nástroje s ohledem na nízké zatěžování monitorovaných stanic. Následně byla provedena analýza databázových systémů pro ukládání metrik a vzhledem k jednoduchosti a nenáročnosti byla zvolena databáze SQLite. Dále je zde popsán způsob přenosu uchovaných metrik do aplikace pro jejich vizualizaci, pomocí HTTP protokolu, s využitím API, vytvořeného v jazyce JavaScript. V této části jsou uvedeny provedené změny v aplikaci pro vizualizaci dat. Hlavní změny se týkaly hlavně rozšíření základní sady metrik. Dále jsou v kapitole uvedena řešení implementačních problému, které v průběhu vývoje nastaly. Poslední část kapitoly je věnována testování systému za třech odlišných situací.

Výstupem práce je nástroj pro sběr hodnot o stavu serveru. Tyto hodnoty je možné přenášet prostřednictvím API a HTTP protokolu do aplikace ve které je možné sledovat časový vývoj hodnot v příslušných grafech. V této aplikaci je možné zobrazovat hodnoty z více monitorovaných stanic. Nástroj je tedy vhodný pro vizualizaci a zkoumání chování serverů v rámci různých síťových topologiích.

Literatura

[1] *How the Web was born* [online]. [cit. 2022-11-07]. ISBN 978-0-19-286207-5.

[2] What Is a Network Protocol, and How Does It Work?. *CompTIA* [online]. 2021 [cit. 2022-11-05]. Dostupné z: https://www.comptia.org/content/guides/what-is-a-network-protocol

[3] HUNT, Craig. *TCP/IP Network Administration, 3rd Edition* [online]. 3rd. O'Reilly Media, 2002 [cit. 2022-11-02]. ISBN 9780596002978. Dostupné z: https://www.oreilly.com/library/view/tcpip-network-administration/0596002971/

[4] TAWDE, Swati. What is TCP/IP?. *Educba* [online]. 2020 [cit. 2022-11-02]. Dostupné z: https://www.educba.com/what-is-tcp-ip/

[5] Datagram. *Technopedia* [online]. 2011 [cit. 2022-11-2]. Dostupné z: https://www.techopedia.com/definition/6766/datagram

[6] ALIENOR. The Network Layers Explained. *Plixer* [online]. 2018 [cit. 2022-11-06]. Dostupné z: https://www.plixer.com/blog/network-layers-explained/

[7] Network layer. *Educba* [online]. 2020 [cit. 2022-11-07]. Dostupné z: https://www.educba.com/network-layer/

[8] KOISHIGAWA, Kris. TCP vs. UDP. *FreeCodeCamp* [online]. 2021 [cit. 2022-11-05]. Dostupné z: https://www.freecodecamp.org/news/tcp-vs-udp/

[9] User Datagram Protocol. *GeeksForGeeks* [online]. 2022 [cit. 2022-11-05]. Dostupné z: https://www.geeksforgeeks.org/user-datagram-protocol-udp/

[10] What is TCP Protocol?. *Educba* [online]. 2020 [cit. 2022-11-02]. Dostupné z: https://www.educba.com/what-is-tcp-protocol/

[11] Three-Way Handshake. *Technopedia* [online]. 2020 [cit. 2022-11-06]. Dostupné z: https://www.techopedia.com/definition/10339/three-way-handshake

[12] Application Layer. *Technopedia* [online]. 2020 [cit. 2022-11-02]. Dostupné z: https://www.techopedia.com/definition/6006/application-layer

[13] GILLIS, Alexander. Web server. *WhatIs.com* [online]. 2020 [cit. 2022-11-09]. Dostupné z: https://www.techtarget.com/whatis/definition/Web-server

[14] BIRZNIEKS, Gunther. What Is a Web Server?. *ServerWatch* [online]. 2021 [cit. 2022-11-15]. Dostupné z: https://www.serverwatch.com/web-servers

[15] Hypertext Transfer Protocol. *ExtraHop* [online]. 2020 [cit. 2022-11-12]. Dostupné z: https://www.extrahop.com/resources/protocols/http/

[16] GUPTA, Lokesh. HTTP Status Codes. *REST API Tutorial* [online]. 2022 [cit. 2022-11-12]. Dostupné z: https://restfulapi.net/http-status-codes/

[17] FIELDING, R., J. GETTYS, J. MOGUL, H. FRYSTYK, L. MASINTER, P. LEACH a T. BERNERS-LEE. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1. *RFC Editor* [online]. 1999 [cit. 2022-11-12]. Dostupné z: https://www.rfc-editor.org/rfc/rfc2616

[18] Evolution of HTTP. *MDN* [online]. 2022 [cit. 2022-11-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

[19] KETTLE, James. HTTP/2: The Sequel is Always Worse. *PortSwigger* [online]. 2021 [cit. 2022-11-12]. Dostupné z: https://portswigger.net/research/http2

[20] BELSHE, M., R. PEON a M. THOMSON. RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2). *RFC Editor* [online]. 2015 [cit. 2022-11-12]. Dostupné z: https://www.rfc-editor.org/rfc/rfc7540

[21] AWATI, Rahul. Hypertext Transfer Protocol Secure. *TechTarget* [online]. 2022 [cit. 2022-11-12]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/HTTPS

[22] About APACHE. *APACHE - HTTP Server* [online]. 2022 [cit. 2022-11-10]. Dostupné z: https://httpd.apache.org/ABOUT_APACHE.html

[23] FAHIM, Faizan. What is Apache Web Server?. *ServerGuy* [online]. 2022 [cit. 2022-11-15]. Dostupné z: https://serverguy.com/servers/what-is-apache-web-server/

[24] Web Application Development. *Stratoflow* [online]. 2022 [cit. 2022-11-13]. Dostupné z: https://stratoflow.com/web-application-development/

[25] What Is Front End?. *Codecademy* [online]. 2022 [cit. 2022-11-15]. Dostupné z: https://www.codecademy.com/resources/blog/what-is-front-end/

[26] ASTARI, S. Hypertext Markup Language Basics Explained. *Hostinger Tutorials* [online]. 2022 [cit. 2022-11-13]. Dostupné z: https://www.hostinger.com/tutorials/what-is-html

[27] What is HTML. *JavaTpoint* [online]. 2022 [cit. 2022-11-13]. Dostupné z: https://www.javatpoint.com/what-is-html

[28] What is CSS?. *MDN* [online]. 2022 [cit. 2022-11-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

[29] What is CSS. *JavaTpoint* [online]. 2022 [cit. 2022-11-13]. Dostupné z: https://www.javatpoint.com/what-is-css

[30] What is JavaScript?. *MDN* [online]. 2022 [cit. 2022-11-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[31] BREWSTER, Cordenne. JavaScript Frameworks. *Trio blog* [online]. 2022 [cit. 2022-11-15]. Dostupné z: https://www.trio.dev/blog/javascript-framework

[32] What Is Back End?. *Codecademy* [online]. 2022 [cit. 2022-11-15]. Dostupné z: https://www.codecademy.com/resources/blog/what-is-back-end/

[33] About node.js. *Node.js* [online]. 2022 [cit. 2022-11-15]. Dostupné z: https://nodejs.dev/en/about/

[34] MÁCA, Jindřich. Úvod do Node.js. *Itnetwork* [online]. 2021 [cit. 2022-11-14]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs

[35] Introduction to Node.js. *Node.js* [online]. 2022 [cit. 2022-11-10]. Dostupné z: https://nodejs.dev/en/learn/

[36] What Is Server Monitoring?. *Splunk.com* [online]. 2020 [cit. 2022-11-12]. Dostupné z: https://www.splunk.com/en_us/data-insider/what-is-server-monitoring.html

[37] INGALLS, Sam. What Is A Virtual Server?. *ServerWatch* [online]. 2021 [cit. 2022-11-15]. Dostupné z: https://www.serverwatch.com/guides/virtual-server/

[38] MAHMUD, Al. Server Performance Monitoring. *ServerWatch* [online]. 2021 [cit. 2022-11-17]. Dostupné z: https://www.serverwatch.com/guides/server-performance-monitoring-guide/

[39] How and Why to Monitor Server CPU Usage. *SenitnelOne* [online]. 2021 [cit. 2022-11-22]. Dostupné z: https://www.sentinelone.com/blog/how-and-why-to-monitor-server-cpu-usage/

[40] VILLINGER, Sandro. What Is RAM and Why Is It Important?. *Avast* [online]. 2022 [cit. 2022-11-22]. Dostupné z: https://www.avast.com/c-what-is-ram-memory

[41] Bandwidth, Packets Per Second, and Other Network Performance Metrics. *Cisco* [online]. [cit. 2022-11-17]. Dostupné z: https://tools.cisco.com/security/center/resources/network_performance_metrics.html

[42] Bit Rate. *Technopedia* [online]. 2018 [cit. 2022-11-17]. Dostupné z: https://www.techopedia.com/definition/2681/bit-rate-br

[43] Database Management System. *Technopedia* [online]. 2022 [cit. 2022-11-13]. Dostupné z: https://www.techopedia.com/definition/24361/database-management-systems-dbms

[44] SQL vs NoSQL: when to use?. *ImaginaryCloud* [online]. 2021 [cit. 2022-11-13]. Dostupné z: https://www.imaginarycloud.com/blog/sql-vs-nosql/

[45] Time series database (TSDB) explained. *Influxdata* [online]. [cit. 2022-11-13]. Dostupné z: https://www.influxdata.com/time-series-database/

[46] NSRAV. Denial of Service. *The Open Web Application Security Project* [online]. [cit. 2022-11-20]. Dostupné z: https://owasp.org/www-community/attacks/Denial_of_Service

[47] What is a denial-of-service attack?. *Cloudflare* [online]. [cit. 2022-11-29]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/

[48] FRANKEDNFIELD, Jake. Denial-of-Service (DoS) Attack. *Investopedia* [online]. 2022 [cit. 2022-11-19]. Dostupné z: https://www.investopedia.com/terms/d/denial-service-attack-dos.asp

[49] What is an ICMP Flood Attack?. *Netscout* [online]. [cit. 2022-11-29]. Dostupné z: https://www.netscout.com/what-is-ddos/icmp-flood

[50] UDP flood. *Ionos* [online]. 2020 [cit. 2022-11-29]. Dostupné z: https://www.ionos.com/digitalguide/server/security/udp-flood/

[51] Ping of Death. *Fortinet* [online]. 2022 [cit. 2022-11-28]. Dostupné z: https://www.fortinet.com/resources/cyberglossary/ping-of-death

[52] TCP SYN Flood. *Imperva.com* [online]. [cit. 2022-11-29]. Dostupné z: https://www.imperva.com/learn/ddos/syn-flood/

[53] Slowloris. *Imperva.com* [online]. [cit. 2022-11-28]. Dostupné z: https://www.imperva.com/learn/ddos/slowloris/

Seznam symbolů a zkratek

TCP	Transmission Control Protocol
IP	Internet Protocol
ICMP	Internet Control Message Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SMTP	Simple Mail Transfer Protocol
DNS	Domain Name System
IPTV	Internet Protocol television
VoIP	Voice over Internet Protocol
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SSL	Secure Sockets Layer
TLS	Transport Layer Security
IT	Information Technology
CPU	Central Processing Unit
RAM	Random Access Memory
SQL	Structured Query Language
NoSQL	Not Only Structured Query Language
TSDB	Time Series Database
JSON	JavaScript Object Notation
DoS	Denial of Service
API	Application Programming Interface

A Obsah elektronické přílohy

Elektronická příloha obsahuje soubory ke sběru a přenášení dat. Dále je v příloze adresář s aplikací pro vizualizaci. Soubory pro sběr se nacházejí v adresáři **server-monitoring/scripts**. Soubor s názvem **server.js** v adresáři **server-monitoring** obsahuje zdrojový kód k nástroji na přenos dat.

```
/ ..... kořenový adresář přiloženého archivu
├── react-monitoring-app. ..... adresář obsahující aplikaci pro vizualizaci dat
├── server-monitoring
│   ├── scripts
│   │   ├── initial_script.sh ..... script pro instalaci základních balíčků
│   │   ├── server-monitoring.sh. ..... script pro sběr a ukládání hodnot
│   │   └── service.txt ..... šablona pro vytvoření služby pomocí systemctl
│   ├── config.json ..... konfigurační soubor Node.js API
│   ├── functions.js
│   ├── node-service.txt. ..... šablona pro vytvoření služby pomocí systemctl
│   ├── package.json
│   ├── package-lock.json
│   ├── readme.md ..... návod na spuštění
│   └── server.js. ..... zdrojový kód Node.js API
└── navod.pdf ..... návod na spuštění
```