

Fakulta informatiky a informačných technológií

FIIT-104199-79816

Michal Záhradník

Rozšírenie prostriedku katalyzer.sk pre kontinuálne meranie v IP sieťach

Bakalárska práca

Študijný program: Internetové technológie

Študijný odbor: 9.2.4 Počítačové inžinierstvo

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky,

FIIT STU, Bratislava

Vedúci práce: Ing. Tomáš Kováčik, PhD.

Máj 2019

ZADANIE BAKALÁRSKEHO PROJEKTU

Meno študenta: **Záhradník Michal**
Študijný odbor: Počítačové inžinierstvo
Študijný program: Internetové technológie
Názov projektu: **Rozšírenie prostriedku katalyzer.sk pre kontinuálne meranie v IP sieťach**

Zadanie:

Monitorovanie premávky je dôležitou súčasťou administrácie a prevádzky každej bezpečnej počítačovej siete. Existujúci merací prostriedok KaTaLyzer.sk je určený na kontinuálne semi-real-time meranie prevádzky v lokálnej počítačovej sieti.

Analyzujte možnosti meracieho prostriedku KaTaLyzer. Navrhnite zlepšenia či rozšírenia tohto meracieho prostriedku (napr. monitorovacia sonda pre Windows systémy či smerovače umožňujúce implementovať v nich rozšírenia; sledovanie stavu siete, zmien a udalostí v sieti; optimalizácia prostriedku; vylepšenia inšpirované inými meracími prostriedkami). Navrhnuté zlepšenia implementujte a implementované riešenie otestujte, vyhodnoťte a zverejnite ako ďalšiu verziu KaTaLyzeru.

Práca musí obsahovať:


Anotáciu v slovenskom a anglickom jazyku
Analýzu problému
Opis riešenia
Zhodnotenie
Technickú dokumentáciu
Zoznam použitej literatúry
Elektronické médium obsahujúce vytvorený produkt spolu s dokumentáciou

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU, Bratislava
Vedúci projektu: Ing. Tomáš Kováčik, PhD.

Termín odovzdania práce v letnom semestri 7. mája 2019

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4
IČO: 00397687, DIČ: 2020845255
IČ DPH: SK2020845255

Bratislava 11.02.2019


Ing. Katarína Jelemenská, PhD.
riaditeľka UPAI

Čestne vyhlasujem, že som túto prácu vypracoval samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave, 7.5.2019

.....

Michal Záhradník

Pod'akovanie

Touto cestou chcem pod'akovať vedúcemu práce, Ing. Tomášovi Kováčikovi PhD., za cenné rady a pripomienky, ktoré mi poskytol pri vypracovávaní práce.

Anotácia

Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLOGIÍ

Študijný program: Internetové technológie

Autor: Michal Záhradník

Bakalárska práca: Rozšírenie prostriedku katalyzer.sk pre kontinuálne meranie v IP sieťach

Vedúci bakalárskej práce: Ing. Tomáš Kováčik, PhD.

Máj 2019

Táto práca sa zaoberá problematikou monitorovania sieťovej prevádzky na zariadeniach s operačným systémom Windows alebo Linux. Analyzuje funkcionality a vlastnosti existujúcich prostriedkov určených na monitorovanie sieťovej prevádzky v reálnom čase, vrátane prostriedku KaTaLyzer. Poukazuje na nedostatky monitorovacieho prostriedku KaTaLyzer a dôvody na jeho pretvorenie pomocou modernejších technológií. Obsahuje funkcionálne a nefunkcionálne požiadavky nového riešenia. Popisuje postup, akým boli jednotlivé technológie porovnávané a zvolené ako vhodné na pretvorenie pôvodného riešenia monitorovacieho prostriedku KaTaLyzer. Obsahuje návrh nového riešenia, popisuje jeho implementáciu a spôsob akým bolo riešenie otestované.

Výsledkom práce je plne funkčný a otestovaný prostriedok schopný monitorovať sieťovú prevádzku v reálnom čase na platformách s operačným systémom Windows alebo Linux.

Monitorovaním sieťovej prevádzky pomocou prostriedku KaTaLyzer je možné odhaliť problémovú sieťovú komunikáciu na zariadeniach, kde je prostriedok nainštalovaný. Po takomto odhalení má zodpovedná osoba, na základe závažnosti, možnosť vykonať príslušnú akciu alebo prijať protipatrenie, aby sa komunikácia alebo problém na zariadení už nevyskytli.

Annotation

Slovak University of Technology Bratislava

FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Degree course: Internet Technologies

Author: Michal Záhradník

Bachelor's Thesis: Extension of katalyzer.sk for continual measurement in IP networks

Supervisor: Ing. Tomáš Kováčik, PhD.

May 2019

This thesis deals with monitoring network traffic on both Windows and Linux devices. It analyzes the functionality and features of existing real-time network traffic monitoring solutions, including KaTaLyzer. It points out the shortcomings of the monitoring tool KaTaLyzer and the reasons for its recreation using modern technologies. It contains the functional and non-functional requirements of the new solution. It describes how the individual technologies were compared and chosen as suitable to recreate the original solution of the KaTaLyzer. It contains a design for a new solution, describes its implementation and how the solution was tested.

The result of this thesis is a fully functional and tested tool capable of monitoring real-time network traffic on both Windows and Linux devices.

By monitoring network traffic with monitoring tool KaTaLyzer, it is possible to detect problematic network communications on devices where the tool is installed. Upon such disclosure, the responsible person can take appropriate actions or take countermeasures, based on seriousness, to prevent such communication or problem on device.

Obsah

1	Úvod.....	1
2	Analýza.....	3
2.1	Analýza existujúceho riešenia KaTaLyzer	3
2.2	Monitorovacie prostriedky	6
2.3	Analýza dostupných technológií.....	8
3	Špecifikácia	17
3.1	Funkcionálne požiadavky	17
3.2	Nefunkcionálne požiadavky.....	18
4	Návrh riešenia	19
4.1	Sonda.....	19
4.2	ELK Stack.....	26
4.3	Porovnanie s pôvodným riešením.....	27
4.4	Výsledná architektúra	29
5	Implementácia	31
5.1	Sonda.....	31
5.2	Konfigurácia ELK Stack.....	36
6	Overenie riešenia.....	37
6.1	Testovanie	37
6.2	Porovnanie výsledkov s nástrojom Wireshark.....	39
	Zhodnotenie	41
	Literatúra	43
	Príloha A: Plán práce - zimný semester	1
	Príloha B: Plán práce - letný semester.....	2
	Príloha C: Používateľská príručka.....	3
	Príloha D: Opis digitálnej časti práce	7

Zoznam obrázkov

Obrázok 1 Grafické rozhranie nástroja KaTaLyzer	4
Obrázok 2 Diagram popisujúci spustenie sondy	20
Obrázok 3 Diagram popisujúci hlavné menu	21
Obrázok 4 Diagram popisujúci monitorovanie sieťovej prevádzky.....	22
Obrázok 5 Koniec diagramu.....	23
Obrázok 6 Návrh triedy pre štatistický záznam	24
Obrázok 7 Dátový model pre štatistický záznam	26
Obrázok 8 Dátový model pre logovací záznam	27
Obrázok 9 Architektúra systému	30
Obrázok 10 Štruktúra riešenia sondy	31

Zoznam tabuliek

Tabuľka 1 Rozdelenie sondy na projekty podľa .NET framework	19
Tabuľka 2 Testovacie konfigurácie pre sondy	37
Tabuľka 3 Testovanie spúšťacích argumentov.....	38
Tabuľka 4 Porovnanie s programom Wireshark	39

Zoznam skratiek a pojmov

ARP	Address Resolution Protocol, protokol slúžiaci na preklad IP adres na MAC
CDP	Cisco Discovery Protocol, protokol určený na šírenie identity CISCO zariadení
Ethernet	Typ počítačovej siete
FTP	File Transfer Protocol, protokol určený na prenos dát
gQUIC	Protokol od spoločnosti Google, ktorý prenáša HTTPS správy s využitím UDP
HTTP	Hypertext Transfer Protocol, protokol určený na prenos dát medzi webovými klientmi a servermi
HTTPS	Verzia protokolu HTTP, ktorá je zabezpečená šifrovaním
ICMP	Internet Control Message Protocol, protokol slúžiaci na prenos kontrolných správ o stave zariadení
IEEE 802.1q	Štandard na označovanie Ethernet rámcov informáciami o VLAN
IEEE 802.3	Set štandardov definujúcich sieť Ethernet
IGMP	Internet Group Management Protocol, protokol slúžiaci na vytvorenie multicast
IP	Internet Protocol, protokol používaný na prenos údajov mimo lokálnu sieť
IPX	Internetwork Packet Exchange, internetový protokol, ktorý sa používal na operačných systémoch Novell NetWare
JSON	JavaScript Object Notation, formát ktorý používajú niektoré služby pri prenose dát
LLDP	Link Layer Discovery Protocol, protokol používaný na šírenie identity sieťových zariadení
MAC	Media Access Control, je jedinečné označenie sieťového rozhrania
OSPF	Open Shortest Path First, smerovací protokol
pcap	Aplikačné rozhranie na zachytávanie sieťovej prevádzky na zariadení
RARP	Reverse Address Resolution Protocol, protokol ktorý sa používa na preklad adres MAC na IP

SIP	Session Initiation Protocol, protokol slúžiaci na riadenie multimediálnych spojení
SLL	Protokol používaný knižnicou libpcap
SQL	Structured Query Language, jazyk používaný v relačných databázach
SSH	Secure Shell, protokol určený na prístup k vzdialenému zariadeniu
TCP	Transmission Control Protocol, spojovo orientovaný transportný protokol
TCP/IP	Sieťový model, ktorý rozdeľuje protokoly do štyroch vrstiev
UDP	User Datagram Protocol, rýchly ale nespoľahlivý transportný protokol
WoL	Wake on LAN, technológia na prebudenie počítača v lokálnej sieti
XML	eXtensible Markup Language, je značkovací jazyk

1 Úvod

Internet a sieťová komunikácia sa stali každodennou súčasťou našich životov. Dnes už len veľmi ťažko nájdeme miesta, kde by sme boli mimo dosahu internetového pripojenia. Internetové pripojenie sa stalo neoddeliteľnou súčasťou každodenného života, nástrojom využívaným pri riadení a chode organizácií, ovplyvňuje takmer všetky aspekty spoločnosti.

S rastúcim počtom používateľov siete internet rastie aj nebezpečenstvo pri jeho používaní. Pre účel zvýšenia bezpečnosti pri používaní siete internet začali vznikať prostriedky určené na monitorovanie sieťovej prevádzky a sieťových zariadení. Tieto majú za úlohu odhaliť v sieti rôzne chyby, nedostatky alebo blížiacu sa hrozbu a dať možnosť autorite zodpovednej za bezpečnosť včas zakročiť.

V tejto práci sa budeme zaoberať monitorovacím prostriedkom KaTaLyzer, ktorý vznikol pre účel monitorovania sieťovej prevádzky v reálnom čase. Budeme analyzovať technológie, ktoré boli použité pri jeho tvorbe a funkcionality, ktorú ponúka. Následne na základe výsledkov analýzy navrhujeme a implementujeme vylepšenie pre monitorovací prostriedok KaTaLyzer.

2 Analýza

2.1 Analýza existujúceho riešenia KaTaLyzer

Témou bakalárskej práce je navrhnuť a implementovať rozšírenie alebo zlepšenie do existujúceho meracieho prostriedku KaTaLyzer. Pre tento účel budeme najskôr analyzovať existujúce riešenie. V analýze sa zameriame na použité technológie a funkcionality. Potom navrhujeme vylepšenie pre KaTaLyzer, ktoré implementujeme.

KaTaLyzer je projekt, ktorého účel je monitorovanie sieťovej prevádzky na sieťovom rozhraní. Aktuálne je podporovaný len na operačných systémoch s Linuxovým jadrom. Kľúčové časti KaTaLyzera sú databáza, front-end a sonda (backend) merajúca prevádzku na uzle, kde je spustená jej inštancia a konfiguračný súbor.

2.1.1 Sonda

V projekte pod pojmom sonda rozumieme program, ktorý na uzle monitoruje sieťovú prevádzku na vybraných sieťových rozhraniach. Sonda v stanovených intervaloch získané informácie ukladá do relačnej databázy. Jej funkcionality závisí od konfiguračného súboru. V súčasnosti dokáže monitorovať prevádzku na základe týchto protokolov:

- Ethernet 2
- IEEE 802.3
- ARP
- RARP
- IP
- IGMP
- ICMP
- IPX
- TCP
- UDP

Sonda je vytvorená v jazyku C. Zdrojový kód, ktorý sondu tvorí, je napísaný veľmi nečitateľne. Dlhé metódy, veľa blokov kódu, nezrozumiteľné komentáre, zle zvolené názvy funkcií a premenných sťažujú orientáciu v zdrojovom kóde a pridávanie novej funkcionality do riešenia. Už pri statickej analýze kódu sme spozorovali veľké množstvo rovnakých blokov kódu. Zdrojový kód javí znaky nízkej údržby a dôraz bol pravdepodobne kladený viac na

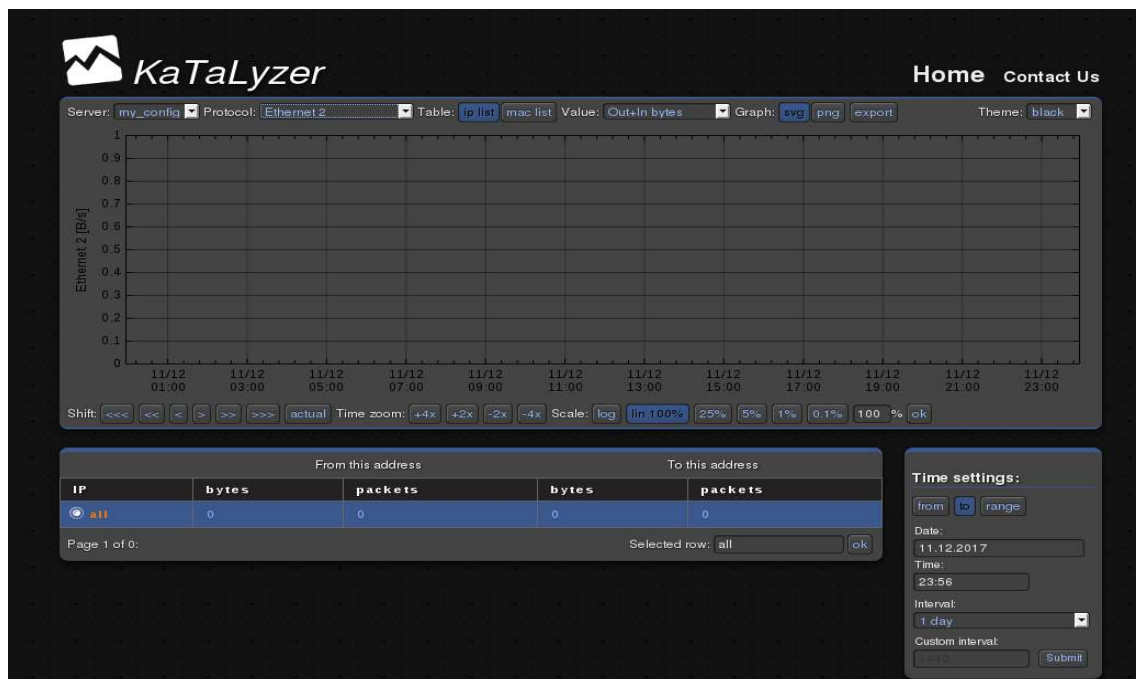
výsledok, ako na pozadie projektu. Kvôli nulovej čitateľnosti zdrojového kódu, by bolo pracovať s takýmto projektom v praxi takmer nemožné.

Identifikovali sme aj problém v implementácii ukladania zozbieraných štatistík zo sondy. Sonda posiela SQL dotazy na relačnú databázu. Takýto prístup sa z bezpečnostných dôvodov už neodporúča používať. Namiesto toho sa vytvárajú aplikačné rozhrania, ku ktorým používateľ nemá prístup a tie komunikujú s databázou.

2.1.2 Front-end

Je to webová stránka vytvorená pomocou skriptovacieho jazyka PHP verzie 5. Hlavnou úlohou stránky je vizualizácia dát, ktoré uložila sonda do databázy. Uložené dáta je možné filtrovať na základe času, protokolu a IP/MAC zdrojových a cieľových adries podľa potrieb používateľa.

Pri skúmaní implementácie sme narazili na niekoľko nedostatkov. Zistili sme, že verzia jazyka, v ktorej je stránka vytvorená, už nie je podporovaná. Aktuálne používaná verzia jazyka PHP je 7.3, ktorá ma aktívnu podporu do decembra roka 2020 (1). Ak by sme chceli vyvíjať projekt ďalej, bolo by pre nás najvyššou prioritou tento nedostatok odstrániť. To pre nás predstavuje mnoho práce navyše. Museli by sme robiť zásah do zdrojových kódov stránky a nahradiť volania, ktoré nová verzia jazyku už neobsahuje, novými.



Obrázok 1 Grafické rozhranie nástroja KaTaLyzer

2.1.3 Databáza

Sonda aj front-end pracujú s relačnou databázou MySQL. Pre každý protokol v databáze existuje päť základných druhov tabuliek:

- Time - čas zachytenia
- IP - IP adresy a počet bytov a paketov
- MAC - MAC adresy a počet bytov a paketov
- IP_SD - ktoré IP adresy medzi sebou komunikovali
- MAC_SD - ktoré MAC adresy medzi sebou komunikovali

Každá takáto tabuľka ma časový prefix. Tento prefix určuje úroveň sumarizácie tabuľky. Pred prefixom sa nachádza názov protokolu. Napríklad, TCP_1m_MAC je názov tabuľky pre protokol TCP, do ktorého sú ukladane záznamy o zdrojovej a cieľovej MAC adrese komunikácie, ktoré boli zozbierané počas jednej minúty.

Pri podrobnejšom skúmaní databázy sme odhalili výrazný nedostatok v štruktúre dát. Databáza totiž neobsahovala žiadne cudzie ani primárne kľúče. Spôsob, akým bola databáza navrhnutá, by po dlhšej dobe viedol k veľkému spomaleniu fungovania riešenia.

2.1.4 Konfiguračný súbor

Front-end aj sonda potrebujú pre fungovanie konfiguračný súbor. Pomocou konfiguračného súboru sa nastavujú prístupové údaje do databázy, sieťové rozhrania a protokoly, ktoré má sonda monitorovať. Riadky začínajúce znakom „#“ sú ignorované. Takéto riadky predstavujú komentáre.

Každé nastavenie sa začína prefixom príslušnej kategórie, po ktorom nasleduje názov konkrétneho nastavenia. Potom cez znak “=” priradíme hodnotu pre nastavenie. V zozname nižšie sú uvedené využívané nastavenia sondy:

- DB – Definuje kategóriu nastavení pre prístupy do databázy
 - DB_HOST – adresa na ktorej je databáza spustená
 - DB_PORT – port na ktorom je spustená databáza
 - DB_NAME – meno databázy
 - DB_USER – používateľ pod ktorým sa chceme prihlásiť
 - DB_PASS – heslo do databázy
- INTERFACE – Definuje sieťové rozhranie pre monitorovanie sieťovej prevádzky

- **PROTOCOL** – Definuje nastavenia protokolov
 - obsahuje viacero protokolov
 - všetky protokoly sú defaultne povolené
 - manuálne povolíme cez `PROTOCOL_[skratka protokolu]=1`
 - manuálne zakážeme cez `PROTOCOL_[skratka protokolu]=0`
 - skratky protokolov môžu byť: ETH, 8023, SLL, ARP, RARP, IP, IGMP, ICMP, IPX, TCP, UDP, CDP, SIP
- **TCP**
 - Definuje ktoré porty TCP protokolu budeme monitorovať
- **UDP**
 - Definuje ktoré porty UDP protokolu budeme monitorovať

Aj ku konfiguračnému súboru máme výhrady. Najzávažnejším problémom je skutočnosť, že obsahuje prístupové údaje k databáze. Toto riešenie predstavuje bezpečnostné riziko. Ďalšou menšou výhradou je použitie vlastného formátu súboru, ktorý bolo nutné naprogramovať. Bolo by vhodnejšie použiť štandardnejší formát súboru, ako napríklad XML alebo JSON.

2.1.5 Výsledok analýzy existujúceho riešenia

Na stave projektu KaTaLyzer sa podpísali dlhé roky bez údržby. Použité technológie a celkový návrh riešenia sú už zastaralé. Zdrojový kód nespĺňa mnoho zásad pre dosiahnutie čitateľnosti. Tieto dôvody nás doviedli k záveru, že projekt treba prepracovať pomocou moderných technológií s väčším dôrazom na kvalitu zdrojového kódu. Preto sme sa rozhodli vo vývoji existujúceho riešenia nepokračovať a jeho myšlienku pretvoriť pomocou modernejších technológií.

2.2 Monitorovacie prostriedky

Okrem monitorovacieho prostriedku KaTaLyzer sa v praxi využívajú aj iné riešenia s podobnou funkcionalitou.

2.2.1 Wireshark

Tento monitorovací prostriedok patrí k najrozšírenejším na svete. Je vyvíjaný ako open-source projekt. Wireshark je podporovaný na platformách Windows, Linux, Unix, macOS. Dokážeme pomocou neho analyzovať sieťovú premávku na rozhraniach zariadenia, kde je nainštalovaný. Pri analýze rámcov nám Wireshark poskytne všetky informácie o jednotlivých

protokoloch. Podporuje všetky známe protokoly a typy sietí, s ktorými sa môžeme v praxi stretnúť. Pre svoje fungovanie potrebuje pcap knižnice. V prípade platformy Windows to je Npcap. Na platforme Linux pracuje s knižnicou libpcap (2).

Wireshark je jedným z najúčinnějších monitorovacích prostriedkov, ktorý využívajú nielen administrátori počítačových sietí, ale aj programátori pri odhaľovaní problémov spojených so sieťovou prevádzkou. Nevýhodou prostriedku Wireshark je, že nie je stavaný na dlhodobú prevádzku bez vonkajšieho zásahu.

2.2.2 *Tcpdump*

Tcpdump je prostriedok slúžiaci na zachytávanie sieťovej prevádzky na operačných systémoch s Linux jadrom. Pri práci používa knižnicu libpcap na zachytávanie sieťovej prevádzky. Výstup z programu môžeme po spustení vidieť v termináli. Prostriedok sme mali možnosť vyskúšať a v porovnaní s nástrojom Wireshark disponuje omnoho menším počtom funkcií.

2.2.3 *SolarWinds NetFlow Traffic Analyzer*

NetFlow Traffic Analyzer je rozšírenie produktu Network Performačne Monitor určeného na monitorovanie zariadení v sieti. Riešenie je vyvíjane spoločnosťou SolarWinds. Najlacnejšia licencia produktu Network Performačne Monitor obsahujúca NetFlow Traffic Analyzer stojí štyritisíc eur. Táto verzia nám dovoľuje monitorovať sto zariadení (3).

Ponúka nám monitorovanie sieťových prvkov pomocou protokolov NetFlow, J-Flow, sFlow, IPFIX, NetStream a iných (4). Získané informácie zo zariadení produkt analyzuje a na základe zistení vytvára vizualizácie o vyťaženosti siete, použitých protokoloch, aplikáciách a rôzne ďalšie. Okrem vizualizácií dokáže pri zadaných nameraných hodnotách odosielať varovné správy. Za veľkú nevýhodu riešenia považujeme absenciu monitorovania sieťovej prevádzky priamo na zariadeniach s operačným systémom Linux alebo Windows.

2.2.4 *PRTG Network Monitor*

Nástroj určený na monitorovanie siete od spoločnosti Paessler. Monitorovanie v prostriedku vykonávajú senzory. Senzor je proces, ktorý sa zameriava na určitú množinu informácií. Pomocou senzorov môžeme monitorovať napríklad dostupnosť alebo vyťaženosť zariadení a samotnú sieťovú prevádzku. Prostriedok nám tiež dáva možnosť senzoru zadať správanie pomocou skriptu a nastavenie odosielania varovných správ pri určitých

nameraných hodnotách. Zaujímavý bol pre nás pravé senzor na monitorovanie sieťovej prevádzky, preto sme monitorovací prostriedok nainštalovali a vyskúšali.

Inštalácia nástroja bola veľmi rýchla a jednoduchá. Pri vytváraní senzora na zachytávanie sieťovej prevádzky sme v sprievodcovi mali možnosť určiť sieťové adaptéry a protokoly, ktoré bude senzor monitorovať. Senzor podporuje dvadsaťjeden protokolov. Okrem protokolov zachytených rámcov zobrazuje senzor informáciu o veľkosti komunikácie medzi dvoma IP adresami. Prostriedok umožňuje monitorovať prevádzku priamo na zariadení, kde je nainštalovaný, alebo monitorovať sieťové prvky pomocou protokolu NetFlow.

Prostriedok je plne dostupný len na platforme Windows. Jeho cena závisí od počtu aktívnych senzorov. Spoločnosť ponúka typ licencie, ktorý je zadarmo s limitom sto senzorov. Cena najlacnejšie platenej verzie na jeden rok je tisícvesto eur. Táto licencia umožňuje využívať až päťsto senzorov (5).

2.3 Analýza dostupných technológií

V kapitole sa budeme venovať analýze dostupných technológií, programovacích jazykov a knižníc, ktoré považujeme za vhodné na pretvorenie projektu. Na výber sme mali hneď niekoľko možností, ako môžeme projekt orientovať. Rozhodli sme sa vytvoriť riešenie, ktoré bude natívne a schopné fungovať na platformách Linux aj Windows. Pri výbere technológií sme museli zohľadniť ich individuálne charakteristiky, aby sme mohli dospieť k čo najlepším výsledkom. Na základe porovnania vyberieme technológie pre tvorbu sondy na zachytávanie prevádzky, úložisko pre zozbierané dáta a rozhranie pre ich vizualizáciu.

2.3.1 Aplikácia na zachytávanie sieťovej prevádzky

V súčasnosti máme možnosť výberu zo širokého spektra rôzne fungujúcich programovacích jazykov. Pri výbere sme zohľadňovali najmä nasledujúce kritériá:

1. Multiplatformovosť (Podpora pre Linux a Windows)
2. Aktivita, podpora a komunita
3. Vlastné skúsenosti s daným programovacím jazykom
4. Systémové nároky

Multiplatformovosť je pre nás veľmi dôležitým parametrom, pretože chceme aby nové riešenie bolo spustiteľné na čo najväčšom počte zariadení. Na druhom mieste je kritérium charakterizujúce životný cyklus technológie. Budeme sa predovšetkým snažiť pracovať s modernými technológiami, ktoré majú aktívnu podporu a komunitu. Na treťom mieste je

kritérium zahŕňajúce vlastné skúsenosti s jazykom. Aby sa nám so zvolenou technológiou dobre pracovalo a aby sme ju čo najlepšie využili je potrebné ju podrobne poznať. Na štvrtom mieste sa umiestnili systémové nároky. Na základe existujúcej sondy odhadujeme nízke nároky na réžiu systémových prostriedkov počas prevádzky, preto na toto kritérium nekladíme veľký dôraz. V porovnaní ho použijeme iba v prípade, že si nebudeme vedieť z kandidátov vybrať iba na základe zohľadnenia ostatných kritérií.

Zostavili sme zoznam technológií, ktoré považujeme za vhodné na tvorbe sondy:

- C
- C++
- Python
- Java
- .NET

Programovací jazyk C

Nízkoúrovňový programovací jazyk, ktorý navrhol a vytvoril Dennis Ritchie s tímom v Bell Labs medzi rokmi 1972 a 1973. V roku 1989 v American National Standard Institute (ANSI) schválili prvý štandard programovacieho jazyka C. V roku 1990 bol tento štandard s malými modifikáciami prijatý Medzinárodnou štandardizačnou Organizáciou (ISO) a verzia sa označuje ako C90. Najnovšia revízia štandardu je označovaná ako C18 a je z roku 2018 (6).

Programovací jazyk C je veľmi rozšírený a aktívne používaný v každej oblasti. Dokonca sa stal aj základom pre mnohé vyššie programovacie jazyky. Je možné v ňom tvoriť riešenia šité priamo na mieru.

S multiplatformovosťou nového riešenia by sme pri výbere programovacieho jazyka C nemali mať žiadne problémy, pretože jazyk má veľmi nízku úroveň platformovej závislosti, čo pre nás predstavuje veľkú výhodu. Jazyk C má stále aktívnu komunitu vývojárov. Na mnohých vysokých školách s informatickým zameraním sa s ním študenti stretnú pri tvorbe rôznych algoritmov. Jazyk ponúka veľké množstvo knižníc, ktoré by sme pri vývoji sondy mohli využiť, čím by sme si vývoj sondy značne uľahčili.

Veľkú prekážku pri výbere tohto jazyka pre nás predstavuje nedostatok vlastných skúseností. V jazyku sme riešili len malé a jednoduché úlohy. Preto sa pre nás z pohľadu efektívnosti vývoja jazyk C nejaví ako správna voľba. Pri vývoji sondy by mohli z dôvodu nedostatku vlastných skúseností nastať problémy, ktoré by mohli značne spomaliť tempo

práce. Navyše, programovací jazyk C je procedurálny a vývoj sondy by sa tak v pokročilých štádiách projektu mohol stať veľmi náročný.

C++

Počiatky tohto programovacieho jazyka siahajú až do roku 1979, kedy Bjarne Stroustrup pracoval na svojej dizertačnej práci. Pri práci používal programovací jazyk Simula, ktorý je považovaný za prvý objektovo orientovaný programovací jazyk. Bjarne Stroustrup bol objektovo orientovaným prístupom fascinovaný a chcel túto paradigmu použiť pri vývoji softvéru. Jazyk Simula bo však veľmi pomalý a preto siahol po jazyku C, ktorý sa stal základom pre vznik nového jazyka C with Classes. Tento bol neskôr premenovaný na C++. Jazyk bol prvý krát komerčne vydaný v októbri 1985 (7).

Jazyk C++ prebral syntax a mnoho vlastností od jazyka C. Vo väčšine prípadov zdrojový kód napísaný v jazyku C dokáže skompilovať C++ kompilátor. Vieme v ňom rovnako, ako aj v jazyku C, vyvíjať optimálne multiplatformové riešenia. Objektovo orientovaný prístup pri riešení problémov, ktorý nám jazyk C++ ponúka, je pre nás omnoho lákavejší, ako procedurálny prístup v prípade jazyka C. Komunity jazykov C++ a C sú navzájom úzko prepojené. Podobne, ako v prípade jazyka C, aj C++ nám umožňuje využiť mnoho pomocných knižníc. Rovnako však, ako v prípade jazyka C, vývoj sondy v C++ by sa mohol skomplikovať, pretože s jazykom nemáme dostatočné skúsenosti. Z toho dôvodu sa nám tento jazyk nejaví ako správna voľba.

Python

Programovací jazyk Python sa radí medzi vyššie objektovo orientované, interpretované, programovacie jazyky. Je omnoho mladší ako jazyk C. Vytvoril ho Guido van Rossum a jeho prvé vydanie sa datuje na rok 1991. Dnes sa považuje za jeden z najprogresívnejších jazykov vôbec. Jeho vývoj zastrešuje organizácia Python Software Foundation a jeho interpreter je uvoľnený ako open-source projekt. Najčastejšie sa dnes používa pri tvorbe webových a serverových aplikácií, umelej inteligencie a pre vedecké výpočty (8).

Python interpreter existuje aj pre Windows platformu, ale jeho správanie na tejto platforme sa nedá považovať za úplne natívne a efektívne. Inštalácia Python projektu na čistý Windows systém môže byť komplikovaná a spojená s rôznymi problémami, obzvlášť ak sa jedná o aplikácie, ktoré majú úzko spolupracovať s operačným systémom.

Nakoľko sa jedná o interpretovaný jazyk, mohli by sme naraziť aj na problémy s výkonom, čo by v našom prípade bolo veľmi nežiadúce. V kombinácii s problematickou

inštaláciou a kompatibilitou s Windows platformou a faktom, že s daným jazykom nemáme skúsenosti, nejaví sa ako správna voľba.

Java

Vývoj tohto programovacieho jazyka začal v roku 1990, v spoločnosti Sun Microsystems. Jeden z inžinierov pôsobiacich v tejto organizácii, Patrick Naughton, bol veľmi nespokojný so stavom aplikačných rozhraní spoločnosti. Jazyky C a C++, ktoré spoločnosť používala, zaberali príliš veľa pamäte. Taktiež absencia garbage collection mechanizmu nútila programátorov k manuálnemu spravovaniu pamäte, čo viedlo k častým chybám. Keď od spoločnosti dostal možnosť vytvoriť nový programovací jazyk, ktorý by nahradil C a C++, rozhodol sa venovať tomuto projektu. Neskôr sa k nemu pridali ďalší vývojári a vzniknutý tím programátorov v Sun Microsystems vytvoril nový objektovo orientovaný jazyk Java. Jazyk bol prvý krát vydaný v roku 1995 (9). V súčasnosti patrí Java medzi najrozšírenejšie najpoužívanejšie programovacie jazyky na svete. Jej uplatnenie môžeme nájsť takmer v každom odvetví.

Veľkú výhodu pre nás predstavuje prostredie Java virtual machine, v ktorom sa spúšťajú aplikácie napísane v jazyku Java. V tomto prostredí vieme rovnaký zdrojový kód spustiť nielen na našich cieľových platformách, ale aj na mnohých iných. Java je aktívny jazyk, ktorý je stále dopĺňaný o nové funkcie a vylepšenia. Ponúka nám na výber mnoho voľne dostupných knižníc, ktoré by nám vedeli uľahčiť prácu, rovnako ako v prípade predchádzajúcich jazykov. S jazykom sme sa už stretli v minulosti pri tvorbe Android aplikácií, ale s multiplatformovým vývojom v jazyku nemáme žiadne skúsenosti.

.NET

.NET je rozsiahla developerská platforma, vyvíjaná spoločnosťou Microsoft, ktorá nám umožňuje vytvárať rôzne typy aplikácií. Framework nám ponúka vyber z niekoľko programovacích jazykov. Najrozšírenejšie sú C#, F# a Visual Basic (10). Prvá verzia frameworku bola vydaná v roku 2002 a framework je pravidelne aktualizovaný.

Pri práci s .NET máme k dispozícii rôzne typy projektov, pričom každý z nich má iný účel. Ich správnou kombináciou je možné sa dopracovať k natívnemu riešeniu pre obe naše cieľové platformy. Podpora a komunita jazyka je v dnešnej dobe stále aktívna. Microsoft čast' svojho frameworku sprístupnil ako open-source projekt, vďaka čomu sa spoločnosti podarilo urýchliť vývoj platformy. Vo frameworku sme sa už v minulosti stretli s jazykom C#, s ktorým máme najviac skúsenosti z doposiaľ všetkých spomenutých technológií. Táto technológia,

rovnako ako predošlé spomenuté, nám umožňuje výber z veľkého množstva voľne dostupných knižníc. Voľba .NET frameworku pre nás predstavuje najefektívnejšiu možnosť z pohľadu vývoja.

Výsledná technológia pre sondu

Spomedzi všetkých spomenutých technológií sa ako najvhodnejšie a najpoužívateľnejšie javia .NET a Java. Sú použiteľné pre vývoj na oboch cieľových platformách. Obe technológie sú v súčasnosti vo veľkej miere používané, majú aktívnu podporu a komunitu vývojárov. Pre oboch kandidátov sa nám tiež podarilo nájsť knižnice, ktoré by nám uľahčili vývoj sondy. Rozhodli sme sa na tvorbu sondy využiť technológiu .NET, najmä vzhľadom na rozsah skúseností s porovnávanými možnosťami a to aj po zvážení malých rozdielov v systémových nárokoch na fungovanie sondy.

2.3.2 .NET Knižnice na zachytávanie rámcov

Aby sme nemuseli programovať zachytávanie sieťovej prevádzky, vyhľadali sme knižnicu ktorá nám túto funkcionality poskytne v NuGet galérii. NuGet je nástroj na správu knižníc pre .NET, zabudovaný v IDE Visual Studio. Podarilo sa nám nájsť hneď dve knižnice, Pcap.Net a SharpPcap, avšak len SharpPcap je použiteľný na oboch cieľových platformách, a preto sa ďalej budeme venovať len tejto knižnici.

SharpPcap

Je to knižnica, ktorá obaľuje volania knižníc libpcap, Npcap a AirPcap. Tieto slúžia na zachytávanie sieťovej komunikácie. Knižnica pôvodne obsahovala aj modul, ktorý zachytenú sieťovú prevádzku interpretoval ako objekty príslušného protokolu. Táto funkcionality bola neskôr extrahovaná do ďalšej knižnice s názvom PacketDotNet. Autorom oboch knižníc je Chris Morgan, knižnice sú vyvíjane ako open-source projekty a ich zdrojový kód sa da pozrieť na GitHub-e. Autor je stále aktívny čo je veľkou výhodou v prípade, že by bolo potrebné rozšíriť funkcionality týchto knižníc.

2.3.3 Perzistencia a vizualizácia sieťovej prevádzky

Správny výber technológie pre kombináciu perzistencie a vizualizácie zozbieraných štatistík je pre nás veľmi dôležitý, pretože môže do veľkej miery ovplyvniť rýchlosť vizualizácie dát, ktoré sonda nazbiera.

Podľa existujúcej verzie projektu KaTaLyzer vieme, aké informácie o sieťovej prevádzke budeme ukladať. Budú to:

- Čas zachytenia
- Zdrojová a cieľová IP
- Zdrojová a cieľová MAC
- Protokoly linkovej, sieťovej, transportnej a aplikačnej vrstvy TCP/IP modelu

Najdôležitejším faktorom pri výbere databázového systému pre nás je rýchlosť agregácie veľkého objemu dát a to aj za cenu vyšších nárokov na diskový priestor. Za účelom výberu vhodnej technológie z dostupných alternatív sme analyzovali možnosť využitia relačných databáz, nerelačných databáz a kombinácie produktov od spoločnosti Elastic .

Relačne databázy

Sú to databázy, v ktorých sa údaje ukladajú v tabuľkách, ktoré majú pevne zadefinovanú štruktúru. Pri správnom rozdelení informácií do viacerých tabuliek by sme pomocou relačných databáz vedeli ušetriť miesto na disku. Takýto návrh by však bol zbytočne komplikovaný a pri väčšom objeme dát by mohol spôsobovať značné spomalenie vizualizácií. Komplikovaný návrh by nám mohol spôsobiť taktiež problémy pri neskorších zmenách v štruktúre ukladaných údajov. Relačné databázy sa preto nejavia byť správnym kandidátom pre naše riešenie.

Nerelačne databázy

Sú to databázy, ktoré nemajú preddefinovanú schému, do ktorej sa ukladajú dáta. Ich schéma je flexibilná a pri pridaní nového stĺpca netreba robiť žiadne migrácie. Nerelačné databázy sú charakteristické tým, že sa dajú horizontálne škálovať a ich rýchlosť agregácie je vyššia, ako v prípade relačných databáz. Nerelačné databázy majú väčšinou jednoduchšiu schému, bez rôznych relácií, čo má za následok väčšie nároky na diskový priestor (11) (12) (13) (14).

ELK Stack

ELK Stack je zoskupením produktov Elasticsearch, Logstash a Kibana. Všetky tri projekty vyvíja spoločnosť Elastic a sú dostupné úplne zadarmo. Logstash predstavuje vstupné rozhranie pre Elasticsearch. Dáta, ktoré prijme, upraví podľa konfigurácie a zapíše do Elasticsearch. Elasticsearch je fulltextový vyhľadávací nástroj, v ktorom sú uložené dáta z Logstash. Kibana je webové rozhranie na vizualizáciu dát z Elasticsearch. Používateľ v ňom môže vytvárať rôzne vizualizácie zo zozbieraných a vyfiltrovaných dát, ktoré sa vykresľujú v reálnom čase. Umožňuje nám aj odosielanie varovných správ pri odhalení výkyvu v nameraných hodnotách na základe definície.

Elasticsearch je technológia, ktorá sa používa na enterprise úrovni, kde vďaka svojej robustnosti ponúka riešenia na problémy spojené s veľkým objemom dát. Technológia je v porovnaní s relačnými a nerelačnými databázami náročnejšia na systémové požiadavky, ale prekonáva ich v rýchlosti, ktorou dokáže spracovávať veľké množstvo dát.

Výsledná technológia pre perzistenciu

Od použitia relačnej alebo nerelačnej databázy nás odradila ich rýchlosť a skutočnosť, že by sme museli pre takúto databázu vytvoriť rozhranie, ktoré by vytváralo vizualizácie zozbieraných informácií. Tento postup by bol mimoriadne prácny, avšak jeho výsledok by pravdepodobne nebol porovnateľný s možnosťami, ktoré nám ponúka využitie ELK Stack. Z tohto dôvodu pre účel perzistencie a vizualizácie použijeme trojicu Elasticsearch, Logstash a Kibana.

2.3.4 Spôsoby inštalácie ELK Stack

Analyzovali sme tri spôsoby, ktorými si používateľ môže nainštalovať ELK Stack. Najzákladnejší spôsob je natívna inštalácia všetkých troch komponentov, ale existuje aj možnosť ponúkať používateľovi komponenty predinštalované a predkonfigurované pomocou virtuálneho stroja alebo pomocou technológie Docker.

Z týchto troch spôsobov inštalácie vyberieme jeden, s ktorým budeme KaTaLyzér ponúkať a ten potom popíšeme v používateľskej príručke. Pri výbere budeme dbať na zložitosť inštalácie, konfigurácie a efektívnosť využitia systémových prostriedkov.

Natívna inštalácia komponentov

Pri natívnej inštalácii je potrebné, aby si používateľ manuálne stiahol, nainštaloval a nakonfiguroval všetky tri komponenty ELK Stack. Preto voľba takéhoto spôsobu inštalácie môže byť pre menej skúseneho používateľa príliš zložitá. Kroky inštalácie by mali istú mieru platformovej závislosti, čo predstavuje riziko.

Obraz virtuálneho stroja

Alternatíva ponúknuť ELK Stack ako predkonfigurovaný obraz virtuálneho stroja sa javí ako vhodnejšie riešenie, než nechať celý proces inštalácie a konfigurácie na používateľovi. Od takéhoto prístupu nás však odrádza réžia spojená s prevádzkou virtuálneho stroja a horšia efektívnosť pri využívaní systémových prostriedkov. Prekážku pre nás predstavuje aj veľkosť virtuálneho stroja a povinnosti spojené s jeho údržbou.

Docker

Docker je technológia veľmi podobná virtualizácii, avšak namiesto virtuálnych strojov používa kontajnery, v ktorých sú spustené jednotlivé služby. Kontajner namiesto vytvorenia nového operačného systému, ako v prípade virtuálneho stroja, využíva možnosti systému v ktorom je spustený. Vďaka takémuto prístupu technológia dosahuje omnoho väčšiu efektívnosť pri využívaní systémových prostriedkov (15).

Výhodou je, že obraz kontajnera pre ELK Stack je voľne dostupný. Vieme ho spustiť pomocou jedného príkazu. Technológia nám umožňuje zadať kroky, ktoré sa majú vykonať po vytvorení kontajnera, napríklad pre účel dodatočnej konfigurácie. Týmto spôsobom môžeme používateľa oslobodiť od komplikovanej konfigurácie, čím pre neho výrazne uľahčíme proces inštalácie.

Nevýhodou je to, že technológiu je potrebné nainštalovať a dodatočná konfigurácia môže byť naračnejšia pre menej skúsených používateľov.

Zvolený spôsob inštalácie ELK Stack

Technológia Docker nám ponúka výhody, ktoré sú pre nás pri výbere spôsobu inštalácie kľúčové. Vieme pomocou nej používateľovi značne uľahčiť inštaláciu a konfiguráciu komponentov pri vysokej efektívnosti využitia systémových prostriedkov. Netreba však zabudnúť na skutočnosť, že používateľ má stále možnosť zvoliť si aj iný spôsob inštalácie, než ten, ktorý je ponúkaný.

2.3.5 Formát komunikácie medzi sondou a Logstash

Z dokumentácie sme zistili že Elasticsearch jednotlivé dáta ukladá ako JSON objekty (16) (17). Takto upravený vstup môžeme priamo posielať pomocou HTTP protokolu rozhraniu Logstash (18).

JSON (JavaScript Object Notation) je formát určený na prenášanie a ukladanie dát (19). Svoju popularitu získal vďaka svojej jednoduchosti. Obsah JSON objektu môže pripomínať slovník, pretože dáta sú v ňom uložené ako zoznam tvorený kľúčom a hodnotou. Pre platformu .NET máme k dispozícii knižnicu Newtonsoft.Json, ktorá nám poskytuje konverziu triedy na JSON objekt a naopak.

3 Špecifikácia

Na základe zistení o existujúcej verzii projektu KaTaLyzer získaných počas analýzy a po dohode s vedúcim práce sme vytvorili funkcionálne a nefunkcionálne požiadavky, ktoré musí nové riešenie spĺňať.

3.1 Funkcionálne požiadavky

3.1.1 Sonda

- Zachytávanie sieťovej prevádzky na vybraných sieťových rozhraniach zariadenia
- Analýza a štatistické spracovanie informácií o sieťovej prevádzke
 - o Názov monitorovaného zariadenia
 - o Sieťová karta zariadenia
 - o Veľkosť zachyteného rámca v bajtoch
 - o Cieľová IP/MAC adresa rámca
 - o Zdrojová IP/MAC adresa rámca
 - o Čas zachytenia rámca
- Sumarizácia informácií o sieťovej prevádzke pre zníženie nákladov na pamäťový priestor
- Perzistencia informácií, ktoré sonda zachytila počas zadefinovaného časového intervalu, na zadefinovaný databázový server
- Možnosť zmeny nastavení pomocou konfiguračného súboru
 - o Interval perzistencie štatistík
 - o Názov zariadenia
 - o Adresa servera
 - o Monitorované sieťové rozhrania
 - o Monitorované protokoly
- Pomocný sprievodca pri vytváraní konfigurácie
- Riešenie musí byť spustiteľné na operačnom systéme Windows a Linux
- Ukladanie informácií o stave sondy počas fungovania do súboru a na server (Logovanie)

3.1.2 Grafické rozhranie

- Vizualizovanie štatistických informácií o sieťovej prevádzke
 - o Veľkosť sieťovej prevádzky v čase meraná v bajtoch
 - o Rozdelenie sieťovej prevádzky podľa TCP/IP vrstiev
 - o Rozdelenie sieťovej prevádzky podľa zariadení a rozhraní
- Filtrovanie na základe parametrov štatistického záznamu
 - o Zobrazenie štatistických záznamov podľa zvoleného časového okna
 - o Zobrazenie štatistických záznamov podľa zvoleného zariadenia alebo sieťovej karty
 - o Zobrazenie štatistických záznamov podľa zvoleného protokolu
 - o Zobrazenie štatistických záznamov podľa zvolených adries IP alebo MAC

3.2 Nefunkcionálne požiadavky

- Aplikácia sonda má efektívne pracovať so systémovými prostriedkami
- Inštalácia riešenia musí byť jednoduchá
- Dlhodobé fungovanie sondy, bez potreby interakcie používateľa
- Stabilita - aplikácia sa musí vedieť vysporiadať s chybami, ktoré môžu pri prevádzke nastať (strata spojenia, odstránený sieťový adaptér)

4 Návrh riešenia

Po zvolení technológií pre jednotlivé komponenty systému, nasleduje návrh riešenia, ktorý bližšie špecifikuje, z akých .NET projektov bude sonda pozostávať. Popíšeme jej fungovanie a spôsob, akým bude ukladať zozbierané štatistiky. Táto kapitola sa bude zameriavať prevažne na sondu, pretože ELK Stack je hotový produkt, ktorý si nevyžaduje implementáciu.

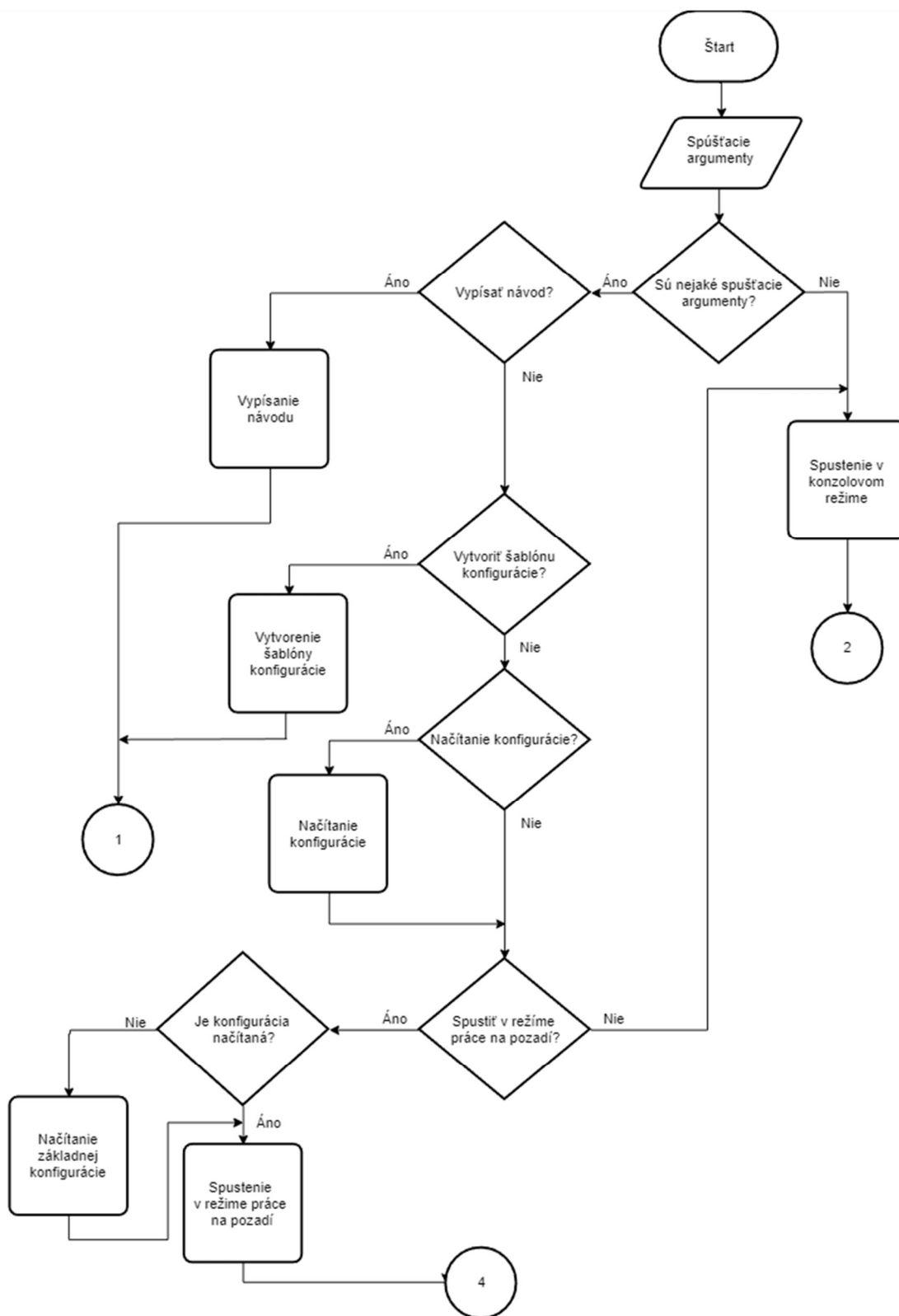
4.1 Sonda

Sonda bude pozostávať z troch projektov. Je to z toho dôvodu, že budeme potrebovať oddeliť platformovo závislý zdrojový kód, ktorý je určený výhradne pre Linux, od zdrojového kódu určeného výhradne pre Windows. Tretí projekt bude obsahovať zdrojový kód, ktorý je spustiteľný pre obe platformy. Projekty budú mať názvy podľa cieľovej platformy. V prípade zdieľaného projektu to bude Shared. Cieľový framework sme pre každý projekt vybrali tak, aby sme sa priblížili k čo najviac natívnemu riešeniu pre danú platformu. Projekt so zdieľaným kódom má cieľový framework kompatibilný s oboma platformovo závislými projektami. Výsledné názvy projektov a ich cieľový framework sú zhrnuté v nasledujúcej tabuľke.

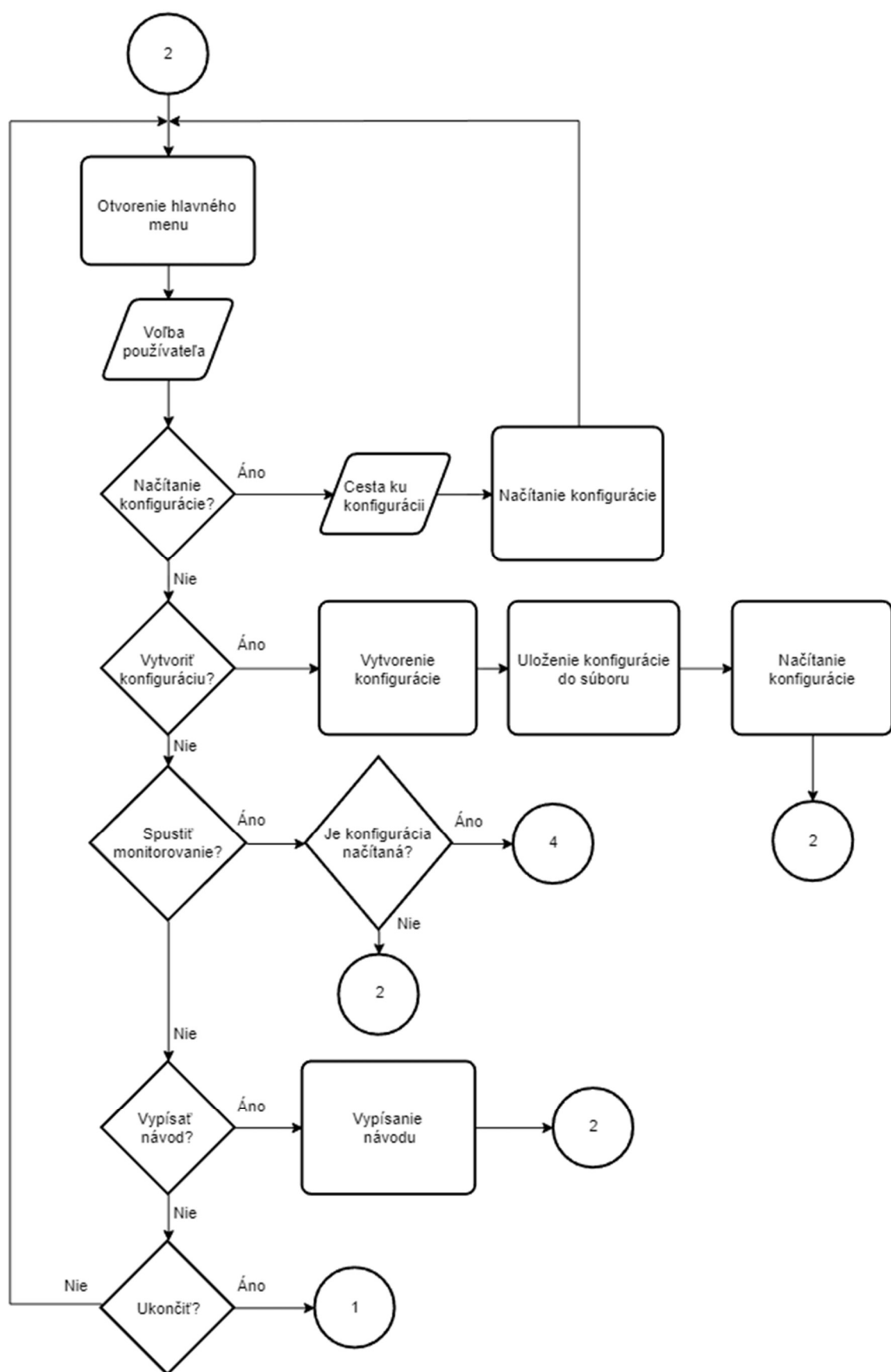
Názov projektu	Cieľový framework
Linux	.NET Core 2.1
Shared	.NET Standard 2.0
Windows	.NET Framework 4.7.2

Tabuľka 1 Rozdelenie sondy na projekty podľa .NET framework

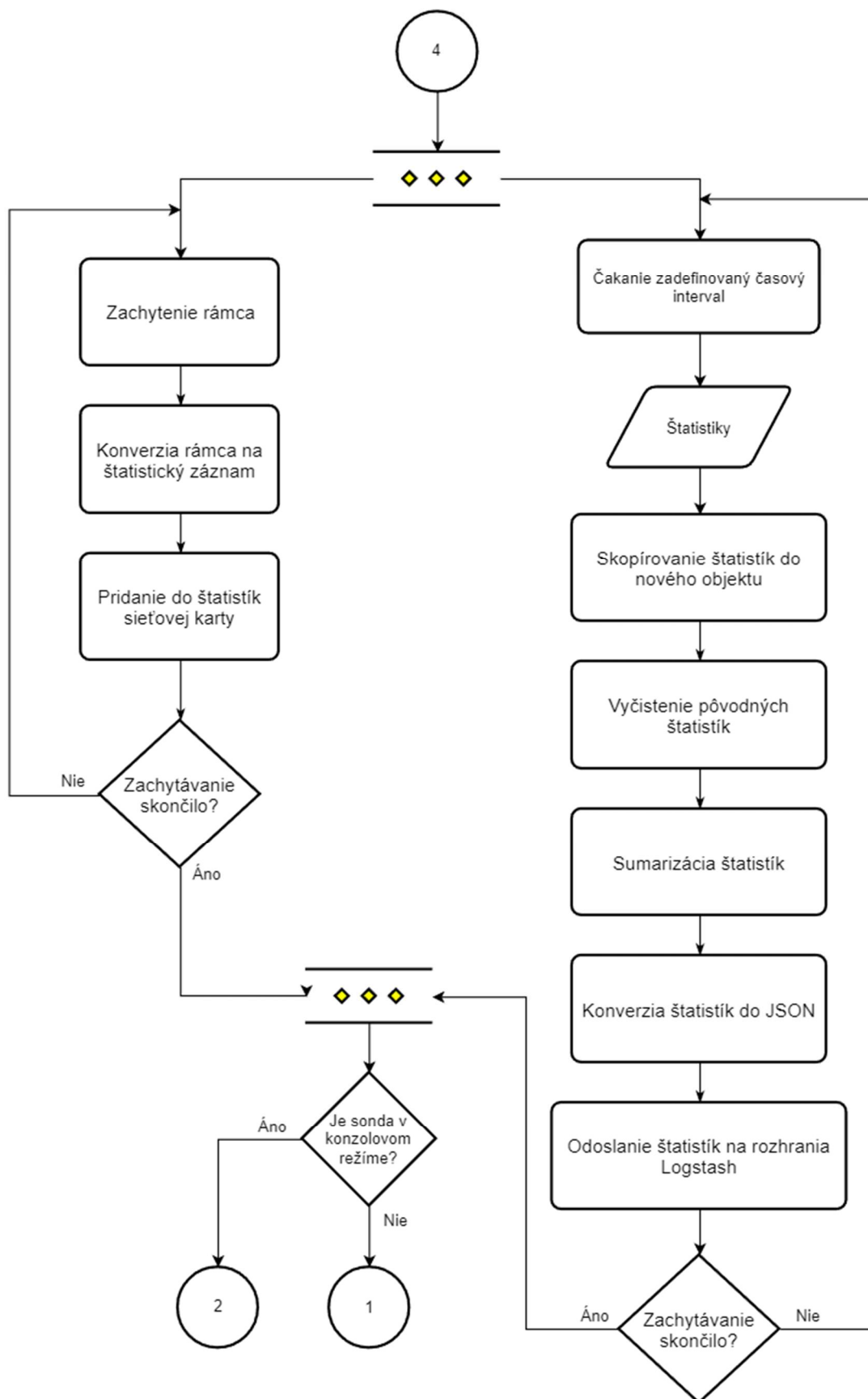
Pre sondu sme vytvorili jeden diagram popisujúci jej správanie. Diagram je rozdelený na štyri časti. Prvá časť je zameraná na spustenie sondy. Je v nej popísané, aké činnosti vykonáva na základe jednotlivých spúšťacích argumentov. Druhá časť popisuje hlavné menu, ktoré sa spúšťa v konzolovom režime. Tretia časť popisuje proces zachytávania a odosielania informácií o sieťovej prevádzke na zariadení. Posledná, štvrtá časť diagramu predstavuje len ukončenie sondy. Funkcionalita sondy je popísaná aj slovne v ďalších častiach kapitoly.



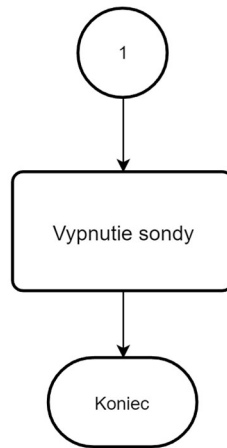
Obrázok 2 Diagram popisujúci spustenie sondy



Obrázok 3 Diagram popisujúci hlavné menu



Obrázok 4 Diagram popisujúci monitorovanie sieťovej prevádzky



Obrázok 5 Koniec diagramu

4.1.1 Konfigurácia sondy

Je potrebné dať používateľovi možnosť si sondu nakonfigurovať podľa svojich potrieb. A to do najväčšie možnej miery. Pre tento dôvod bude naša sonda používať konfiguračný súbor, pomocou ktorého bude používateľ môcť zadefinovať sonde nasledujúce parametre:

1. Názov zariadenia, ktorý bude slúžiť na identifikáciu pôvodu dát
2. Interval, v ktorom sa budú logy ukladať a odosielat' na server
3. Sieťové karty, ktoré budeme monitorovať
4. Protokoly, ktoré budeme monitorovať
5. Adresa servera, na ktorý budeme posielat' štatistiky

Ako formát konfiguračného súboru sme vybrali JSON reťazec. JSON formát sa jednoducho číta a je veľmi rozšírený. Navyše knižnica `Newtonsoft.Json` za nás vyrieši načítanie súboru do programového objektu konfigurácie alebo uloženie objektu konfigurácie do súboru. To nám značne uľahčí manipuláciu s konfiguráciou.

4.1.2 Spustenie sondy

Pri spustení sondy kontrolujeme všetky spúšťacie argumenty, s ktorými bola sonda spustená. Rozlišujeme tieto štyri spúšťacie argumenty:

1. Vypísanie jednoduchého návodu na obsluhu sondy
2. Vytvorenie šablóny pre konfiguračný súbor
3. Cesta ku konfiguračnému súboru
4. Režim práce na pozadí

Používateľovi ponúkame na výber z dvoch režimov, v ktorých môže sonda vykonávať svoju funkcionálnosť.

Základný režim bude predstavovať spustenie sondy ako konzolovej aplikácie, ktorá reaguje na vstup od používateľa. Používateľovi sa po spustení otvorí základné menu, kde bude mať na výber vytvorenie alebo načítanie konfigurácie, spustenie zachytávania sieťovej prevádzky a zobrazenie pomoci pre používateľa.

Druhý režim sondy je určený pre prácu na pozadí. Používateľ ho môže aktivovať pomocou spúšťacích parametrov. Sonda potom nebude čakať na vstup od používateľa a na základe konfigurácie v spúšťacích parametroch, sa okamžite spustí zachytávanie sieťovej prevádzky. V prípade, že sa konfigurácia nenachádzala v spúšťacích argumentoch, načíta sa konfigurácia uložená pri spúšťacom súbore s názvom config.json.

4.1.3 Reprezentácia štatistík



Obrázok 6 Návrh triedy pre štatistický záznam

Jeden štatistický záznam bude obsahovať všetky informácie, ktoré sme získali z rámca. V projekte bude záznam reprezentovaný pomocou triedy, ktorá bude uložená v zdieľanom projekte. Jeho kód bude použiteľný pre obe platformy.

4.1.4 Zachytávanie rámcov a vytváranie štatistík

Na zachytávanie rámcov použijeme knižnicu SharpPcap. Zachytávanie bude prebiehať na všetkých sieťových adaptéroch, ktoré sú nastavené v konfigurácii. Sieťový adaptér, po zachytení rámca, odošle rámec na spracovanie. Vytvoríme nový objekt štatistického záznamu na základe informácií, ktoré sme o rámci získali. Tento nový objekt si uložíme do množiny štatistík pre príslušný adaptér. Celý tento proces bude prebiehať v samostatnom vlákne, cyklicky, kým zachytávanie nebude ukončené.

4.1.5 Sumarizácia a perzistencia štatistík

Sonda po uplynutí stanoveného intervalu, ktorý je zadaný v konfiguračnom súbore, vytvorí kópiu štatistických záznamov. Pôvodné štatistické záznamy vyčistí a nad kópiou vykoná sumarizáciu štatistických záznamov pre každý adaptér. Sumarizovane štatistiky potom pošle v HTTP-POST metóde na adresu Logstash servera, ktorú získa z konfiguračného súboru. Obsah HTTP správy budú zozbierané štatistiky skonvertované do JSON reťazca. Výsledné štatistiky bude používateľ môcť prehliadať cez webový nástroj Kibana.

4.1.6 Logovanie

Logovanie slúži predovšetkým vývojárom. V prípade výskytu chyby je vývojár schopný z logov vyčítať, kde sa v programe stala chyba a čo jej predchádzalo. Preto aj riešenie, ktoré navrhujeme, bude obsahovať logovanie.

Logy, ktoré bude naše riešenie ukladať, budú poskytovať pre vývojárov nasledujúce informácie:

- Závažnosť udalosti
- Telo logu – výnimka alebo informácia o stave aplikácie
- Dodatočné informácie – parametre metódy
- Kedy bol log vytvorený
- Verzia aplikácie

Logy budeme ukladať na dve miesta súčasne. Primárne ich budeme zapisovať do súboru logs.txt k sonde. V prípade, že je načítaná konfigurácia a server je dostupný, logy zapíšeme aj do Elasticsearch pomocou Logstash, podobne ako štatistiky, s tým rozdielom, že logy budú v odlišnom indexe. Kontrola logov bude možná cez nástroj Kibana.

4.2 ELK Stack

Tri projekty (Elasticsearch, Logstash, Kibana), ktoré názov ELK Stack zoskupuje je možné nainštalovať samostatne. Aby sme vytvorili čo možno najjednoduchšiu inštaláciu pre používateľa, použijeme technológiu Docker a existujúci obraz, ktorý obsahuje trojicu projektov.

4.2.1 Dátový model pre štatistické záznamy



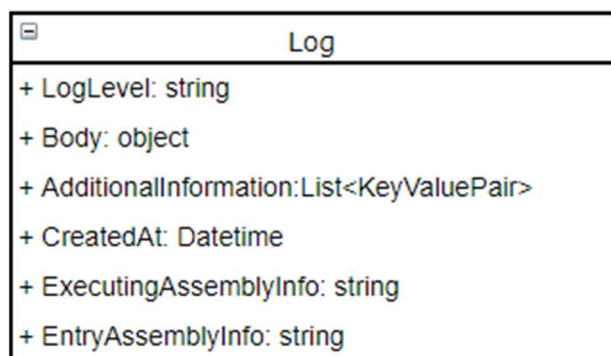
Obrázok 7 Dátový model pre štatistický záznam

Naším cieľom je vyhnúť sa zložitému dátovému modelu, ktorý by sťažoval prácu menej skúseným používateľom. Prípadné rozšírenia takéhoto zložitého dátového modelu by mohli byť omnoho náročnejšie na implementáciu.

Preto pre štatistické záznamy volíme jednoduchý dátový model, kde do jedného dokumentu uložíme všetky informácie. Vytváranie vizualizácií a filtrovanie bude potom jednoduchšie a bez nutnosti použitia komplikovaných dotazov.

4.2.2 Dátový model pre logovacie záznamy

Rovnako ako v prípade štatistík, aj logovacie záznamy budú mať jednoduchý dátový model a všetky potrebné informácie o logoch budeme uchovávať v jednom dokumente.



Obrázok 8 Dátový model pre logovací záznam

4.2.3 Vizualizácia zozbieraných štatistík

Pri vizualizáciách v nástroji Kibana budeme používať všetky parametre jedného štatistického záznamu. V nástroji preto vytvoríme deväť základných náhľadov pre používateľa, ktoré si bude môcť importovať. Tieto náhľady budú vyobrazovať veľkosť prevádzky podľa:

- Veľkosti premávky v čase
- Zariadenia a sieťových kariet
- Štyroch vrstiev TCP/IP modelu
- Zdrojovej a cieľovej MAC adresy
- Zdrojovej a cieľovej IP adresy
- Zdrojového a cieľového portu

4.2.4 Vizualizácia zozbieraných logov

Logovacie záznamy budú obsahovať tri náhľady, v ktorých budeme logy vyobrazovať podľa počtu výskytov v čase, podľa úrovne závažnosti a jeden náhľad bude vytvorený pre zobrazenie podrobností logovacích záznamov.

4.3 Porovnanie s pôvodným riešením

V podkapitole sa budeme zaoberať rozdielmi vo funkcionalite navrhnutého a pôvodného riešenia. Rozdiely zväčša predstavujú vylepšenia, ktoré budeme implementovať do sondy alebo nám ich funkcionalitu poskytujú zvolené technológie. Vyskytli sa však aj opačné prípady, kedy sme funkcionalitu pôvodného riešenia nezapracovali do návrhu nového riešenia.

4.3.1 Podporované protokoly

Knižnica, ktorú bude naše riešenie používať na vyhodnocovanie sieťovej prevádzky už obsahuje implementáciu pre väčšinu protokolov. Podporu pre protokoly, ktoré knižnica nepozná, budeme musieť implementovať v našom riešení. Vytvorili sme zoznam protokolov rozdelený podľa štyroch vrstiev TCP/IP modelu. Pri vrstvách uvádzame, ktoré informácie okrem názvu protokolu budeme získavať, ak to protokol zachyteného rámca umožní. Protokoly vo vrstvách delíme na také, ktoré sú implementované v knižnici a ktorých podporu budeme implementovať.

1. Vrstva sieťového rozhrania
 - a. Dodatočne informácie sú zdrojová a cieľová MAC adresa
 - b. Protokoly podporované knižnicou
 - i. ETHERNET, IEEE 802.1q, LLDP, SLL, WoL
 - c. Protokoly, ktorých podporu budeme implementovať
 - i. CDP, IEEE 802.3
2. Sieťová vrstva
 - a. Dodatočne informácie sú zdrojová a cieľová IP adresa
 - b. Protokoly podporované knižnicou
 - i. ICMPv4, ICMPv6, IGMPv2, IP, IPv6, ARP, RARP, OSPFv2
3. Transportná vrstva
 - a. Dodatočne informácie sú zdrojový a cieľový port
 - b. Protokoly podporované knižnicou
 - i. TCP, UDP
 - c. Protokoly, ktorých podporu budeme implementovať
 - i. gQUIC
4. Aplikačná vrstva
 - a. Protokoly ktorých podporu budeme implementovať
 - i. FTP, HTTP, HTTPS, SIP, SSH

Pre protokol IPX, ktorý bol podporovaný v existujúcom riešení, knižnica na vyhodnocovanie sieťovej prevádzky nemá implementáciu a my ho implementovať nebudeme, pretože samotný protokol IPX bol nahradený protokolom IP a v dnešnej dobe sa už nepoužíva.

Výsledok porovnania podpory protokolov navrhnutého a existujúceho riešenie je taký, že riešenie, ktoré sme navrhli podporuje osem nových protokolov a bola zrušená podpora pre jeden protokol.

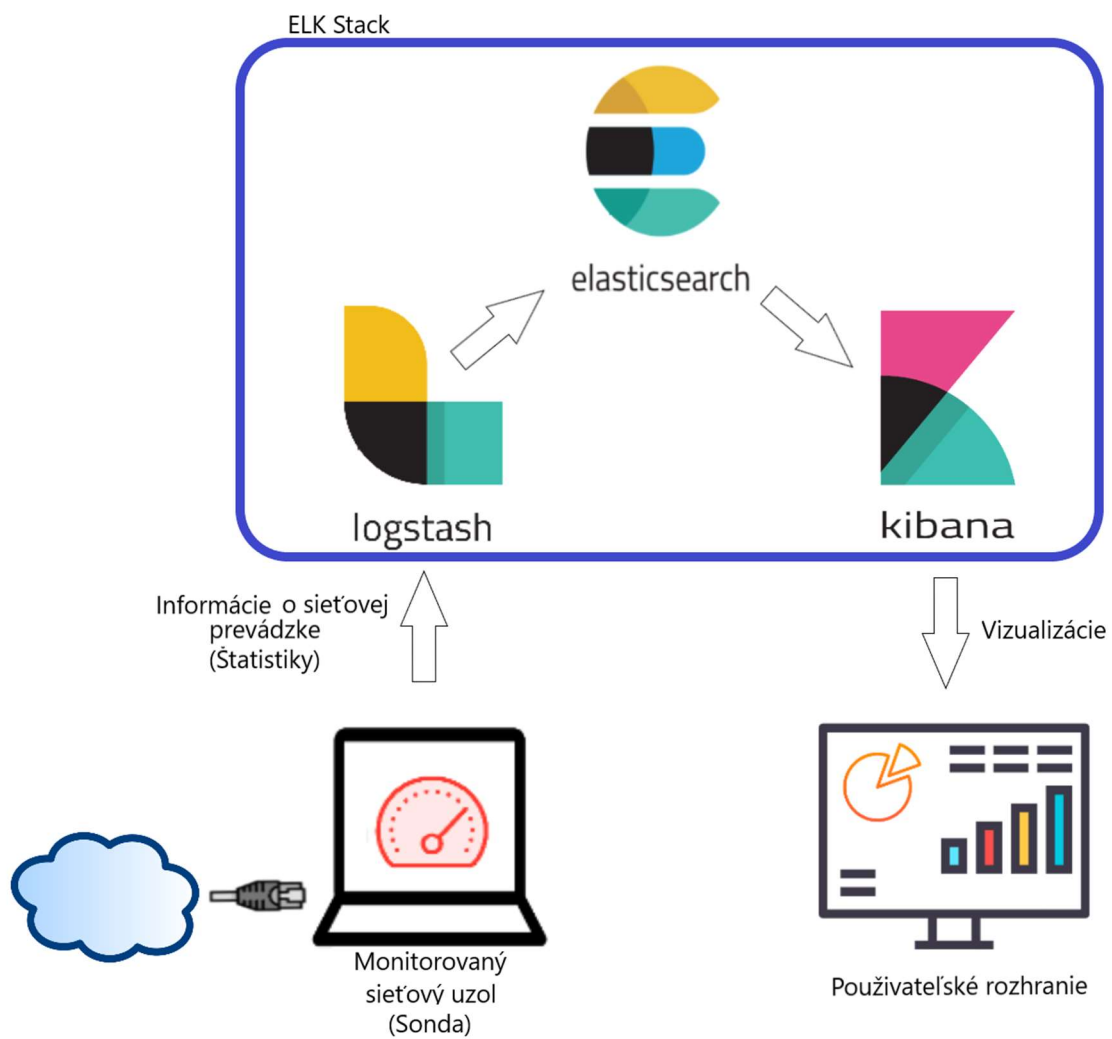
4.3.2 Webové rozhranie

Ak by v pôvodnom riešení dve sondy, nainštalované na dvoch rôznych zariadeniach, používali jedno a to isté webové rozhranie spolu s databázou, tak používateľ ktorý by informácie o sieťovej prevádzke prezeral by nevedel určiť zdroj dát. Dôvodom je chýbajúca informácia o zariadení, ktoré dáta poslalo. Štatistické záznamy, ktoré odosiela naša sonda túto chýbajúcu informáciu obsahujú a je teda možné rovnakú databázu a webové rozhranie použiť pre viacero zariadení, pričom používateľ bude mať stále prehľad o tom, ktoré zariadenie je zdrojom zozbieraných štatistík.

4.4 Výsledná architektúra

Architektúra nášho riešenia sa podobá na architektúru existujúceho riešenia. Rozdiely v riešeníach sú prevažne technologického charakteru. Komunikáciu medzi databázou (Elasticsearch) a sondou sme rozšírili o rozhranie (Logstash), ktoré náš vstup ukladá do databázy. Výsledná architektúra sa skladá zo štyroch častí:

1. Sonda, ktorá posiela údaje do Logstash
2. Logstash, rozhranie ktoré zapisuje údaje do Elasticsearch
3. Elasticsearch, úložisko dát
4. Kibana, webové rozhranie na vizualizáciu dát z Elasticsearch

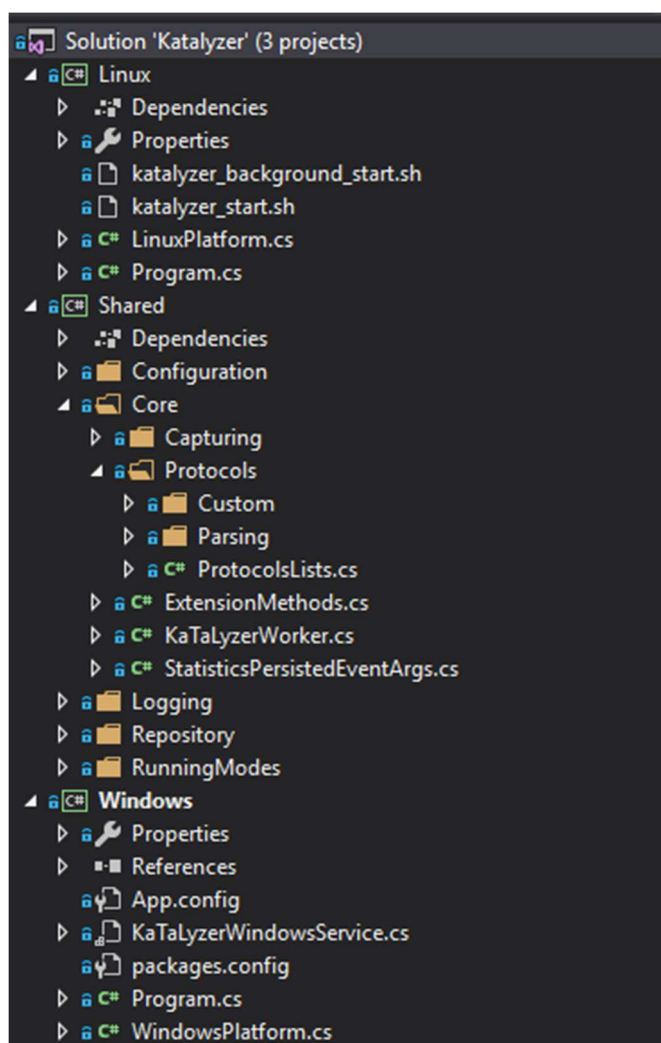


Obrázok 9 Architektúra systému

5 Implementácia

5.1 Sonda

Sondu sme implementovali v nástrojí Visual Studio 2017 Community Edition, v jazyku C#. Riešenie sme rozdelili do troch projektov. Projekt Linux a Windows sú spúšťacie pre cieľovú platformu. Projekt Shared obsahuje funkcionálnosť, ktorá nie je platformovo závislá.



Obrázok 10 Štruktúra riešenia sondy

5.1.1 Spracovanie argumentov pri spúšťaní sondy

Pri implementácii spracovania spúšťacích argumentov na Linux aj Windows verzii sondy sme použili externú knižnicu CommandLineParser. Pomocou anotácií, ktoré nám knižnica ponúka, sme vytvorili triedu CommandLineOptions. Táto trieda reprezentuje podporované argumenty sondy. Trieda ArgumentsResolver v metóde ResolveArguments vytvorí pomocou knižnice na spracovanie argumentov inštanciu triedy CommandLineOptions. V metóde ďalej

kontrolujeme jednotlivé atribúty objektu `CommandLineOptions` a na základe výsledkov vykonávame príslušné akcie. Konkrétne sú to vypísanie návodu na obsluhu, vytvorenie šablóny konfiguračného súboru, načítanie konfigurácie a spustenie v režime práce na pozadí.

5.1.2 Načítavanie sieťových rozhraní

Načítavanie sieťových rozhraní je platformovo závislá operácia. Je to z toho dôvodu že Windows a Linux nemajú rovnaký sled volaní, pomocou ktorých by sme získali zoznam sieťových rozhraní. Nerieši to za nás ani knižnica `SharpPcap`.

Pre cieľové platformy sme vytvorili abstraktnú triedu `PlatformDependencies`, ktorá obsahuje abstraktnú metódu `GetCurrentDevices`. Tá nám po zvolaní vráti `List<PcapDevice>`, čo je zoznam sieťových rozhraní. Od abstraktnej triedy dedia triedy `WindowsPlatform` a `LinuxPlatform`, v ktorých sa nachádza implementácia metódy na získanie zoznamu sieťových kariet zariadenia. Tá pozostáva z využitia správneho listu so zariadeniami, ktoré nám ponúka `SharpPcap`. V prípade platformy Linux to je list `LibPcapLiveDeviceList`. Pre Windows to je `WinPcapDeviceList`. Pri oboch platformách pracujeme len so zariadeniami, ktoré nemajú atribút `PcapDevice.Interface.FriendlyName` prázdny. Atribút predstavuje názov sieťového rozhrania. Pri filtrovaní sme použili LINQ výraz so syntaxou metódy.

V prípade, že bude sieťová karta počas monitorovania zakázaná a potom opäť povolená, je potrebné aktualizovať referenciu na túto sieťovú kartu. Trieda `NetworkAdapter` predstavuje rozhranie na monitorovanie sieťovej karty zariadenia. Keď sa počas zachytávania sieťovej prevádzky v metóde `StartCapturing` vyskytne chyba, pokúsime sa zistiť, či nemohla byť spôsobená práve nesprávnou referenciou na sieťovú kartu. Metóda `GetActualDevice` v abstraktnej triede `PlatformDependencies` vráti platnú referenciu na sieťovú kartu zariadenia a pokúsime sa pomocou nej zachytávanie obnoviť.

5.1.3 Konfigurácia sondy

Pre účel uchovania konfigurácie počas behu sondy sme vytvorili triedu `Settings`, ktorá reprezentuje nastavenia. `ConfigurationManager` je trieda, ktorá je určená na manipuláciu s konfiguráciou. Je implementovaná ako singleton, čo nám zaručí že nikdy nevznikne viac ako jedna inštancia tejto triedy a nedôjde tak k nekonzistencii nastavení v jednotlivých moduloch. Trieda na správu konfigurácie nám umožňuje načítať alebo uložiť konfiguráciu a pomocou nej pristupujeme k jednotlivým atribútom aktuálnej konfigurácie.

Metóda SaveConfiguration slúži na uloženie konfigurácie. Je volaná pri vytváraní šablóny pre konfiguráciu alebo pri vytváraní konfigurácie pomocou sprievodcu. Metóda LoadConfiguration slúži na načítanie konfigurácie zo súboru. Je volaná počas spracovania vstupných argumentov alebo z hlavného menu.

5.1.4 Implementácia objektu pre štatistiky

Pre štatistický záznam, do ktorého ukladáme informácie o jednom zachytenom rámci sme vytvorili triedu StatisticsRecord. Obsahom triedy sú jednotlivé atribúty jedného záznamu a dva konštruktory slúžiace na vytvorenie inštancie záznamu. Jeden z konštruktorov je bez parametrov a vytvorí prázdnu inštanciu záznamu. Druhý konštruktor slúži na skopírovanie existujúceho záznamu do novej inštancie objektu.

Pri jednotlivých atribútoch štatistického záznamu sme pomocou anotácií, ktoré nám ponúka knižnica Newtonsoft.Json a .NET, nastavili poradie atribútov pri konverzii do JSON objektu a inicializačnú hodnotu atribútov.

5.1.5 Zachytávanie rámcov a vytváranie štatistík

Rámce zachytávame v triede NetworkAdapter. Trieda obsahuje dve metódy, pomocou ktorých vieme zachytávanie zapnúť a vypnúť. Sieťovú kartu musíme pripraviť na zachytávanie zavolaním metódy PcapDevice.Open. Následne v cykle pomocou funkcie PcapDevice.GetNextPacket zachytávame prevádzku na sieťovej karte.

Zachytenú prevádzku, v podobe SharpPcap objektu typu RawCapture, odošleme na spracovanie pomocou vyvolania udalosti NetworkAdapter.PacketCaptured. V prípade chyby vyvoláme udalosť ErrorOccured. Na udalosti sieťovej karty registrujeme metódy ProcessPacket a AdapterErrorOccured triedy KaTaLyzerWorker. Metóda ProcessPacket spracuje rámec. Metóda AdapterErrorOccured vyvolá ďalšiu udalosť, ktorá v konzolovom režime spôsobí vypísanie chybového hlásenia. V režime práce na pozadí sa chyba len zaznamená na mieste výskytu.

V triede ExtensionMethods sme implementovali rozširujúce volanie ConvertToStatistics nad objektom RawCapture. Metóda slúži na vytvorenie štatistického záznamu zo zachyteného rámca. Toto rozšírenie voláme v metóde ProcessPacket v triede KaTaLyzerWorker. Metóda po konverzii zaradí štatistický záznam k príslušnej sieťovej karte, ktoré sú uložené v premennej _networkAdaptersStatistics

5.1.6 Sumarizácia štatistík

Pre zoznam štatistík sieťového adaptéra sme implementovali rozširujúcu sumarizačnú metódu. Pomocou LINQ výrazov, ktoré nám ponúka C#, vykonáme nad objektom typu `List<StatisticsRecord>` agregáciu `GroupBy` podľa parametrov jedného štatistického záznamu, okrem veľkosti. Takto nám vzniknú skupiny štatistických záznamov, kde sú všetky parametre okrem ich veľkosti rovnaké. Zvolíme si prvý element skupiny ako referenčný a vytvoríme z neho kópiu. Veľkosť novému štatistickému záznamu nastavíme ako sumu všetkých veľkostí v skupine pomocou LINQ funkcie `Sum`.

5.1.7 Perzistencia štatistík

Perzistenciu vykonáva časovač, ktorý sa nachádza v triede `KaTaLyzerWorker`. Po uplynutí nastaveného intervalu sa zozbierané štatistické záznamy všetkých sieťových kariet odložia. Vykonáme nad nimi sumarizáciu. Sumarizované štatistické záznamy potom pomocou metódy `PersistStatistics` v triede `StatisticsRepository` odošleme na HTTP rozhranie Logstash servera.

Štatistiky odosielame ako objekt triedy `StatisticsMessage`. Trieda obsahuje reťazec, ktorý určuje cieľový index pre uloženie dát do Elasticsearch, a tiež názov zariadenia. Jej obsahom sú aj samotné dáta. Štatistiky posielame ako `List<NetworkAdapterStatistics>`. Odosielané dáta obsahujú zoznam sieťových kariet, v ktorom jeden element reprezentuje sieťovú kartu a zoznam jej štatistík.

Štruktúra odosielaných dát bola implementovaná cez dva zoznamy, aby sme dosiahli čo možno najnižšiu veľkosť pri jej odosielaní. Štatistické záznamy sa do podoby dátového modelu dostanú až počas spracovania v Logstash.

5.1.8 Implementácia logovania

Implementovali sme triedu `LogMessage`, ktorej atribúty predstavujú jednotlivé informácie, ktoré zaznamenávame počas behu sondy. Jej inštanciu vytvára `KaTaLyzerLogger` v metóde `Log`. Po vytvorení inštancie logovacieho objektu sa v metóde `LogToServerAndFile` vykoná uloženie záznamu do súboru a na vzdialený server.

Zápis do súboru sme museli synchronizovať pre viacero vlákien, aby nedošlo k výnimke. Pre účel synchronizácie sme použili objekt `ReaderWriterLock`, ktorý nám .NET ponúka. Pri

zápise na server používame triedu LogsRepository, ktorá jednotlivé logy odosiela na HTTP vstup servera Logstash.

5.1.9 Implementácia režimu práce na pozadí

Pre obe platformy sme implementovali režim pre prácu na pozadí. Po jeho spustení používateľ nevie sondu ovládať. Používateľ vie sonde iba poskytnúť konfiguráciu, pomocou ktorej vykonáva monitorovanie na zariadení. Fungovanie režimu je rozdielne pre Windows a Linux. Abstraktná trieda PlatformDependencies obsahuje metódu RunInBackground, ktorá spustí sondu na príslušnej platforme v režime práce na pozadí. Implementácia metódy RunInBackground sa nachádza v triedach LinuxPlatform a WindowsPlatform.

Windows

Pri platforme Windows je sonda v režime práce na pozadí spúšťaná operačným systémom ako služba. V projekte Windows sme vytvorili triedu KaTaLyzerService, ktorá predstavuje službu. V nej spustíme zachytávanie sieťovej prevádzky v samostatnom vlákne, pomocou triedy BackgroundJob, ktorá na rozdiel od triedy ConsoleUi, nepracuje s konzolovým vstupom a výstupom. Sondu je možné v tomto režime spustiť iba ako službu. Po pridaní argumentu `,-b` pri štandardnom spustení sonda nebude fungovať. Postup, akým sa služba vytvára, je popísaný v používateľskej príručke.

Linux

Pri implementácii režimu práce na pozadí pre Linuxové operačné systémy sme zvolili iný prístup. Samotný operačný systém nám ponúka prostredie, v ktorom je možné sondu spustiť v režime práce na pozadí bez dodatočnej implementácie. Pre odstránenie vstupu a výstupu sme sondu spustili pomocou triedy BackgroundJob, ktorá túto funkcionality mala implementovanú už pre platformu Windows.

Na Linuxových operačných systémoch spúšťame sondu pomocou príkazu `dotnet`, za ktorý ako argument vložíme cestu ku spúšťaciemu súboru sondy. V prípade, že používateľ sondu spúšťa v režime práce na pozadí musí použiť niekoľko ďalších príkazov, aby sonda ostala spustená aj po zatvorení terminálu. Spúšťanie v takomto prípade vyžaduje od používateľa určitú úroveň skúsenosti s Linuxovými operačnými systémami.

Pre zjednodušenie spúšťania sondy sme vytvorili dva skripty. Skript `katalyzer_start.sh` spustí sondu v konzolovom režime, pričom argumenty, s ktorými je skript spustený, budú posunuté na spracovanie sondy. Druhý skript `katalyzer_background_start.sh` slúži na spustenie

v režime práce v pozadí. Použili sme príkaz `nohup` aby sme zabránili vypnutiu aplikácie po zavretí terminálu. Výstup z programu ignorujeme.

5.2 Konfigurácia ELK Stack

Technológii Docker sme museli zadať postup, podľa ktorého kontajner ELK Stack nakonfiguruje. Ten je rozdelený do dvoch súborov:

- `docker-compose.yml`
 - Nastavenia sieťových portov
 - Odkazuje na súbor, pomocou ktorého podrobne nastavíme kontajner
- `Dockerfile`
 - Obsahuje príkazy, ktoré sa majú vykonať po vytvorení kontajnera

5.2.1 Logstash

Aby sme dosiahli požadované správanie rozhrania Logstash, museli sme mu ho najskôr zadať pomocou konfiguračného súboru `logstash.conf`. Jeho obsah je rozdelený do troch sekcií:

- `input`
 - Povolenie HTTP vstup v JSON formáte na konkrétnom porte
- `filter`
 - Definujeme úpravy vstupov
 - Nastavenie cieľového indexu pre Elasticsearch
- `output`
 - Definujeme výstupy z Logstash

Pri vytváraní Docker kontajnera sa potom konfiguračný súbor presunie na miesto, odkiaľ ho Logstash načíta. Presunutie je zadefinované v súbore `Dockerfile`.

5.2.2 Kibana a Elasticsearch

Kibana ani Elasticsearch si pre správne fungovanie celého systému, okrem nastavenia portov, nevyžadovali žiadnu dodatočnú konfiguráciu.

6 Overenie riešenia

Riešenie sme testovali už počas vývoja, ale pre overenie nášho finálneho riešenia sme navrhli testy, pomocou ktorých sme overili funkčnosť jednotlivých častí. Na záver sme naše riešenie porovnali s najrozšírenejším sieťovým analyzátorom Wireshark.

6.1 Testovanie

6.1.1 Testovanie sprievodcu pre vytvorenie konfigurácie

Po spustení sondy v konzolovom režime sa otvorí hlavné menu. To obsahuje voľbu pre vytvorenie novej konfigurácie pomocou implementovaného sprievodcu. Pri testovaní sme pomocou sprievodcu vytvorili a uložili dve rôzne konfigurácie. Potom sme skontrolovali, či sa parametre konfigurácii zhodujú s parametrami, ktoré sme zadali pri vytváraní konfigurácií.

Pri kontrole konfigurácií sme zistili, že obe konfigurácie boli vytvorené podľa zadaného vstupu od používateľa, čím sme overili funkčnosť sprievodcu.

6.1.2 Testovanie základnej funkcionality

Pri teste základnej funkcionality sme na jednom zariadení spustili dve sondy v konzolovom režime, bez spúšťacích parametrov. Pre overenie konfigurovateľnosti sme im načítali rozdielne konfigurácie cez príslušnú voľbu v hlavnom menu. Po úspešnom načítaní konfigurácie sme pokračovali spustením zachytávania sieťovej prevádzky na oboch sondách. Konfigurácie sond sú popísané v tabuľke.

	Prvá sonda	Druhá sonda
Interval odosielenia	5 sekúnd	5 sekúnd
Názov zariadenia	ConfTest1	ConfTest2
Monitorované adaptéry	Všetky	Len jeden sieťový adaptér
Monitorované protokoly	Všetky	Ethernet, IPv4, IcmpV4
Adresa servera	http://mystellar.tools:31311	http://mystellar.tools:31311

Tabuľka 2 Testovacie konfigurácie pre sondy

O funkčnosti načítavania názvu zariadenia, intervalu na odosielenie štatistík a adresy servera, na ktorý odosielame informácie o sieťovej prevádzke, sme sa presvedčili už pri implementácii. Preto sa pri testovaní týmto parametrom nebudeme venovať v takej miere, ako zoznamu monitorovaných sieťových kariet a protokolov.

Pri oboch testoch sme informácie o zachytenej sieťovej prevádzke kontrolovali v nástroji Kibana. V nástroji Kibana sa pre sondy zobrazovali rôzne výsledky spôsobené rozdielnou konfiguráciou. Takto sme overili funkčnosť konfigurovateľnosti, vytvárania štatistík a ich ukladanie do Elasticsearch.

6.1.3 Test spracovania argumentov

Výsledky spracovania argumentov sme najskôr kontrolovali pomocou ladiaceho nástroja Visual Studio, kde sme pozorovali zmeny vo vykonávaní zdrojového kódu po zadaní jednotlivých spúšťacích argumentov. Takto sme sa uistili, že boli vykonané správne vetvy programu, overovali sme ich funkčnosť na základe akcií sondy.

Argument	Výsledok
Žiadny	Spustenie v konzolovom režime
-h	Vypísanie návodu nasledujúce ukončením
-t	Vytvorenie šablóny config.json
-c	Načítanie konfiguráciu
-b	Aktivácia režimu pre prácu v pozadí

Tabuľka 3 Testovanie spúšťacích argumentov

Pri testovaní spracovania spúšťacích argumentov sme neodhalili žiadne nedostatky, čím sme overili funkčnosť modulu.

6.1.4 Test režimu práce na pozadí

Režim sme museli otestovať samostatne na oboch cieľových platformách, pretože jeho implementácia bola platformovo závislá. Pre obe platformy sme si pripravili konfiguračné súbory, ktoré sme vytvorili pomocou sprievodcu.

Linux

Sondu sme spustili pomocou skriptu určeného pre spustenie sondy na pozadí. Po spustení sa vytvoril nový proces sondy, ktorý sme mali možnosť vidieť vo výpise procesov. Sondu sme kontrolovali aj po vypnutí terminálu pomocou nástroja Kibana, v ktorom sme videli, že stále zachytáva prevádzku.

Windows

Pri platforme Windows spúšťame sondu ako službu operačného systému. Najskôr sme teda vytvorili službu s oprávneniami potrebnými na zachytávanie sieťovej prevádzky.

Následne sme službu spustili. Funkčnosť služby sme skontrolovali v nástroji Kibana, kde sme videli informácie o zachytenej sieťovej prevádzke.

Pri testovaní režimu práca na pozadí sme oboch platformách neodhalili nedostatky. Testami sa nám podarilo overiť funkčnosť režimu práce na pozadí pre obe platformy.

6.2 Porovnanie výsledkov s nástrojom Wireshark

Naše výsledné riešenie sme porovnávali s najpoužívanejším nástrojom na analýzu sieťovej prevádzky, Wireshark, a to tak, že na jednom zariadení sme súbežne spustili zachytávanie pomocou vytvorenej KaTaLyzer sondy a monitorovacieho prostriedku Wireshark. Po približne desiatich minútach sme zachytávanie v nástroji Wireshark zastavili.

V nástrojoch Wireshark a Kibana sme vyfiltrovali údaje zaznamenané za sledovaný časový interval o veľkosti desať minút. Po zvolení časového okna sme informácie o zachytenej prevádzke v oboch nástrojoch filtrovali podľa konkrétneho protokolu. Výsledky testovacích meraní sme zaznamenali do nasledujúcej tabuľky. Testovanie prebehlo na oboch platformách.

Namerane hodnoty v bajtoch				
	Protokol	KaTaLyzer	Wireshark	Rozdiel hodnôt
Windows	Ethernet	363405391	363476369	0.020%
	IPv4	363233857	363419137	0.051%
	IPv6	31608	33416	5.411%
	TCP	8658315	8722649	0.738%
	UDP + GoogleQUIC	354607150	354611546	0.001%
	ARP	21546	23220	7.209%
	ICMP	219706	219706	0.000%
Linux	Ethernet	892226	893484	0.141%
	IPv4	843325	866905	2.720%
	IPv6	21955	22181	1.019%
	TCP	435544	435544	0.000%
	UDP + GoogleQUIC	429736	429736	0.000%
	ARP	3402	4338	21.577%
	ICMP	22200	22200	0.000%

Tabuľka 4 Porovnanie s programom Wireshark

Zhodnotenie

Cieľom bakalárskej práce bolo analyzovať možnosti meracieho prostriedku KaTaLyzer a na základe vykonanej analýzy následne navrhnúť a implementovať jeho vylepšenie alebo rozšírenie. Po vykonaní analýzy súčasného stavu nástroja KaTaLyzer sme sa rozhodli vylepšiť tento nástroj jeho kompletným prepracovaním s využitím modernejších technológií.

Oproti pôvodnej verzii, nové navrhnuté, implementované a otestované riešenie obsahuje aj podstatné rozšírenia funkcionality meracieho prostriedku. Najvýraznejším rozšírením je, že novú verziu je možné využívať nie len na platforme Linux, ale aj Windows. Toto rozšírenie rapídne zvýšilo počet potencionálnych používateľov, nakoľko platforma Windows patrí medzi najrozšírenejšie vo svete. Nové riešenie rozlišuje oproti pôvodnému taktiež väčší počet protokolov a bolo vytvorené s dôrazom na využitie vhodných technológií.

Presnosť novovytvoreného nástroja bola otestovaná porovnaním zachytených hodnôt vyfiltrovaných pre jednotlivé protokoly s hodnotami nameranými pomocou konkurenčného nástroja Wireshark.

Výsledky meraní na platforme Windows ukázali priemerný rozdiel nameraných hodnôt medzi oboma nástrojmi iba 1,92%. Na základe výsledkov meraní, ktoré sme realizovali na platforme Linux, bol stanovený rozdiel medzi hodnotami novej verzie KaTaLyzer a nástroja Wireshark o trochu väčší, priemerne 3,64%. Najväčšie rozdiely sa vyskytli pri zachytávaní protokolu ARP na platforme Windows. Celkovo je však možné zhodnotiť, že priemerné rozdiely medzi nameranými hodnotami sú minimálne. Tieto zistenia sú dôkazom presnosti a funkčnosti novej verzie monitorovacieho prostriedku KaTaLyzer.

Na základe uvedeného považujeme cieľ bakalárskej práce za splnený a výsledkom práce je nová verzia KaTaLyzer, ktorá je funkčná, vytvorená pomocou modernejších technológií a schopná fungovať na väčšom počte zariadení.

Literatúra

1. **The PHP Group.** Supported Versions. [Online] [Dátum: 28. Apríl 2019.] <http://php.net/supported-versions.php>.
2. **Wireshark Foundation.** Wireshark Frequently Asked Questions. [Online] 1. Apríl 2019. [Dátum: 1. Máj 2019.] <https://www.wireshark.org/faq.html>.
3. **SolarWinds Worldwide, LLC.** IT Management Software & Monitoring Tools. [Online] [Dátum: 1. Máj 2019.]
4. **Solarwinds.** Set up network devices to export NetFlow data. [Online] [Dátum: 28. Apríl 2019.] <http://www.solarwinds.com/documentation/en/flarehelp/netflow/content/nta-setting-up-network-devices-to-export-netflow-data-manually-sw75.htm?cshid=OrionNetFlowAGSettingNetFlow>.
5. **Paessler AG.** PRTG Network Monitor. [Online] [Dátum: 1. Máj 2019.] <https://www.paessler.com/prtg>.
6. **ExternCode.** C Standards – Story of C from ANSI C to C18. [Online] 19. September 2018. [Dátum: 28. Apríl 2019.] <https://www.externcode.com/c-standards/>.
7. **trytoprogram.** History of C++. [Online] [Dátum: 28. Apríl 2019.] <http://www.trytoprogram.com/cplusplus-programming/history/>.
8. **OpenEDG Python Institute.** What is Python? [Online] [Dátum: 28. Apríl 2019.] <https://pythoninstitute.org/what-is-python/>.
9. **Mercer, Jamie.** A Short History of Java. [Online] 23. Máj 2017. [Dátum: 28. Apríl 2019.] <https://dzone.com/articles/a-short-history-of-java>.
10. **Microsoft.** What is .NET? [Online] [Dátum: 1. Máj 2019.] <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet>.
11. **MongoDB, Inc.** Relational Vs Non Relational Database. [Online] [Dátum: 28. Apríl 2019.] <https://www.mongodb.com/scale/relational-vs-non-relational-database>.
12. **i'm programmer.** BEST NOSQL DATABASES 2019 – MOST POPULAR AMONG PROGRAMMERS. [Online] 1. Január 2019. [Dátum: 28. Apríl 2019.] <https://www.improgrammer.net/most-popular-nosql-database/>.

13. **MongoDB, Inc.** What Is A Non Relational Database. [Online] [Dátum: 28. Apríl 2019.] <https://www.mongodb.com/scale/what-is-a-non-relational-database>.

14. **General.** NoSQL Performance Benchmark 2018 – MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB. [Online] 14. Február 2018. [Dátum: 28. Apríl 2019.] <https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/>.

15. **opensource.com.** What are Linux containers? [Online] [Dátum: 28. Apríl 2019.] <https://opensource.com/resources/what-are-linux-containers?intcmp=7016000000127cYAAQ>.

16. **Elastic.** What Is a Document? [Online] [Dátum: 29. Apríl 2019.] <https://www.elastic.co/guide/en/elasticsearch/guide/current/document.html>.

17. —. Data In, Data Out. [Online] [Dátum: 29. Apríl 2019.] <https://www.elastic.co/guide/en/elasticsearch/guide/current/document.html>.

18. —. Http input plugin. [Online] [Dátum: 29. Apríl 2019.] <https://www.elastic.co/guide/en/logstash/current/plugins-inputs-http.html>.

19. **json.org.** Introducing JSON. [Online] [Dátum: 29. Apríl 2019.] <https://www.json.org/>.

Príloha A: Plán práce - zimný semester

Plán práce

Týždeň semestra	Cieľ	Výsledný stav
1-3	Nasadenie a analýza pôvodnej verzie KaTaLyzera	Splnené
4-5	Analýza dostupných technológií	Splnené
6-7	Návrh prototypu	Splnené
8-9	Zachytávanie sieťovej prevádzky	Splnené
10-11	Perzistovanie štatistík	Nesplnené
12	Dokumentácia k prototypu	Splnené

Vyhodnotenie

Cieľom práce na zimný semester bolo vytvoriť prototyp pre platformu Windows, ktorý bude schopný naplňať databázu už existujúceho riešenia. Kvôli nedostatočnej analýze existujúceho riešenia sme príliš neskoro narazili na problémy spojené s ukladaním štatistík do databázy. To spôsobilo, že plán sa nám nepodarilo splniť podľa očakávaní.

Príloha B: Plán práce - letný semester

Plán práce

Týždeň semestra	Cieľ	Výsledný stav
1-2	Analýza technológií pre nový KaTaLyzer	Splnené
3	Vytvorenie špecifikácie pre nový KaTaLyzer	Splnené
4	Vytvorenie nového prototypu	Splnené
5	Návrh riešenia	Splnené
6-10	Implementácia	Splnené
11	Testovanie	Splnené
12	Práca na dokumente	Splnené

Vyhodnotenie

Pôvodnú verziu sme v zimnom semestri nedostatočne analyzovali, čo nás dostalo do veľkého časového sklzu. Tento sklz sa nám cez letný semester podarilo nakoniec dohnať a plán práce sme splnili.

Príloha C: Používateľská príručka

Obsahom používateľskej príručky sú návody na inštaláciu, konfiguráciu a ovládanie jednotlivých komponentov. Používateľská príručka bola vytvorená pre operačné systémy Windows a Linux. Kroky pri inštalácii technológií na iných distribúciách operačného systému Linux sa môžu odlišovať a odporúčame postupovať podľa ich aktuálnej dokumentácie.

ELK Stack

Inštalácia technológie Docker

Pred inštaláciou ELK Stack pomocou technológie Docker je potrebné, aby samotná technológia bola prítomná na cieľovom zariadení spolu s nástrojom na vytváranie kontajnerov. Postup inštalácie je pre Windows a Linux odlišný. Pri ich inštalácii odporúčame postupovať podľa oficiálnych dokumentácií dostupných na nasledujúcich odkazoch:

- Jadro technológie – <https://docs.docker.com/install/>
- Nástroj na vytváranie kontajnerov - <https://docs.docker.com/compose/install/>

Inštalácia ELK Stack

Po inštalácii technológie Docker nasleduje vytvorenie kontajnera, v ktorom bude spustený celý ELK Stack. Kontajner vytvoríme postupným vykonaním nasledovných krokov:

1. Podľa platformy otvorte PowerShell alebo terminál ako administrátor
2. Krok pre platformu Linux
 - a. Otvorte súbor `/etc/sysctl.conf`
 - b. Zvýšte alebo doplňte premennú `vm.max_map_count` aspoň na 262144 (<https://www.elastic.co/guide/en/elasticsearch/reference/current/vm-max-map-count.html>)
 - c. Uložte zmeny
3. Prepnite sa do zložky s názvom „ElkDocker“
4. Spustite príkaz „`docker-compose up --force-recreate --build`“

Po treťom kroku Docker začne vytvárať kontajner podľa konfigurácie, ktorá sa nachádza v súbore `docker-compose.yml`. Operácia môže nejaký čas trvať a je dôležité sledovať hlásenia, ktoré sú vypisované. Vytvorený kontajner má otvorené porty všetkých komponentov, preto ho

neodporúčame používať v produkčnom prostredí bez úprav. Nižšie uvedené odkazy obsahujú užitočné informácie ohľadom vytvárania a nastavovania kontajnera.

- ELK Stack kontajner – <https://elk-docker.readthedocs.io/>
- Nastavenia – <https://docs.docker.com/compose/compose-file/>

Kibana

Po úspešnom vytvorení a spustení kontajnera ELK Stack bude nástroj Kibana dostupný na porte 5601. Nebude však obsahovať vizualizácie. Tie je potrebné manuálne vytvoriť, alebo importovať. Import je možný v sekcii Management => Kibana => Saved Objects, kde sa nachádza tlačidlo „Import“. Po kliknutí na tlačidlo „Import“ sa otvorí okno, kde je možné nahrať súbor „KibanaSettings.json“. Ten obsahuje nastavenia pre nástroj Kibana. Manuálne vytváranie vizualizácií je možné v sekcii Visualize. Z vizualizácií sa potom používateľ môže vytvoriť nástenku v sekcii Dashboard.

Nastavenia pre Kibana, ktoré ponúkame k riešeniu, obsahujú dve nástenky, ktoré je možno otvoriť v sekcii Dashboard. Tie poskytujú základne prehľady o logoch a zozbieraných štatistikách pomocou jednotlivých sond.

Sonda

Pre správnu funkčnosť sondy potrebujeme najskôr nainštalovať .NET prostredie a pcap knižnicu ktorá je platformovo špecifická. Samotnú sondu neinštalujeme, pretože spúšťacie súbory už máme k dispozícii.

Inštalácia pcap knižníc

Na platforme Linux pracujeme s knižnicou libpcap. Jej inštalácia je možná zadáním príkazu do terminálu. V prípade platformy Windows to je Npcap. Ten stiahneme z webovej stránky knižnice a nainštalujeme pomocou sprievodcu.

- Príkaz pre Linux – `sudo apt-get install libpcap-dev`
- Webová stránka knižnice Npcap – <https://nmap.org/npcap/>

Inštalácia .NET prostredia

Platforma Windows dôkaze neprítomnosť prostredia odhaliť a zobrazí sa vám sprievodca, pomocou ktorého chýbajúce časti nainštalujete. Pre Linux potrebujeme prostredie nainštalovať

manuálne. Inštalácia sa pre jednotlivé distribúcie operačného systému Linux odlišuje a odporúčame postupovať podľa oficiálneho návodu pre inštaláciu .NET Core 2.1 Runtime pre Linux, dostupného na odkaze uvedenom nižšie.

- Windows – Systém ponúkne inštaláciu chýbajúcich častí .NET Framework
- Linux – Je potrebná manuálna inštalácia prostredia podľa návodu na stránke <https://dotnet.microsoft.com/download/linux-package-manager/ubuntu18-04/runtime-2.1.2>, kde sa dá zvoliť konkrétna distribúcia operačného systému

Konfigurácia

Manuálna konfigurácia je možná spustením sondy s argumentom „-t“. Vytvorí sa súbor config.json. Ten bude obsahovať šablónu konfigurácie, ktorá je špecifická pre zariadenie. Tento súbor je potom potrebné dodatočne upraviť.

Sonda ponúka aj sprievodcu pre vytvorenie konfigurácie, ktorého je možné spustiť pomocou príslušnej voľby po zapnutí sondy bez argumentov. Sprievodca prevedie používateľa jednotlivými nastaveniami sondy a výslednú konfiguráciu uloží do súboru.

Pri vytváraní konfigurácie je potrebné nastaviť správnu adresu na server Logstash. V prípade, že ste postupovali podľa návodu a ELK Stack máte spustený ako Docker kontajner, ktorému ste neupravovali sieťové porty, adresu servera nastavte podľa nasledovného príkladu: <http://<adresa na server>:31311>, kde do <> uvediete skutočnú adresu na server.

Monitorovanie sieťovej prevádzky

Monitorovanie sa spúšťa pomocou voľby v hlavnom menu alebo zadáním argumentu určeného na spustenie sondy v režime práce na pozadí. Aby sonda mohla vykonávať monitorovanie, potrebuje mať načítaný konfiguračný súbor. Ten načítame pomocou spúšťačieho argumentu alebo pomocou voľby v hlavnom menu.

Režim prace na pozadí

Tento režim sa pre obe platformy aktivuje rozdielnym spôsobom. Pred jeho použitím v oboch prípadoch odporúčame aby sa používateľ overil správnosť konfigurácie cez štandardný režim.

Linux

Pre platformu odporúčame používať skript `katalyzer_background_start.sh`, ktorý sondu spustí na pozadí. Po jeho spustení sa nezobrazí žiadna informácia pre používateľa. Používateľ má možnosť kontrolovať funkčnosť prostredníctvom nástroja Kibana a logov, ktoré sa nachádzajú v súbore `logs.txt`.

Windows

Pre spustenie sondy v režime práce na pozadí je potrebné vytvoriť službu operačného systému. To dosiahneme zadáním „`sc.exe create KaTaLyzerService binPath= "<cesta k KaTaLyzerWindows.exe > -b""`“ do príkazového riadku spusteného s administrátorskými právami. Vytvorenie služby môžeme overiť vizuálnou kontrolou v nástroji na správu služieb. V nástroji môžeme tiež nastaviť aj ostatné parametre vytvorenej služby. Dôležité je uistiť sa, že služba je spúšťaná s oprávneniami dostačujúcimi na zachytávanie sieťovej prevádzky.

Príloha D: Opis digitálnej časti práce

Evidenčné číslo práce v informačnom systéme: FIIT-104199-79816

Obsah digitálnej časti práce (archív ZIP):

Priečinok:	Opis:
\ElkDocker	Súbory pre vytvorenie Docker kontajnera
\Instalation	Stručný postup inštalácie v anglickom jazyku
\KaTaLyzerSonda	Zdrojový kód pre sondu
\Dokumenty	Záverečná práca vo formáte PDF
\SondaLinux	Spúšťacie súbory pre Linuxovú verziu sondy
\SondaWindows	Spúšťacie súbory pre Windows verziu sondy

Názov odovzdaného archívu: BP_prilohy_digital_MichalZahradnik.zip