TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Python modul pro rychlé zjišťování informací o síťových zařízeních

Diplomová práce

Studijní program:	N2612 Elektrotechnika a informatika
Studijní obor:	Informační technologie
Autor práce:	**Bc. Lukáš Mázl**
Vedoucí práce:	Mgr. Jiří Vraný, Ph.D. Ústav nových technologií a aplikované informatiky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Python modul pro rychlé zjišťování informací o síťových zařízeních

Jméno a příjmení: **Bc. Lukáš Mázl**
Osobní číslo: M19000155
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Seznamte se s problematikou tvorby modulů pro jazyk Python pomocí jazyka C / C++ a s problematikou zjišťování informací o síťových zařízeních na OS Linux.
2. Porovnejte vybrané dostupné nástroje pro zjišťování informací o síťových zařízeních na OS Linux z hlediska rychlosti a škálovatelnosti při nasazení v systémech s velkým množstvím síťových zařízení, typicky v systémech s velkým množstvím Docker, Kubernetes či OpenShift kontejnerů.
3. S využitím získaných informaci navrhněte a implementujte Python modul pro rychlé zjišťování informací o síťových zařízeních v systémech s velkým množstvím síťových zařízení.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah pracovní zprávy: 40 – 50 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] BOVET, Daniel Pierre a Marco CESATI. Understanding the Linux kernel. 3rd ed. Sebastopol: O'Reilly, c2006. ISBN 05-960-0565-2.
[2] Python/C API Reference Manual [online]. Python Foundation, 2020 [cit. 2020-10-07]. Dostupné z: https://docs.python.org/3/c-api/index.html.

Vedoucí práce: Mgr. Jiří Vraný, Ph.D.
Ústav nových technologií a aplikované informatiky

Datum zadání práce: 19. října 2020
Předpokládaný termín odevzdání: 17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2020

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

16. května 2021 Bc. Lukáš Mázl

Python modul pro rychlé zjišťování informací o síťových zařízeních

Abstrakt

Cílem této diplomové práce je vytvořit rozšiřující modul pro jazyk Python, který by umožnil rychlé získání informací o síťových prvcích dostupných na zařízení. Hlavní vlastností nově vytvářeného modulu by měla být především rychlost, protože existuje již několik podobných modulů, ale většina z nich neumožní rychlé získání informací pro velký počet NIC. Již existující moduly jsou v této práci popsány, a to včetně popisu způsobu jejich přístupu k získávání informací.

Před vytvářením modulu muselo dojít k přípravě prostředí a skriptů k porovnání rychlosti jednotlivých existujících modulů. Skripty byly spuštěny několikrát pokaždé pro jiný počet fiktivních NIC definovaných na virtuálním stroji. Nový modul vytvořený v této práci využívá Python/C API, které umožňuje napsání kódu v jazyce C a následnou interpretaci pro jazyk Python. Výsledný modul je v porovnání s existujícími moduly výrazně rychlejší.

Práce se kromě vytvoření nového rychlého modulu zabývá i opravou chyby, která způsobuje pomalé načítání. Analýza a oprava byla poskytnuta formou pull requestu do projektu Netifaces.

Klíčová slova: **Python/C API, síťová rozhraní, NIC, Ethtool, Netifaces, Open source**

Python module for efficient network interface monitoring

Abstract

The aim of this thesis is to create an extension module for the Python language, which would allow fast retrieval of information about NIC available on the device. The main features of the created module should be high speed, because there are already several modules, but at first they will not allow fast fetching of information for a large number of NICs. Existing modules are described in this work and their way of approaching to obtain information.

Before creating a module, there must be a preparatory environment and scripts to compare the speed of each existing module. The scripts were run several times each time for a different number of fictitious NICs defined on the virtual machine. The new module created in this work uses the Python / C API, which allows writing code in C language and subsequent interpretation in Python. The resulting module is faster compared to existing modules.

In addition to creating a new fast module, the work also involves and corrects errors that load slowly. The analysis and correction was provided in the form of a pull request to the Netifaces repository.

Key words: **Python/C API, Network interface controller, NIC, Ethtool, Netifaces, Open source**

Poděkování

Děkuji vedoucímu diplomové práce Mgr. Jiřímu Vranému, Ph.D za vypsání tématu této diplomové práce a panu Ing. Jiřímu Hnídkovi Ph.D za cenné rady při vytváření nového modulu.

Obsah

Seznam obrázků

Seznam tabulek

Použité zkratky

NIC	Network interface controller
CLI	Command Line Interface
API	Application programable interface
DNS	Domain Name System
IP	Internet Protocol
REPL	Read–eval–print loop
OS	Operační systém

Jednotky

MB	megabyte
GB	gigabyte
MS	milisekund

Vysvětlení ostatních zkratek se nachází přímo v textu.

1 Úvod

Cílem této diplomové práce je vytvoření rozšiřujícího modulu pro jazyk Python, který by dokázal v krátké době získat informace o síťových prvcích dostupných na aktuálním zařízení. Zejména v rámci systémů s velkým počtem síťových rozhraní je potřeba vývoje nového modulu pro Python aktuální. Typickým příkladem takového prostředí může být server na hostující Docker kontejnery. Takové servery mohou mít několik set či tisíc síťových rozhraní a získání informací o všech těchto rozhraních může být časově náročné. Během testování stávajících Python modulů se ukázalo, že nejsou optimální. V některých případech trvalo získání informací o všech rozhraních i několik minut.

Problém, který je řešen v této diplomové práci, je zaměřen hlavně na servery běžící na linuxovém jádře. Proto došlo k rozhodnutí zaměřit se primárně na vývoj modulu pro tento operační systém. I přesto, že byl model vytvořen pouze pro linuxové prostředí, analýza a oprava Netifaces modulu, které jsou popsané v závěru této práce, pomohla vyřešit i problém na operačních systémech Windows a MacOS.

Tato práce ukazuje, jaké kroky jsou potřebné pro vytvoření vlastního Python modulu a k jeho instalaci do Python repositáře. Před začátkem samotné implementace byla provedena analýza již dostupných modulů, které dokážou vrátit seznam informací o rozhraních. Moduly zmíněné v této diplomové práci byly vybrány úmyslně proto, aby bylo možné ukázat princip získávání informací o rozhraní. Některé moduly mají podobný princip (např. využívají stejnou knihovnu), ale mění se u nich API.

V praktické části diplomové práce je vypracovaná analýza představující podobu modulu, který splňuje předpoklad rychlejšího procesu získávání informací než aktuální existující modely. Kromě této analýzy je uveden i postup testování jednotlivých modulů. Aby bylo možné existující skripty objektivně porovnat, bylo potřeba vytvořit testovací skripty. Kvůli rozdílným přístupům a jinému API bylo zapotřebí vytvořit pro každý modul jiný skript, který musel načtení všech rozhraní a informací k nim řešit svým vlastním individuálním způsobem.

Během testování rychlosti modulů bylo zjištěno, že nově vytvořený modul a modul Netifaces mají velmi rozdílné výsledky (v řádu minut), přičemž ale fungují na stejném principu. Na základě tohoto zjištění vznikla analýza problému Netifaces, která popisuje hlavní důvod neefektivity zmíněného modulu. Po vypracování analýzy byla vytvořena oprava, která byla nabídnuta formou pull requestu do veřejného repozitáře Netifaces.

2 Kontejnery a jejich orchestrace

V této diplomové práci je řešen problém, který byl objeven v prostředí Docker kontejnerů. Problém způsobil dlouhé čekání (v řádu minut) v systému s velkým počtem síťových rozhraní. Před začátkem řešení tohoto problému je vhodné představit kontejnery a jejich výhody. V této kapitole jsou popsány základní informace o Docker kontejnerech a platformách, které se starají o jejich orchestraci.

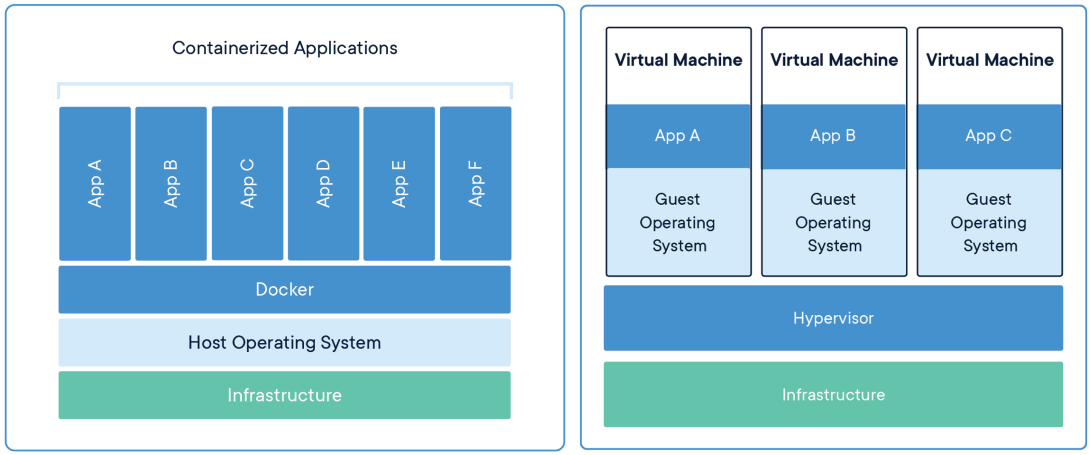
2.1 Docker

Docker je kontejnerizační nástroj navržený k vývoji, nasazování a běhu aplikací. Kontejnery umožňují vývojáři vytvořit balíček s aplikací a všemi jejími částmi, které jsou potřeba (knihovny, konfigurace). Takto vytvořený balíček je následně možné spustit v lokálním prostředí, nebo jej rovnou poslat k nasazení do Cloudu. Řeší tím problém se složitou přenositelností mezi jednotlivými prostředími. Pro vývojáře to znamená, že se mohou soustředit pouze na psaní kódu a nemusí se zabývat systémem, na kterém aplikace bude v budoucnu spuštěna.

Kontejnery jsou abstrakcí aplikační vrstvy, která seskupuje kód se závislostmi dohromady. Několik kontejnerů může běžet na stejném stroji a sdílet operační systém a jeho jádro s ostatními kontejnery. Každý kontejner běží v izolovaném procesu. Kontejnery zabírají méně místa než virtuální stroje (kontejnerové obrazy mají typicky velikost do desítek MB), kromě toho mohou také zvládat více aplikací najednou. [1]

Virtuální stroje jsou abstrakcí na fyzickém hardwaru, umožnující z jednoho stroje

vytvořit několik severů. Hypervisor umožňuje několika virtuálním strojům běžet na jednom fyzickém stroji. Každý virtuální stroj má svůj vlastní operační systém, aplikace a nezbytné binární soubory. Knihovny obsahují několik desítek GB a z toho důvodu mohou být pomalejší při startu. [1]

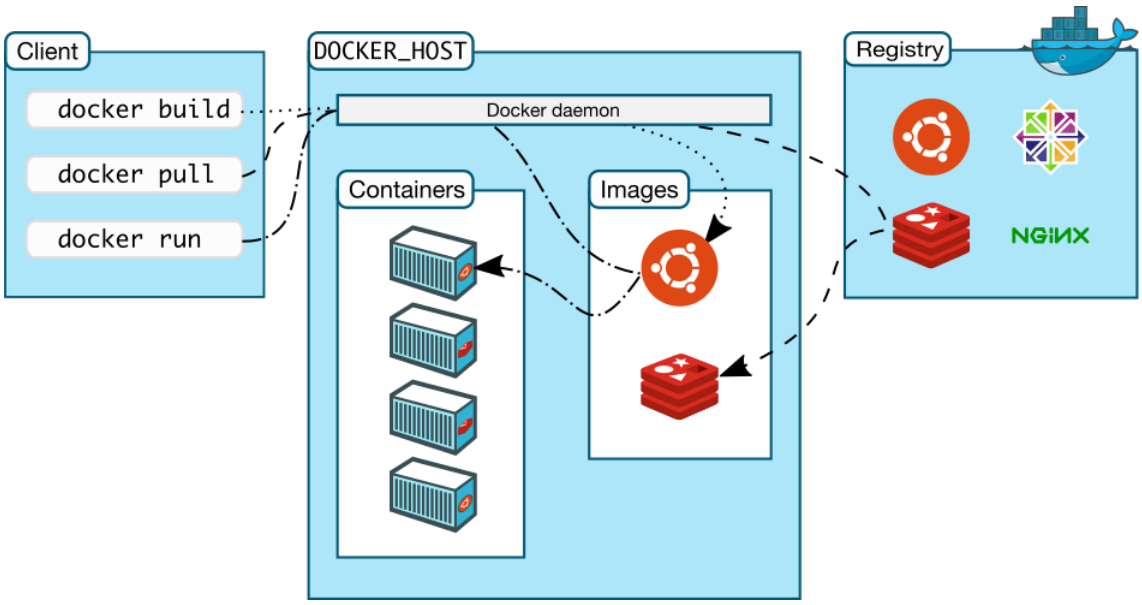


(a) Architektura Docker systému (b) Architektura virtuálního stroje

Obrázek 2.1: Ukázka rozdílu architektur mezi Docker a Virtuálním strojem [1]

Celý Docker se skladá z následujících komponent:

- **Daemon** - Čeká na požadavky z API a stará se o Docker objekty, kterými jsou obrazy (images), kontejnery, sítě a datové svazky. Docker daemon může komunikovat s ostatními daemony za účelem hromadného spravování Docker service.
- **Klient** - Docker klient je primární způsob komunikace s Docker daemonem. Když je použit příkaz *docker run*, klient odešle příkaz Docker daemonovi, který jej vykoná. Klient může komunikovat s více než jedním daemonem.
- **Registry** - Registry ukládají Docker obrazy (images). Docker Hub je veřejným registrem, takže ho může používat každý. Docker je nakonfigurován tak, aby bylo možné do tohoto repozitáře automaticky nahlédnout v případě, kdy nenajde obraz lokálně. Mimo tohoto registru je možné nakonfigurovat i své vlastní privátní registry.
- **Objekty** - Při používání Dockeru jsou vytvářeny a používány obrazy, kontejnery, sítě, datové svazky, pluginy a jiné objekty. Tyto objekty jsou dále popsány zde:



Obrázek 2.2: Architektura Docker systému [2]

– **Image** - Image nebo v překladu obraz je šablona (template) určená pouze pro čtení. Tato šablona obsahuje informace podle nichž je vytvořena instance kontejneru. V konfiguraci může být nastaveno na jakém základě má být image postavena (např. *ubuntu*) nebo jaké aplikace v ní poběží.

– **Kontejner** - Kontejner je instancí image, se kterou je manipulováno pomocí CLI nebo Docker API. Kontejner může být nastartován, zastaven, přemístěn nebo smazán. Je možné uložit aktuální stav kontejneru jako image a podle této image následně vytvářet další kontejnery.

Docker využívá klient-server architekturu. Docker klient komunikuje s Docker daemonem, který sestavuje, spouští a rozmisťuje kontejnery. Klient a daemon mohou běžet na stejném systému nebo spolu mohou vzdáleně komunikovat. Ke komunikaci je využíváno REST API, UNIX soket nebo síťové rozhraní. Mezi další klienty patří Docker Composer, který umožňuje spustit aplikaci skládající se z více kontejnerů. Na obrázku 2.2 je znázorněna komunikace mezi klientem, daemonem a registrem obrazů. [2]

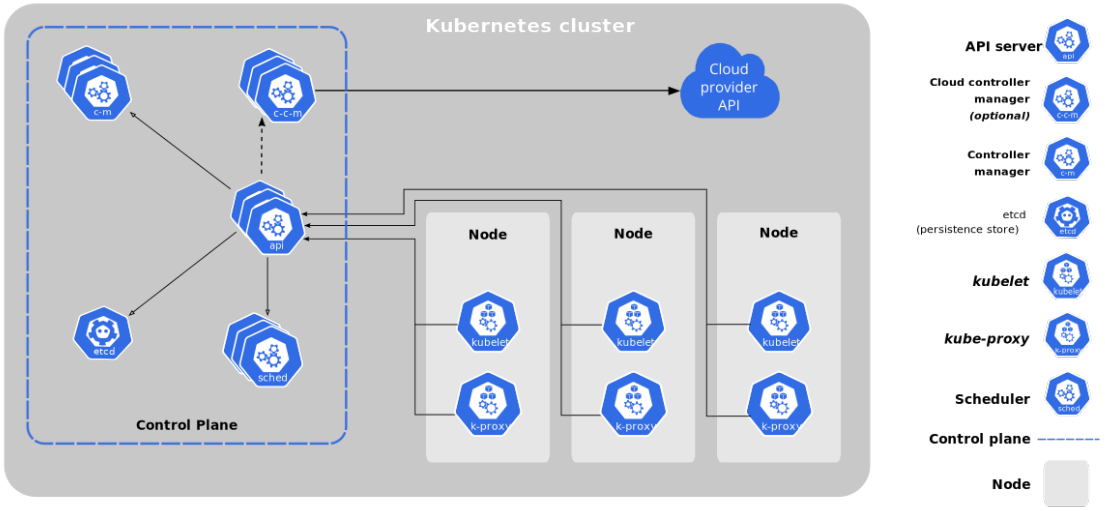
2.2 Kubernetes

Název Kubernetes v originále pochází z Řecka a znamená kormidelník nebo pilot. Google v roce 2014 vytvořil open-source projekt, jehož součástní bylo i Kubernetes. V tomto projektu Google zkombinoval přes 15 let zkušeností s během aplikací v produkci s nejlepšími nápady a postupy z komunity. [3]

Kontejnery jsou dobrým způsobem jak sestavit a spustit aplikaci. V produkčním prostředí je potřeba tyto kontejnery spravovat a mít jistotu, že nedojde k jejich výpadku. Další problém v produkci může nastat například v případě, kdy jeden kontejner přestane fungovat, což zapůsobí i na jiný kontejner, který se, pro zachování správné funkčnosti, potřebuje restartovat. Aby došlo k vyřešení těchto problému vznikl projekt Kubernetes. Ten poskytuje následující možnosti:

- **Objevování služeb a load balancing** - Kubernetes umí odhalit kontejnery na základně DNS jména nebo jejich IP adresy. Pokud je využití kontejneru jinými systémy vysoké, je schopný tuto zátěž rozprostřít na jiný kontejner v síti (load balancing).
- **Self-healing** - Kubernettes umožňuji restartovat kontejner který přestane fungovat, nebo ukončit jeho běh, pokud neodpoví v požadovaném čase.
- **Automatický rollout a rollback aplikace** - Kubernetes umožňují popsat jak má vypadat výsledný stav kontejnerů. Zjednodušují nasazení, nebo případný rollback aplikace. Automaticky vytváří nové kontejnery pro nasazení, odebírá existující a předává existující prostředky novému kontejneru.
- **Orchestrizace uložiště** - Kubernetes umožňuje automatické napojení úložiště systému, kterými jsou lokální úložiště, veřejní poskytovatelé cloud služeb.

V okamžiku nasazení a spuštění Kubernetes vzniká cluster. Celý cluster je možné rozdělit na část výpočetní a část kontrolní. Ve výpočetní části jsou uzly (nodes), na kterých běží kontejnerizované aplikace. Kontrolní části (control plane) vyhodnocují a vytváří globální rozhodnutí, která ovlivňují cluster. Příkladem takového vyhodnocení může být detekování nedostupnosti uzlu a následného



Obrázek 2.3: Architektura Kubernetes systému [4]

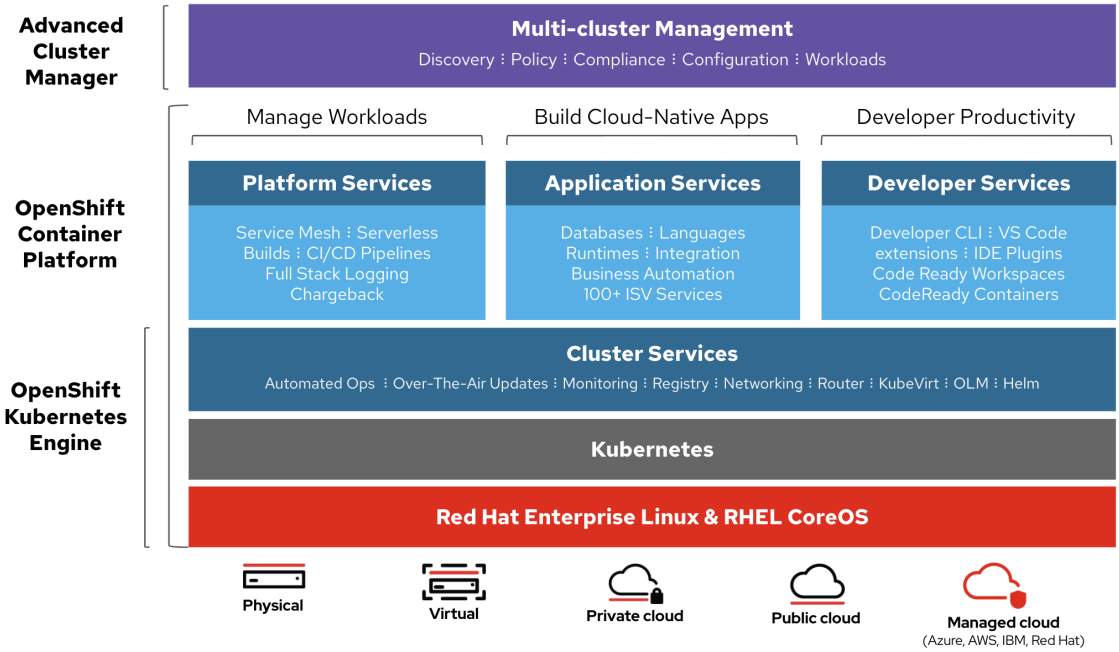
vyhodnocení, že uzel nebo nějaká z jeho aplikací by měla být kvůli nedostupnosti odebrána, nebo restartována. Celá architektura Kubernetes je znázorněna na obrázku 3.2. [4]

2.3 OpenShift Container Platform

OpenShift Container Platform je platforma pro vývoj a provoz kontejnerových aplikací. Je navržena tak, aby umožňovala aplikacím a datovým centrům, která ji využívají, expanzi z několika strojů a aplikací na tisíce strojů. Tato platforma je založena na základních principech Kubernetes. Na stejné technologii, která slouží jako základ pro masivní telekomunikaci, streamování videí, bankovnictví a jiné aplikace. [5]

OpenShift Container Platform poskytuje podnikům vylepšení Kubernetes včetně následujících vylepšení:

- **Hybridní cloud** - Clustery mohou běžet na veřejných cloudových platformách a nebo v privátním data centru.



Obrázek 2.4: Architektura OpenShift Container Platform [6]

- **Integrované Red Hat technologie** - Hlavní komponenty v OpenShift Container Platform vychází z Red Hat Enterprise Linux a příbuzných Red Hat technologií.
- **Open source** - Zdrojové kódy jsou veřejně dostupné, což způsobuje rychlou inovaci i rychlost vývoje.

Ačkoliv Kubernetes exceluje ve správě aplikací, neurčuje ani nespravuje požadavky na úrovni platformy (ani procesy nasazení). OpenShift Container Platform využívá operační systém RHCOS (Red Hat Enterprice Linux CoreOs), který je navržen a přizpůsoben k tomu, aby pracoval s aplikacemi a kontejnery. Díky čemuž vytváří nové nástroje pro rychlejší instalaci kontejnerové aplikace. [5]

3 Vytváření rozšiřující modulů pro Python v jazyce C/C++

Oblíbenost jazyka Python v posledních letech roste. Python je často využíván pro matematické výpočty nebo pro strojové učení. Část Python modulů je napsána v jazyce C/C++, což umožňuje pracovat na nižší programové úrovni - a navíc mnohem rychleji. Pokud existuje knihovna napsaná v C/C++ je možné funkce propojit pomocí Python/C API. Výhodou takového postupu je větší rychlost. Toto řešení ovšem vede ke složitějšímu ladění případných chyb, které nastanou v Části kódu napsaného v C/C++.

V této kapitole je popsána práce s Python/C API a postup pro vytvoření vlastního modulu. Jsou zde znázorněny potřebné části, které jsou důležité pro vytvoření nového modulu a jeho instalaci.

3.1 Python/C API

Python/C API umožňuje propojení části kódu napsaného v jazyce C/C++ s Python kódem. Pro využití tohoto propojení existují minimálně tři důvody.

První důvod byl již zmíněn výše. Pokud existuje knihovna, která řeší problém pro který není ve standardní Python knihovně řešení, je možné vytvořit C program který zpřístupní funkce dané knihovny a umožní provolání funkci z Pythonu. Příkladem takovéto knihovny je knihovna komunikující se specifickou částí hardwaru.

Druhým důvodem je zrychlení bloku kódu napsaného v Pythonu. Jazyk Python je interpretovaný, to znamená že kód je nejdříve přeložen do mezi-kódu, který je potom spuštěn na interpretovi. Pokud použijeme knihovnu/kód napsaný v C/C++ dojde ke zrychlení, protože kód nemusí být nadále kompilován/překládán. Kód je kompilován pouze při instalaci modulu.

Třetím důvodem je využití Python jako skriptovacího jazyka. Pokud spustíme Python, pustí se REPL (read, eval, print, loop) umožňující vkládat postupně příkazy k provedení a jejich výsledky po vykonání vypíše, následně pokračuje čekáním na další příkaz. Vytvoříme-li nový modul, můžeme REPL využívat pro C/C++.

Pro práci s API je potřeba knihovna *Python.h*, která je dostupná v Python balíčcích pro vývojáře (python-dev). Jazyk Python je objektově orientovaným, jazyk C takovým jazykem není. Jak je tedy možné mezi těmito jazyky předávat data? Knihovna Python.h tento problém řeší pomocí struktury PyObject. PyObject je základní struktura, pomocí které se předávají data mezi C kódem a Python kódem. Pokud vytváříme funkci v C a definujeme návratový typ, tak tímto návratovým typem by měl být pointer právě na tuto strukturu. [7]

Pro vytvoření nového modulu jsou potřebné tyto 3 části:

- implementace logiky
- definice modulu
- zavaděč modulu pro kompilaci a instalaci

Pokud jsou vytvořeny zmíněné tři části, je možné modul nainstalovat a používat. Definice modulu a zavaděče budou detailněji popsány v následujících sekcích.

3.2 Definice modulu

První volající se funkce při inicializaci nového modulu je *PyInit_ModulName*. Ta má za úkol inicializovat celý modul a vrátit *PyMOINIT_FUNC*, která je získána po zavolání funkce *PyModule_Create()*. Vstupní hodnotou do funkce *PyModule_Create()* je struktura *PyModuleDef*.

Struktura *PyModuleDef* definuje všechny informace o nově vytvářeném modulu. Ve většině případů je pouze jedna definice v celém kódu. Může ale dojít k výjimkám, kde je potřeba více těchto definic (např. rozdílné hlavní verze Pythonu). Pro *PyModuleDef* je možné nastavit tyto proměnné.

- **PyModuleDef_Base m_base** - PyModuleDef_HEAD_INIT je jediná možná hodnota.
- **const char *m_name** - Název modulu.
- **const char *m_doc** - Mokumentace k modulu.
- **Py_ssize_t m_size** - Pokud je nastavena záporná numerická hodnota, znamená to, že není využíváno sub-interpretů a je použit pouze globální interpret.
- **PyMethodDef* m_methods** - Odkaz na strukturu PyMethodDef, ve které jsou definované metody modulu.

Na obrázku 3.1 je zobrazen příklad definice modulu, která umožňuje vytvoření modulu ModuleName.

```c
static struct PyModuleDef ModuleNameDef =
    {
        PyModeleDef_HEAD_INIT,
        "Module_name",
        "Dokumentation",
        NULL,
        -1,
        module_functions
    };

PyMODINIT_FUNC PyInit_moduleName(void)
{
    return PyModule_Create(&ModuleNameDef);
}
```

Obrázek 3.1: Příklad definice nového Python modulu v jazyce C

Definice funkcí pro modul je podobná definici celého modulu. Pro vytvoření je potřeba vytvořit *PyMethodDef* strukturu a tu naplnit daty. Celkem je potřeba pro definici jedné funkce vyplnit 4 vstupní parametry zobrazené zde:

- Název metody
- Ukazatel na implementaci funkce
- Příznak indikující jak má dojít k provolání funkce
  - **METH_NOARGS** - Funkce je bez parametrů. Není potřeba kontrolovat, zda je na vstupu nějaká hodnota.
  - **METH_VARARGS** - Funkce očekává na vstupu 2 parametry. První je *self* objekt, druhý (často nazvaný *args*) je tuple objekt reprezentující všechny argumenty. Tyto parametry je možné získat pomocí *PyArg_ParseTuple()* nebo *PyArg_UnpackTuple()*
  - **METH_FASTCALL** - Na vstupu funkce jsou 3 parametry. První je *self* objekt, druhý je pole objektů, třetí velikost pole.
- ukazatel na definici dokumentace k funkci

3.3 Zavaděč Python modulu

Úkolem zavaděče je připravit prostředí pro kompilaci modulu napsaného v C a tento nový modul nainstalovat. Často se soubor s tímto zavaděčem pojmenovává *setup.py*. Soubor *setup.py* je spuštěn, když má dojít k instalaci modulu do systému. Instalace zahrnuje kontrolu závislostí na knihovnách nebo operačním systému. Po dokončení kontroly nastává kompilace zdrojového C kódu.

Jazyk Python je multiplatformní, tzn. může běžet na více platformách. Z pohledu operačních systémů se může jednat o platformy Windows a Linux, nebo z pohledu hardwarových platforem o ARM nebo x86-64. V takovém případě se mohou lišit i knihovny pracující s danou platformou. Při vytvoření modulu je potřeba na tento fakt myslet a přizpůsobit kód. Jedním z řešení pro vytvoření multiplatformního modulu je zapojení C maker a příkazu #ifdef. Kód dostupný pro Linux bude definovaný v těle #ifdef stejně tak, jak je vidět na kódu uvedeném níže.

```
#if defined(HAS_IFADDRESS)
  // code for Linux
#else
  // code for Windows
#end
```

Obrázek 3.2: Ukázka využití direktiv v jazyce C

V případě, že makro není definováno, překladač tuto část kódu ignoruje a nezkompiluje. Direktivum *HAS_IFADDRESS* nemusí být definováno v C souboru, ale může být definováno právě souborem *setup.py*, který ho před kompilací nastaví/nenastaví podle splněných kriterii. Podle splněných podmínek by v *setup.py* měla být nastavena direktiva pro překlad a kompilaci. Kromě definice těchto direktiv je dalším podstatným krokem definice příslušných knihoven třetích stran, které musí být v okamžiku kompilace dostupné.

Pokud jsou všechny kontroly v pořádku, může dojít k samotné kompilaci a následné instalaci modulu. Pro tyto kroky je možné použít modul setuptool a jeho metodu setup. Vstupními parametry do metody setup jsou informace o modulu, zdrojové kódy, nebo skripty patřící k modulu.

Po spuštění příkazu ***python*** *setup.py install* dojde k instalaci modulu. Nový modul bude následně dostupný v prostředí Python.

4 Existující moduly pro získávání informací o síťových rozhraních

Tato kapitola je věnována již existujícím modulům, které jsou veřejně dostupné pro Python. Vybrány jsou ty moduly, na nichž lze demonstrovat rozdílný přístup získávání informací. Na konci každé sekce je uvedena tabulka zobrazující hlavní aspekty modulu, kterými jsou například velikost API (počet funkcí, které modul poskytuje), aktuální verze modulu nebo podporované verze Pythonu.

4.1 CTypes

Jedním ze způsobů řešení problému s načítáním informací o síťových rozhraních je CTypes, který je dostupný přímo v jazyce Python. V tomto případě se nejedná o modul, který by byl přímo zaměřený na získávání informací o rozhraních, ale umožňuje dynamické linkování knihoven, pomocí kterých lze tyto informace získat. Pomocí CTypes je možné využít systémovou knihovnu řešící přístup k NIC.[8]

Pokud bychom chtěli využít systémovou knihovnu k získání informací o rozhraní, muselo by dojít k těmto krokům:

- importování ctypes modulu
- importování systémové knihovny
- připravení parametrů
- provolání funkce

- transformace výsledků

Tento způsob načítání síťových rozhraní ale zahrnuje nutnost předešlé znalosti knihovny a struktur, se kterými má na úrovni knihovny komunikovat. Z tohoto důvodu není tento princip optimální. Důvodem je především časové hledisko - vývojář pravděpodobně stráví více času samostudiem, přičemž ve výsledku dojde k vytvoření modulu, který už někdo vytvořil. V tomto konkretním případě se může jednat o modul Ifaddr, jenž funguje popsaným způsobem.

V této práci je využíván hlavičkový soubor ifaddrs.h, který je součástí systémových knihoven Linuxu. Z tohoto důvodu je této knihovně věnována větší část textu, aby bylo možné pochopit, kterými daty a strukturami se práce zabývá.

4.1.1 ifaddrs.h

Pro linuxovou distribuci existuje knihovna *ifaddrs.h*, která umožňuje přístup k informacím o síťových rozhraních. Knihovna má následující funkce:

- **int** getifaddrs(**struct** ifaddrs **ifap)
- **void** freeifaddrs(**struct** ifaddrs *ifa)

V podstatě se jedná pouze o jednu funkci, která vrací seznam s rozhraními, protože první funkce umožňuje získání dat, které se tímto zavoláním alokují v paměti. Druhá funkce složí k uvolnění (dealokování) těchto dat z paměti.

Funkce *getifaddrs()* vytváří seznam struktur popisujících síťová rozhraní na lokálním prostředí. Vstupním parametrem je ukazatel na strukturu ifaddrs, který má být naplněn daty. Pokud dojde k úspěšnému získání dat, je vrácena hodnota 0. V opačném případě je vrácena hodnota -1, která je značením chyby (příkladem vrácení chyby je nedostatečné oprávnění).

Protože seznam, který je vrácen funkcí, dynamicky alokuje paměť, musí po skončení práce s daty dojít k jeho dealokování. Pro dealokaci dat je nutné zavolat

funkci *freeifaddrs()*.

Pro lepší představu o datech, která jsou následně využita pro další operace, je zde uveden přesný popis struktury ifaddrs zobrazený na obrázku 4.1.

```
struct ifaddrs
{
    struct ifaddrs  *ifa_next;
    char            *ifa_name;
    unsigned int     ifa_flags;
    struct sockaddr *ifa_addr;
    struct sockaddr *ifa_netmask;
    union {
        struct sockaddr *ifu_broadaddr;
        struct sockaddr *ifu_dstaddr;
    } ifa_ifu;
    void            *ifa_data;
};
```

Obrázek 4.1: Definice struktury ifaddrs v jazyce C

- **ifa_next** - Ukazatel na další prvek v seznamu. Pokud další prvek neexistuje, je zde nastavena hodnota NULL.
- **ifa_name** - Odkazuje na název rozhraní.
- **ifa_flags** - Seznam příznaků na daném rozhraní, které jsou vráceny v podobě jednoho integeru. Příkladem takového příznaku může být informace, zda je dané rozhraní aktivní, jestli se jedná o loopback, nebo zda podporuje multicast.
- **ifa_addr** - Ukazuje na informaci o adrese a rodině protokolu, která je podporována.
- **ifa_netmask** - Stejná strukturu jako je v případě **ifa_addr**. Na rozdíl od **ifa_addr** je vrácena informace o broadcast adrese sítě.
- **ifa_data** - Ukazatel na void data, pro případ IPv4 jsou zde data typu Tx (vysílaná data), Rx (přijatá data).

Tato knihovna je využívána v modulech Ifaddrs nebo Netifaces pro zjišťování informací o síťových rozhraních, protože se jedná o systémovou knihovnu operačního systému Linux.

Uvažujme situaci, kdy máme na zařízení definované jedno rozhraní (například s názvem eth0), které podporuje IPv4 a IPv6. V takovém případě tato knihovna vrací 3 záznamy, které zjednodušeně budou vypadat takto:

- $eth0, AF\_INET, ...$
- $eth0, AF\_INET6, ...$
- $eth0, AF\_PACKET, ...$

Na ukázce 4.1.1 je vidět, že pokud chceme získat seznam názvů všech rozhraní, je potřeba pro jedno rozhraní iterovat minimálně 3x. Také je potřeba si všimnout, že nejsou vrácena jenom rozhraní s IPv4 a IPv6, ale v seznamu přibyl záznam s AF_PACKET.

4.1.2 IP Helper

Internet Protokol Helper (IP Helper) API umožňuje vyhledávat a modifikovat síťová nastavení na lokálním počítači pro platformu Windows. IP Helper také zpřístupňuje notifikační mechanizmus umožňující upozornit okolní procesy o změně v konfiguraci. Nevýhodou je, že není zaručena úplná dostupnost na všech windows platformách.

IP Helper je navržen pro využití při C/C++ programování. Programátoři by měli být obeznámeni se správou síťových prvků a TCP/IP koncepty. Tuto systémovou knihovnu je možné využít pro následující účely:

- Vyhledávání informací o síťové konfiguraci
- Správa síťových adaptérů

- Správa rozhraní
- Vyhledávání informací o vysílaných datech

Součástí této knihovny je hlavičkový soubor Iphlpapi.h, který umožňuje získat informace o síťových prvcích. Ten je využíván v modulech Netifaces a Ifaddrs. [9, 10]

4.2 Get_NIC

Modul Get_NIC je veřejně dostupný v Github repozitáři pod MIT licencí. Tato licence umožňuje využívat daný modul pro privátní účely, další distribucí či komerční využití.

V případě využití linuxového příkazu ,,ip link show" byl výsledek skoro okamžitý. Tato skutečnost vedla k myšlence vytvořit první testovací modul, který by vzal výstup ze zmíněného příkazu a vybral by z něj potřebné informace. Po krátkém hledání se zjistilo, že takovýto modul již existuje a nazývá se Get_NIC. Během testování se ukázalo, jak moc tento přístup není optimální. [11, 12]

Modul využívá výstupu z příkazu ,,ip link show", ze kterého následně vybere důležité informace a vrátí je formou objektů. Výhodou modulu je jednoduchá implementace. Nevýhodu je závislost na programu ,,ip link show" a jeho formátování výstupu. V případě, kdy není program dostupný, je vyhozena výjimka s chybou, že nelze modul používat. Nejedná se o optimální řešení, protože se změnou výstupního formát v příkazu ,,ip link show" musí dojít ke změně i v tomto modulu.

Licence	MIT
Operační systém	Linux
Verze pythonu	3+
Velikost API	1
Aktuální verze	0.0.3
Poslední verze	21.7.2019

Tabulka 4.1: Přehled informací k modulu Get_nic

4.3 Ifaddr

IfAddr je malý Python modul, který umožňuje najít všechny IP adresy na počítači. Je podporován pro Linux, MacOs a Windows. Kromě těchto zmíněných platforem by mělo být možné využívat i pro OpenBSD, ale jak je uvedeno přímo v popisu projektu, nebylo využití pro tuto platformu nikdy důkladně testováno. Stejně jako Get_nic, je Ifaddr dostupný pod MIT licencí.

Hlavní myšlenkou tohoto modulu je využití systémové knihovny pro získání informací o rozhraních. O propojení Pythonu se systémovými knihovnami se stará modul ctypes, který umožňuje dynamické linkování knihoven. Díky tomu není potřeba kompilace kódu a je možné se tímto způsobem vyhnout chybám spojeným s kompilací. Toto dynamické linkování ale může znamenat pomalejší práci a delší čekání na výsledek. [13, 14]

Licence	MIT
Operační systém	Windows, Linux, MacOs
Verze pythonu	2.7, 3.5+
Velikost API	1
Aktuální verze	0.1.7
Poslední verze	6.7.2020

Tabulka 4.2: Přehled informací k modulu Ifaddr

4.4 Python-ethtool

Ethtool je standardní linuxový nástroj pro dotazování a konfiguraci síťových rozhraní, především pro ethernetová zařízení. Tento nástroj může být využit pro získávání rozšířených statistik o zařízení a k ovládání rychlosti a toku dat pro ethernetová zařízení. Pro tento nástroj byl vytvořen Python propojující modul umožňující nástroj Ethtool používat na úrovni kódu. Lze pomocí něho například získat informace o síťových prvcích nebo nastavovat přenosové rychlosti. [15]

K získání informací využívá primárně ioctl operace, které umožňují systémová volání na zařízení, u nichž není možné operace popsat regulérním systémovým voláním. [17]

V současné době již neprobíhá aktivní vývoj. Modul je pouze udržován pro systémy, které ho využívají. Místo tohoto modulu je doporučováno využití Netifaces, který je popsaný níže. [16]

Licence	GNU General Public License v2
Operační systém	Linux
Verze Pythonu	2.7, 3.5, 3.6 a 3.7
Velikost API	23
Aktuální verze	0.14
Poslední verze	18. 9. 2018

Tabulka 4.3: Přehled informací k modulu Python-ethtool

4.5 Netifaces

Posledním uvedeným modulem pro zjišťovaní informací o síťových prvcích je Netifaces. Tento modul je, stejně jako předešlý, veřejně dostupný a může být využíván jako náhrada za Ethtool. Implementace modulu využívá systémových knihoven pro získávání informací o rozhraních a tyto informace vrací pomocí PyObject. Modul je možné použít pro Windows, Linux nebo MacOs. [18]

Rozdíl mezi tímto modulem a Ifaddrs je v kompilaci na cílovém počítači. Jak již bylo zmíněno, Ifaddrs nepotřebuje kompilaci zdrojového kódu při instalaci, protože využívá ctypes volající systémové knihovny daného systému. Naopak Netifaces musí být při instalaci zkompilován, protože jeho zdrojový kód je napsán v jazyce C a propojen s Python pomocí C/Python API. Právě fakt, že je napsán v jazyce C, umožňuje větší rychlost tohoto modulu. [19, 14]

Další rozdíl mezi modulem Netifaces a IfAddrs je ve velikosti API. Ifaddrs umožňuje získání všech rozhraní v jedné funkci. Netifaces má funkci pro získání seznamu rozhraní, informace o rozhraní a získání gateways. Na druhou stranu, jak se ukazuje v následujících kapitolách, je přístup Netifaces k systémovým knihovnám neoptimální.

Mezi výhody tohoto modulu - kromě rychlosti - patří i podrobná dokumentace popisující funkce a postupy. V dokumentaci jsou vysvětleny příklady využití pro jednotlivé funkce.

Licence	MIT
Operační systém	Windows, Linux, MacOS
Verze Pythonu	2.7, 3.5+
Velikost API	4
Aktuální verze	0.10.9
Poslední verze	2.1.2019

Tabulka 4.4: Přehled informací k modulu Netifaces

5 Popis implementace nového modulu

Tato kapitola se zabývá implementací modulu pro rychlé zjištění informací o síťových prvcích. Vytvářený modul se jmenuje NicSpeedyModul, přičemž tento název byl vytvořen na základě hlavní vlastnosti modulu. Výsledný modul by měl být rychlý, jednoduchý a zaměřený na platformu Linux, na kterém běží většina Docker kontejnerů.

5.1 Příprava prostředí

Prvním krokem po nastudování dokumentace a rozhraní knihoven byla příprava prostředí, ve kterém by bylo možné si toto pomalé chování vyzkoušet. Z toho důvodu byl vytvořen virtuální stroj s obrazem linuxové distribuce Debian. Pro tento virtuální stroj byly vytvořeny inicializační skripty, které umožňují podle vstupních parametrů vytvořit daný počet fiktivních síťových rozhraní.

Kromě těchto inicializačních skriptů byly vytvořeny i pomocné skripty k odebrání fiktivních rozhraní. Důvodem byla potřeba operativně měnit počty rozhraní pro získání lepšího přehledu o náročnosti získání všech dostupných dat.

5.2 Vytvoření testovacích skriptů

Aby bylo možné objektivně porovnat náročnost jednotlivých Python modulů, bylo potřeba vytvořit testovací skripty, které mají měřit čas potřebný k získání

informací o všech zařízeních. Pro každý vybraný modul byl vytvořen separátní Python program. Vytvořený Python program byl následně spuštěn s časovačem, aby bylo možné zjistit, jak dlouho program běžel.

V těchto testovacích skriptech by se neměly navíc vyskytovat žádné operace, které by mohly ovlivnit testování a jeho výsledek. Moduly nemají jednotné rozhraní pomocí kterého lze přistupovat k těmto informacím. Došlo tedy k vytvoření odlišných skriptů se stejným záměrem, tj. získat co nejvíce informací o všech prvcích, a přitom nedělat operace navíc.

Jako příklad vytvořeného testovacího prvku je zde uveden modul Netifaces. Tento modul nemá funkci, která by dokázala provést proces získání informací v jednom kroku, proto bylo zapotřebí vykonat kroky dva. Prvním krokem bylo získání všech síťových rozhraní, druhým krokem bylo získání informací podle názvu rozhraní na vstupu funkce. V rámci této diplomové práce byla zjištěna chyba, kterou se podařilo opravit a modul tím zároveň vylepšit. Tato oprava byla formou pull requestu poskytnuta do repozitáře Netifaces.

```
import netifaces

addrs = []
interfaces = netifaces.interfaces()
for interface in interfaces:
    addrs.append(netifaces.ifaddresses(interface))

print(addrs)
```

Obrázek 5.1: Ukázka testovacího skriptu pro Netifaces v jazyce Python

Jak je vidět na obrázku 5.1, nedocházelo k žádným jiným operacím. Mimo jiné je zde vidět i možnost výpisu do výstupu program. Výpis výstupu do konzole vždy zpomaluje běh programu, z tohoto důvodu byly všechny výstup směrovány při spuštění do */dev/null*.

Počet rozhraní	**Get_nic**		**Ifaddr**	
	Průměr	Směr. odchylka	Průměr	Směr. odchylka
10	0,0657	0,0084	0,0587	0,0010
20	0,0663	0,0080	0,2570	0,6033
50	0,0647	0,0238	0,0633	0,0057
100	0,0837	0,0062	0,0783	0,0082
200	0,0990	0,0077	0,1123	0,0089
500	0,1297	0,0124	0,2340	0,0138
1000	0,2593	0,0045	0,5303	0,0178
2000	0,7070	0,0061	1,4690	0,0647
5000	3,3920	0,0248	9,5250	0,7691

Tabulka 5.1: Výsledky měření pro Get_nic a Ifaddr. Hodnoty jsou uvedeny v sekundách

Počet rozhraní	**Netifaces**		**Ethtool**	
	Průměr	Směr. odchylka	Průměr	Směr. odchylka
10	0,0593	0,0010	0,0573	0,0027
20	0,0610	0,0047	0,0590	0,0061
50	0,0680	0,0018	0,0670	0,0031
100	0,0987	0,0037	0,0963	0,0010
200	0,2430	0,0018	0,2447	0,0062
500	1,5150	0,0602	1,5713	0,0975
1000	8,2853	0,3169	8,4580	0,0248
2000	44,4293	1,8585	43,9547	1,8619
5000	436,1543	3,4256	435,6790	2,2054

Tabulka 5.2: Výsledky měření pro Netifaces a Ethtool. Hodnoty jsou uvedeny v sekundách

V tabulkách 5.1 a 5.2 jsou vidět výsledky měření rychlosti jednotlivých modulů. Celkem proběhly tři měření, z jejíž výsledků byl vypočítán průměr a

směrodatná odchylka. V rámci tohoto měření nejhorší výsledky vykazovaly moduly Netifaces a Ethtool. Pro 5000 síťových rozhraní je nutné čekat skoro 7 minut. Důvod, proč k takovým vysokým hodnotám dochází, spočívá v nesprávném principu. V modulu Netifaces k problému přispívá i špatné použití knihovny *ifaddr.h*. Problematickému fungování modulu Netifaces a jeho opravě je v této diplomové práci věnována samostatná kapitola 6.

5.3 Analýza pro nový modul

Před začátkem implementace nového modulu je potřeba upřesnit, jakým způsobem bude výsledný modul fungovat. V předchozích kapitolách jsou zmíněny již existující moduly a principy jejich fungování, ale ne všechny moduly jsou optimální z hlediska rychlosti. Cílem je vytvořit takový modul, který bude rychlejší než dosavadní moduly. Nyní je dobré si uvědomit, který z modulů jde pro nás nejlepší cestou.

Mezi prvními můžeme pomyslně vyškrtnout modul Get_nic, a to z důvodu principu jeho fungování. Základní myšlenkou tohoto modulu je zavolat jiný proces, který na výstup vypíše seznam a informace o síťových rozhraních. Get_nic tento výstup zpracovává. Toto není nejvhodnější způsob, protože při delším provozu modulu může dojít k potížím (změna formátování výstupu). Get_nic přidává do celého modulu závislost na příkazu „ip link show". Tento příkaz musí mít striktní výstupní formát. Z tohoto plyne první podnět pro vývoj, tj. modul by neměl být závislý na jiných zdrojích.

Při bližším zkoumání můžeme konstatovat, že modul Ifadrr a Netifaces jsou si v mnoha aspektech podobné. Oba tyto moduly využívají pro získávání informací knihovnu *iffaddr.h*, liší se pouze v místě, na němž ji využívají. Ifadrr využívá stejnojmennou knihovnu *iffaddr.h* na úrovni Python modulu, zatímco Netifaces ji využívá na úrovni jazyka C.

I přesto, že podle tabulky 5.2 je Netifaces o mnoho pomalejší, nejedná se v tomto případě o problém způsobený využíváním knihovny na úrovni jazyka C.

Naopak, Netifaces je v tomto ohledu rychlejší. Netifaces není modul, který by bylo možné použít na všech rozhraních. Důvod je popsán v kapitole 6. Z analýzy tedy plyne, že pokud má být výsledný modul rychlejší, vhodným způsobem je využití jazyka C.

Netifaces využívá *iffaddr.h* pro získání seznamu dostupných rozhraní, zatímco Ethtool k získání seznamu využívá adresář /dev/net a ioctl volání. U tohoto přístupu platí předpoklad, že pomalé načítání je způsobené I/O operacemi.

Na základě prezentované analýzy byla pro vytvoření modulu stanovena následující kritéria:

- neměl by být závislý na jiné knihovně či programu
- měl by být napsán v jazyce C
- měl by využívat knihovnu *iffaddr.h*

5.4 Implementace

Ve chvíli, kdy se prostřednictvím analýzy vyjasnila kritéria, bylo možné přistoupit k samotné implementaci modulu. Tato sekce je rozdělena do několika dalších částí, aby bylo možné oddělit všechny činnosti, které implementace zahrnovala. Vytváření NicSpeedyModule probíhalo způsobem ,,ze zdola nahoru", což znamená, že nejprve byly vytvořeny hlavní funkce pro přístup k síťovým prvkům a jejich datům. Následně byly pro tyto funkce vytvořeny funkce další, které již byly součástí definice C/Python API.

Záměrem bylo oddělit logiku získávání informací od části, kde je definováno API modulu. Takto oddělená logika může být použita i pro jiné účely, např. stejná logika může být použita pro zpřístupnění informací i pro Java prostředí.

5.4.1 Zpřístupnění informací pomocí knihovny iffaddr.h

V části 4.1.1 již byla popsána knihovna *iffaddr.h*, která umožňuje přístup k informacím o síťových prvcích. Problém s více záznamy pro jedno rozhraní nebyl řešen, protože primárním úkolem bylo zpřístupnit co nejrychleji všechny informace.

U každého záznamu, který byl vrácen z funkce *getifaddrs()*, došlo k transformaci do PyObject. V tomto kroku znovu vyvstává problém pramenící z rozdílnosti struktur jazyka C a Python - aby bylo možné pracovat se strukturami C na úrovni Python kódu, je zapotřebí jejich transformace. Data byla tedy transformována do PyList struktury, která je ekvivalentní s datovou strukturou seznamu v jazyce Python.

Na této úrovni kódu není řešen problém s několika záznamy pro stejný název rozhraní. Výsledkem je seznam všech záznamů (tak jak je vrací *getifaddrs*), pouze dochází k transformaci záznamu ze struktury do seznamu. Aby bylo možné filtrovat záznamy, byly vytvořeny další funkce, které umožňují vrátit filtrovaný seznam. Zmíněným filtrem může být použití logiky, která kontroluje, zda záznam vrácený z *getifaddrs* patří do požadované protokolové rodiny (IPv4 nebo IPv6). Případně je možné získat informace pouze o aktivních zařízeních.

5.4.2 Popis API modulu

Nově vytvořený modul obsahuje 7 nových funkcí, které je možné používat pro získání informací o rozhraní. Zde je uveden seznam těchto funkcí s jejich popisem:

- **get_all_system_if()** - vrací seznam všech dostupných rozhraní ve stejném formátu jako vrací getifaddrs.
- **get_all_ipv4_if()** - vrací seznam rozhraní, která patří do rodiny IPv4.
- **get_all_ipv6_if()** - vrací seznam rozhraní, která patří do rodiny IPv6.

- **get_all_by_AF(int af)** - na vstupu funkce obdrží hodnotu AF, podle které se výstupní seznam vyfiltruje.
- **get_active_if()** - vratí pouze seznam s názvem rozhraní, která jsou aktivní.
- **get_info_about_active_devices()** - na vstupu této funkce je jméno rozhraní, pro které mají být vrácené informace bez ohledu na AF.
- **get_ip(char* ifname)** - podle názvu rozhraní vrátí IPv4 adresu.

5.4.3 Vytvoření smoke testů

Smoke testy (v překladu do češtiny se občas používá termín ,,zahořovací testy") se využívají pro rychlé ověření správného nainstalování a chování modulu. Jedná se o test, který se provádí ještě před funkcionálním testováním.

Pro tento modul byl vytvořen soubor, který obsahuje smoke testy pro každou z funkcí. Hlavním úkolem tohoto testu bylo zkontrolovat, jestli všechny funkce definované v modulu API projdou, aniž by došlo k pádu systému. Během implementace došlo k několika chybám způsobujícím pád celého Python prostředí. Toto nečekané ukončení programu bylo způsobeno chybným C kódem přistupujícím do části paměti, která byla chráněná. Důvodem vytvoření smoke testu bylo objevení podobných chyb při instalaci.

Realizace testu byla vyřešena pomocí pytest, který postupně spouští definované testy a kontroluje stav doběhnutí. Pytest byl vytvořen pro každou z funkcí modulu.

5.4.4 Instalace nového modulu

Aby bylo možné nový modul vyzkoušet, je zapotřebí vytvořit *setup.py* soubor, který nový modul přeloží a nainstaluje do repozitáře s Python moduly. Tento *setup.py* soubor importuje modul *setup*. K překladu je potřeba zavolat funkci *setup*, jež má vstupní parametry definovány jako jméno nově vznikajícího modulu a cestu k C souboru, v němž je modul naprogramován.

Vzhledem k tomu, že modul byl cílen na platformu Linux, nebylo potřeba nastavovat direktiva pro překlad ani přidávat žádné další validace či se odkazovat na knihovny třetích stran (soubor *ifaddr.h* paří mezi systémové knihovny Linuxu). Po instalaci modulu byly automaticky spuštěny smoke testy, aby bylo možné rychle odhalit chybu v implementaci.

5.5 Porovnání výsledků

Po úspěšné implementaci nadešla fáze testování rychlosti NicSpeedyModule. Testování dopadlo uspokojivě. Hlavní výhodou nově vzniklého modulu je fakt, že má již připravenou funkci, která dokáže vrátit všechny informace o rozhraních ve stejný čas. Díky tomu byl modul v těchto ohledech mnohem rychlejší než modul Netifaces a Ethtool. Porovnání NicSpeedyModule s již existujícími moduly je uvedeno v tabulkách 5.3 a 5.4.

Počet rozhraní	**NicSpeedyModule**		**Get_nic**		**Ifaddr**	
	Průměr	Směr. odchylka	Průměr	Směr. odchylka	Průměr	Směr. odchylka
10	0,0243	0,0118	0,0657	0,0084	0,0587	0,0010
20	0,0280	0,0031	0,0663	0,0080	0,2570	0,6033
50	0,0283	0,0057	0,0647	0,0238	0,0633	0,0057
100	0,0223	0,0074	0,0837	0,0062	0,0783	0,0082
200	0,0290	0,0124	0,0990	0,0077	0,1123	0,0089
500	0,0313	0,0027	0,1297	0,0124	0,2340	0,0138
1000	0,0423	0,0084	0,2593	0,0045	0,5303	0,0178
2000	0,0677	0,0045	0,7070	0,0061	1,4690	0,0647
5000	0,1603	0,0072	3,3920	0,0248	9,5250	0,7691

Tabulka 5.3: Porovnání rychlosti NicSpeedyModule s Get_Nic a Ifaddr. Naměřené hodnoty jsou uvedeny v sekundách.

Počet rozhraní	**NicSpeedyModule**		**Netifaces**		**Ethtool**	
	Průměr	Směr. odchylka	Průměr	Směr. odchylka	Průměr	Směr. odchylka
10	0,0243	0,0118	0,0593	0,0010	0,0573	0,0027
20	0,0280	0,0031	0,0610	0,0047	0,0590	0,0061
50	0,0283	0,0057	0,0680	0,0018	0,0670	0,0031
100	0,0223	0,0074	0,0987	0,0037	0,0963	0,0010
200	0,0290	0,0124	0,2430	0,0018	0,2447	0,0062
500	0,0313	0,0027	1,5150	0,0602	1,5713	0,0975
1000	0,0423	0,0084	8,2853	0,3169	8,4580	0,0248
2000	0,0677	0,0045	44,4293	1,8585	43,9547	1,8619
5000	0,1603	0,0072	436,1543	3,4256	435,6790	2,2054

Tabulka 5.4: Porovnání rychlosti NicSpeedyModule s Netifaces a Ethtool. Naměřené hodnoty jsou uvedeny v sekundách.

Z tabulky 5.4 je vidět velký rozdíl mezi rychlostí modulu vytvořeného v této diplomové práci a Netifaces. Tato skutečnost vedla k vytvoření opravy pro Netifaces, jejíž postup a jednotlivé fáze jsou popsány v následující kapitole. Hlavní motivací pro vytvoření opravy bylo přispět touto diplomovou prací do open source repozitáře. Netifaces je napsaný v C, má tedy potenciál fungovat stejně rychle jako modul NicSpeedyModule. Kromě Netifaces má problém s časem i modul Ethtool, ale vývoj na tomto modulu je již neaktivní, tudíž nemělo smysl pro něj vytvářet opravu. Výhoda opravy chyby v Netifaces spočívá ve skutečnosti, že tento modul funguje i pro jiné operační systémy než Linux - oprava bude prospěšná i pro Windows systémy.

6 Oprava chyby v repozitáři Netifaces

Moduly Netifaces a NicSpeedyModule jsou si principiálně podobné. Oba jsou napsané v jazyce C a využívají stejnou knihovnu. Proč jsou tedy rychlostně tak výrazně rozdílné? Odpověď na tuto otázku byla nalezena prostřednictvím analýzy a vyústila ve vytvoření opravy. Popis analýzy a procesu opravy je obsahem této závěrečné kapitoly. Výsledná oprava byla poskytnuta formou pull requestu do repozitáře Netifaces.

6.1 Nalezení problému

Před začátkem analýzy je potřeba vytvořit ,,fork" z veřejného repozitáře. Git fork umožňuje vytvořit kopii repozitáře, tak aby nedocházelo k vývoji přímo nad veřejným repozitářem. Všechna logika pro získávání informací o rozhraních je napsána v souboru *netifaces.c*. Protože je modul Netifaces multiplatformní, musí zde být logika pro Windows, Linux a MacOS. Bohužel zmíněná logika je zahrnuta v celém *netifaces.c* souboru, což činí tento soubor dlouhým kolem 3 000 řádků. Ze začátku nebylo jednoduché se v takovém dlouhém kódu zorientovat.

V obdobných případech se vývojář, který objeví chybu nebo nemůže najít řešení pro svůj kód, obrací na cílový repozitář formou ,,Issues". Github má položku s ,,Issues" u každého repozitáře, využívá se pro reportování chyb, k předání zpětné vazby nebo monitorování dalších vývojových kroků. U Netifaces bylo založeno ,,Issue", jehož obsahem bylo právě upozornění na rychlostní rozdíl.

V nalezeném ,,Issue" bylo popsáno neoptimální chování při získávání seznamu s názvy rozhraní. Během získávání seznamu rozhraní je potřeba kontrolovat, zda název již není v seznamu přidán. C/Python nemá datovou strukturu typu množina (set), která by tuto unikátnost zajistila, ale lze kontrolovat, jestli objekt již v seznamu neexistuje. Tímto způsobem bylo řešeno vkládání názvů rozhraní i v tomto případě.

Autor ,,Issue" upozorňoval na možné zpomalení rychlosti v důsledku popsaného procesu. Pokud by existoval počítač čítající 1 000 rozhraní, muselo by dojít vždy ke kontrole, zda tento název již v seznamu není (tzn. musí se projít celý seznam znovu). Netifaces využívá pro čtení na Linuxu funkci getifaddrs z *ifaddr.h*. Ve skutečnosti ale není pro 1 000 rozhraní vrácen seznam s 1 000 záznamy, je vrácen seznam minimálně s 3 000 záznamy (uvažujeme-li, že používáme standardní protokoly rodiny IPv4 a IPv6).

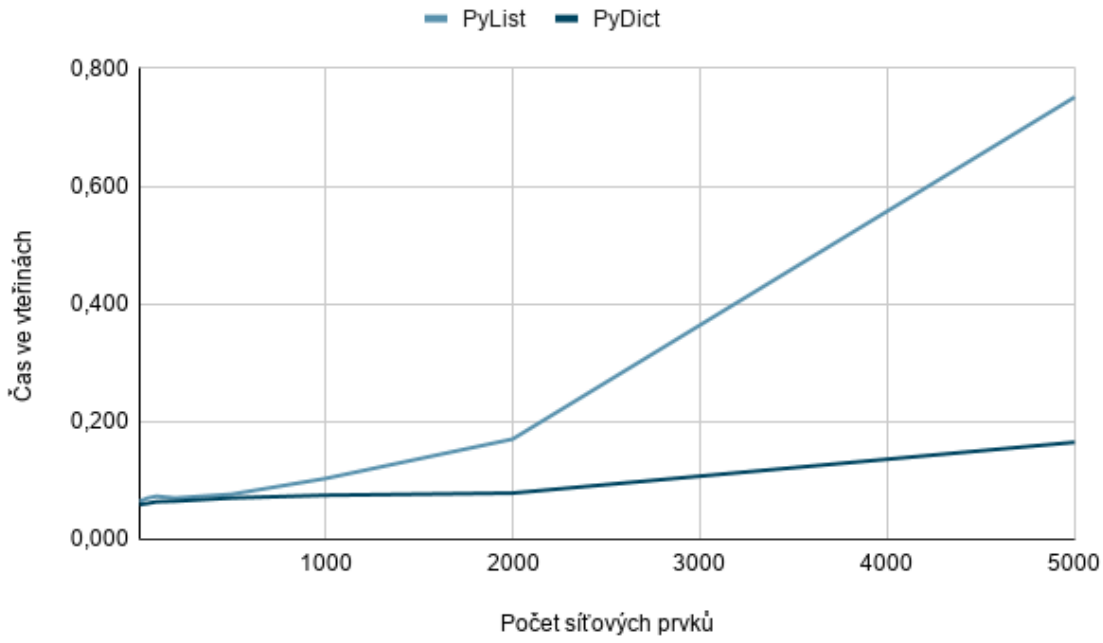
Problém popsaný v ,,Issue" upozorňující na neoptimální chování byl opraven změnou datové struktury. Místo PyList se data začala vkládat do PyDict. Po vložení záznamu do slovníku, kde byl klíčem název rozhraní a hodnota byla ,,None"(null hodnota pro Python), byl vrácen pouze seznam klíčů z PyDict. Po této změně byla patrná změna rychlosti, ta je vidět v tabulce 6.2. Na grafu 6.2 je znázorněna změna, kterou způsobila úprava datové struktury. Problém s rychlostí nadále ale existoval.

Po vyřešení problému při analýze testovacího kódu 5.1 krok po kroku bylo zjištěno, že chyba není v Netifaces. Chyba byla v principu získávání informací o rozhraní.

Uvažujme 1 000 síťových rozhraní podporujících IPv4 a IPv6, z hlediska funkce *getifaddrs*, to znamená 3 000 záznamů. Při dotázání se na seznam názvu rozhraní dojde tedy minimálně k 3 000 iteracím. Tento krok je však ještě v pořádku. Problém nastává v následném dotazování se na jednotlivá rozhraní. Pokud bychom se blíže zaměřili na implementaci funkce *ifaddresses()*, zjistíme jak moc náročné je zjistit informace o jednom rozhraní.

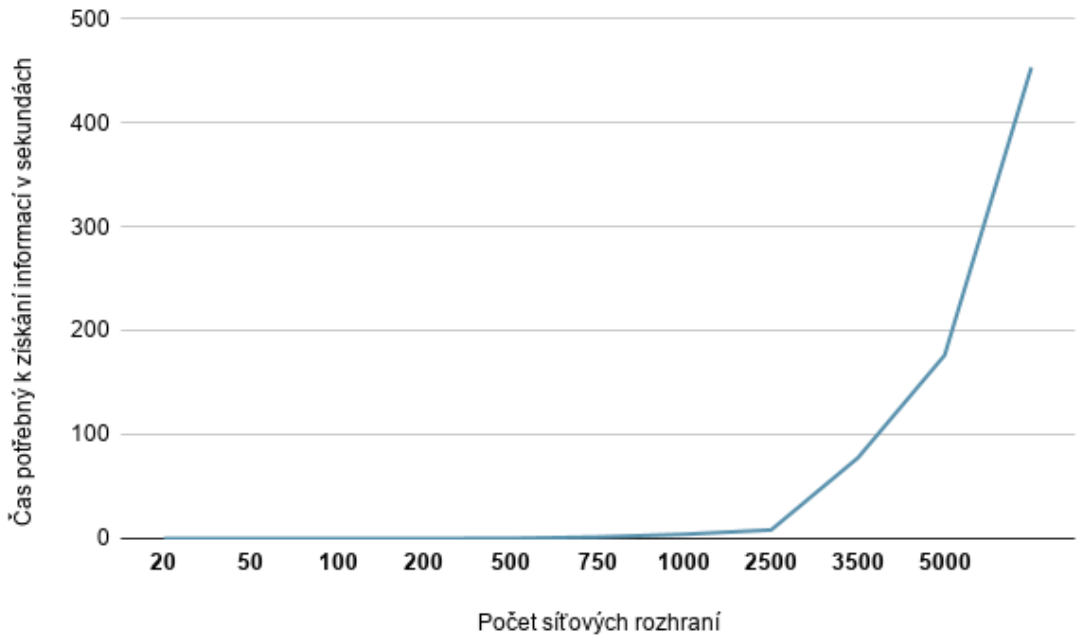
Počet rozhraní	**PyList**		**PyDict**	
	Průměr	Směr. odchylka	Průměr	Směr. odchylka
10	0,068	0,002	0,058	0,004
20	0,066	0,002	0,060	0,007
50	0,070	0,005	0,061	0,008
100	0,073	0,001	0,064	0,002
200	0,070	0,006	0,065	0,006
500	0,077	0,004	0,070	0,001
1000	0,103	0,009	0,075	0,002
2000	0,170	0,007	0,079	0,001
5000	0,751	0,007	0,165	0,005

Tabulka 6.1: Rozdíl v použití struktury PyDict a PyList při ukládání názvů rozhraní. Hodnoty v tabulce jsou uvedeny v sekundách



Obrázek 6.1: Rozdíl po změně datové struktury.

Na vstupu funkce *ifaddresses()* je název rozhraní, pro nalezení rozhraní s tímto názvem je nutné iterovat znovu přes 3 000 záznamů a vkládat je do seznamu. Toto je ovšem pouze pro jeden záznam, abychom našli informaci o všech rozhraních, je nutné udělat 3 000 000 iterací.



Obrázek 6.2: Graf znázorňující růst náročnosti získání informací pomocí modulu Netifaces. Hodnoty jsou uvedeny v sekundách

Jedním z nejlehčích způsobů, jak tento problém vyřešit, je snížit počet iterací. Pro opravu bylo potřeba vytvořit novou funkci, která by iterovala právě jednou přes seznam všech záznamů a vrátila tak informace o všech rozhraních najednou.

6.2 Vytvoření opravy

Jak již bylo zmíněno dříve v textu, soubor *netifaces.c* má přes 3 000 řádků kódu, tento fakt způsoboval špatnou orientaci v kódu. Pro vytvoření nové funkce nebylo možné pouze vytvořit novou funkci, protože by v kódu vznikly duplicity

(např. u transformace struktury do PyObject). Muselo dojít k refaktorování kódu. Části, které se staraly o transformaci struktury do PyObjectu, byly vyjmuty do samostatných funkcí. Tyto funkce bylo možné volat z existující funkce ifaddresses a z nové funkce pro vrácení všech rozhraní. Kromě velikosti celého kódu komplikovala vývoj i multiplatformost modulu. Tento modul musel totiž fungovat na operačních systémech Windows, Linux i MacOS

Po refaktorizaci byla vytvořena funkce *allifaddresses()*. Tato funkce vracela datovou strukturu slovníku, kde klíčem byl název rozhraní a hodnotou byl seznam všech informací, které pro rozhraní byly dostupné. Před samotnou kompilací bylo nutné zkontrolovat, zda dochází k dealokaci struktur. Mimo to bylo nutné zajistit, aby se do modulu pro Linux nepřidávaly definované funkce pro Windows. Nejednalo se o problém, který by bylo potřeba řešit od začátku, protože již před změnou C makra označovala jaká část má být zkompilována pro danou platformu. Bylo ale nutné zkontrolovat, že tato makra jsou i pro nově vytvořené funkce správně.

6.3 Testování

Testování muselo proběhnout pro všechny podporované platformy. Bylo potřeba zajistit prostředí pro testování na Windows, Linux a MacOS. Stejně jako NicSpeedyModul, tak i Netifaces využívá pro rychlé otestování smoke test. Protože byla přidána nová funkce, muselo dojít k upravení tohoto testu tak, aby otestoval i tuto novou funkci. Smoke test byl uložen v souboru *test.py*, do tohoto souboru bylo ke konci přidáno testování nové funkce. Výsledky byly porovnávány s výsledky z funkce ifaddress.

Kromě testování na různých platformách bylo potřeba otestovat i podporované verze Pythonu. Netifaces podporuje Python od verze 2.7. Modul musel být testován na verzích Pythonu od 2.7 do 3.8 (včetně x86 verzí) pro každou platformu zvlášť. Kvůli velkému množství různých prostředí by bylo náročné modul testovat manuálně. Naštěstí tento repozitář využívá nástrojů AppVeyor a Travis. Tyto

nástroje umožňují automatické testy v případě žádosti o pull request.

Dojde-li k žádosti o pull request nebo k novému commitu do repozitáře, jsou tyto testy automaticky spuštěny. Konfigurace pro testy je součástí Netifaces repozitáře. Nástroje Travis a AppVeyor fungují z uživatelského pohledu podobně. Při spuštění jsou vytvořeny požadované testy, ty se následně postupně spouští v požadovaném prostředí. V prvním kroku dojde ke stažení kódu z repozitáře. Po stažení je spuštěna instalace modulu a následně je prováděn smoke test. Pokud proběhne bez chyby, je uznán jako úspěšný. V případě, že dojde k chybě (stačí jediná), celá série testů je označena jako neúspěšná. Toto značí, že pro daný commit testy neprochází.

Nástroj Travis je využíván pro kontrolu na platformách Linux a MacOS. AppVeyour je využíván pro kontrolu na platformě Windows. V této části nastal problém s otestováním Python pro verze 2.7 a 3.4 ve Windows prostředí. Tyto verze jsou již nějakou dobu nepodporované, a proto v případě, kdy dojde ke spuštění testu Python 2.7/3.4, vypisují chybu s knihovnou. Tento problém by šel na lokálním prostředí vyřešit tím, že by se příslušné knihovny doinstalovaly. Skutečnost je bohužel taková, že knihovny nejsou v repozitářích Microsoft dostupné, proto není možné chybu nasimulovat a opravit.

Testy pro Windows s verzemi 2.7 a 3.4 by bylo možné odebrat, ale toto rozhodnutí bylo ponecháno na vlastníkovi repozitáře. Jinak by mohlo dojít k mylnému dojmu, že jsou testy upravovány tak, aby oprava fungovala. Proto byl tento fakt při vytváření žádosti o pull request zmíněn. Testy pro vyšší verze Pythonu prošly bez problémů.

6.4 Dosažené výsledky

Po dokončení testování mohla nastat fáze měření rychlosti. Měření probíhalo stejně jako v případě NicSpeedyModule, bylo tedy zapotřebí nasimulovat zařízení s různými počty síťových rozhraní. Ve chvíli, kdy byla rozhraní připravena, byl spuštěn testovací skript pro změření rychlost. Výsledky tohoto měření jsou uvedeny v tabulce 6.2.

Počet rozhraní	Netifaces		Netifaces s opravou	
	Průměr	Směr. odchylka	Průměr	Směr. odchylka
10	0,059	0,001	0,051	0,007
20	0,061	0,005	0,049	0,005
50	0,068	0,002	0,051	0,005
100	0,099	0,004	0,049	0,002
200	0,243	0,002	0,057	0,005
500	1,515	0,060	0,061	0,007
1000	8,285	0,317	0,072	0,004
2000	44,429	1,858	0,118	0,007
5000	436,154	3,426	0,221	0,006

Tabulka 6.2: Tabulka s měřením rychlosti Netifaces po přidání nové funkce. Uvedené hodnoty jsou v jednotkách sekund.

Hodnoty uvedené v tabulce 6.2 ukazují, jak oprava pomohla ke zrychlení. V nejnáročnějším případě, kdy mělo zařízení 5 000 rozhraní, trvalo průměrně získání informací modulu bez opravy 436,154 sekund (tj. 7 minut a 16 sekund). S opravou - resp. s přidáním nové funkce - se doba průměrně snížila na 221 ms.

6.5 Průběh odevzdávání opravy

Ve fázi, kdy byly všechny testy hotovy, byl výsledek nabídnut formou pull requestu do repozitáře Netifaces. Ještě před úplným vytvořením requestu bylo potřeba změnit formátování kódu, protože se omylem stalo, že ho IDE změnilo. Může se zdát, že toto je pouze maličkost, ale není tomu tak. Každá změna je v pull requestu vidět, i přeformátování. Důvodem je, aby byly pro code review (kontrola kódu) všechny změny přehledné. Bohužel předpoklad čitelnosti nebyl úplně naplněn, protože se jednalo o větší změny, tak se podařilo alespoň usnadnit práci tím, že není potřeba kontrolovat změnu ve formátování.

Pro vytvoření pull requestu bylo zapotřebí sepsat komentář ke změnám a důvody, které vedly k vytvoření změn. Pro pull request byla poskytnuta tabulka s naměřenými rychlostmi před opravou a po opravě. K této tabulce byl přidán i graf znázorňující exponenciální růst v modulu bez opravy.

Kvůli problému s testy pro Python 2.7/3.4 (popsané již v 6.3), neprošly 2 z 19 testů. Bohužel oprava pro tyto testy není momentálně jednoduchá, protože není možné si na lokálním prostředí chyby zreplikovat a vyzkoušet. Vzhledem k tomu, že testy neprošly pouze u nepodporovaných verzí Pythonu, soudím, že by s tím nemusel být problém. Aktuálně pull request stále čeká na schválení vlastníkem repozitáře (stav ke květnu 2021).

7 Závěr

Hlavním cílem této diplomové práce bylo vytvoření modulu pro jazyk Python, který by dokázal rychle zjistit informace o velkém množství síťových prvků. Pro splnění tohoto cíle bylo nutné seznámit se s postupy pro vytváření rozšiřujících modulů pro jazyk Python a také se seznámit s již existujícími moduly.

V teoretické části diplomové práce jsou popsány již existující moduly pro získávání informací o rozhraních s důrazem kladeným na představení hlavní myšlenky, s níž byly implementovány. Výběr modulů probíhal v souladu s cílem poskytnout možnost porovnání jednotlivých rozdílných přístupů. Kromě popisu již existujících modulů práce rovněž ukazuje, jakým způsobem je možné realizovat vlastní rozšiřující Python modul a předkládá i jednotlivé kroky, podle nichž k vytvoření vlastního modulu došlo.

Praktická část této práce je věnována vytváření nového Python modulu, který umožňuje rychlý přístup k informacím o síťových rozhraních. Došlo k vytvoření analýzy a z ní vyplývajících kritérií, která by měl nový modul splňovat. V práci je popsán celý postup implementace modulu i způsob, jehož prostřednictvím byly již existující moduly testovány, tak aby výsledky byly objektivní. Výsledkem popsaného procesu je vznik nového NicSpeedyModulu, který je ve srovnaní s ostatními existujícími moduly významně rychlejší. Modul Netifaces, který je jedním z nejpoužívanějších modulů díky své jednoduchosti a možnosti využití na více platformách, dokáže získat informace o všech síťových rozhraních v čase do 7,5 minuty. Nově vytvořený modul NicSpeedyModule dokáže tuto dobu zmenšit na 160 ms.

Po vytvoření modulu NicSpeedyModule proběhlo několik testovacích měření a výsledky mezi ním a Netifaces byly velmi rozdílné, a to i přesto, že jsou si moduly podobné principem. Nad rámec této práce byla nejdříve vypracována analýza popisující hlavní rozdíly mezi vytvořeným modulem v této práci a Netifaces. Aby problém nezůstal pouze u analýzy, byl do repozitáře Netifaces vytvořen pull request, který opravuje nedostatek toho modulu. Oprava zrychluje získání informací o všech rozhraních ze 7,5 minuty na 160 ms. Oproti NicSpeedyModule je Netifaces multiplatformní, z toho důvodu muselo dojít k opravě pro Linux, Windows a MacOS. Na zmíněných platformách bylo potřeba otestovat různé verze Pythonu, aby bylo možné určit, zda je oprava prakticky použitelná, a zda nezpůsobí obtíže v rámci stávající implementace modulu.

Úspěšně se podařilo vytvořit modul NicSpeedyModule, který je veřejně dostupný na adrese https://github.com/LukasMazl/NicSpeedyModule pod MIT licencí. Nově vytvořený modul umožňuje rychlý přístup k informacím o rozhraních. Díky této práci se podařilo odhalit problém, který způsobuje pomalé načítání informací o rozhraních v modulu Netifaces. Díky opravě, která je vyústěním prezentované práce, není problém s pomalým načítáním informací vyřešen jenom pro linuxovou distribuci, ale i pro Windows a MacOs.

Literatura

[1] What is a Container? — App Containerization — Docker. Empowering App Development for Developers — Docker [online]. Copyright © 2021 Docker Inc. All rights reserved [cit. 12.05.2021]. Dostupné z: https://www.docker.com/resources/what-container#/package_software

[2] Docker overview — Docker Documentation. Docker Documentation — Docker Documentation [online]. Copyright © 2013 [cit. 12.05.2021]. Dostupné z: https://docs.docker.com/get-started/overview/

[3] What is Kubernetes? — Kubernetes. Kubernetes [online]. Copyright © 2021 The Kubernetes Authors [cit. 15.05.2021]. Dostupné z: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[4] Kubernetes Components — Kubernetes. Kubernetes [online]. Copyright © 2021 The Kubernetes Authors [cit. 15.05.2021]. Dostupné z: https://kubernetes.io/docs/concepts/overview/components/

[5] Red Hat OpenShift Overview — Red Hat Developer. Cloud Developer Tutorials and Software from Red Hat — Red Hat Developer [online]. Copyright © 2021 Red Hat Inc. [cit. 15.05.2021]. Dostupné z: https://developers.redhat.com/products/openshift/overview

[6] About OpenShift Kubernetes Engine — About — OpenShift Container Platform 4.6. Home — Official Red Hat OpenShift Documentation [online]. Copyright © 2021 Red Hat, Inc. [cit. 15.05.2021]. Dostupné z: https://docs.openshift.com/container-platform/4.6/welcome/oke_about.html

[7] Extending Python with C or C++ — Python 3.9.5 documentation [online]. Copyright © [cit. 16.05.2021]. Dostupné z: https://docs.python.org/3/extending/extending.html

[8] ctypes — A foreign function library for Python — Python 3.9.5 documentation [online]. Copyright © [cit. 11.05.2021]. Dostupné z: https://docs.python.org/3/library/ctypes.html

[9] GetAdaptersAddresses function (iphlpapi.h) - Win32 apps — Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 11.05.2021]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getadaptersaddresses

[10] IP Helper - Win32 apps — Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 11.05.2021]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/iphlp/ip-helper-start-page

[11] get-nic · PyPI. PyPI · The Python Package Index [online]. Copyright © 2021 [cit. 11.05.2021]. Dostupné z: https://pypi.org/project/get-nic/

[12] GitHub - tech-novic/get-nic-details: Get Network Interface details. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 11.05.2021]. Dostupné z: https://github.com/tech-novic/get-nic-details

[13] ifaddr · PyPI. PyPI · The Python Package Index [online]. Copyright © 2021 [cit. 11.05.2021]. Dostupné z: https://pypi.org/project/ifaddr/

[14] GitHub - pydron/ifaddr: Python Library to enumerate all network interfaces. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 11.05.2021]. Dostupné z: https://github.com/pydron/ifaddr

[15] 11.8. Ethtool Red Hat Enterprise Linux 6 — Red Hat Customer Portal. Red Hat Customer Portal - Access to 24x7 support and knowledge [online]. Copyright © 2021 Red Hat, Inc. [cit. 11.05.2021]. Dostupné z: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s1-ethtool

[16] ethtool · PyPI. PyPI · The Python Package Index [online]. Copyright © 2021 [cit. 11.05.2021]. Dostupné z: https://pypi.org/project/ethtool/

[17] GitHub - fedora-python/python-ethtool: Deprecated Python bindings for the ethtool kernel interface. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 11.05.2021]. Dostupné z: https://github.com/fedora-python/python-ethtool

[18] netifaces · PyPI. PyPI · The Python Package Index [online]. Copyright © 2021 [cit. 11.05.2021]. Dostupné z: https://pypi.org/project/netifaces/

[19] GitHub - al45tair/netifaces. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 11.05.2021]. Dostupné z: https://github.com/al45tair/netifaces

Příloha I - Zdrojové kódy

- Zdrojový kód pro NicSpeedyModule - NicSpeedyModule-1.0.0.zip

Příloha II - Tabulky naměřených hodnot

Počet rozhraní	NicSpeedyModule			Netifaces		
	1.měření	2.měření	3.měření	1.měření	2.měření	3.měření
10	0,030	0,026	0,017	0,059	0,060	0,059
20	0,030	0,027	0,027	0,060	0,059	0,064
50	0,026	0,032	0,027	0,068	0,069	0,067
100	0,027	0,019	0,021	0,098	0,101	0,097
200	0,021	0,032	0,034	0,242	0,244	0,243
500	0,031	0,030	0,033	1,499	1,492	1,554
1000	0,037	0,046	0,044	8,394	8,383	8,079
2000	0,070	0,068	0,065	44,483	45,450	43,355
5000	0,161	0,156	0,164	438,386	435,030	435,047

Počet rozhraní	Ethtool			Get_nic		
	1.měření	2.měření	3.měření	1.měření	2.měření	3.měření
10	0,057	0,059	0,056	0,064	0,062	0,071
20	0,057	0,057	0,063	0,066	0,071	0,062
50	0,069	0,066	0,066	0,055	0,059	0,080
100	0,096	0,097	0,096	0,087	0,080	0,084
200	0,245	0,248	0,241	0,102	0,101	0,094
500	1,613	1,509	1,592	0,129	0,123	0,137
1000	8,442	8,464	8,468	0,257	0,262	0,259
2000	45,158	43,485	43,221	0,709	0,703	0,709
5000	437,031	435,424	434,582	3,408	3,382	3,386

Počet rozhraní	**ifaddr**		
	1.měření	2.měření	3.měření
10	0,059	0,059	0,058
20	0,650	0,059	0,062
50	0,067	0,062	0,061
100	0,081	0,073	0,081
200	0,107	0,117	0,113
500	0,230	0,229	0,243
1000	0,534	0,519	0,538
2000	1,498	1,481	1,428
5000	9,186	10,014	9,375

Příloha III - Naměřené hodnoty po opravě

Počet rozhraní	PyList			PyDict		
	1. měření	2. měření	3. měření	1. měření	2. měření	3. měření
10	0,069	0,067	0,068	0,057	0,061	0,057
20	0,067	0,065	0,065	0,057	0,058	0,064
50	0,069	0,068	0,073	0,063	0,056	0,064
100	0,073	0,074	0,073	0,063	0,065	0,064
200	0,070	0,067	0,074	0,069	0,063	0,063
500	0,078	0,074	0,078	0,070	0,071	0,070
1000	0,109	0,102	0,099	0,076	0,074	0,075
2000	0,166	0,173	0,172	0,078	0,079	0,079
5000	0,749	0,749	0,756	0,162	0,168	0,165

Počet rozhraní	Netifaces			Netifaces s opravou		
	1. měření	2. měření	3. měření	1. měření	2. měření	3. měření
10	0,059	0,060	0,059	0,053	0,054	0,047
20	0,060	0,059	0,064	0,052	0,047	0,048
50	0,068	0,069	0,067	0,054	0,052	0,048
100	0,098	0,101	0,097	0,050	0,048	0,048
200	0,242	0,244	0,243	0,060	0,054	0,057
500	1,499	1,492	1,554	0,065	0,059	0,058
1000	8,394	8,383	8,079	0,075	0,070	0,072
2000	44,483	45,450	43,355	0,114	0,120	0,121
5000	438,386	435,030	435,047	0,225	0,218	0,220