České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra telekomunikační techniky

Diplomová práce

# Pokročilý systém správy datových sítí a snadné řešení provozních problémů

**Bc. Miroslav Hudec**

**Studijní program**: Elektronika a komunikace
**Studijní obor**: Komunikační systémy a sítě
**Vedoucí práce**: doc. Ing. Leoš Boháč, Ph.D.

Praha, Květen 2018

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25.5.2018

.....

Bc. Miroslav Hudec

## Poděkování

Chtěl bych poděkovat svému vedoucímu, doc. Ing. Leoši Boháčovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnoval.

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hudec** Jméno: **Miroslav** Osobní číslo: **425033**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

Studijní obor: **Komunikační systémy a sítě**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Pokročilý systém správy datových sítí a snadné řešení provozních problémů**

Název diplomové práce anglicky:

**Advanced Management and Troubleshooting of Data Networks**

Pokyny pro vypracování:

Cílem je vytvořit softwarový systém pro usnadnění hromadné správy, řešení problémů a dokumentace datových sítí, především na zařízeních Cisco. Hlavním cílem je vhodným způsobem rozšířit možnosti volně dostupného SDN řadiče Cisco APIC-EM, potažmo Cisco DNA Center. Dalším cílem bude vytvořit SW rozhraní, které je schopno získávat informace o sítí jak prostřednictvím SDN řadiče, tak i pomocí starších metod, čímž se umožní i podporu starších zařízení, které jsou stále v sítích používány. Získané údaje budou poté jednotně integrované ve vytvořené základové SW platformě, které umožní vytvářet i webové aplikace dle konkrétních požadavků.

Seznam doporučené literatury:

[1] Avramov, L.: The Policy Driven Data Center with ACI: Architecture, Concepts, and Methodology. Indianapolis, IN: Cisco Press, 2015. ISBN 978-1-58714-490-5.
[2] Nadeau, T.D.; Gray, K.: SDN: Software Defined Networks. Beijing: O'Reilly, 2013. ISBN 978-1-44934-230-2.
[3] Marschke, D.; Doyle, J.; Moyer, P.: Software Defined Networking (SDN): Anatomy of OpenFlow. Raleigh, NC: Lulu, 2015. ISBN 978-1-48342-723-2.
[4] Donovan, J.; Prabhu, K.:. Building the Network of the Future: Getting Smarter, Faster, and More Flexible with a Software Centric Approach. Boca Raton: CRC Press, Taylor & Francis, Group, 2017. ISBN 978-1-13863-152-6.

Jméno a pracoviště vedoucí(ho) diplomové práce:

**doc. Ing. Leoš Boháč, Ph.D., katedra telekomunikační techniky FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **04.01.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

| doc. Ing. Leoš Boháč, Ph.D.<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Ing. Pavel Ripka, CSc.<br>podpis děkana(ky) |
|---|---|---|

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

# Anotace

V souvislosti s nárůstem počtu aktivních prvků v dnešních počítačových sítích je stále obtížnější spravovat tyto technologie tradičními postupy. Vzniká proto potřeba po prostředcích umožňujících automatizaci některých úkonů správy datových sítí s využitím softwarových nástrojů. Tato práce se zabývá analýzou aktuálního stavu v dané problematice a realizací takového softwarového nástroje, který značně usnadňuje hromadnou správu síťových zařízení a řešení běžných provozních problémů. Vzniklý nástroj NUAAL jsem primárně koncipoval k rozšíření možností volně dostupného SDN řadiče Cisco APIC-EM, avšak je zároveň využitelný zcela samostatně pro získávání strukturovaných dat přímo ze síťových prvků. Nástroj je napsán v programovacím jazyce Python a je primárně zaměřen na podporu aktivních prvků společnosti Cisco Systems, Inc., avšak umožňuje přidání podpory pro zařízení dalších výrobců. Hlavním cílem tohoto nástroje je poskytnutí jednotného programového rozhraní pro získávání dat z různých zařízení či síťových kontrolerů v jednotném formátu.

## Klíčová slova

Softwarově definované sítě (SDN), automatizace datových sítí, síťové kontrolery, Cisco APIC-EM, Python

# Summary

As a consequence of the continuous increase in the number of active devices in today's computer networks, it is becoming increasingly difficult to manage these networks by traditional means. There is a rising need for tools which will allow automation of network management tasks by using programmatic approach. This paper aims to analyze the current situation in the field of networking and provides a software tool to ease management of computer networks and help to solve some of the typical operational problems. The created tool NUAAL was primarily designed to enhance functions of freely available SDN controller Cisco APIC-EM, however, it can be used completely on its own in order to retrieve structured data directly from network devices. The tool is written in Python programming language and mainly supports Cisco Systems' devices while allowing to add different vendors support in the future. The main purpose of this tool is to provide a unified program interface for retrieving data from different devices in one common format.

## Index terms

Software defined networks (SDN), network automation, network controllers, Cisco APIC-EM, Python

# Obsah

# Kapitola 1

# Úvod

V oblasti datových komunikací je v posledních letech zjevně patrný trend nárůstu počtu zařízení, která se připojují do celosvětové sítě Internet. Tento růst můžeme přisuzovat rozvoji na poli chytrých zařízení, jako jsou telefony, tablety a osobní počítače, které jsou stále dostupnější pro větší počet uživatelů. Druhým důležitým faktorem je také trend Internet of Things (IoT), v rámci něhož se do Internetu připojují takřka libovolná elektronická zařízení. Kromě tradiční spotřební elektroniky do tohoto sektoru spadají také specifická miniaturizovaná zařízení, která jsou schopna díky mnoha senzorům zprostředkovat informace o svém okolí. Dle současného trendu tak lze předpokládat, že během roku 2020 bude k Internetu připojeno více než 20 miliard zařízení. [1] Rostoucí trend v oblasti datových sítí můžeme sledovat také v korporátním sektoru. Výpočetní technika se stala nezbytnou součástí pro provozování ať už velkých, tak i menších podniků. V důsledku provozování stále většího počtu nejrůznějších služeb a aplikací jsou kladeny vyšší nároky na příslušná IT oddělení, která mají za úkol zajistit co možná nejspolehlivější provoz těchto služeb. Je zřejmé, že naprosto fundamentálním předpokladem spolehlivého fungování těchto klíčových komponent podnikání je právě hladce fungující datová síť.

## 1.1 Správa datových sítí

Pod pojmem *správa datové sítě* rozumíme proces řízení a kontroly všech prvků infrastruktury a služeb provozovaných v této síti. Slovo *proces* je v této definici velmi důležité. Správa sítě představuje kontinuální činnost zahrnující konfiguraci, monitoring, testování a dokumentování všech komponent podílejících se na přenosu dat. Korektně definovaná a aplikovaná správa je naprosto fundamentálním předpokladem pro korektní a deterministické chování dané sítě. Existuje řada doporučení, jakým způsobem spravovat síť v souladu s principy procesního řízení. Jedním z nejrozšířenějších souborů těchto prověřených konceptů a postupů je Information Technology Infrastructure Library (ITIL), který popisuje jak lépe plánovat, využívat a zkvalitňovat využití informačních technologií. Z této řady doporučení vyplývá mimo jiné skutečnost, že základním předpokladem pro kvalitní provozování datové sítě je udržování aktuální dokumentace, která

popisuje celkový stav dané sítě. Absence takovéto dokumentace přináší řadu problémů jak pro řešení každodenních úkonů spojených s provozováním počítačové sítě, tak pro její budoucí rozšíření či obnovu. Oproti tomu kvalitně zpracovaná a aktuální dokumentace umožňuje rychleji a lépe vyhodnotit možné dopady dílčích konfiguračních změn, a předejít tak nepředvídatelnému chování celé sítě či výpadku některých služeb. Dokumentace síťové infrastruktury by samozřejmě měla být jednotná a centrálně udržovaná, aby se předešlo existenci několika paralelních verzí téže dokumentace, z nich však žádná není zcela úplná a aktuální. V některých případech může sítovou dokumentaci tvořit řada nejrůznějších tabulek, výkresů a textových souborů, které jsou nějakým způsobem průběžně aktualizovány a sdíleny mezi jednotlivými síťovými administrátory. Pro potřeby rozsáhlejších a komplexnějších sítí se však tato metoda stává značně neefektivní a náchylná na lidské chyby. Pro potřeby takovýchto sítí je proto většinou nasazen některý z řady dostupných softwarů, který se snaží tento úkol administrátorům usnadnit. Software pro síťovou správu většinou poskytuje tyto základní funkce:

- **Přehled síťové topologie** - Grafické zobrazení všech síťových prvků a jejich vzájemné fyzické propojení
- **Správa konfigurací** - Periodická archivace konfiguračních souborů jednotlivých zařízení pro možnost případné obnovy konfigurace, případně monitorování změn
- **Monitoring provozních údajů** - Vyčítání provozních dat z jednotlivých zařízení, nejčastěji prostřednictvím protokolu Simple Network Management Protocol (SNMP)
- **Monitoring chybových hlášení** - Příjem a případná analýza chybových zpráv prostřednictvím protokolu Syslog

Některé z výše uvedených funkcí, které podléhají určitým standardům mohou být delegovány na software dedikovaný pro tento účel. Typicky se jedná zejména o dohled prostřednictvím SNMP či zpracování zpráv z protokolu Syslog. Určitý problém však nastává při zpracování konfiguračních dat zařízení. Konfigurační soubory a příkazy jsou specifické pro produkty daného výrobce, či dokonce jen konkrétní platformy síťových prvků. Ačkoliv převažuje snaha udržovat sítě takzvaně "jednobarevné"(tedy používat prvky téhož výrobce), často se můžeme v rámci jedné sítě setkat se zařízeními mnoha výrobců. V takovém případě může nastat potřeba nasadit více softwarů specifických pro dané výrobce, či spravovat centrálně pouze prvky jednoho výrobce a zbylá zařízení ošetřovat ručně. V obou případech se však koncept jednotného řízení rozpadá do menších celků. Existují samozřejmě i softwary, které umožňují do určité míry spravovat zařízení různých značek, avšak jejich funkce jsou v porovnání s *vendor-specific* softwary omezené.

Je třeba uvést, že výše zmíněná řešení pro síťovou správu nepředstavují cestu ke skutečné automatizaci počítačové sítě. Ačkoliv jsou možnosti některých řešení velmi široké, nejsou zaměřeny na samotnou konfiguraci síťových prvků či poskytnou informace o konfiguraci zařízení ve strojově zpracovatelném formátu. Za tímto účelem existují síťové řadiče (angl. *network cont-*

*rollers*). Představitelem softwaru pro správu sítě může být například *Cisco Prime Infrastructure* zaměřený na síťové prvky společnosti *Cisco Systems*.

### 1.1.1 Úvod do problematiky automatizace datových sítí

Zmiňovaný nárůst počtu síťových prvků vede k tomu, že tyto datové sítě je obtížné spravovat *tradičním* způsobem, tedy prostřednictvím interakce člověka s příslušným zařízením. V oblasti výpočetní infrastruktury, tedy serverů a datových center, je dnes pojem *automatizace* poměrně dobře zaveden. Právě díky ní je možné provozovat rozsáhlá datová centra s tisícovkami fyzických serverů, které provozují další desítky a stovky virtuálních strojů. V takovémto objemu již není možné, aby jednotlivé úkony spojené s provozem těchto výpočetních farem zajišťoval člověk. Díky nástrojům jako *OpenStack*, *Docker* či *Ansible* je spuštění a konfigurace nového virtuálního stroje otázkou několika minut a pár kliknutí v grafickém rozhraní. Ačkoliv se o pojmu automatizace, či softwarového řízení, v souvislosti s datovými sítěmi hovoří již řadu let, rychlost jejího reálného nasazení je poměrně pomalá. Dalo by se říci, že jediným odvětvím kde můžeme skutečně mluvit o softwarovém řízení datové infrastruktury, jsou právě zmiňovaná datová centra.

Ohromný počet výpočetních prostředků samozřejmě vyžaduje adekvátní počet síťových prvků. Absence možnosti konfigurovat a spravovat tyto prvky obdobnou cestou, jako je tomu u serverů, by samozřejmě vedla ke značnému zpomalení celého procesu. Velkou výhodou prostředí datových center oproti podnikovým sítím je také to, že lze definovat jasná, konkrétní pravidla a politiky, které lze striktně vynucovat jednoduše proto, že stroje se budou chovat tak, jak budou nastaveny. Navíc i přes svou objemnou povahu jsou datová centra poměrně homogenními prostředími, což úkol jejich automatického řízení také značně zjednodušuje. V oblasti podnikových sítí je však situace zcela odlišná. Jednou ze základních odlišností je již samotná povaha datového provozu. Zatímco v datových centrech je naprostá většina datových toků typu *'east-west'*, tedy mezi jednotlivými stroji v rámci téhož datacentra, v podnikových sítích zastupuje většinu provoz typu *'north-south'*, tedy komunikace typu klient - server, potažmo komunikace mimo podnikovou síť, tedy do Internetu či datacentra.

Rozdílnost povahy provozu v těchto prostředích s sebou přináší také síťové prvky optimalizované pro daný typ nasazení. Vezmeme-li například produktové portfolio společnosti *Cisco Systems*, nalezneme přepínače pro prostředí datových center v řadě označené *Nexus*, zatímco přepínače pro podnikové prostředí v řadě *Catalyst*. Obdobně u společnosti *Brocade* nalezneme datacentrové přepínače v rodině *VDX* a jejich podnikové souputníky v rodině *ICX*. Ačkoliv se stále jedná o přepínače, jejichž základní princip je de-facto stejný, datacentrové verze mají typicky vyšší propustnost a menší latenci, než verze podnikové. Dalším rozlišovacím parametrem je také paleta funkcí jednotlivých platforem. Kupříkladu jedním z primárních požadavků na podnikové sítě je kontrola přístupu do sítě spojená s ověřováním uživatelů a zařízení. Za tímto účelem bývá v sítích stále více nasazen protokol IEEE 802.1X, který musí síťové prvky podporovat. Na druhou stranu, v prostředí datového centra je poměrně nepravděpodobné, že by potenciální

útočník získal fyzický přístup do sítě. Bezpečnostní technici datacenter se tak mnohem více zajímají o možnosti identifikace potenciálně škodlivého provozu. Liší se dokonce i samotná architektura. Datacentrové sítě jsou zaměřeny zejména na vysokou propustnost a hustotu portů, v důsledku čehož upouštějí od klasického hierarchického designu a přechází se k modelu *leaf-spine*, který umožňuje lepší horizontální škálování. Oproti tomu podnikové sítě jsou více "rozložené"a dodržují klasickou hierarchii, tedy rozdělení prvků do centrální, distribuční a přístupové vrstvy.

Již bylo zmíněno, že je to právě odvětví datacentrových sítí, kde lze reálně uplatňovat automatizovanou správu jednotlivých prvků. Je to díky tomu, že zařízení určená pro tento typ nasazení disponují rozhraními pro programový přístup ke konfiguračním parametrům. Tato rozhraní, označovaná jako Application Programming Interface (API) poskytují metody pro vyčítání a ukládání konfiguračních příkazů ve strojovém formátu. Těmto rozhraním a jejich využití se budu podrobněji věnovat v kapitole 2.2 *Rozhraní pro komunikaci*. Prozatím řekněme, že tato rozhraní jsou na rozdíl od běžných konfiguračních rozhraní (jako například Command Line Interface *(Příkazová řádka)* (CLI)) navržena tak, aby umožňovala snadné zpracování dat prostřednictvím programů, aplikací a skriptů. Problém podnikového odvětví síťových prvků je právě ten, že zařízením často chybí tato rozhraní potřebná k programovému přístupu, a pokud jsou dostupná, je tomu tak pouze na novějších síťových prvcích a možnosti těchto rozhraní jsou poměrně omezená. V některých případech lze programový přístup k zařízením zprostředkovat pomocí síťového řadiče, anglicky *controlleru*.

Kontroler je fyzický či virtuální server, který komunikuje se zařízeními prostřednictvím některého z dostupných rozhraní a získaná data dále zpracovává či poskytuje k dalšímu zpracování. Rozhraní, kterými kontroler komunikuje s jednotlivými zařízeními se označují jako *southbound* rozhraní, mezi něž se nejčastěji řadí protokoly jako Secure Shell (SSH), SNMP či *NETCONF*. Pro přístup ke zprostředkovaným datům, tedy ke komunikaci se samotným kontrolerem, se využívá rozhraní označované jako *northbound*. To je nejčastěji realizováno prostřednictvím Representational state transfer (REST) API založeném na protokolu Hypertext Transfer Protocol (HTTP). Podrobněji se problematice síťových kontrolerů věnuje kapitola 2.3 *Síťové řadiče*. Motivací pro nasazení některého ze síťových řadičů však nebývá primárně možnost programového přístupu k zařízením, ale potřeba centralizované správy celé sítě.

## 1.2 Zaměření a cíle práce

Tato práce je zaměřena na návrh a realizaci softwarového nástroje pro usnadnění řešení nejběžnějších provozních problémů datových sítí s využitím konceptů automatizace a programového přístupu k síťovým prvkům. Jedním z hlavních cílů je rozšířit možnosti strojového zpracování konfiguračních a provozních dat takovým způsobem, aby bylo možné na základě těchto dat generovat dokumentaci popisující stav dané sítě a usnadnit nalezení možných chyb v konfiguraci. Výsledný nástroj by měl umožňovat získávat tato data jak prostřednictvím některých

síťových řadičů (zejména Cisco Application Policy Infrastructure Controller - Enterprise Module (APIC-EM)), tak prostřednictvím přímého přístupu k síťovým prvkům. Získaná data by mělo být rovněž možné využít k tvorbě webových aplikací.

# Kapitola 2

# Analýza

Úvodní část této kapitoly předkládá některé z nejběžnějších problémů spojených s provozováním a správou datových sítí. Další část se věnuje popisu možných postupů pro získávání relevantních dat ze síťových prvků ať už prostřednictvím některého ze síťových řadičů, tak přímou komunikací se síťovým zařízením. Zároveň se tato kapitola věnuje popisu některých síťových řadičů společnosti Cisco, jejich možnostem a případným omezením.

## 2.1 Časté problémy při správě sítí

Již v úvodní kapitole bylo zmíněno, že správa a provoz datové sítě je neustávajícím procem zahrnujícím činnosti od řešení běžných problémů po kontinuální budování a obnovu dané sítě. Většina těchto úkonů zahrnuje kontrolu a dokumentování konfigurace jednotlivých prvků, ať už pro potřeby *troubleshootingu*, tak pro návrh spuštění nových síťových služeb či výměnu některých zařízení za nová. Klíčovou komponentou tohoto procesu je dokumentace sítě, popisující její aktuální stav. Zpracování kvalitní dokumentace není lehkým úkolem, avšak její průběžná aktualizace bývá často větším problémem. Postupem času dochází ke změnám v konfiguracích síťových zařízení, které však v mnoha případech nejsou příslušně zdokumentovány. Důvodů pro tento jev je jistě mnoho, avšak k nejčastějším bychom mohli zařadit například to, že příslušný síťový inženýr, který konfigurační změnu prováděl, nepřikládal této změně dostatečnou důležitost a špatně posoudil její možný budoucí dopad. Obecně vzato, nejčastějším problémem sužujícím popis je ať už nechuť jednotlivých administrátorů zabývat se její aktualizací, či jednoduše lidská chyba. Pokud navíc není dokumentace vhodně vedena, může se postupem času stát nepřehlednou kombinací tabulek, výkresů a útržků konfigurace, ve které lze jen velmi těžko najít relevantní informace. Všechny tyto aspekty vedou ve výsledku k jednomu jevu: Pokud je třeba provést konfigurační změnu či získat informace o příslušné konfiguraci, je často jednodušší, a hlavně spolehlivější, podívat se přímo na konfiguraci daného zařízení. Jednoduché přihlášení do příkazové řádky a spuštění několika příkazů poskytne technikovi potřebné odpovědi a jistotu,

že získané informace jsou aktuální a správné. Ačkoliv je tento postup ve své podstatě naprosto legitimní a zdaleka nejrozšířenější, naráží na problémy škálovatelnosti. Pokud je potřeba spustit onu sadu dotazovacích příkazů ne na několika málo zařízeních, ale na několika desítkách zařízení, celý úkon se stává nesrovnatelně časově náročnějším. Celý úkon, který představuje připojení do zařízení, zadání příslušného příkazu a zapsání požadované informace, nevyžaduje žádnou zvláštní kvalifikaci na straně technika, avšak může zabrat i vysoce kvalifikovanému člověku čas i v řádu desítek hodin. Velmi podstatným problémem u takto vysoce repetitivních úkonů je také možnost lidské chyby. Pokud člověk už podvacáté spouští tentýž příkaz, může se snadno stát, že něco přehlédne. Nemusí se přitom vždy jednat jen o správce dané sítě, ale například o technika dodavatelské společnosti, který byl pověřen kontrolou stavu zmiňované sítě například v souvislosti s návrhem jejího obnovení a výměny některých zastaralých zařízení. Tento technik by tak ve výsledku strávil nepřiměřené množství času nad tímto v podstatě triviálním úkonem. Oproti lidem jsou počítače ideální pro vykonávání opakujících se činností. Softwarový nástroj, který je schopen komunikovat se síťovým prvkem a porozumět daným výstupům dokáže takovouto operaci vykonat nikoliv v řádu hodin, ale desítek sekund. Následně by poskytl požadované a relevantní informace v přehledné podobě technikovi, který by na jejich základě mohl pokračovat.

Získávání informací ze síťových prvků je asi nejčastějším a zároveň nejzdlouhavějším úkonem v životě každého síťového administrátora. Není proto divu, že se síťoví správci snaží tuto činnost usnadnit. Jednou z možností, jak takového usnadnění docílit, je vytvoření nejrůznějších skriptů, které jsou schopny získat požadované informace. Tento způsob však přináší řadu úskalí, kterým se bude věnovat následující část 2.2. Druhou, mnohem mocnější možností je nasazení síťového řadiče.

## 2.2 Rozhraní pro komunikaci

Tato část se věnuje analýze dostupných rozhraní, prostřednictvím kterých lze komunikovat se síťovými prvky, síťovými řadiči nebo i webovými aplikacemi. Každé z těchto rozhraní má svá specifika a je vhodné pro odlišné typy úkonů. Diskutovaná rozhraní budou tato:

- Command Line Interface *(Příkazová řádka)* (CLI)
- Simple Network Management Protocol (SNMP)
- NETCONF
- REST API

### 2.2.1 Příkazová řádka

Příkazová řádka (CLI) představuje nejrozšířenější a zároveň nejstarší rozhraní pro komunikaci nejen se síťovými prvky. Vzdálený přístup k příkazové řádce bývá realizován prostřednictvím *virtuálního terminálu*, ke kterému se přistupuje buď protokolem *Telnet* či jeho šifrovanou al-

ternativou SSH. Jedná se o textové rozhraní, jehož hlavní výhodou je jeho rozšíření a univerzálnost. Takřka každý síťový prvek disponuje tímto rozhraním, což z něj činní první volbu pro přístup ke konfiguraci zařízení. Oproti alternativám, jako je například webové grafické rozhraní, je příkazová řádka zároveň nejefektivnějším způsobem komunikace mezi síťovým administrátorem a konkrétním síťovým prvkem. Efektivita tohoto rozhraní je dána tím, že poskytuje výstupy v takové podobě, aby byly pro člověka co možná nejpřehlednější a snadno čitelné. To je zároveň hlavní nevýhodou tohoto rozhraní pro použití coby komunikačního kanálu mezi softwarovým nástrojem a zařízením. Na ukázkách 2.1 a 2.2 je patrně vidět rozdíl mezi textovým výstupem určeným pro člověka a reprezentací téhož výstupu ve formátu JavaScript Object Notation (JSON) pro programové zpracování.

```
VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Fa0, Fa1, Fa2, Fa3
10   MGMT                              active
100  VLAN0100                          active
1002 fddi-default                      act/unsup
1003 token-ring-default                act/unsup
1004 fddinet-default                   act/unsup
1005 trnet-default                     act/unsup
```

Ukázka 2.1: Textový výstup příkazu `show vlan brief`

```
[
{'id': 1, 'name': 'default', 'status': 'active', 'access_ports': ['Fa0', 'Fa1', 'Fa2', 'Fa3']},
{'id': 10, 'name': 'MGMT', 'status': 'active', 'access_ports': []},
{'id': 100, 'name': 'VLAN0100', 'status': 'active', 'access_ports': []},
{'id': 1002, 'name': 'fddi-default', 'status': 'act/unsup', 'access_ports': []},
{'id': 1003, 'name': 'token-ring-default', 'status': 'act/unsup', 'access_ports': []},
{'id': 1004, 'name': 'fddinet-default', 'status': 'act/unsup', 'access_ports': []},
{'id': 1005, 'name': 'trnet-default', 'status': 'act/unsup', 'access_ports': []}
]
```

Ukázka 2.2: Reprezentace příkazu `show vlan brief` ve formátu JSON

Kromě nevhodného výstupního formátu pro programové zpracování má však příkazová řádka další omezení, zejména pro konfiguraci. Asi nejvýznamnějším z nich je absence správy transakcí. Konfigurace síťového prvku může být velmi komplexním úkolem, zahrnujícím řadu dílčích úkonů nezbytných ke korektní funkčnosti. Pokud by došlo k chybě při provádění některého z dílčích příkazů, konfigurace zařízení by se ocitla v jakémsi nedefinovatelném stavu mezi nenakonfigurovaným a nakonfigurovaným. Pro tyto scénáře by bylo třeba naprogramovat složité postupy pro vrácení již nakonfigurovaných změn. Dalším problémem je neexistující, nebo velmi limitované předávání chybových hlášení. Pokud dojde k chybě při zadávání konfiguračního příkazu, je často vrácena jen velmi obecná chybová hláška. Pro interaktivní komunikaci s člověkem je to sice vhodná volba, avšak zpracování těchto hlášení na straně programu může být opět problematické. Nejčastěji zmiňovaným problémem CLI rozhraní je však to, že není zaručena podoba jednot-

livých výstupů. Teoreticky se tak může stát, že například aktualizací softwaru zařízení dojde k nepatrným změnám v textovém výstupu, což pro lidského operátora není žádným problémem, ale způsobí nefunkčnost skriptu spoléhajícího na konkrétní strukturu výstupu. V praxi však příliš často ke změnám v textových výstupech nedochází, nicméně je třeba s touto možností počítat.

### 2.2.2 Simple Network Management Protocol

Protokol SNMP byl navržen pro získávání především provozních dat z nejrůznějších typů síťových zařízení. Z naprosté většiny dnešních zařízení lze prostřednictvím tohoto protokolu získávat informace například o vytížení procesoru, paměti, teplotě zařízení, množství přenášených dat na jednotlivých rozhraních a mnoho dalších. Jednotlivé hodnoty jsou reprezentovány proměnnými, z nichž každá má svůj unikátní identifikátor, označovaný jako Object Identifier (OID), který je reprezentován číselným řetězcem. Pro usnadnění práce s těmito identifikátory se používá databáze Management Information Base (MIB), která svazuje číselnou reprezentaci s lépe uchopitelným textovým řetězcem. Příkladem OID identifikátoru může být například `1.3.6.1.2.1.2.2.1.6.1`, které odpovídá textová verze z MIB databáze `iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifPhysAddress`. S využitím příslušného softwaru lze hodnoty jednotlivých proměnných vykreslovat do grafů a získat tak přehled o časovém průběhu. Lze také definovat limitní hodnoty pro příslušné proměnné, při jejichž překročení se spustí určitá akce - tou bývá nejčastěji upozornění administrátora prostřednictvím například E-mailu či SMS. Pokud tak kupříkladu vytížení procesoru centrálního přepínače přesáhne hodnotu 90%, dohledový systém na tuto skutečnost upozorní příslušného technika, který může situaci začít neprodleně řešit. Další možností je zasílání speciálních chybových zpráv, takzvaných *trapů* přímo ze zařízení, dojde-li k nějaké definované události. SNMP je proto hojně využíván v systémech pro síťovou správu, kde tvoří základní komponentu dohledu nad provozním stavem sítě.

Kromě hodnot reprezentujících provozní údaje existují také proměnné obsahující konfigurační údaje. Tyto proměnné lze nejen vyčítat, ale také zapisovat. Zápisem těchto údajů tak lze v podstatě měnit konfiguraci síťových prvků, což byl také jeden ze záměrů autorů tohoto protokolu. Avšak vzhledem k absenci standardizovaného a automatického postupu pro nalezení (správných) MIB modulů se systém číselných identifikátorů ukázal být pro tento účel příliš komplexní a k nasazení SNMP coby protokolu pro konfiguraci síťových prvků tak ve větší míře nikdy nedošlo.

### 2.2.3 NETCONF

Na konci 80. let minulého století vyvinula organizace Internet Engineering Task Force (IETF) protokol SNMP, který se postupně stal velmi populárním protokolem pro správu počítačových sítí. Na počátku 21. století však bylo již zřejmé, že navzdory původnímu záměru nebyl SNMP

používán pro konfiguraci síťových prvků, ale především pro jejich monitoring. V červnu roku 2002 se proto sešli členové Internet Architecture Board (IAB) se správci sítí, aby diskutovali vzniklou situaci. Výsledky tohoto jednání jsou popsány v dokumentu RFC 3535 [2]. Z tohoto jednání jasně vyplynulo, že síťoví správci pro konfiguraci zařízení používali primárně CLI, které mělo mnoho aspektů, které správcům vyhovovaly. Jednou z hlavních předností CLI byla právě jeho textová povaha, která byla mnohem přehlednější než OID řetězce u SNMP. Další překážkou v rozšíření SNMP coby konfiguračního protokolu byla skutečnost, že mnozí výrobci vůbec neumožňovali plnohodnotnou konfiguraci svých zařízení jeho prostřednictvím. Síťovým operátorům se také velmi zamlouvala možnost konfigurovat prvky prostřednictvím skriptů, avšak v tomto ohledu mělo naopak CLI značné nedostatky, především nepředvídatelnost výstupu. Povaha a struktura výstupu se často měnila, což vedlo k častým chybám a nepředvídaným výsledkům.

Přibližně ve stejné době využívala společnost Juniper Networks postup pro konfigurování síťových prvků založený na XML. Tento přístup byl přednesen IETF a širší komunitě, což vyústilo v sestavení pracovní skupiny NETCONF v květnu roku 2003. Cílem této skupiny bylo vytvoření protokolu, který by více vyhovoval potřebám síťových operátorů a výrobcům aktivních prvků.

Výsledkem snažení této skupiny byl vznik protokolu NETCONF. Ten poskytuje mechanismy pro instalaci, manipulaci a odstranění konfigurací síťových prvků. Jeho transakce jsou vybudované na jednoduché Remote Procedure Call (RPC) vrstvě. Pro přenos konfiguračních dat je využito XML, stejně jako pro protokolové zprávy, které jsou přenášeny prostřednictvím zabezpečeného protokolu. Koncepce protokolu NETCONF může být rozdělena do 4 vrstev:

1. *Content layer* - konfigurační data a upozornění
2. *Operations layer* - definuje sadu základních transakcí pro získávání a úpravu konfiguračních parametrů
3. *Messages layer* - poskytuje mechanismus pro zakódování RPC zpráv a upozornění
4. *Secure Transport layer* - poskytuje spolehlivý a bezpečný přenos zpráv mezi klientským zařízením a síťovým prvkem (serverem), nejčastěji je využit protokol SSH

**Typy transakcí** - základní protokol definuje tyto protokolové transakce:

- `<get>` - Získání aktuální konfigurace a stavových informací ze zařízení
- `<get-config>` - Získání celé konfigurace či její části ze zvoleného úložiště
- `<edit-config>` - Úprava konfigurace v daném úložišti pomocí vytvoření, sloučení nebo smazání obsahu
- `<copy-config>` - Kopírování celé konfigurace z jednoho umístění do druhého
- `<delete-config>` - Smazání konfigurace z daného úložiště
- `<lock>` - Uzamčení konfiguračního úložiště
- `<unlock>` - Odemknutí konfiguračního úložiště

- `<close-session>` - Vyžádání korektního ukončení NETCONF relace
- `<kill-session>` - Vynucení ukončení NETCONF relace

Transakce v protokolu NETCONF poskytují jednoduchý, nezávislý mechanismus pro kódování zpráv.

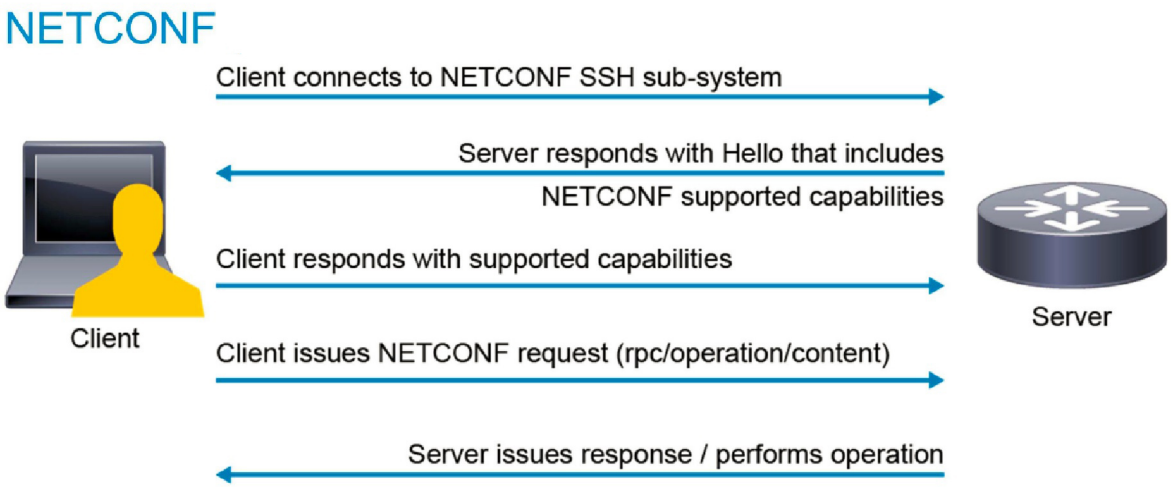
- RPC invokace - `<rpc>` zprávy
- RPC výsledek - `<rpc-reply>` zprávy
- Upozornění - `<notification>` zprávy

Každá zpráva představuje Extensible Markup Language (XML) dokument. RPC výsledky jsou vázány na RPC invokace prostřednictvím atributu `<message-id>`. Zprávy tak mohou být posílány za sebou bez nutnosti čekání na odpověď před vysláním další zprávy. RPC zprávy jsou definovány v dokumentu RFC 6241.

### Využití více konfiguračních úložišť

Jedním ze základních požadavků při návrhu NETCONF protokolu bylo umožnit existenci vícero "paralelních"konfigurací na jednom zařízení, aby bylo možné provádět konfigurační změny v konfiguraci, která není na zařízení v danou dobu aktivní (například v terminologii Cisco IOS `running-config`) a po provedení potřebných změn tyto změny najednou aplikovat. NETCONF proto s tímto požadavkem počítá a umožňuje spravovat několik konfigurací a tyto konfigurace následně libovolně aplikovat, upravovat, kopírovat či odstraňovat.

### Průběh komunikace v Protokolu NETCONF




Obrázek 2.1: NETCONF komunikace Klient-Server

Po přihlášení do SSH-subsystému pošle server (síťový prvek) klientovi seznam svých podporovaných funkcí, načež klient serveru pošle zpět seznam jím podporovaných funkcí. Následně může klient posílat serveru příkazy.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
</capabilities>
<session-id>203499</session-id>
</hello>]]>]]>
```

Ukázka 2.3: Ukázka NETCONF zprávy server hello

Zpráva `hello` obsahuje podporované datové modely, podle kterých se řídí následující příkazy. Tím se dostáváme k problematice modelování síťových prvků.

#### Modely YANG

Aby bylo možné sestavit korektní příkazy pro dané zařízení, je třeba znát jaké funkce dané zařízení podporuje. Za tímto účelem vznikají tzv. datové modely, které popisují možnosti daného zařízení. Jako příklad uveďme možnosti konfigurace jediného fyzického rozhraní. To může mít velkou řadu možných nastavení, u některých z nich je třeba zadat vhodné hodnoty a podobně. Například: Jedná se o L2 nebo o L3 rozhraní? Pokud L2, jedná se o access port či trunk port? Do jaké VLAN patří? Pokud jde o L3 rozhraní, jakou má IP adresu a masku? Má IP adresa správný formát? A samozřejmě mnoho dalších. Do značné míry se jedná o totéž, jako je MIB tabulka u SNMP, i když v mnohem přívětivějším pojetí. Za účelem takového popisu vznikly datové modely YANG, které se umožňují programátorům značně abstrahovat od fyzických zařízení a psát daný kód v souladu s těmito modely. Tyto modely jsou většinou nabízené přímo výrobcem daného zařízení, nebo je zde možnost takový model vytvořit a následně jej do zařízení nahrát. Bohužel NETCONF existuje déle než YANG modely a zdaleka ne všechna zařízení s touto možností počítají [3].

## 2.3 Síťové řadiče

### 2.3.1 Přehled síťových řadičů

Síťový řadič (angl. *controller*) představuje základní stavební kámen softwarově definované sítě (SDN). Termín SDN se v posledních letech stal natolik rozšířeným (či nadužívaným), že jeho všeobecně uznávaná definice prakticky neexistuje. Původním záměrem SDN bylo rozdělení síťové

infrastruktury do dvou logických vrstev: *controll plane* a *data plane*. *Controll plane* vrstva představuje celkovou logiku sítě, která se zabývá řízením datových toků [4]. Určuje tedy jakým způsobem budou jednotlivé pakety a rámce v síti putovat. *Data plane* je v tomto pojetí realizována samotnými fyzickými zařízeními, která však sama o sobě disponují pouze minimální logikou a zprostředkovávají jednoduše "přeposílání paketu z portu na port". Veškerá pravidla provozu tak mohou být definována na jednom centrálním místě (kontroleru) a následně jsou nasazena do síťových prvků. Ve výsledku by tak koncová zařízení mohla být od různých výrobců a potenciálně mnohem levnější, než zařízení disponující vlastní logikou. Jakmile by síťový prvek přijal paket, pro který by neměl vhodné pravidlo, dotázal by se kontroleru, jakým způsobem s tímto paketem naložit. Kontroler by na základě informací z hlavičky daného paketu a informací o síti v danou chvíli určil, na jaké výstupní rozhraní má být paket předán. Tuto komunikaci mezi síťovým prvkem a kontrolerem nejčastěji zajišťuje protokol *OpenFlow*. Tímto přístupem vznikne virtualizovaná síť, která je schopna se velmi rychle přizpůsobovat příchozím požadavkům. Vzhledem k tomu, že veškerá logika sítě je obsažena v softwarovém kontroleru, je možné se do jisté míry chovat k celé síti jako k softwaru a měnit její chování prostřednictvím nejrůznějších aplikací. Jedním z rozšířených projektů v oblasti takovýchto SDN kontrolerů je například *OpenDayLight*.

Centralizace veškeré logiky datové sítě je však zároveň hlavní nevýhodou tohoto přístupu. Vzniká tím totiž *single point of failure*, což znamená, že v případě výpadku síťové konektivity ke kontroleru, či poruše samotného kontroleru, dojde zároveň k výpadku celé sítě. Aby se předešlo možnému výpadku kontroleru, je nezbytné provozovat paralelně více kontrolerů v režimu vysoké dostupnosti (angl. High Availability (HA)). Kromě toho je samozřejmě nezbytné navrhnout efektivní a redundantní konektivitu ke kontroleru, což s sebou přináší další výzvy. Další problém představuje samozřejmě bezpečnost tohoto řešení - v případě, že se útočníkovi podaří získat přístup ke kontroleru, má efektivní kontrolu nad celou sítí. V neposlední řadě je problémem latence - komunikace síťového prvku s kontrolerem trvá nezanedbatelnou dobu, která se může negativně projevit na výsledné kvalitě služby, což je problém především u takových služeb, které podléhají Service Level Agreement (SLA). Ve výsledku je tak potřeba zprovoznit celou řadu komplexních funkcionalit, čímž se provozování spolehlivé softwarově definované sítě stává nelehký úkol [5].

Aby se minimalizovala možná rizika spojená s výpadkem centrálního kontroleru, je možné logiku sítě (*controll plane*) provozovat dále do určité míry decentralizovaně na jednotlivých prvcích, avšak provádění konfiguračních změn přenechat na kontroleru. Pokud v takovém případě dojde k výpadku kontroleru, nebude sice možné provádět řízené změny v chování sítě, avšak síť samotná zůstává nadále samostatně funkčním celkem. Kontroler tak i nadále přináší mnohé výhody, ale není už kriticky nezbytnou komponentou. Právě tento přístup k problematice Software Defined Network (SDN) využívají i kontrolery společnosti Cisco, které budou blíže popsány v následující části této kapitoly.

Již bylo zmíněno, že termín SDN je poměrně široký a těžko uchopitelný. Proto pod něj lze

zahrnout i takové postupy, které pomocí nejrůznějších softwarových nástrojů a skriptů komunikují se síťovými prvky za účelem jejich konfigurace. Jedním z takovýchto nástrojů je například *Ansible*, což je open-source softwarový framework navržený původně pro správu a orchestraci serverů s operačním systémem Linux. Díky své síle a jednoduchosti ale začal rychle pronikat i do dalších odvětví IT, síťové prvky nevyjímaje. Ačkoliv se tradiční routery a switche výrazně liší od Linuxových serverů, díky modulární povaze Ansiblu je možné vytvořit moduly usnadňující komunikaci i s těmito zařízeními. V současnosti disponuje Ansible širokou paletou modulů pro většinu největších výrobců síťových zařízení. Za těmito moduly stojí často buď sami výrobci, nebo jsou výtvorem široké komunity obklopující tento projekt. Výhodou Ansiblu oproti tradičním kontrolerům je například to, že jej lze často provozovat na osobním počítači síťového inženýra, s minimálními nároky na výpočetní výkon. Jedná se však především o textový nástroj a postrádá tedy líbívá grafická rozhraní, která jsou standardní součástí běžných síťových kontrolerů.

Ať už je řeč o jakékoliv z mnoha rozličných forem softwarově definovaných sítí, všechny bohužel naráží na jeden společný problém: nedůvěru síťových administrátorů. Centralizované řízení celé síťové infrastruktury prostřednictvím jediného kontroleru je sice z mnoha důvodů lákavé, je ale zároveň přinejmenším znepokojivé. Mnoho administrátorů má z pochopitelných důvodů obavy předat plný přístup ke konfiguraci všech zařízení v síti nějakému softwaru se slepou vírou, že se v onom softwaru neobjeví chyba. Tyto nástroje sice dokážou spravovat a konfigurovat veškeré aktivní prvky v rekordním čase, často s minimem úsilí, avšak ve stejně rekordním čase mohou zapříčinit výpadek celé sítě. Při doposud běžném konfiguračním postupu, kdy technik provádí změny konfigurace jednoho zařízení prostřednictvím příkazové řádky, může samozřejmě dojít k chybě. Vetšinou ale taková chyba nezpůsobí nedostupnost veškerých služeb v síti a je možné ji v přiměřeném čase opravit. Pokud se ale takováto chyba spuštěním jediného příkazu na orchestračním nástroji projeví ne na jednom, ale na desítkách zařízení, je to samozřejmě situace zcela jiná. Nelze tedy říct, že by tyto nástroje dokázaly eliminovat lidské chyby, protože je naopak mohou často znásobit do ohromných rozměrů. Takovýmto chybám z nedbalosti lze samozřejmě předcházet vhodným testováním a prací s rozvahou. I tak je zde ale otázka: Do jaké míry může administrátor sítě věřit, že není chyba v samotném orchestračním nástroji? Jakou má jistotu, že v některém nepředvídaném případě nedojde k chybě, která bude mít fatální následky pro uživatele, síť samotnou a samozřejmě právě jejího správce? Odpověď je vcelku jednoduchá: minimální, či žádnou. Chyby v softwaru zkrátka existovaly vždy, a existovat pravděpodobně i zůstanou. Pokud správce či síťový inženýr na takovou chybu narazí, může samozřejmě zkontaktovat vývojáře daného nástroje či zástupce výrobce, a společnými silami se dopracovat k nápravě problému. Otázkou v takovém případě zůstává, zda-li to bude i nadále problém onoho technika, který chybu odhalil.

### 2.3.2 APIC-EM

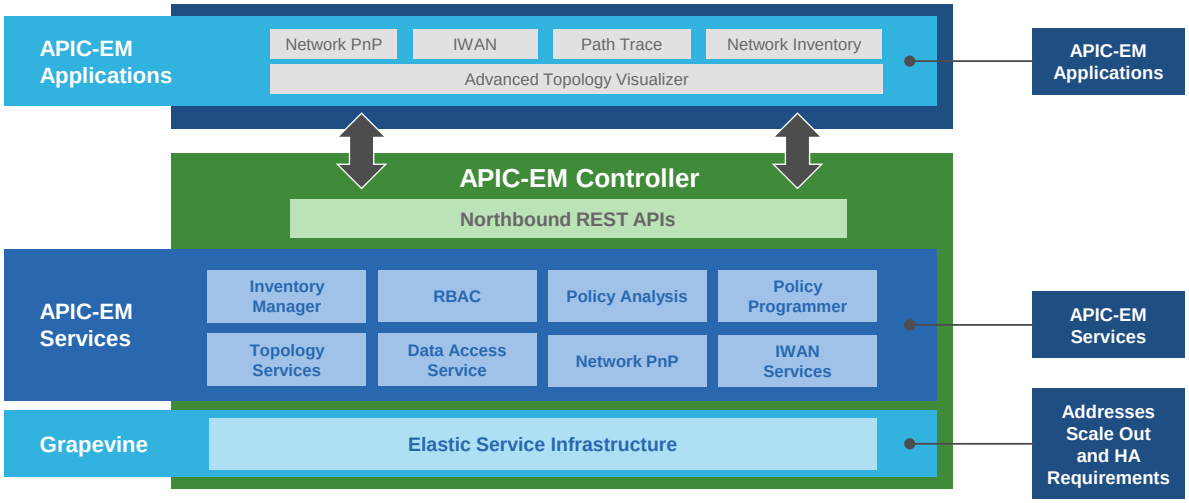
Síťový kontroler APIC-EM představuje SDN strategii společnosti Cisco pro prostředí podnikových sítí. V počátcích byl označován termínem *ACI-EM*, z čehož je patrné, že se jedná o jakousi alternativu Cisco Application Centric Infrastructure (ACI), která je navržena pro nasazení v datových centrech. Stejně jako ACI má i APIC-EM za úkol zprostředkovávat centrální přístup ke všem síťovým komponentám, avšak vzhledem k odlišnostem těchto dvou prostředí se různí i dostupné funkce. V prostředí datových center se například příliš často nesetkáváme s bezdrátovou infrastrukturou, která je ale jedním ze stěžejních bodů podnikové sítě. Dalším rozdílem je například přístup k síťové bezpečnosti, která je v datových centrech často prosazována už na okraji sítě (angl. *network edge*), zatímco v podnikových sítích bývá zajištěna formou centralizovaných firewallů [6].

Základní myšlenku však oba tyto kontrolery sdílejí: mají za úkol zabezpečit prosazování síťových politik (angl. *policy*) za účelem provozu aplikací v dané síti. Oproti tradičnímu přístupu, kdy administrátor musí nejdřív požadované politiky navrhnout a následně nakonfigurovat na jednotlivých zařízeních, v případě použití *"application-driven"* kontroleru se může soustředit pouze na jejich definovaní a samotné nasazení již přenechat na kontroleru. Kontroler již na základě předaných definic zajistí korektní provedení potřebných změn v konfiguraci jednotlivých zařízení, většinou na základě výrobcem stanovených doporučení.

Kromě prosazování síťových politik a faktické konfigurace sítě nabízí kontrolery také velmi atraktivní možnosti pro přístup k údajům z jednotlivých zařízení. Takto získané údaje lze vhodným způsobem přeměnit v potenciálně velmi užitečné informace, ať už pro tvorbu dokumentace sítě, usnadnění hledání a odstraňování chyb či jako výchozí údaje pro případné konfigurační změny. Právě těmito možnosti se bude zabývat další část této práce.

#### Základní parametry APIC-EM

APIC-EM má oproti jiným softwarovým nástrojům společnosti Cisco jednu velkou výhodu: je zcela zdarma. Díky tomu si jej může takřka kdokoliv stáhnout a nainstalovat, přičemž jediným kritériem pro provoz je splnění minimálních systémových požadavků. Ty v případě *single-node* instalace představují 6 CPU jader o taktu alespoň 2.4 GHz, 64 GB operační paměti a 500 GB volného prostoru na disku. Lze jej nainstalovat jak na fyzický server, tak do virtuálního prostřední VMware ESXi [7]. Architekturu APIC-EM popisuje obrázek 2.2. Jednotlivé služby jsou spouštěny jako LXC kontejnery nad operačním systémem Linux Ubuntu. Správu těchto virtuálních kontejnerů zajišťuje Grapevine [8]. Pro běžnou uživatelskou interakci slouží webové grafické rozhraní, které nabízí několik aplikací, reprezentující specifické funkce kontroleru. Tyto aplikace budou podrobněji popsány v další části této kapitoly.




Obrázek 2.2: Architektura APIC-EM

Prvním krokem po úspěšném dokončení instalace APIC-EM je přirozeně přidání existujících zařízení do databáze. Samotným přidáním zařízení není nijak ovlivněna jeho konfigurace, jedná se tedy o poměrně bezpečný krok. Nalezení dostupných síťových prvků realizuje nástroj *Discovery*, který nabízí dvě možnosti:

- CDP
- Range

Možnost *CDP* umožňuje automaticky nalézt všechna dostupná zařízení prostřednictvím protokolu Cisco Discovery Protocol (CDP). Tato metoda vyžaduje zadání IP adresy jednoho zařízení (označované jako *seed*), spolu s přístupovými údaji (uživatelské jméno, heslo a heslo při přístup do privilegovaného režimu). APIC-EM se následně pokusí k tomuto zařízení připojit a nalézt sousedící prvky. Poté se pokusí zkontaktovat i tato zařízení, najít jejich sousedy a tak dále. Ačkoliv je tato metoda velmi jednoduchá, skýtá i některá úskalí. Některá z nich budou podrobněji diskutována v následující kapitole.

Druhá z možností, *Range* vychází z předpokladu kvalitního návrhu sítě, kdy jsou rozhraní určená pro správu zařízení (označovaná jako *management* rozhraní) součástí jednoho adresního rozsahu. V rámci tohoto rozsahu se APIC-EM pokouší připojit k jednotlivým adresám (zařízením).

Výsledkem úspěšného discovery procesu je přidání všech podporovaných zařízení do databáze, která je periodicky aktualizována. Díky údajům z této databáze je tak možné sestavit interaktivní graf reprezentující topologii dané sítě. K informacím obsaženým v této databázi je možné přistupovat buď přímo v rámci grafického rozhraní na záložce *Device inventory*, či využitím REST API. Informace obsažené v databázi jsou získávány prostřednictvím rozhraní, označovaných jako *south bound* API. Tato rozhraní zahrnují:

- CLI

- SNMP
- NETCONF
- RESTCONF (REST API)

### Aplikace APIC-EM

Kromě údajů obsažených v informační databázi kontroleru, jako jsou údaje o jednotlivých zařízeních, jejich rozhraních, konfiguracích a podobně, poskytuje APIC-EM také aplikace, které poskytují další funkce. Některé z těchto aplikací jsou součástí základní instalace kontroleru, jiné je třeba doinstalovat či případně zakoupit příslušnou licenci. Mezi dostupné aplikace patří například:

- *Topology*: Poskytuje interaktivní zobrazení síťové topologie včetně připojených koncových zařízení
- *Path Trace*
- *Network Plug and Play*
- *EasyQoS*

#### Path Trace

Aplikace *Path Trace* umožňuje simulovat průchod datového toku skrze síťovou infrastrukturu, a to čistě na základně informací získaných ze zařízení. Tuto aplikaci tak lze použít k ověření konektivity mezi dvěma uzly v síti, aniž by bylo nezbytné reálně posílat jakákoliv data. Aplikace využívá údaje z MAC tabulek, směrovacích tabulek a bezpečnostních pravidel (*access listů*), aby určila jakým způsobem bude testovaný rámec v síti přenášen. Pro spuštění testu je třeba zadat pouze zdrojovou a cílovou IP adresu, přičemž obě adresy musí patřit připojeným zařízením. Dále lze zadat také MAC adresy požadovaných uzlů, případně protokol čtvrté vrstvy OSI modelu a čísla zdrojových a cílových portů. Aplikace na základě těchto údajů spolu se znalostí síťové topologie určí u každého zařízení vstupní a výstupní rozhraní, otestuje zda simulovaný paket není v rozporu s některým z access listů a následně celou trasu paketu graficky vykreslí. Pokud je simulovaný paket po cestě "zahozen"některým pravidlem, je toto místo v trase označeno spolu s informací o konkrétním pravidlu. Zároveň lze pro jednotlivá rozhraní zobrazit statistiky vytíženosti a případně ztrátovosti.

Typický příklad využití této aplikace je následující: Administrátor sítě obdrží stížnost od uživatele, který se nemůže ze svého počítače připojit k síťovému úložišti. Administrátor zadá potřebné údaje do aplikace *Path Trace*, a během chvíle je schopen říci, zda je ohlášená nedostupnost služby způsobena problémem se síťovou konektivitou, či nikoliv.

#### Network Plug and Play

Tato aplikace je patrně jednou z nejatraktivnějších, které APIC-EM nabízí. Umožňuje totiž nasazení nových zařízení do sítě bez nutnosti jakékoliv konfigurace na straně zařízení. V rámci aplikace je nejdříve třeba vytvořit pravidlo pro dané zařízení. Pravidlo obsahuje základní údaje

o zařízení, jako je zamýšlený název zařízení, sériové číslo, přístupové údaje, požadovanou verzi operačního systému a podobně. Součástí pravidla je také konfigurační soubor pro dané zařízení, případně konfigurační šablona obsahující proměnné. Tyto proměnné typicky zahrnují název zařízení (*hostname*), IP adresu na rozhraní určeném pro správu či další požadované údaje. Na základě vyplnění hodnot pro tyto proměnné je následně vygenerován potřebný konfigurační soubor.

Následuje samotné připojení nového zařízení do sítě. Novější zařízení Cisco, jejichž verze operačního systému je pro tuto možnost připravena, obsahují *PnP* agenta. Tento agent se spustí při prvotním zapnutí zařízení a snaží se nalézt *PnP* server, v tomto případě APIC-EM. Adresa kontroleru může být zařízení sdělena buď prostřednictvím Domain Name System (DNS) záznamu, případně speciální hodnotou v odpovědi Dynamic Host Configuration Protocol (DHCP) serveru. PnP klient na zařízení totiž nastaví výchozí rozhraní do režimu DHCP klient, aby mohl navázat IP konektivitu. Jakmile zná PnP agent adresu kontroleru, naváže s ním prvotní kontakt. Pokud kontroler rozpozná ohlašující se zařízení na základě některého z definovaných pravidel, je možné pomocí jediného kliknutí zahájit proces nastavování zařízení. Pokud žádné pravidlo neodpovídá, je zařízení zobrazeno v příslušné sekci, kde je třeba doplnit požadované údaje. Jakmile je zahájen proces nastavování zařízení (*provisioning*), nahraje APIC-EM do zařízení příslušnou verzi operačního systému a následně i požadovanou konfiguraci.

Tento postup zvolila například jedna nejmenovaná společnost v České republice, která potřebovala zajistit obnovu více než 300 síťových prvků. Pro potřeby tohoto projektu byl nainstalován kontroler APIC-EM, do něhož byla nahrána jednotná konfigurační šablona pro tato zařízení. Na základě informací od dodavatelské společnosti byla hromadně definována pravidla pro jednotlivá zařízení na základě jejich výrobních čísel. Technici dodavatelské společnosti tak mohli provést pouze fyzikou instalaci zařízení v dané lokalitě. Po zapnutí zařízení zkontaktovalo kontroler, který již zajistil instalaci operačního systému a nahrání potřebné konfigurace. Jediným úkolem technika zadavatelské společnosti tedy následně bylo jedno kliknutí v grafickém rozhraní, kterým potvrdil konkrétní zařízení.

Automatizovaná instalace zařízení je velmi efektivním způsobem pro nasazení velkého počtu zařízení. Pokud je proces předem dostatečně otestován, v podstatě odpadá riziko lidské chyby. Takovýto postup je výhodnější jak po časové, tak finanční stránce, neboť vyžaduje minimální úsilí a klade jen velmi nízké požadavky na kvalifikaci technika provádějícího fyzickou instalaci zařízení.

### 2.3.3 DNA Center

Mohlo by se zdát, že Digital Network Architecture Center (DNAC) je jakousi další vývojovou větví kontroleru APIC-EM. To je do určité míry pravda, neboť DNAC v sobě zahrnuje právě APIC-EM. Kromě toho však přináší zcela nový přístup k problematice správy podnikových sítí.

S pojmem DNAC také úzce souvisí termín Software Defined Access (SDA), který razantním způsobem mění pohled na tradiční přístupové sítě.

Tradičně je přístupová vrstva sítě tvořena přepínači (switchi), přičemž uživatelská a jiná koncová zařízení jsou logicky rozdělena do virtuálních sítí na úrovni spojové vrstvy OSI modelu (Virtual Local Area Network (VLAN)). Většinou je každá takováto virtuální síť je zakončena na L3 prvku (síťová vrstva), kterým může být router či L3 switch. V rámci tohoto prvku jsou následně definována pravidla, která určují, jakým způsobem může probíhat komunikace mezi jednotlivými virtuálními sítěmi. V případě komunikace v rámci jedné VLAN sítě je provoz *přepínán* (*switchován*), zatímco datové pakety směřující do jiné VLAN sítě jsou *směrovány* (*routovány*) prostřednictvím centralizovaného L3 prvku.

V případě SDA je však veškerý provoz v síti směrován, a to už od úrovně přístupového prvku. Tento rozdíl přináší řadu změn. Zjednodušeně lze říci, že směrovaný provoz je mnohem snáze kontrolovatelný, než provoz přepínaný. Díky logické IP adresaci zařízení a využití směrovacích protokolů lze docílit mnohem determinističtější povahy provozu, avšak za cenu mnohem náročnější konfigurace na straně síťových prvků. Kontroler DNAC nabízí právě schopnost provádět tyto konfigurační změny. Pomocí virtuálních L3 sítí - VxLAN, které jsou tradičně doménou spíše datových center, je nad fyzickou topologií sítě (označovanou jako *underlay*) vytvořena logická síť (označována jako *overlay* či *fabric*). Uživatelská zařízení jsou přiřazována do příslušných virtuálních sítí na základě ověření svojí identity prostřednictvím protokolu IEEE 802.1X. V rámci této virtuální *overlay* sítě lze následně snadno definovat pravidla provozu mezi jednotlivými skupinami - virtuálními sítěmi.

Aby bylo možné zprovoznit tuto směrovanou virtualizovanou síť, musí všechny prvky v dané síti disponovat L3 funkcemi a podporou VxLAN. V souvislosti s příchodem DNAC vydala společnost Cisco také novou řadu přepínačů (*Catalyst 9000*), které jsou plně připraveny pro nasazení v takovéto síti. Kromě přepínačů z této řady jsou podporovány také některé starší prvky, které disponují potřebnými funkcemi.

Kromě definování pravidel v SDA sítí slouží DNAC také jako centrální prvek pro monitoring stavu celé sítě. Kromě tradičního zpracovávání informací získaných protokolem SNMP umožňuje DNAC zpracovávat také informace o datových tocích v síti a vytvářet statistiky ukazující kvalitu jednotlivých služeb v síti.

Stejně jako APIC-EM, i DNAC disponuje programovým rozhraním REST API. Vzhledem k tomu, že součástí kontroleru DNAC je právě APIC-EM, který slouží jako prostředník pro komunikaci se síťovými prvky, není překvapením, že i syntaxe a struktura příkazů REST API je u obou kontrolerů velmi podobná. S příchodem placeného kontroleru DNAC však trochu vyvstává otazník nad budoucností jeho zdarma dostupného předchůdce. Podle dostupných informací však lze usuzovat, že APIC-EM bude i nadále podporován.

# Kapitola 3

# Návrh softwarového nástroje

Tato kapitola se věnuje návrhu a realizaci softwarového nástroje, jehož cílem je získávání dat ve vhodném formátu pro další zpracování za účelem usnadnění správy a řešení běžných provozních úkonů při provozu datových sítí. Jedním z prvotních záměrů pro tento nástroj bylo rozšíření možností kontroleru APIC-EM i na zařízení, která nejsou tímto kontrolerem podporovaná. Mezi taková zařízení spadají jednak starší prvky společnosti Cisco, tak potenciálně zařízení jiných výrobců. Nástroj by tedy měl poskytovat jednotné rozhraní pro získávání konfiguračních a provozních dat ze zařízení ať už prostřednictvím APIC-EM, tak cestou přímé komunikace s aktivními prvky sítě. Nástroj by zároveň mělo být možné integrovat s existujícími nástroji pro správu sítě, či jej využít pro tvorbu například webových aplikací.

## 3.1 Základní cíle

- Získávání konfiguračních a stavových údajů ze zařízení ve formátu vhodném pro programové zpracování
- Usnadnění časově náročných úkonů při správě, údržbě a troubleshootingu datových sítí
- Poskytnutí jednotného programového rozhraní bez ohledu na typ síťového prvku

## 3.2 Analýza dostupných možností

### 3.2.1 Programovací jazyk

Prvotní volbou při tvorbě jakéhokoliv softwaru je samozřejmě volba vhodného programovacího jazyka. Možností je zde přirozeně celá řada, avšak zvolený jazyk by měl splňovat alespoň tyto požadavky:

- *Vysokoúrovňový jazyk*: neboli jazyk s vysokou mírou abstrakce od specifických požadavků operačního systému či hardwaru. Díky této abstrakci je takový jazyk snáze uchopitelný i pro lidi, kteří se běžně programováním či skriptováním nezabývají. Díky tomu lze zároveň takovýto jazyk provozovat na různých platformách, bez nutnosti změn zdrojového kódu.
- *Vhodný pro zamýšlené použití*: v tomto případě pro nasazení v oblasti datových sítí. Pro usnadnění některých úkonů by měly existovat volně dostupné knihovny s dostatečnou uživatelskou základnou.
- *Zohlednění existujících nástrojů*: má-li být výsledný software snadno integrovatelný s některými z existujících nástrojů, použití stejného jazyka je samozřejmě výhodou.

Vysokoúrovňových, multiplatformních a volně dostupných programovacích jazyků lze najít mnoho, kupříkladu *Java*, *C#* nebo *Ruby*. Pokud se ale zaměříme na problematiku datových sítí, velmi brzy zjistíme, že v tomto odvětví je jasným číslem jedna *Python*. Obecně lze říci, že *Python* má poměrně plochou učící křivku - tedy že jeho základy je možné se naučit v poměrně krátké době. Většina školení, návodů či tutoriálů z oblasti programovatelných sítí rovněž využívá právě tento jazyk, díky čemuž lze snáze a rychleji zvládnout základní principy a postupy tohoto odvětví. Je to do značné míry i díky tomu, že existuje řada volně dostupných knihoven zaměřujících se na problémy spojené s programových přístupem k datovým sítím, kolem kterých existují poměrně rozsáhlé komunity uživatelů, které neustále pracují na jejich testování a zdokonalování. Mezi jeden z takovýchto hojně rozšířených projektů patří například právě již dříve zmiňovaný orchestrační framework *Ansible*. Neposledním faktorem hovořícím ve prospěch Pythonu je i například to, že je preferován samotnými výrobci síťových prvků - kupříkladu společnost Cisco staví veškeré výukové materiály z oblasti síťové programovatelnosti právě na tomto jazyce. Integrace jazyka Python jde dokonce tak daleko, že v novějších verzích operačního systému Cisco IOS XE je dostupný Python interpretr, který umožňuje spouštět uživatelské skripty přímo na síťových prvcích. Právě z těchto důvodů byl pro tvorbu zamýšleného nástroje zvolen Python, přičemž určitou roli hraje, jako vždy, osobní preference.

### 3.2.2 Komunikační rozhraní

Právě samotné získávání údajů z aktivních prvků představuje jednu z hlavních výzev této práce. V ideálním případě by komunikační rozhraní mělo být schopno poskytovat data v některém ze strojově zpracovatelných formátů, jako je například JSON či XML. V případě komunikace se síťovým kontrolerem se jedná o poměrně jasnou volbu, kterou představuje rozhraní REST API. Většina soudobých kontrolerů disponuje tímto rozhraním, jehož prostřednictvím lze poměrně snadno získávat požadovaná data ve vhodném formátu, nejčastěji JSON. Vzhledem k tomu, že tato rozhraní jsou realizována prostřednictvím jednoho z nejhojněji využívaných protokolů HTTP, není problém nalézt vhodnou knihovnu pro zprostředkování této komunikace. V případě přímé komunikace se síťovým prvkem je však volba komunikačního rozhraní mnohem složitější.

V předchozí části práce věnované různým typům komunikačních rozhraní 2.2 byly popsány jejich základní vlastnosti, výhody a nevýhody. Možný výběr tedy zahrnuje tato rozhraní:

- Command Line Interface *(Příkazová řádka)* (CLI)
- Simple Network Management Protocol (SNMP)
- Protokol NETCONF
- REST API

Z dříve uvedených důvodů můžeme z výběru rovnou vyloučit protokol SNMP. Ačkoliv je vhodný pro periodické vyčítání provozních údajů, pro čtení či dokonce zapisování konfiguračních dat se zjevně nehodí. Oproti tomu REST API by se zdálo být nejvhodnějším kandidátem pro zadaný účel. Díky stejnému komunikačnímu protokolu, jaký je použit u síťových kontrolerů, by bylo možné využít takřka tutéž sadu funkcí pro oba případy, a získávat data rovnou ve velmi podobné struktuře a formátu. Smutnou realitou ovšem je, že toto rozhraní není zdaleka dostatečně rozšířené. Omezíme-li se na produkty společnosti Cisco, zjistíme, že tímto rozhraním disponují pouze datacentrové přepínače z rodiny Nexus s operačním systémem NX OS. Podíváme-li se na zařízení s operačním systémem IOS či IOS XE, určená pro nasazení v podnikových sítích, nalezneme toto rozhraní pouze v posledních verzích systému IOS XE, a to pouze na vybraných platformách ASR a CSR1000V. Vzhledem k tomu, že jedním ze základních cílů navrhovaného řešení má být právě podpora starších zařízení, není vhodné volit REST API jako primární komunikační rozhraní. Není zde však ani důvod toto rozhraní zcela zavrhovat, neboť do budoucna jistě nabízí velký potenciál. Potenciálně nejlepším kandidátem tedy zůstává protokol NETCONF. Situace s rozšířeností tohoto protokolu je sice značně přívětivější, než je tomu u modernějšího REST API, avšak zdaleka nelze říci, že by tento protokol byl dostupný na všech platformách síťových prvků. Ačkoliv byl standard pro NETCONF vydán již v roce 2006, v současných sítích se stále vyskytuje velké množství zařízení, které jej buď nepodporují vůbec, nebo jen ve velmi omezené míře. Navíc podpora datových modelů Yet Another Next Generation (YANG) pro tento protokol byla v případě Cisco zařízení přidána také až v novějších verzích operačních systémů. Ačkoliv existují knihovny pro komunikaci prostřednictvím tohoto protokolu, samotné zpracování získaných dat je poněkud problematickým úkolem. Vzhledem k různé míře podpory na různých platformách a verzích operačního systému lze jen těžko předvídat strukturu a obsah získaných dat. Tento nedostatek se snaží vyřešit právě datové modely YANGs, které se ale také pro odlišné platformy různí. Navíc některé implementace protokolu NETCONF s těmito modely nepočítají. Ačkoliv tento protokol jistě může najít využití pro programovou komunikaci se zařízeními, nelze jej v současné době považovat za široce podporované a koherentní rozhraní [3]. Posledním možným rozhraním tak ironicky zůstává právě to rozhraní, které nikdy nebylo navrženo pro komunikaci se softwarem, ale s lidským operátorem. Tímto rozhraním je velmi dobře známá příkazová řádka, zkráceně CLI. Právě toto rozhraní využívá navržený softwarový nástroj jako primární komunikační kanál pro síťové prvky.

**Proč CLI?**

Ačkoliv se toto rozhraní nemusí zdát pro daný záměr příliš vhodné, existuje řada důvodů, která mluví ve prospěch jeho použití:

1. *Široce rozšířené*: Toto rozhraní je zdaleka nejrozšířenějším rozhraním využívaným pro konfiguraci všech síťových prvků. To znamená, že naprostá většina zařízení, se kterými se můžeme v počítačové síti setkat, minimálně tímto rozhraním disponuje, a to bez ohledu na stáří zařízení či verzi operačního systému.

2. *Povolené ve výchozím stavu*: Jedním ze standardních prvních kroků při konfiguraci síťového zařízení je právě umožnění přístupu k CLI prostřednictvím protokolu SSH či Telnet. Díky tomu není třeba žádné další konfigurace, aby bylo možné se zařízením komunikovat. Ačkoliv zprovoznění alternativních rozhraní, jako například NETCONF, vyžaduje minimální konfiguraci, už můžeme začít narážet na problémy s možnými bezpečnostními riziky, interními předpisy dané společnosti a podobně. Pomineme-li tyto možné administrativní překážky, stále se nacházíme v situaci, kdy je v prvotní fázi nezbytné ručně překonfigurovat všechna zařízení, což může být časově značně náročný proces.

3. *Familiérní syntaxe*: Díky velkému rozšíření jsou jednotlivé příkazy a očekávané výstupy poměrně dobře známy. Síťový technik tak ve většině případů ví, jaký příkaz je třeba použít pro získání požadovaných informací. Této skutečnosti využívá například REST API rozhraní operačního systému NX OS, kdy lze dotaz zformulovat prostřednictvím stejného příkazu, jako při použití běžné příkazové řádky. Toto je samozřejmě výhodou také pro budoucí přidávání podpory pro další příkazy a pomáhá udržovat přidružený zdrojový kód přehlednější.

4. *Stálost výstupního formátu*: Ačkoliv textový výstup příkazové řádky nepodléhá žádnému standardizovanému formátu, ve většině případů zůstává tento formát identický pro mnoho rozdílných platforem a verzí operačního systému. Nicméně je třeba vždy počítat s možností změny výstupního formátu a zajistit snadné přidání podpory i pro nový formát.

Základní vlastnost tohoto široce používaného textového rozhraní však nadále zůstává velkou nevýhodou pro programové zpracování. Výstupní data nejsou nikterak formátována a je tedy potřeba přijít s řešením, které dokáže tyto nestrukturované údaje přeměnit do vhodnějšího formátu. Pro strojové zpracování textových řetězců se hojně využívají *regulární výrazy* (angl. *regular expressions*). Regulární výrazy (zkráceně *regex* či *regexp*) představují definované vzory pro textové řetězce. Daný vzor je definován speciálním řetězcem (*regex string*), který umožňuje použití zástupných znaků. Díky tomuto vzoru lze ve zpracovávaném textu nalézt takové části, které odpovídají zadanému výrazu. To ale znamená, že pro každý textový výstup příkazové řádky je nezbytné sestavit jeden či více takovýchto regex řetězců. Tento postup se tak může zdát poměrně neefektivním. Pro jeho použití mluví ale mimo jiné i skutečnost, že i samotné

kontrolery společnosti Cisco (APIC-EM, DNAC) využívají pro vyčítání informací ze zařízení právě tuto metodu. Podrobněji se problematice zpracování textových výstupů věnuje část *Zpracování textových výstupů* 4.2.3.

# Kapitola 4

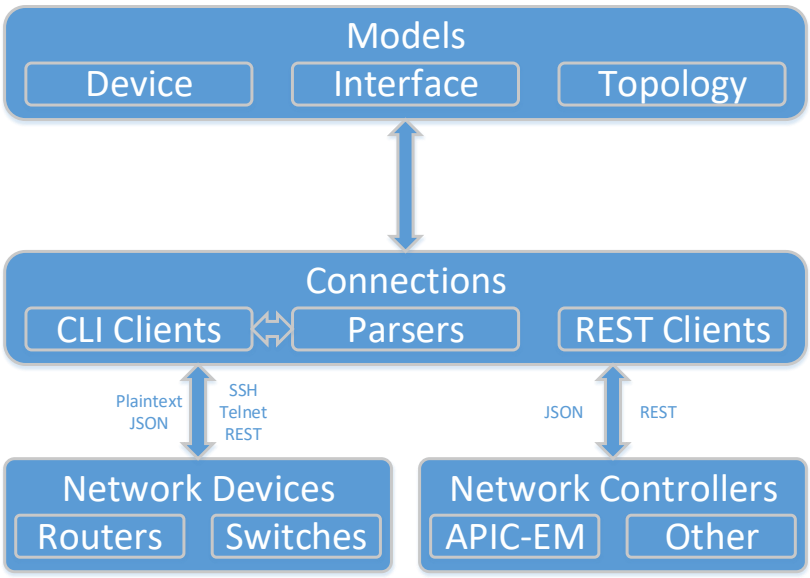
# Nástroj NUAAL

## 4.1 Architektura nástroje NUAAL

Pracovní název vytvářeného nástroje, NUAAL, představuje zkratku pro *Network Unified Abstraction API Library*. Vzhledem k povaze nástroje bylo nutné navrhnout vhodnou architekturu, která by umožnila efektivně plnit kladené požadavky. Nástroj tvoří knihovna napsaná v jazyce Python, obsahující dílčí moduly zaměřené na specifické funkce. Aby nástroj umožnil rozšířit některé funkce kontroleru Cisco APIC-EM, musí kromě komunikace se samotným kontrolerem umožňovat komunikaci i se síťovými prvky prostřednictvím příkazové řádky. Získaná data by měla být poskytována v jednotném formátu, ať už se jedná o spojení s využitím kontroleru či bez něj. Právě zde vyvstává první problematická otázka, které údaje zahrnout do výsledné datové sady a které nikoliv. Je zřejmé, že samotný kontroler nelze přimět k tomu, aby poskytoval údaje, které nejsou obsaženy v jeho databázi. V tomto případě je tedy i struktura i obsah informací poměrně pevně dán. Zároveň je třeba zmínit, že ačkoliv APIC-EM poskytuje cenné informace o síťových prvcích, jeho databáze zdaleka neobsahuje všechny potenciálně získatelná data.

V případě přímé komunikace se zařízením máme však plnou kontrolu nad tím, které informace chceme získávat. Omezovat se pouze na stejné údaje, které poskytuje APIC-EM by znamenalo snížení potenciálu celého nástroje. Má-li být NUAAL skutečně multiplatformním nástrojem, je třeba brát v potaz, že zařízení různých výrobců budou poskytovat vždy trochu jiné údaje. Definovat jednotnou strukturu dat už na nejnižší úrovni by bylo značně obtížné, a znamenalo by nutnost omezit obsah získaných dat na definované společné minimum. Z tohoto důvodu je architektura knihovny rozdělena do dvou základních vrstev: Nižší vrstva (označovaná dále jako *low-level* či *komunikační* vrstva) zahrnuje objekty realizující samotnou komunikaci a získávání dat ze síťových prvků, kontrolerů či dalších datových zdrojů. Pokud se jedná o čistě textová data, zajistí tato vrstva také jejich převedení do strojově zpracovatelného formátu, kterým je v tomto případě JSON. Tím je zajištěno, že výstupní data z této vrstvy obsahují veškeré dostupné

informace specifické pro daný typ připojení. Vyšší vrstva (dále označovaná jako *high-level* či *modelová* vrstva) je tvořena objekty, které dále zpracovávají data poskytnutá komunikační vrstvou takovým způsobem, aby výstupní data této vrstvy odpovídala pevně stanovenému formátu, bez ohledu na typ použitého typu spojení. Tímto způsobem je zajištěno, že objekt reprezentující datové rozhraní síťového prvku bude poskytovat informace ve stejném formátu bez ohledu na to, zda byly tyto informace vyčteny z kontroleru, aktivního prvku Cisco či jiného výrobce. Tuto logickou strukturu popisuje následující obrázek 4.1.




Obrázek 4.1: Architektura nástroje NUAAL

Celá knihovna se přitom snaží dodržovat principy objektově orientovaného programování (OOP). Funkce a metody společné pro více typů spojení jsou proto obsaženy v nadřazených objektech, ze kterých objekty zprostředkující konkrétní typ spojení dědí. Veškeré objekty komunikující prostřednictvím REST API tak například dědí společné funkce z rodičovského objektu `RestBase`, objekty komunikující přes CLI dědí z objektu `CliBase` a podobně. Tento přístup tak značně zjednodušuje budoucí rozšíření knihovny o nové typy spojení.

## 4.2 Komunikační vrstva

Komunikační vrstva knihovny se skládá z objektů obsažených v následujících modulech:

- *Connections*: Tento modul zahrnuje veškeré objekty sloužící k vytvoření datového spojení s cílovým zařízením. Modul je dále rozdělen na submoduly:
  - *CLI*: Obsahuje objekty reprezentující spojení se síťovými prvky prostřednictvím protokolů SSH a Telnet
  - *API*: Zahrnuje objekty pro komunikaci prostřednictvím rozhraní REST API, tedy například pro komunikaci s APIC-EM
- *Parsers*: Tento modul obsahuje knihovny potřebné pro získání strukturovaných dat z čistě textových výstupů. Kromě samotných parsovacích funkcí jsou zde obsaženy i databáze regulárních výrazů potřebných pro parsování.

### 4.2.1 Programový přístup k REST API

Vzhledem ke skutečnosti, že rozhraní REST API je založeno na protokolu HTTP, lze pro komunikaci využít velmi rozšířenou knihovnu `requests`. Následující ukázka 4.1 demonstruje základní užití knihovny `requests` pro provedení testovacího volání metodou `get`:

```
>>> import requests
>>> # Volani metodou HTTP GET
>>> response = requests.get("https://jsonplaceholder.typicode.com/posts/1")
>>> # Vypsani stavoveho kodu HTTP
>>> print(response.status_code)
200 # <- OK
>>> # Vypsani prijatych dat ve formatu JSON
>>> print(response.json())
{
'userId': 1,
'id': 1,
'title': 'sunt aut facere repellat provident occaecati ...',
'body': 'quia et suscipit\nsuscipit recusandae consequuntur ...'
}
>>>
```

Ukázka 4.1: Základní použití knihovny requests

Pro komunikaci se síťovým kontrolerem je však třeba poněkud komplexnějšího příkazu. Typicky je minimálně třeba nastavit hlavičky požadavku a poskytnout údaje pro autentizaci uživatele. Z tímto účelem existuje v knihovně NUAAL objekt `RestBase`, který má za úkol co nejvíce zjednodušit komunikaci přes toto HTTP rozhraní. Na základě tohoto objektu je vytvořen objekt `ApicEmBase`, který poskytuje funkce a metody specifické právě pro kontroler APIC-EM. Pro úspěšné navázání komunikace s kontrolerem je nejprve nezbytné získat autorizační token (označovaný jako *Service Ticket*). Tento token je generován samotným kontrolerem na základě

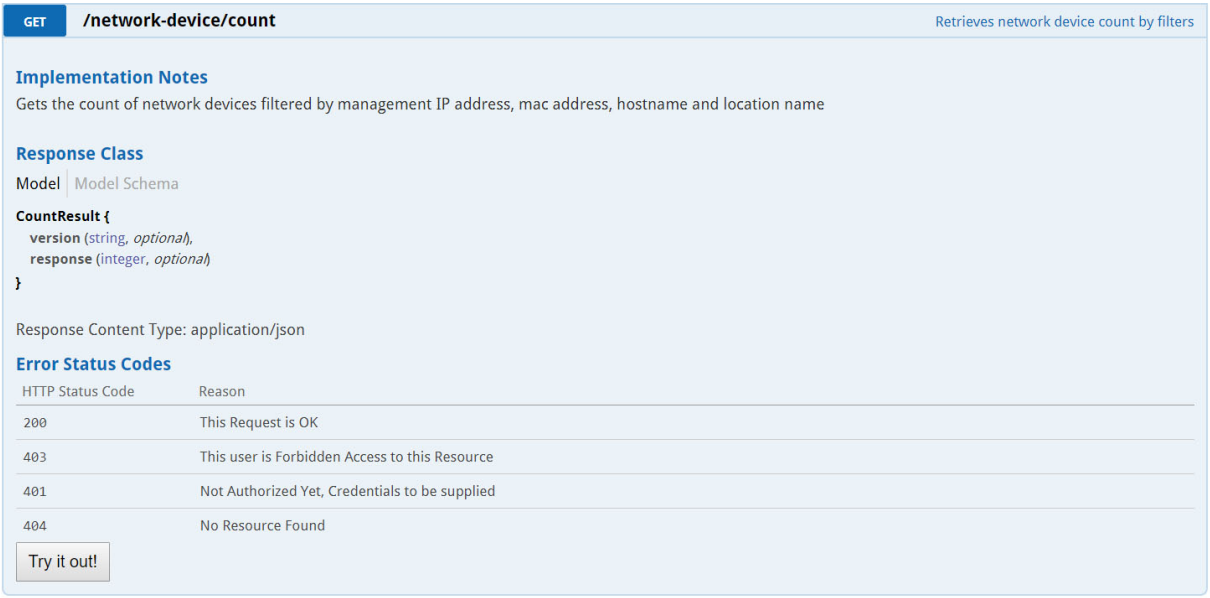
ověření prostřednictvím uživatelského jména a hesla. Platnost tohoto tokenu je časově omezena a po jeho vypršení je třeba vygenerovat token nový. Pro následující demonstraci je využit volně přístupný APIC-EM, který slouží právě pro účely testování REST API komunikace. Jeho adresa je `https://sandboxapicem.cisco.com`, přístupové údaje jsou *devnetuser / Cisco123!*. Nyní k samotné ukázce:

```
>>> from nuaal.connections.api import ApicEmBase
>>> provider = {
...     "url": "https://sandboxapicem.cisco.com",
...     "username": "devnetuser",
...     "password": "Cisco123!"
... }
>>> apicem = ApicEmBase(**provider)
APIC-EM - INFO - Successfully updated Service Ticket: ST-4431-dWgrUhMUHznMTVHoZg1g-cas
APIC-EM - INFO - Storing credentials.
>>> # Ziskani poctu zarizeni v databazi
>>> apicem.get(path="/network-device/count")
13 # <- Pocet zarizeni
```

Ukázka 4.2: Inicialzace spojení s APIC-EM REST API

V Ukázce 4.2 nejprve importujeme příslušnou třídu `ApicEmBase`. Následně vytvoříme objekt `provider`, což je slovník (datová struktura `dict`), který obsahuje základní informace pro vytvoření spojení: adresu kontroleru, jméno uživatele a heslo pro autentizaci. Kromě těchto základních parametrů je možné přidat i další volitelné parametry, které jsou popsány v dokumentaci dané třídy (viz *Příloha 1: Dokumentace nástroje NUAAL*). V dalším kroku vytvoříme instanci objektu `ApicEmBase`, kterému předáme objekt `provider` jako seznam argumentů (pomocí operátoru `**`). Vzniklou instanci označíme například jako `apicem`. Během inicializace se objekt pokusí připojit ke kontroleru na zadané adrese s příslušnými autentizačními údaji. Ověření uživatele je u APIC-EM realizováno voláním HTTP metody *POST*, v jejímž těle je obsaženo uživatelské jméno a heslo. Pokud připojení a autentizace proběhne v pořádku, zobrazí se informační hláška oznamující úspěšné získání pověřovacího tokenu. Ve výchozím stavu jsou přístupové údaje ukládány do zvláštního souboru, a to včetně autorizačního tokenu. Díky tomu lze objekt inicializovat bez nutnosti uvádění přístupových údajů přímo ve zdrojovém kódu, a zároveň je možné opakovaně pracovat se stejným tokenem po celou dobu jeho platnosti, bez nutnosti generovat při každé inicializaci nový. Po úspěšně vytvořeném spojení lze pokračovat zadáváním konkrétních dotazů, v případě ukázky se jedná o jednoduchý dotaz na počet zařízení obsažených v databázi kontroleru, pomocí metody *GET* na cestu `/network-device/count`. Veškeré dostupné funkce REST API u APIC-EM jsou popsány v on-line dokumentaci, která je přímo součástí webového rozhraní kontroleru. Dokumentace je rozdělená do jednotlivých sekcí, kde jsou popsány konkrétní funkce, jejich parametry a datová struktura odpovědi. Na obrázku 4.2 je ukázán příklad pro funkci `/network-device/count`. Kromě základního popisu dané funkce je zde popsána i struktura odpovědi, v případě této funkce se jedná pouze o celočíselnou hodnotu (*integer*) reprezentující počet zařízení. On-line dokumentace API rozhraní zároveň umožňuje přímé testování zvolené funkce přímo uvnitř webového prostředí. Ačkoliv většina funkcí vyžaduje pouze URL cestu k požadovaným datům, některé funkce umožňují de-

finovat vyhledávací atributy pomocí parametrů HTTP dotazu. Díky tomu lze značně omezit rozsah získaných dat pouze na relevantní informace. Bohužel, zdaleka ne všechny příkazy tento způsob filtrování podporují a je proto potřeba samotnou filtraci provádět až na straně klienta. Za tímto účelem obsahuje knihovna NUAAL objekt `Filter`, který bude podrobněji popsán v sekci *Zpracování formátovaných dat* 4.3.2.




Obrázek 4.2: Ukázka on-line dokumentace APIC-EM REST API

Většina funkcí REST API jednoduše poskytuje data obsažená v periodicky aktualizované databázi APIC-EM. Získání takovýchto informací je proto relativně rychlé a lze jej zpracovat v rámci jednoho HTTP dotazu. Některé funkce či aplikace APIC-EM však vyžadují přístup k samotným spravovaným zařízením, nebo spouštějí časově náročnější operace nad databázovými daty. V takových případech nelze odpověď odeslat ihned, neboť je třeba počkat na dokončení potřebných úkonů. Z tohoto důvodu umožňuje API kontrolovat průběh spuštěných úloh. Jednotlivé úlohy jsou reprezentovány *tasky*, na jejichž aktuální stav se lze dotazovat pomocí jejich identifikátoru. V případě spuštění časově náročnějšího příkazu proto odpověď obsahuje právě identifikátor daného úkolu. Lze tak periodicky kontrolovat stav pomocí metody *GET* na adresu `/task/<identifikátor>`. Za tímto účelem obsahuje objekt `ApicEmBase` funkci `task_handler`, která na základě identifikátoru periodicky kontroluje stav daného úkolu a následně poskytne identifikátor, kde lze získat výstup požadovaného příkazu.

### 4.2.2 Programový přístup k CLI

Vzhledem k jednomu z primárních cílů této práce, kterým je rozšířit některé funkce kontroleru APIC-EM i na zařízeních, která nejsou tímto kontrolerem podporována, je nezbytné do určité míry zreplikovat činnost samotného kontroleru. Ačkoliv rozhraní příkazové řádky není

primárně navrženo pro použití se softwarovými nástroji, lze jej použitím vhodných postupů použít i k tomuto účelu. Primárním úkolem je samozřejmě vytvoření samotného spojení, či datového kanálu, jehož prostřednictvím lze data ze zařízení získávat. Samotný přenos dat realizují protokoly SSH a Telnet, oba široce rozšířené a podrobně zdokumentované. Softwarová implementace protokolu SSH je však vzhledem k šifrování mnohem náročnější, než je tomu u nešifrovaného Telnetu. Pro Python existuje řada knihoven realizujících SSH spojení, avšak nejrozšířenější je knihovna *paramiko*. Tu využívá mimo jiné i dříve zmíněný nástroj Ansible, lze proto usuzovat, že se jedná o kvalitní a pravidelně udržovanou knihovnu. Kryptografické funkce jsou realizovány pomocí samostatné knihovny *pycrypto*. Ačkoliv *paramiko* značně usnadňuje vytváření SSH spojení, v souvislosti se síťovými prvky je zde řada dalších problémů. Jedním z nich jsou například rozměry virtuálního terminálu, které souvisejí se stránkováním textovévo výstupu. Každá relace SSH má na začátku nastaveny výchozí hodnoty šířky a délky virtuálního terminálu (VTY), udávající počet znaků na řádek a počet řádků, který lze zobrazit. Pokud tedy počet řádků textového výstupu přesáhne tento limit, je většinou potřeba interakce ze strany uživatele, která umožní výpis dalšího obsahu. Takovýto postup je sice žádoucí v případě lidské interakce, avšak představuje problém při softwarem realizované komunikaci. Většina zařízení naštěstí umožňuje tyto limitní hodnoty v rámci dané relace upravit, například u zařízení Cisco následujícím způsobem:

```
1 Switch>terminal width 0
2 Switch>terminal length 0
```

Ukázka 4.3: Úprava rozměrů virtuálního terminálu

Tyto dva příkazy je potřeba zadat na začátku každé vytvořené relace. U jiných výrobců se však samozřejmě potřebné příkazy liší. Dalším specifikem příkazových řádek síťových prvků jsou také dvě úrovně přístupu, kdy jedna slouží především ke zobrazování údajů a druhá umožňuje provádění konfiguračních změn. V terminologii síťových prvků Cisco se jedná o *User EXEC Mode* a *Privilege EXEC Mode*. Pro přepínání mezi těmito dvěma úrovněmi slouží příkazy `enable` a `disable`. Pro přístup do privilegovaného režimu je navíc třeba zadat speciální přístupové heslo (*enable secret*). Obdobný postup nalezneme i u zařízení jiných výrobců.

K překlenutí těchto specifických požadavků jednotlivých zařízení využívá nástroj NUAAL knihovnu *Netmiko* [9]. Její název napovídá, že se jedná o jakousi nadstavbu nad široce rozšířenou knihovnou *paramiko*, která se zaměřuje právě na řešení specifických potřeb komunikace se síťovými prvky. Autorem této knihovny je *Kirk Byers*, který se věnuje nejrůznějším odvětvím automatizace v oblasti datových sítí. Mimo jiné je emeritním držitelem jedné z nejvyšších certifikací v oblasti routing a switching na zařízeních Cisco (Cisco CCIE Routing&Switching), není proto překvapením, že většina jeho práce se zaměřuje právě na tato zařízení. Knihovna *Netmiko* si však klade za cíl být co možná nejvíce multiplatformí, seznam podporovaných a pravidelně testovaných operačních systémů proto obsahuje:

- Arista vEOS
- Cisco ASA
- Cisco IOS
- Cisco IOS-XE
- Cisco IOS-XR
- Cisco NX-OS
- Cisco SG300
- Dell OS10
- HP Comware7
- HP ProCurve
- Juniper Junos
- Linux

Kromě toho je částečně nebo experimentálně podporováno více než třicet nejrůznějších operačních systémů od výrobců jako Alcatel, Huawei, Avaya, Brocade, Fortinet, Dell, Extreme či Enterasys. Kromě samotného autora se na dalším vývoji a údržbě této knihovny podílí několik desítek dalších přispěvatelů, díky čemuž je stále velmi pravidelně aktualizována, jak je vidět na webovém repozitáři *GitHub* [10].

Právě z důvodu usnadnění komunikace se síťovými prvky využívá nástroj NUAAL *Netmiko*. Stejně jako v případě spojení přes REST API i zde existuje v knihovně NUAAL rodičovský objekt poskytující společné funkce všem dalším objektům, které realizují spojení s různými síťovými prvky prostřednictvím CLI. Díky této třídě, nesoucí název `CliBase`, lze poměrně snadno vytvářet další třídy poskytujícími specifické funkce pro dané typy zařízení. Třída `CliBase` do značné míry kopíruje logiku objektů z knihovny *Netmiko*, avšak přidává některé specifické vlastnosti a funkce pro potřeby nástroje NUAAL. Jednou z takovýchto vlastností je například automatická volba protokolu realizujícího komunikaci. Ačkoliv je z bezpečnostních důvodů preferován protokol SSH, v některých organizacích se primárně používá protokol Telnet. Navíc se stále můžeme setkat i se staršími zařízeními, která protokol SSH nepodporují.

Ke komunikaci se zařízeními s operačním systémem IOS (případně IOS XE) slouží v knihovně NUAAL třída `Cisco_IOS_Cli`. Veškeré parametry této třídy se snaží dodržovat konvence nastavené knihovnou *Netmiko* tak, aby byla práce s oběma knihovnami co možná nejvíce uživatelsky přívětivá. Následující Ukázka 4.4 demonstruje základní příklad vytvoření komunikačního kanálu prostřednictvím protokolu SSH, odeslaní diagnostického příkazu a přijmutí korespondujícího textového výstupu:

```
>>> from nuaal.connections.cli import Cisco_IOS_Cli
>>> provider = {
...     "ip": "10.19.51.142",
...     "username": "admin",
...     "password": "cisco"
... }
>>> with Cisco_IOS_Cli(**provider) as device:
...     response = device._send_command("show clock")
...
ParserModule-cisco_ios - INFO - Creating ParserModule Object for cisco_ios
Connection-10.19.51.142 - INFO - Connected to '10.19.51.142' using 'cisco_ios'.
Connection-10.19.51.142 - INFO - Successfully disconnected from device 10.19.51.142
>>> print(response)
*18:38:53.373 UTC Mon Apr 29 2018
>>>
```

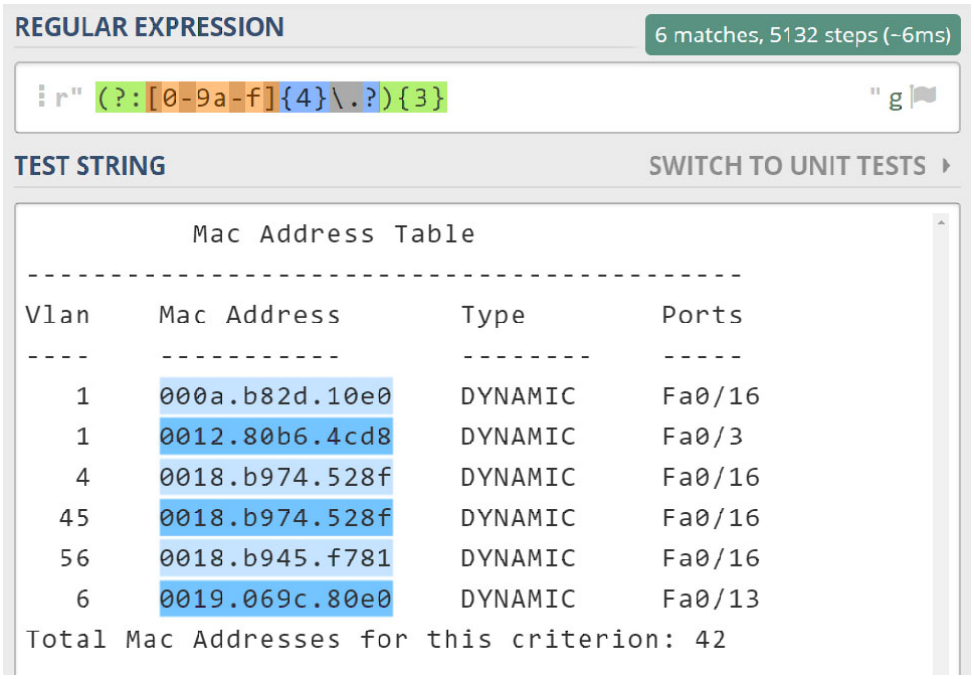
Ukázka 4.4: Inicialzace spojení se zařízením prostřednictvím CLI

První řádek z ukázky importuje potřebný objekt `Cisco_IOS_Cli`. Následuje vytvoření slovníku obsahujícího nezbytné údaje pro vytvoření spojení - IP adresu zařízení, jméno uživatele a jeho heslo. Na sedmém řádku následuje inicializace objektu s příslušnými parametry prostřednictvím kontextového manageru. Kontextový manager zajišťuje vytvoření objektu pouze po dobu práce s tímto objektem. V tomto případě mimo jiné zajišťuje, že po provedení příkazů v daném bloku dojde ke korektnímu ukončení SSH relace. Použití kontextového manageru není podmínkou, avšak značně usnadňuje práci jak v případě použití interaktivního režimu, tak v případě "jednorázového"přístupu k zařízení. Osmý řádek reprezentuje jedinou operaci se vzniklým objektem, v tomto případě funkci `_send_command("show clock")`. Tato funkce odešle zadaný příkaz do zařízení a vrátí textový výstup ze zařízení. Ten je pro budoucí použití uložen do proměnné `response`. Následují tři logovací hlášky informující o úspěšném navázání spojení a jeho následném ukončení. Pro zobrazení podrobnějších hlášek je možné při inicializaci objektu přidat parametr `DEBUG=True`, díky čemuž se zobrazí informace o všech provedených úkonech. Pro zachování minimalistického výstupu je tento parametr ve výchozím stavu nastaven na hodnotu `False`. Nakonec následuje příkaz pro vypsání obsahu proměnné `response`, v tomto případě čas a datum nastavený na zařízení.

Samotné vyčítání textových výstupů v nestrukturované formě sice může v některých ojedinělých případech ušetřit čas síťového inženýra, avšak samo o sobě neumožňuje další programové zpracování dat. Za tímto účelem je třeba z neformátovaného textu nejdříve získat relevantní údaje, které lze uložit v některém ze strukturovaných formátů. Této problematice se věnuje následující sekce.

### 4.2.3 Zpracování textových výstupů

Jedním z pilířů knihovny NUAAL je modul `Parsers`, který obsahuje třídy pro převedení neformátovaných textových řetězců do formátu JSON. K parsování textu využívá knihovna NUAAL regulární výrazy, se kterými umožňuje pracovat standardní modul Pythonu - knihovna `re`. Práce s regulárními výrazy představuje jednu z nejpoužívanějších metod pro zpracování textových dat, a to ať už strukturovaných či nikoliv. Regulární výraz je reprezentován speciálním textovým řetězcem (*regex stringem*), který umožňuje používat zástupné znaky. Tyto řetězce je poté možné použít k prohledávání textu. Například regulární řetězec `\d` vyhledá v textu všechny číslice `0-9`, `[a-z]` vyhledá všechna malá písmena `a-z` a podobně. Zároveň je možné využít kvantifikátory jako `*`, `+` a `?`. Představme si situaci, kdy bychom chtěli z textového výstupu Cisco zařízení získat všechny MAC adresy obsažené v přepínací tabulce:




Obrázek 4.3: Hledání MAC adres pomocí regulárního výrazu

Tato ukázka pochází z webové aplikace `regex101.com`, která umožňuje testovat regulární výrazy v grafickém prostředí. Na obrázku je v části *Test String* textový výstup příkazu `show mac address-table`. V poli *Regular Expression* je samotný regulární řetězec, pomocí něhož jsou MAC adresy vyhledávány. Cisco zařízení používají pro MAC adresy formát skládající se ze tří skupin po čtyřech hexadecimálních znacích. Tyto skupiny jsou odděleny tečnou. V regulárním výrazu takovouto skupinu reprezentuje výraz `[0-9a-f]{4}`, tedy právě čtyři znaky, které jsou buď číslice, nebo písmena od `a` do `f`. Po této skupině následuje tečka (`.`), která má však v rámci regulárních výrazů speciální význam - zastupuje právě jeden libovolný znak. Pro omezení hledání pouze a konkrétně na tečku je třeba použít výraz `\.`. Po poslední skupině však již tečka nenásleduje, což lze ošetřit použitím kvantifikátoru `?`, který udává, že jemu předcházející výraz se v textu musí vyskytovat maximálně jednou, případně se nemusí vyskytovat vůbec. Následuje už jen kvatifikátor {3}, udávající, že se celá předcházející skupina (ohraničená zeleně podbarvenými závorkami) musí opakovat právě třikrát. Stejnou operaci jako demonstruje ukázka ve webovém nástroji lze v Pythonu realizovat následujícím příkazem:

```
>>> import re
>>> mac_addresses = re.findall(pattern=r"(?:[0-9a-f]{4}\.?){3}", string=text_output)
>>> print(mac_addresses)
['000a.b82d.10e0', '0012.80b6.4cd8', '0018.b974.528f',
'0018.b974.528f', '0018.b945.f781', '0019.069c.80e0']
```

Ukázka 4.5: Hledání MAC adres pomocí regulárního výrazu v Pythonu

Na prvním řádku je naimportován potřebný modul `re`, ze kterého je použita funkce `findall()`. Ta vrací seznam všech řetězců, které odpovídají zadanému vzoru (`pattern`). Samotný textový obsah je v tomto případě uložen v proměnné `text_output`. Výsledky hledání jsou uloženy do proměnné `mac_addresses` a následně vypsány do terminálu.

Výše popsaná ukázka je sice dostatečně jednoduchá pro demonstraci základního využití regulárních výrazů, avšak jako taková neposkytuje příliš užitečné údaje. Abychom tuto ukázku upravili do přínosnější podoby, bylo by potřeba kromě samotné MAC adresy uchovat také údaje o příslušné VLAN síti, fyzickém rozhraní za kterým se tato adresa nachází a případně typ, udávající zda se jedná o dynamicky naučenou či staticky zadanou adresu. K tomu už je zřejmě potřeba poněkud komplexnějšího regulárního výrazu. Při práci s takto komplexními výrazy je vhodné využít další funkce, jako je možnost definování pojmenovaných skupin. Každé skupině ohraničené kulatými závorkami lze přiřadit popis, který udává co daná skupina obsahuje. Tyto popisy lze následně využít jako identifikátory hodnot v datové struktuře slovníku (*dictionary*), který je tvořen právě páry *klíč:hodnota*. V takovém případě může strukturovaná podoba téhož textového výstupu vypadat následovně:

```
[
{'vlan': 1, 'mac': '000a.b82d.10e0', 'type': 'DYNAMIC', 'ports': 'Fa0/16'},
{'vlan': 1, 'mac': '0012.80b6.4cd8', 'type': 'DYNAMIC', 'ports': 'Fa0/3'},
{'vlan': 4, 'mac': '0018.b974.528f', 'type': 'DYNAMIC', 'ports': 'Fa0/16'},
{'vlan': 45, 'mac': '0018.b974.528f', 'type': 'DYNAMIC', 'ports': 'Fa0/16'},
{'vlan': 56, 'mac': '0018.b945.f781', 'type': 'DYNAMIC', 'ports': 'Fa0/16'},
{'vlan': 6, 'mac': '0019.069c.80e0', 'type': 'DYNAMIC', 'ports': 'Fa0/13'}
]
```

Ukázka 4.6: Reprezentace výstupu `show mac address-table` ve formátu JSON

Jedná se o datovou strukturu seznamu (`list`), jejímiž prvky jsou slovníky (`dict`) se stejnými klíči (identifikátory) *vlan*, *mac*, *type* a *ports*. Stejnou strukturu, tedy slovníky jako prvky seznamu, využívá právě i REST API kontroleru APIC-EM.

Pro textový výstup z předchozí ukázky je možné sestavit poměrně jednoduchý regulární výraz, neboť výstup sám obsahuje pouze několik málo hodnot obsažených v přehledné textové tabulce. U komplexnějších výstupů však brzy nastává vícero problémů. Jedním z nich je, že s rostoucí "složitostí"textového výstupu se i adekvátně zvyšuje komplexnost samotného regulárního výrazu, který se tak stává náchylnější k chybám. Druhým a zásadnějším problémem je, že i drobná změna textového výstupu může způsobit, že přestane odpovídat vzoru předepsanému regexovým řetězcem. Tento problém lze vyřešit v základě dvěma způsoby: V prvním případě je možné vytvořit nový vzor, který bude souhlasit s daným, byť jen lehce pozměněným textem. V takovém případě se ale velké části regulárních výrazů budou opakovat, čímž se celý systém stává mnohem hůře udržovatelným. Druhou možností je ošetřit "problematické"sekce výrazu pomocí kvantifikátorů. Tím se zajistí, že i absence některé očekávané části textu nezpůsobí nefunkčnost celého výrazu, ovšem za cenu nesrovnatelně vyšší složitosti potřebného předpisu.

Existují samozřejmě knihovny, které se snaží práci s regulárními výrazy zjednodušit. Jednou z takových knihoven je například *TextFSM* [11], kterou vyvinula pro své interní potřeby společnost Google. Knihovna byla následně uvolněna i pro použití širší veřejnosti. Základním principem této knihovny je možnost přiřazovat regulární výrazy do speciálních proměnných, ze kterých se poté sestavuje celý vzorový řetězec. Kupříkladu tak lze přiřadit regulární řetězec z předchozí ukázky do proměnné `%MACADDR`, a následně už vždy používat pouze referenci na tuto proměnou.

Díky tomu je výsledný výraz mnohem jednodušší a přehlednější, než v případě čistě regexového řetězce. Díky využití této knihovny je zároveň i zdrojový kód samotné aplikace jednodušší, neboť většina práce je realizována právě v rámci *TextFSM*. Při testování této knihovny od Googlu pro potřeby nástroje NUAAL se však ukázalo, že neposkytuje dostatečnou flexibilitu v případě komplexnějších textových výstupů. *TextFSM* byla totiž primárně navržena na zpracování textových tabulek, kde opravdu práci značně usnadňuje. Bohužel v případě síťových prvků nejsou zdaleka všechny výstupy reprezentovány tabulkami a bylo proto nezbytné zvolit alternativní přístup. Knihovna NUAAL proto využívá kompromisní řešení mezi složitostí regulárních výrazu a samotného zdrojového kódu, který s pomocí těchto výrazů text zpracovává. Jádro samotného parsování tvoří v knihovně NUAAL dvě třídy:

- *PatternsLib*: Obsahuje potřebné regulární výrazy
- *ParserModule*: Obsahuje logiku pro zpracování textu

***PatternsLib*** představuje centrální úložiště veškerých používaných regulárních výrazů. Samotné výrazy jsou uloženy ve struktuře slovníku, který je organizován na základě potřeb jednotlivých podporovaných textových výstupů. Na základě příkazu, který na zařízené generuje požadovaný textový výstup (například `show mac address-table`) lze tak snadno získat všechny potřebné regulární řetězce. Pro konkrétní účel se však nemusí jednat o jeden konkrétní řetězec, ale například o seznam možných použitelných výrazů. Jednotlivé předpisy tak mohou být značně jednodušší a věnovat se pouze specifické části textového výstupu. Díky tomu je i možné snáze reagovat na případné změny, kdy lze jednoduše přidat další předpis zpracovávající konkrétní část výstupu. Pro inicializaci této třídy je potřeba nejprve zadat identifikátor typu zařízení, jehož výstupy mají být zpracovány. Na základě této informace jsou všechny potřebné regulární výrazy nejprve zkompilovány a následně předány samotné parsovací třídě. Každý výraz je před samotným použitím nejprve potřeba zkompilovat. Tato kompilace může buď proběhnout automaticky v rámci některé z funkcí modulu `re`, nebo jako v tomto případě předem. Vzhledem k tomu, že samotná kompilace regulárního výrazu z podoby textového řetězce do podoby instance objektu `pattern` trvá ne zcela zanedbatelnou dobu, je jistě efektivnější předávat výrazy již v podobě vhodné k okamžitému použití.

***ParserModule*** obsahuje veškeré funkce tvořící celkovou logiku parsovacího jádra knihovny NUAAL. Jak již bylo mnohokrát zmíněno, komplexnost textových výstupů se pro různé příkazy značně liší. Jako příklad lze uvést příkaz `show interfaces`. Tento příkaz zobrazí poměrně značné množství nejrůznějších údajů o všech datových rozhraních daného zařízení. Délka textového výstupu pro jedno rozhraní se tradičně pohybuje v rozmezí dvaceti až třiceti řádek v závislosti na typu rozhraní. Představíme-li si plně obsazený modulární přepínač s 10 sloty pro datové karty, přičemž každá karta bude obsahovat 48 rozhraní, dostáváme se přibližně na $10 \cdot 48 \cdot 30 = 14400$ řádek, které je nutné zpracovat. Pracovat v takovémto rozsahu s jediným regulárním výrazem, který by představoval jedno rozhraní, by bylo velmi obtížné. Výraz by sám musel být poměrně složitý, aby dokázal postihnout všechny možné variace výstupu u různých typů rozhraní. Zpracování v této třídě proto probíhá ve více krocích či úrovních. Relativní složitost výstupu konkrétního příkazu je udána jeho umístěním ve třídě `PatternsLib`. Příkazy

s jednoduššími výstupy, jako `show mac address-table`, `show ip arp` nebo `show inventory` jsou obsaženy pouze v rámci jedné úrovně - tedy že je zle zpracovat v rámci jednoho kroku. U složitějších výstupů, jako právě například `show interfaces` je potřeba textový výstup v prvním kroku nejdříve vhodně připravit pro další zpracování. V tomto případě první krok zajistí, že textový výstup obsahující oněch teoretických 14400 řádků bude rozdělen do 480 textových řetězců, každý obsahující informace o jednom rozhraní. Tyto řetězce jsou dále předány následujícímu kroku. V tomto kroku se použijí jiné regulární vzory než v předchozím kroku a z jejich pomocí se postupně prohledá všech 480 textových řetězců. Výsledným výstupem bude opět seznam, obsahující slovníky reprezentující jednotlivá rozhraní. Klíče těchto slovníků jsou převzaty z názvů pojmenovaných skupin uvnitř regulárních výrazů. Tento postup demonstruje následující ukázka:

```
from nuaal.Parsers import CiscoIOSParser

# Prikaz pouzity k ziskani vystupu
command = "show etherchannel summary"
# Textovy vypis ulozeny v promenne text
text = """Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------------------------
2      Po2(SU)         LACP      Gi2/14(P)   Gi2/24(P)
4      Po4(SU)         LACP      Gi2/8(P)    Gi2/43(P)
"""
# Inicializace parsovaci tridy
parser = CiscoIOSParser()
# Vystup 1. kroku
l0_out = parser._level_zero(text=text, patterns=parser.patterns["level0"][command])
print(l0_out)
[{
"group": 2,
"portchannel": "Po2",
"status": "SU",
"protocol": "LACP",
"ports": "Gi2/14(P)   Gi2/24(P)\n"
},
{
"group": 4,
"portchannel": "Po4",
"status": "SU",
"protocol": "LACP",
"ports": "Gi2/8(P)    Gi2/43(P)\n"
}]
```

Ukázka 4.7: Parsování `show etherchannel summary` - 1. krok

Vidíme, že už první krok poskytuje strukturovaný výstup. Zaměříme-li se však na hodnoty klíče `ports`, zjistíme, že se stále jedná o textový řetězec obsahující všechna fyzická rozhraní příslušící do virtuálního rozhraní *portchannel*. Abychom rozdělili i tento řetězec, je třeba dalšího kroku. Tento postup je v rámci knihovny zautomatizován a uživatel tedy nepotřebuje žádnou hlubší znalost knihovny. Pro získání plně strukturovaného výstupu lze jednoduše použít funkci `autoparse()`, která předá přímo finální podobu dat.

```
output = parser.autoparse(text=text, command="show etherchannel summary")
print(output[0])
{
 "group": 2,
 "portchannel": "Po2",
 "status": "SU",
 "protocol": "LACP",
 "ports": [
  {
  "port": "Gi2/14",
  "status": "P"
  },
  {
  "port": "Gi2/24",
  "status": "P"
  }
 ]
}
```

Ukázka 4.8: Parsování `show etherchannel summary`

Nyní jsou již jednotlivá fyzická rozhraní reprezentována samostatnými slovníky. Parsovací modul knihovny NUAAL je možné používat zcela samostatně, například pro zpracování dříve získaných výstupů uložených v textových souborech. Primárním cílem tohoto modulu je však integrace do objektů, které realizují také samotnou komunikaci se zařízeními. Instance parsovacího modulu je proto vždy součástí třídy komunikující se zařízením, jako například `Cisco_IOS_Cli`. Díky tomu je možné jednoduše získávat ze zařízení přímo strukturovaná data. Tomuto postupu se věnuje následující část.

### 4.2.4 Komunikace se zařízeními

Tato část je průnikem kapitol 4.2.2 a 4.2.3 a popisuje celkový způsob, jakým lze získávat strukturovaná data ze síťových prvků prostřednictvím příkazové řádky. Jak bylo uvedeno v předešlé kapitole, komunikace se zařízením je řízena prostřednictvím objektů, jako je například `Cisco_IOS_Cli`. Při inicializaci této třídy je kromě vytvoření samotného spojení se zařízením vytvořena také instance příslušného parsovacího modulu, v tomto případě `CiscoIOSParser`. Alternativně může být použita již existující instance potřebného modulu, kterou mohou jednotlivá spojení sdílet. Objekt realizující spojení nabízí funkce, které poskytují již přímo strukturovaná data ze zařízení. Názvy těchto funkcí začínají předponou `get_`, jako například `get_vlans()`, `get_interfaces()`, `get_arp()` a podobně. Samotné názvy funkcí napovídají, jaké údaje příslušná funkce poskytuje. Data získaná prostřednictvím jednotlivých funkcí jsou zároveň ukládána do paměti příslušného objektu, díky čemuž lze snáze získat jednotný přístup ke všem dostupným datům. Ukázka 4.9 demonstruje použití funkce `get_version()`:

```
>>> from nuaal.connections.cli import Cisco_IOS_Cli
>>> provider = {
... "username": "admin",
... "password": "cisco",
... "ip": "10.19.51.142"
... }
>>> with Cisco_IOS_Cli(**provider) as device:
...     device.get_version()
...     data = device.data
...
ParserModule-cisco_ios - INFO - Creating ParserModule Object for cisco_ios
Connection-10.19.51.142 - INFO - Connected to '10.19.51.142' using 'cisco_ios'.
Connection-10.19.51.142 - INFO - Successfully disconnected from device 10.19.51.142
>>> print(data)
{
 "ipAddress": "10.19.51.142",
 "hostname": "C9300-1",
 "version": [
  {
  "vendor": "Cisco",
  "software": "IOS XE",
  "version": "16.06.02",
  "platform": "C9300-24T",
  "systemMemory": "869104K/6147K",
  "hostname": "C9300-1",
  "uptime": "5 weeks, 6 days, 7 hours, 25 minutes",
  "imageFile": "flash:packages.conf",
  "experimental_version": null
  }
 ]
}
```

Ukázka 4.9: Získávání strukturovaných dat - get_version()

Uvedený příklad je velmi podobný, jako použití demonstrované v ukázce 4.4. Obdobným způsobem lze velmi rychle získat informace například o datových rozhraních, nainstalovaných modulech (produktová a sériová čísla jednotlivých částí zařízení), sítích VLAN, licencích, sousedních zařízeních a podobně. S takto získanými strukturovanými daty lze již provádět libovolné operace. Lze například snadno odhalit chyby v konfiguraci VLAN sítí či *trunk* portů, či zjistit která zařízení jsou provozována se zastaralou verzí operačního systému. Pro usnadnění práce s více zařízeními současně obsahuje knihovna NUUAAL třídu `CliMultiRunner`, která umožňuje provést tutéž sadu příkazů na několika zařízeních zároveň. Jednotlivé instance komunikujících objektů jsou inicializovány uvnitř samostatných vláken, díky čemuž je možné paralelně přistupovat k většímu počtu zařízení. Jediné potřebné parametry jsou přístupové údaje k zařízením, seznam IP adres zařízení a seznam požadovaných akcí.

## 4.3 Modelová vrstva

Aby bylo možné efektivně pracovat se získanými daty, je třeba definovat společnou strukturu těchto dat, respektive množinu vlastností pro dané entity. Kupříkladu entita (objekt) reprezentující datové rozhraní, by měl obsahovat stejnou sadu vlastností (klíčů) bez ohledu na to, z jakého typu zařízení pocházejí zdrojová data. Za tímto účelem obsahuje knihovna NUAAL zvláštní typy objektů (dále označovaných jako *modely*), které slouží právě ke změně formátu

dat poskytovaných nižší vrstvou. Jako základ společného formátu je v nástroji NUAAL zvolen právě formát používaný kontrolerem APIC-EM. V některých ohledech však bylo třeba tento formát lehce upravit tak, aby umožňoval přenášet i takové informace, které neposkytuje databáze APIC-EM. Příkladem může být údaj o VLAN sítích, které jsou přenášeny na *trunk* rozhraních. Informace o tom, které VLAN sítě (respektive které 802.1Q tagy) mohou být na rozhraní přenášeny, je poměrně důležitá, avšak APIC-EM poskytuje pouze číslo takzvané nativní VLAN sítě. Proto je výsledný formát v nástroji NUAAL upraven tak, aby umožňoval ukládat informace o všech VLAN sítích na daném rozhraní.

Kromě prosté změny formátu výstupních dat plní tyto modely další důležitou funkci, kterou je spojování údajů z více zdrojů. V některých případech není možné získat kompletní sadu informací prostřednictvím jediného diagnostického příkazu. Chtěli bychom například získat kompletní informace o VLAN sítích na zařízení, pravděpodobně nejdříve sáhneme po příkazu `show vlan brief`. Takto získáme seznam všech virtuálních sítích aktivovaných na daném zařízení spolu se seznamem fyzických rozhraní, která do každé ze sítí patří. Tento výpis však zobrazuje pouze rozhraní, která jsou v takzvaném *access* módu. Abychom zjistili, přes která *trunk* rozhraní mohou být rámce dané VLAN sítě přepínány, potřebujeme ještě alespoň příkaz `show interfaces trunk`. Na základě těchto dvou výpisů je funkce `get_vlans()` schopna poskytnou kompletní informace o virtuálních sítích a jejich přidruženým rozhraním. Toto použití demonstruje následující ukázka 4.10.

```
>>> from nuaal.connections.cli.GetCliHandler import GetCliHandler
>>> from nuaal.Models.Cisco_IOS_Model import CiscoIOSModel
>>> provider = {"username": "admin", "password": "cisco"}
>>> connection = GetCliHandler(device_type="cisco_ios",
                               ip="192.168.100.181",
                               **provider)
ParserModule-cisco_ios - INFO - Creating ParserModule Object for cisco_ios
>>> model = CiscoIOSModel(connection)
>>> model.get_vlans()[751]
{
 "vlanId": 751,
 "name": "TESTING-VLAN",
 "status": "active",
 "untaggedPorts": [
  "GigabitEthernet0/25",
  "GigabitEthernet0/26",
  "GigabitEthernet0/27"
 ],
 "taggedPorts": [
  "Port-channel30"
 ]
}
```

Ukázka 4.10: Použití objektu Cisco_IOS_Model

Po naimportování příslušných tříd a vytvoření slovníku `provider` s autentizačními údaji je zavolána funkce `GetCliHandler`, která na základě parametru `device_type` poskytne příslušný objekt pro vytvoření spojení. V toto případě se opět jedná o Cisco zařízení, funkce proto vrátí instanci objektu `Cisco_IOS_Cli`. Reference této vzniklé instance (uložená v proměnné `connection`) je předána objektu CiscoIOSModel, který ji používá pro komunikaci se zařízením.

Na závěr pomocí příkazu `model.get_vlans()[751]` získáme JSON reprezentaci VLAN sítě 751. Kromě jejího jména jsou zde obsaženy i klíče `untaggedPorts` a `taggedPorts`, které představují rozhraní v módu *access* a *trunk*. Vzhledem k tomu, že každý objekt realizující spojení může teoreticky poskytovat odlišně formátované datové struktury, musí ke každému takovému objektu existovat i příslušný model, který převede strukturu do jednotného, předem definovaného formátu. Ačkoliv se tento postup může zdát neefektivním, při tvorbě nástrojr NUAAL se ukázalo, že je tato metoda značně rychlejší, než snaha sjednotit formát už na úrovni regulárních výrazů.

### 4.3.1 Síťová topologie

Kromě samotné práce s jednotlivými objekty (zařízeními) jsou v rámci knihovny NUAAL zahrnuty i třídy, které umožňují práci s celými sadami zařízení, která mohou tvořit celou danou síť. Do této skupiny patří například objekty z modulu *Discovery*, které umožňují prohledávání celé sítě. Prvním z těchto objektů je `IP_Discovery`, který umožňuje sestavit reprezentaci fyzické topologie sítě na základě seznamu adres zařízení. Z těchto zařízení jsou získány informace o jejich sousedech pomocí funkce `get_neighbors()`, která využívá data z protokolů *CDP* či *LLDP*. Z těchto informací je následně sestavena samotná topologie.

Druhá třída z modulu *Discovery*, `Neighbor_Discovery`, se snaží řešit situaci, kdy není dostupný seznam všech zařízení. Pro použití této metody je třeba znát pouze adresu jednoho zařízení, které je použito jako základ topologie. Díky protokolu CDP je možné získat IP adresy sousedních zařízení, ke kterým se nástroj následně také připojí. Tento postup je cyklicky opakován, dokud nejsou navštívena všechna dostupná zařízení. Výsledkem je poté seznam všech nalezených zařízení spolu s reprezentací fyzické topologie sítě. Možný problém při tomto postupu však představuje samotné chování protokolu CDP. Ten totiž ve výchozím nastavení poskytuje buď IP adresu virtuálního rozhraní *Loopback0*, nebo IP adresu s nejnižší číselnou hodnotou. Může se tak stát, že objevená IP adresa zařízení nebude pro nástroj NUAAL dostupná, čímž bude znemožněno i nalezení dalších zařízení. Tato nevýhoda by se však měla v případě dobře navržených sítí projevovat minimálně.

Samotné sestavení topologie v nástroji NUAAL realizuje samostatná třída `Topology`. Ta na základě informací o sousednosti zařízení sestaví reprezentaci fyzické topologie. V budoucnu by měla tato třída umožňovat také generování logické topologie, ať už na druhé vrstvě OSI modelu (VLAN), či třetí (virtuální routery, VRF).

```
>>> from nuaal.Discovery import Neighbor_Discovery
>>> provider = {"username": "admin", "password": "cisco"}
>>> disc = Neighbor_Discovery(provider)
>>> disc.run("192.168.100.181")
NeighDisc - INFO - Discovering device (192.168.100.181)
NeighDisc - INFO - Processing 2 neighbor(s) of device SW1
NeighDisc - INFO - Discovered new neighbor SW5 of device SW1.
NeighDisc - INFO - Discovered new neighbor SW2 of device SW1.
NeighDisc - INFO - All outputs of current round have been processed.
NeighDisc - INFO - Discovering device SW5
NeighDisc - INFO - Discovering device SW2
NeighDisc - INFO - Processing 2 neighbor(s) of device SW5
NeighDisc - INFO - Discovered new neighbor SW4 of device SW5.
NeighDisc - INFO - Neighbor SW1 of device SW5 has already been visited.
NeighDisc - INFO - Processing 3 neighbor(s) of device SW2
NeighDisc - INFO - Neighbor SW4 of device SW2 has already been discovered, waiting to
    be visited.
NeighDisc - INFO - Discovered new neighbor SW3 of device SW2.
NeighDisc - INFO - Neighbor SW1 of device SW2 has already been visited.
NeighDisc - INFO - All outputs of current round have been processed.
NeighDisc - INFO - Discovering device SW4
NeighDisc - INFO - Discovering device SW3
NeighDisc - INFO - Processing 3 neighbor(s) of device SW4
NeighDisc - INFO - Neighbor SW5 of device SW4 has already been visited.
NeighDisc - INFO - Neighbor SW2 of device SW4 has already been visited.
NeighDisc - INFO - Neighbor SW3 of device SW4 has already been discovered, waiting to
    be visited.
NeighDisc - INFO - Processing 2 neighbor(s) of device SW3
NeighDisc - INFO - Neighbor SW4 of device SW3 has already been visited.
NeighDisc - INFO - Neighbor SW2 of device SW3 has already been visited.
NeighDisc - INFO - All outputs of current round have been processed.
Topology - INFO - Building topology based on 5 visited devices.
Topology - INFO - Discovered total of 5 nodes and 6 links
```

Ukázka 4.11: Použití objektu Neighbor_Discovery

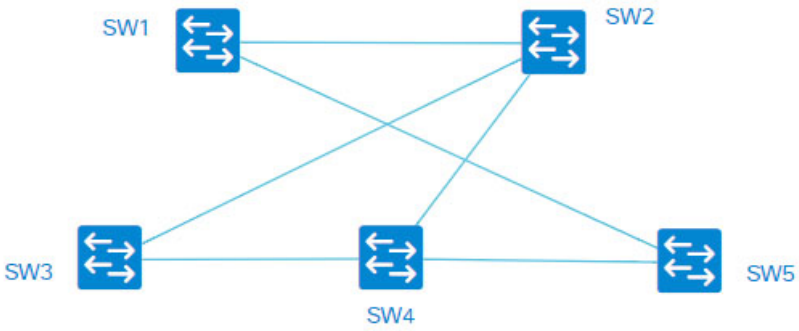
Ukázka 4.11 demonstruje použití třídy `Neighbor_Discovery` pro automatické nalezení všech dostupných zařízení v síti. Inicializačním parametrem třídy je již dobře známý objekt `provider`, který obsahuje přihlašovací údaje k zařízením. Po inicializaci objektu je spuštěna funkce `run`, jejímž parametrem je IP adresa prvotního zařízení. Z následujících informačních hlášení je dobře patrný postup celého procesu. Poslední zpráva informuje o celkovém počtu nalezených zařízení a spojení mezi nimi.

Třída `Topology` je navržena tak, aby umožňovala co nejsnazší spolupráci s nástrojem *NextUI*. Tento nástroj slouží pro vykreslování síťových topologií v rámci webových aplikací. Tato knihovna je součástí projektu *OpenDaylight*, tedy již dříve zmíněného SDN kontroleru. Na vývoji samotné knihovny *NextUI* se však značnou měrou podílela právě společnost Cisco, díky čemuž je *NextUI* také součástí APIC-EM či DNAC. Samotná knihovna je volně dostupná na portálu GitHub [12]. Samotné údaje o topologii jsou v rámci *NextUI* používány ve formátu JSON. Pro získání vhodně formátovaných dat z předešlé ukázky 4.11 lze postupovat následovně.

```
>>> from nuaal.Discovery.Topology import CliTopology
>>> topo = CliTopology()
>>> topo.build_topology(data=disc.data)
Topology - INFO - Building topology based on 5 visited devices.
Topology - INFO - Discovered total of 5 nodes and 6 links
>>> topo.next_ui()
{
 "nodes": [
  {"id": 0, "name": "SW1"},
  {"id": 1, "name": "SW5"},
  {"id": 2, "name": "SW2"},
  {"id": 3, "name": "SW4"},
  {"id": 4, "name": "SW3"}
 ],
 "links": [
  {"source": 0, "target": 1},
  {"source": 0, "target": 2},
  {"source": 1, "target": 3},
  {"source": 2, "target": 3},
  {"source": 2, "target": 4},
  {"source": 3, "target": 4}
 ]
}
```

Ukázka 4.12: Použití objektu Topology

Data získaná v tomto formátu lze snadno použít jako datový zdroj pro knihovnu *NextUI*. Příklad vykreslení topologie ukazuje Obrázek 4.4. Topologie vykreslená v rámci webové stránky je zároveň interaktivní a poskytuje řadu pokročilých funkcí. U jednotlivých zařízení například lze zobrazovat podrobnější informace, jako model síťového prvku či verzi jeho softwaru. V topologii lze také zvýrazňovat cesty, a tak například přehledně zobrazit logické topologie.




Obrázek 4.4: Příklad vykreslení topologie pomocí NextUI

### 4.3.2 Zpracování formátovaných dat

Pro usnadnění některých operací se strukturovanými daty obsahuje knihovna NUUAL kromě hlavních tříd také pomocné funkce a třídy navržené k těmto účelům. Tyto funkce umožňují manipulace s komplexními datovými strukturami, jako je filtrování na základě požadovaných hodnot, převod formátu pro potřeby tabulkového výpisu a podobně. Příkladem mohou být objekty z modulu `Writers`, které umožňují přetvořit získaná data do formátu Comma Separated Value (CSV) či vytvořit sešit Microsoft Excel ve formátu `.xlsx`. Díky tomu lze data snadno

distribuovat v široce přijímaném formátu i lidem bez znalosti programování. V součinnosti se třídou `Filter`, která umožňuje získávat z komplexních struktur pouze ta data, která odpovídají vyhledávacím kritériím, tak lze velmi snadno a rychle vytvořit uživatelsky přívětivé přehledy či statistiky.

## 4.4 Testování nástroje

Při tvorbě a testování nástroje NUAAL bylo použito co možná nejvíce různých platforem s různými verzemi operačních systému IOS a IOS XE. Testování probíhalo jak na fyzických zařízeních, tak na virtualizovaných platformách. Mezi testované platformy Cisco zařízení patří například:

- WS-C2960
- WS-C3750
- WS-C3650
- WS-C4500
- WS-C6500
- WS-C6807
- WS-C2960X
- WS-C3560
- WS-C3850
- WS-C9300
- CSR 1000v
- ASR 1000 Series
- Cisco 800 Series Routers
- Cisco ISR G2
- Cisco 4000 Series ISR

Vzhledem k rozmanité paletě testovaných verzí operačních systému *IOS* a *IOS XE* (verze 12.x, 15.x a 16.x) lze odůvodněně předpokládat kompatibilitu nástroje NUAAL i s dalšími platformami používajícími tyto operační systémy. Pro ověření budoucí rozšiřitelnosti nástroje i na zařízení jiných výrobců bylo zároveň otestováno použití knihovny *Netmiko* například na prvcích *Hewlett-Packard*.

Pro testování REST API kontroleru APIC-EM byl použit zejména volně dostupný kontroler (na adrese `https://sandboxapicem.cisco.com`), který je určen právě pro potřeby vývojářů. Tento kontroler je v době psaní této práce provozován ve verzi 1.3.1.9. Pro otestování některých novějších či pokročilejších funkcionalit byl také využit APIC-EM nasazený ve společnosti *ALEF NULA, a.s.*, provozovaný ve verzi 1.5.

Návrh nástroje se zároveň neobešel bez nezbytného testování dalších možných komunikačních kanálů, jako je například protokol NETCONF či SNMP. Testování základní třídy realizující spojení prostřednictvím API zahrnovalo také kontroler Cisco Evolved Programmable Network Manager (EPNM), DNAC či datacentrové přepínače *Nexus* s operačním systémem *NX OS*.

## 4.5 Instalace a distribuce nástroje

Knihovna NUAAL dodržuje doporučované a všeobecně používané postupy pro distribuci knihoven v jazyce Python. Veškeré zdrojové kódy jsou dostupné na webovém portálu GitHub, který

slouží zároveň pro ohlašovaní případných nalezených chyb a umožňuje spolupráci více vývojářů. Součástí knihovny je také její dokumentace, která je napsaná ve formátu *reStructuredText* (zkráceně *ReST*), který nabízí široké možnosti formátování právě pro účely dokumentování softwarových nástrojů [13]. S využitím nástroje *Sphinx* je zároveň možné tuto dokumentaci rozšířit o automaticky generované popisy jednotlivých tříd a funkcí, které jsou převzaty přímo ze zdrojových souborů. Tímto způsobem je možné získat jak statickou verzi dokumentace ve formátu PDF, tak její interaktivní verzi, kterou je možné nalézt na adrese nuaal.readthedocs.io. Dokumentace je psána v anglickém jazyce.

Samotný instalační balíček obsahující všechny potřebné soubory je dostupný z on-line repozitáře PyPI. Instalaci nástroje lze díky tomu provést pomocí nástroje `pip` zadáním následujícího příkazu do příkazové řádky:

```
pip install nuaal
```

Nástroj NUAAL byl vyvíjen a testován v Pythonu ve verzi 3.6. Pro správnou funkci nástroje je proto třeba používat Python ve verzi 3 a vyšší. Přidání podpory pro Python verze 2.7 je plánováno v budoucích verzích.

## 4.6 Rozšíření nástroje NUAAL

V aktuální verzi je knihovna Network Unified API Abstraction Library (NUAAL) zaměřena především na získávání informací ze zařízení ve strojově zpracovatelném formátu určenému zejména na další programové zpracování. Budoucí verze knihovny by měly kromě přidání podpory pro další zařízení zahrnovat také funkce pro analýzu získaných dat, které by umožňovaly automaticky detekovat případné problémy. Pro usnadnění práce s tímto nástrojem je také plánován vývoj grafického rozhraní prostřednictvím webové aplikace. Pro tento účel se zdá být nejvhodnějším webový framework *Django*, který je stejně jako knihovna NUAAL napsán v jazyce Python. Výsledná webová aplikace by mohla kromě přehlednějšího přístupu k získaným údajům zobrazovat také nejrůznější grafické reprezentace těchto dat, statistiky či vykreslovat síťové topologie s využitím dříve zmíněné knihovny *NextUI*.

Samotná data ze zařízení by také bylo možné obohatit o údaje poskytované některými webovými službami. Například společnost Cisco nabízí přístup k informační databázi prostřednictvím REST API. Tato databáze obsahuje mimo jiné také informace o známých softwarových chybách v operačních systémech IOS, možných bezpečnostních zranitelnostech, informace o podpoře jednotlivých platforem a další užitečné údaje. Zakomponování těchto dat by bylo značným přínosem například pro bezpečnostní audit síťových prvků či pro potřeby náhrady síťových prvků za jejich novější nástupce.

V neposlední řadě by měla být přidána také podpora pro automatizační nástroj Ansible. Integrace s tímto nástrojem souvisí s tvorbou příslušných modulů pro Ansible, které by využívaly

existující funkce knihovny NUAAL. Ačkoliv je integrace s tímto nástrojem jistě v mnoha ohledech přínosná, knihovna NUAAL by nadále měla zůstat samostatně fungujícím celkem bez nutnosti využití jiných nástrojů.

Aby knihovna NUAAL umožňovala programové řízení datových sítí, je samozřejmě nutné vytvořit funkce pro manipulaci konfigurací síťových prvků. Tento požadavek však přináší řadu možných rizik, která byla zmíněna dříve. Aby bylo možné naplnit i tento cíl, je nejdříve nutné spolehlivě zvládnout fázi získávání konfiguračních údajů, a následně dlouze a zodpovědně testovat zásahy do konfigurace zařízení. Přidáním takovýchto funkcí yb se stal nástroj jistě mocnějším, avšak při neopatrném použití by mohl napáchat značné škody. V prvotní fázi by mělo jít tedy spíše o generování konfiguračních souborů na základě definovaných šablon a údajů získaných z již provozovaných zařízení. Tyto soubory by však měly být před samotným nahráním do zařízení pečlivě zkontrolovány síťovým administrátorem.

# Kapitola 5

# Závěr

Hlavním cílem této práce bylo vytvořit softwarový nástroj, který by usnadňoval hromadnou správu síťových prvků, umožňoval rychleji řešit provozní problémy a poskytoval komplexní přehled o stavu dané sítě. Tyto požadavky naplňuje vytvořený nástroj NUAAL. Jedná se o knihovnu napsanou v jazyce Python, kterou tvoří přibližně třicet tříd zprostředkovávajících komunikaci se zařízeními, zpracování textových dat do strukturovaného formátu JSON, práci se strukturovanými daty a další operace. Jedním z požadavků bylo také vhodné rozšíření možností volně dostupného SDN kontroleru Cisco APIC-EM, který umožňuje pokročilou správu datových sítí, které jsou postaveny na síťových prvcích společnosti Cisco. Nástroj NUAAL proto obsahuje třídy umožňující přístup k údajům z tohoto kontroleru prostřednictvím programového rozhraní REST API a zároveň poskytuje funkce pro automatizaci některých pokročilejších operací prostřednictvím zmíněného kontroleru.

Aby bylo možné poskytnout obdobné funkce i na jinak nepodporovaných zařízeních, bylo třeba realizovat programové rozhraní pro komunikaci s individuálními síťovými prvky. Pro tento účel bylo po analýze dostupných možností zvoleno rozhraní příkazové řádky (CLI), což přineslo řadu výzev v souvislosti se zpracováním textových výstupů do strukturovaného, strojově zpracovatelného formátu. Za tímto účelem bylo třeba vytvořit sadu regulárních výrazů a přidružených funkcí, které textové výstupy zpracovávají. Díky volbě příkazové řádky však není třeba provádět žádné konfigurační změny na straně síťových prvků, díky čemuž je nástroj NUAAL snadno nasaditelný v existujících sítích. Sada poskytovaných údajů je přitom minimálně stejná, jako je tomu u informační databáze kontroleru APIC-EM.

Knihovna NUAAL je navržená tak, aby umožňovala tvorbu webových aplikací využívajících získaná data a snadnou integraci s jinými automatizačními nástroji, jako je například Ansible. V aktuální verzi knihovnu tvoří přibližně šest tisíc řádek zdrojových kódů, přičemž aktuálně podporuje pouze síťové prvky společnosti Cisco Systems. Funkce knihovny však lze poměrně snadno rozšířit i na síťové prvky jiných výrobců, a to při zachování jednotného programového

rozhraní.

Během psaní této práce je nástroj NUAAL testován pro možné komerční využití ve společnosti ALEF NULA, a.s., přičemž některé jeho části jsou již používány pro konkrétní potřeby dalších společností. Knihovna je volně dostupná na webovém portálu GitHub a obsahuje kompletní dokumentaci v anglickém jazyce.

## 5.1 Přínosy práce

Díky knihovně NUAAL lze výrazně urychlit některé z rutinních, avšak jinak časově náročných úkonů spojených se správou datových sítí. Díky získávání údajů ze zařízení ve strojovém formátu lze snadno vytvořit uživatelsky definovatelné přehledy obsahující klíčové informace, bez nutnosti zdlouhavého pročítání textových výstupů. Poskytnuté informace lze následně využít pro odhalení konfiguračních chyb, provozních problémů či tvorbu dokumentace dané sítě.

## 5.2 Budoucí rozšíření

Další vývoj knihovny bude zahrnovat rozšíření funkcí na zařízení dalších výrobců, tvorbu webového grafického rozhraní pro snazší práci s nástrojem a vytvoření modulů umožňujících integraci s automatizačním nástrojem Ansible. Kromě toho je plánováno přidání podpory pro další SDN kontrolery společnosti Cisco a provázání s webovými službami, poskytujícími například informace o chybách či bezpečnostních rizicích na konkrétních zařízeních. V budoucnu by měl nástroj také umožňovat provádění konfiguračních změn na zařízeních a poskytovat tak možnost programového řízení datových sítí.

# Seznam použitých zdrojů


[1] *Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017, Up 31 Percent From 2016* - Gartner, Inc. [Online, dostupné z: `https://www.gartner.com/newsroom/id/3598917`]. [cit. 3.3.2018].

[2] Jürgen Schönwälder. *IETF - RFC 3535* - Overview of the 2002 IAB Network Management Workshop; March 2003, DOI: 10.17487/RFC3535. [Online, dostupné z: `https://rfc-editor.org/rfc/rfc3535.txt`]. [cit. 1.4.2018].

[3] H. Ji, B. Zhang, G. Li, X. Gao, and Y. Li. *Challenges to the New Network Management Protocol: NETCONF* - 2009 First International Workshop on Education Technology and Computer Science; March 2009, Volume 1, 832-836, DOI: 10.1109/ETCS.2009.189.

[4] Software-defined networking, *Wikipedia, the free encyclopedia*. [Online, dostupné z: `https://en.wikipedia.org/wiki/Software-defined_networking`]. [cit. 1.5.2018].

[5] Benzekki Kamal, El Fergougui Abdeslam, and Elbelrhiti Elalaoui Abdelbaki. *Software-defined networking (SDN): A survey* - Security and Communication Networks; 2016, Volume 9/18, 5803-5833 DOI: 10.1002/sec.1737. [Online, dostupné z: `https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737`]. [cit. 10.4.2018].

[6] Eric Chou. *Mastering Python Networking: Advanced networking with Python*. Packt Publishing, Birmingham, UK, 2017. ISBN:978-1-78439-700-5, On-line dostupné z `http://www.packtpub.com/`.

[7] *Cisco Application Policy Infrastructure Controller Enterprise Module – Release 1.5* - Cisco Systems, Inc. [Online, dostupné z: `https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/application-policy-infrastructure-controller-enterprise-module/datasheet-c78-730594.pdf`], June 2017. [cit. 3.5.2018].

[8] Adam Radford, *APIC-EM - Deep Dive - DEVNET-1093* - Cisco Live, Melbourne. [Online, dostupné z: `https://www.ciscolive.com/global/on-demand-library`], 2016. [cit. 3.5.2018].

[9] Kirk Byers, *Netmiko's documentation* - ReadTheDocs.io. [Online, dostupné z: `http://netmiko.readthedocs.io/en/latest/`]. [cit. 3.5.2018].

[10] Kirk Byers, *Netmiko - Multi-vendor library to simplify Paramiko SSH connections to network devices* - GitHub, Inc. [Online, dostupné z: `https://github.com/ktbyers/netmiko`]. [cit. 3.5.2018].

[11] Google, *TextFSM* - GitHub, Inc. [Online, dostupné z: `https://github.com/google/textfsm`]. [cit. 3.5.2018].

[12] OpenDaylight, *Next UI* - GitHub, Inc. [Online, dostupné z: `https://github.com/opendaylight/next`]. [cit. 12.5.2018].

[13] reStructuredText, *Wikipedia, the free encyclopedia*. [Online, dostupné z: `https://en.wikipedia.org/wiki/ReStructuredText`]. [cit. 12.5.2018].

# Seznam použitých zkratek

**ACI** Application Centric Infrastructure 15

**API** Application Programming Interface 4, 7, 16, 17, 19, 21, 22, 26–28, 31, 34, 43, 44, 46

**APIC-EM** Application Policy Infrastructure Controller - Enterprise Module 5, 15–20, 24, 25, 27–29, 34, 39, 41, 43, 46

**CDP** Cisco Discovery Protocol 16, 40

**CLI** Command Line Interface *(Příkazová řádka)* 4, 7, 8, 10, 16, 22, 23, 26, 31, 46

**CSV** Comma Separated Value 42

**DHCP** Dynamic Host Configuration Protocol 18

**DNAC** Digital Network Architecture Center 18, 19, 24, 41, 43

**DNS** Domain Name System 18

**EPNM** Evolved Programmable Network Manager 43

**HA** High Availability 13

**HTTP** Hypertext Transfer Protocol 4, 21, 27, 28

**IAB** Internet Architecture Board 10

**IETF** Internet Engineering Task Force 9, 10

**IoT** Internet of Things 1

**ITIL** Information Technology Infrastructure Library 1

**JSON** JavaScript Object Notation 8, 21, 25, 41, 46

**MIB** Management Information Base 9, 12

**NUAAL** Network Unified API Abstraction Library 44–47

**OID** Object Identifier 9, 10

**REST** Representational state transfer 4, 7, 16, 17, 19, 21, 22, 26, 27, 31, 34, 43, 44, 46

**RPC** Remote Procedure Call 10, 11

**SDA** Software Defined Access 19

**SDN** Software Defined Network 12, 13, 15, 46, 47

**SLA** Service Level Agreement 13

**SNMP** Simple Network Management Protocol viii, 2, 4, 7, 9, 10, 12, 17, 19, 22

**SSH** Secure Shell 4, 8, 10, 23, 27, 30–32

**VLAN** Virtual Local Area Network 19, 39

**XML** Extensible Markup Language 11, 21

**YANG** Yet Another Next Generation 22

# Seznam obrázků

# Seznam ukázek

# Seznam příloh

1. **Příloha 1:** Dokumentace nástroje NUAAL
2. **Příloha 2:** Přiložené CD

   Diplomová práce ve formátu PDF

   Dokumentace nástroje NUAAL ve formátu PDF

   Instalační balíček nástroje NUAAL, obsahující zdrojové kódy