ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
CZECH UNIVERSITY OF TECHNOLOGY IN PRAGUE


FAKULTA ELEKTROTECHNICKÁ
KATEDRA TELEKOMUNIKAČNÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING
DEPARTMENT OF TELECOMMUNICATIONS ENGINEERING

# SPRÁVA A ŘÍZENÍ DATOVÝCH SÍTÍ POMOCÍ SOFTWAROVĚ ŘÍZENÉ SÍTĚ

NETWORK MANAGEMENT AND CONTROL USING SOFTWARE DEFINED NETWORKING CONCEPT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. MARTIN MIKÉSKA

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. LEOŠ BOHÁČ, Ph.D.

PRAHA 2019

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Správa a řízení datových sítí pomocí softwarově řízené sítě“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Praha ..... .....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Leoši Boháčovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Praha ..... .....

podpis autora(-ky)

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mikéska** Jméno: **Martin** Osobní číslo: **466724**

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:

**Správa a řízení datových sítí pomocí softwarově řízené sítě**

Název diplomové práce anglicky:

**Network Management and Control Using Software Defined Networking Concept**

Pokyny pro vypracování:

Cílem práce je identifikovat současné problémy v řízení a správě složitých datových sítí a s využitím konceptu softwarového řízení SDN na bázi zvoleného kontroléru navrhnout obecnou platformu, která by umožnila modulární vývoj těchto datových sítí. Navržená platforma by měla dostatečně abstrahovat detaily dílčích prvků různých technologií a poskytnout dobré sémantické API pro začlenění do budoucích systémů řízení. Praktickou částí bude vytvoření jednoduchého nástroje, který zjednoduší řešení některých problémů správy a řízení sítě, jejichž soubor bude výstupem analýzy problémů v sítích reálných provozovatelů.

Seznam doporučené literatury:

AVRAMOV, Lucien. The policy driven data center with ACI: architecture, concepts, and methodology. Indianapolis, IN: Cisco Press, 2015. ISBN 978-1-58714-490-5.
NADEAU, Thomas D. a Kenneth GRAY. SDN: software defined networks. Beijing: O'Reilly, 2013. ISBN 978-1449342302.
MARSCHKE, Doug, Jeff DOYLE a Pete MOYER. Software defined networking (SDN): anatomy of OpenFlow. Raleigh, NC: Lulu, 2015. ISBN 978-1483427232.
DONOVAN, John a Krish PRABHU. 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 978-1138631526.

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: _____

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

Rychle rostoucí počet zařízení připojených do sítě mění celkový charakter datového provozu, kde tradiční sítě naráží na své limity ve škálovatelnosti a flexibilitě. Celosvětový posun k automatizaci procesů a rychlejšímu nasazování služeb vede ke značnému rozšiřování softwarových technologií. Diplomová práce podrobně vysvětluje princip softwarově definovaných sítí, popisuje zásadní rozdíly oproti tradičním sítím, shrnuje vlastnosti, bezpečnostní problematiku, protokol OpenFlow, budoucí vize, přínos spojení s technologií NFV a poskytuje přehled SDN produktů pro podnikové sítě. Hlavní částí práce je analýza současných problémů při správě reálných datových sítí, kde jsem identifikoval celou řadu nápadů k automatizaci síťových procesů. Na základě této analýzy jsem vytvořil softwarový nástroj v programovacím jazyce Python. Vzniklý nástroj EasyPnP, v kombinaci s vybraným síťovým kontrolérem, usnadňuje proces hromadného přidávání nových či stávajících zařízení do sítě, šetří velké množství času a eliminuje chyby způsobené manuální instalací každého zařízení zvlášť. Zároveň slouží jako ukázka programovatelného přístupu k síťovým zařízením. Nástroj je primárně koncipován k vylepšení Plug and Play funkce SDN kontrolérů Cisco APIC-EM a Cisco DNA-C, avšak umožňuje modulární rozšíření pro síťové kontroléry jiných výrobců.

## KLÍČOVÁ SLOVA

Softwarově definované sítě, SDN, OpenFlow, umělá inteligence, virtualizace síťových funkcí, NFV, Cisco APIC-EM, Cisco DNA-C, Python, EasyPnP.

## SUMMARY

The fast-growing number of network devices changes the whole data traffic pattern and traditional networks reach their scalability and flexibility limits. The global automation and agility adoption leads to a significant software technologies expansion. This master thesis explains in detail the software defined networking principles, describes their essential differences from traditional networks, summarizes features, security issues, OpenFlow protocol, future visions, benefits of coexistence with the NFV technology and provides enterprise SDN products overview. The main part of the thesis comprises of an analysis of real network management problems, where a number of ideas for network process automation have been identified. Based on this analysis a software tool in Python programming language has been created. The EasyPnP tool together with a selected network controller simplifies the new or existing network devices mass deployment, saves a lot of time and eliminates human errors. It is useful example of programmable access to the network devices as well. This tool is primarily designed to enhance a Plug and Play function of the SDN controllers Cisco APIC-EM and Cisco DNA-C, however it allows modular extensions for third-party network controllers.

## INDEX TERMS

Software Defined Networking, SDN, OpenFlow, Artificial Intelligence, Network Functions Virtualization, NFV, Cisco APIC-EM, Cisco DNA-C, Python, EasyPnP.

MIKÉSKA, Martin. *Správa a řízení datových sítí pomocí softwarově řízené sítě*: diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, Katedra telekomunikační techniky, 2019. 121 s. Vedoucí práce byl doc. Ing. Leoš Boháč, Ph.D.

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM UKÁZEK

# SEZNAM TABULEK

# ÚVOD

Ve světě informačních technologií dochází k obrovskému nárůstu zařízení připojených do sítě. V současné době téměř každý vlastní chytrý mobilní telefon, notebook, tablet, hodinky apod. Uživatel často vyžaduje jejich vzájemné propojení a zároveň i konektivitu do celosvětové sítě Internet. Rostoucí počet rozmanitých zařízení mění celkový charakter internetového provozu a ovlivňuje zavedené procesy v síťové oblasti. Tato skutečnost přináší otázky, které je nutné řešit. Jedná se především o rychlé nasazení nových služeb, efektivní řízení datového toku, spolehlivé zajištění bezpečnosti a jednoduchou správu sítě.

Z analýzy společnosti GSMA Intelligence, zabývající se statistickým vyhodnocováním globálních dat mobilních operátorů, vyplývá, že na konci roku 2017 využívalo mobilní služby přibližně 5,0 miliard unikátních uživatelů. Do roku 2025 společnost předpokládá nárůst až na 5,9 miliard uživatelů, tj. 71 % budoucí světové populace[1]. K udržení tohoto tempa musí poskytovatelé internetových služeb ISP (Internet Service Provider) modernizovat svoji síťovou infrastrukturu a navyšovat přenosové rychlosti datové komunikace [1]. Zvyšující se počet síťových zařízení ovlivňuje i životní prostředí. Několik studií společností Webb a Lambert & Company udává, že informační a komunikační technologie od svého vzniku spotřebovaly více než 4,7 % celosvětové energie, a že emise $CO_2$ produkované tímto odvětvím by do roku 2020 mohly dosáhnout na 2-10 % celkových emisí $CO_2$ [2].

Celosvětový posun k rychlejšímu nasazování služeb vedl již na konci prvního desetiletí dvacátého století k velkému rozmachu softwarových technologií. Naopak můžeme pozorovat pokles specializovaných hardwarových systémů a nestandardních řídicích nástrojů. Postupně se začaly objevovat NGN (Next Generation Networks), česky tzv. sítě nové generace. Nástupem nových trendů, známých jako digitalizace, automatizace a orchestrace[2], se v IT čím dál častěji setkáváme se dvěma koncepty moderních datových sítí. Jsou to SDN (Software Defined Networking, viz kap. 1), česky softwarově definované sítě, a NFV (Network Functions Virtualization, viz kap. 2), česky virtualizace síťových funkcí. Jejich hlavní myšlenkou je otevřená standardizace a abstrakce funkcí, což vede k vytváření hardwarově nezávislých virtuálních platforem.

Uvedený vývoj neustále pokračuje, vznikají nové produkty a tzv. „sítě budoucnosti“, zaměřené na efektivní využití zdrojů, automatizaci správy síťových zařízení, inteligentní směrování paketů, rychlé zavádění nových služeb, zjednodušené řízení sítě a celou řadu dalších aspektů. Cílem je uspokojit aktuální potřeby a požadavky uživatelů, kde je rychlost velmi často prioritou číslo jedna. To je důvod, proč se softwarově definované sítě začínají více prosazovat nejenom v datových centrech, ale postupně i v podnikových sítích.

[1] V roce 2025 dle odhadů webu https://www.gsma.com/r/mobileeconomy/.

[2] Pojem „orchestrace“ představuje vzájemnou koordinaci systémů.

# CÍLE PRÁCE

V diplomové práci podrobně vysvětlím princip softwarově definovaných sítí a popíši rozdíly oproti architektuře tradičních sítí. Rozeberu základní funkce technologie SDN, zhodnotím výhody i nevýhody a nastíním budoucí vize tohoto přístupu. Okrajově také objasním protokol OpenFlow. Poté vysvětlím virtualizaci síťových funkcí, její význam, vlastnosti a přínosnou koexistenci NFV se softwarově definovanými sítěmi. Dále představím nabídku aktuálních open-source nástrojů a placených produktů různých firem pro podnikové sítě. Cílem teoretické části je velmi detailní uvedení do problematiky softwarově definovaných sítí, částečné i do virtualizace síťových funkcí.

Po dohodě s vedoucím diplomové práce, doc. Ing. Leošem Boháčem, Ph.D., jsem se primárně zaměřil na zpracování praktické části, přestože zadání bylo původně specifikováno spíše pro teoretický návrh obecné platformy řízení sítě. Proto bude největším přínosem diplomové práce právě praktická část, kde vytvořím modulární nástroj, který bude zjednodušovat či automatizovat některý ze síťových procesů. Nástroj bude vytvořen na základě analýzy současných problémů v sítích reálných provozovatelů. Cílem je poskytnout síťovým administrátorům jednoduchý produkt, který využívá funkcí síťového kontroléru, a vytvořit tak mezikrok do světa softwarově definovaných sítí pro často konzervativní správce.

# 1 SOFTWAROVĚ DEFINOVANÉ SÍTĚ

V této úvodní kapitole vysvětlím, co to vlastně softwarově definované sítě jsou, popíši jejich princip, funkce, základní charakteristiky, protokol OpenFlow a rozeberu aktuální přínos i budoucí vize tohoto přístupu.

Pojem SDN je v síťovém světě velmi rozšířen. Proto můžeme najít velké množství informací, které popisují tento přístup, avšak někdy až příliš odlišně, neúplně nebo velmi složitě. Abychom správně pochopili architekturu, funkce a výhody softwarově definované sítě, musíme nejprve porozumět tradiční architektuře, jež nadále ve většině stávajících sítí převládá.

## 1.1 Architektura tradiční sítě

Správa tradičních sítí začíná být s rostoucí komplexností velmi obtížná. Společnosti musí splňovat stále více bezpečnostních předpisů a zároveň roste poptávka po mobilitě. Základním znakem tradiční sítě je decentralizovaná architektura, což znamená, že rozhodnutí o zpracování protokolové datové jednotky PDU[1] (Protocol Data Unit) probíhá v každém zařízení zvlášť. Z tohoto důvodu je konfigurace jednotlivých síťových prvků časově náročná a náchylná k chybám. Následující text vychází především z odborného článku [3] a knihy [4].

### 1.1.1 Struktura síťového zařízení

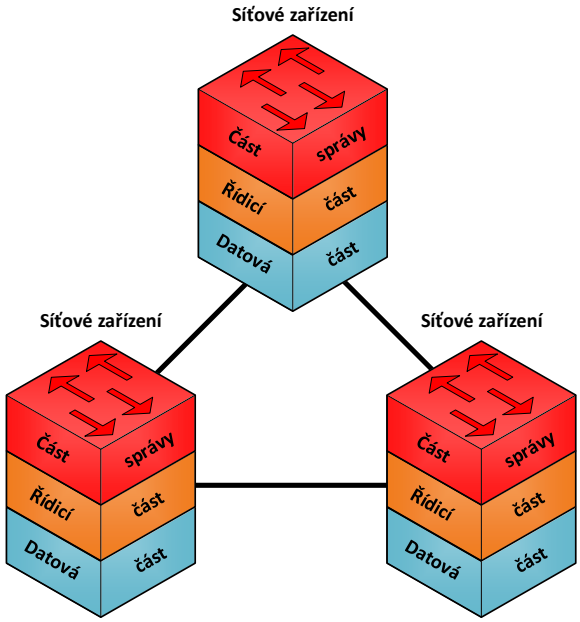
Jak můžeme vidět na obr. č. 1.1, zařízení v tradiční síti lze rozdělit do tří základních funkčních částí (také rovin či vrstev). Aby nedošlo k logické záměně s vrstvovým modelem ISO/OSI, budeme dále v textu používat dělení na části. Každá část vykonává v rámci daného zařízení důležitou roli zajišťující správnou funkci sítě. Jedná se o:

- **Část správy**
  Zahrnuje softwarové služby, nástroje i protokoly pro správu sítě (např. CLI, SNMP, NETCONF), používané ke vzdálenému monitorování a konfigurování funkcí, rozhraní, bezpečnostních zásad atd.
- **Řídicí část**
  Představuje firmwarovou část obsahující operační systém, protokoly a funkce, jež zaručují korektní zpracování datových jednotek. Jejím hlavním úkolem je udržovaní aktuálních informací v příslušných tabulkách (např. CAM, RIB) síťových prvků. Dále je odpovědná za různé řídicí a směrovací protokoly (např. OSPF, MPLS, BGP, STP), které plní uvedené tabulky informacemi a zprostředkují pohled na celkovou topologii sítě. V této části rovněž definujeme pravidla pro řízení datových toků (např. QoS, ACL).

[1] Specifický blok řídicích informací a uživatelských dat, který se v síti přenáší jako jeden celek. V praxi často používáme slovo „paket“. Detailněji na webu https://www.techterms.com/definition/pdu.

- **Datová část**
  Je v podstatě hardware síťových zařízení zajišťující přeposílání datových jednotek dle instrukcí řídicí části. K přenosu dat využíváme fyzická rozhraní a informace v příslušných tabulkách. Datová část dále obstarává aktualizaci TTL (Time To Live), ověření kontrolního součtu, správu vyrovnávací paměti, úpravu záhlaví aj.

Základní princip můžeme shrnout do jedné věty – v části správy provedeme konfiguraci daného zařízení, řídicí část vytváří v souladu s konfigurací patřičné instrukce, a datová část pak odpovídajícím způsobem přeposílá data. Datová část zpracovává každou příchozí datovou jednotku zvlášť, což vyžaduje vysoký výpočetní výkon v reálném čase, proto se typicky realizuje hardwarově. Naopak řídicí část nemá tak striktní rychlostní ani časové omezení, tudíž ji lze realizovat firmwarově [5].




Obr. 1.1: Architektura tradiční sítě. Překresleno dle [6].

V architektuře tradičních sítí jsou všechny tři části pevně implementované v každém síťovém zařízení. Celá síť je vysoce decentralizovaná. To vychází z původní myšlenky Internetu, kdy prioritou bylo zajistit především odolnost sítě před výpadkem. Právě odolnost je hlavní výhodou decentralizovaných sítí, jež mají stále své pevné uplatnění

### 1.1.2 Směrování

Zásadní odlišností tradiční sítě oproti softwarově definované je způsob předávání dat mezi síťovými prvky. Pevným spojením řídicí a datové části se provádí rozhodnutí o přeposlání datových jednotek v každém zařízení zvlášť. Mluvíme buď o přepínání na 2. (linkové) vrstvě nebo o směrování na 3. (síťové) vrstvě referenčního OSI (Open System Interconnection)

modelu. Síťové zařízení (převážně přepínač nebo směrovač) odstraní z datové jednotky hlavičku dané vrstvy a zpracuje ji. Tímto způsobem zařízení postupuje až ke své příslušné vrstvě. Poté jsou hlavičky opět postupně vytvořeny a nově zapouzdřená datová jednotka je přeposlána následujícímu síťovému zařízení k dalšímu zpracování. Takto putuje datový provoz od zdroje k cíli.

Každý síťový prvek si udržuje své vlastní informace o síti – obsažené například uvnitř CAM (Content Addressable Memory) tabulky u přepínače nebo RIB (Routing Information Base) tabulky u směrovače – a pomocí různých směrovacích či řídicích protokolů je distribuuje mezi ostatní zařízení.

Přesným popisem přepínání ani směrování se dále nebudeme zabývat. Výstižné a jednoduché vysvětlení uvedených pojmů najdeme například v internetovém článku [7].

### 1.1.3 Omezení

Nekonzistentní konfigurace jsou v dnešních sítích běžnou záležitostí. Kromě toho je velká většina podnikových sítí postavena na produktech různých výrobců. Právě kvůli odlišnostem v konfiguraci a typech jednotlivých zařízení může docházet k nežádoucímu chování sítě jako je ztráta paketů, vytváření smyček nebo nevhodných cest, zpoždění při přenosu apod.

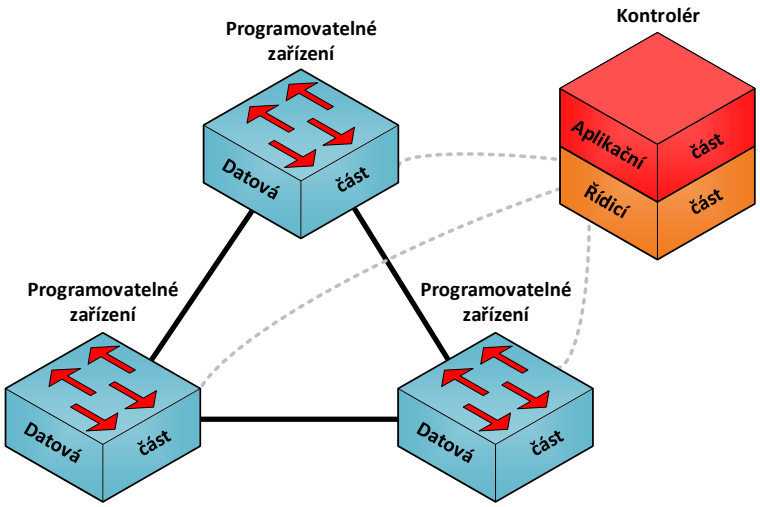
Hlavní nevýhodou tradičních sítí je poměrně statická a rigidní architektura. Je-li zapotřebí spustit novou funkci, službu, či aplikovat nové pravidlo, musíme konfigurovat jednotlivá síťová zařízení samostatně, a to s příkazy často specifickými pro daného dodavatele. K usnadnění správy sítě nabízí většina z nich svá proprietární řešení. Administrátoři sítí jsou tak nuceni provozovat různé produkty odlišných dodavatelů a s nimi i specializované týmy odborníků, kteří mají rozsáhlé znalosti všech příslušných typů zařízení. Výsledkem je finančně nákladná správa síťové infrastruktury s dlouhou návratností investičních cyklů, což snižuje flexibilitu sítě, brání inovacím a rychlému rozvoji. Náhradou bývají tzv. „middleboxy“, které pomáhají vytvořit potřebné funkce, zároveň však danou síť činí složitější.

Další nevýhodou je malá škálovatelnost, kdy síť není schopná efektivně reagovat na náhlé změny síťového provozu. Tradiční architekturu je rovněž složité segmentovat. Kromě počítačů, tabletů, telefonů apod., se inteligentními zařízeními stávají bezpečnostní systémy, kamery, bíle zboží atd. Je otázkou, jak všechna tato zařízení různých výrobců začlenit do sítě bezpečným a strukturovaným způsobem. Často jsou tyto prvky připojeny do sítě jako běžná síťová zařízení, což představuje obrovské riziko. Externí uživatelé mohou mít přes napadená zařízení přístup k celé síti. Například hackeři používající internetové připojení chytrých zařízení nebo prodejci využívající tzv. „zadních vrátek“ svého produktu. V obou případech neexistuje žádný důvod, aby měli k dané síti přístup. Bohužel nedostatky popsané dříve činí segmentaci sítě náročným procesem a vedou tak k uvedeným bezpečnostním problémům [8].

Nejen výše uvedená omezení dala za vznik myšlence softwarově definovaných sítí. V následujícím textu tento pojem podrobně vysvětlím.

## 1.2 Architektura SDN

Architektura softwarově definovaných sítí umožňuje efektivně a rychle reagovat na uživatelské požadavky i změny v síti prostřednictvím centralizované řídicí jednotky, která je na obr. č. 1.2 vyobrazena jako kontrolér. Představuje tak centralizovaný způsob návrhu a správy síťové infrastruktury. Jedná se o přístup oddělující datovou a řídicí část, což umožňuje vnímat celou síť s dostatečnou abstrakcí funkcí nižších vrstev [9].




Obr. 1.2: Architektura SDN sítě. Překresleno dle [6].

Některé dostupné informační zdroje chybně spojují pojem SDN se vším, co zahrnuje software. Nejčastější jsou však tyto čtyři základní definice [3]:

- Oddělení řídicí a datové části. Ze síťových zařízení se tak stávají jednoduché prvky přeposílající datové jednotky. Řídicí část předává instrukce datové části pomocí komunikačního protokolu (blíže v podkapitole 1.2.3).
- Řídicí část (inteligence sítě) je přesunuta do centralizovaného zařízení, tzv. SDN kontrolér nebo NOS (Network Operating System). Zpravidla se jedná o softwarovou platformu běžící na serveru, která usnadňuje řízení prvků datové části využitím abstraktního pohledu na celou síť.
- Programovatelná síť umožňující řízení i správu prostřednictvím mnoha aplikací, jež komunikují s kontrolérem pomocí API (Application Programming Interface). Kontrolér poté vynucuje požadavky u prvků datové části.
- Přepínání i směrování probíhá na základě datového toku. Ten je definován sadou hodnot a instrukcí mezi zdrojovým a cílovým zařízením. Abstrakce datového toku umožňuje sjednotit chování různých síťových prvků včetně přepínačů, směrovačů, firewallů, middleboxů, serverů atd.

Myšlenkou SDN je vytvoření dynamické a vysoce programovatelné síťové infrastruktury, která dokáže řídit základní komponenty sítě a zároveň je separována od aplikací a síťových služeb. To by mělo umožnit lepší škálovatelnost i spolehlivost, jednodušší segmentaci, větší efektivitu i flexibilitu, programovatelnost, centrální správu a dynamickou architekturu [9].

Přesunout síťovou infrastrukturu z tradiční na softwarově definovanou obvykle není možné ze dne na den, neboť potřebujeme nejen dostatek finančních zdrojů, ale i znalostí. Pro přechod na SDN infrastrukturu existují dva hlavní přístupy [10]:

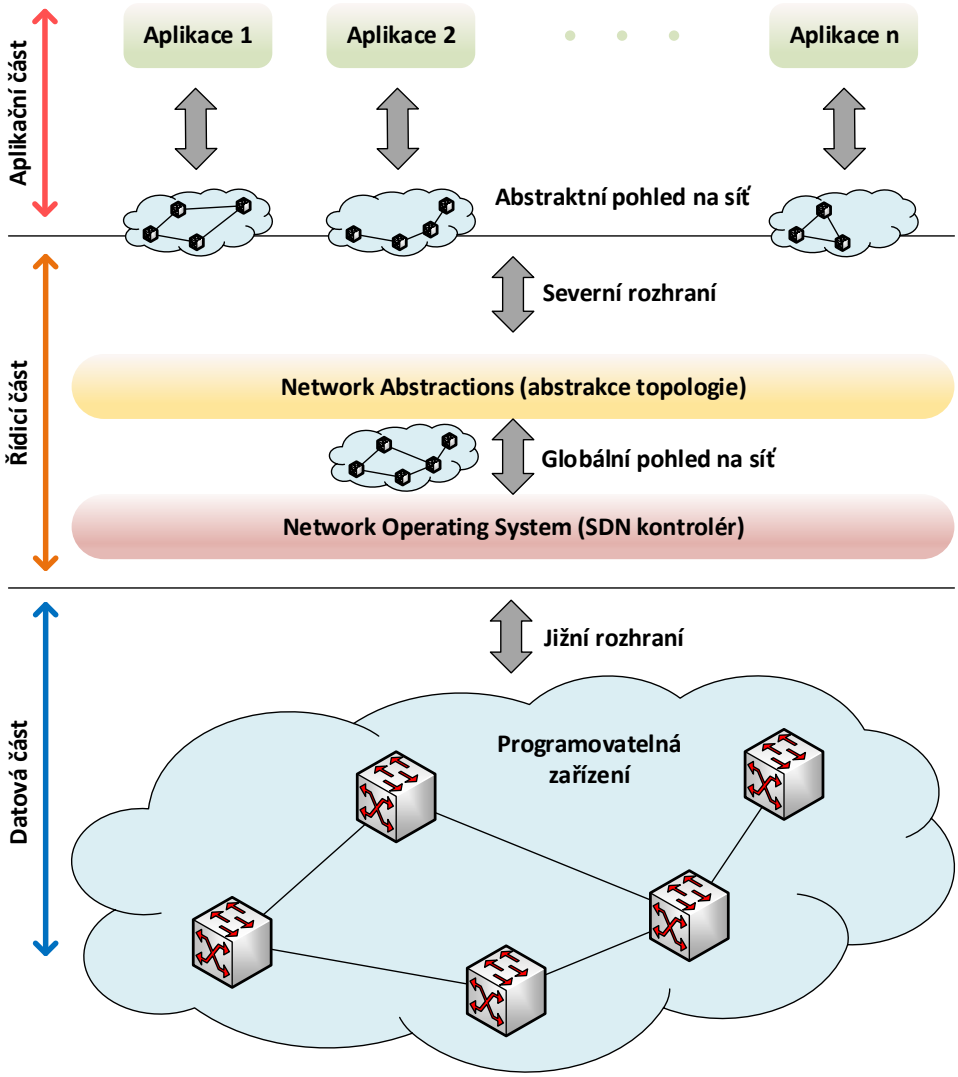
- **Smíšený přístup**
  Část sítě je nahrazena SDN zařízeními, což umožňuje využívat novou technologii bez narušení stávající sítě. Může tak docházet k účinným testům a k identifikování problémů v procesu nasazení SDN. Smíšený přístup tedy udržuje tradiční síť oddělenou od softwarově definované. Nevýhodou je správa dvou odlišných architektur.
- **Hybridní přístup**
  Vyžaduje použití hybridních přepínačů a směrovačů, jež dokážou komunikovat nejen s moderními prvky podporujícími SDN technologii, ale i s prvky staršími. Díky hybridním zařízením dochází k postupné obměně síťové infrastruktury, kde se kombinuje tradiční architektura se softwarově definovanou. Nevýhodou je větší pořizovací cena těchto zařízení.

V následujících podkapitolách se podíváme na základní součásti softwarově definované sítě, zobrazené v obr. č. 1.3. Text vychází především z odborného článku [3] a knih [4], [11]. Získané informace jsem doplňoval z internetových článků citovaných dále v textu.

### 1.2.1 Funkční části

Architektura softwarově definovaných sítí typicky sestává z těchto tří základních funkčních částí:

- **Datová část**
  Obsahuje síťová zařízení, která jsou vzájemně propojena metalickým přenosovým médiem, optickým kabelem nebo bezdrátově. Společně tvoří datovou část se stejnými vlastnostmi jako v tradiční architektuře. Nachází se zde pravidla a předávací tabulky pro práci s příchozími datovými jednotkami. Řídicí část předává datové části instrukce dle určitých údajů (MAC adresa, IP adresa, VLAN ID aj.). Zařízení poté může pakety přeposlat, zahodit, replikovat nebo zpracovat. Nejsou-li potřebné instrukce k dispozici, pakety se předají řídicí části k dalšímu zpracování.
- **Řídicí část**
  Zahrnuje signalizační, řídicí i směrovací protokoly, správu relací, autentizaci AAA (Authentication, Authorization and Accounting) a celou řadu dalších funkcí. Hlavní komponentou je jeden či více SDN kontrolérů bežících na fyzickém nebo virtuálním serveru. Kontrolér vyplňuje předávací tabulky pro sestavení nejvhodnější cesty, zpracovává aplikační události a informace o datovém provozu, zajišťuje bezpečnost,




Obr. 1.3: Základní rozdělení SDN sítě. Překresleno dle [3].

poskytuje globální pohled na celou síť, konfiguruje parametry i atributy prvků apod. S centralizovanou řídicí jednotkou je daleko snazší získat využitelné informace o datovém toku v síti v reálném čase a na jejich základě rychle i účinně rozhodovat.

Primárním úkolem řídicí části je tedy vytváření instrukcí, které kontrolér předává prvkům datové části přes jižní rozhraní (viz obr. č. 1.3). Zpracování těchto instrukcí probíhá v hardwaru daných zařízení, což je tradičně účinná metoda, jelikož hardwarový rozhodovací proces je velmi rychlý a snižuje tak celkové zpoždění [12].

- **Aplikační část**
  Nachází se nad řídicí částí. Jedná se o skupinu aplikací, které využívají severní API rozhraní k zavedení nástrojů pro řízení sítě, správu konfigurace, hlášení poruch, sledování událostí, monitorování datového provozu aj. Díky těmto aplikacím je síť plně

programovatelná, což umožňuje jednoduše rozšiřovat stávající funkce a služby. Prostředí je tak snadno přizpůsobitelné rychle se měnícím potřebám a uživatelským požadavkům. Aplikační část dále definuje bezpečnostní zásady, které kontrolér aplikuje na zařízení datové části.

### 1.2.2 Síťové prvky

V rámci této podkapitoly přiblížím tři hlavní síťové komponenty tvořící základ softwarově definovaných sítí:

- **Programovatelná zařízení**
  Vše, co bylo v předchozí podkapitole 1.2.1 zmíněno o datové části, zároveň patří k popisu jednotlivých SDN zařízení. Jedná se o hardwarové (někdy i softwarové) prvky, které vykonávají sadu základních síťových instrukcí s příchozími datovými jednotkami dle pokynů z kontroléru.

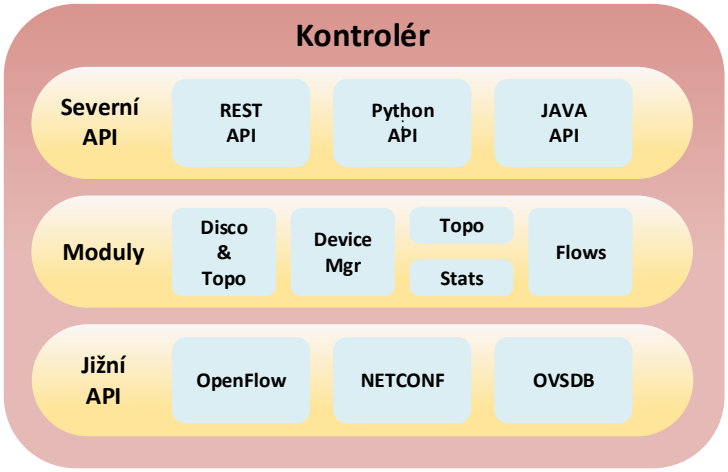
  Tabulky toků (angl. flow tables) jsou základní datové struktury předávacích tabulek v SDN zařízeních, jež podporují protokol OpenFlow (viz sekce 1.3). Tyto tabulky sestávají z jednotlivých informací a instrukcí, jak zpracovat příchozí pakety datového toku. Každý tok se postupně porovnává se všemi údaji v tabulce toků a vybere se první úplná shoda (bude podrobněji vysvětleno v podkapitole 1.3.2). Následně se provede akce dle příslušných instrukcí. Pokud není nalezena žádná shoda, paket je zahozen nebo předán kontroléru. Dle takto vytvořených pravidel se zařízení může chovat jako směrovač, přepínač, firewall, load balancer apod.

  Hardwarová zařízení s podporou technologie SDN se začínají postupně prosazovat v podnikových sítích. Softwarová zařízení se naopak projevují jako jedno z nejvíce slibných řešení pro datová centra s virtualizovanou síťovou infrastrukturou.

- **Kontrolér**
  Tzv. „mozek“ sítě, nacházející se mezi aplikační a datovou částí. Představuje prostředníka mezi uživatelem a síťovou infrastrukturou. Jedná se o strategický bod, který udržuje globální přehled o celé síti, implementuje konfigurace a síťové zásady. Přes jižní API rozhraní řídí všechna SDN zařízení a poskytuje severní API rozhraní pro různé programovatelné aplikace. Tato rozhraní si blíže popíšeme v podkapitole 1.2.3. Vzhledem k tomu, že kontrolér často spravuje velké množství síťových zařízení, se veškerá zpracování provádí na vysoce výkonném stroji s velkým procesorovým výkonem a s velkou kapacitou paměti.

  Jak můžeme vidět na obr. č. 1.4, kontrolér typicky obsahuje sadu modulů, které provádí různé úlohy. Mezi ně patří zjišťování síťových i koncových prvků, řízení datového toku, správa zařízení i topologie, sledování statistik atd. Moduly udržují lokální databáze obsahující aktuální informace o stavu sítě a jsou rozšiřovány pomocí různých aplikací. Ty vylepšují jejich účinnost a obstarávají pokročilejší funkce [13].

  Existuje velké množství jak komerčních, tak open-source kontrolérů. Nejznámější z nich si ukážeme v sekci 3.2 a 3.3.




Obr. 1.4: SDN kontrolér. Překresleno dle [4].

- **Aplikace**
  Patří do aplikační části a pro komunikaci s kontrolérem využívají severní API rozhraní (viz obr. č. 1.3). Technologii SDN lze nasadit v libovolném síťovém prostředí, od domácích a podnikových sítí, až po sítě poskytovatelů služeb a datových center. Tato různorodá prostředí vedou k celé řadě dostupných aplikací, které nabízí tradiční funkce jako je grafické uživatelské rozhraní pro správu kontroléru, vyvažování provozní zátěže, definování bezpečnostních pravidel, reakce na změny v topologii (selhání spojení, přidání zařízení) aj. Další příklady zahrnují funkce pro vynucování pravidel QoS, řízení mobility v bezdrátových sítích, snížení spotřeby energie, přesměrování provozu atd.

  Spojením kontroléru se zařízením dosáhneme pouze základních funkcí jednoduchého přepínače či směrovače. Aplikace poté implementují složitější funkce a inteligentní služby. Rozmanitost těchto aplikací v kombinaci se skutečným využitím je jednou ze silných stránek SDN architektury.

Aplikace jsou zároveň odpovědné za správu tabulek toků v SDN zařízeních podporujících protokol OpenFlow. Jejich úprava se provádí pomocí kontroléru ve dvou základních režimech:

- **Proaktivní režim**
  Definice toků jsou vytvořeny při spuštění aplikace a přetrvávají, dokud není provedena změna konfigurace. Mezi proaktivní patří také toky upravené na základě současného provozního zatížení.
- **Reaktivní režim**
  Toky jsou definovány v reakci na příchozí paket přenesený do kontroléru. Na jeho základě vytvoří aplikace novou definici toku, aby zařízení mohlo rychle zpracovávat další pakety tohoto typu.

### 1.2.3 Komunikační rozhraní

K výměně informací od aplikací přes kontrolér k síťovým zařízením musí existovat vhodný komunikační protokol. V SDN architektuře poskytuje kontrolér dvě hlavní otevřená komunikační rozhraní:

- **Jižní (angl. southbound) rozhraní**
  Využíváno pro přenos informací z centrálního prvku řídicí části, tedy kontroléru, směrem k zařízením v datové části. Umožňuje provádět změny v reálném čase dle uživatelských požadavků a aktuálních potřeb.

  Prvním otevřeným standardem tohoto rozhraní byl protokol OpenFlow (blíže si ho popíšeme v sekci 1.3), který nadále zůstává jedním z nejběžnějších. Není to však jediný standard SDN, ačkoli někteří používají pojmy SDN a OpenFlow zaměnitelně. V některých případech umožňuje kontrolér kombinovat OpenFlow s jinými standardy či proprietárními protokoly.

  Mezi další standardy jižního rozhraní patří OVSDB (Open vSwitch Database), ForCES (Forwarding and Control Element Separation), POF (Protocol Oblivious Forwarding), ROFL (Revised OpenFlow Library), HAL (Hardware Abstraction Layer), PAD (Programmable Abstraction of Data Path), OpenState, Cisco OpFlex atd. Každý z výše uvedených se vyznačuje specifickými vlastnostmi, které si nyní v krátkosti přiblížíme [3]:

  - **OVSDB** poskytuje pokročilé možnosti správy pro přepínače Open vSwitch[2]. Umožňuje například vytvoření více virtuálních instancí pro QoS, konfiguraci tunelů datových cest, shromažďování statistických údajů apod. Jedná se o doplňkový protokol k OpenFlow.
  - **ForCES** navrhuje pružnější přístup k tradičnímu řízení sítě, tedy bez nutnosti centralizovaného kontroléru. Řídicí a datová část je oddělena, ale potenciálně může být umístěna na stejném síťovém prvku. Řídicí část můžeme spravovat i firmwarem třetí strany.
  - **POF** je jedním z přímých konkurentů OpenFlow, který si klade za cíl zdokonalit a zjednodušit přeposílání datových jednotek. Místo tabulek toků zavádí sadu obecných instrukcí s názvem FIS (Flow Instruction Set), přičemž v datové části není zapotřebí žádný další protokol.
  - **OpFlex** přesunuje část správy sítě zpět k zařízením (podobně jako ForCES) s hlavním cílem zlepšit dostupnost a škálovatelnost. Protokol je vyvíjený společností Cisco.
  - **ROFL** i **OpenState** navrhuje abstraktní vrstvu, která minimalizuje rozdíly různých verzí protokolu OpenFlow. Poskytuje tak softwarovým vývojářům čisté API a zjednodušuje vývoj aplikací.
  - **HAL** i **PAD** sjednocuje rozhraní mezi protokolem OpenFlow a heterogenním hardwarovým zařízením.

[2] Více informací na webu https://www.openvswitch.org/.

Také CLI (Command Line Interface), SNMP (Simple Network Management Protocol) a NETCONF (Network Configuration Protocol) představují standardy jižního rozhraní. Pracovní skupina IETF (Internet Engineering Task Force) dále vyvinula standard I2RS (Interface to the Routing System), který dovoluje využití tradičních protokolů jako je OSPF, MPLS, BGP a IS-IS.

- **Severní (angl. northbound) rozhraní**
  Umožňuje komunikaci mezi kontrolérem v řídicí části a aplikacemi či službami v aplikační části. Jedná se o sadu otevřených programovatelných rozhraní, která usnadňují inovaci, umožňují automatizaci, pomáhají řídit datový provoz a nasazovat nové služby.

  Severní rozhraní poskytuje abstrakci síťových zařízení a celkové topologie. To znamená, že nabízí obecné rozhraní, které umožňuje, aby softwarové aplikace fungovaly bez znalosti vlastností jednotlivých zařízení. Díky tomu lze vyvíjet aplikace či služby, které fungují na zařízeních různých výrobců, přestože se mohou podstatně lišit v implementaci. Zásluhou této abstrakce můžeme virtualizovat síť, tedy oddělovat síťovou službu od fyzické sítě.

  Nejběžnější je RESTful API založené na technologii REST (Representational State Transfer) využívající protokol HTTP (HyperText Transfer Protocol), běžně používaný pro přenos webového provozu. RESTful API je jednoduché a rozšiřitelné, proto se stalo dominantní metodou volání API napříč sítěmi. Dalším používaným rozhraním je Java API či Python API.

  Výrobci kontroléru navrhují svá vlastní severní rozhraní s konkrétní definicí. Mluvíme například o programovacích jazycích Frenetic, Nettle, NetCore, Procera, Pyretic, NetKAT, SFNet, PANE a celé řadě dalších, které poskytují výkonné mechanismy usnadňující vývoj softwarových modulů i aplikací.

V současné době neexistuje otevřený standard severního rozhraní, což je považováno za velký nedostatek architektury SDN. Některé kompetentní orgány se proto aktivně snaží o jeho standardizaci. Tato situace představuje vynikající příležitost pro spolupráci výrobců s uživatelskou komunitou. Uvedená problematika je zásadní nejen pro podporu přenositelnosti aplikací, ale i pro interoperabilitu odlišných řídicích platforem. Pravděpodobně však vznikne více standardů severního rozhraní, protože požadavky síťových aplikací jsou velmi odlišné.

### 1.2.4 Typy architektury

V předchozím textu jsme většinu času uvažovali centralizovanou architekturu s jedním kontrolérem. Zvyšující se složitost sítí a rostoucí oblíbenost SDN technologie vede k prosazování architektury s více kontroléry. K této problematice přistupujeme dvěma způsoby, kde oba mají své výhody i nevýhody [14]:

- **Fyzicky centralizovaná**
  V centralizované SDN architektuře se používá pouze jeden kontrolér, který řídí všechny komponenty síťové infrastruktury. Nastávají tak tři zásadní otázky – jak řešit účinnost, škálovatelnost a dostupnost. Zajistit dostatečnou účinnost a škálovatelnost je s použitím jediného kontroléru náročný úkol. Stejně tak dostupnost, která se dále dělí na bezpečnost a redundanci, což jsou dva nejdůležitější aspekty řídicího prvku, jelikož se jedná o jediný bod selhání (angl. single point of failure). Nejvyšší prioritou je zaručení bezpečnosti kontroléru, kdy musíme zabránit neoprávněnému přístupu, abychom neztratili kontrolu nad celou sítí. Klíčová je i redundance, která je zapotřebí v případě výpadku řídicího prvku. Naopak výhodou je, že nemusíme řešit žádnou komunikaci ani synchronizaci mezi více kontroléry.

  Všechny tyto argumenty přiměly návrháře sítí k integraci většího počtu kontrolérů, čímž je možné výše uvedené problémy minimalizovat. Někteří výrobci (Beacon, NOX) se snaží pouze o logické rozdělení jednoho kontroléru. V tomto případě je ale zřejmé, že se stále jedná o centralizovanou architekturu.
- **Fyzicky distribuovaná**
  V distribuované SDN architektuře se nachází větší počet kontrolérů, které vzájemně spolupracují, aby dosáhly určité úrovně výkonu a škálovatelnosti. Liší se především jejich odpovědnost a pozice uvnitř sítě. Fyzicky distribuovaná architektura může být logicky centralizovaná nebo logicky distribuovaná:
  - V **logicky centralizované** mají všechny kontroléry stejnou odpovědnost a jejich informace jsou synchronizované. Každý kontrolér má globální přehled o celé síti. Zátěž se dělí mezi více řídicích prvků, které se ale k nižším vrstvám tváří jako jeden.
  - V **logicky distribuované** má každý kontrolér přehled pouze o svojí doméně, za niž je odpovědný. V této doméně může libovolně rozhodovat a nové údaje sdílí k ostatním kontrolérům.

  Dále existuje plochá a hierarchická architektura:
  - V **ploché architektuře** jsou kontroléry umístěny horizontálně na jedné úrovni. Řídicí část se tak skládá pouze z jedné vrstvy. Každý kontrolér má stejnou odpovědnost, ale pouze částečný pohled na síť. Tento přístup poskytuje větší odolnost proti selhání, ale řízení kontrolérů je náročnější.
  - V **hierarchické architektuře** jsou kontroléry rozděleny vertikálně mezi více úrovní. Řídicí část má několik vrstev, obvykle dvě nebo tři. Kontroléry mají různé odpovědnosti a rozhodují na základě částečného pohledu na síť. Spodní vrstva obvykle obsahuje místní (angl. local) kontroléry, zatímco horní vrstva obsahuje jeden páteřní (angl. root) kontrolér. Přetrvává tedy problém jediného bodu selhání, přestože se jedná pouze o selhání jedné vrstvy. Řízení kontrolérů je však snazší.

Uvedené přístupy mohou snížit celkovou odezvu v porovnání s centralizovanou architekturou. Poslední dělení fyzicky distribuované (zároveň ale logicky centralizované) architektury je na dynamickou a statickou:

- V **dynamické architektuře** jsou vazby i pozice mezi kontroléry a SDN zařízeními proměnné, což činí síť flexibilní.
- Ve **statické architektuře** jsou vazby i pozice neměnné, což zvyšuje odolnost a snižuje režii sítě.

Synchronizace informací mezi více kontroléry probíhá přes východní-západní (angl. east-west) rozhraní, které zatím není standardizováno. Logicky centralizované architektury využívají systém výměny oznámení nebo zasílání zpráv. Logicky distribuované architektury používají především známe směrovací protokoly jako jsou BGP, OSPF a IS-IS.

Počet kontrolérů a jejich pozice v distribuovaných SDN architekturách ovlivňuje celkový výkon řídicí části. Abychom určili správný počet kontrolérů i jejich vhodné síťové umístění, musíme zohlednit následující tři faktory [14]:

- Požadované časové limity, zejména celková odezva a zpoždění.
- Požadovaná metrika jako dostupnost, spolehlivost, šířka pásma apod.
- Síťová topologie a infrastruktura.

Správným umístěním řídicích prvků v sítích WAN (Wide Area Network) se zmenší zpoždění mezi kontrolérem a SDN zařízeními (případně mezi dvěma kontroléry), minimalizuje se celkový čas odezvy a zvyšuje se reakční schopnost sítě.

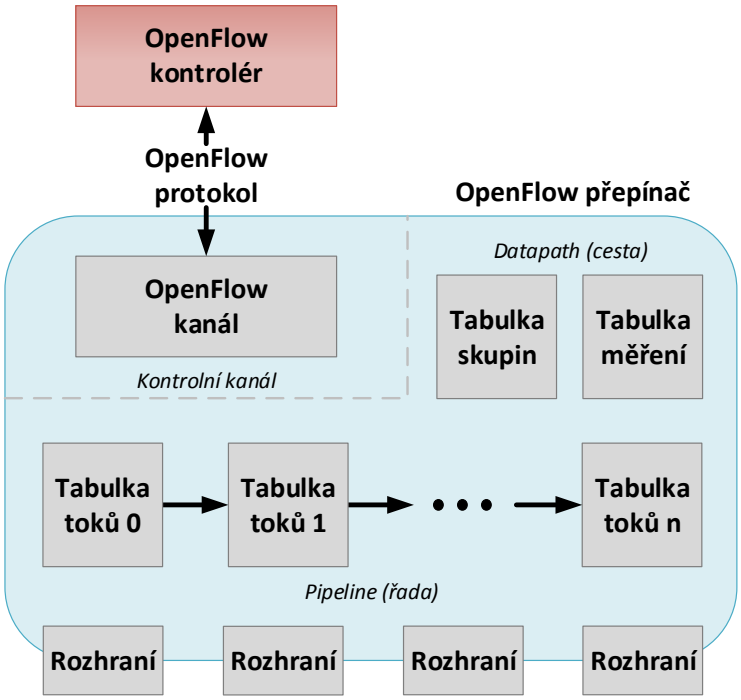
## 1.3 OpenFlow

Problematika protokolu OpenFlow je velmi rozsáhlá. Proto se o ní zmíním jen okrajově, abychom porozuměli hlavním principům. Můžeme však nalézt celou řadu odborné literatury zabývající se touto tématikou. Doporučuji knihu [15] a především samotnou specifikaci [16], kde je vše velmi detailně i srozumitelně popsáno. Při zpracování této sekce jsem převážně vycházel z těchto informačních zdrojů, není-li u textu uvedeno jinak.

Často je OpenFlow a SDN považováno za jednu a tutéž technologii. Ve skutečnosti je OpenFlow jen jednou z mnoha částí v celkové architektuře SDN. Jak jsem již uvedl v podkapitole 1.2.3, jde o nejpoužívanější a nejrozšířenější otevřený standard jižního rozhraní, který umožňuje řídicí části komunikovat s částí datovou. Existuje mnoho výrobců síťových zařízení, kteří podporují OpenFlow, například Cisco, Juniper, Big Switch Networks, Brocade, Arista, Extreme Networks, IBM, Dell, NoviFlow, HP, NEC aj.

Vývoj OpenFlow specifikace probíhá již řadu let. V roce 2008 vytvořila skupina lidí, kteří se neformálně setkali na Stanfordské univerzitě, neziskovou internetovou organizaci openflow.org podporující tento koncept. Od svého založení byla organizace určena k experimentům s přepínáním v otevřených sítích s důrazem na komerční využití, právě pro-

střednictvím implementací této veřejné specifikace. První oficiální vydání, verze 1.0.0, se objevilo v prosinci 2009. Dne 21. března 2011 byla založena společnost ONF (Open Network Foundation) za účelem urychlit komercializaci SDN. U zrodu společnosti stály firmy Deutsche Telekom, Facebook, Google, Microsoft, Yahoo! a Verizon. Skládá se z řady pracovních skupin a jednou z hlavních výhod je, že její správní radu tvoří technologičtí odborníci, techničtí ředitelé i spolupracovníci velkých firem (příklad viz výše). Tato skutečnost pomáhá zabránit tomu, aby ONF podporovala zájmy jednoho dodavatele nad jiným. Po vydání verze 1.1.0 se další vývoj specifikace přesunul z openflow.org pod záštitu ONF [4].




Obr. 1.5: Struktura OpenFlow prostředí. Překresleno dle [16].

OpenFlow specifikace definuje jak protokol, který se používá mezi kontrolérem a přepínačem, tak i chování, které se očekává od přepínače. Všimněme si, že používám název přepínač, nikoliv zařízení. Je to proto, že přepínač je výraz použitý přímo ve specifikaci. Obecně se ale začíná více prosazovat termín zařízení, jelikož již existují například bezdrátové přístupové body, které jsou řízeny OpenFlow kontrolérem.

### 1.3.1 Přepínač

Obrázek č. 1.5 znázorňuje základní strukturu OpenFlow prostředí. Kontrolér komunikuje s přepínačem pomocí protokolu přes zabezpečený kanál (může být jeden či více), kde je zajištěno asymetrické šifrování založené na TLS (Transport Layer Security), ačkoliv jsou povolena i nešifrovaná TCP spojení (Transmission Control Protocol). Ta se používají převážně v datových centrech, kde je kontrolér a přepínač v jednom prostředí, čímž dochází

k minimalizaci řídicích zpráv. OpenFlow definuje fyzická, logická a rezervovaná rozhraní (v praxi běžně používáme slovo „porty“). Přepínač se přes tato rozhraní může připojit k jiným přepínačům, síťovým zařízením, případně k zařízením koncového uživatele. Jeho architektura se dále skládá ze tří typů tabulek, které jsou obvykle implementovány v hardwaru nebo firmwaru [17]:

- **Tabulka toků (angl. flow table)**
  Obsahuje informace, dle kterých se zpracovávají příchozí pakety. Těchto tabulek může v rámci jednoho přepínače existovat více. Podrobnější vysvětlení bude následovat v podkapitole 1.3.2.
- **Tabulka skupin (angl. group table)**
  Jednotlivé toky mohou být sjednoceny v tabulce skupin, která se tak skládá ze skupinových záznamů. Instrukce poté ovlivňují celou skupinu toků.
- **Tabulka měření (angl. meter table)**
  Umožňuje celou řadu činností souvisejících s kvalitou a výkonem jednotlivých toků.

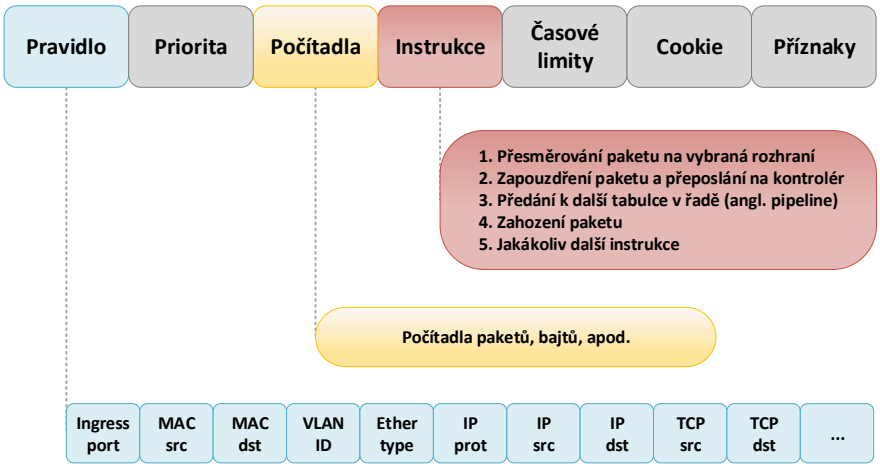
V OpenFlow architektuře musí vždy existovat minimálně jeden kontrolér komunikující s jedním či více přepínači. Následující čtyři body popisují základní princip OpenFlow řešení [4]:

- Kontrolér vytváří a upravuje tabulku toků uvnitř přepínače přidáváním či odstraňováním záznamů jednotlivých toků.
- Přepínač vyhodnocuje příchozí pakety a hledá shodu v tabulce toků, poté provádí příslušné instrukce.
- Pokud přepínač nenalezne žádnou shodu v tabulce toků, paket je zapouzdřen a předán kontroléru, popřípadě zahozen.
- Kontrolér aktualizuje tabulku toků v přepínači s příchodem nových, prozatím nepřiřazených, paketů.

Nyní je vhodné definovat termín „tok“. Obecně platí, že tok je posloupnost paketů procházející sítí, jež v záhlaví sdílí určitou sadu hodnot odpovídající konkrétnímu záznamu v tabulce toků. Příkladem je tok skládající se z paketů se stejnou zdrojovou i cílovou IP adresou, se stejným VLAN identifikátorem, rozhraním apod. Význam bude podrobněji vysvětlen v následující podkapitole [17].

### 1.3.2 Tabulka toků

Každý OpenFlow přepínač má jednu nebo více tabulek toků. Jedná se o základní stavební blok architektury logických přepínačů. Tabulky jsou uspořádány v řadě za sebou (angl. pipeline), přičemž jsou číselně označeny vzestupně od nuly. Tabulka toků obsahuje informace a instrukce k určitému toku, které se používají pro porovnávání i zpracovávání paketů. Tyto informace a instrukce nazývám dále v textu jako položky. Jak můžeme vidět na obr. č. 1.6, jednotlivé položky se skládají z následujících polí [17]:




Obr. 1.6: Uspořádání tabulky toků. Překresleno dle [3].

- **Pravidlo (angl. match fields)**
  Slouží jako kritérium pro určení shody s údaji v paketu. Je-li nalezena shoda s pravidlem, paket patří k danému toku. Pravidlo sestává z několika povinných hodnot jako vstupní rozhraní, MAC adresy, IP protokol, IP adresy a TCP/UDP rozhraní. Používají se i další hodnoty, pokud je podporuje přepínač a verze OpenFlow.
- **Priorita (angl. priority)**
  Každé položce se přidělí priorita zpracování, kde 0 reprezentuje nejnižší hodnotu.
- **Počítadla (angl. counters)**
  Aktualizují se v případě shody paketu s pravidlem. Používají se ke sledování různých statistik, které se vztahují k danému toku.
- **Instrukce (angl. instructions)**
  Definují, co je potřeba provést, shoduje-li se paket s pravidlem. Nejčastěji jde o přesměrování paketu na vybraná rozhraní, přeposlání na kontrolér, předání k další tabulce toků atd.
- **Časové limity (angl. timeouts)**
  Vypršení určitého času (hard_timeout) nebo času nečinnosti (idle_timeout) způsobí odstranění položky z tabulky toků.
- **Cookie (angl. cookie)**
  Hodnota definovaná kontrolérem k filtrování statistik, změně nebo smazání vybraného toku. Nevyužívá se při zpracovávání paketů.
- **Příznaky (angl. flags)**
  Určují, jak spravovat jednotlivé položky toku.

Výchozí položkou v tabulce toků je zpravidla tzv. prázdný (angl. table-miss) vstup. Má nejnižší prioritu a vždy se shoduje s libovolnými údaji v hlavičce paketu. Jakmile paket dorazí na přepínač, je postupně porovnán s položkami v první tabulce toků (s číslem 0)

od nejvyšší priority po nejnižší. Pokud není nalezena žádná shoda a v tabulce neexistuje výchozí položka, paket je zahozen. Existuje-li výchozí položka, obvykle následuje jedna ze tří akcí [17]:

- Přeposlání paketu na kontrolér, který definuje nový tok nebo paket zahodí.
- Předání paketu k další tabulce toků v řadě s vyšším číslem.
- Zahození paketu.

Je-li v tabulce toků nalezena shoda u více položek, upřednostňuje se položka s vyšší prioritou. Poté mohou následovat například tyto akce [17]:

- Aktualizace všech čítačů souvisejících s položkou.
- Vykonání instrukcí přiřazených k dané položce.
- Předání paketu k další tabulce toků, skupin nebo měření, popřípadě jeho přesměrování na výstupní rozhraní.

Pokud proběhne přesměrování paketu na výstupní rozhraní, provede se příslušná sada instrukcí a paket je zařazen do fronty, kde čeká na své odeslání.

### 1.3.3 Protokol

Definuje specifické zprávy vyměňované mezi OpenFlow kontrolérem a OpenFlow přepínačem, čímž jednoznačně charakterizuje tento standard. Příklad komunikace je uveden na obr. č. 1.7 a vybrané zprávy jsou popsány v tab. č. 1.1. Protokol umožňuje kontroléru přidávat, upravovat či odstraňovat položky v tabulce toků přepínače přes zabezpečený kanál. Jsou podporovány tři typy zpráv [17]:

- **Kontrolér-přepínač**
  Iniciovány kontrolérem a v některých případech vyžadují reakci přepínače. Slouží k řízení nebo ověření stavu přepínače, a také k přímému směrování paketů na jeho výstupní rozhraní.
- **Asynchronní**
  Odesílány přepínačem a používají se k informování kontroléru o příchodu paketu, událostech v sítí i změnách stavu. Neexistuje-li žádná shoda v tabulce toků, používá přepínač tento typ zpráv k přeposlání paketu na kontrolér.
- **Symetrické**
  Posílány kontrolérem nebo přepínačem bez předchozího požadavku. Využívají se pro specifické účely jako sestavování spojení, vzájemné informování, měření zpoždění apod.

Protokol OpenFlow zajišťuje spolehlivé doručení a zpracování řídicích zpráv bez automatického potvrzení (angl. acknowledgements). Přijme-li zařízení instrukci, kterou nedokáže vykonat, musí odeslat chybovou zprávu.

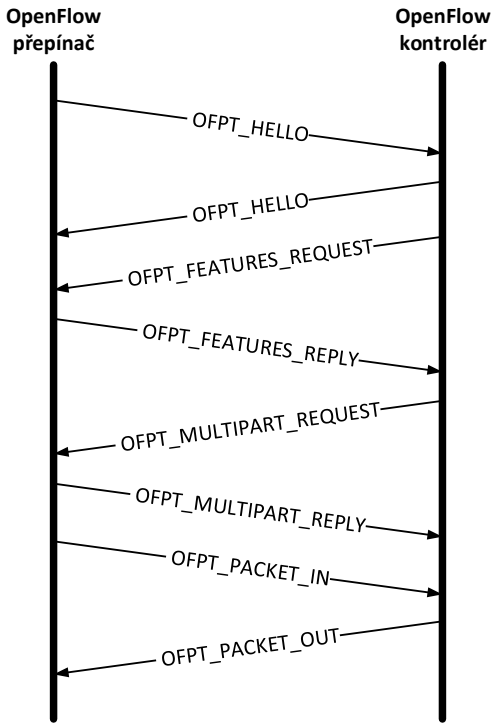
Tab. 1.1: OpenFlow zprávy. Převzato z [17].

| **Kontrolér-přepínač** | |
|---|---|
| Features | Informace o funkcích přepínače. |
| Configuration | Nastavení konfiguračních parametrů. |
| Modify-State | Přidání, odstranění či úprava položek v tabulce toků a nastavení vlastností rozhraní přepínače. |
| Read-State | Shromažďování informací z přepínače, například aktuální konfigurace, statistiky, funkce atd. |
| Packet-Out | Přesměrování paketu na výstupní rozhraní přepínače. |
| Barrier | Požadavek k informování o dokončeném zpracování instrukcí. |
| Role-Request | Žádost o konfigurování role OpenFlow kanálu. Užitečné v případě připojení přepínače k více kontrolérům. |
| Asynchronous-Configuration | Žádost o filtrování asynchronních zpráv. Užitečné v případě připojení přepínače k více kontrolérům. |
| **Asynchronní** | |
| Packet-In | Přesměrování paketu na kontrolér. |
| Flow-Removed | Informování kontroléru o odstranění položky z tabulky toků. |
| Flow-Monitor | Informování kontroléru o změně položky v tabulce toků. |
| Port-Status | Oznámení kontroléru o změně stavu rozhraní. |
| **Symetrické** | |
| Hello | Synchronní zprávy při sestavování spojení mezi kontrolérem a přepínačem. |
| Echo | Žádost o odezvu ze strany kontroléru nebo přepínače. Je vyžadována odpověď. |
| Error | Notifikace o chybě nebo problému. |
| Experimenter | Slouží k experimentálním účelům. |

### 1.3.4 Verze

Technologie OpenFlow se vyvíjela s každou další verzí její specifikace. Většinou šlo o odstranění problémů předchozí verze a přidání nových funkcí. Přestože z počátku bylo hlavní myšlenkou především experimentální používání, postupně docházelo k výraznému prosazování tohoto standardu s cílem interoperability. Proto bylo nutné zajistit zpětnou kompatibilitu. Existuje však mnoho funkcí, které byly uvedeny v dřívějších verzích, ale v aktuální již nejsou podporovány. Dále není pravidlem, že OpenFlow zařízení vždy implementuje nejnovější verzi.

Verze 1.0.0 byla vydána 31. prosince 2009. Považuje se za první oficiální a referenční verzi, ačkoli předtím existovalo mnoho předběžných vydání pro experimentální účely. Uvádí základní myšlenku technologie OpenFlow, vysvětluje fyzické rozhraní přepínače, tabulku toků se 3 položkami a 12 povinnými poli, proces hledání shody, tři typy zpráv apod. Následovala verze 1.1.0 uvedená 28. února 2011, která oproti předchozí verzi podporovala více tabulek toků, tabulku skupin, virtuální rozhraní přepínače, VLAN značky aj. Dále vyšla 5. prosince 2011 verze 1.2.0, která obsahovala rozšíření procesu hledání shody, podporovala IPv6 a prostředí s více kontroléry. Následovali verze 1.3.0-1.4.1, vydané v prů-




Obr. 1.7: Příklad OpenFlow komunikace. Překresleno dle [4].

běhu let 2012 až 2015, s rozšířenou podporou IPv6 hlaviček, tabulkou měření, počítadly toků atd. Aktuální verze 1.5.1 vyšla 26. března 2015. Byla přidána možnost zpracování PPP (Point-to-Point Protocol) rámce, IP paketu, shoda dle výstupního rozhraní aj. Rozdíly v jednotlivých verzích jsou podrobně popsány ve specifikaci [16].

## 1.4 Shrnutí vlastností

Softwarově definované sítě přináší celou řadu výhod a stávají se ideální pro dnešní dynamické aplikace. Nesou s sebou však také jistá bezpečnostní rizika a problémy způsobené především centralizovaností kontroléru. V této sekci si některé klady i zápory přiblížíme.

### 1.4.1 Výhody

Technologie SDN poskytuje výkonný přístup k řízení komplexních sítí s vysokými a proměnnými nároky, proto se stává běžnou součástí sítí poskytovatelů, rozsáhlých firem, cloudových infrastruktur apod, jež pracují s velkým množstvím dat [17]. Organizace často používají SDN ke snížení složitosti své sítě, lepší kontrole zásad a odstranění závislostí různých výrobců. Nadále je však možné využívat klasické síťové technologie jako je VLAN, MPLS aj [9]. V předchozím textu, při popisu jednotlivých částí SDN architektury, jsem již některé výhody zmínil. Patří mezi ně především [18]:

- **Programovatelná síť**
  Síť je přímo programovatelná oddělením řídicích funkcí od datové části směřující provoz, což umožňuje spravovat síť vlastními nebo otevřenými (angl. open-source) automatizačními nástroji. Správci sítí mohou velmi rychle konfigurovat, zabezpečovat a optimalizovat síťová zařízení prostřednictvím hardwarově nezávislých programů.
- **Centralizovaná správa**
  Správa sítě je centralizována do kontroléru, který vytváří globální pohled na celou síť. Aplikace a síťová zařízení vnímají kontrolér jako jediný řídicí prvek sítě. Tímto centrálním prvkem ovládáme celou síť, získáváme přehled o jednotlivých zařízeních, k dispozici máme různé statistiky a grafy, na jejichž základě můžeme vytvářet nové síťové zásady apod. Důležitou součástí jsou také informace o datovém provozu a upozornění na vzniklé události. Tento přístup je však zároveň zdrojem několika problému, jež budou popsány v podkapitole 1.4.2 a 1.4.4.
- **Otevřený standard**
  Využívání síly open-source vývojové komunity výrazně urychluje vývoj v SDN prostředí. Vedle usnadnění výzkumu i experimentování poskytují otevřená rozhraní interoperabilitu zařízení různých výrobců, což vytváří konkurenční prostředí snižující spotřebitelské náklady. Výsledkem je mnohem větší počet jednotlivců a organizací schopných tvořit aplikace při řešení dnešních síťových problémů. To vede k lepšímu a rychlejšímu technologickému pokroku. Zjednodušuje se také návrh i provoz sítě, jelikož ovládáme prvky prostřednictvím kontroléru namísto konfigurace několika zařízení s příkazy specifickými pro konkrétního výrobce [4].
- **Flexibilita a inovace**
  Přístup SDN dovoluje správcům dynamicky přizpůsobovat infrastrukturu i optimalizovat tok datového provozu, zavádět nové typy aplikací i služeb, aby rychle plnili cíle a měnící se potřeby zákazníků. Tyto skutečnosti zvyšují výkonnost i škálovatelnost sítě a mohou zároveň nabídnout nové příjmy i vyšší finanční zisk.
- **Integrace virtuálního prostředí**
  Sjednocení prostředí virtuálních serverů s technologií SDN umožňuje centralizovanou správu síťových zásad pro virtuální i fyzické zdroje. Správnou integraci zajišťuje VEB (Virtual Ethernet Bridge) ovládaný kontrolérem [12].
- **Snížení CapEx (Capital Expenditures)**
  Česky kapitálové náklady nebo také investiční náklady, jsou výdaje na pořízení nových a obnovu starých zařízení. SDN tuto potřebu potenciálně snižuje a spíše podporuje model „pay-as-you-grow“[3]. Zakoupením a vhodným naprogramováním SDN přepínače můžeme nahradit i jiná síťová zařízení, například firewall. Musíme však počítat s určitým výkonnostním omezením, jelikož se nejedná o specializovaný hardware pro danou činnost.

[3] Jedná se o používaný obchodní model, kdy zákazníci postupně dokupují zařízení nebo licence dle svých aktuálních kapacitních potřeb.

- **Snížení OpEx (Operational Expenditures)**
  Česky provozní náklady, jsou výdaje k zajištění běžných činností pro funkční chod společnosti. SDN automatizuje správu a orchestraci, čímž dochází ke snížení celkového času řízení sítě a minimalizují se chyby způsobené manuální konfigurací jednotlivých zařízení zvlášť. Kromě zvýšení spolehlivosti, přesnosti i konzistence konfigurací se zvyšuje také rychlost reakce, kdy v případě žádosti náhlé změny dokáže správce upravit stávající konfigurace mnohem rychleji.

### 1.4.2 Nevýhody

Základní nevýhody, vznikající centralizovaností sítě s jedním kontrolérem, jsme si již letmo nastínili v podkapitole 1.2.4. Můžeme je minimalizovat integrací většího počtu kontroléru, což s sebou však přináší jiné typy problémů. V obou případech mluvíme hlavně o [4]:

- **Zpoždění**
  Jelikož SDN zařízení vyžaduje pokyny od kontroléru, je pravděpodobné, že určitý počet rozhodnutí bude trpět zpožděním. To poroste se zvyšujícím se počtem prvků a jejich požadavků. Kontrolér běžící na serveru musí být schopný reagovat dostatečně rychle, aby zpoždění nemělo žádný vliv na provoz sítě. Proto záleží na výkonnostních parametrech daného serveru.
- **Škálovatelnost**
  Centralizovaný kontrolér přebírá odpovědnost za tvorbu topologie, stanovení cest, optimalizaci toků, reakci na změny atd. S rostoucím počtem zařízení v síti vznikají otázky, s jak velkým počtem prvků je jediný kontrolér schopný manipulovat a kdy dojde k překročení jeho kapacitního limitu.

  Tyto problémy můžeme řešit přidáním dalších kontrolérů, s čímž přichází nové otázky, např. vzájemná komunikace, ideální počet, výkonnost serverů apod. Pro výměnu všech informací mezi více kontroléry se používá východní-západní (angl. east-west) rozhraní. Zatím však nedošlo k jeho standardizaci, tudíž se nedá považovat za naprosto jednoznačné a spolehlivé. Pokud neuváženě zvyšujeme počet kontroléru v síti, roste nejenom finanční náročnost, ale také složitost sítě, čímž se odchylujeme od původního konceptu SDN. Je tedy důležité správně odhadnout vhodný počet kontrolérů vůči celkovému množství zařízení v síti [19].
- **Vysoká dostupnost**
  Pojem známý pod anglickou zkratkou HA (High Availability), jehož cílem je zajistit dohodnutou úroveň provozní výkonnosti a spolehlivosti. Pro SDN síť to znamená, že centralizovaný kontrolér nesmí představovat jediný bod selhání (angl. single point of failure). Musí tedy existovat redundantní kontrolér, aby byl dostupný určitý procesní výkon v případě selhání jednoho z nich. Synchronizační data musí být zrcadlena tak, aby všechny kontroléry mohly sjednoceně ovládat síťová zařízení. Rovněž spojení musí být redundantní, aby se zajistila funkční cesta ze zařízení k alespoň jednomu kontroléru.

- **Bezpečnost**
  Centralizovaný systém je náchylný na síťové útoky více než distribuovaný, jelikož se útočník soustředí pouze na jediný cílový bod. Je důležité správně a dostatečně chránit kontrolér, ale i komunikační kanály mezi ním a síťovými zařízeními. O bezpečnosti mluvím detailněji v následujících podkapitolách 1.4.3 a 1.4.4.
- **Softwarové a hardwarové selhání**
  Neexistuje operační systém, který by neobsahoval žádné chyby. Může se tedy stát, že kontrolér či zařízení selže, stejně tak nutný upgrade způsobí výpadek. Z těchto důvodu je vhodné zajistit výše popsanou vysokou dostupnost.
- **Podpora zařízení**
  Technologie SDN a související nástroje často podporují pouze nové produkty, které již v operačním systému implementují potřebné funkce a vlastnosti. Nelze tedy využívat nepodporovaná zařízení v konceptu softwarově definovaných sítí [19].
- **Implementace**
  Většina ukázkových implementací předpokládá úplný přechod z tradiční sítě na softwarově definovanou. V reálných sítích je však zpravidla limitujícím aspektem rozpočet, proto musí docházet k postupnému nasazovaní nové technologie. Řešíme tak koexistenci dvou rozdílných architektur i kompatibilitu mezi existujícími síťovými prvky a SDN zařízeními. Další problém může vzniknout při spojení mezi více doménami. Většina prací se zaměřuje na centralizované řízení jedné administrativní domény, což ovšem není vhodné pro případ několika SDN sítí, které jsou nezávisle řízeny větším počtem kontrolérů [20].

### 1.4.3 Bezpečnostní výhody

Bezpečnost v technologii SDN je minimálně stejně důležitá jako v tradičních sítích, proto na ni klademe velmi vysoké nároky. Softwarově definované sítě přináší nové bezpečnostní funkce, které zvyšují reakční schopnost a minimalizují výpadek služeb během útoku. Příkladem může být ochrana virtualizovanou službou, poskytující bezpečnost ve formě SaaS (Software-as-a-Service)[4], jež je součástí cloudového prostředí. Tato skutečnost představuje vývoj bezpečnostní architektury z tradičního modelu, založeného na ochraně perimetru pomocí hardwarových zařízení, k dynamickému modelu, který využívá virtualizované bezpečnostní funkce [11].

Globální pohled na celou síť značně ovlivňuje bezpečnost SDN sítě, jelikož každé zařízení shromažďuje a aktivně reportuje statistiky veškerého provozu směrem ke kontroléru. Zde dochází k hlavní odlišnosti od tradičních distribuovaných sítí, kde zařízení vyžadují vzájemnou výměnu informací a také jistý čas konvergence, aby dokázala určit stav zbývající části sítě. Mezi bezpečnostní výhody patří [21]:

[4]Model SaaS (česky „software jako služba“) umožňuje uživatelům připojení ke cloudovým aplikacím a jejich zpoplatněné využívání přes Internet.

- **Detekce narušení**
  Kontrolér může aktivovat celosíťový systém detekce narušení IDS (Intrusion Detection System), který analyzuje shromážděné statistiky provozu ze všech zařízení za účelem nalezení škodlivých dat. Systém se dělí na detekci zneužití a detekci anomálií. U detekce zneužití jsou definovány profily známých útoků a narušení je hlášeno, pokud chování sítě odpovídá některému z profilů. Při detekci anomálií je profilován běžný síťový provoz, kdy nedochází k žádným škodlivým činnostem. Pokud se chování sítě podstatně liší od těchto profilů, je hlášeno narušení v síti. Tento přístup může identifikovat nové útoky a nevyžaduje důkladné znalosti o škodlivém chování, ale zároveň produkuje více falešných poplachů v porovnání s detekcí zneužití. V tradičních sítích je IDS zařízením, které je instalováno v určité části sítě, a má tak menší detekční schopnosti kvůli omezené viditelnosti.
- **Detekce škodlivého zařízení**
  Napadené zařízení může cíleně vynechávat některé příchozí pakety, zároveň je však schopné správně zpracovat různé kontrolní příkazy, aby skrylo své škodlivé chování. Technologie SDN řešení tohoto problému usnadňuje, jelikož zařízení periodicky předává kontroléru statistiky přijatých, odeslaných a zahozených paketů. Analýza těchto informací usnadňuje určení škodlivého zařízení, popřípadě zužuje jejich seznam.
- **Forenzní analýza**
  Kontrolér zaznamenává informace v reálném čase, což výrazně usnadňuje forenzní analýzu. Můžeme zpětně procházet statistiky, datové toky, různé údaje, změny v síti aj. Tato metoda je užitečná k vytváření účinných obranných mechanismů proti opakujícím se výskytům stejných nebo podobných útoků. Dále můžeme identifikovat osobu odpovědnou za provedení nepovolených změn uvnitř sítě.
- **Kontrolní mechanismy**
  Jedním z příkladů jsou podmíněná pravidla, která kontrolér definuje uvnitř zařízení. Jejich aktivace probíhá při splnění určitých podmínek, jež se obvykle vztahují k různým statistikám (např. počet přijatých paketů specifického toku, během určitého časového období, přesahuje předdefinovanou hranici). Pravidlo určuje, jak má zařízení reagovat při splnění podmínky, což poskytuje odolnost proti útokům, zejména typu DoS (Denial of Service)[5]. Reakcí může být změna cílové adresy určitých paketů tak, aby byly doručeny do „honeypotu“. To je izolované monitorované místo, které slouží jako past pro shromažďování dalších informací o škodlivých aktivitách.
- **Řízení provozu**
  SDN usnadňuje řízení datového provozu díky myšlence toků, kde kombinace hodnot záhlaví určuje zacházení s paketem. Kontrolérem můžeme dynamicky omezit například TCP pakety přicházející z určitého zařízení. Tím zachytíme škodlivý provoz přímo na konkrétním prvku. V tradičních sítích je tato funkce zprostředkována firewallem nebo přístupovými listy.

[5] Jde o typ útoku na internetové služby, jehož cílem je tuto službu znepřístupnit ostatním uživatelům.

### 1.4.4 Bezpečnostní nevýhody

Technologie SDN s sebou přináší také určité bezpečnostní hrozby, jež se dají rozdělit do tří kategorií dle cílové oblasti. Jde o útoky na řídicí část, datovou část, popřípadě na komunikaci mezi nimi. Ke každému typu útoku v krátkosti uvádím protiopatření, které je možné použít ke snížení rizika napadení. Patří sem [21]:

- **Odepření služeb (angl. Denial of Service)**
  Jelikož přepínače datové části mají omezenou kapacitu úložiště, nelze v tabulkách toků pokrýt všechny možné kombinace záhlaví paketů. Z tohoto důvodu využíváme reaktivního režimu, který byl zmíněn v podkapitole 1.2.2. Nenajde-li zařízení odpovídající pravidlo toku pro příchozí paket, uloží jej dočasně do vyrovnávací paměti a následně předá kontroléru. Ten paket zpracuje a vytvoří novou položku v tabulce toků. Popsaný mechanismus je zranitelný útoky typu DoS, kdy útočník zaplavuje přepínač pakety, které patří k neurčeným tokům. Vyrovnávací paměť se rychle zaplní a brání ukládání relevantních paketů.

  Můžeme použít proaktivní režim, kdy přepínač nečeká na přijetí nových paketů k vyžádání odpovídajícího pravidla, ale raději ukládá maximální počet položek do tabulky toků.
- **Distribuované odepření služeb (angl. Distributed Denial of Service)**
  Řídicí část je citlivá na útoky typu DDoS, kdy dochází k napadení většího počtu zařízení v síti. Útok způsobí zaplnění vyrovnávacích pamětí přepínačů, je generováno mnoho dotazů odesílaných do kontroléru, jehož procesní výkon dosáhne maxima. Výsledkem je zpoždění odpovědí či zahazování paketů.

  Řešením je využití více kontrolérů, tedy fyzicky distribuovaná SDN architektura, avšak s logickou centralizací. Tato problematika byla popsána v podkapitole 1.2.4. Každý přepínač je připojen k více kontrolérům, kde jeden z nich je vybrán jako hlavní (angl. master). Důležité je zajistit zabezpečené propojení mezi kontroléry. Otázkou však zůstává jejich optimální rozmístění.
- **Zneužití přístupu ke kontroléru**
  Kontrolér je v centralizované architektuře jediná řídicí část celé sítě, proto je velmi důležité zajistit, aby nemohlo dojít k jeho zneužití. Nastane-li tak, může útočník ovládat všechna zařízení v síti a použít je k zahájení útoků, například nasměrování všech paketů k cílovému hostiteli, službě aj.

  Ke snížení rizika můžeme využít většího počtu kontrolérů. Řešení není efektivní, pokud všechny kontroléry běží na stejné nebo podobné serverové platformě, protože mohou sdílet shodné chyby, což útočníkovi umožňuje získat přístup ke všem kontrolérům v síti současně. Základním předpokladem je tedy rozmanitost platforem.
- **Šifrování paketů**
  Přestože směrování paketů na základě tabulky toků přináší nové možnosti správy sítě, není z bezpečnostního hlediska zcela zřejmé, jak zacházet se šifrovanými pakety, kde nejsou viditelné všechny hodnoty záhlaví. Podobný problém vytváří síťové tune-

lování, které zapouzdřuje šifrované pakety do jiných paketů, což útočníkům umožňuje překonat bezpečnostní zásady definující přístup k síti.

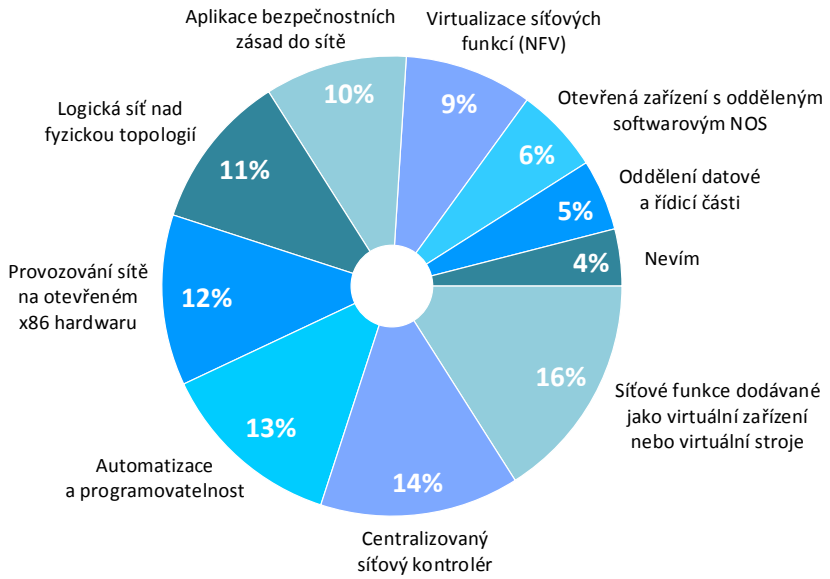
Řešením je vytvoření vzorových modelů (analýzou datového provozu) k identifikaci zašifrovaných paketů.

- **Útok MITM (Man-in-the-middle)**
  Odesílání nešifrovaných komunikačních zpráv mezi zařízením a kontrolérem činí spojení náchylné k narušení. Útočník může odposlouchávat vyměňované informace, popřípadě dokáže upravovat či vytvářet položky v tabulce toků.

  Spojení musí být šifrované a mělo by obsahovat informaci, která jednoznačně identifikuje konkrétní zařízení. Součástí zprávy by měla být také časová známka, jež zabraňuje opakovaným útokům.

## 1.5 Přínos a budoucí vize

Většina aspektů předurčuje SDN k velmi nadějné budoucnosti. Tato technologie je silně podporována dodavateli a lídry na trhu se síťovými komponenty, nasazena v datových centrech po celém světě, aktivně vyvíjena a obklopena rozsáhlou otevřenou (angl. open-source) komunitou. Tyto skutečnosti povzbuzují vývojáře i společnosti k vytváření nových aplikací, které mění tradiční způsoby správy a řízení sítě.




Obr. 1.8: Graf vnímání definice SDN v podnicích. Překresleno dle [22].

Graf na obr. č. 1.8 ukazuje, že síťoví administrátoři nemají jasnou představu o tom, co se pod pojmem SDN skrývá. Hlavní přínosy, které technologie SDN vnesla nejen do podnikových sítí, byly detailně popsány v předchozích částech, zejména v sekci 1.4. Lze je shrnout do těchto bodů [9]:

- Abstrakce síťových zdrojů a vytvoření programově konfigurovatelných zařízení.
- Zjednodušení složitých síťových topologií a odstranění závislostí dodavatelů.
- Výraznější zabezpečení sítě, snadnější prosazování bezpečnostních zásad, efektivnější kontrola přístupu, monitorování sítě a analýza dat.
- Lepší škálovatelnost, mobilita, údržba apod.
- Podpora nových konceptů v oblasti IoT a integrace cloudových služeb.

Existuje mnoho dalších kladných argumentů, stejně tak jako záporných. Vždy záleží na preferencích a potřebách jednotlivých společností. Každá technologie se však vyvíjí určitým směrem, dají se proto předpokládat budoucí kroky. Pro následný text jsem vycházel především z knihy [11] a z článků citovaných dále v textu.

### 1.5.1 Aktuální trendy

S obrovským nárůstem generování dat musíme vyvíjet dokonalejší technologie pro jejich zachytávání a analyzování. Tempo se exponenciálně zrychluje především kvůli počtu IoT (Internet of Things) zařízení. Do roku 2020 bude k dispozici přibližně 20 až 50 miliard připojených zařízení a globální provoz bude vyšší než 2,3 ZB (zettabyte) [11]. Tato skutečnost významně ovlivňuje schopnost odvodit cenné informace. Proto se zvyšuje hodnota dat, společně se zájmem o strojové učení ML (Machine Learning) i umělou inteligenci AI (Artificial Intelligence), jež řídí automatizaci a zvyšují interakci mezi lidmi a stroji. Současné trendy můžeme rozdělit do několika bloků:

- **Integrace umělé inteligence**
  Množství dostupných dat, nové tréninkové metody pro hluboké učení DL (Deep Learning) a využívání grafických procesorů (používaných v počítačích, herních konzolích apod.) k modelování neuronových sítí NN (Neural Networks), umožnilo moderní rozvoj AI i vznik praktických aplikací. Techniky ML a DL jsou iterativní povahy, neustále se učí a optimalizují. Umělá inteligence činí rozhodnutí založená na datech, vzorcích a předchozích výsledcích. Nástroje, vytvořené na základě velkého množství dat, představují znatelný pokrok v oblastech jako je detekce objektů, rozpoznávání řeči a zpracování přirozeného jazyka.
- **Automatizace procesů**
  Podniky po celém světě usilují o finanční zisk z obrovského objemu generovaní provozních dat. Pomocí AI dochází k automatizování či optimalizaci vybraných procesů, což společnostem přináší značný obchodní růst. Snižuje se riziko vnesení lidské chyby a zlepšuje se kvalita pracovních míst odstraněním stereotypních či nudných úkolů, čímž se zvyšuje produktivita práce [23].
- **Změna požadavků**
  Rozvíjející se vztah mezi lidmi a AI povede k odlišnému stanovení priorit pro dovednosti a znalosti pracovníků. Zásadní bude obecné porozumění, empatie, tolerance a schopnost spolupracovat. Jedná se o lidské vlastnosti, které nemůže stroj repliko-

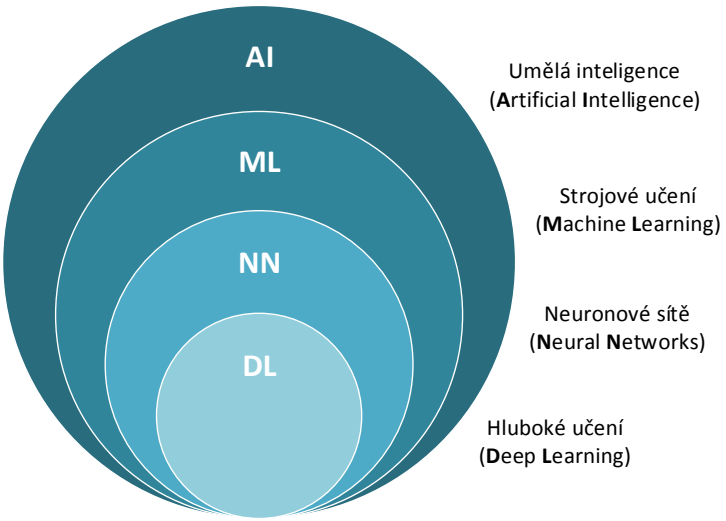
vat, dokáže je pouze obohatit či rozšířit. Lidé mohou využívat AI k řešení složitých problémů strukturovaným způsobem, ale sami musí být zodpovědní za konečné rozhodnutí i výsledky.

- **Bezpečnost a soukromí**
  Vzhledem k rostoucímu využívání dat se vyskytují obavy ohledně zabezpečení, správy totožnosti a soukromí. Tyto otázky začaly ovlivňovat činnost na Internetu. Téměř polovina uživatelů se více stará o soukromí a omezuje své internetové aktivity. Světové ekonomické fórum označilo kybernetickou kriminalitu za jedno z největších globálních rizik.

S touto rychlou evolucí Raymond Kurzweil[6] věří, že do roku 2029 dosáhneme jedinečnosti (angl. singularity) – bodu, kdy AI překoná lidskou inteligenci. Tato predikce vyvolává řadu společenských otázek nejen ohledně zásad pro návrh AI aplikací.

### 1.5.2 Umělá inteligence

V současné době často slyšíme nebo čteme o umělé inteligenci, strojovém učení, neuronových sítích apod. I v předchozím textu jsem několikrát zmínil zkratky AI, ML, NN a DL. Jedná se o tzv. „buzzwords“, tedy slova, která jsou aktuálně velmi populární. Níže jednotlivé pojmy vysvětlím, abychom správně chápali jejich význam a přínos v síťovém světě.




Obr. 1.9: Vzájemná souvislost mezi AI, ML, NN a DL. Překresleno dle [24].

Obr. č. 1.9 představuje vzájemnou souvislost zmíněných pojmů. Hluboké učení i neuronové sítě jsou podmnožinou strojového učení, které je podmnožinou umělé inteligence. Přestože jsou úzce spjaty, mají různé významy [24]:

[6] Raymond „Ray“ Kurzweil (* 12. února 1948) je americký vynálezce, futurolog a jedna z klíčových postav technologického vývoje ve společnosti Google.

- **Umělá inteligence – AI**
  Oblast počítačové vědy (angl. computer science), kde dochází k produkci inteligentních strojů či programů, jež dokáží plnit komplexní úkoly vyžadující lidskou inteligenci, jako je vizuální vnímání, rozpoznávání řeči, obrazu, jazykový překlad atd. Jinými slovy, umělá inteligence se zabývá řešením úkolů, které jsou relativně snadné pro lidi, ale obtížné pro stroje.

  Hlavní součástí výzkumu AI je znalostní inženýrství. Stroje mohou reagovat jako lidé pouze v případě, mají-li přesné informace popisující dané chování. Inicializace zdravého rozumu, uvažování a samostatného řešení je dalším krokem vývoje [25].

- **Strojové učení – ML**
  Aplikuje principy počítačové vědy, především statistickou a prediktivní analýzu, k vytváření statistických modelů používaných k předvídání nebo identifikaci vzorů na základě předchozích informací či pozorovaných datasetů (tzv. „Big Data“).

  Strojové učení je typ umělé inteligence, jež umožňuje softwarovým aplikacím přesnější predikci výsledků bez výslovného programování. Program napodobující lidské chování je umělá inteligence. Pokud však parametry nejsou automaticky odvozeny z dat, nejedná se o strojové učení. Systémy ML se musí neustále učit a přizpůsobovat přijímáním informací, jejich zpracováváním a ukládáním do paměti.

- **Neuronové sítě – NN**
  Jejich myšlenkou je napodobit obrovské množství hustě propojených mozkových buněk. Zatímco elementární jednotkou mozku je neuron, základním stavebním kamenem neuronové sítě je perceptron. Systém se naučí věci, rozpozná vzory a učiní rozhodnutí jako člověk. Vše bez přímého programování.

  Neuronové sítě se skládají z několika propojených uzlů (simulující biologické neurony mozku), které jsou organizovány ve vrstvách. Každému spojení je přiřazena určitá váha, každému perceptronu (uzlu) prahová hodnota a přenosová (někdy také přechodová nebo aktivační) funkce – jedná se o parametry neuronové sítě [26].

  Uzly přijmou data, zpracují je a výsledky předají dalším připojeným uzlům. Data se do systému přivádí prostřednictvím vstupní vrstvy (angl. input layer), která se váže na jednu skrytou vrstvu (angl. hidden layer). Zde se provádí vlastní zpracování a výpočty. Skrytá vrstva poté odkazuje na výstupní vrstvu (angl.output layer) [27]. Grafické zobrazení popsaného vztahu vidíme na obr. č. 1.10.

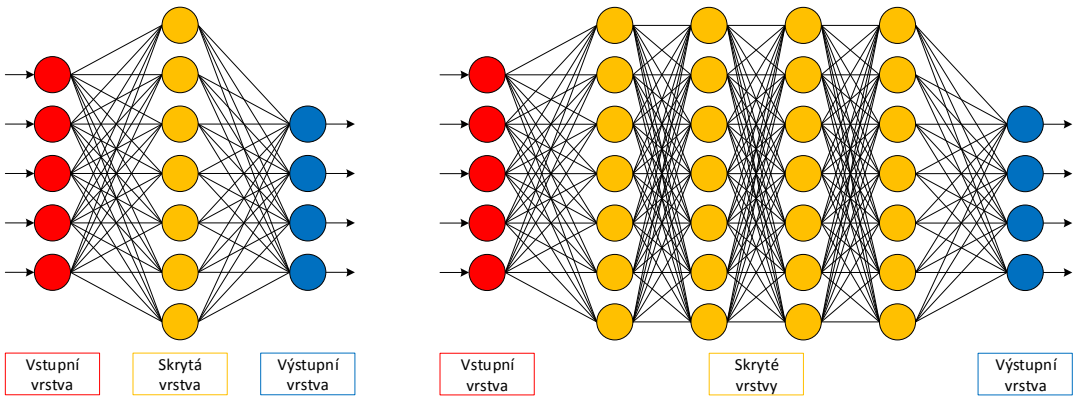
  Proces učení probíhá následovně. Systému poskytneme vstup, u něhož očekáváme odpovídající výstup. Ten se porovná s výstupem neuronové sítě a jejich vzájemný rozdíl se rozloží (nejčastěji pomocí algoritmu zpětného šíření) přes danou síť. Dle výsledku se upraví příslušné váhy a pomocí přenosové funkce se vypočítá tzv. chyba sítě. Cílem je minimalizovat tuto chybu tak, aby model vytvářel správnou odezvu výstupního signálu na signál vstupní [27].

- **Hluboké učení – DL**
  Také známé jako hluboká neuronová síť, je jedním z přístupů k ML. Mezi další patří rozhodovací strom, logická indukce, shluková analýza, bayesovská síť aj.

Skládá se z hierarchie vrstev (zpravidla 2 a více skrytých vrstev). První z nich zpracovává vstupní data a na jejich základě vytváří výstupní funkce, které jsou vstupem pro následující vrstvu. S roustoucím počtem vrstev (nebo počtem perceptronů na skrytou vrstvu) se zvyšuje schopnost vytváření složitějších funkcí. Výstupní vrstva tyto funkce kombinuje a vytváří predikci pro daný vstup. To je základní rozdíl oproti neuronové síti (viz obr. č. 1.10), která neobsahuje více skrytých vrstev a nemůže proto tvořit tak komplexní funkce. Hluboké učení si můžeme představit jako vícevrstvý model neuronové sítě [27].

Hluboké učení vyžaduje obrovské datové sady kvůli složitosti a velikosti svých modelů. Často obsahuje miliony laditelných váhových parametrů. Vzhledem k množství strukturovaných a nestrukturovaných dat i parciálních derivací, které je nutné zpracovat při každé iteraci, potřebuje DL vysoký výpočetní výkon. Zároveň roste časová náročnost tohoto zpracování. Odměnou jsou velmi přesné výsledky v oblasti predikce a klasifikace dat [27].

Umělá inteligence vstoupila do našeho každodenního života v podobě různých nástrojů a aplikací, např. rozpoznávání řeči i obrazu, vyhledávání informací, filtrování spamu, autonomní auta, hudební doporučení (Spotify, YouTube), osobní virtuální asistenti (Cortana od Microsoft, Siri od Apple, Alexa od Amazonu), streamování (Netflix), sociální media (upozornění na reklamy, hashtagy) a celá řada dalších [25].




Obr. 1.10: Rozdíl mezi NN (vlevo) a DL (vpravo). Překresleno dle [27].

Neuronové sítě a systémy hlubokého učení patří k nejperspektivnějším nástrojům AI pro řešení velmi složitých problémů. Aktivní využívání umělé inteligence v podnikových sítích považuji za blízkou a velmi přínosnou budoucnost.

### 1.5.3 Budoucnost sítí

Nejen v podnikových sítích se postupem času předpokládá větší kombinace technologie SDN s umělou inteligencí. Uvnitř společností se budou vytvářet specializovaná oddělení zaměřená na zpracování produkčního datového provozu. Strojové učení bude využívat

tato data k automatizaci některých síťových i pracovních procesů. Tím se bude značně usnadňovat práce zaměstnanců i síťových administrátorů. Mezi již běžné aplikace ML a AI, týkající se softwarově definovaných sítí, patří [11]:

- Automatická optimalizace a regenerace sítě (angl. self-optimizing networks).
- Kybernetická bezpečnost a analýza hrozby (angl. cybersecurity and threat analytics).
- Detekce a správa poruch (angl. fault management).
- Automatizace kognitivních procesů (angl. cognitive process automation).
- Poskytování lepších zákaznických zkušeností (angl. customer experience).

Strojové učení může dále předpovídat anomálie nebo události na základě časově proměnných signálů. Odhaduje, u kterých zařízení by mohlo v následujících dnech, týdnech nebo měsících dojít k poškození. Při kombinaci těchto předpovědí s jinými daty (předcházející sezónní trendy, meteorologické události, špičková zátěž) může AI poskytovat správcům sítí různá doporučení s cílem zmírnit dopad na zákazníky.

Systémy se učí pravidelným opakováním a shromažďováním dat, čímž neustále přizpůsobují své modely. To vede k optimalizaci prostředí, zefektivnění procesů, zlepšení produktivity zaměstnanců a snížení nákladů i chybovosti.

# 2 VIRTUALIZACE SÍŤOVÝCH FUNKCÍ

V této kapitole stručně vysvětlím virtualizaci síťových funkcí NFV (Network Functions Virtualization), představím její definici, architekturu, hlavní vlastnosti a rozeberu vzájemný vztah mezi NFV a SDN.

Přestože virtualizaci síťových funkcí v praktické části nijak nevyužívám, pro všeobecný přehled a ucelení tématiky softwarově definovaných sítí je vhodné tento pojem znát.

## 2.1 Definice NFV

V tradičních sítích jsou různé funkce implementované jako specializované zařízení založené na proprietárním hardwaru se softwarovým operačním systémem. U těchto zařízení nelze softwarová a hardwarová část oddělit.

Virtualizace síťových funkcí přesouvá síťové funkce z proprietárního hardwaru do softwaru běžícího na virtuálních strojích VM (Virtual Machine) standardních průmyslových serverů, tzv. COTS (Commercial off-the-shelf)[1]. Jednotlivé virtuální síťové funkce VNF (Virtual Network Function) umisťujeme dle potřeby na různá místa v síti. Tím eliminujeme nutnost instalace nových proprietárních zařízení. Můžeme konfigurovat i hybridní scénáře, kde funkce běžící na virtualizovaných zdrojích koexistují s funkcemi fyzických zdrojů [28].

Technologie NFV směřuje ke cloudovému modelu a vznikla jako iniciativa průmyslu (poskytovatelé sítí, mobilní operátoři) s cílem zvýšit flexibilitu nasazení nových síťových funkcí a služeb. Výsledkem jsou nižší investiční i provozní náklady, rychlejší spouštění VNF, vyšší návratnost investic, otevřenost virtualizované sítě a více příležitostí k testování nových technologií [29].

### 2.1.1 Myšlenka a motivace

Motivace k využití technologie NFV vyplývá z rostoucího množství proprietárních hardwarových zařízení v sítích, což přináší následující negativní důsledky [28], [30]:

- Nasazení nové síťové funkce nebo služby obvykle vyžaduje instalaci dalšího proprietárního hardwarového zařízení a hledání vhodného fyzického prostoru.
- Nákup nového hardwaru znamená dodatečné investiční výdaje a navýšení současných provozních nákladů na instalaci, správu, provoz i elektrickou energii.
- K těmto nákladům se přidávají další výdaje za školení administrátorů, kteří musí mít dostatečné znalosti nezbytné pro správu hardwarových zařízení.
- Tato zařízení rychle dosahují konce své životnosti, což vede k neustálému opakování cyklu návrh-integrace-nasazení s nízkým ekonomickým přínosem.

[1] Zpravidla se jedná o zařízení vybavená univerzálními procesory jako x86, Power8, ARM aj., namísto výkonných specifických procesorů ASIC [31].

Aby společnosti splnily požadavky neustále se rozvíjejícího síťového prostředí, musí zvyšovat flexibilitu a urychlovat inovace. Výše uvedené skutečnosti brání zavádění nových síťových funkcí a služeb, čímž se snižuje potenciální finanční zisk společností [30].

Cílem NFV je pozměnit způsob návrhu, správy a nasazení síťové infrastruktury pomocí virtualizační technologie. Hlavní myšlenkou je sjednocení mnoha hardwarových zařízení do standardních serverových platforem, přepínačů a úložišť [28].

### 2.1.2 Virtualizovaná zařízení

V pevných i mobilních sítích můžeme pomocí NFV technologie virtualizovat velkou většinu zařízení, služeb a aplikací. Virtuální instance poté nasazujeme v datových centrech, síťových uzlech, koncových bodech apod. Mezi běžně virtualizované síťové komponenty patří [32], [33]:

- **Síťová zařízení:** Nízkovýkonné směrovače a přepínače, NAT, DNS.
- **Mobilní síťová zařízení:** Registry HLR/HSS, brány GGSN/PDN, uzly NB/eNB, jednotky RNC.
- **Zákaznická zařízení:** Funkce v domácích směrovačích a set-top boxech.
- **Tunelovací brány:** IPSec, SSL.
- **Analyzátory provozu:** DPI, měření QoE, monitorování SLA.
- **NGN signalizace:** SBC, IMS.
- **Řídicí zařízení:** Servery AAA, řídicí platformy.
- **Optimalizační zařízení:** CDN, vyrovnávací paměti, vyvažovače zátěže, různé akcelerátory.
- **Bezpečnostní zařízení:** Firewally, proxy servery, systémy detekce narušení, antivirové a antispamové programy.

Standardní serverové platformy teprve nedávno získaly natolik vysoký výkon, aby byly schopné konkurovat proprietárním zařízením v porovnání ceny, spotřeby elektřiny a spolehlivosti. Během posledních let dochází k významnému zvýšení propustnosti, navýšení počtu procesorových jader a urychlení procesorového (x86) zpracování paketů. Zásluhou této evoluce můžeme virtualizovat téměř všechny síťové funkce, mimo ty, které vyžadují velmi vysoké agregované rychlosti (řádově nad 50 Gbps) [34].

### 2.1.3 Specifikace

Hlavní úlohu při vytváření NFV dokumentů má pracovní skupina ISG NFV (Industry Specification Group for Network Functions Virtualization), vytvořená pod záštitou organizace ETSI (European Telecommunications Standards Institute). Byla založena v roce 2012 sedmi předními světovými provozovateli telekomunikačních služeb (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica a Verizon). Členství se od té doby

rozrostlo o dodavatele síťových zařízení, společnosti působící v oblasti síťových technologií, další poskytovatele telekomunikačních i cloudových služeb apod. V současnosti čítá necelých 350 členů [30].

ETSI ISG NFV definuje požadavky a specifikuje hardwarovou i softwarovou architekturu, která je nezbytná pro nasazení virtuálních síťových funkcí. Skupina řídí pokyny pro vývoj a publikuje výsledky PoC (Proof of Concept) svého výzkumu, aby podpořila implementaci NFV v průmyslu. Cílem je sjednocení požadavků a určení technických výzev, které je třeba překonat [29]. Standardizační úkoly řeší pět samostatných skupin [35]:

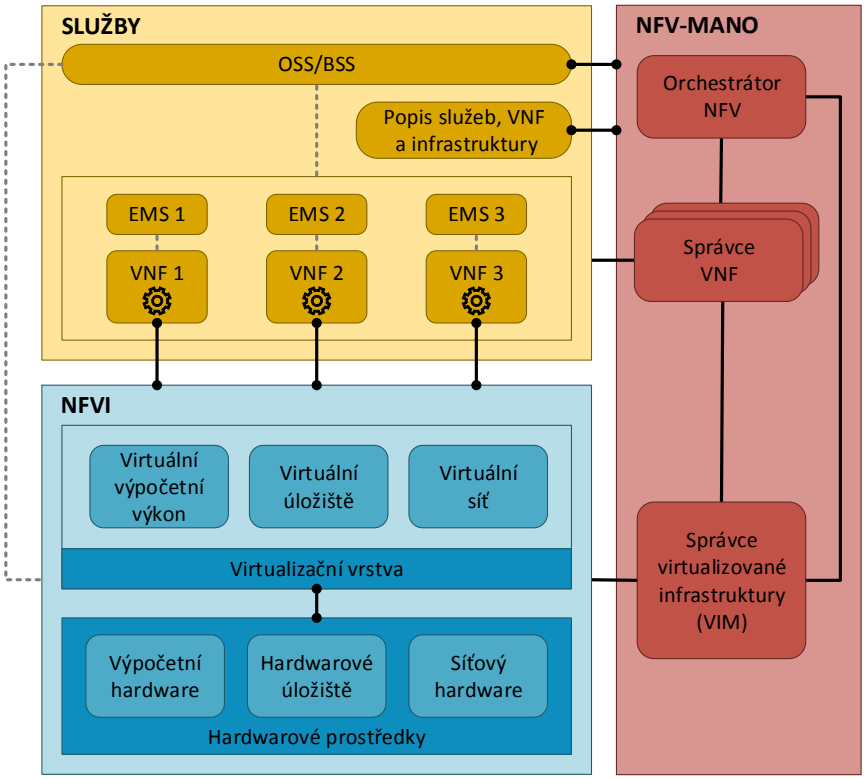
- Virtualizační architektura (angl. Architecture of the Virtualization).
- Správa a orchestrace (angl. Management and Orchestration).
- Softwarová architektura (angl. Software Architecture).
- Bezpečnost (angl. Security Expert Group).
- Výkon a přenositelnost (angl. Perform ance and Portability Expert Group).

Od svého založení vydala skupina více než 100 publikací[2], které zahrnují přednormalizační studie, standardizace, podrobné specifikace, doporučení atd. ISG NFV zveřejnila první sérii specifikací v říjnu 2013. Následně většinu z nich aktualizovala na konci roku 2014 a počátkem roku 2015 [29].

[2]Veškeré dokumenty na webu https://www.etsi.org/.

## 2.2 Architektura NFV

ETSI definuje architekturu NFV vymezením jednotlivých bloků a referenčních bodů mezi nimi, což umožňuje vzájemnou spolupráci odlišných systémů. Detailní popis a vysvětlení všech částí i referenčních bodů nalezneme v oficiální dokumentaci [36] nebo přehledněji v knize [30].




Obr. 2.1: Architektura NFV technologie. Překresleno dle [37].

Tato část představuje architekturu NFV a její princip. Obrázek č. 2.1 zobrazuje jednotlivé komponenty, jejich vzájemné propojení a především základní rozdělení architektury NFV do tří hlavních bloků [28]:

- **Služby**
  Zahrnují sadu virtuálních síťových funkcí VNF (Virtual Network Function), jež jsou implementovány v jednom nebo více virtuálních strojích VM (Virtual Machine) a běží na fyzických i virtualizovaných prostředcích infrastruktury NFVI (Network Functions Virtualization Infrastructure).

  Virtuální síťové funkce spravuje systém řízení prvků EMS (Element Management System). Ten je zároveň odpovědný za jejich tvorbu, konfiguraci, monitorování, výkon a bezpečnost. EMS dále poskytuje informace o VNF pro systém podpory provozu OSS (Operations Support System), který spolu se systémem podpory podnikání BSS (Business Support System) pomáhá poskytovatelům nasazovat a spravovat koncové

služby jako jsou objednávky, fakturace, pravidelná obnova zařízení, řešení vzniklých problémů apod. Patří sem také databáze k ukládání informací a datových modelů, popisující proces nasazení, životní cyklus funkcí, služeb a zdrojů.

- **NFVI**
  Součástí jsou veškeré hardwarové i softwarové zdroje, připojení k datovým centrům a veřejným i soukromým cloudům. Fyzické zdroje obsahují výpočetní, úložný a síťový hardware, který zajišťuje zpracování, ukládání a připojení pro virtuální síťové funkce VNF prostřednictvím virtualizační vrstvy.

  Vytvoření virtualizační vrstvy není konkrétně specifikováno. Můžeme využít například hypervisor[3], jenž jednoduše abstrahuje a rozděluje fyzické prostředky na logické části, které poté přiřazuje jednotlivým VM, na nichž běží VNF. Dalším řešením je zprostředkování virtualizační vrstvy pomocí operačního systému, který přidává vhodný software na hardwarový server a implementuje VNF jako aplikace.

- **NFV-MANO**
  Zaměřuje se na všechny specifické virtualizační úkoly nezbytné v NFV prostředí. NFV-MANO (NFV Management and Orchestration) se skládá z orchestrátoru NFV, správce VNF a správce virtualizované infrastruktury VIM (Virtualized Infrastructure Manager).

  Orchestrátor (nebo-li tzv. koordinátor) zodpovídá za instalaci a konfiguraci nových balíčků síťových služeb NS (Network Service), správu jejich životních cyklů, zajištění zdrojů, ověřování, autorizaci atd. Správce VNF se stará o životní cyklus instancí VNF. Správce VIM přiděluje nebo uvolňuje fyzické i virtualizované prostředky NFVI na základě žádosti z orchestrátoru či správce VNF.

Uvedený popis funkčních bloků lze shrnout následovně. NFVI spolu s VIM poskytuje a spravuje prostředí virtualizovaných zdrojů i základní hardwarové prostředky. VNF zajišťuje softwarovou implementaci síťových funkcí společně se systémem řízení prvků EMS a jedním nebo více VNF správci. Hlavní řídicí částí je orchestrátor NFV, který vytváří kontrolní vrstvu.

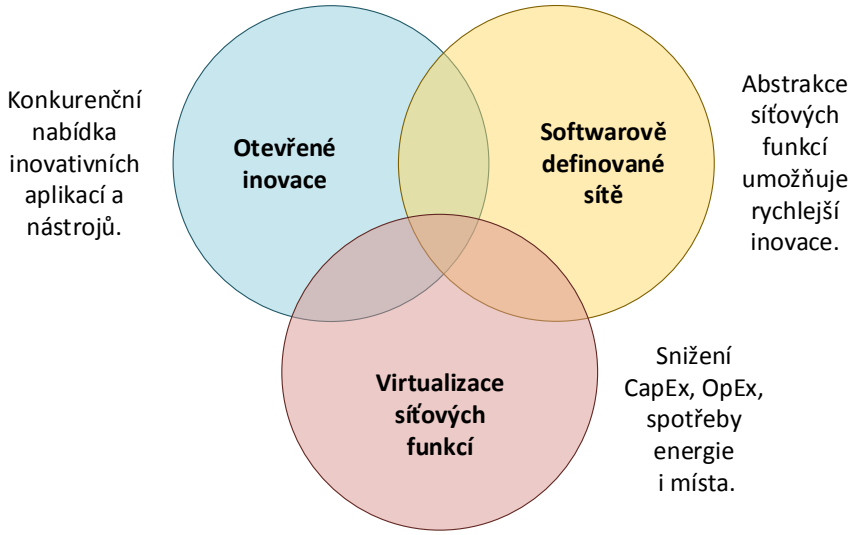
Jednotlivé bloky jsou propojené prostřednictvím definovaných rozhraní (tzv. referenčních bodů), jež se používají pro komunikaci mezi různými částmi NFV architektury. Jejich oddělení poskytuje otevřené a inovativní NFV prostředí.

Referenční body mezi VNF a NFVI (a mezi subjekty v rámci NFVI) zajišťují abstrakci i virtualizaci zdrojů a následné nasazení VNF, což umožňuje výběr hardwarových prostředků dle aktuálních potřeb. Referenční body mezi NFV-MANO a VNF resp. NFVI (stejně jako mezi subjekty v rámci NFV-MANO) se zabývají celkovým řízením a provozem NFV prostředí. Dále jsou dílčí bloky a jejich referenční body navrženy k využívání tradičních systémů řízení sítě (tj. OSS a BSS), což umožňuje součinnost NFV s funkcemi, které běží na starších zařízeních [37].

[3]Hypervisor je program umožňující více virtuálním strojům, operačním systémům nebo aplikacím sdílet jeden fyzický zdroj. Detailněji na webu [38].

## 2.3 Vztah mezi NFV a SDN

Obě technologie jsou úzce spjaté, vzájemně se doplňují, ale nejsou na sobě nijak závislé. Přináší především jednodušší správu, vyšší flexibilitu a rychlejší inovace. Koncept NFV může být implementován bez využití technologie SDN. Jejich kombinací však můžeme dosáhnout lepších výsledků a zvýšit celkový potenciál obou řešení [32].




Obr. 2.2: Vztah NFV s SDN. Překresleno dle [32].

Vzájemný vztah mezi virtualizací síťových funkcí a softwarově definovanou sítí je znázorněn na obr. č. 2.2. Odlišnosti lze obecně popsat těmito body [33]:

- NFV je o virtualizaci síťových zařízení, kdežto SDN je o virtualizaci sítí.
- NFV přenáší funkce síťových služeb z proprietárních fyzických zařízení na virtualizované servery, zatímco SDN odděluje řídicí a datovou část síťového zařízení.
- NFV provozuje síťové funkce virtuálním způsobem, ale SDN přesouvá řídicí funkce do centralizovaného kontroléru a poskytuje programovatelnou architekturu.
- NFV snižuje CapEx, OpEx, spotřebu energie a místa. SDN poskytuje síťové abstrakce funkcí nižších vrstev, což umožňuje flexibilní správu sítě a rychlejší inovace.

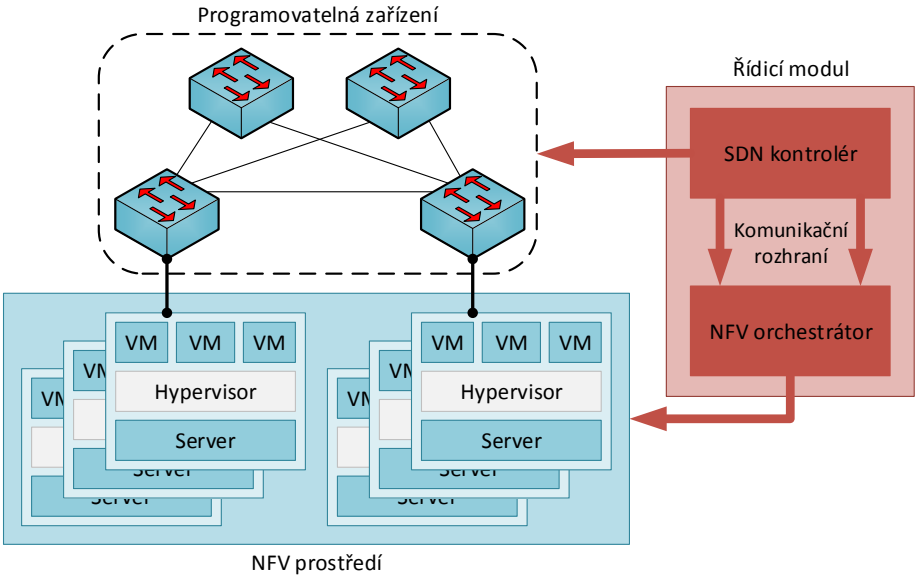
Oddělení řídicí a datové části přináší v NFV prostředí celou řadu výhod. Funkce běží jako instance na standardních serverech, kde dochází k přidělování fyzických i virtualizovaných prostředků jednotlivým službám. Tato skutečnost má vliv na zvýšení výkonu a škálovatelnosti při vynuceném přerozdělování prostředků [32]. SDN dále poskytuje programovatelné síťové rozhraní mezi dílčími VNF, což usnadňuje řízení datového toku a správu těchto instancí [33].

Stejně tak NFV poskytuje prostředky pro softwarově definované sítě, na nichž mohou běžet programovatelná SDN zařízení. Tím je podporována myšlenka použití standardních

serverů namísto velkého množství specializovaného proprietárního hardwaru [32]. Pomocí NFV můžeme virtualizovat i SDN kontrolér, což umožňuje dynamickou migraci řídicího prvku na optimální místo [33].

### 2.3.1 Vzájemná integrace

Softwarově definovaný NFV systém kombinuje virtualizaci a centralizovaný prvek, aby minimalizoval náklady za poskytnutí služby a maximalizoval využití síťových zdrojů. Efektivnější používání zdrojů přináší snížení počtu hardwarových zařízení i síťových procesů. Automatizovaná konfigurace služeb či funkcí výrazně snižuje časovou náročnost správy sítě a zároveň snižuje pravděpodobnost lidské chyby, což nabízí lepší škálovatelnost. Na druhou stranu, nasazení a poskytování nových typů služeb obvykle vede k dlouhému a opakovanému procesu ověřování a testování. Automatizací NFV prostředí můžeme tuto dobu nasazení výrazně zkrátit a snížit tak provozní náklady. Informace pro tuto podkapitolu jsem získával z odborného článku [33].




Obr. 2.3: Softwarově definovaný NFV systém. Překresleno dle [33].

Systém je znázorněn na obr. č. 2.3. Skládá se z řídicího modulu, programovatelných zařízení a NFV prostředí. Kontrolér zajišťuje řízení sítě a správu prvků stejně jako v běžné SDN architektuře. V NFV prostředí jsou využívány standardní servery k implementaci virtuálních síťových funkcí VNF. Na serverech fungují hypervisory pro podporu virtuálních strojů VM, na nichž běží jednotlivé VNF instance. NFV prostředí tedy poskytuje přizpůsobitelnou a programovatelnou datovou část nahrazující funkce hardwarového firewallu, IDS, proxy serveru atd.

SDN kontrolér a NFV orchestrátor společně tvoří řídicí modul. Kontrolér řídí orchestrátor prostřednictvím standardního komunikačního rozhraní. Na základě definovaných zásad

i topologie sítě vypočítá řídicí modul optimální přiřazení VNF určitým VM a vytvoří optimalizované směrovací cesty. Nasazení vybraných VNF zajistí NFV orchestrátor. SDN kontrolér poté směřuje datový provoz přes určitou posloupnost VM a síťových zařízení, čímž se dosahuje požadovaných výsledků.

## 2.4 Vlastnosti

Jako každá technologie i virtualizace síťových funkcí přináší řadu výhod a nevýhod. Důležitým předpokladem pro úspěšnou realizaci NFV je splnění definovaných požadavků, s čímž souvisí i jistá bezpečnostní rizika a výkonová omezení. V této sekci si některé z uvedených otázek přiblížíme.

### 2.4.1 Výhody

Efektivní implementace NFV technologie poskytuje nejen provozovatelům sítí mnoho výhod, mezi které patří [30], [32]:

- **Snížení CapEx a OpEx**
  Využívání standardních serverových platforem, sjednocování zdrojů a podpora modelu „pay-as-you-grow“ odstraňuje plýtvaní fyzickým hardwarem, energií i místem. Tyto skutečnosti snižují investiční náklady na nákup specializovaných proprietárních zařízení. Zároveň snižují i provozní náklady na udržování a správu rozmanitých síťových zařízení různých výrobců.
- **Rychlá inovace**
  Zkracuje se čas potřebný k nasazení nových síťových funkcí i služeb, což umožňuje rychle plnit obchodní požadavky, zákaznické potřeby a tržní příležitosti. Tím se zlepšuje návratnost investičních cyklů.
- **Otevřený systém**
  Snadno dostupná virtualizovaná zařízení urychlují vývoj a testování funkcí či služeb. Tím se snižuje riziko spojené s nasazením nových produktů. Zásluhou definovaných komunikačních rozhraní se usnadňuje vzájemná spolupráce různých systémů.
- **Flexibilita**
  Jednoduchou správou funkcí i služeb můžeme rychle reagovat na měnící se požadavky zákazníků. Také je možné zavést cílené služby dle aktuálních geografických nebo zákaznických informací. NFV umožňuje použití stejné serverové platformy pro různé funkce, služby i zákazníky.
- **Optimalizace energie**
  Virtualizační techniky umožňují přesunutí pracovní zátěže (obvykle mimo špičku, angl. off-peak) na menší počet serverů. Nevyužité servery můžeme buď úplně vypnout, popřípadě přepnout do úsporného režimu.

### 2.4.2 Požadavky a výzvy

K dosažení výše uvedených výhod musí být technologie NFV navržena a implementována tak, aby splňovala řadu požadavků i technických výzev [30], [32]:

- **Přenositelnost a integrace**
  Nasazení virtuálních síťových funkcí na různých serverových platformách. Musí být definováno jednotné rozhraní oddělující softwarové instance od základního hardwaru. Zároveň musí být umožněna kombinace serverů, hypervisorů i virtualizovaných zařízení různých dodavatelů. Na problém přenositelnosti se zaměřuje odborný článek [31].
- **Výkon**
  Zajištění dostatečného výkonu, jelikož technologie NFV využívá standardní průmyslové servery namísto specializovaných proprietárních zařízení. Musí být zaručeno co nejnižší výkonnostní omezení, aby se minimalizoval vliv na zpoždění, propustnost a vlastní zpracování.

  Získání vysokého výkonu standardních serverů je již několik let oblastí kreativity a inovací v IT průmyslu. Protože se očekává, že aplikační software bude nezávislý na hardwarové platformě, nemůže zahrnovat knihovny specifické pro daný typ hardwaru. Navíc přítomnost hypervisoru mezi aplikacemi a hardwarovými prostředky zabraňuje plnému přístupu ke všem možnostem fyzického zdroje. Pro zvýšení výkonu VNF existují softwarové i hardwarové akcelerační techniky, jimiž se detailně zabývají odborné články [31] a [39].
- **Alokace zdrojů**
  V architektuře NFV je síťová služba souborem zřetězených virtuálních síťových funkcí. Proto musí být dosaženo rychlého, škálovatelného a dynamického složení těchto VNF. Správu a řízení služeb zajišťuje NFV orchestrátor.

  Efektivní přidělování zdrojů je jednou z hlavních výzev pro účinné nasazení jednotlivých VNF. Této problematice se věnuje odborný článek [28].
- **Bezpečnost**
  Spouštění jednotlivých virtuálních funkcí nebo služeb nesmí narušit bezpečnost a dostupnost NFV architektury. Virtuální zařízení by mělo být stejně bezpečné jako fyzické zařízení. Proto je důležité využívat certifikovaných hypervisorů.

  Současné bezpečnostní otázky řeší odborný článek [40]. Věnuje se nejen útokům na samotné VNF, ale i bezpečnostním hrozbám v softwarově definované NFV.
- **Automatizace**
  Nasazení virtuálních síťových funkcí musí být plně automatizované. Rychlé obnovení VNF může vylepšit odolnost a dostupnost služeb.
- **Řízení a orchestrace**
  Jednoduché systémy pro koordinaci a řízení virtuálních funkcí, fyzických serverů, virtualizované infrastruktury aj.

- **Kompatibilita**
  Podpora proprietárních fyzických síťových zařízení a jejich kompatibilita s otevřenými virtuálními síťovými prvky.
- **Stabilita a jednoduchost**
  Řízení velkého množství virtualizovaných zařízení nesmí mít vliv na stabilitu sítě. S tím souvisí požadavek na jednoduchost serverových platforem.

Technologie NFV s sebou přináší celou řadu otázek, které musíme řešit. Jedná se však o poměrně novou technologii, tudíž dochází k neustálým inovacím, pečlivému vývoji a odstraňování problémů. Nezanedbatelnou výhodou je, že se na rozvoji podílí množství významných světových společností a poskytovatelů služeb.

# 3 NÁSTROJE PRO SDN

V diplomové práci se zabývám využitím SDN technologie v podnikových sítích, proto se zaměřím na produkty tohoto odvětví. Pro datacentrová prostředí je k dispozici celá řada dalších nástrojů, které zde však nebudu uvádět.

## 3.1 Přepínače

Přepínače podporující SDN mohou být softwarového i hardwarového typu. Softwarový SDN přepínač bývá implementován ve standardní serverové platformě a umožňuje spojení mezi síťovými prvky a aplikacemi. Hardwarový SDN přepínač bývá hybridním zařízením, jež obstarává klasické přepínání rámců a podporuje protokol OpenFlow. V následujících dvou podkapitolách uvedu některé z dostupných produktů.

### 3.1.1 Softwarové SDN přepínače

V tabulce č. 3.1 jsou uvedeny softwarové SDN přepínače, vhodné pro podnikové LAN sítě, kde některé z nich jsou komerční, ostatní open-source.

Tab. 3.1: Dostupné softwarové SDN přepínače.

| **Název** | **Společnost** | **Dostupnost** |
|---|---|---|
| B4N SwitchOS | Brain4Net | Komerční |
| BESS | Berkeley NetSys Lab | Open-source |
| CPqD | CPqD | Open-source |
| Indigo | Project Floodlight | Open-source |
| Lagopus Switch | Lagopus | Open-source |
| LINC Switch | FlowForwarding | Open-source |
| OcNOS | IP Invision | Komerční |
| Open vSwitch | Linux Foundation | Open-source |
| OpenContrail vSwitch | Tungsten Fabric | Open-source |
| PF1000 | NEC | Komerční |
| PicOS | Pica8 | Komerční |
| Snabb Switch | Snabb | Open-source |
| Switch Light | Big Switch Networks | Open-source |
| VortiQa Switch | NXP Semiconductors | Komerční |
| VPP | FD.io | Open-source |
| ZeroTier Switch | ZeroTier | Komerční |

Aktualizovaný seznam open-source SDN nástrojů (softwarové přepínače, směrovače, kontroléry, simulátory, emulátory, operační systémy atd.) nalezneme ve webovém repozitáři GitHub[1]. Z uvedeného webu jsem vycházel při tvorbě této podkapitoly. Srovnání některých softwarových SDN přepínačů nalezneme v odborném článku [41].

[1]Dostupné na webu https://github.com/sdnds-tw/awesome-sdn.

### 3.1.2 Hardwarové SDN přepínače

V tabulce č. 3.2 jsou vypsáni významní výrobci hardwarových SDN přepínačů a seznam jejich jednotlivých produktů. Tyto přepínače jsou vhodné pro podnikové LAN sítě.

Tab. 3.2: Dostupné hardwarové SDN přepínače.

| **Společnost** | **Produkty** |
|---|---|
| Alcatel-Lucent | OmniSwitch – 9900, 6900, 6865, 6860, 6560, 6465, 6450, 6350 |
| Allied Telesis | SwitchBlade – x8100, x908; Series – x950, x930, x610, x550, x530, x510, x310 |
| Arista | 7500, 7300, 7280R, 7250X, 7160, 7050X |
| Aruba Networks | 5400R, 3810, 2930M, 2930F, 2920, 2540, 2530 |
| Centec | V580, V350, V330, V150 |
| Cisco Systems | Catalyst – 9600, 9500, 9400, 9300, 9200, 6800, 6500, 4500E, 3850, 3650, 3560-CX, 2960-L |
| Dell EMC | N4000, N3000, N2000, N1500, N1100 |
| Edgecore Networks | AS – 7700, 5800; ECS – 4620, 4510, 4210, 4120, 4100, 3500 |
| Extreme Networks | BlackDiamond – X8, 8000, 7100; Series – X870, X770, X690, X590, X465 |
| H3C | S7500X, S6520X, S5560S, S5130S, S3100V3 |
| HP Enterprise | Altoline – 6960, 6940, 6920, 6900; Series – 10500, 8200, 5500, 5400, 5130, 3800, 2920 |
| Huawei Technologies | S12700, S9700, S7700, S6720-HI, S5730-HI, S5720-HI |
| Juniper Networks | QFX – 10016, 10008, 10002, 5200, 5100; EX9200 |
| NEC | PF5248, PF5240 |
| Nuage Networks | 7850 NSG-E200/300 series, 7210 SAS |
| Pica8 | P5401, P5101, P3930, P3922, P3297 |
| Quanta | T5032, T5016, T3048, T3040, T1048 |

Uvedený seznam hardwarových SDN přepínačů není zdaleka konečný. Existuje řada dalších dodavatelů i jiných typů SDN produktů, např. směrovače, přístupové body, bezdrátové kontroléry, operační systémy apod. Při tvorbě této podkapitoly jsem vycházel především z odborného článku [42]. Zde jsou uvedeny i odkazy na webové stránky jednotlivých výrobců, kde jsem vyhledával nabídku aktuálních produktů, jejichž portfolio se neustále rozšiřuje.

## 3.2 Komerční kontroléry

V současné době se vývojem SDN kontrolérů zabývají téměř všichni významní výrobci síťových zařízení jako Cisco Systems, HP Enterprise, Huawei Technologies, Juniper Networks atd. Přehled známých komerčních kontrolérů pro podnikové sítě je znázorněn v tabulce č. 3.3. Většina z nich je inspirována open-source projekty.

Tab. 3.3: Dostupné komerční SDN kontroléry.

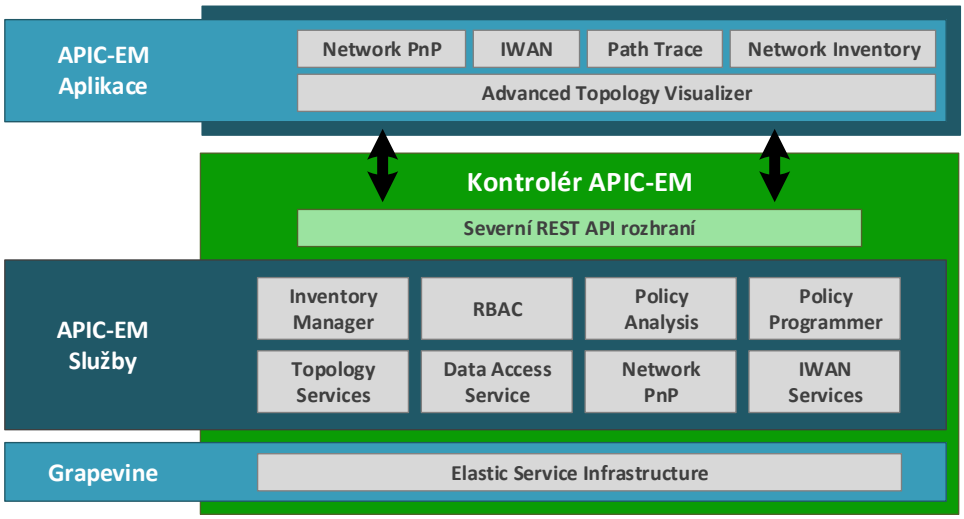
| **Název** | **Společnost** | **Jižní rozhraní** |
|---|---|---|
| Agile Controller-Campus | Huawei Technologies | NETCONF, SSH, HTTPS |
| Altiplano | Nokia | SNMP, NETCONF |
| APIC-EM | Cisco Systems | CLI, SNMP, NETCONF |
| DNA-C | Cisco Systems | CLI, SNMP, NETCONF |
| DR2000 ADCampus Director | H3C | OpenFlow |
| NorthStar | Juniper | BGP, PCEP, NETCONF, SNMP |
| OneController | Extreme Networks | OpenFlow, OVSDB, SNMP, XML |
| PF6800 SDN Controller | NEC | OpenFlow |
| SD-Branch | Aruba Networks | OpenFlow |
| VAN SDN Controller | HP Enterprise | OpenFlow |
| Vista Manager EX | Allied Telesis | SNMP |
| VortiQa | NXP | OpenFlow |

Nezávislých srovnání komerčních SDN kontroléru pro datové LAN sítě není mnoho. Hlavním důvodem je, že v tomto prostředí se technologie SDN začala prosazovat teprve nedávno. Například pro sítě datových center můžeme nalézt celou řadu srovnávání jednotlivých SDN kontrolérů, kde se porovnává výkonnost, účinnost, cena, uživatelská přívětivost atd. Proto zde uvádím alespoň nezávislé hodnocení společnosti Miercom, která ve zprávě [43] srovnává aktuální řešení společností Cisco Systems, Huawei Technologies a Aruba Networks (dceřinná společnost HP Enterprise) pro podnikové sítě. Porovnávají se v ní tři klíčové oblasti – automatizace sítě, segmentace sítě, monitorování a správa sítě. Z uvedeného srovnání vychází nejlépe řešení společnosti Cisco Systems, konkrétně Cisco DNA-C.

### 3.2.1 Cisco APIC-EM

Cisco APIC-EM (Application Policy Infrastructure Controller – Enterprise Module) je SDN kontrolér pro drátové i bezdrátové podnikové sítě, jež zajišťuje automatizaci síťové infrastruktury a abstrakci síťových funkcí. Mezi využitelné aplikace kontroléru patří:

- **PnP (Plug and Play)** – Zajišťuje nasazení nových zařízení do sítě bez předchozí konfigurace (angl. zero-touch deployment). V kontroléru je vytvořeno pravidlo (název, sériové číslo, platforma, operační systém aj.) pro konkrétní zařízení. Jakmile zařízení kontaktuje kontrolér, přiřadí se mu jeho připravené pravidlo na základě unikátního sériového čísla. Následně proběhne automatické nasazení (angl. provisioning). Více podrobností v dokumentaci PnP [44].
- **Path Trace** – Umožňuje simulaci datového toku mezi dvěma uzly v síti, čímž můžeme ověřit jejich vzájemnou konektivitu. Aplikace sbírá data z jednotlivých zařízení na definované trase (MAC tabulky, směrovací tabulky, přístupové listy). Tím určí, jakým způsobem bude datová jednotka přenášena v síti. Veškeré informace se zobrazí




Obr. 3.1: Architektura kontroléru Cisco APIC-EM. Překresleno dle [45].

uživateli, společně se statistikami vytíženosti a ztrátovosti jednotlivých rozhraní. To umožňuje interaktivní odstraňování problémů v síťové infrastruktuře.

- **EasyQoS** – Usnadňuje aplikaci QoS pravidel dle definovaných zásad společnosti. Síťové aplikace jsou rozděleny do tří kategorií – významné (angl. relevant), výchozí (angl. default), nepodstatné (angl. irrelevant). Následně jsou pravidla aplikována na jednotlivá zařízení v určeném rozsahu sítě.
- **Topology** – Zobrazuje síťovou topologii včetně připojených koncových zařízení a poskytuje základní informace jako např. název zařízení, platforma, IP adresa, operační systém apod.

Architekturu kontroléru APIC-EM vidíme na obr. č. 3.1. Všechny funkce kontroléru jsou dostupné prostřednictvím severního REST API rozhraní, což umožňuje integraci vlastních nástrojů či aplikací. Ke komunikaci se zařízeními datové části se používá jižní rozhraní s protokoly CLI, SNMP a NETCONF.

Kontrolér APIC-EM je dostupný zdarma, což je jeho největší předností. Jedná se o softwarový nástroj, který lze instalovat na standardní fyzický server i do virtuálního prostředí VMware ESXi. Musíme však splnit tyto minimální systémové požadavky:

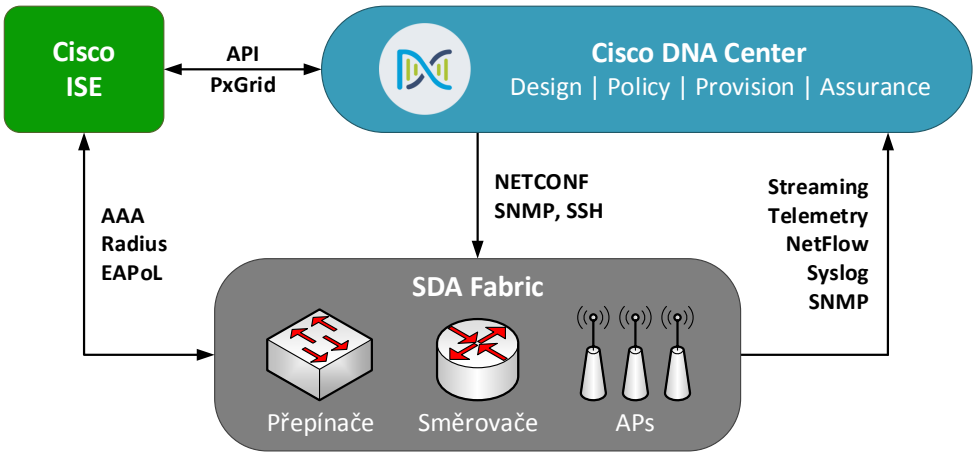
- **Server**: 64bitový x86
- **Procesor**: 6-jader, rychlost 2,4 GHz
- **RAM**: 64 GB
- **Úložiště**: 500 GB

Další informace nalezneme v dokumentaci výrobce [46], ze které jsem vycházel při zpracování této podkapitoly.

### 3.2.2 Cisco DNA-C

Cisco DNA-C (Digital Network Architecture – Center) je centralizovaná hardwarová platforma zajišťující automatizaci podnikových LAN, WLAN i WAN prostředí. Jde o nástupce kontroléru Cisco APIC-EM.

Důležitým poznatkem je, že kontrolér DNA-C ve své vnitřní architektuře zahrnuje, kromě funkčních softwarových balíčků, také APIC-EM jako součást zprostředkující komunikaci mezi grafickým uživatelským rozhraním DNA-C a síťovými zařízeními. Dohromady tak přináší řídicí nástroj pro automatické nasazení síťových prvků, segmentaci sítě, aktivní monitoring, řešení problémů (angl. troubleshooting) a správu celé síťové infrastruktury. Všechny funkce kontroléru jsou dostupné prostřednictvím severního REST API rozhraní. Integrace dalších nástrojů a aplikací umožňuje rychlejší zavádění nových služeb, čímž se zvyšuje flexibilita sítě. Ke komunikaci se zařízeními datové části se používá jižní rozhraní s protokoly SNMP a NETCONF [47].




Obr. 3.2: Řešení Cisco DNA-C. Překresleno dle [47].

Jak můžeme vidět na obr. č. 3.2, pojem DNA-C představuje rovněž celkové architektonické řešení sítě, které s sebou přináší termín SDA (Software Defined Access). Ten výrazně mění pohled na tradiční datové sítě a lze shrnout do těchto bodů [48]:

- Pomocí technologie virtuálních sítí VXLAN je nad fyzickou topologií sítě (angl. underlay) vytvořena logická síť (angl. overlay, fabric).
- „Underlay“ je tradiční datová síť, kterou tvoří fyzická síťová zařízení (přepínače, směrovače, bezdrátové kontroléry aj.). V tomto konceptu může být jednoduchá bez použití bezpečnostních prvků. „Overlay“ je softwarově definovaná síť, kde uplatňujeme veškerá pravidla jako řízení přístupu k síti, bezpečnostní zásady apod.
- Na základě ověření identity protokolem IEEE 802.1X jsou koncová zařízení přiřazena do patřičných virtuálních sítí. Veškerý datový provoz těchto zařízení je poté směrován.

- Přináší jednotnou drátovou a bezdrátovou síť, kde správce může oddělovat datový provoz jednotlivých uživatelů do různých virtuálních sítí, případně i v rámci jedné virtuální sítě pomocí tzv. mikrosegmentace.
- Zásluhou logické IP adresace je uživatel z pohledu koncového zařízení (telefon, notebook) připojen stále na stejném logickém místě, přestože se pohybuje a jeho fyzické umístění se mění. Ať se tedy uživatel nachází v rámci sítě kdekoli, jeho IP adresa je stále stejná. Tato skutečnost výrazně usnadňuje mobilitu uživatelů.
- Využívá záměru IBN (Intent-Based Networking), kde správce definuje čeho chce dosáhnout, nikoliv jak toho dosáhnout. Veškerou síťovou konfiguraci provede kontrolér DNA-C, což při realizaci konkrétních změn přináší výraznou časovou úsporu a především minimalizaci chyb.
- Prostředí je dále možné rozšířit o nástroj Cisco ISE (Identity Services Engine), který předává bezpečnostní informace o zařízeních, uživatelích, hrozbách a zranitelnostech do grafického uživatelského rozhraní kontroléru DNA-C.

Kontrolér DNA-C je v současné době velmi aktivně řešená platforma a dochází k jejímu neustálému vývoji. Jedná se o klíčovou technologii společnosti Cisco Systems, která by měla udávat krok v SDN přístupu pro LAN i WAN sítě a představovat nový pohled na budování moderních datových sítí. Další informace nalezneme například v dokumentaci výrobce [49], popřípadě na vývojářském webu [50].

## 3.3 Open-source kontroléry

Nejznámější open-source SDN kontroléry použitelné pro podnikové sítě jsou vypsány v tabulce č. 3.4. Součástí tabulky je i programovací jazyk a jižní rozhraní uvedených kontrolérů.

Tab. 3.4: Dostupné open-source SDN kontroléry.

| **Název** | **Jazyk** | **Jižní rozhraní** |
|---|---|---|
| Floodlight | Java | OpenFlow |
| Kandoo | Go | OpenFlow |
| Maestro | Java | OpenFlow |
| ONOS | Java | OpenFlow, OVSDB, NETCONF, BGP |
| OpenContrail | Java/Python | BGP, XMPP, NETCONF |
| OpenDaylight | Java/Python | OpenFlow, BGP, PCEP, NETCONF |
| OpenMUL | C/Python | OpenFlow, OVSDB, NETCONF |
| OPNFV | Java/Python | OpenFlow |
| POX | Python | OpenFlow, OVSDB |
| Ryu | Python | OpenFlow, OVSDB, NETCONF |
| Ryuretic | Python | OpenFlow |
| Trema | C/Ruby | OpenFlow |

Síla otevřené vývojové komunity má vliv nejen na velmi rychlý rozvoj open-source SDN kontrolérů, ale zároveň i na množství dostupných informací, nezávislých srovnání a studií. Důkladný rozbor vlastností představených kontrolérů však není cílem této práce. Uvedu zde alespoň několik informačních zdrojů, které hodnotí, porovnávají a testují některé z uvedených open-source SDN kontrolérů.

V odborném článku [51] jsou představeny hlavní funkce open-source kontrolérů a pomocí nástroje Cbench se testuje jejich výkonnost, propustnost a zpoždění. Další odborný článek [52] popisuje celou řadu open-source kontrolérů, u nichž se porovnává jejich architektura a výkonnostní charakteristiky. Celkový souhrn problematiky SDN technologie, současný stav a nové výzvy přináší odborný článek [53]. V něm se uvádí, že 61 % aktivních SDN nasazení spravuje OpenDaylight kontrolér. Za ním následuje s 23 % ONOS kontrolér.

### 3.3.1 OpenDaylight

Dle diplomové práce [54] se jedná o nejrozšířenější open-source SDN kontrolér pro automatizaci sítě jakékoli velikosti a rozsahu. Nabízí centralizovanou správu sítě, dynamickou optimalizaci, rychlé nasazení síťových služeb apod.

Projekt ODL (OpenDaylight) vznikl z myšlenky SDN s cílem zaměřit se na flexibilitu sítě pomocí programového přístupu. Byl představen v roce 2013 a původně jej vedly společnosti IBM a Cisco Systems. Poté přešel pod záštitu společnosti Linux Foundation, získal širokou podporu veřejné komunity a dnes patří k nejoblíbenějším open-source SDN kontrolérům. Často se používá jako základ pro komerční řešení.

Architektura ODL je založena na mikroslužbách, které slouží k řízení aplikací, protokolů a zásuvných modulů. Interakci mezi aplikacemi a kontrolérem zajišťuje severní REST API rozhraní. Zařízení s podporou OpenFlow se připojují ke dvěma nebo více VM prostřednictvím TCP. Kontrolér ODL dále podporuje tradiční protokoly jako NETCONF, BGP, PCEP a OVSDB.

### 3.3.2 ONOS

Kontrolér ONOS (Open-Source Network Operating System) slouží k vytváření výkonných softwarově definovaných sítí. Byl představen v roce 2014 a do dnešního dne následovalo dalších 10 vydání. Jádro ONOS a jeho základní služby jsou psány v jazyce Java jako dílčí svazky, které se načítají do softwarové struktury Karaf OSGi. Protože ONOS běží v Java VM, může být implementován v různých serverových platformách. Podporované protokoly jižního rozhraní jsou OpenFlow, NETCONF, BGP a OVSDB. Pro integraci vlastních aplikací poskytuje kontrolér severní REST API rozhraní.

ONOS podporuje různé aplikace pro řízení, konfiguraci a správu sítě. Patří sem například Device Subsystem (spravuje síťová zařízení datové části), Link Subsystem (spravuje propojení jednotlivých zařízení), Host Subsystem (spravuje koncové uživatele a jejich umístění v síti), Topology Subsystem (zobrazuje síťovou topologii), Flow Rule Subsystem (spravuje pravidla datového provozu) aj. Tato podkapitola vychází z diplomové práce [54].

## 3.4 Komerční vs. open-source

Open-source produkty se vyvíjí pozoruhodným tempem, jelikož umožňují spoluprací otevřené vývojové komunity, jejíž cílem je řešit vzniklé výzvy a vytvářet efektivní řešení. Do open-source projektů jsou rovněž zapojeny výzkumné týmy univerzit po celém světě. Avšak otevřená řešení již dávno nejsou pouhou oblastí výzkumu [54].

Volba vhodného SDN kontroléru se pro každou síť liší. Vždy je nutné stanovit přesné požadavky, např. výkon, funkce, vlastnosti, cena atd. Varianta využití open-source produktu se často nabízí jako ideální především z důvodu, že investiční náklady jsou nulové. Mezi administrátory dané sítě by však měl být člověk, který takovému produktu detailně rozumí, bude se intenzivně starat o síťovou infrastrukturu a především zaručí provoz bez výpadků. Takový zaměstnanec je obvykle vysoce finančně nákladný. Společnosti, které takovým zaměstnancem nedisponují, si často nemohou dovolit hodinové či denní výpadky. Proto spíše volí komerční řešení s podporou, kdy jakýkoli problém nahlásí dodavateli konkrétního produktu, který musí zaručit nápravu dle stanovených SLA (Service Level Agreement).

Tyto skutečnosti jsou důležité pro správnou volbu jakéhokoli produktu. Každá společnost má jiné priority a je nutné dopředu zvážit, zda se open-source řešení opravdu vyplatí, či naopak zda komerční řešení není při nízkých požadavcích příliš nákladné.

# 4 NÁVRH SOFTWAROVÉHO NÁSTROJE

Praktickou částí diplomové práce je identifikovat současné problémy v řízení i správě složitých datových sítí a s využitím konceptu SDN, na bázi zvoleného kontroléru, vytvořit nástroj zjednodušující některý z těchto problémů. Tato kapitola tedy popisuje danou analýzu, následné kroky při návrhu softwarového nástroje, jeho architekturu a jednotlivé stavební bloky s vysvětlením fundamentálních objektů, metod i funkcí.

## 4.1 SDN v praxi

Rostoucí počet síťových zařízení vede ke složitější správě sítě pomocí populárního příkazového řádku CLI (Command Line Interface), nebo-li tzv. „konzole“. Administrátoři ovládají a konfigurují každý prvek zvlášť, což přináší značnou časovou zátěž a zvyšuje pravděpodobnost pochybení. Proto často přistupují k vytváření jednoduchých skriptů, které danou činnost automatizují.

V současné době je automatizace silně rozšířena především v datacentrovém prostředí. Díky ní můžeme provozovat datová centra obsahující obrovský počet fyzických serverů, na nichž běží tisíce virtuálních strojů VM. Takové prostředí již není možné spravovat pouze pomocí běžné konzole. Řešením jsou komerční i open-source nástroje, které usnadňují nasazení a správu jednotlivých virtuálních instancí pomocí grafického uživatelského rozhraní GUI (Graphical User Interface). Zároveň zajišťují softwarové řízení celé datacentrové infrastruktury. Konfigurace nového VM je pak otázkou několika kliknutí a samotné spuštění trvá jen pár minut.

Infrastruktura datových center je obvykle jednodušší a poměrně homogenní, navíc většinu datového provozu tvoří vlastní komunikace uvnitř datacentra, což značně usnadňuje definici i následné vynucování zásad. Naopak datové sítě jsou spíše heterogenní a jejich datový provoz je velmi rozmanitý. Tyto skutečnosti zpomalují automatizaci v rámci datových sítí. Přesto se technologie SDN již běžně používá ve velkých páteřních WAN sítích a pomalu se začíná prosazovat také v menších podnikových LAN sítích.

Během posledních dvou let můžeme pozorovat velký nárůst technických školení zaměřených na SDN technologii. Také portfolio produktů jednotlivých výrobců více cílí do oblasti podnikových sítí, kde je SDN ve svém počátku. Avšak několikaleté budování sítí tradičním způsobem vytváří prostředí, které je sice statické, ale rovněž stabilní a odolné. Navíc správci sítí jsou často konzervativní k novým konceptům a nevyžadují přílišné změny uvnitř svých sítí. Dalším aspektem jsou vysoké počáteční náklady na nákup SDN zařízení, licencí i řídicího prvku. Správci sítí obvykle nemají potřebné finance, popřípadě nevidí dostatečný užitek vůči zaplacené ceně. S nasazením nové technologie dále souvisí potřeba učit se novým věcem.

Z těchto důvodů jsem se rozhodl vytvořit nástroj, který využívá myšlenky SDN i funkcí síťového kontroléru, abych správcům sítí předvedl výhody tohoto řešení a pomohl tak změnit jejich odmítavý postoj vůči softwarově definovaným sítím.

## 4.2 Identifikace problémů

Správa sítě obsahující velký počet zařízení patří mnohdy k časově náročným úkonům vyžadujícím široké znalosti různých technologií, protokolů, operačních systémů odlišných výrobců apod. Každé IT oddělení či jednotliví technici se ve své síti potýkají s určitým typem problémů, které jsou často velmi specifické dle kombinace využívaných technologií či produktů různých firem. Zde narážíme především na jedno z nejdiskutovanějších témat, což je „single-vendor“ vs. „multi-vendor“ prostředí.

V odvětví IT dochází k neustálému vývoji a inovaci, kde se každý výrobce snaží předčit svoji konkurenci a nabídnout zákazníkům řešení, které šetří náklady a zároveň zvyšuje výkonnost. Především kvůli omezenému rozpočtu většiny firem nastává situace, kdy jsou jednotlivé technologie implementovány různými výrobci síťových zařízení. Pořízení nejlepšího softwaru od jednoho výrobce a jeho zprovoznění na nejkvalitnějším hardwaru jiného výrobce se může zdát jako ideální řešení, jak zajistit nejrychlejší a nejsilnější IT prostředí za nejnižší možnou cenu. Avšak realita je často dosti odlišná. Výkon řešení je vždy limitován platformou, na které běží. Zpočátku může takový systém fungovat skvěle, ale postupem času se díky softwarovým „upgradům“ či bezpečnostním „patchům“ dostáváme do nepředvídatelných stavů. Řešení vzniklých problémů mezi různými výrobci je poté daleko komplikovanější, časově zdlouhavé a tím i finančně náročnější. Uvedený příklad je však spíše extrémní situací, ke které nemusí vůbec dojít. Pokud se rozhodneme pro kombinované IT prostředí, musíme analyzovat nejen pořizovací cenu, ale také náklady na údržbu dané technologie a přizpůsobení celé infrastruktury. Až poté dokážeme lépe odhadnout, zda se nižší počáteční investice opravdu vyplatí [55].

Abych dokázal lépe odhalit problémy a spletitosti správy sítě reálných provozovatelů, oslovil jsem významnější zákazníky firmy Networksys a.s., což je přední česká IT společnost zabývající se nejnovějšími technologiemi pro datová centra, softwarově definované a počítačové sítě, zejména zaměřená na síťová zařízení firmy Cisco Systems. Důvodem tohoto rozhodnutí je stále probíhající pracovní stáž ve zmiňované firmě. Na základě osobních schůzek či emailové komunikace jsem sesbíral řadu aktuálně řešených problémů a také nápadů pro zjednodušení i automatizaci. Zde uvádím vybrané z nich:

- Synchronizace VLAN mezi podnikovou částí a datacentrovou částí jedné síťové infrastruktury.
- Automatizace funkce CoA (Change of Authorization) při nepřípustném chování klienta uvnitř sítě.
- Zjednodušení přidávání většího počtu telefonních zařízení do CUCM (Cisco Unified Communications Manager).
- Pravidelné porovnávání aktuálních konfigurací síťových prvků proti definované šabloně a případné upozornění na inkonzistence.
- Možnost zpětného ověření konkrétního příkazu na vybraných zařízeních uvnitř síťové infrastruktury.

- Monitoring různých údajů (např. oznámení z QoS, SNMP zprávy, vyčítání informací z metalických i optických rozhraní) a následné zobrazování tabulek, statistik, grafů atd.
- Na základě určité definované události v síťovém prostředí nebo na síťovém prvku vyvolat příslušnou akci.
- Automatické zálohování konfigurací aktivních zařízení na disk s možností přepsání starších souborů, kde nebyla provedena žádná změna.
- Vizualizace sítě se zobrazením vytíženosti rozhraní jednotlivých prvků pro snadnější řešení problémů v rámci dohledového centra.
- Dle určitých příznaků vyhledávat a analyzovat bezpečnostní rizika v rámci sítě, poté doporučit úpravu či změnu konfigurace.
- Automatizace procesu PnP (Plug and Play) a tvorba konfigurací pro hromadné nasazení většího počtu zařízení do sítě.
- Sledování utilizace jednotlivých rozhraní určitých zařízení a následné doporučení nového zapojení při generační obměně.
- Pro všechna zařízení v síti získávat automaticky z databáze výrobce informace o bezpečnostních zranitelnostech, známých softwarových chybách, konci podpory atd.

Jak můžeme vidět z uvedeného seznamu, správci sítí mají různé nápady, které by jim usnadnily práci v často opakovaných úkonech. Dá se říct, že se nejedná o problémy způsobující nefunkčnost sítě. Jde spíše o banální záležitosti, které se ale v případě většího počtu zařízení stávají stereotypní, což obvykle do takových úkolů vnáší faktor lidské chyby z nekoncentrovanosti. Proto je velmi vhodné tyto věci automatizovat, čímž minimalizujeme zanesení chyby a výrazně snižujeme časovou náročnost. Uživatel poté zpracovává až konečný výsledek, nikoliv tzv. mezikroky.

### 4.2.1 Problém k řešení

Většina techniků alespoň jednou řešila problém hromadné obměny prvků, kde rozdílem v jednotlivých konfiguracích bylo pouze několik údajů, obvykle název (angl. hostname), IP adresa, používaná rozhraní apod. Nové zařízení je nutno nejprve vybalit z krabice, nasadit požadovanou verzi operačního systému, restartovat jej, převést či vytvořit konfiguraci z původního zařízení, následně zkontrolovat a přejít k dalšímu. V případě jednotek zařízení se jedná o poměrně snadný úkol, nicméně pokud jsou jich desítky či stovky, může být takový úkol velmi zdlouhavý a pro člověka příliš nezáživný. Se zvyšujícím se počtem zařízení pak vysoce klesá pozornost a zároveň roste pravděpodobnost chyby.

Z tohoto důvodu bylo mnohokrát probíraným tématem zmiňované Plug and Play, které jsme si blíže popsali v podkapitole 3.2.1. Tato funkce má obrovský potenciál využití ve většině sítí. Z vlastní zkušenosti jsem však znal určitá úskalí, která bránila jejímu jednoduchému užívání a často tak uživatele odradila od svého použití. Podrobně si tato negativa popíšeme v následující podkapitole 4.2.2.

### 4.2.2 Analýza dostupných možností

Jelikož průzkum probíhal mezi zákazníky využívající technologie firmy Cisco Systems, zaměřil jsem se na analýzu dostupných možností u této společnosti. Zároveň je nutné zdůraznit, že se pohybujeme v oblasti LAN infrastruktury podnikových sítí, tudíž se v diplomové práci nezabývám datacentrovými možnostmi.

K automatizaci procesu (uvedeného v předchozí podkapitole 4.2.1) můžeme využít funkci PnP u vhodných Cisco management produktů jako je PI (Prime Infrastructure), APIC-EM nebo DNA-C. Jak bylo uvedeno v podkapitole 3.2.2, kontrolér DNA-C je v současné době velmi aktivně řešená platforma a dochází k jejímu neustálému vývoji. Jedná se o klíčovou technologii, která by měla udávat krok v SDN přístupu a představovat nový pohled na budování moderních sítí. Důležitým poznatkem je, že DNA-C v sobě zahrnuje APIC-EM, jako součást zprostředkující komunikaci mezi grafickým uživatelským rozhraním DNA-C a síťovými prvky. Při psaní této práce přináší kontrolér DNA-C ve verzi 1.1.7 velmi podobnou PnP funkci jako je v nejnovějších verzích 1.6.2 kontroléru APIC-EM, avšak označenou jako „beta“. V dalších verzích kontroléru DNA-C se tedy očekávají jisté změny PnP funkce.

Z tohoto důvodu se primárně zaměřím na popis procesu v APIC-EM a PI. Existují tři způsoby, jak připravit instrukce k provedení automatického nasazení (angl. deployment) požadovaných zařízení:

- Musíme vytvořit konfigurace a nahrát je do kontroléru APIC-EM. Již tento první krok je časově náročný a náchylný na chyby. Poté je možné vytvořit projekt a postupně přidávat (pomocí dialogového okna) jednotlivá zařízení, kde pro každé vyplníme název, platformu, sériové číslo, image (jestliže chceme, aby se nahrál také požadovaný operační systém), správnou konfiguraci a volitelně další možnosti.

  Alternativně můžeme v tomto případě využít souboru ve formátu CSV (Comma Separated Values), ve kterém k sériovým číslům dílčích zařízení přiřadíme odpovídající konfiguraci a všechny další údaje. Příprava tohoto souboru nepatří k nejjednodušším úkonům a je potřeba dodržovat definovaný formát.
- Druhou variantou je využití šablony (psanou jazykem Velocity[1]), kde však musíme při přidávání každého zařízení do projektu ručně vyplnit nejen všechny údaje k tomuto zařízení (název, platformu, sériové číslo atd.), ale také všechny proměnné hodnoty této šablony, což je opět časově velice náročné a zároveň náchylné na chyby.

  Zde je zřejmé, že v takovém stavu je PnP vhodné jen pro malý počet zařízení, pokud vůbec, což naprosto eliminuje potenciál této funkce. Můžeme tedy konstatovat, že samotný APIC-EM neumožňuje rychlé a jednoduché přidávání většího počtu zařízení do projektu.
- Hromadné přidávání zařízení do projektu je možné pouze v PI pomocí funkce Bulk Import, kde je však nutné použít složitě formátovaný CSV soubor, který není jednoduše zpracovatelný. Pro představu je velmi zkráceně zobrazen v ukázce č. 4.1.

[1] Dokumentace na webu http://velocity.apache.org/engine/devel/vtl-reference.html.

```
#########################################################################,#
#  ##START-OF-INSTRUCTIONS##,#
#  Kindly dont tamper this comment section as this section provides
   instructions to fill this csv file.,#
#  User should provide values for the device plug and play profile
   variables for each device as separate entries.,#
#  If default values provided already and wishes to use them user can
   go as blank comma separated field.,#
#  If happened to be mandatory variable and no default value  provided at
   the time of template creation and if its field also happened to be blank
   in csv this entry will go invalid.,#
#  Kindly do not alter the field names and their order.,#
#  If any device already exists the CSV import will update the plug and
   play details for that device.,#
#  If any device doesn't already exist the CSV import will add that
   device.,#
#  CSV import cannot be used for deletion of device(s).,#
#  CSV contains two parameters with name Device Name. Provide unique and
   identical values for them. Do not leave this field blank.,#
#  Make sure each entered value is a valid value for the respective field.
   The values will not be validated during the import.,#
#  In Apic mode 'Copy Running Configuration to Startup Configuration' field
   is mandatory and has to be set to 'TRUE'.,#
#  This csv is exported for pnp profile with credential profile id. It
   doesn't need device mangaement parameters..,#
#  ##END-OF-INSTRUCTIONS##,#
#########################################################################,#
*  Device Name,"sw01","sw02","sw03"
#  Provisioning Details  #
   Device Serial Number (It is Mandatory in Apic Mode),
   "FOC2014W0VQ","FOC2014W0VA","FOC2014W0VZ"
   Location Group (Only if 'Device Serial Number' is provided),
   "NWS","NWS","NWS"
   Description,
   "","",""
   PID,
   "","",""
*  Device Name,"sw01","sw02","sw03"
#  Configuration Template : CLI : sablona-pnp  #
#  Configuration CLI : sablona-pnp  #
   *hostname [hostname]*,"sw01","sw02","sw03"
   *address [address]*,"172.16.1.101","172.16.1.102","172.16.1.103"
   *switch_type [switch_type]*,"C9300-24P","C9300-24P","C9300-24P"
   *range access portu [access_range]*,"g0/1-8","g0/1-8","g0/1-8"
   *range trunk portu [trunk_range]*,"g0/9-10","g0/9-10","g0/9-10"
#  Device Management Parameters  #
   Configure Management credential to Device,"FALSE","FALSE","FALSE"
```

Ukázka 4.1: Soubor pro Bulk Import v PI.

Vytvoření takového souboru pro desítky či stovky zařízení se stává nelehkým úkolem s velkou pravděpodobností chyby. Z praktického hlediska je soubor pro větší počet prvků naprosto nepřehledný, navíc je značně náchylný na syntaxi, kdy všechny nulové položky musí být nahrazeny prázdnými uvozovkami a je nutné použít kódování ASCII. Jestliže uživatel nesplní všechny požadavky, proces neproběhne správně. Bohužel ale není upozorněn, kde se určitá chyba nachází. Pokud je soubor připraven v pořádku, je možné jej v PI přiřadit k vytvořenému profilu s odpovídající konfigurační šablonou. Následně jsou všechny informace předány do APIC-EM, kde se vytvoří projekt s uvedenými zařízeními.

Zde vidíme další zásadní problémy tohoto přístupu. Nejenom, že je nutné mít v sítí managementní nástroj PI, ale zároveň s ním je nutné provozovat i síťový kontrolér APIC-EM. Import údajů z PI do APIC-EM navíc probíhá nezanedbatelnou dobu, jelikož se musí vygenerovat a poté přenést jednotlivé konfigurace, což ve výsledku trvá asi 25 vteřin pro jedno zařízení. Stejný čas platí i pro mazání zařízení z projektu. Tato třetí varianta hromadného nasazení pomocí PI se tedy také neukazuje jako vhodný způsob.

Všechny výše popsané nedostatky aktuálních řešení mě přivedly k myšlence vytvořit nástroj, který by fungoval jako prostředník mezi správcem sítě a síťovým kontrolérem. Jeho cílem by bylo zjednodušení či odstranění současných překážek, což by přineslo uživatelům další možnosti nejenom při nasazování nových zařízení do sítě, ale také při generační obměně a hromadné změně konfigurací. To by mohlo přinutit často konzervativní techniky k využívání funkcí síťového kontroléru, čímž by lépe pronikli do světa softwarově definovaných sítí.

### 4.2.3 Základní myšlenka a cíle

Primárním cílem je usnadnění a urychlení procesu PnP odstraněním tvorby složitého CSV souboru, což pro uživatele znamená znatelné zjednodušení hromadného nasazení nových či stávajících zařízení, bez nutnosti použití Prime Infrastructure. To je zároveň hlavní myšlenka vyvíjeného nástroje. Mezi další cíle patří:

- Automatické vytváření konfigurací pro vybraná zařízení na základě definovaných údajů a informací.
- Kompletní a přehledná správa procesu Plug and Play skrze vyvíjený nástroj.
- Jednoduchá obměna konfigurací stávajících zařízení.
- Možnost automatického vytváření síťových zařízení v bezpečnostním nástroji Cisco ISE (Identity Services Engine).
- Modulárnost nástroje pro využití kontrolérů různých výrobců.

Jedná se o vytvoření takového nástroje, který bude zpracovávat a předávat instrukce od uživatele do kontroléru i naopak. Vznikne tak jednoduchá správa celého procesu PnP, kdy není potřeba otvírat grafické uživatelské rozhraní vybraného kontroléru.

## 4.3 Tvorba nástroje

Aby název patřičně vystihoval hlavní funkci vyvíjeného nástroje, zvolil jsem jednoduché pojmenování **EasyPnP**, což logicky napovídá, že hlavním cílem je zjednodušení procesu Plug and Play.

V následujících podkapitolách rozeberu volbu programovacího jazyka, popíši hlavní předpoklady správného použití nástroje a představím jeho blokovou architekturu.

### 4.3.1 Programovací jazyk

Při tvorbě jakéhokoliv softwaru je prvním zásadním krokem volba vhodného programovacího jazyka. Možností je jako obvykle celá řada, například Java, PHP, C++, Python, Ruby, Perl atd. Vždy je důležité vybrat jazyk, který je pro naši práci nejvhodnější, což může být zpočátku náročné. Proto bychom měli nejprve definovat oblast, v níž bude software používán. V mém případě se jedná o síťové prostředí a komunikaci s kontrolérem i zařízeními pomocí API volání, popřípadě SSH (Secure Shell) protokolem. Zde se nabízí především programovací jazyk Python, který se pro datové sítě hodí nejvíce a zároveň vyhovuje určeným požadavkům:

- Vysokoúrovňový (angl. high-level) jazyk, tedy s vyšší mírou abstrakce. Zápis kódu je pro člověka srozumitelnější a přehlednější, program je obvykle kratší a lépe čitelný, čímž se zmenšuje pravděpodobnost výskytu chyb.
- Multiplatformní, nebo-li přenositelný mezi různými počítačovými platformami (Windows, MacOS, Linux) bez nutnosti provádět změny v kódu.
- Otevřený (angl. open-source) software, který musí být dostupný zdarma a nabízet širokou škálu knihoven i balíčků s podporou uživatelské komunity.
- Jednoduchý systematický jazyk, vhodný pro uživatele, který se programováním či skriptováním běžně nezabývá.

Pro Python hovoří mnoho dalších aspektů. Jedná se o interpretovaný jazyk, tudíž nevyžaduje kompilaci. To je výhoda při experimentování a vývoji, zároveň však nevýhoda z pohledu rychlosti překladu rozsáhlejšího kódu. Dále je víceúčelový, tudíž využitelný v řadě oblastí díky všestranným funkcím a menším programovým kódům. Nezanedbatelným faktorem je také jeho podpora samotnými výrobci síťových zařízení, kde v některých novějších operačních systémech (např. Cisco IOS XE) můžeme nalézt integrovaný Python interpretr k přímému spouštění uživatelských skriptů. Pro svoji jednoduchost a efektivitu se stal široce používaným jazykem programátorů i správců sítí. Je snadný k naučení, disponuje velkým množstvím výukových materiálů nejen z oblasti síťové programovatelnosti a vlastní obrovskou komunitu aktivních uživatelů.

Všechny tyto důvody mě vedly k volbě jazyka Python, u kterého se aktuálně používají dvě verze, 2.x a 3.x, které nejsou vzájemně kompatibilní. Pro tvorbu svého nástroje jsem využil Python verze 3.6, který odstraňuje řadu nedostatků a chybných návrhů předchozích verzí jazyka.

### 4.3.2 Hlavní předpoklady

Jedinými dvěma předpoklady pro funkční využití nástroje EasyPnP jsou vyplněná XLS tabulka se základními údaji o konfigurovaných zařízeních (viz tab. č. 4.1) a konfigurační šablona s definovanými proměnnými (viz ukázka č. 4.2). Na základě těchto dvou vstupů jsou nástrojem vygenerovány konfigurace pro jednotlivá zařízení dle řádků tabulky (viz ukázka č. 4.3). Pro následné nahrávání konfigurací i vytváření pravidel musíme zajistit dosažitelné připojení ke zvolenému kontroléru. To však není potřeba k samotnému generování konfigurací.

- **Tabulka**
  Jako výchozí soubor pro jednoduché formulování údajů k jednotlivým zařízením jsem zvolil tabulku ve formátu XLS. Tato klasická „excelovská“ tabulka (viz tab. č. 4.1) je známá velké většině uživatelů a zároveň je, oproti CSV formátu, značně uživatelsky přívětivá. Přesto je však stále jednoduše programově zpracovatelná. Dalším důvodem pro toto rozhodnutí je možnost vygenerování XLS souboru se sériovými čísly a s označením platformy na základě vydané objednávky, což správcům sítí přináší snížení časové náročnosti při přípravě takového dokumentu.

  Tabulka obsahuje názvy, které označují její jednotlivé sloupce. Tyto parametry jsou citlivé na velikost písmen (angl. case sensitive), nesmí tedy být nijak modifikovány ani mazány. Dělíme je do tří skupin:

  - **Povinné**
    - **hostName** – název zařízení
    - **serialNumber** – sériové číslo zařízení
    - **platformId** – označení platformy
    - **site** – umístění zařízení
    - **ipAddress** – IP adresa zařízení
    - **ipMask** – maska sítě
  - **Povinně volitelné**
    - **image** – požadovaný operační systém
    - **devCert** – použití certifikátu
    - **userName** – uživatelský přístup
    - **passWord** – uživatelské heslo
    - **radius** – povolení RADIUS protokolu
    - **radiusSecret** – heslo pro RADIUS
    - **tacacs** – povolení TACACS protokolu
    - **tacacsSecret** – heslo pro TACACS
  - **Volitelné**
    - **xxxxxxx** – jakýkoliv další parametr

Tab. 4.1: Vzor vyplněné tabulky.

| **hostName** | **serialNumber** | **platformId** | **site** | **ipAddress** | **ipMask** | **...** |
|---|---|---|---|---|---|---|
| sw01 | FOC2014W0VQ | C9300-24P | NWS | 172.16.1.101 | 255.255.0.0 | ... |
| sw02 | FOC2014W0VA | C9300-24P | NWS | 172.16.1.102 | 255.255.0.0 | ... |
| sw03 | FOC2014W0VZ | C9300-24P | NWS | 172.16.1.103 | 255.255.0.0 | ... |

Povinné parametry musí být vždy vyplněny pro každé zařízení, jelikož jsou součástí kódu programu, a jsou zásadní pro správné generování konfigurací i následné vytváření pravidel v kontroléru. Povinně volitelné parametry musí v tabulce zůstat, ale nemusí být vyplněny. Nechceme-li například zadávat své heslo, necháme zmiňovanou část prázdnou. Tyto parametry se aplikují v některých funkcích nástroje. Pokud nejsou vyplněny a vy zvolíte funkci, ve které se používají, program vás automaticky upozorní, abyste tabulku korektně upravili. Poslední skupinou jsou volitelné parametry, které se využívají pouze v šabloně. V tabulce můžeme vytvořit sloupec s libovolným názvem, vyplnit pro něj údaje u jednotlivých zařízení per řádek, a poté tento parametr vydefinovat také ve zvolené části šablony. Při následném generování konfigurací dojde k jejich spárování.

Jestliže je jakýkoliv z výše uvedených parametrů použitý v šabloně, měl by být správně vyplněn. V opačném případě zůstane prázdné místo ve výsledné konfiguraci, což může způsobit nepředvídatelné chování zařízení.

- **Šablona**

  Jak jsem již krátce zmínil v podkapitole 4.2.2, pro psaní šablon se velmi často používá jazyk Velocity, kde se jednotlivé proměnné definují pomocí znaku dolaru ($). Toto řešení mi nepřijde příliš vhodné, protože někdy je součástí konfigurace také hash, obsahující právě tento znak. V takovém případě musíme speciálně definovat, že daný znak dolaru neuvozuje proměnnou.

  Proto jsem se rozhodl zvolit jiný jazyk pro psaní konfiguračních šablon. Vybíral jsem mezi TypeScript, Handlebars, Pug, Jinja2 a Mako. První tři uvedené se hodí především pro jazyk JavaScript, zbylé dva jsou vhodné šablonové jazyky pro Python, jež se vzájemně liší pouze nepatrnými syntaktickými rozdíly. Z osobní preference jsem zvolil jazyk Jinja2[2], jenž má souborovou příponu jnj.

  Na rozdíl od jazyku Velocity se proměnné v Jinja2 definují pomocí dvojitých složených závorek. Název proměnné v šabloně (ukázka č. 4.2) musí přesně odpovídat názvu parametru v tab. č. 4.1. Zde jsou pro jednoduchost uvedeny pouze povinné parametry, struktura této tabulky se však od reálně používané nijak neliší. Je zřejmé, že tímto způsobem můžeme vytvořit jakýkoliv počet volitelných parametrů, tudíž dokážeme jednoduše zpracovávat také konfigurační šablony, kde se u jednotlivých zařízení liší i větší počet údajů. Nástroj EasyPnP následně tyto dva vstupy zpracovává a velmi rychle generuje konfigurace pro každý vyplněný řádek tabulky. V této

[2]Dokumentace na webu http://jinja.pocoo.org/docs/2.10/.

ukázce jsou tedy výsledkem tři konfigurace (první z nich je v ukázce č. 4.3) pro tři různá zařízení. Proces je velmi intuitivní a jednoduchý. V podkapitole 4.4.3 si podrobně vysvětlíme kód, který zajišťuje uvedené generování konfigurací.

```
hostname {{hostName}}
!
enable secret Cisco123!
!
aaa new-model
username Admin secret 4 Cisco123!
!
int vlan 1
description SN {{serialNumber}}
ip address {{ipAddress}} {{ipMask}}
!
end
```

Ukázka 4.2: Konfigurační šablona.

```
hostname sw01
!
enable secret Cisco123!
!
aaa new-model
username Admin secret 4 Cisco123!
!
int vlan 1
description SN FOC2014W0VQ
ip address 172.16.1.101 255.255.0.0
!
end
```

Ukázka 4.3: Vygenerovaná konfigurace.

Velmi důležitou součástí reálných konfiguračních šablon je EEM (Embedded Event Manager) skript, který podporuje většina operačních systémů Cisco IOS, a který umožňuje odstraňovat problémy v síti jednoduchou detekcí událostí. Článek [56] výstižně popisuje využitelnost EEM skriptu společně s ukázkovými příklady.

V případě EasyPnP používám EEM skript k samostatnému zadávání příkazů do zařízení po provedení automatického nasazení (angl. deployment). Předložený EEM skript v ukázce č. 4.4 funguje tak, že po úspěšném nasazení zařízení se spustí časovač (zde 120 sekund), a poté se zadají definované příkazy. Povšimněme si především autorizačních příkazů. Využíváme-li PnP funkci, musíme je přidávat do zařízení vždy pomocí EEM skriptu. V produkčních sítích totiž často dochází k situaci, kdy TACACS či RADIUS server přeruší spojení kvůli autorizačním příkazům v nahrávané konfiguraci. Nasazení sice obvykle proběhne správně, ale uživatel o tom není nijak informován. Tento problém řešíme tím, že do zařízení nahrajeme konfiguraci, jejíž součástí je zmiňovaný EEM skript obsahující všechny autorizační příkazy.

Dále zde vidíme příkazy, které po sobě odstraní informace o využití PnP funkce. Smaže se profil, vytvořená VLAN, popis u připojeného rozhraní, samotný EEM skript společně s EEM uživatelem, a poté dojde k uložení aktuální konfigurace. Součástí EEM skriptu může být v podstatě jakýkoliv příkaz.

```
event manager session cli username USER privilege 15
event manager applet POST_EASYPNP
event timer countdown time 120
action 1.0 cli command "enable"
action 1.1 cli command "config t"
action 2.0 cli command "aaa author config-commands"
action 2.1 cli command "aaa author exec default group ISE if-auth"
action 2.2 cli command "aaa author commands 15 default group ISE none"
```

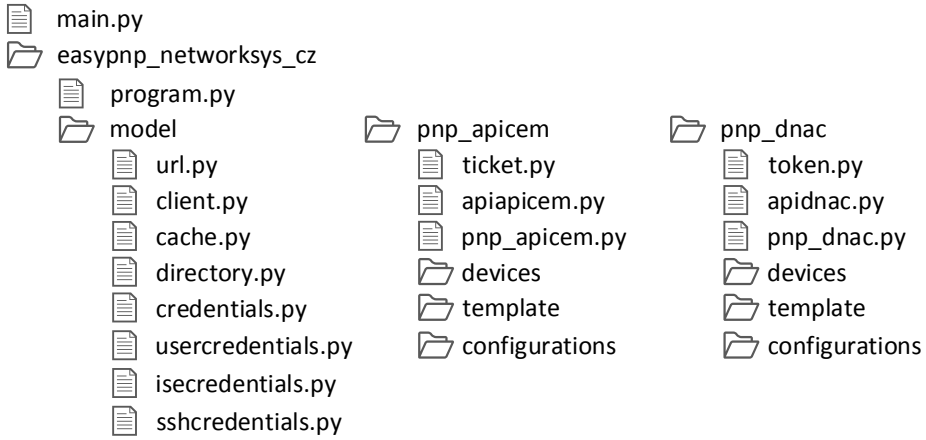
```
action 2.3 cli command "aaa author network default group ISE-DOT1X"
action 3.0 cli command "no service config"
action 3.1 cli command "no pnp profile pnp-zero-touch"
action 3.2 cli command "no interface vlan {{startupVlan}}"
action 3.3 cli command "interface {{connectPort}}"
action 3.4 cli command "no macro description CISCO_SMI_EVENT"
action 3.5 cli command "exit"
action 3.6 cli command "no event manager applet POST_EASYPNP"
action 3.7 cli command "no event manager session cli username USER"
action 3.8 cli command "end"
action 3.9 cli command "wr mem"
action 4.0 cli command "end"
```

Ukázka 4.4: Příklad EEM skriptu.

Hlavní předpoklady pro správné fungování vyvíjeného nástroje již známe. Nyní přejdu k popisu navržené architektury.

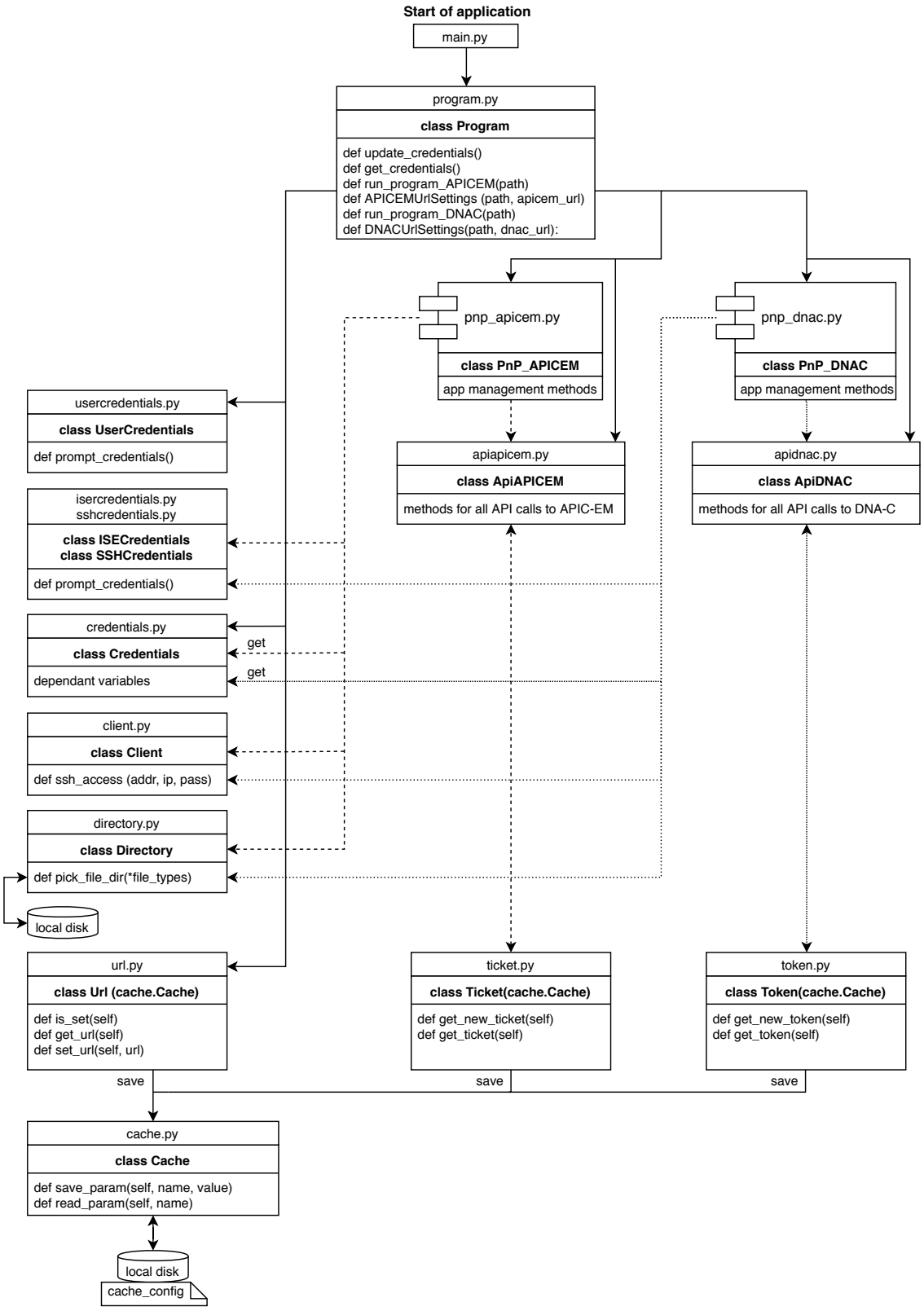
### 4.3.3 Bloková architektura

Program jsem se snažil rozdělit na logické části dle pravidel UML (Unified Modeling Language), aby byl jednoduše rozšiřitelný a modulární pro další kontroléry různých výrobců. Souborová struktura programu je zobrazena na obr. č. 4.1, kde můžeme vidět zanoření jednotlivých, aktuálně používaných, souborů a složek.




Obr. 4.1: Souborová struktura nástroje EasyPnP.

Spuštění programu probíhá pomocí objektu `main.py`, což je hlavní inicializační soubor. Na něj je navázán objekt `program.py`, který můžeme označit za řídicí jednotku celého nástroje. Součástí programu je modelová část (složka `model`), obsahující všechny třídy a metody, které jsou sdíleny modulovou částí (složka `pnp_apicem` a `pnp_dnac`), kde se již nachází třídy a metody pro konkrétní kontrolér. Tímto je zajištěna modulárnost nástroje. Bloková architektura popisující vazby mezi těmito soubory je zachycena na obr. č. 4.2. Zároveň jsou na obrázku uvedeny jednotlivé třídy a některé vybrané metody.




Obr. 4.2: Bloková architektura nástroje EasyPnP.

## 4.4 Objekty nástroje

V rámci této sekce představím logické bloky, fundamentální metody i důležité funkce nástroje EasyPnP. V jednotlivých ukázkách budou zobrazeny různé úseky kódu, kde je z důvodu rozsáhlosti a zachování přehlednosti nutné některé části zestručnit, popřípadě úplně vynechat. Kompletní zdrojové kódy jsou však volně dostupné ve webovém repozitáři GitHub[3], společně s popisem a dokumentací v anglickém jazyce.

Jak jsem již uvedl v podkapitole 4.3.3, program je koncipován pro modulární využití. Z tohoto důvodu je jeho struktura rozdělena do tří logických bloků a to na modelovou, programovou a modulovou část. Mimo tyto bloky patří pouze soubor `main.py`, skrze který dochází ke spouštění programu.

- **main.py**

Tento soubor neobsahuje žádnou třídu ani metodu. Po spuštění v CLI (při použití Python verze 3.6 příkazem `py main.py` v příslušné složce programu) dojde k vyvolání hlavního menu, kde máme aktuálně na výběr mezi kontrolérem Cisco APIC-EM a Cisco DNA-C.

```
1  from easypnp_networksys_cz import program
2  MENU_NAME = "MENU"
3  #Start aplikace EasyPnP
4  while True:
5      try:
6          print("\n" + MENU_NAME + ":")
7          command = input("1 - APIC-EM\n 2 - DNA-C\n 3 - END\n")
8          if command == "1":
9              #Vstup do kontroléru Cisco APIC-EM
10             program.Program.run_program_APICEM(MENU_NAME)
11         elif command == "2":
12             #Vstup do kontroléru Cisco DNA-C
13             program.Program.run_program_DNAC(MENU_NAME)
14         elif command == "3" or command == "q" or command == "Q":
15             #Ukončení aplikace EasyPnP
16             break
17     except RuntimeError:
18         continue
```

Ukázka 4.5: Celý obsah souboru `main.py`.

V ukázce č. 4.5 vidíme, že se zde nachází pouze import souboru `program.py`, abychom mohli zavolat jeho metody na řádku 10 i 13, a proměnná pro název menu. Poté následuje smyčka, kde volíme mezi dvěma kontroléry a ukončením programu. Řádek 6 zobrazuje uživateli aktuální zanoření v jednotlivých částech menu. Zadáním příslušného čísla dle instrukcí na řádku 7 se dostaneme do námi zvolené části programu. Tento výběr je ošetřen proti výjimce „RuntimeError“ pomocí zachytávajícího bloku try/expect.

[3]Dostupné na webu https://github.com/mikesm11/EasyPnP.

### 4.4.1 Modelová část

První a zároveň hlavní logickou část nástroje EasyPnP zastřešuje 8 souborů (viz obr. č. 4.1). Metody a funkce modelové části jsou sdíleny programovou a modulovou částí. Lze tak jednoduše přidávat další moduly (kontroléry) bez úprav v modelové části. Kód je tedy přehlednější a jednoduše rozšiřitelný. Dále v textu představím většinu základních metod a funkcí jednotlivých souborů.

- **usercredentials.py, sshcredentials.py, isecredentials.py**

Metody obsažené v těchto souborech slouží k vyvolání dialogového okna pro zadávání přístupových údajů (angl. credentials) ke kontroléru (`usercredentials.py`), k jednotlivým zařízením (`sshcredentials.py`) nebo k Cisco ISE (`isecredentials.py`). Jejich obsah je téměř totožný, liší se pouze v počtu proměnných.

```
1  import tkinter as tk
2  class UserCredentials:
3      __root, __user, __pwd = None
4      __user_val, __pwd_val = None
5      #Metoda pro vyvolání dialogového okna
6      def prompt_credentials():
7          #Parametry dialogového okna
8          UserCredentials.__root = tk.Tk()
9          UserCredentials.__root.geometry('300x160')
10         UserCredentials.__root.title('Enter your credentials')
11         #Ohraničení dialogového okna
12         parent = tk.Frame(UserCredentials.__root, padx=10, pady=10)
13         parent.pack(fill=tk.BOTH, expand=True)
14         #Kolonky pro uživatelské vstupy
15         UserCredentials.__user = UserCredentials.__make_entry
16                             (parent, "Username:", 16)
17         UserCredentials.__pwd = UserCredentials.__make_entry
18                             (parent, "Password:", 16, show="*")
19         #Tlačítko pro potvrzení
20         b = tk.Button(parent, borderwidth=4, text="Login", width=10,
21                             pady=8, command=UserCredentials.__submit)
22         #Přizpůsobení velikosti okna dle textu
23         b.pack(side=tk.BOTTOM)
24         UserCredentials.__user.focus()
25         UserCredentials.__pwd.bind('<Return>',UserCredentials.__ent_submit)
26         #Zaostření a zobrazení výše definovaného dialogového okna
27         parent.focus_force()
28         parent.mainloop()
29         #Navrácení zadaných uživatelských vstupů
30         if not UserCredentials.__user_val or not UserCredentials.__pwd_val:
31             return None
32         return UserCredentials.__user_val, UserCredentials.__pwd_val
```

Ukázka 4.6: Část souboru `usercredentials.py`.

K vyvolání dialogového okna je použita standardní knihovna Tkinter[4], pro tvorbu grafického uživatelského rozhraní v Pythonu. Tato knihovna má velké množství předností, snadno se používá, umožňuje rozšiřování o další ovládací prvky a je volně dostupná.

V Pythonu neexistují privátní datové členy, ke kterým by byl přístup pouze v rámci jednoho objektu. Je zde však zaužívaná konvence, kdy použitím jednoduchého nebo dvojitého podtržítka označujeme lokální část kódu, ať už je to funkce, metoda nebo proměnná. Jedno podtržení naznačuje, že datový člen je „privátní“ a měli bychom k němu přistupovat pouze uvnitř dané třídy. Použitím dvojího podtržení dojde k tzv. „zmrazení“ názvů atributů třídy, čímž se zabrání konfliktům v kódu. V tomto případě proměnná `__root` představuje proměnnou `_UserCredentials__root`.

Metoda `prompt_credentials` je tedy veřejná a slouží k vyvolání dialogového okna. Jednotlivé části kódu jsou popsány v ukázce č. 4.6. Zde se také volají zbylé 3 privátní metody třídy `UserCredentials`, kde `__ent_submit` i `__submit` obstarávají zachycení uživatelských hodnot do proměnných, a `__make_entry` vytváří kolonky pro uživatelský vstup.

- **credentials.py**

Soubor slouží pouze pro ukládání uživatelských údajů ke kontroléru, k Cisco ISE a pro SSH přístup k jednotlivým zařízením. Do těchto proměnných se informace vkládají pouze při běhu programu. Pokud jej ukončíme a následně znovu spustíme, údaje nejsou nikde zaznamenány. Na lokální disk se však ukládá URL (Uniform Resource Locator) adresa kontroléru a autentizační řetězec (více při popisu souboru `cache.py` v této podkapitole).

```
1 class Credentials():
2     controller_username = ""
3     controller_password = ""
4     ise_address = ""
5     ise_username = ""
6     ise_password = ""
7     ssh_address = ""
8     ssh_username = ""
9     ssh_password = ""
```

Ukázka 4.7: Celý obsah souboru `credentials.py`.

Jak můžeme vidět v ukázce č. 4.7, obsahem je třída `Credentials` a v ní 8 globálních proměnných, jež jsou přístupné pro ostatní části programu.

- **client.py**

Uvedený soubor se používá pro automatické mazání a restartování síťových prvků. Díky této funkci můžeme vzdáleně vyčistit zvolené zařízení a následně na něj znovu, pomocí programu EasyPnP a vybraného kontroléru, nahrát novou systémovou konfiguraci. Můžeme

[4]Dokumentace na webu https://docs.python.org/3/library/tk.html.

tak velmi jednoduše provádět hromadnou obměnu konfigurací většího počtu zařízení bez nutnosti ručního zásahu.

```
1  import time
2  import paramiko
3  class Client:
4    def ssh_access(ssh_address, ssh_username, ssh_password, printResp=True):
5       try:
6           #Vytvoření nové instance
7           client = paramiko.SSHClient()
8           #Automatické přidání certifikátu
9           client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
10          #Navázání SSH relace
11          client.connect(hostname=ssh_address, username=ssh_username,
12                         password=ssh_password)
13          #Vytvoření interpretu
14          remote_conn = client.invoke_shell()
15          #Definice zadávaných příkazů
16          remote_conn.send('\n')
17          time.sleep(1)
18          remote_conn.send("erase /all nvram:\n")
19          time.sleep(3)
20          remote_conn.send("\n")
21          remote_conn.send("\n")
22          time.sleep(1)
23          remote_conn.send("reload\n")
24          time.sleep(3)
25          remote_conn.send("no\n")
26          remote_conn.send("\n")
27          time.sleep(3)
28          remote_conn.send("reload\n")
29          time.sleep(3)
30          remote_conn.send("\n")
31          remote_conn.send("\n")
32          return True
33      except Exception as e:
34          return False
```

Ukázka 4.8: Celý obsah souboru `client.py`.

K navázání SSH spojení je použita knihovna paramiko[5]. Pro Python můžeme nalézt velké množství knihoven realizující SSH relace, ale právě tato je nejznámější, často aktualizovaná, s kvalitní dokumentací a samozřejmě volně přístupná. Implementuje šifrovaný protokol SSHv2 a poskytuje funkce klienta i serveru. Zmíněná knihovna disponuje celou řadou vlastností. Pro potřeby nástroje EasyPnP však dostačují funkce okomentované a vysvětlené v ukázce č. 4.8.

[5]Dokumentace na webu http://www.paramiko.org/.

Po úspěšném navázání spojení a vytvoření interaktivního interpretu se do zvoleného prvku pošlou příkazy uvedené na řádcích 16-31. Zde jsem řešil především problém s očekávaným dialogem při restartování Cisco zařízení. Pokud se konfigurace uložená v NVRAM liší od aktuálně běžící, jste po zadání příkazu `erase /all nvram:` dotázání, zda chcete konfiguraci uložit. V takovém případě je nutné zadat příkaz `no` a potvrdit jej. Tuto situaci řeší řádky 23-26. Pokud se však obě konfigurace neliší (nedošlo k žádné změně od posledního uložení konfigurace do NVRAM příkazem `write memory`), uživatel není nijak dotazován a je pouze nutné příkaz `reload` potvrdit. Tato druhá situace se nachází na řádku 28-31. Vysvětlení je tedy následovné. Jestliže jsem dotázán o uložení konfigurace, řádkem 25 odmítnu a řádkem 26 odmítnutí potvrdím. Zařízení se začne restartovat a další příkazy neproběhnou. Pokud však dotázán nejsem, restart se nejprve řádkem 25 přeruší, avšak řádkem 28 je opět vyvolán, kde již následuje pouze potvrzení tohoto příkazu na řádku 30. Tímto jsem jednoduše docílil požadovaného výsledku a v obou případech dojde k úspěšnému provedení restartu. Mezi jednotlivými příkazy je použita knihovna time[6], která pozastaví provedení volajícího vlákna na uvedený počet sekund. Vyhneme se tak zahlcení kanálu a zmírníme následky zpoždění způsobeného nekvalitním připojením.

Tato metoda se používá dvojím stylem, jež se liší ve způsobu získání parametrů k vytvoření požadované SSH relace. V prvním případě postupného mazání zařízení se jako parametry k navázání spojení používají informace v proměnných `ssh_address`, `ssh_username` a `ssh_password` třídy `Credentials`, do nichž jsou uživatelské údaje uloženy pomocí třídy `SSHCredentials`. V druhém případě hromadného mazání zařízení se jako parametry k navázání spojení používají informace v XLS tabulce, tedy konkrétně `ipAddress`, `userName` a `passWord`. Blíže budou obě funkce vysvětleny v sekci 5.2.

- **directory.py**

Obsahem jsou 2 metody, které umožňují zvolit tabulku a šablonu. Pro správný běh programu je výběr souborů omezen pomocí parametru pouze na přesný typ přípony. Tabulka s definovanými zařízeními musí být ve formátu XLS (přípona xlsx), konfigurační šablona musí být ve formátu Jinja2 (přípona jnj).

```
1  from tkinter import filedialog
2  import tkinter as tk
3  class Directory:
4    #Metoda pro výběr souboru
5    def pick_file_dir(*file_types):
6       #Vytvoření nové instance
7       root = tk.Tk()
8       root.withdraw()
9       #Není-li přípona určena parametrem, pracovat s údajem "all"
10      if file_types == ():
11          file_types = ('all',)
12      #Parametrem umožnit výběr pouze podporovaného typu souboru
13      supported = Directory.__get_supported_types(file_types)
```

[6]Dokumentace na webu https://docs.python.org/3/library/time.html.

```
14      #Vyvolání dialogového okna
15      file_path = filedialog.askopenfile(filetypes=supported)
16      #Vrátit cestu k souboru
17      if file_path is None:
18          return None
19      return file_path.name
20
21  #Privátní metoda pro zobrazení parametrem definovaných typů souboru
22  def __get_supported_types(*types):
23      #Prázdná n-tice (angl. tuple)
24      out_types = ()
25      for t in types[0]:
26      #Pokud je parametrem "all", umožnit výběr všech typů souboru
27          if t == 'all':
28              out_types = (('all files', '*.*'),) + out_types
29              continue
30          #Vyplnění n-tice definovaným typem souboru k výběru
31          desc = t + ' files'
32          match = '*.' + t
33          out_types = ((desc, match),) + out_types
34      #Pokud je n-tice prázdná, umožnit výběr všech typů souboru
35      if out_types == ():
36          out_types = (('all files', '*.*'),)
37      #Vrátit n-tici se soubory definovaného typu
38      return out_types
```

Ukázka 4.9: Celý obsah souboru `directory.py`.

Jednotlivé části kódu jsou detailně popsány v ukázce č. 4.9. K vyvolání grafického dialogového okna používám opět knihovnu Tkinter (podobně jako v ukázce č. 4.6), ale tentokrát především její modul `filedialog`, který slouží k interaktivnímu výběru souboru z vnitřního úložiště.

Výše uvedená veřejná metoda `pick_file_dir` je volána jednotlivými moduly. Po úspěšném výběru souboru dojde k jeho zkopírování na lokální disk do složky vybraného kontroléru (viz souborová struktura na obr. č. 4.1) programu EasyPnP. Tabulka je kopírována do složky `devices`, šablona do složky `template`. Důvodem je uchování zvolených souborů i po ukončení programu a také stálost cesty k těmto souborům.

- **cache.py**

Jak název napovídá, metody tohoto souboru zajišťují ukládání vybraných dat na lokální disk, konkrétně do textového souboru `cache_config`. Obsahem je URL adresa kontroléru a příslušný autentizační řetězec. V souborové struktuře se `cache_config` nachází na úrovni hlavního inicializačního souboru `main.py`, před prvním spuštěním programu EasyPnP však neexistuje. Soubor je vytvořen až po zadání IP adresy nebo DNS záznamu zvoleného kontroléru. Následným vyplněním přístupových údajů se po úspěšné autentizaci uloží také autentizační řetězec, jenž se používá pro všechna další API volání. Tyto údaje se

automaticky obnovují aktuálně platnými hodnotami. Soubor má datovou strukturu typu slovník (angl. dictionary) a záznam pro jeden kontrolér vypadá následovně:

▷ {"apicem_url": {"url": "10.22.30.11"}, "ticket": {"ticket": "ST-100-Nob12niU9c-cas"}}

Pro jeden typ kontroléru není možné uložit více hodnot. Pokud však zvolím jiný typ kontroléru, vytvoří se nový záznam jako další položka uvedeného slovníku.

Převážně zde jsem se snažil dodržovat principy objektově orientovaného programování OOP (Object Oriented Programming). Třída `Cache` je rodičovskou třídou, ze které dědí třídy `Url`, `Ticket` a `Token`. Ty budou popsány dále v textu. Nejprve si v ukázce č. 4.10 vysvětlíme veřejnou část třídy `Cache`.

```
1  import json
2  class Cache:
3      #Vytvoření konstruktoru
4      def __init__(self, name):
5          #Název textového souboru pro ukládání informací
6          self.__file_name = 'cache_config'
7          #Parametr, se kterým potomek inicializuje třídu Cache
8          self.__child_name = name
9
10     #Veřejná metoda pro ukládání záznamů do cache
11     def save_param(self, param_name, param_value):
12         #Do objektu c uložit aktuální záznamy z cache_config
13         c = self.__read_cache()
14         #Pokud inicializační parametr v cache není, vytvořit pro něj záznam
15         if not c.get(self.__child_name):
16             c[self.__child_name] = {}
17         #Do zvoleného záznamu přiřadit příslušnou hodnotu
18         c[self.__child_name][param_name] = param_value
19         #Následně uložit nový obsah objektu c do souboru cache_config
20         self.__save_cache(c)
21
22     #Veřejná metoda pro čtení záznamů z cache
23     def read_param(self, param_name):
24         #Do objektu c uložit aktuální záznamy z cache_config
25         c = self.__read_cache()
26         #Vrátit parametrem definovanou hodnotu z cache, jinak vrátit None
27         if c.get(self.__child_name) and c[self.__child_name].get(param_name):
28             return c[self.__child_name][param_name]
29         return None
```

Ukázka 4.10: První část souboru `cache.py`.

Základem OOP je inicializační metoda, která je volána třídou potomka s příslušným parametrem. Tento parametr je klíčem ve slovníku textového souboru `cache_config`. K tomuto klíči je poté přiřazena určitá hodnota pomocí metody `save_param`. Pro získávání údajů ze souboru slouží metoda `read_param`. Tyto metody využívají další privátní funkce třídy `Cache`, které jsou vysvětleny v ukázce č. 4.11.

```
    #Privátní metoda pro získání údajů ze souboru cache_config
    def __read_cache(self):
      content = None
      #Pokud soubor existuje, uložit jeho obsah do proměnné content
      try:
          file = open(self.__file_name, 'r')
          content = file.read()
          file.close()
      #Pokud soubor neexistuje, vrátit prázdný slovník
      except FileNotFoundError:
          return {}
      #Načíst proměnnou content v JSON formátu a vrátit výsledek
      try:
          content = json.loads(content)
          return content
      #Pokud nastane chyba při konverzi, vrátit prázdný slovník
      except json.JSONDecodeError:
          return {}

    #Privátní metoda pro uložení údajů do souboru cache_config
    def __save_cache(self, content):
      #Převést proměnnou content do JSON formátu
      try:
          content = json.dumps(content)
      #Pokud nastane chyba, vrátit False
      except:
          return False
      #Otevřít soubor a uložit do něj obsah proměnné content
      try:
          file = open(self.__file_name, 'w')
          file.write(content)
          file.close()
      #Pokud nastane chyba při zápisu do souboru, vrátit False, jinak True
      except:
          return False
      return True
```

Ukázka 4.11: Druhá část souboru `cache.py`.

Obě privátní funkce se starají o přímou interakci s textovým souborem `cache_config`. První metoda `__read_cache` obstarává načtení obsahu souboru do proměnné, která je vrácena volající metodě. Druhá metoda `__save_cache` zajišťuje uložení nových údajů do zmiňovaného souboru.

Třída potomka (`Url`, `Ticket`, `Token`) volá veřejnou metodu rodičovské třídy `Cache`, která skrze uvedené privátní metody provádí požadované úkony. Soubor `url.py` bude popsán následovně, soubory `ticket.py` a `token.py` v podkapitole 4.4.3.

- **url.py**

Uvedený soubor zahrnuje 4 metody pro práci s URL adresou zvoleného kontroléru. Aby se v rámci běhu programu nemusela pro každou komunikaci s kontrolérem číst jeho adresa ze souboru `cache_config`, ukládám ji do lokální proměnné `__url`, a také do proměnné `__urlController`, jež je součástí modulové třídy (v tomto případě `ApiAPICEM` nebo `ApiDNAC`) obsahující všechna API volání k vybranému kontroléru. Pokud však dojde k ukončení programu EasyPnP, URL adresa je zachována pouze v textovém souboru `cache_config`. Při dalším spuštění programu se použije adresa z tohoto souboru a aktualizuje se v případě neplatnosti nebo vynucení změny.

```
1  from easypnp_networksys_cz.model import cache
2  class Url(cache.Cache):
3    #Lokální proměnné
4    __URL_ADDRESS = 'url'
5    __url = ''
6    #Vytvoření konstruktoru
7    def __init__(self, urlName):
8      #Mnohoděličnost, vyvolá inicializaci třídy Cache s def. parametrem
9      super(Url, self).__init__(urlName)
10     #Při inicializaci uložit do lokální proměnné __url hodnotu z cache
11     self.__url = self.read_param(self.__URL_ADDRESS)
12
13   #Kontrola, zda lokální proměnná __url obsahuje data
14   def is_set(self):
15     if self.__url is None:
16         return False
17     return True
18
19   #Pokud lokální proměnná __url obsahuje data, vrátit její hodnotu
20   def get_url(self):
21     if self.is_set():
22         return self.__url
23     return ''
24
25   #Uložit zadanou URL adresu do lokální proměnné __url i do cache
26   def set_url(self, url):
27     self.__url = url
28     self.save_param(self.__URL_ADDRESS, url)
```

Ukázka 4.12: Celý obsah souboru `url.py`.

Na začátku ukázky č. 4.12 importuji rodičovskou třídu `Cache`, ze které dědím vlastnosti i funkce. Poté vytvářím lokální proměnné s nimiž pracuji v rámci třídy `Url`. Následují metody zajišťující zpracování zadané URL adresy. Tyto metody jsou volány pouze z programové části, tedy třídou `Program`.

### 4.4.2 Programová část

Druhou logickou část nástroje EasyPnP představuje jediný soubor `program.py`, který níže popíši a vysvětlím jeho základní metody a funkce. Jedná se v podstatě o řídicí jednotku, jež obstarává správné zadání adresy kontroléru, přístupových údajů a spuštění vybraného modulu.

- **program.py**

V tomto souboru se nachází stejnojmenná třída `Program`, která obsahuje 2 systémové statické metody, dále 6 statických metod pro kontrolér APIC-EM a 6 statických metod pro kontrolér DNA-C.

```
1  def update_credentials():
2    while True:
3      try:
4          command = input("Invalid or no credentials. Continue? (y/n):")
5          if command == "Y" or command == "y":
6              result = usercredentials.UserCredentials.prompt_credentials()
7              if result is None:
8                  continue
9              credentials.Credentials.controller_username = result[0]
10             credentials.Credentials.controller_password = result[1]
11             return True
12         elif command == "N" or command == "n":
13             return False
14     except RuntimeError:
15         continue
```

Ukázka 4.13: Metoda `update_credentials` souboru `program.py`.

První ze systémových metod můžeme vidět v ukázce č. 4.13. Tato metoda zajišťuje obnovení přístupových údajů při vypršení autentizačního řetězce nebo při první autentizaci uživatele vůči zvolenému kontroléru. Nejprve je řádkem 6 zavolána metoda z třídy `UserCredentials`, výsledek z dialogu je poté na řádcích 9 a 10 uložen do lokálních proměnných třídy `Credentials`.

```
1  def get_credentials(c_user=False, c_pass=False):
2    try:
3      if c_user == True:
4          controller_username = credentials.Credentials.controller_username
5          return controller_username
6      if c_pass == True:
7          controller_password = credentials.Credentials.controller_password
8          return controller_password
9    except RuntimeError:
10     pass
```

Ukázka 4.14: Metoda `get_credentials` souboru `program.py`.

Druhou systémovou metodou v ukázce č. 4.14 pouze vracíme přístupové údaje ke kontroléru, které jsou uloženy v lokálních proměnných třídy `Credentials`. Tyto uživatelské údaje se však používají pouze u jediného API volání, a to pro získání autentizačního řetězce. Je-li tedy autentizace úspěšná, v odpovědi získáme řetězec, který poté používáme pro autentizaci u všech dalších API volání.

Dále následují řídicí metody určené jednotlivým kontrolérům. Jelikož jsou v této části metody pro APIC-EM a DNA-C velmi podobné, popíšeme si pouze ty, které souvisí s prvně jmenovaným modulem.

```
1  def run_program_APICEM(path):
2    base_path = path
3    #Vytvoření instance třídy Url s parametrem apicem_url
4    apicem_url = url.Url('apicem_url')
5    #Pokud není nastavena URL adresa kontroléru, vynutí její zadání
6    if not apicem_url.is_set():
7       if not Program.APICEMUrlSettings(path, apicem_url):
8          return
9    #Uložení adresy také do proměnné __UrlController třídy ApiAPICEM
10   apiapicem.ApiAPICEM.set_url(apicem_url.get_url())
11   #Spustí smyčku pro dialogové menu APIC-EM
12   while True:
13      #Proměnná informující o hloubce zanoření v menu
14      path = Program.__getAPICEMmenupath(base_path)
15      try:
16         #Zobrazení dalších možností v rámci kontroléru APIC-EM
17         Program.APICEMdialogMenu(path)
18         command = input("What do you want to do (enter a number or q)?")
19         ...
20
21 def __getAPICEMmenupath(path):
22   return path + ' > APIC-EM(' + apiapicem.ApiAPICEM.get_url() + ')'
23
24 def APICEMdialogMenu(path):
25   print("\n" + path + ":\n",
26      "1 - Create ticket\n",
27      "2 - Network devices\n",
28      "3 - EasyPnP\n",
29      "4 - Settings\n",
30      "5 - Back")
```

Ukázka 4.15: Část metody `run_program_APICEM` souboru `program.py`.

Hlavní statickou metodou kontroléru APIC-EM je `run_program_APICEM`, jejíž parametr `path` slouží k předávání informací o hloubce zanoření v menu. Uživatel má díky tomu přehled, ve které části programu EasyPnP se nachází. V ukázce č. 4.15 je zobrazena i vysvětlena hlavní část této metody. Součástí jsou také další dvě metody, které jsou volány na řádku 14 a 17. Nejdůležitějším článkem je řádek 7, kde se volá metoda `APICEMUrlSettings`, kterou si popíšeme v další ukázce č. 4.16.

```
1  def APICEMUrlSettings(path, apicem_url):
2    #Zobrazit informaci o zanoření v menu
3    print("\n" + path + " > Settings:")
4    if apicem_url.is_set():
5       print("APIC-EM address: " + apicem_url.get_url())
6    else:
7       print("APIC-EM address is not set, please configure!")
8    #Volba mezi IP nebo DNS záznamem
9    command = input("1 - IP\n 2 - DNS\n New controller?")
10   if command == "1":
11      #Do proměnné ip_input uložit zadanou IP adresu
12      ip_input = input("New IP of controller: ")
13      #Regulární výraz pro kontrolu validity zadané IP adresy
14      ip_pat = '^((\d|\d\d|1\d\d|[2][0-5][0-5])\.){3}
15                  (\d|\d\d|1\d\d|[2][0-5][0-5])\$'
16      #Pokud se nejedná o validní IP adresu, vynutit nové zadání
17      if not re.match(ip_pat, ip_input):
18          print("ERROR! Invalid IP format!")
19          return False
20      #Kontrola, zda se jedná o adresu zvoleného typu kontroléru
21      try:
22          result = apiapicem.ApiAPICEM.api_connection(ip_input)
23          #Pokud ne, vynutit nové zadání adresy
24          if result == False:
25              return False
26          #Pokud ano, uložit ji do cache_config i proměnné __UrlController
27          else:
28              print("Saved successfully!")
29              apicem_url.set_url(ip_input)
30              apiapicem.ApiAPICEM.set_url(ip_input)
31              return True
32      except Exception as e:
33          print("Something's wrong: " + str(e))
34   elif command == "2":
35      ...
```

Ukázka 4.16: Část metody `APICEMUrlSettings` souboru `program.py`.

Uvedená metoda se používá pro zadání URL adresy zvoleného kontroléru při prvním spuštění programu (dokud není vytvořen soubor `cache_config`) nebo při úmyslné změně této adresy. Máme na výběr mezi IP a DNS záznamem. Pokud zvolíme IP záznam, kontroluje se jeho validita pomocí regulárního výrazu, jenž je definovaný na řádku 14-15. Jedná se o speciální textový řetězec, který využívá různé zástupné znaky k prohledávání textu. Tím je zajištěno, aby uživatel zadal korektní IPv4 adresu. K porovnání, zda zadaná IP adresa odpovídá definovanému regulárnímu výrazu, používám standardní knihovnu re[7]. Regulární výrazy popisuje velké množství knih, webů či internetových článků. Osobně však doporučuji přečíst dokumentaci použité knihovny, uvedenou v poznámce pod čarou, kde

[7]Dokumentace na webu https://docs.python.org/3/library/re.html.

jsou jednotlivé položky podrobně vysvětleny. K testování vytvořených řetězců je vhodný například webový nástroj regex101.com, který nabízí přehledné grafické prostředí.

```
def APICEMgetNetworkDevices():
  #Uložení výsledku API volání do proměnné r_json
  r_json = apiapicem.ApiAPICEM.api_get_network_devices()
  if not r_json == False:
     print("\n" + "  View all devices in network:")
     #Proměnná pro indexování jednotlivých zařízení
     countDev = 0
     #Zobrazit hlavičku tabulky
     print('{!s:3}'.format('No:') + '{!s:15}'.format('Serial num:') + ...)
     try:
         #Zobrazit jednotlivá zařízení
         for i in r_json["response"]:
             countDev += 1
             print('{!s:3}'.format(str(countDev)) +
                   '{!s:15}'.format(i["serialNumber"]) + ...)
     except Exception as e:
         print("Something's wrong: " + str(e))
     while True:
         cmd = input("Which device do you want to see the interface?")
         if cmd == "q" or cmd == "Q":
             break
         try:
             cmd = int(cmd)
         except ValueError:
             continue
         try:
             #Na základě zadaného indexu vyplnit další proměnné
             if (cmd - 1 >= 0) and (cmd - 1 < len(r_json["response"])):
               deviceID = r_json["response"][cmd - 1]["id"]
               deviceHN = r_json["response"][cmd - 1]["hostname"]
               #Zobrazení všech rozhraní vybraného zařízení
               print("All interfaces of device " + deviceHN + ":")
               Program.APICEMgetNetworkDevicesInterfaces(deviceID, False)
               #Zobrazení aktivních rozhraní vybraného zařízení
               print("Only UP interfaces of device " + deviceHN + ":")
               Program.APICEMgetNetworkDevicesInterfaces(deviceID, True)
               break
         except Exception as e:
             print("Something's wrong: " + str(e))
```

Ukázka 4.17: Metoda `APICEMgetNetworkDevices` souboru `program.py`.

V ukázce č. 4.17 se nachází metoda pro zobrazení síťových zařízení v APIC-EM. Tuto funkci můžeme označit jako doplněk sloužící ke kontrole, zda jsou zařízení, po úspěšném provedení procesu Plug and Play, objevena procesem Discovery. U vybraného prvku si následně můžeme pomocí další metody `APICEMgetNetworkDevicesInterfaces` zobrazit jeho rozhraní.

### 4.4.3 Modulová část

Třetí a současně poslední logickou částí nástroje EasyPnP jsou moduly pro jednotlivé kontroléry. Dle souborové struktury na obr. č. 4.1 se momentálně jedná o modul pro kontrolér Cisco APIC-EM (složka `pnp_apicem`) a modul pro kontrolér Cisco DNA-C (složka `pnp_dnac`).

Jak bylo zmíněno v podkapitole 4.2.2, kontrolér DNA-C v sobě zahrnuje APIC-EM, jako součást zprostředkující komunikaci mezi grafickým uživatelským rozhraním DNA-C a síťovými prvky. Při vývoji programu EasyPnP, konkrétně jeho modulu DNA-C, přinášel kontrolér verze 1.1.7 velmi podobnou PnP funkci jako byla v nejnovějších verzích 1.6.2 kontroléru APIC-EM. Modul DNA-C je tedy velmi podobný modulu APIC-EM. Rozdíly v jednotlivých modulech jsou nepatrné. Liší se v zachytávání výjimek, odpovědí kontroléru a při definici knihovny request (detailnější popis bude následovat) zprostředkující dílčí API volání. Protože je DNA-C stále aktivně vyvíjenou platformou, v každé nové verzi dochází u PnP funkce ke změně struktury jednotlivých API volání. Jakmile se tento vývoj ustálí, díky modulárnosti programu EasyPnP můžu kód jednoduše upravit. Nyní je tedy modul DNA-C funkční pro stejnojmenný kontrolér jen do verze 1.1.7. Z tohoto důvodu budu v následujícím textu popisovat pouze modul pro kontrolér APIC-EM. Soubory `token.py`, `pnp_dnac.py` ani `apidnac.py` nebudou součástí tohoto dokumentu, avšak můžeme vycházet z popisu obdobných souborů `ticket.py`, `pnp_apicem.py` a `apiapicem.py` uvedených dále v textu.

- **ticket.py**

Příslušná třída `Ticket` spravuje autentizační řetězec, jenž se používá pro všechna API volání mimo jediného, u něhož se tento uživatelský identifikátor získává. Autentizační řetězec se společně s URL adresou příslušného kontroléru ukládá na lokální disk do textového souboru `cache_config`.

```
1  from easypnp_networksys_cz.model import cache
2  from easypnp_networksys_cz.pnp_apicem import apiapicem
3  class Ticket(cache.Cache):
4     #Vytvoření konstruktoru
5     def __init__(self):
6       #Mnohodědičnost, vyvolá inicializaci třídy Cache s def. parametrem
7       super(Ticket, self).__init__('ticket')
8       #Při inicializaci uložit do proměnné __ticket hodnotu z cache
9       self.__ticket = self.read_param('ticket')
10
11    #Vrátí textovou reprezentaci vlastní instance __ticket
12    def __str__(self):
13      return str(self.__ticket)
14
15    #Vytvoří nový aut. řetězec a uloží jej do proměnné __ticket i do cache
16    def get_new_ticket(self):
17      self.__ticket = apiapicem.ApiAPICEM.api_get_ticket()
```

```
18        self.save_param('ticket', self.__ticket)
19        return self.__ticket
20
21    #Vrátí autentizační řetězec uložený v proměnné __ticket
22    def get_ticket(self):
23        return self.__ticket
```

Ukázka 4.18: Celý obsah souboru `ticket.py`.

Na začátku ukázky č. 4.18 importuji třídu `Cache`, ze které dědím funkce, a třídu `ApiAPICEM`, z níž na řádku 17 volám metodu pro získání autentizačního řetězce. Při inicializaci se aktivuje třída `Cache` a do lokální proměnné `__ticket` se uloží aktuální hodnota autentizačního řetězce ze souboru `cache_config`. Poté se automaticky vrací jeho textová reprezentace. Následuje metoda obstarávající získání nového autentizačního řetězce a metoda pro navrácení obsahu z lokální proměnné. Obě tyto funkce jsou volány pouze třídou `ApiAPICEM`.

Řetězec se ukládá do lokální proměnné `__ticket`, aby se v rámci běhu programu nemusela číst jeho hodnota ze souboru `cache_config` pro každé API volání zvlášť. Pokud však dojde k ukončení programu EasyPnP, autentizační řetězec je zachován pouze ve zmíněném souboru. Při dalším spuštění programu se načte uložená hodnota a aktualizuje se v případě vypršení platnosti.

- **pnp_apicem.py**

Jedná se o soubor, který spravuje modul kontroléru APIC-EM a jeho veškeré PnP funkce (ty budou představeny v podkapitole 5.2.2). Soubor obsahuje třídu `PnP_APICEM`, jejíž součástí je 32 statických metod, a čítá cca 1200 řádků kódu. Z tohoto důvodu zde vysvětlím pouze několik klíčových metod.

Třída `PnP_APICEM` je rozdělena do 4 logických kódových bloků, tj. `Menu`, `Project`, `Configuration` a `Device`. V každém bloku se nachází metody obstarávající úkony dle jejich zařazení a názvu. Jednotlivé metody využívají funkce z modelové části a pro práci s kontrolérem volají funkce třídy `ApiAPICEM`, které provádí konkrétní API volání. Většina metod třídy `PnP_APICEM` zpracovává poskytnuté výsledky nebo vytváří obsah pro tato API volání. Vazby jsou znázorněny v blokové architektuře na obr. č. 4.2.

```
1  from easypnp_networksys_cz.model import directory
2  from shutil import copyfile
3  # Metoda zobrazující hlavní menu funkce EasyPnP
4  def pnp_APICEM_menu_main(path):
5      path = path + " > EasyPnP"
6      # Spustí smyčku pro výběr možnosti
7      while True:
8          try:
9              # Zobrazí možnosti hlavního menu funkce EasyPnP
10             PnP_APICEM.pnp_APICEM_menu_main_dialog(path)
11             command = input("What do you want to do (enter a number or q)?")
```

```
12          if command == "1":
13              try:
14                  # Do proměnné uloží cestu k vybrané XLS tabulce
15                  user_tab = directory.Directory.pick_file_dir("xlsx")
16                  if user_tab is None:
17                      print("ERROR! File with devices was not uploaded!")
18                  else:
19                      # Překopíruje XLS tabulku do lokálního umístění
20                      copyfile(user_tab, PnP_APICEM.devices_tab)
21                      print("File with devices was uploaded!")
22              except Exception as e:
23                  print("Something's wrong: " + str(e))
24          elif command == "2":
25              ...
26
27  def pnp_APICEM_menu_main_dialog(path):
28    print("\n" + path + ":\n",
29       "1 - Upload XLS\n",
30       "2 - Upload template\n",
31       "3 - Project\n",
32       "4 - Configuration\n",
33       "5 - Device\n",
34       "6 - Back")
```

Ukázka 4.19: Část metody `pnp_APICEM_menu_main` souboru `pnp_apicem.py`.

Součástí logického kódového bloku `Menu` je 10 statických metod, které zprostředkují zobrazení jednotlivých dialogových menu. Příkladem je metoda `pnp_APICEM_menu_main` v ukázce č. 4.19, jejíž parametr `path` informuje uživatele o umístění v menu nástroje EasyPnP. Na řádku 10 se volá metoda zobrazující možnosti tohoto menu, která je rovněž součástí ukázky. Řádek 15 volá metodu třídy `Directory` k vybrání XLS souboru a řádek 20 zajišťuje jeho překopírování do lokálního umístění. K tomuto účelu využívám knihovnu shutil[8], konkrétně její modul `copyfile`. Důvodem je stálost cesty k souboru a jeho uchování po ukončení programu. Totožně probíhá i výběr konfigurační šablony, pouze se metoda `pick_file_dir` volá s parametrem `jnj`.

Logický kódový blok `Project` obsahuje 6 statických metod, jež zajišťují všechny úkony související s projektem. Patří sem zobrazení všech projektů v kontroléru, vytvoření vlastního projektu, jeho smazání a vyčištění.

```
1  import os.path
2  import jinja2
3  # Metoda pro vytváření konfigurací
4  def pnp_APICEM_make_configuration_excel():
5    # Kontrola načtení XLS tabulky a konfigurační šablony
6    if not os.path.isfile(devices_tab) or not os.path.isfile(template_file):
7       if not os.path.isfile(devices_tab):
```

[8]Dokumentace na webu https://docs.python.org/3/library/shutil.html.

```
8           print("ERROR! The table of devices was not loaded correctly!")
9       if not os.path.isfile(template_file):
10          print("ERROR! The template was not loaded correctly!")
11      return
12  try:
13      # Načítání dat z konfigurační šablony pomocí knihovny jinja2
14      templateLoader = jinja2.FileSystemLoader(searchpath=".")
15      templateEnv = jinja2.Environment(loader=templateLoader)
16      templateFinal = templateEnv.get_template(template_file)
17  except Exception as e:
18      print("Something's wrong: " + str(e))
19  try:
20      # Načítání dat z XLS tabulky (proměnná data_set je seznam slovníků)
21      data_set = PnP_APICEM.pnp_APICEM_read_excel(devices_tab)
22      try:
23          cycle = 0
24          number = 0
25          # Vytváření konfigurací dle jednotlivých slovníku v data_set
26          for i in data_set:
27              # Vyplnění proměnných v šabloně dle hodnot prvního slovníku
28              outputText = templateFinal.render(i)
29              conf_path = PnP_APICEM.conf_folder + i['hostName'] + '_conf'
30              conf_name = i['hostName'] + '_conf'
31              cycle += 1
32              # Existuje-li konfigurace stejného jména
33              if os.path.isfile(conf_path):
34                  print("File " + conf_name + " ..., rewriting!")
35                  number += 1
36                  try:
37                      # Přepiš konfiguraci uvnitř lokální složky
38                      with open(conf_path, 'w') as conf_file:
39                          conf_file.write(outputText)
40                  except Exception as e:
41                      print("Something's wrong: " + str(e))
42              else:
43                  try:
44                      # Ulož nově vytvořenou konfiguraci do lokální složky
45                      with open(conf_path, 'w') as conf_file:
46                          conf_file.write(outputText)
47                      print("File " + conf_name + " was created!")
48                      number += 1
49                  except Exception as e:
50                      print("Something's wrong: " + str(e))
51          print("STATS! Successfully: " + str(number) + " / " + str(cycle))
52      except Exception as e:
53          print("Something's wrong: " + str(e))
54  except Exception as e:
55      print("Something's wrong: " + str(e))
```

Ukázka 4.20: Metoda `pnp_APICEM_make_configuration_excel` souboru `pnp_apicem.py`.

Nejzásadnější metoda třídy `PnP_APICEM` je zobrazena v ukázce č. 4.20. Tato metoda spadá do logického kódového bloku `Configuration` a zajišťuje vytváření jednotlivých konfigurací dle konfigurační šablony a řádků XLS tabulky. Součástí jsou dále metody ke čtení tabulky, zobrazení všech konfigurací v kontroléru, jejich nahrávání a mazání. Kódový blok `Configuration` obsahuje dohromady 5 statických metod.

Na začátku metody (řádek 6) se kontroluje knihovnou os.path[9], zda byla správně vložena XLS tabulka a konfigurační šablona. Bez těchto dvou souborů nelze vytvářet konfigurace. Na řádcích 14-16 se načítají data z konfigurační šablony pomocí knihovny Jinja2. Tento název již známe z podkapitoly 4.3.2, kde byl představen stejnojmenný jazyk k psaní šablon. Řádkem 21 se načítají data z XLS tabulky využitím metody, jež je vysvětlena v ukázce č. 4.21. Práci se souborem obstarává knihovna xlrd[10], konkrétně její modul `open_workbook`. Tato metoda převádí jednotlivé řádky tabulky na tzv. seznam slovníků (angl. list of dictionaries), který se ukládá do proměnné `data_set`. Díky tomu lze dílčí slovníky (jeden slovník se rovná jednomu řádku tabulky) indexovat jako položky seznamu. Na řádku 28 se poté vyplní proměnné v šabloně dle definovaných hodnot uvnitř slovníku. Vytvořené konfigurace se ukládají do adresáře `configurations` zvoleného kontroléru (viz obr. č. 4.1), aby bylo možné je zkopírovat na požadované místo. Nástroj EasyPnP lze tedy používat i pro jednoduchou a automatickou tvorbu konfigurací, aniž by bylo nutné využívat dalších funkcí.

```
1  from xlrd import open_workbook
2  # Metoda ke čtení XLS tabulky
3  def pnp_APICEM_read_excel(devices_tab):
4    # Načítání dat pomocí knihovny xlrd
5    excel = open_workbook(devices_tab)
6    # Prázdný seznam
7    data = []
8    for s in excel.sheets():
9       # Pro každý řádek tabulky
10      for row in range(1, s.nrows):
11          # Vytvoří seznam s parametry jednotlivých sloupců nultého
12          # řádku tabulky ("hostName", "serialNumber", "ipAddress" atd.)
13          col_names = s.row(0)
14          # Vytvoří prázdný slovník
15          col_value = {}
16          # Pro každý název sloupce ze seznamu col_names
17          for name, col in zip(col_names, range(s.ncols)):
18              # Uloží hodnotu sloupce aktuálního řádku do proměnné value
19              value = s.cell(row, col).value
20              # Převede hodnotu na textový řetězec
21              try:
22                  value = str(int(value))
23              except:
24                  pass
```

[9] Dokumentace na webu https://docs.python.org/3/library/os.path.html.
[10] Dokumentace na webu https://xlrd.readthedocs.io/en/latest/.

```
25                # Přidá dvojici ("název":"hodnota") do slovníku col_value
26                col_value.setdefault(name.value, value)
27            # Přidá nový slovník do seznamu "data" a pokračuje dalším řádkem
28            data.append(col_value)
29    # Metoda vrací seznam slovníků
30    return data
```

Ukázka 4.21: Metoda `pnp_APICEM_read_excel` souboru `pnp_apicem.py`.

Logický kódový blok `Device` zastřešuje 11 statických metod pro veškeré úkony s nasazovanými zařízeními. Patří sem metody zprostředkující zobrazení všech síťových prvků v projektu, a také metody zajišťující vytváření definovaných zařízení jak ve zvoleném kontroléru, tak zároveň v Cisco ISE. Nejrozsáhlejší metody souvisí s mazáním zařízení z projektu, kde je na výběr jejich jednotlivé nebo hromadné odstranění. Součástí této funkce je také možnost vyčištění a restartování prvků třídou `Client`.

```
1  import base64
2  # Metoda k vytváření parametrů pro přidání zařízení do APIC-EM a ISE
3  def pnp_APICEM_get_project_for_post_devices_toISE():
4    r_json = apiapicem.ApiAPICEM.api_get_pnp_project()
5    ...
6    while True:
7      cmd = input("In which project do you want to create the devices?")
8      ...
9      if (cmd - 1 >= 0) and (cmd - 1 < len(r_json["response"])):
10       projectID = r_json["response"][cmd - 1]["id"]
11       # Volání metody pro získání přihlašovacích údajů k ISE
12       if PnP_APICEM.pnp_APICEM_get_ISE_infomation() == False:
13          break
14       if not os.path.isfile(PnP_APICEM.devices_tab):
15          print("ERROR! The table of devices was not loaded correctly!")
16          return
17       else:
18          ise_addr = credentials.Credentials.ise_address
19          ise_user = credentials.Credentials.ise_username
20          ise_pass = credentials.Credentials.ise_password
21          # Načítání dat z XLS tabulky
22          data_set = PnP_APICEM.pnp_APICEM_read_excel(devices_tab)
23          # URL adresa k vytváření zařízení v ISE
24          ise_url = "https://"+ ise_addr +":9060/ers/config/networkdevice"
25          # Přihlašování do ISE pomocí řetězce vzniklého kódováním base64
26          ise_cred = (ise_user + ":" + ise_pass).encode("utf-8")
27          ise_cred_b64 = (base64.b64encode(ise_cred)).decode("utf-8")
28          # API volání zprostředkované třídou ApiAPICEM
29          apiapicem.ApiAPICEM.api_post_pnp_project_devices_toISE(...
30              ...projectID, data_set, ise_cred_b64, ise_url)
31          break
```

Ukázka 4.22: Část metody `pnp_APICEM_get_project_for_post_devices_toISE`.

V ukázce č. 4.22 se nachází část metody, která vytváří parametry pro nasazení definovaných zařízení ve zvoleném kontroléru a zároveň pro jejich přidání do Cisco ISE. Nejprve je nutné zvolit projekt, do něhož budou zařízení přidána. Poté se volají metody k získání přihlašovacích údajů do ISE, kontroluje se vložení tabulky a vytváří se lokální proměnné. Řádek 24 definuje URL adresu. Na řádcích 26 a 27 se vytváří, využitím knihovny base64[11], řetězec pro přihlašování k ISE dle zadaných údajů. Nakonec se URL adresa, řetězec a další parametry předávají metodě třídy `ApiAPICEM`, jež vytvoří záhlaví (angl. header) i potřebný obsah (angl. payload) pro konkrétní API volání.

- **apiapicem.py**

Soubor obsahuje třídu `ApiAPICEM` a v ní 21 statických metod, které zpracovávají veškeré žádosti o API volání, a poté vrací odpovědi metodám modulové i programové části. Třída `ApiAPICEM` čítá přes 900 řádků kódu, proto jsou metody rozděleny do 4 logických kódových bloků dle jejich účelu, tj. `System`, `Project`, `Configuration` a `Device`. Mimo tyto bloky se nachází pouze dvě metody, uvedené v ukázce č. 4.23. Ty zajišťují práci s lokálními proměnnými, aby se v rámci běhu programu nemusela pro každé API volání načítat URL adresa kontroléru ze souboru `cache_config`. Obě metody jsou volány pouze třídou `Program`.

```
1  from easypnp_networksys_cz.pnp_apicem import ticket
2  class ApiAPICEM:
3    # Lokální proměnné
4    __ticket = ticket.Ticket()
5    __urlController = ''
6    __urlControllerAPI = ''
7
8    # Veřejná metoda, jež nastavuje lokální proměnné pro URL adresu
9    def set_url(new_url):
10       ApiAPICEM.__urlController = new_url
11       ApiAPICEM.__urlControllerAPI = "https://" + new_url + "/"
12
13   # Veřejná metoda, jež vrací URL adresu kontroléru
14   def get_url():
15       return ApiAPICEM.__urlController
```

Ukázka 4.23: Metody `set_url` a `get_url` souboru `apiapicem.py`.

Téměř všechny vlastnosti kontroléru jsou přístupné prostřednictvím severního rozhraní (vysvětleno v podkapitole 1.2.3), kde mezi nejběžnější patří RESTful API využívající protokol HTTP. Rozhraní API představuje soubor pravidel, který umožňuje programový přístup k funkcím vybraného zařízení (v tomto případě síťového kontroléru). REST je sada pokynů, která určuje, jak má API vypadat. Čtyři základní funkce CRUD (Create, Read, Update, Delete) jsou převedeny na HTTP metody typu POST (přidání), GET (čtení), PUT (změna) a DELETE (mazání). Každé API volání se skládá z několika parametrů [57]:

[11] Dokumentace na webu https://docs.python.org/3/library/base64.html.

- Zařízení* (angl. endpoint) – jedinečná URL adresa
- Metoda* (angl. method) – vybraný HTTP požadavek
- Záhlaví* (angl. header) – jednotlivé instrukce
- Parametry (angl. parameters) – upřesňující parametry API volání
- Data (angl. payload) – odesílané informace

Povinné parametry jsou označeny hvězdičkou. Dílčí API volání se nazývá požadavek (angl. request), získaná data se nazývají odpověď (angl. response). Každý požadavek musí splňovat parametry dle typu volání. Při všech API voláních pracujeme se strukturovaným formátem JSON (JavaScript Object Notation), který je vhodný pro programové a strojové zpracování. Zároveň je poměrně jednoduše čitelný (datová struktura je typu „slovník“). Někdy se také používá formát XML (eXtensible Markup Language).

```
1  import json
2  import requests
3  # Metoda k přidání zařízení do PnP projektu kontroléru APIC-EM
4  def api_post_pnp_project_devices(projectID, data_set):
5    try:
6      # Proměnné pro konečné statistiky
7      cycle, number, num_conf, num_im = 0
8      # Do n-tice se uloží všechny konfigurace nahrané v APIC-EM
9      confTuple = ApiAPICEM.api_get_pnp_files("config")
10     # Pro každý dílčí slovník v proměnné data_set
11     for i in data_set:
12       cycle += 1
13       # Nalezne ID konfigurace porovnáním záznamů ve slovníku a n-tici
14       try:
15          confName = i["hostName"] + "_configuration"
16          configId = [id for cn, id in confTuple if cn == confName][0]
17       except Exception as e:
18          configId = None
19       ...
20       # Vytvoří záhlaví s instrukcemi a autentizačním řetězcem
21       header = {"content-type": "application/json",
22                 "X-Auth-Token": ApiAPICEM.__ticket.get_ticket()}
23       # Vytvoří konkrétní obsah potřebný k metodě POST
24       payload = [{"hostName": i["hostName"],
25                   "serialNumber": i["serialNumber"],
26                   "platformId": i["platformId"]}]
27       # Přidá ID patřičné konfigurace k vytvořenému obsahu
28       if configId is not None:
29          payload[0]["configId"] = configId
30          print("Configuration "+ confName +" was added to the device")
31          num_conf += 1
32       else:
33          print("There is no configuration "+ confName +" in the APIC-EM")
34          pass
35       ...
```

```
36        try:
37            # Provede API volání se všemi potřebnými parametry
38            response = requests.post(ApiAPICEM.__urlControllerAPI +
39                'api/v1/pnp-project/' + projectID + '/device',
40                data=json.dumps(payload), headers=header, verify=False)
41            # Zobrazí stavový HTTP kód odpovědi
42            print(str(response))
43            # Převede odpověď do strukturovaného formátu JSON
44            r_json = response.json()
45            # Získá informace o dokončeném API volání
46            result = ApiAPICEM.api_get_task(r_json["response"]["taskId"])
47            ...
48      # Zobrazí konečné statistiky informující o výsledcích API volání
49      print("\n" + "STATS!" ...)
50    except Exception as e:
51      print("Something's wrong: " + str(e))
```

Ukázka 4.24: Část metody `api_post_pnp_project_devices` souboru `apiapicem.py`.

Metoda k přidávání zařízení do PnP projektu kontroléru APIC-EM je zkráceně zobrazena v ukázce č. 4.24. Řádkem 9 je vytvořena n-tice (angl. tuple) obsahující dvojice (název, ID) všech konfigurací v kontroléru. Tato n-tice se na řádku 16 porovnává s názvem konfigurace v proměnné `data_set` (převedené údaje z XLS tabulky). Je-li nalezena shoda, uloží se patřičné ID do proměnné `configId`. Na řádku 21-22 se vytváří záhlaví s instrukcemi a autentizačním řetězcem. Používáme-li HTTP metodu typu POST, musíme vytvořit obsah, který se bude nahrávat na kontrolér. To se děje na řádcích 24-26. Rovněž na řádku 29 se k vytvořenému obsahu přidává zmiňovaný parametr `configId`. Samotné API volání se provádí řádkem 38. K tomu využívám standardní knihovnu requests[12], která se stala velmi populární pro svoji jednoduchost. Knihovna zpracuje všechny parametry, odešle vybraný HTTP požadavek a následně vrací odpověď. Ta je na řádku 44 převedena do strukturovaného formátu JSON. V této ukázce je použita i knihovna json[13], kdy se metodou `json.dumps(payload)` převede vytvořený obsah do textového řetězce, aby byly splněny podmínky pro toto API volání. Stavový HTTP kód na řádku 42 udává, zda požadavek proběhl v pořádku.

K vytváření definovaných zařízení v Cisco ISE se používá jiná metoda, která je ovšem velmi podobná té na ukázce č. 4.24. Rozdíl je pouze v instrukcích záhlaví a ve formátu obsahu. Aby proběhlo REST API volání na Cisco ISE, je nutné vytvořit speciálního uživatele patřícího do skupiny ERS Admin, který má povolený API přístup pro čtení a zápis. Takového uživatele vytvoříme (v Cisco ISE verze 2.3) v části Administration –> Admin Access –> Administrators –> Admin Users.

[12]Dokumentace na webu https://2.python-requests.org/en/master/.
[13]Dokumentace na webu https://docs.python.org/3/library/json.html.

# 5 NÁSTROJ EASYPNP

Produkt EasyPnP v kombinaci s vybraným síťovým kontrolérem efektivně řeší problém přípravy složitého konfiguračního CSV souboru a dokáže výrazně ulehčit proces hromadného přidávání nových či stávajících zařízení do sítě, ušetřit velké množství času a eliminovat chyby způsobené manuální instalací každého zařízení zvlášť. Je vhodný pro techniky a síťové administrátory, kteří chtějí vyzkoušet nasazování zařízení do provozu s co nejmenším úsilím.

## 5.1 Testování nástroje

Ke správnému testování či využívání nástroje EasyPnP je nutné dodržovat určitá pravidla, která jsou popsána v následujících podkapitolách. Jejich korektním dodržením je zaručena spolehlivá funkčnost nástroje.

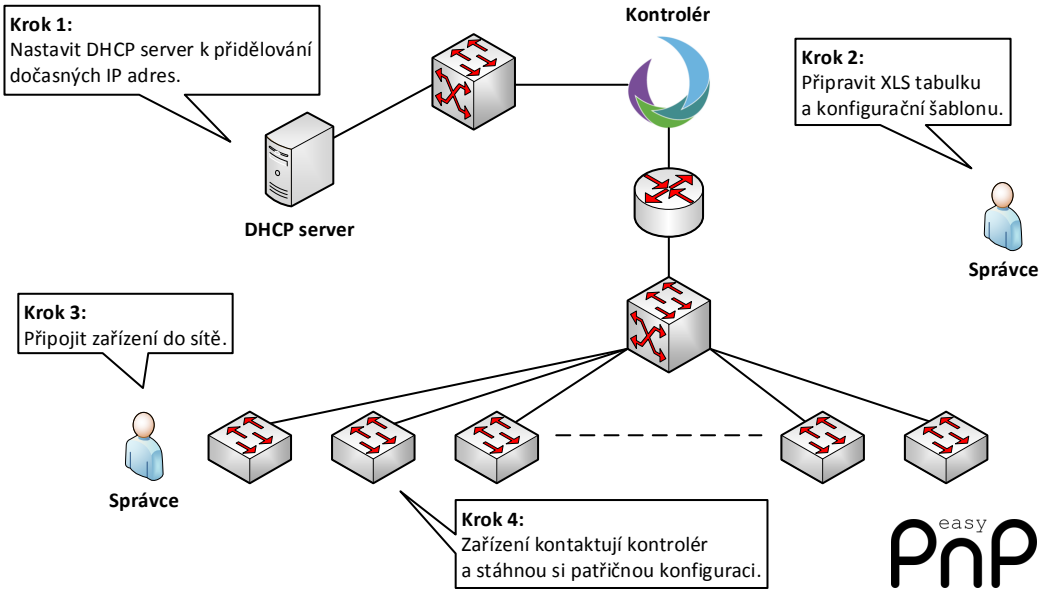
### 5.1.1 Klíčové předpoklady

Hlavní předpoklady nástroje EasyPnP byly představeny v podkapitole 4.3.2. Jedná se o vyplněnou XLS tabulku se základními údaji o konfigurovaných zařízeních a konfigurační šablonu s definovanými proměnnými. Na základě těchto dvou vstupů jsou nástrojem vygenerovány konfigurace pro jednotlivá zařízení dle řádků tabulky. Pro následné kroky však musíme zabezpečit další sounáležitosti. Mluvím především o zajištění dosažitelného připojení ke zvolenému kontroléru, aby bylo možné na něj nahrát vygenerované konfigurace a vytvořit pravidla pro proces Plug and Play. Dále je nutné, pro správné použití PnP funkce, nastavit DHCP (Dynamic Host Configuration Protocol) server k přidělování dočasných IP adres novým zařízením, aby mohly komunikovat s kontrolérem. Výměna informací probíhá při výchozím nastavení ve VLAN 1. Chceme-li využívat jinou virtuální síť, musíme na přepínač, do něhož zapojujeme nová zařízení, přidat speciální příkaz. Ten vidíme v ukázce č. 5.1. Součástí je také příklad DHCP rozsahu pro přidělování prvotních IP adres.

```
# Vytvoření DHCP serveru pro přidělování prvotních IP adres
ip dhcp pool APIC-PNP
   network 10.22.31.0 255.255.255.0
   default-router 10.22.31.1
   dns-server 10.22.22.22
   # Zde se definuje adresa zvoleného kontroléru
   option 43 ascii "5A1D;B2;K4;I10.22.30.11;J80"

# Příkaz pro změnu výchozí VLAN procesu PnP
pnp startup-vlan 101
```

Ukázka 5.1: Základní předpoklady použití PnP funkce.




Obr. 5.1: Schéma celého PnP procesu.

Kompletní aplikační proces je zobrazen ve čtyřech krocích na obr. č. 5.1. V pravém dolním rohu si můžeme povšimnout reklamního loga nástroje EasyPnP. Ten v uvedeném procesu zajišťuje nejen generování konfigurací na základě vložené tabulky a šablony, ale i nahrávání vytvořených konfigurací spolu s pravidly pro jednotlivá zařízení do vybraného kontroléru, volitelně také do Cisco ISE. Zároveň skrze něj můžeme spravovat celý PnP proces bez nutnosti otevírat grafické uživatelské rozhraní kontroléru. Všechny funkce a vlastnosti nástroje EasyPnP budou popsány v sekci 5.2. Jakmile zařízení kontaktuje kontrolér, přiřadí se mu jeho připravené pravidlo (tedy konfigurace, operační systém a všechny další definované parametry) na základě unikátního sériového čísla. Následně proběhne automatické nasazení (angl. deployment, provisioning). Tím je PnP proces dokončen a zařízení je připraveno k běžnému použití.

### 5.1.2 Ověřená prostředí

Nástroj EasyPnP jsem vyvíjel a testoval v programovacím jazyce Python verze 3.6, která není s verzí 2.7 kompatibilní a její podpora se neplánuje. Pro správnou funkčnost nástroje je tedy nutné používat Python alespoň ve verzi 3 a vyšší.

Pro testování REST API volání modulu APIC-EM, a zároveň celé funkce Plug and Play, jsem používal stejnojmenný kontrolér v různých verzích. Poslední nejvyšší odzkoušená verze je 1.6.2 s vnitřní aplikací PnP verze 1.6.3. Testování probíhalo na kontroléru společnosti Networksys a.s. Dále je možné využít také volně dostupný kontrolér na adrese[1] uvedené v poznámce pod čarou (přístupové údaje jsou devnetuser/Cisco123!), který firma Cisco Systems poskytuje pro potřeby vývojářů.

[1]Kontrolér Cisco APIC-EM na webu https://sandboxapic.cisco.com/.

K testování REST API volání spolu s celkovou funkčnosti modulu DNA-C jsem používal stejnojmenný kontrolér nasazený opět ve společnosti Networksys a.s. Jak však bylo zmíněno v úvodu podkapitoly 4.4.3, v době vývoje tohoto modulu byl kontrolér DNA-C ve svém počátku, konkrétně ve verzi 1.1.7, pro niž je funkčnost nástroje odzkoušena. Během postupného vývoje této platformy však došlo ke znatelné změně struktury API volání pro PnP funkci. Aktuálně je tedy tento modul funkční pouze do výše uvedené verze. Po ustálení vývoje API volání uvedeného kontroléru je v plánu upravení kódu pro nejnovější verze. Znovu je možné využít, pro různé testy a seznámení se s touto platformou, volně přístupný kontrolér společnosti Cisco Systems na adrese[2] uvedené v poznámce pod čarou (přístupové údaje jsou devnetuser/Cisco123!).

### 5.1.3 Distribuce nástroje

Kompletní zdrojové kódy nástroje EasyPnP jsou volně dostupné ve webovém repozitáři GitHub[3], společně s popisem a dokumentací v anglickém jazyce. Součástí je také textový soubor `requirements.txt`, jenž obsahuje seznam všech potřebných balíčků a knihoven.

Na základě osobních zkušeností doporučuji v Pythonu využívat virtuální prostředí venv[4]. Jedná se o oddělené prostředí, kde instalace knihoven nezasahuje do systémového nastavení ani do jiných virtuálních prostředí. Tím je možné jednoduše oddělit různé projekty. V každém prostředí může být nainstalována jiná sada (či jiné verze) knihoven.

```
# Vytvoření virtuálního prostředí v MacOS nebo Linux
cd .../EasyPnP
python3.6 -m venv venv
# Spuštění venv                    # Vypnutí venv
source venv/bin/activate           (venv) deactivate

# Vytvoření virtuálního prostředí ve Windows (git-bash)
cd .../EasyPnP
py -3 -m venv venv
# Spuštění venv                    # Vypnutí venv
source venv/Scripts/activate       (venv) deactivate

# Instalace potřebných balíčků a knihoven při spuštěném venv
(venv) pip install -r requirements.txt
```

Ukázka 5.2: Vytvoření virtuálního prostředí a instalace knihoven.

V ukázce č. 5.2 jsou popsány všechny nezbytné příkazy. Nejprve je potřeba stáhnout nástroj EasyPnP z webového repozitáře do zvoleného lokálního umístění. Zde otevřeme vhodný terminál dle operačního systému a zadáme příkazy k vytvoření i spuštění virtuálního prostředí. V něm můžeme bezpečně nainstalovat potřebné balíčky a knihovny.

[2]Kontrolér Cisco DNA-C na webu https://sandboxdnac.cisco.com/.
[3]Dostupné na webu https://github.com/mikesm11/EasyPnP.
[4]Dokumentace na webu https://docs.python.org/3/library/venv.html.

## 5.2 Funkce nástroje

Nástroj EasyPnP je navržen jak pro interakci s kontrolérem, tak pro samostatné generování konfigurací. Hlavním účelem je však kompletní správa celého procesu Plug and Play, aby uživatel nemusel pracovat s GUI zvoleného kontroléru. Nástroj poskytuje všechny potřebné funkce, které v této kapitole postupně představím.

Doporučuji využít také videonávod[5], kde jsem nástroj detailně okomentoval a vysvětlil jeho veškeré funkce i možnosti. Video shrnuje problematiku PnP procesu a obsahuje ukázku automatického nasazení zařízení pomocí nástroje EasyPnP.

### 5.2.1 Spuštění nástroje

Program je koncipován jako dialogová komunikace v příkazovém řádku. Přestože hlavní myšlenkou SDN je výrazné opuštění od CLI, stále toto klasické uživatelské rozhraní zůstává velké většině síťových správců (a nejenom jim) nejbližší. Navíc je CLI vhodné pro vývoj a testování.

Spuštění nástroje provedeme tak, že v příkazovém řádku otevřeme složku, kde se program nachází. Poté spustíme hlavní inicializační soubor (při použití Python verze 3.6 příkazem `py main.py`), což můžeme vidět na obr. č. 5.2. Dostaneme se do hlavního menu, kde máme na výběr momentálně dva kontroléry, a to APIC-EM nebo DNA-C.

```
C:\Users\mikeska\Documents\GitHub\EasyPnP>py main.py

MENU:
 1 - APIC-EM
 2 - DNA-C
 3 - END
 Your choice (enter a number or q)? 1
```

Obr. 5.2: Hlavní menu nástroje EasyPnP.

Pohyb v programu se provádí „odentrováním“ čísla u příslušné položky. Pro návrat zpět nebo pro ukončení funkce se používá konkretní definované číslo nebo vždy malé či velké písmeno Q (z anglického quit). Zvolíme číslo 1 pro vstup do APIC-EM.

```
MENU > Settings:
 APIC-EM address is not set, please configure before continue!
 1 - IP
 2 - DNS
 New controller (enter a number or q)? 1
  New IP of controller: 10.22.30.11
   Saved successfully!
```

Obr. 5.3: První zadání URL adresy zvoleného kontroléru.

Jelikož se jedná o první běh programu, v textovém souboru `cache_config` není uložena žádná IP adresa zvoleného kontroléru. Je nutné ji tedy zadat. Zde volíme mezi IP a DNS záznamem. V obou případech je ošetřeno, aby uživatel zadal adresu vybraného kontroléru,

[5]Dostupné na webu https://youtu.be/GOxJ5qpehj4.

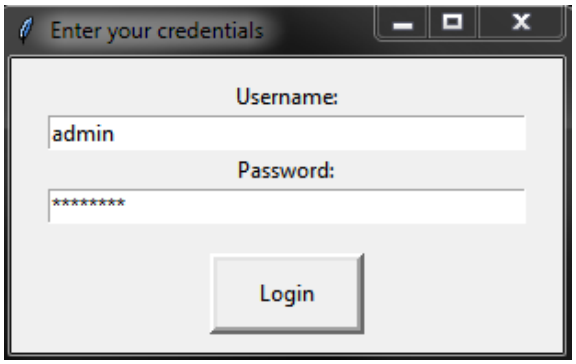
v tomto případě APIC-EM. Po zadání adresy se program snaží o jeho kontaktování. Pokud kontrolér neodpovídá, je vrácena chybová hláška. Jestliže je adresa správná a konektivita funkční, záznam se uloží do souboru `cache_config`, tak jako na obr. č. 5.3.

```
MENU > APIC-EM(10.22.30.11):
 1 - Create ticket
 2 - Network devices
 3 - EasyPnP
 4 - Settings
 5 - Back
 What do you want to do (enter a number or q)?
```

Obr. 5.4: Hlavní menu kontroléru APIC-EM.

Tím se přesuneme do hlavního menu kontroléru APIC-EM. Vrchní řádek zobrazuje zanoření v nástroji spolu s aktuální adresou (viz obr. č. 5.4). Dále volíme funkce dle příslušných čísel:

**1)** Ke správnému využívání všech REST API volání je potřeba vytvořit autentizační řetězec. Jedná se o jediné API volání, kde se posílají uživatelské údaje (angl. credentials). Zvolíme tedy možnost 1 a jsme požádáni o zadání přístupových údajů ke kontroléru pomocí dialogového okna (viz obr. č. 5.5). Je-li autentizace úspěšná, v odpovědi se vrátí autentizační řetězec, jenž se následně používá jako autentizace pro veškerá další API volání. Tento řetězec, neboli tiket v případě APIC-EM, ukládám lokálně do souboru `cache_config`, společně s URL adresou kontroléru. Záznamy ukládám při běhu programu také do různých proměnných, jež byly popsány v sekci 4.4. Výsledkem je, že u dalších API volání nejsme dotazováni o zadávání uživatelských údajů a URL adresy kontroléru.




Obr. 5.5: Dialogové okno k zadávání uživatelských údajů.

Vyprší-li časová platnost autentizačního řetězce, nástroj tuto situaci automaticky rozpozná. U jakéhokoliv dalšího API volání si vyžádá nové zadání uživatelských údajů. Tím se vytvoří platný autentizační řetězec a zároveň se provede požadovaný proces

**2)** Zobrazí seznam všech síťových zařízení ve zvoleném kontroléru. Slouží k ověření, zda automaticky nasazená zařízení byla objevena procesem `Discovery`. Následně můžeme indexem vybrat konkrétní prvek, u něhož chceme zobrazit rozhraní.

**3)** Obsahuje všechny důležité funkce nástroje EasyPnP pro úspěšné provedení celého Plug and Play procesu. Tato část je velmi obsáhla s velkým množstvím různých možností. Proto si ji detailně popíšeme v následující podkapitole 5.2.2.

**4)** Slouží k novému nastavení URL adresy zvoleného kontroléru. Funkce je stejná jako při prvotním zadávání adresy. Před uložením se kontroluje správnost záznamu a konektivita ke kontroléru.

**5)** Číslem 5, popřípadě zadáním malého či velkého písmena Q, se vrátíme zpět do hlavního menu nástroje EasyPnP.

### 5.2.2 Část EasyPnP

Volbou pod číslem 3 se přesuneme do PnP menu zvoleného kontroléru (v tomto případě APIC-EM). Zde máme na výběr další funkce, jak vidíme na obr. č. 5.6. Vrchní řádek poskytuje neustálý přehled o tom, ve které části programu se nacházíme.

```
MENU > APIC-EM(10.22.30.11) > EasyPnP:
 1 - Upload XLS
 2 - Upload template
 3 - Project
 4 - Configuration
 5 - Device
 6 - Back
 What do you want to do (enter a number or q)?
```

Obr. 5.6: PnP menu kontroléru APIC-EM.

Potřebné kroky k automatickému nasazení síťových zařízení jsou seřazeny intuitivně za sebou. Pod jednotlivými čísly se nachází tyto možnosti:

**1)** Prvním krokem je vložení tabulky se základními údaji o konfigurovaných zařízeních do nástroje EasyPnP. Přesnou strukturu i potřebné parametry tabulky jsme si podrobně představili v podkapitole 4.3.2. Tabulka musí být ve formátu XLS (přípona xlsx) a výběr probíhá pomocí grafického dialogového okna. Po úspěšném výběru souboru dojde k jeho zkopírování na lokální disk do složky `devices` zvoleného modulu (viz souborová struktura na obr. č. 4.1). Důvodem je stálost cesty k souboru a jeho uchování po ukončení běhu programu.

**2)** Druhým krokem je vložení konfigurační šablony ve formátu Jinja2 (přípona jnj), opět pomocí grafického dialogového okna. Ze stejného důvodu jako v prvním případě se vybraná šablona kopíruje na lokální disk, ale tentokrát do složky `template` zvoleného modulu nástroje EasyPnP. Všechny patřičné náležitosti konfigurační šablony i správnou definici proměnných jsme si vysvětlili v podkapitole 4.3.2.

**3) Projektová část:**

Ve třetím kroku musíme vytvořit PnP projekt, do něhož budeme přidávat pravidla pro automatické nasazení síťových zařízení.

```
MENU > APIC-EM(10.22.30.11) > EasyPnP > Project:
 1 - List projects
 2 - Create project
 3 - Delete project
 4 - Clean project
 5 - Back
 What do you want to do (enter a number or q)?
```

Obr. 5.7: Projektové menu kontroléru APIC-EM.

Na obr. č. 5.7 vidíme projektové menu zvoleného kontroléru. Jednotlivé položky zajišťují tyto funkce:

**1)** Zobrazí aktuální projekty v kontroléru. Součástí výpisu jsou informace o každém projektu – stav, název, počet zařízení, čas vytvoření atd.

**2)** Slouží k vytvoření nového projektu v kontroléru. Nástroj EasyPnP vyzve k zadání konkrétního názvu a poté provede API volání.

**3)** Umožňuje kompletní smazání projektu spolu se všemi přiřazenými zařízeními a konfiguracemi. To je obrovskou výhodou oproti GUI kontroléru APIC-EM, kde pokud smažeme projekt, který není prázdný, zařízení a konfigurace k němu zůstanou přiřazeny. Poté může nastat situace, kdy chceme tato zařízení nahrát do nového projektu, ale kontrolér nahlásí, že zařízení již existují v jiném projektu. Ať se snažíme jakkoliv, zařízení do něj nedostaneme.

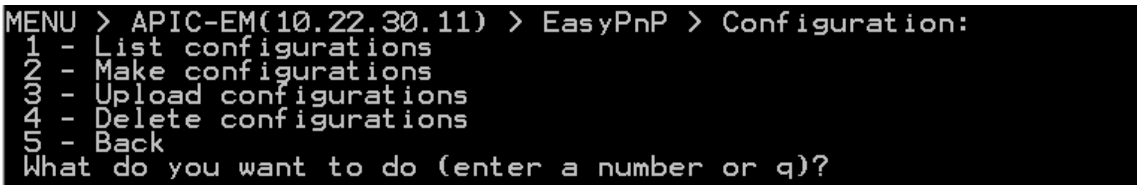
Výše uvedený problém řeší nástroj EasyPnP. Obsahuje-li projekt zařízení, při jeho mazání jsme na tuto skutečnost upozorněni. Následně musíme potvrdit, zda jej opravdu chceme smazat. Zvolíme-li ano, smažou se všechna zařízení i konfigurace přiřazené k tomuto projektu, poté se smaže celý projekt. Tím je zajištěno, že žádné zařízení nezůstane přiřazeno k neexistujícímu projektu.

**4)** Funguje podobně jako předchozí funkce s jediným hlavním rozdílem. Smažou se všechna zařízení i konfigurace přiřazené ke zvolenému projektu, ale samotný projekt zůstane, bude „vyčištěn“. Tato funkce řeší situaci, kdy projekt chceme smazat, ale následně jej znovu používat pod stejným názvem. Tím je ušetřen krok vytváření nového projektu.

**5)** Číslem 5, popřípadě zadáním malého či velkého písmena Q, se vrátíme zpět do PnP menu kontroléru APIC-EM.

**4) Konfigurační část:**

Čtvrtým krokem je vygenerování konfigurací a jejich nahrání na kontrolér. Nástroj EasyPnP zpracovává vloženou XLS tabulku i konfigurační šablonu a velmi rychle generuje konfigurace pro každý vyplněný řádek tabulky. Časová náročnost je minimální oproti ruční tvorbě konfigurací pro každé zařízení zvlášť.




Obr. 5.8: Konfigurační menu kontroléru APIC-EM.

Na obr. č. 5.8 vidíme konfigurační menu zvoleného kontroléru. Dle příslušných čísel volíme tyto funkce:

**1)** Zobrazí všechny konfigurace nahrané do kontroléru. Součástí výpisu jsou informace o každé konfiguraci – název, velikost a formát.

**2)** Slouží k samotnému generování konfigurací. Nástroj informuje uživatele, zda proces proběhl v pořádku, případně že konfigurace s tímto názvem již existovala a došlo k jejímu přepsání. Vygenerované konfigurace se ukládají na lokální disk do složky `configurations` zvoleného modulu (viz souborová struktura na obr. č. 4.1). Nezávisle na kontroléru lze nástroj EasyPnP využívat pro jednoduchou a automatickou tvorbu konfigurací, proto potřebujeme lokální přístup k těmto souborům.

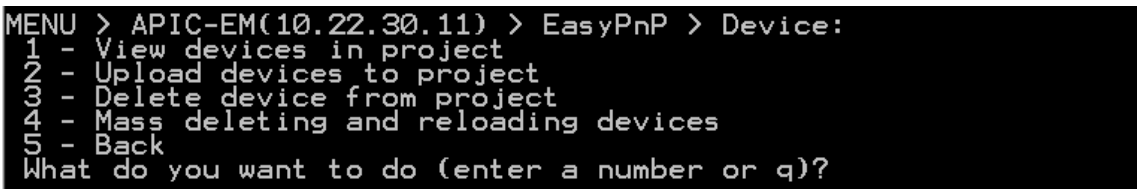
**3)** Zajišťuje nahrávání vygenerovaných konfigurací do kontroléru a o této skutečnosti informuje uživatele. Existuje-li v kontroléru konfigurace se stejným názvem, nástroj proces přeruší, aby nedošlo k neúmyslným změnám.

**4)** Zobrazí veškeré konfigurace nahrané do kontroléru. Zvolením patřičného indexu smažeme vybranou konfiguraci. Chceme-li smazat všechny, zadáme index 0.

**5)** Číslem 5, popřípadě zadáním malého či velkého písmena Q, se vrátíme zpět do PnP menu kontroléru APIC-EM.

**5) Část pro zařízení:**

Pátým a současně posledním krokem je vytvoření pravidel v PnP projektu zvoleného kontroléru pro automatické nasazení síťových zařízení. Volitelně můžeme zařízení přidat také do Cisco ISE. Rychlost procesu závisí na kvalitě datového připojení a na rychlosti zpracování jednotlivých API volání.




Obr. 5.9: Menu pro síťová zařízení kontroléru APIC-EM.

Na obr. č. 5.9 vidíme menu pro síťová zařízení zvoleného kontroléru. Konkrétní čísla obstarávají tyto funkce:

**1)** Zobrazí všechna zařízení ve vybraném PnP projektu daného kontroléru. Součástí výpisu jsou různé informace o zařízeních – název, platforma, sériové číslo, stav aj. Uživatel má díky tomu přehled, v jakém stavu se jednotlivá zařízení nachází. Existují tyto stavy:

- **Pending** – Čekající na připojení.
- **Start_provisioning** – Započetí procesu automatického nasazení.
- **Provisioning_image** – Nahrávání příslušného operačního systému.
- **Provisioning_config** – Nahrávání příslušné konfigurace.
- **Provisioned_config** – Ukládání nahrané konfigurace.
- **Provisioned** – Automatické nasazení proběhlo v pořádku.

**2)** Slouží k vytváření pravidel pro zařízení definovaná v XLS tabulce. Zde jsou na výběr dvě možnosti, a to vytvoření pravidel ve zvoleném kontroléru, případně ve zvoleném kontroléru i v Cisco ISE.

Vybereme-li první možnost, nástroj EasyPnP se zeptá, ve kterém PnP projektu chceme pravidla pro automatické nasazení vytvořit. Následně proběhnou všechna nutná API volání kontroléru a jsou zobrazeny výsledné statistiky. Není-li v kontroléru nalezena konfigurace pro nahrávané zařízení, nástroj nás vyzve, zdali přesto chceme pokračovat.

Vybereme-li druhou možnost, nástroj EasyPnP si opět vyžádá zadání PnP projektu, v němž chceme pravidla pro automatické nasazení vytvořit. Poté nás upozorní, že jsme zvolili možnost s vytvořením zařízení v Cisco ISE, což je nutné potvrdit. Tím je zobrazeno dialogové okno (podobné tomu na obr. č. 5.5), do něhož musíme zadat přístupové údaje k Cisco ISE společně s jeho IP či DNS adresou. Nejedná-li se o adresu Cisco ISE, uživatel je na tuto skutečnost upozorněn a proces se přeruší, stejně jako při zadání uživatelských údajů bez patřičného povolení (viz poslední odstavec podkapitoly 4.4.3). Jsou-li všechny uvedené požadavky splněny, proběhnou náležitá API volání a zobrazí se konečné statistiky. Zařízení jsou přidána do zvoleného kontroléru a zároveň vytvořena v Cisco ISE (odzkoušeno pro verzi 2.3) v části Administration –> Network Devices.

**3)** Zajišťuje mazání jednotlivých zařízení z vybraného PnP projektu daného kontroléru. Po zvolení projektu se zobrazí všechna zařízení, která jsou k němu přiřazena. Zde konkrétním indexem vyberu, které zařízení chci smazat. Nástroj se zeptá, zda chci zvolené zařízení také vyčistit a restartovat. Máme tři možnosti:

- **Q/q** – Proces mazání se přeruší.
- **N/n** – Zařízení se smaže z projektu a současně se z kontroléru odstraní konfigurace přiřazená k tomuto prvku.
- **Y/y** – Zobrazí se dialogové okno (podobné tomu na obr. č. 5.5), do něhož musíme zadat uživatelské údaje a IP adresu pro přístup k zařízení. Nástroj

EasyPnP se pokusí navázat SSH spojení. Je-li relace sestavena, do zařízení se pošlou příkazy `erase /all nvram:` a `reload` (detailněji vysvětleno v podkapitole 4.4.1 při popisu souboru `client.py`). Poté se smaže zařízení z projektu a odstraní se jeho konfigurace z kontroléru. Proces mazání se přeruší, pokud se SSH spojení z jakéhokoliv důvodu nepodaří sestavit.

Tato funkce řeší situaci, kdy uživatel nechce, například z důvodu bezpečnosti, vyplňovat své uživatelské jméno (angl. username) a uživatelské heslo (angl. password) do XLS tabulky, ale raději tyto údaje zadá skrytě do dialogového okna.

**4)** Umožňuje hromadné mazání i restartování zařízení z vybraného PnP projektu daného kontroléru. Funguje podobně jako předchozí funkce, ale k navázání spojení využívá údaje z XLS tabulky. Zvolením projektu se zobrazí zařízení, jež jsou k němu přiřazena. Nástrojem EasyPnP jsme dotázáni, zda chceme všechna zařízení smazat, vyčistit a restartovat. Máme dvě možnosti:

- **N/n** – Proces mazání se přeruší.
- **Y/y** – K navázání SSH spojení se využívají údaje z XLS tabulky, konkrétně `ipAddress`, `userName` a `passWord`. Předpokladem je proto vložená XLS tabulka s uvedenými parametry. Nejsou-li parametry vyplněny, nástroj EasyPnP požádá o jejich doplnění. Aby proběhlo smazání správného zařízení, provádí se dvojitá kontrola, kdy název zařízení (angl. hostname) i sériové číslo v tabulce musí odpovídat názvu i sériovému číslu v PnP projektu. Pokud jsou splněny všechny požadavky, nástroj EasyPnP se pokusí navázat SSH spojení. Je-li relace sestavena, do zařízení se automaticky zadají příkazy `erase /all nvram:` a `reload` (detailněji vysvětleno v podkapitole 4.4.1 při popisu souboru `client.py`), smaže se zařízení z projektu a odstraní se jeho konfigurace z kontroléru. Není-li splněn některý z požadavků, případně se nepodaří sestavit SSH relace, proces mazání přejde k dalšímu zařízení.

Díky této funkci můžeme jednoduše obměňovat konfigurace stávajících zařízení pro celý PnP projekt zvoleného kontroléru. Stačí tímto způsobem smazat a současně restartovat zařízení, vygenerovat nové konfigurace a zařízení opět přidat do PnP projektu.

Dále je možné hromadně mazat a restartovat vybraná zařízení. Vložíme novou XLS tabulku obsahující zařízení, jež chceme smazat. Poté provedeme kroky této funkce. Pokud je vše potřebné správně vyplněno, z PnP projektu se smažou zařízení uvedená v XLS tabulce a ostatní zůstanou. Musí však souhlasit název, sériové číslo a zároveň i IP adresa.

**5)** Číslem 5, popřípadě zadáním malého či velkého písmena Q, se vrátíme zpět do PnP menu kontroléru APIC-EM.

**6)** Číslem 6, popřípadě zadáním malého či velkého písmena Q, se vrátíme zpět do hlavního menu kontroléru APIC-EM.

Následováním uvedených pěti hlavních kroků dosáhneme vytvoření pravidel v PnP projektu zvoleného kontroléru. Poté stačí zařízení pouze připojit do sítě. Jakmile zařízení kontaktuje kontrolér, nalezne se shoda dle sériového čísla, přiřadí se mu jeho připravené pravidlo (tedy konfigurace, operační systém a další definované parametry) a proběhne automatické nasazení (angl. deployment, provisioning). Tím je celý proces Plug and Play dokončen a zařízení je připraveno k běžnému použití.

Ostatní funkce obstarávají kompletní správu celého PnP procesu, aby uživatel nemusel otvírat GUI kontroléru. Nástroj ošetřuje většinu chybových stavů, aby nedošlo k neúmyslným změnám v síti.

## 5.3 Nasazení nástroje

Nástroj EasyPnP lze využívat v kombinaci s vybraným kontrolérem pro automatické nasazení síťových zařízení nebo samostatně pro jednoduché generování konfigurací.

Jelikož je nástroj volně dostupný, mohou jej využívat jak specializované společnosti, tak samotní správci podnikových sítí. V této sekci doporučím dvě varianty použití a představím své poznatky z testování nástroje, které jsem získal při realizaci několika projektů v sítích reálných provozovatelů.

### 5.3.1 Varianty nasazení

Jedná se o osobní doporučení, jak nejlépe nabízet nástroj EasyPnP svým zákazníkům. Existují dvě varianty:

- **Konfigurace uvnitř sítě**
  Podmínkou této varianty je instalace vybraného kontroléru v síti zákazníka, aby bylo možné využívat PnP funkci. Nástroj EasyPnP komunikuje s kontrolérem buď lokálně nebo vzdáleně pomocí VPN (Virtual Private Network). Dle požadavků zákazníka je připravena XLS tabulka i konfigurační šablona a jsou vytvořena patřičná pravidla v kontroléru. Zákazník poté pouze připojí zařízení do libovolného místa v síti, což spustí jeho automatickou konfiguraci.
- **Předkonfigurování dodávaných zařízení**
  Pro tuto variantu není nutné instalovat kontrolér do sítě zákazníka. Dle objednávky je připravena XLS tabulka se sériovými čísly dodávaných zařízení. Zákazník vyplní zbylé parametry a poskytne konfigurační šablonu, případně ukázkovou konfiguraci. Na základě těchto souborů se zákazníkovi předají zařízení, která jsou již připravená k přímému zapojení do sítě.

### 5.3.2 Poznatky z testování

Během testování nástroje EasyPnP a při nasazení v reálných sítích jsem čelil celé řadě nepředvídatelných problémů. Přepínač, do něhož zapojujeme zařízení k automatickému nasazení, budu dále v textu označovat jako „upstream“ přepínač. Mezi základní poznatky patří:

- Rozhraní upstream přepínače, kam připojujeme zařízení, musí být v „trunk“ módu (příkaz `switchport mode trunk`).
- Musí být zapnuto CDP (Cisco Discovery Protocol), proto se na rozhraních upstream přepínače nesmí vyskytovat příkazy `no cdp enable`, `no cdp tlv app` apod.
- Musí být aktivní DTP (Dynamic Trunk Protocol), tudíž na rozhraních upstream přepínače nesmí být příkaz `switchport nonegotiate`.
- Je-li rozhraní upstream přepínače přiřazeno k agregované lince (tzv. port channel)[6], musíme zkontrolovat výše uvedené body i v logickém rozhraní.

Důležité je upozornit, že veškeré testování probíhalo v sítích, jež převážně využívají zařízení firmy Cisco Systems. V odlišných prostředích se výše uvedené problémy nemusí vyskytovat, mohou však existovat jiné.

### 5.3.3 Úspěšné projekty

Nástroj EasyPnP je aktivně využíván v několika reálných sítích a slouží také jako ukázka programovatelnosti současných zařízení, v tomto případě síťových kontrolérů. Mezi úspěšné projekty řadím:

- K testování nástroje EasyPnP jsem primárně využíval kontroléry společnosti Networksys a.s., kde probíhal i samotný vývoj v rámci pracovní stáže. Zde se nástroj nadále aktivně využívá pro předkonfigurování dodávaných zařízení a nabízí se zákazníkům jako specifický produkt k síťovým kontrolérům. Nástroj jsem rovněž prezentoval na technických seminářích této společnosti.
- Největším realizovaným projektem bylo automatické nasazení síťových zařízení v prostorách Vysoké školy chemicko-technologické v Praze (VŠCHT). Zde jsem ve spolupráci síťových administrátorů výpočetního centra vytvářel pravidla v kontroléru Cisco APIC-EM pomocí nástroje EasyPnP. Výsledkem bylo úspěšné nasazení 209 kompaktních přepínačů, konkrétně Cisco Catalyst C3560-CX.
- Mezi významné úspěchy řadím pozvánku k prezentaci nástroje EasyPnP na světové konferenci Cisco Live 2018 v Orlandu (USA, Florida), kterou pořádala společnost Cisco Systems. Nástroj EasyPnP je rovněž zařazen mezi inovativní aplikace kontroléru Cisco DNA-C na vývojářském webu[7] společnosti.

[6] Spojuje více fyzických linek do jednoho logického rozhraní.

[7] Uvedeno na webu https://developer.cisco.com/ecosystem/dnacenter/partners/DNAC0018/.

## 5.4 Plánovaná rozšíření nástroje

V IT prostředí dochází k neustálému vývoji produktů, nástrojů a aplikací, jejichž cílem je usnadnění složitých síťových procesů. Aby zůstal produkt lákavý pro uživatele, je potřeba jej nepřetržitě vylepšovat. Možných směrů rozšiřování nástroje EasyPnP je několik:

- Úprava kódu pro nejnovější verze síťového kontroléru Cisco DNA-C.
- Grafické uživatelské rozhraní namísto dialogové komunikace v příkazovém řádku.
- Přidání podpory dalších kontrolérů, jež umožňují API volání a funkci automatického nasazení síťových zařízení.

Zkušenosti, návrhy a přání jednotlivých uživatelů napomáhají ke zdokonalování nástroje EasyPnP i k odstraňování jeho nedostatků.

# 6 ZÁVĚR

V teoretické části diplomové práce jsem se zaměřil na podrobné vysvětlení principů softwarově definovaných sítí. Představil jsem architekturu SDN a její hlavní rozdíl oproti architektuře tradičních sítí, což je oddělení řídicí a datové části síťových zařízení. Dále jsem popsal protokol OpenFlow, shrnul výhody, nevýhody i bezpečností otázky SDN technologie, identifikoval aktuální trendy a budoucí vize ve spojitosti s umělou inteligencí. Součástí je přehled komerčních a open-source nástrojů vhodných pro SDN prostředí podnikových sítí. V teoretické části jsou rovněž vysvětleny principy druhé z uvedených technologií, tedy virtualizace síťových funkcí. Jelikož jsem NFV technologii nepoužíval v praktické části, byla zde popsána spíše okrajově.

V praktické části diplomové práce jsem se zabýval analýzou současných problému v řízení i správě složitých datových sítí reálných provozovatelů, konkrétně zákazníků firmy Networksys a.s. Prostřednictvím osobních schůzek, telefonické či e-mailové komunikace, jsem sesbíral řadu aktuálně řešených problémů a nápadu pro zjednodušení i automatizaci. Na základě této analýzy jsem v programovacím jazyce Python vyvinul softwarový nástroj automatizující funkci Plug and Play prostřednictvím REST API rozhraní zvoleného síťového kontroléru. Vzniklý nástroj EasyPnP zajišťuje automatické vytváření konfigurací pro vybraná zařízení na základě definovaných údajů a ve spolupráci se síťovým kontrolérem Cisco APIC-EM či Cisco DNA-C usnadňuje proces hromadného přidávání nových či stávajících zařízení do sítě, šetří velké množství času a eliminuje chyby způsobené manuální instalací každého zařízení zvlášť. Volitelně umožňuje vytváření zařízení v nástroji Cisco ISE a modulární rozšíření pro síťové kontroléry jiných výrobců. Nástroj EasyPnP zpracovává a předává instrukce od uživatele do kontroléru i naopak. To poskytuje kompletní a přehlednou správu celého PnP procesu bez nutnosti používat GUI vybraného kontroléru.

Detailně zpracovaná teoretická část může sloužit jako informační zdroj pro studenty telekomunikačních a síťových oborů, jelikož českojazyčných prací zaměřených na technologie moderních datových sítí prozatím není mnoho. Veškeré zdrojové kódy nástroje EasyPnP jsou volně přístupné ve webovém repozitáři GitHub[1] a obsahují kompletní popis v anglickém jazyce. Ty mohou sloužit jako ukázka programovatelného přístupu k síťovým zařízením. Správcům sítí přináším nástroj, který využívá funkcí síťového kontroléru a vytváří tak mezikrok do světa softwarově definovaných sítí. Vývoj nástroje probíhal ve firmě Networksys a.s., kde se produkt nadále využívá pro předkonfigurování dodávaných zařízení. Jedním z úspěšně realizovaných projektů bylo hromadné nasazení síťových zařízení v prostorách Vysoké školy chemicko-technologické v Praze.

[1] Dostupné na webu https://github.com/mikesm11/EasyPnP.

# LITERATURA


[1] *The Mobile Economy 2018* [online]. GSMA Intelligence, aktualizováno 1. 5. 2018, [cit. 20. 9. 2018]. Dostupné z URL: <https://www.gsma.com/mobileeconomy/wp-content/uploads/2018/05/The-Mobile-Economy-2018.pdf>.

[2] MALEKI, A., HOSSAIN, M., GEORGES, J., RONDEAU, E., DIVOUX, T. *An SDN Perspective to Mitigate the Energy Consumption of Core Networks* [online]. ResearchGate, Září 2017, [cit. 20. 9. 2018]. Dostupné z URL: <https://www.researchgate.net/publication/319876305_An_SDN_Perspective_to_Mitigate_the_Energy_Consumption_of_Core_Networks_-_GEANT2>.

[3] KREUTZ, D., RAMOS, V., M., F., VERÍSSIMO, E., P., ROTHENBERG, E., CH., AZODOLMOLKY, S., UHLIG, S. *Software-Defined Networking: A Comprehensive Survey* [online]. Proceedings of the IEEE, 2015, 103(1), 14-76 s, [cit. 22. 9. 2018]. DOI 10.1109 / JPROC 2014.2371999. Dostupné z URL: <https://ieeexplore.ieee.org/document/6994333>.

[4] GÖRANSSON, P., BLACK, C. *Software Defined Networks: A Comprehensive Approach*. Waltham: Morgan Kaufmann, 2014. 353 s. ISBN 978-0-12-416675-2.

[5] GAVRYLIUK, O. *Směrování SDN podle přenášeného obsahu* [online]. Bakalářská práce. FIT VUT Brno, 2016 [cit. 22. 9. 2018]. Dostupné z URL: <https://www.vutbr.cz/studenti/zav-prace/detail/96595>.

[6] STALLINGS, W. *Data and Computer Communications* [online]. Tenth edition. Pearson Education – Prentice Hall, 2013, [cit. 23. 9. 2018]. Dostupné z URL: <https://slideplayer.com/slide/10685850/>.

[7] BOUŠKA, P. *Základy počítačových sítí* [online]. © 2005 – 2018, aktualizováno 9. 5. 2010 [cit. 26. 9. 2018]. Dostupné z URL: <https://www.samuraj-cz.com/serie/zaklady-pocitacovych-siti/>.

[8] WATERS, N. *Traditional vs Software Defined Networking* [online]. IPknowledge, 2017, [cit. 26. 9. 2018]. Dostupné z URL: <https://docplayer.net/28719982-Traditional-vs-software-defined-networking.html>.

[9] KLEYMAN, B. *Understanding the Rise and Impact of SDN and NFV* [online]. Data Center Knowledge, aktualizováno 18. 12. 2015, [cit. 2. 10. 2018]. Dostupné z URL: <https://www.datacenterknowledge.com/archives/2015/12/18/understanding-the-rise-and-impact-of-sdn-and-nfv>.

[10] POODARI, K. *Flipping the switch: A mixed vs. hybrid approach to SDN deployments* [online]. NXP, aktualizováno 5. 11. 2014, [cit. 27. 10. 2018]. Dostupné z URL: <https://blog.nxp.com/networking/flipping-the-switch-a-mixed-vs-hybrid-approach-to-sdn-deployments>.

[11] 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. 439 s. ISBN 978-1-1386-3152-6.

[12] MINIMAN, S. *Networking Revolution: Software Defined Networking and Network Virtualization* [online]. ©2013, aktualizováno 14. 1. 2013, [cit. 6. 10. 2018]. Dostupné z URL: <http://wikibon.org/wiki/v/Networking_Revolution:_Software_Defined_Networking_and_Network_Virtualization>.

[13] *What are SDN Controllers (or SDN Controllers Platforms)?* [online]. SDxCentral, ©2012 – 2018, [cit. 7. 10. 2018]. Dostupné z URL: <https://www.sdxcentral.com/sdn/definitions/sdn-controllers/>.

[14] BLIAL, O., MAMOUN, B., M., BENAINI, R. *An Overview on SDN Architectures with Multiple Controllers* [online]. Journal of Computer Networks and Communications, 2016, 1-8 s, [cit. 3. 11. 2018]. DOI 10.1155 / JPROC 2016.9396525. Dostupné z URL: <https://www.hindawi.com/journals/jcnc/2016/9396525/>.

[15] MORREALE, P., A., ANDERSON, J., M. *Software Defined Networking: Design and Deployment.* New York: CRC Press, 2015. 186 s. ISBN 978-1-4822-3863-1.

[16] *OpenFlow Switch Specification Version 1.5.1* [online]. Open Networking Foundation, aktualizováno 26. 3. 2015, [cit. 13. 10. 2018]. Dostupné z URL: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>.

[17] STALLINGS, W. *Software-Defined Networks and OpenFlow* [online]. The Internet Protocol Journal, 2013, 16(1), [cit. 14. 10. 2018]. Dostupné z URL: <https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html>.

[18] *What is Definition of Software Defined Networking (SDN)?* [online]. SDxCentral, ©2012 – 2018, [cit. 18. 10. 2018]. Dostupné z URL: <https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>.

[19] LI, T. *What are the pros and cons of software-defined networking?* [online]. Quora, aktualizováno 6. 3. 2015, [cit. 20. 10. 2018]. Dostupné z URL: <https://www.quora.com/What-are-the-pros-and-cons-of-software-defined-networking>.

[20] FARHADY, H., LEE, H., NAKAO, A. *Software-Defined Networking: A survey* [online]. Computer Networks, 2015, 81(1), 79-95 s, [cit. 30. 10. 2018]. DOI 10.1016 / JPROC 2015.0202014. Dostupné z URL: <https://www.sciencedirect.com/science/article/abs/pii/S1389128615000614>.

[21] DABBAGH, M., RAYES, A., HAMDAOUI, B., GUIZANI, M. *Software-Defined Networking Security: Pros and Cons* [online]. IEEE Communications Magazine, 2015, 53(6), [cit. 27. 10. 2018]. DOI 10.1109 / JPROC 2015.7120048. Dostupné z URL: <https://www.researchgate.net/publication/274315299_Software-Defined_Networking_Security_Pros_and_Cons>.

[22] ZURIER, S. *Small SDN providers cost, flexibility appeal to enterprises* [online]. TechTarget, Únor 2018, [cit. 20. 11. 2018]. Dostupné z URL: <https://searchnetworking.techtarget.com/feature/Small-SDN-providers-cost-flexibility-appeal-to-enterprises>.

[23] NAVEEN, J. *The expansion of hyper-automation may be a bad omen for all physical process companies* [online]. Allerin, aktualizováno 2. 6. 2018, [cit. 20. 11. 2018]. Dostupné z URL: <https://www.allerin.com/blog/were-afraid-the-expansion-of-hyper-automation-may-be-a-bad-omen-for-all-physical-process-companies>.

[24] RAY, T. *Demystifying Neural Networks, Deep Learning, Machine Learning, and Artificial Intelligence* [online]. Stoodnt Inc., aktualizováno 29. 3. 2018, [cit. 30. 11. 2018]. Dostupné z URL: <https://www.stoodnt.com/blog/ann-neural-networks-deep-learning-machine-learning-artificial-intelligence-differences/>.

[25] DHANDE, M. *What is Artificial Intelligence, Machine Learning and Deep Learning?* [online]. Geospatial World, aktualizováno 4. 6. 2017, [cit. 30. 11. 2018]. Dostupné z URL: <https://www.geospatialworld.net/blogs/artificial-intelligence-machine-learning-and-deep-learning/>.

[26] BILAL, A. *Artificial Neural Networks and Deep Learning* [online]. Medium, aktualizováno 30. 1. 2018, [cit. 30. 11. 2018]. Dostupné z URL: <https://becominghuman.ai/artificial-neural-networks-and-deep-learning-a3c9136f2137>.

[27] TAHSILDAR, S. *What is the difference between Neural Networks and Deep Learning?* [online]. Quora, aktualizováno 23. 7. 2018, [cit. 30. 11. 2018]. Dostupné z URL: <https://www.quora.com/What-is-the-difference-between-Neural-Networks-and-Deep-Learning>.

[28] HERRERA, G., J., BOTERO, F., J. *Resource Allocation in NFV: A Comprehensive Survey* [online]. IEEE Transactions on Network and Service Management, 2016, 13(3), 518-532 s, [cit. 1. 12. 2018]. DOI 10.1109 / JPROC 2016.2598420. Dostupné z URL: <https://ieeexplore.ieee.org/document/7534741>.

[29] *What is ETSI ISG NFV?* [online]. SDxCentral, © 2012 – 2019, [cit. 2. 12. 2018]. Dostupné z URL: <https://www.sdxcentral.com/networking/nfv/definitions/etsi-isg-nfv/>.

[30] STALLINGS, W. *Foundations of Modern Networking: SDN, NFV, QoE, IoT and Cloud*. New Jersey: Pearson Education, Addison-Wesley Professional, 2015. 560 s. ISBN 978-0-1341-7539-3.

[31] CHATRAS, B., OZOG, F., F. *Network functions virtualization: the portability challenge* [online]. IEEE Network, 2016, 30(4), 4-8 s, [cit. 1. 12. 2018]. DOI 10.1109 / MNET 2016.7513857. Dostupné z URL: <https://ieeexplore.ieee.org/document/7513857>.

[32] *Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges And Call for Action* [online]. White Paper. European Telecommunications Standards Institute, vydáno 22. 10. 2012, [cit. 15. 12. 2018]. Dostupné z URL: <https://portal.etsi.org/nfv/nfv_white_paper.pdf>.

[33] LI, Y., CHEN, M. *Software-Defined Network Function Virtualization: A Survey* [online]. IEEE Access, 2015, 3(1), 2542-2553 s, [cit. 15. 12. 2018]. DOI 10.1109 / JPROC 2015.2499271. Dostupné z URL: <https://ieeexplore.ieee.org/document/7350211>.

[34] PODHRADSKÝ, P., HELEBRANDT, P, HALAGAN, T, DROZD, I. *Sítě budoucnosti – SDN a NFV*. Praha: TechPedia, České vysoké učení technické v Praze, Fakulta elektrotechnická, 1. vydání, 2017. 38 s. ISBN 978-80-01-06246-3.

[35] PUJOLLE, G. *Software Networks: Virtualization, SDN, 5G and Security*. London: ISTE Ltd, 2015. 260 s. ISBN 978-1-848-21694-5.

[36] *Network Functions Virtualisation (NFV): Architectural Framework* [online]. European Telecommunications Standards Institute, vydáno 01. 12. 2014, [cit. 20. 12. 2018]. Dostupné z URL: <https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf>.

[37] *Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress* [online]. White Paper. European Telecommunications Standards Institute, vydáno 15. 10. 2013, [cit. 20. 12. 2018]. Dostupné z URL: <https://portal.etsi.org/nfv/nfv_white_paper2.pdf>.

[38] SHAW, K. *What is a hypervisor?* [online]. Network World, aktualizováno 19. 12. 2017, [cit. 22. 12. 2018]. Dostupné z URL: <https://www.networkworld.com/article/3243262/what-is-a-hypervisor.html>.

[39] LINGUAGLOSSA, L., LANGE, S., RÉTVÁRI, G., ROSSI, D., ZINNER, T. *Survey of Performance Acceleration Techniques for Network Function Virtualization* [online]. Proceedings of the IEEE, 2019, 107(4), 746-764 s, [cit. 27. 4. 2019]. DOI 10.1109 / JPROC 2019.2896848. Dostupné z URL: <https://ieeexplore.ieee.org/document/8666751>.

[40] ALJUHANI, A., ALHARBI, T. *Virtualized Network Functions security attacks and vulnerabilities* [online]. IEEE CCWC, 2017, [cit. 27. 4. 2019]. DOI 10.1109 / CCWC 2017.7868478. Dostupné z URL: <https://ieeexplore.ieee.org/document/7868478>.

[41] FANG, V., LÉVAI, T., HAN, S., RATNASAMY, S., RAGHAVAN, B., SHERRY, J. *Evaluating Software Switches: Hard or Hopeless?* [online]. University of California at Berkeley, Electrical Engineering and Computer Sciences, aktualizováno 12. 10. 2018, [cit. 9. 2. 2019]. Dostupné z URL: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-136.pdf>.

[42] *Special Report: OpenFlow and SDN – State of the Union* [online]. SDxCentral, vydáno 2016, [cit. 10. 2. 2019]. Dostupné z URL: <https://www.opennetworking.org/wp-content/uploads/2013/05/Special-Report-OpenFlow-and-SDN-State-of-the-Union-B.pdf>.

[43] *Enterprise Campus Architecture Comparative Assessment* [online]. Miercom, vydáno 10. 10. 2018, [cit. 1. 5. 2019]. Dostupné z URL: <http://miercom.com/cisco-dna-competitive-pv/>.

[44] *Cisco Network Plug and Play Solution Data Sheet* [online]. Cisco Systems, aktualizováno 21. 3. 2016, [cit. 4. 5. 2019]. Dostupné z URL: <https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/application-policy-infrastructure-controller-enterprise-module/datasheet-c78-735964.html>.

[45] *APIC Enterprise Module API Overview* [online]. Cisco Systems, [cit. 4. 5. 2019]. Dostupné z URL: <https://developer.cisco.com/docs/apic-em/>.

[46] *Cisco Application Policy Infrastructure Controller Enterprise Module – Release 1.5 Data Sheet* [online]. Cisco Systems, aktualizováno 29. 6. 2017, [cit. 4. 5. 2019]. Dostupné z URL: <https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/application-policy-infrastructure-controller-enterprise-module/datasheet-c78-730594.html>.

[47] HILL, C., MILLER, D., ZACKS, D., SUHR, J. *Cisco Software-Defined Access: Enabling Intent-Based Networking* [online]. San Jose: Cisco Systems, 2nd edition, vydáno 2019, [cit. 8. 5. 2019]. Dostupné z URL: <https://www.cisco.com/c/dam/en/us/products/se/2018/1/Collateral/nb-06-software-defined-access-ebook-en.pdf>.

[48] NETWORKSYS. *DNA softwarově definované sítě* [online]. Netguru Network News, aktualizováno 10. 12. 2018, [cit. 8. 5. 2019]. Dostupné z URL: <https://www.netguru-nn.com/dna-softwarove-definovane-site/>.

[49] *Cisco DNA Center 1.2 Data Sheet* [online]. Cisco Systems, aktualizováno 27. 3. 2019, [cit. 9. 5. 2019]. Dostupné z URL: <https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/dna-center/nb-09-dna-center-data-sheet-cte-en.html>.

[50] *Cisco DNA Center Platform Overview* [online]. Cisco Systems, [cit. 9. 5. 2019]. Dostupné z URL: <https://developer.cisco.com/docs/dna-center/#!cisco-dna-center-platform-overview/>.

[51] SALMAN, O., ELHAJJ, H., I., KAYSSI, A., CHEHAB, A. *SDN controllers: A comparative study* [online]. 18th Mediterranean Electrotechnical Conference, aktualizováno 23. 6. 2016, [cit. 20. 5. 2019]. DOI 10.1109 / MELCON 2016.7495430. Dostupné z URL: <https://ieeexplore.ieee.org/abstract/document/7495430>.

[52] PALIWAL, M., SHRIMANKAR, D., TEMBHURNE, O. *Controllers in SDN: A Review Report* [online]. ResearchGate, Červen 2018, [cit. 20. 5. 2019]. Dostupné z URL: <https://www.researchgate.net/publication/325706050_Controllers_in_SDN_A_Review_Report>.

[53] COX, H., J., CHUNG, J., DONOVAN, S., IVEY, J., RUSSELL, J., C., RILEY, G., OWEN, L., H. *Advancing Software-Defined Networks: A Survey* [online]. IEEE Access, 2017, 3, 25487-25526 s, [cit. 20. 5. 2019]. DOI 10.1109 / ACCESS 2017.2762291. Dostupné z URL: <https://ieeexplore.ieee.org/abstract/document/8066287>.

[54] SAKELLAROPOULOU, D. *A Qualitative Study of SDN Controllers* [online]. Master's thesis. Athens University of Economics and Business, 2017 [cit. 20. 5. 2019]. Dostupné z URL: <https://mm.aueb.gr/master_theses/xylomenos/Sakellaropoulou_2017.pdf>.

[55] ŠVENDA, J. *Vyplatí se jednobarevnost nebo multi-vendor řešení?* [online]. Channel World, aktualizováno 4. 11. 2013, [cit. 2. 11. 2018]. Dostupné z URL: <https://channelworld.cz/redakcni-komentare/vyplati-se-jednobarevnost-nebo-multi-vendor-reseni-10088>.

[56] MATTKE, T. *Working with the Embedded Event Manager (EEM)* [online]. Router Jockey, © 2008 – 2018, [cit. 10. 11. 2018]. Dostupné z URL: <https://routerjockey.com/2010/06/14/working-with-the-embedded-event-manager-eem/>.

[57] LIEW, Z. *Understanding And Using REST APIs* [online]. Smashing Magazine, aktualizováno 17. 1. 2018, [cit. 30. 3. 2019]. Dostupné z URL: <https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/>.

# SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

| | |
|---|---|
| AAA | Authentication, Authorization and Accounting |
| ACL | Access Control List |
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| APIC-EM | Application Policy Infrastructure Controller – Enterprise Module |
| ASCII | American Standard Code for Information Interchange |
| ASIC | Application Specific Integrated Circuit |
| BGP | Border Gateway Protocol |
| BSS | Business Support System |
| CAM | Content Addressable Memory |
| CapEx | Capital Expenditures |
| CDN | Content Delivery Network |
| CDP | Cisco Discovery Protocol |
| CLI | Command Line Interface |
| CoA | Change of Authorization |
| COTS | Commercial off-the-shelf |
| CRUD | Create, Read, Update, Delete |
| CSV | Comma Separated Values |
| CUCM | Cisco Unified Communications Manager |
| DDoS | Distributed Denial of Service |
| DHCP | Dynamic Host Configuration Protocol |
| DL | Deep Learning |
| DNA-C | Digital Network Architecture – Center |
| DNS | Domain Name System |
| DoS | Denial of Service |
| DPI | Deep Packet Inspection |

| | |
|---|---|
| DTP | Dynamic Trunk Protocol |
| EEM | Embedded Event Manager |
| EMS | Element Management System |
| eNB | evolved Node B |
| ETSI | European Telecommunications Standards Institute |
| FIS | Flow Instruction Set |
| ForCES | Forwarding and Control Element Separation |
| GGSN | Gateway GPRS Support Node |
| GSMA | Groupe Spécial Mobile Association |
| GUI | Graphical User Interface |
| HA | High Availability |
| HAL | Hardware Abstraction Layer |
| HLR | Home Location Register |
| HP | Hewlett-Packard |
| HSS | Home Subscriber Server |
| HTTP | HyperText Transfer Protocol |
| I2RS | Interface to the Routing System |
| IBM | International Business Machines |
| IBN | Intent-Based Networking |
| ID | Identification |
| IDS | Intrusion Detection System |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IMS | IP Multimedia Subsystem |
| IOS | Internetworking Operating System |
| IoT | Internet of Things |
| IP | Internet Protocol |

| | |
|---|---|
| IPSec | IP Security |
| IPv4 | Internet Protocol version 4 |
| IPv6 | Internet Protocol version 6 |
| ISE | Identity Services Engine |
| ISG | Industry Specification Group |
| IS-IS | Intermediate System to Intermediate System |
| ISO | International Organization for Standardization |
| ISP | Internet Service Provider |
| IT | Information Technology |
| JSON | JavaScript Object Notation |
| LAN | Local Area Network |
| MAC | Media Access Control |
| MANO | Management and Orchestration |
| MITM | Man-in-the-middle |
| ML | Machine Learning |
| MPLS | Multiprotocol Label Switching |
| NAT | Network Address Translation |
| NB | Node B |
| NETCONF | Network Configuration Protocol |
| NFV | Network Functions Virtualization |
| NFVI | Network Functions Virtualization Infrastructure |
| NGN | Next Generation Networks |
| NN | Neural Networks |
| NOS | Network Operating System |
| NS | Network Service |
| NVRAM | Non-Volatile Random Access Memory |
| ODL | OpenDaylight |

| | |
|---|---|
| ONF | Open Network Foundation |
| ONOS | Open-Source Network Operating System |
| OOP | Object Oriented Programming |
| OpEx | Operational Expenditures |
| OSI | Open System Interconnection |
| OSPF | Open Shortest Path First |
| OSS | Operations Support System |
| OVSDB | Open vSwitch Database |
| PAD | Programmable Abstraction of Data Path |
| PCEP | Path Computation Element Protocol |
| PDN | Packet Data Network |
| PDU | Protocol Data Unit |
| PI | Prime Infrastructure |
| PnP | Plug and Play |
| PoC | Proof of Concept |
| POF | Protocol Oblivious Forwarding |
| PPP | Point-to-Point Protocol |
| QoE | Quality of Experience |
| QoS | Quality of Services |
| RADIUS | Remote Authentication Dial In User Service |
| REST | Representational State Transfer |
| RIB | Routing Information Base |
| RNC | Radio Network Controller |
| ROFL | Revised OpenFlow Library |
| SaaS | Software-as-a-Service |
| SBC | Session Border Controller |
| SDA | Software Defined Access |

| | |
|---|---|
| SDN | Software Defined Networking |
| SLA | Service Level Agreement |
| SNMP | Simple Network Management Protocol |
| SSH | Secure Shell |
| SSL | Secure Sockets Layer |
| STP | Spanning Tree Protocol |
| TACACS | Terminal Access Controller Access Control System |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| TTL | Time To Live |
| UDP | User Datagram Protocol |
| UML | Unified Modeling Language |
| URL | Uniform Resource Locator |
| VEB | Virtual Ethernet Bridge |
| VIM | Virtualized Infrastructure Manager |
| VLAN | Virtual Local Area Network |
| VM | Virtual Machine |
| VNF | Virtual Network Function |
| VPN | Virtual Private Network |
| VXLAN | Virtual eXtensible LAN |
| WAN | Wide Area Network |
| WLAN | Wireless Local Area Network |
| XML | eXtensible Markup Language |

# SEZNAM PŘÍLOH

# A OBSAH PŘILOŽENÉHO CD

Přiložený disk obsahuje následující soubory a adresáře:

- **Mikéska-Martin-DP.pdf**
  Elektronická verze diplomové práce ve formátu PDF.
- **EasyPnP.zip**
  Adresář se zdrojovým kódem nástroje EasyPnP.
- **Videonávod-EasyPnP-cz.mp4**
  Videonávod v českém jazyce k nástroji EasyPnP ve formátu MP4.
- **Datasheet-EasyPnP-cz.pdf**
  Marketingový materiál nástroje EasyPnP v českém jazyce ve formátu PDF.
- **Datasheet-EasyPnP-en.pdf**
  Marketingový materiál nástroje EasyPnP v anglickém jazyce ve formátu PDF.