VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DYNAMICKÝ STAV MODELU OMNET++ POMOCÍ SNMP

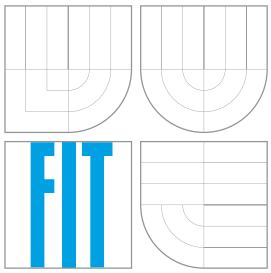
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB SMEJKAL

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DYNAMICKÝ STAV MODELU OMNET++ POMOCÍ SNMP

DYNAMIC STATE OF OMNET++ MODEL VIA SNMP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JAKUB SMEJKAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. VLADIMÍR VESELÝ
SUPERVISOR

BRNO 2012

Abstrakt

Práce se zabývá přípravou a získáváním dynamických dat o sledované síti pro simulační nástroj OMNeT a jeho přídavek INET Framework, který je rozšiřován v rámci projektu ANSA na Fakultě Informatiky VUT. Jako prostředek k dosažení požadované funkcionality byla zvolena prioritně technologie SNMP. Na detekci topologie spolupracují protokoly CDP, LLDP a v teoretické rovině upravený algoritmus prohledávání do šířky. Výstupem je soubor popisující počítačovou sít syntakticky vymezený jazykem NED.

Abstract

This work outlines ways of getting dynamic state into OMNeT++ model. SNMP as major technology was selected for reaching the goals. Protocols CDP and LLDP are participating in topology detection and in theoretical layer Breadth-first search is used. The output of this tool is file syntactically specified by NED language which is describing computer network.

Klíčová slova

Omnet++, počítačové sítě, správa sítě, simulace, SNMP, MIB, SMI, BER, VUT FIT, ANSA, OMNeT++, INET, BFS.

Keywords

Omnet++, computer networks, network management, simulation, SNMP, MIB, SMI, BER,VUT FIT, ANSA, OMNeT++, INET, BFS.

Citace

Jakub Smejkal: Dynamický stav modelu OMNeT++ pomocí SNMP, diplomová práce, Brno, FIT VUT v Brně, 2012

Dynamický stav modelu OMNeT++ pomocí SNMP

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Veselého

.....
Jakub Smejkal
20. května 2012

Poděkování

Zde bych rád poděkoval svému vedoucímu Ing. Vladimíru Veselému za odbornou pomoc a přiblížení problematiky simulace počítačových sítí. Dále uvádím recept vhodný pro sportovce. Jedná se o palačinku, kterou je možné dělat jak ve slané, tak sladké formě. Osobně doporučuji spíše sladkou variantu, na kterou popíši recept níže. Pro recept budeme potřebovat 8 vajec, 120g ovesných vloček, kakao, 70g proteinového prášku ideálně s čokoládovou nebo vanilkovou příchutí, volitelně nějaká semínka, případně ořechy na zpestření. Všechny tyto ingredience dáme do mixéru, který je schopen je dostatečně rozsekat. Podíl kakaa volíme podle toho jak chceme mít palačinku sladkou, většinou ja potřeba přidat tak pět kávových lžiček kvůli výraznému podílu ovesných vloček, který slouží na zahuštění směsi. Alternativně můžeme použít neslazené kakao a doplnit sladidlem stevia. Po dostatečném promíchání a rozdrcení směsi v mixéru, kde by měla vzniknout hutná substance, můžeme vhodné množství vložit na pánev a opékat podle tloušťky vrstvy. Pokud šířka dosahuje jednoho centimetru je třeba péct pomaleji po dobu i více jak pěti minut. Palačinka se dá konzumovat samostatně, případně se dá vymyslet další příloha podle fantazie. Nutriční hodnoty jsou při průměrné velikosti vajec(cca 50g) a 75% proteinu cca 150g bílkovin, 100g sacharidů, 68g tuku.

© Jakub Smejkal, 2012.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

Složitost sítí a technologií obecně je v současné době stále větší. Proto také vzrůstají nároky na jejich údržbu a problematiku řešení problémů obecně. V této souvislosti už probíhá mnoho snah na různých úrovních přístupu. Ať se bavíme pouze o zdokonalování aktuálních softwarových nástrojů, nebo o tvorbě kompletně nových přístupů k návrhu a testování systémů. V této oblasti má určitě svoje místo disciplína jménem simulace. Do této kategorie spadá také nástroj OMNeT++, kolem kterého se soustřeďují aktivity z několika bakalářských a diplomových prací na VUT FIT. Simulace jako taková je velmi efektivní nástroj pro pozorování systému, o kterém nemáme kompletní informace. Výše zmíněný simulační nástroj má již velké množství funkcionality integrováno, ať z předešlých kvalifikačních prací vzniklých na naší fakultě, nebo přímo od vývojářů a přidružených projektů. Popisovaná práce se tedy snaží opět o krok přiblížit tento nástroj k námi požadovaným cílům, pro správu počítačové sítě VUT.

1.1 Struktura práce

Tato diplomová práce v úvodu popisuje důvody pro tvorbu a zdokonalování níže uváděného nástroje Omnet++ a s tím spojené nasazování simulace obecně jako prvku ověřování funkčnosti a zdokonalování počítačové sítě. Po tomto motivačním úvodu následují teoretické kapitoly, která popisují protokoly LLDP, CDP a SNMP, který je v diplomové práci majoritně používanou technologií k dosažení požadovaných cílů. Dále pak uvedeme simulační technologii OMNeT a jeho další rozšíření. Jakmile dojde k popisu těchto participujících technologií, je možné přistoupit k části pojednávající o teoretické stránce prozkoumávání topologie a s tím spojenou volbou a úpravou prohledávacího algoritmu. Následně se dostaneme k implementaci nástroje jako takového a použitým externím nástrojům. Na tuto kapitolu naváže testování v rámci sítě VUT. V závěru popíšeme stručně tvorbu diplomové práce a možný návazný vývoj.

Kapitola 2

Protokol SNMP

2.1 Historie protokolu SNMP

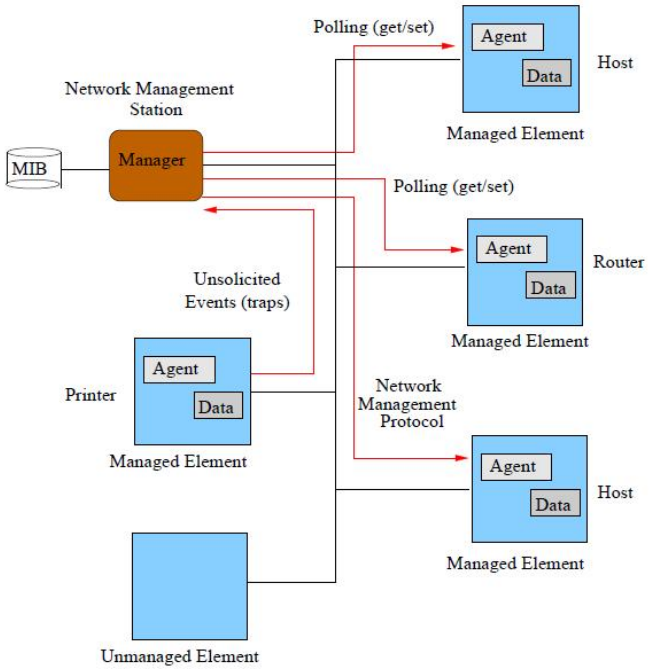
Vzhledem ke stále se zvětšující komplexnosti správy počítačových sítí vznikla potřeba určitého protokolu, který by umožňoval efektivní vzdálenou správu jednotlivých prvků. V tomto směru byl předchůdcem samotného SNMP protokol SGMP navržen roku 1987, který se ale omezoval pouze na management směrovače. Umožňoval získávat nebo nastavovat několik základních prvků těchto zařízení a využíval stejně jako SNMP protokol UDP. Dalším pokusem na tomto poli byl objektově orientovaný protokol CMIP, který se ale nedočkal příliš velkého rozšíření a časem zanikl. V tomto směru se ale SNMP dařilo o poznání lépe, především díky relativně jednoduché implementaci, vysoké flexibilitě protokolu a tedy široké škále využití[20]. SNMP (Simple Network Management Protocol) je tedy asynchronní, transakčně orientovaný protokol založený na modelu klient/server. Jeho první verze s názvem SNMPv1 vznikla v roce 1989, kdy byla také napsána první specifikace RFC 1157. Následně roku 1990 byl přijat institucí IAB (Internet Acitivities Board) jako standard sítě internet. Tato první verze představovala absolutní základ a byla často kritizována za nedostatečnou zabezpečenost, poskytovala pouze autentizaci klientů na základě "community stringu", které byly zasílány v otevřené podobě.

Z tohoto popudu došlo ke vzniku SNMPv2. Tato verze ovšem nebyla příliš masově nasazována, především kvůli neshodám v otázkách bezpečnosti a přehnané složitosti protokolu. Celkem vzniklo několik variant SNMPv2. První označovaná jako SNMPv2p (RFC 1441 - RFC 1452) poskytovala zvýšenou úroveň bezpečnosti, plus schopnost komunikace Manager-Manager a také získávání většího množství informací najednou pomocí "BULK RETRIEVAL"operace. Právě tato verze byla, ale diskutována jako obtížná na implementaci, a proto se příliš nerozšířila. Další nesla název SNMPv2c (RFC 1901 - RFC 1908) a patřila k nejnasazovanějším z celé druhé generace. Poskytovala vylepšení SNMPv2p ovšem bez komplikovaného autentizačního mechanismu, tedy veškerá bezpečnost byla opět na základě "community stringů". Poslední varianta byla pojmenována jako SNMPv2u (RFC 1909 - RFC 1910), která představovala kompromis mezi dvěma předcházejícími, poskytovala větší zabezpečení, které ale nebylo tak obtížně implementovatelné. Vylepšený bezpečnostní mechanismus SNMPv2u byl nadále přijat jako základ k vzniku zabezpečení SNMPv3.

Současně nejaktuálnější verze nese název SNMPv3 (RFC 3411 - RFC 3418). V roce 2004 byla přijata jako standard protokolu SNMP. SNMPv3 poskytuje vylepšené konfigurační možnosti a především řeší hodně bezpečnostních problémů předchozích verzí. Nová verze umožňuje již důvěrnost a integritu zpráv především díky zavedení šifrování pomocí DES/AES. Standardy SNMPv1 a SNMPv2 byly následně označeny jako zastaralé.

2.2 Obecné principy SNMP

Protokol SNMP tedy slouží ke vzdálené správě síťových prvků. Je důležité zmínit, že pod pojmem SNMP si nelze představovat pouze komunikační protokol, jelikož k fungování celého systému správy sítě je třeba definovat i další standardy, které představují databázovou část celého modelu. Princip fungování SNMP se nejlépe pochopí ze zobrazené struktury možného nasazení celého systému. V tomto schématu vystupuje několik elementů. Jsou to Manažer a Agent. Tyto dvě entity představují základ filozofie celého systému, proto je níže popíši přesněji.



Obrázek 2.1: Struktura SNMP

Element Manažera v systému SNMP představuje druhou stranu komunikace s Agentem. Slouží jako bod výstupních informací o sledované síti, které prezentuje správci. Jeho funkce je realizována obvykle softwarově, ta provádí interpretaci SNMP odpovědí od Agentů a také formulování dotazů. Mimo to je schopen také přijímat předem nevyžádané zprávy, které slouží k indikaci urgentních stavů jednotlivých zařízení. Na stejné stanici jako běží Manažer se obvykle nachází NMS software, který realizuje vizuální prezentaci získaných dat a také tvorbu statistik.

Agenta reprezentuje software, který běží na systému, jež si přejeme sledovat. Jeho hlavní funkcí je správná interpretace dotazů od Manažera, představuje tedy jakéhosi prostředníka mezi spravovaným zařízením a SNMP entitou. Agent sleduje cílové zařízení a ukládá pro nás zajímavá data do MIB. Pokud se vyskytne krizová situace, o které by měl vědět Manažer neprodleně, je Agent schopen vyslat zprávu sám bez předchozího dotazu od Manažera. Tento software může běžet na široké škále zařízení, tedy nejen směrovače, přepínače, ale také servery, tiskárny, stanice, případně jemně specializovaný hardware.

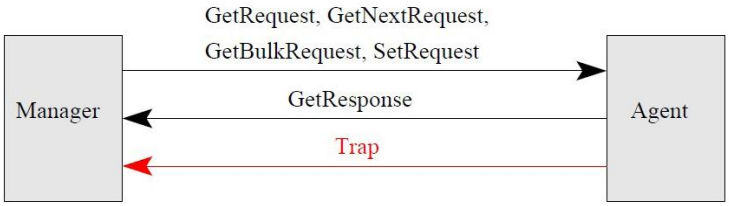
Specializovaným případem Agenta je jeho proxy verze. Jeho využití tkví ve dvou směrech a to je komunikace se zařízeními, které nepodporují SNMP protokol, kde tedy plní vlastní MIB daty od tohoto prvku, případně proxy Agenti mohou redukovat potřebný traffic nutný

k získání požadovaných dat o vzdálené síti. Provádějí to způsobem, že se sami dotazují na prvky v této síti a opět kumulují data ve vlastní MIB, kde jsou pak na požádání dostupné našemu Manažerovi, což nám ušetří velké množství dotazů do sledované sítě.

Základem SNMP je tedy komunikace mezi Agentem a Manažerem, tu můžeme rozdělit do dvou následujících kategorií.

- **Polling** - tento typ představuje klasický formát dotaz a na ní následující odpověď.
- **Trap** - v tomto případě figuruje pouze jedna zpráva odeslaná Agentem, která je následně přijata Manažerem. Zde také platí, že komunikace je nepotvrzovaná.

SNMP pro přenos svých zpráv využívá transportní protokol UDP, kde pro první kategorii komunikace slouží port 161 a pro přijímání trapů port 162. Pro ilustraci jsme použili následující schéma[15].



Obrázek 2.2: Komunikace SNMP

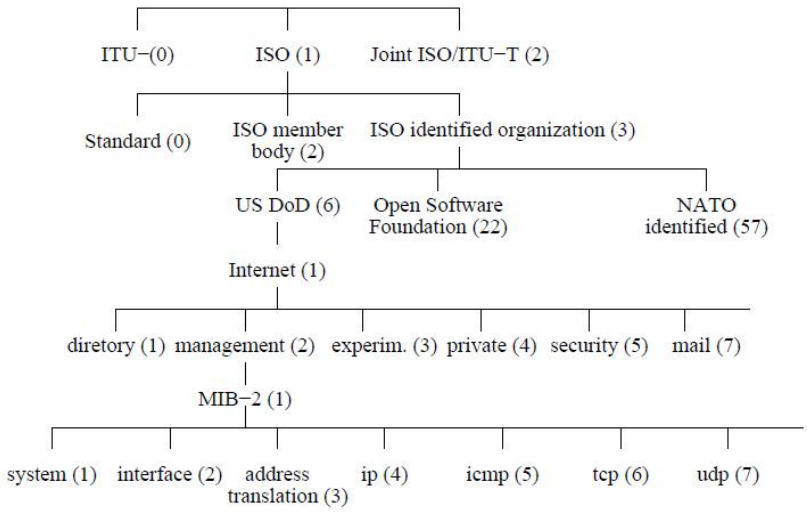
Nyní je již známo základní schéma komunikace, proto dále popíši databázovou část celého systému. S tím souvisí pojem Management information base ve zkratce MIB a Structure and Identification of Management Information tedy SMI.

Entita MIB představuje jakési virtuální uložiště, nebo databázi jednotlivých zařízení. Slouží především k popisu prvků, objektů a zařízení, které máme zájem sledovat. Samotné zařízení nemusí být omezeno na implementaci jedné MIB, ale může jich zavést i více. Jednotlivé prvky této databáze jsou popsány pomocí podmnožiny standardu ASN.1, který nazýváme SMI. Ten zaručuje správný formát prvků databáze a tedy jejich správné zpracování Manažerem nebo Agentem. Objekty v MIB můžeme pak rozdělit do dvou skupin.

- **Skalární** - pomocí tohoto prvku jsou popisovány jednoduché parametry zařízení (například rychlost rozhraní).
- **Složené** - zde je reprezentována složitější struktura, která je z principu určitým způsobem provázána (například směrovací tabulka).

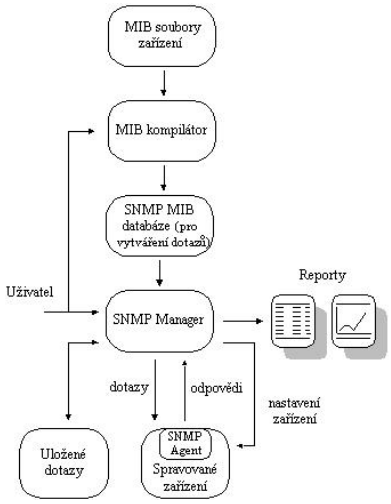
Abychom jednotlivé prvky byli schopni najít, existuje systém jejich značení, který se nazývá Object Identifier ve zkratce OID. Tento identifikátor je tvořen sadou přirozených čísel, která jsou oddělena tečkami, které určují jednotlivé hierarchické úrovně významu. Ve výsledku je pak pomocí těchto OID vytvořen strom. Pomocí něj jsme schopni dosáhnout náš hledaný objekt postupným načítáním částí OID. MIB lze i uživatelsky modifikovat a přidávat tak větve pro vlastní zařízení. Dále existuje i větev, kam jsou umístěny podstromy, které nebyly obecně přijaty, nazývá se Experimental. Pro lepší ilustraci přikládám příklad zanoření do stromu.

Jak již bylo zmíněno SMI je podmnožinou standardu ASN.1, kde hlavním důvodem existence způsobu tohoto popisu, je nutnost mít prostředek k definování prvků systému tak, aby byly čitelné pro člověka, ale zároveň aby byly zpracovatelné počítačem. Pomocí tohoto



Obrázek 2.3: Strom MIB

standardu tedy jsou definovány základní datové struktury a typy, které používá protokol SNMP a i výše zmíněná MIB. U jednotlivých elementů je pak nutné uvést jejich jméno, syntaxi a kódování. Toto jméno objektu slouží jako jednoznačný identifikátor, syntaxí se pak určí jeho datový typ a definice kódování je nutná pro správnou serializaci objektů na přenosový kanál. Jak již bylo zmíněno výše, MIB lze modifikovat a tím rozšířit naše správcovské kapacity. Před finálním použitím modifikované MIB zapsané pomocí SMI, je ji třeba přeložit pomocí MIB překladače. Tento proces je nastíněn následujícím obrázkem.



Obrázek 2.4: Překlad MIB

2.3 Verze SNMP

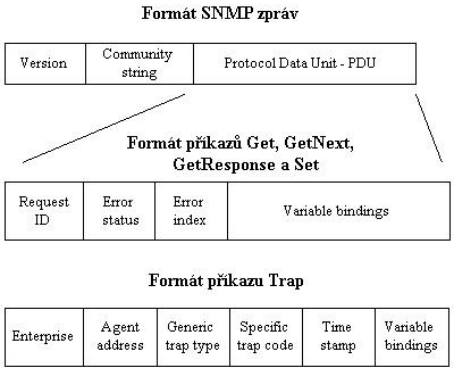
Po uvedení základních principů SNMP přistoupím blíže k popisu jednotlivých verzí tohoto protokolu. Popíši postupně SNMPv1, SNMPv2 a SNMPv3 včetně jejich specifik.

2.3.1 SNMPv1:

První verze tohoto protokolu implementovala základní sadu zpráv pro komunikaci mezi Manažerem a Agentem. Níže je jejich podrobnější popis.[21]

- **Get Request** - tento požadavek je zasílán Manažerem směrem k Agentovi s cílem číst z jeho MIB databáze. Ve svém požadavku specifikuje cílovou informaci podle již zmíněného OID.
- **Get Next Request** - díky hierarchické struktuře MIB, je možné se dotazovat na další element databáze pomocí zjednodušeného dotazu. Logika spočívá v zapamatování si posledního požadavku na čtení Agentem, který následně interpretuje Get Next Request jako požadavek na hierarchicky nižší element MIBu oproti posledně požadovanému.
- **Set Request** - pomocí Set Request Manažer nastavuje hodnoty v Agentově MIB databázi.
- **Get Response** - tato zpráva je odesílána Agentem směrem k Manažerovi a slouží jako odpověď na jeden z Get Requestů.
- **Trap** - trapy jsou odesílány jak Agentem tak příležitostně i Manažerem. V případě Agenta slouží k signalizaci urgentních stavů monitorovaného zařízení. Zatímco u Manažera v realizaci SNMPv1 slouží pro komunikaci mezi jednotlivými Manažery.

Jakmile jsou známy zprávy použité v komunikaci, mohu nastínit jejich formát. Ten lze nejlépe ilustrovat opět pomocí schématu.



Obrázek 2.5: Formát zpráv

Zpráva se dělí do dvou hlavních celků a to je hlavička a PDU, kde v hlavičce můžeme najít verzi SNMP a dále Community string. Ten představuje kompletní zabezpečení v rámci SNMPv1. Tento řetězec reprezentuje heslo omezující přístup k danému zařízení. Používají

se dva typy hesla a to pro zápis a pro čtení. Ta jsou pak přenášena v otevřené podobě, proto je tuto ochranu snadné překonat a je to jeden z nejkritizovanějších aspektů SNMPv1.[16]

Dále ve zprávě najdeme samotné PDU, které se liší podle toho zda pracujeme s formátem zprávy trap nebo nějakým ze zbývajících. Jednu položku mají ale stejnou a tím jsou Variable bindigs. Tato položka představuje množinu dvojic, kde první z dvojice obsahuje OID objektu v MIB a druhá pak jeho hodnotu. Při dotazování se Agenta Manažerem je ve fázi zasílání požadavku na data nejprve vyplněna pouze první položka z dvojice a druhá je prázdná, a pak následně při odpovědi jsou již doplněny obě.

- **Tvar zprávy pro Trap:**
  - **Enterprise** - zde se identifikuje typ objektu, od kterého zpráva pochází.
  - **Agent address** - značí adresu objektu, který vygeneroval zprávu.
  - **Generic trap type, Specific trap code** - specifikuje typ události, který je reportován. (linkDown,linkUp,...)
  - **Time stamp** - zobrazuje čas vygenerování trapu.
- **Tvar zprávy pro ostatní:**
  - **Request ID** - identifikuje který požadavek náleží které reakci.
  - **Error status** - indikuje chybu a typ, nastavuje se při odpovědi na dotaz.
  - **Error index** - spojuje chybu s proměnou z pole variable bindings.

V rámci SNMPv1 pak byly definovány následující datové typy:

- **Základní:**
  - **Integer** - v 32 bitové podobě
  - **Octet String**
  - **Null**
  - **OID** - referencuje objekty v MIB
- **Komplexní:**
  - **Sequence**
  - **Get Request PDU**
  - **Get Next Request PDU**
  - **Get Response PDU**
  - **Set Request PDU**
  - **Trap PDU**
- **Aplikační:**
  - **Network Address** - obecná síťová adresa.
  - **IpAddress** - v specifikaci SNMPv1 pouze IPv4.
  - **Counter** - nezáporné číslo, je možné ho pouze inkrementovat, po přetečení nuluje.

– **Gauge** - nezáporné celé číslo, které je možné i dekrementovat, je omezeno maximální a minimální hodnotou.
– **Time Ticks** - počet hodinových tiků s přesností na setiny vteřiny.
– **Opaque** - slouží pro přenos libovolných dat v ASN.1, před přenosem je zakódován do octet stringu.
– **Integers** - celočíselný typ.
– **Unsigned Integer** - celočíselný typ bez znaménka.

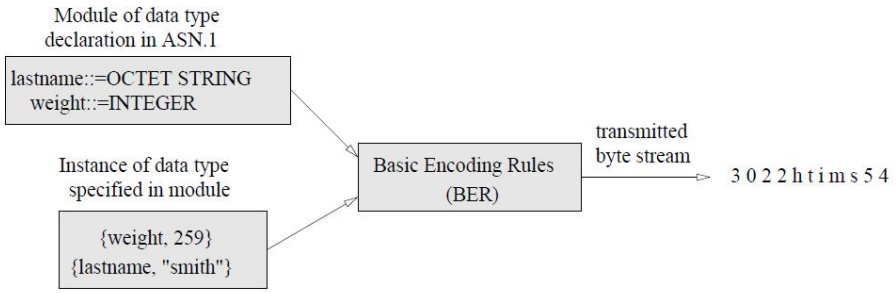
Pokud již známe typy zpráv, jejich tvar a datové typy, můžeme sestavit a odeslat SNMPv1 zprávu, kterou je ale potřeba zakódovat pro přenos. Tato akce se provádí pomoci kódování BER. Celý princip kódování je definován následující trojicí.

Obrázek 2.6: Formát zpráv

Skládá se z typu, délky a finálních dat. V některých případech, které vyvstanou i u SNMP obsahují některé položky, jež je nutné zakódovat, jiné další položky odlišného typu. Ty je ovšem také nutné zakódovat, proto zde dochází k zanoření, které je reflektováno v BER.

Obrázek 2.7: Formát zpráv

Velmi názorně je vidět kódování jednotlivých prvků z následujícího příkladu:



- **smith – typ 4 (octet string), length 5, value “smith”**
- **259 – typ 2 (integer), length 2, value 0x103**

Obrázek 2.8: Formát zpráv

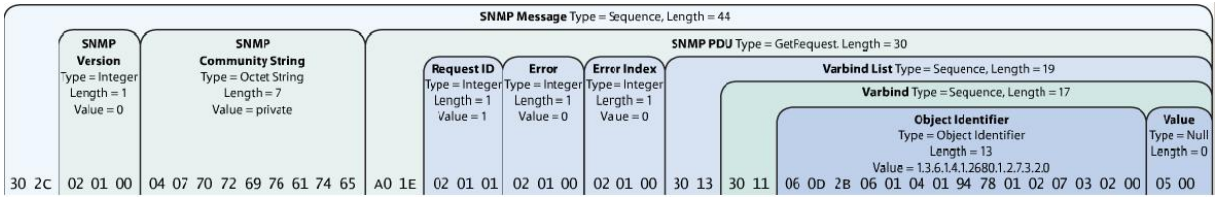
Kromě těchto pravidel obsahuje BER další. První odkazuje na kódování OID konkrétně prvních dvou čísel tohoto identifikátoru. To je třeba provádět podle vzorce, kde x značí první prvek a y druhý.

$$(40 * x) + y \tag{2.1}$$

Další pravidlo se týká zpracování čísel v OID, které jsou větší než 127. Tyto prvky je třeba rozložit do více bytů. Můžeme uvést příklad pro číslo rozložené na dva byty. To je pak po přijetí zpracováno pomocí následujícího vzorce. Zde x a y znamená zakódovanou hodnotu a z již dekódovanou.

$$(x * 128) + y = z \tag{2.2}$$

Pokud dodržíme všechna tato pravidla, jsme schopní SNMP zprávu správně zakódovat a odeslat, zde uvádíme příklad požadavku Get Request na objekt identifikovaný pomoci OID 1.3.6.1.4.1.2680.1.2.7.3.2.0.



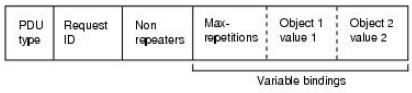
Obrázek 2.9: Formát zpráv

2.3.2 SNMPv2:

Druhá generace SNMP přinesla několik vylepšení, zvlášť v oblasti bezpečnosti, která se ale neujala kvůli obtížnosti implementace. Vzniklo několik specifikací, ale prakticky standardem pro tuto generaci je považováno SNMPv2c, které je nejnasazovanější, a proto je zde popíši podrobněji.[17][18][23] Formát rámce pro komunikaci zůstal de facto stejný. Bylo ale provedeno rozšíření datových typů z 32 bitových hodnot na 64 bitové a také přibyly nové typy zpráv.

- **Get Bulk** - jedná se o dotaz na větší množství informací od Agenta, tato zpráva má také odlišný formát, který popíši níže.
- **Inform** - nahrazuje komunikaci mezi Manažery, která v SNMPv1 probíhala pomocí trapů a na rozdíl od nich je nutné tyto zprávy potvrzovat.
- **Response** - představuje odpověď na předcházející zprávu Inform.

Jak již bylo zmíněno formát rámců pro komunikaci zůstal téměř totožný, až na nový typ zprávy Get Bulk. Do segmentu přibyly dva nové prvky, které se jmenují Non Repeater a Max Repetitions, ty nahradily původní hodnoty signalizující chybu v dotazu (Error status, Error index). První z nich značí počet prvků, které se mají načítat běžným způsobem. Další pak již náleží objektům, pro které se má aplikovat opakované načítání. To může probíhat například v rámci tabulky. Exaktně symbolizuje počet načtení.



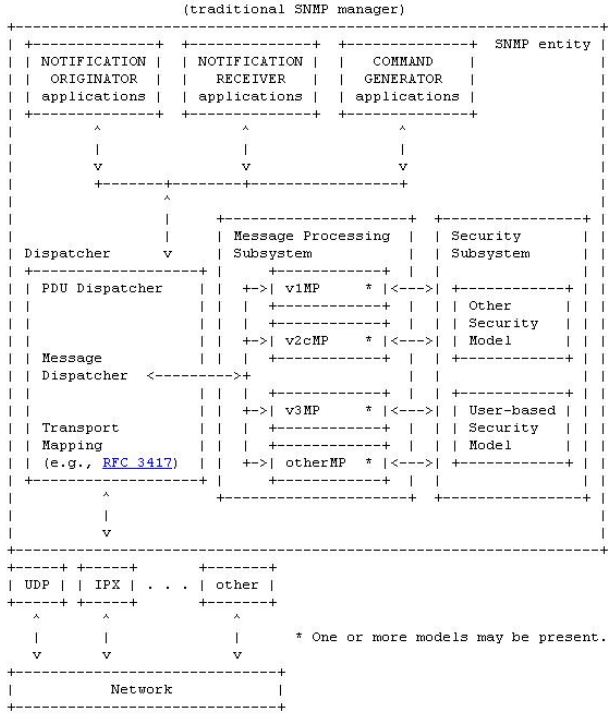
Obrázek 2.10: SNMPv2 Get Bulk struktura

2.3.3 SNMPv3:

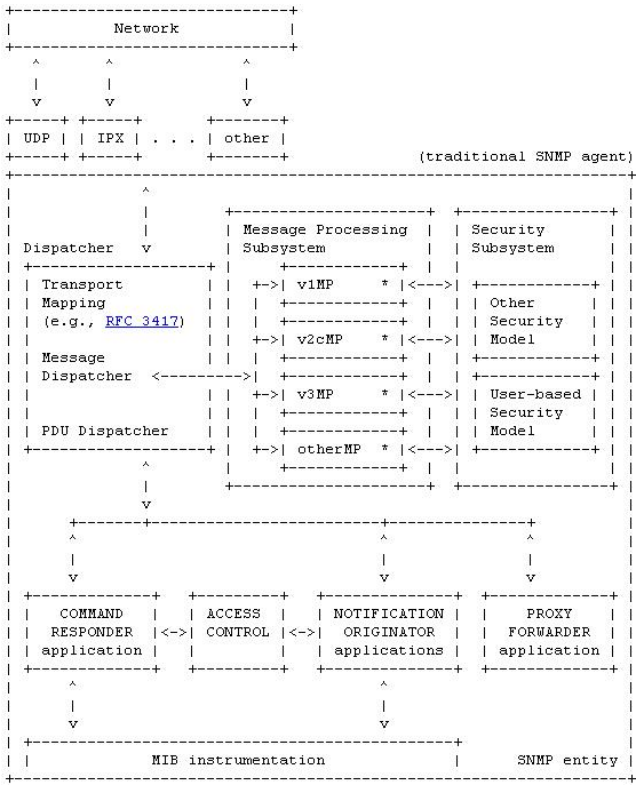
Nejnovější generace SNMP doplňuje především kýženou bezpečnost protokolu. Datové typy, druhy zpráv zůstávají stejné. Mění se ale formát rámců pro komunikaci, ty popíši níže.[19][24] SNMPv3 implementuje tyto následující pojmy bezpečnosti:

- **Autentifikaci** - datové přenosy jsou autentizovány pro ověření identity odesílajícího.
- **Soukromí** - nová podpora šifrování, zprávy tedy není možné číst ze strany potenciálního útočníka a ani provádět jejich neautorizovanou modifikaci.
- **Přístupová práva** - je možné určit uživatelům přístupová práva, která omezují jak operace nad daty, tak přístup k nim.
- **Aktuálnost zpráv** - nemožnost jejich přehrání(Antireplay protection).

V nové generaci se také setkáme s modulárním přístupem, kde každé SNMP aktivní zařízení zpracovává zprávy na několika úrovních. První úroveň řeší zprávy jako takové, druhá se stará přímo o PDU, kde najdeme již příkazy jako Get Request, Get Response a další. Tyto dvě vrstvy kooperují při přijímání a odesílání zpráv a jsou centrálně řízeny nadřazenou entitou PDU Dispatcher. Dispatcher obsahuje důležitou komponentu celého systému SNMPv3 a tou je Engine, které se stará o posílání a přijímání zpráv, jejich autentizaci, autorizaci, šifrování a kontroluje přístup k MIB objektům. Jako takové má vlastní id, které se nazývá snmpEngineID a jednoznačně určuje danou SNMP entitu, tedy zařízení, které je SNMPv3 aktivní. Blíže si celý systém můžeme představit pomocí následujícího schématu, který zobrazuje Manažera a následně Agenta.

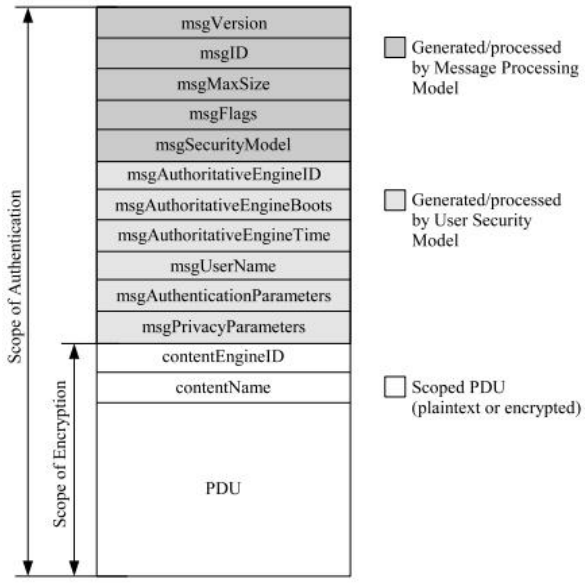


Obrázek 2.11: Struktura Managera



Obrázek 2.12: Struktura Agenta

Dále popíši formát SNMPv3 zprávy a s tím spojenou bezpečnostní politiku této verze. Více níže uvedené schéma.



Obrázek 2.13: Struktura SNMPv3 zprávy

Celou zprávu lze rozdělit do tří částí:

- **Část pro systém zpracování zpráv**
  - **Version** - SNMP verze.
  - **ID** - unikátní identifikátor, který spojuje dotaz a odpověď.
  - **Msg Size** - informace o maximální možné délce, kterou je schopen zpracovat odesílatel této zprávy.
  - **Msg Flags** - představuje řetězec obsahující ve třech nejméně významných bitech tři flagy (reportableFlag, privFlag, authFlag) - ty určují úrovně zabezpečení.
  - **Security Model** - informace pro příjemce o použitém bezpečnostním modelu odesílatelem (důležité pro správné zpracování zprávy).
- **Část pro bezpečnostní systém**
  - **AuthoritativeEngineID** - id autoritativního engine participujícího na komunikaci. Například pokud dojde k vygenerování zprávy pomocí engine jedné strany, tak tato popisovaná hodnota bude identifikovat zdroj pro typ Trap, Response, nebo Report. Pokud se bude jednat o variantu Get, GetNext, GetBulk, Set, nebo Inform, tak ji interpretujeme jako cíl.
  - **AuthoritativeEngineBoots** - hodnota specifikuje počet restartů SNMP entity.
  - **AuthoritativeEngineTime** - reflektuje časové rozmezí mezi posledním restartem SNMP entity a současností.
  - **User Name** - uživatelské jméno odesílajícího.
  - **AuthenticationParameters** - parametry autentifikace, pokud nejsou využity, tak nabývá hodnoty null.
  - **PrivacyParameters** - parametry zabezpečující soukromí SNMP zprávy, stejně jako předchozí hodnota může nabývat null, pokud nejsou použity.
- **Samotné PDU**
  - **contextEngineID** - tato hodnota, stejně jako její následující, slouží k rozlišení dvou objektů, které mají v MIB struktuře stejné OID.
  - **contextName**
  - **PDU**

Nový bezpečnostní model SNMPv3 se nazývá User-based Security Model nebo ve zkratce USM. Nová generace implementuje všechny výše zmíněné bezpečnostní prvky za pomocí komunikace v časových oknech. Zde se kontroluje stáří zprávy, zda se stále nachází v daném okně. V rámci autentifikace je nutné uchovávat informace o uživatelských účtech. Ty pak hrají roli v ověřování totožnosti uživatele, za pomocí uživatelského jména, hesla a kontrolního součtu. Ten se tvoří z celé zprávy. Sdíleným tajemstvím je klíč authKey o velikosti 16 oktetů. Ten je generován každou stranou ze snmpEngineID a uživatelského hesla. Používá se HMAC-MD5-96 a HMAC-SHA-96. První je 128 bitová MD5 a druhá 160 bitová SHA hashovací funkce, jejichž výstup je zkrácen na 96 bitů. Samotná zpráva se pak šifruje pomocí symetrické šifry CBC-DES.

Dalším prvkem nového modelu zabezpečení je systém View-based Access Control Model neboli VACM. Ten se stará o přístupová práva, která vymezují uživatelům dostupnost jednotlivých objektů v MIB databázi. VACM pracuje s jednotlivými skupinami, do kterých musí být uživatelé přiřazeni. Na základě platného kontextu, což je určitá pojmenovaná množina spravovaných objektů. Tedy akce týkající se uživatele z určité skupiny v rámci jednoho kontextu pak určují jeho práva, ty jsou pak ještě modifikována bezpečnostním modelem, který může být podle verze SNMP v1,v2c,v3, a také úrovní zabezpečení. Ta je definovaná pomocí Flagů, které můžeme najít v SNMPv3 zprávě pod položkou Msg Flags. Při plných právech může uživatel provádět varování, čtení, zapisování. Kromě těchto parametrů je možné uživateli omezit pohled na MIB pomoci pohledů (view). Ty jsou pak schopny znepřístupnit celé podstromy MIB.

2.4 Dostupné implementace SNMP

Při vývoji nových SNMP aplikací se obvykle nezačíná od píky, tedy od sestavování jednotlivých zpráv, ale je tu možnost využít již předpřipravených nástrojů. Ty jsou většinou opensource, nebo zadarmo pro akademické využití. Níže uvedu nejpoužívanější.

- **Net-SNMP** - tento software byl prvně vytvořen v roce 1992 a je pravděpodobně nejznámější na SNMP scéně. Podle aktivity stále žije a je vyvíjen, jelikož poslední release byl 3. ledna 2011. Net-SNMP představuje celý balík software, od démonů, přes CLI nástroje, až po API, které je možné přiložit k vlastnímu projektu. Projekt je vyvíjen opensource komunitou a podporuje mnoho jazyků jako Perl, Python, C. Ve své implementaci myslí i na nejnovější verzi SNMP, z těch starší podporuje v1 a v2c.
- **OpenSNMP** - je prezentováno jako multivláknové SNMPv3 engine, které je psané v C++. Celý systém je pojat objektově a poslední update byl v roce 2009. Zdá se podle reportů, že obsahuje několik chyb.
- **SNMP++** - představuje SNMP API, které je psané v C++. Podporuje stejné verze SNMP jako Net- SNMP. Dostupnost je opět zdarma. Kromě toho lze v souvislosti s tímto API najít i Agent++, který představuje kompletní projekt pro implementaci SNMP Agenta. Je to balík C++ tříd, s podporou v1,v2c a v3 SNMP verze, který je pro potřeby akademického použití zdarma.

Kapitola 3

Protokoly CDP a LLDP

Tito dva zástupci síťových technologií patří do třídy protokolů specializujících se na vyhledávání prvků v topologii a zprostředkování informací o nich. Lze tak vyčíst přímo z jejich názvů, kde zkratka CDP znamená Cisco Discovery protocol a LLDP Linka Layer Discovery Protocol. Kromě nich můžeme narazit i na varianty od jiných firem, jako jsou například Calbetron Discovery Protocol, Extreme Discovery Protocol, Foundry Discovery Protocol nebo Nortel Discovery Protocol.

Hlavní činností je tedy odesílání a přijímání informací jak o lokálním zařízení, tak o sousedních. Těchto vlastností pak využívají nejrůznější programy pro reprezentaci informací o počítačových sítích. Mezi ně patří například následující.

- **Network topology, network management system** - s jejich pomocí může adekvátně reprezentovat topologii sítě.
- **Inventory management system** - ten je schopen zjistit připojená zařízení k cílovému přepínači.
- **Emergency services-Location** - pomocí lokalizace usnadňuje konfiguraci IP telefonu. Konkrétně poskytuje informaci o portu do kterého je zařízení připojeno.
- **VLAN configuration** - tato vlastnost spadá do stejné kategorie jako předchozí, umožňuje přepínači informovat koncový prvek o VLAN použité pro přenos hlasu.
- **Power negotiation** - zde se zaměřujeme na stanovení optimální hodnoty napájení poskytované telefonnímu zařízení. Ta se stanoví na základě vyjednávacího protokolu, který operuje typicky mezi přepínačem a již zmíněným telefonem. Jako prerekvizita pro tuto funkcionalitu je technologie PoE podporovaná na obou zařízeních.

Soubor předchozích vlastností by měl poskytovat takzvanou dynamickou konfiguraci sítě. V ideálním případě by správce provedl pouze připojení koncových zařízeních a absolutní minimum zásahů do jejich nastavení.

3.1 Cisco Discovery Protocol

Pokud se podívám blíže na reprezentanta od společnosti Cisco, tak jeho přítomnost zaznamenáme od roku 1994, kdy byl uveden na trh. Tento protokol je proprietární a není tedy otevřeným standardem. V provozu ho tedy uvidíme pouze na výrobcích Cisco a několika dalších, které mají uzavřené licence s dříve jmenovanou společností. CDP tedy provádí sběr

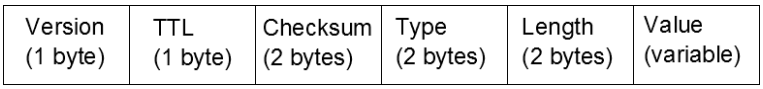
a distribuci informací o síti, které jsou nejrůznějšího druhu. Tuto aktivitu provádí na všech dostupných rozhraních, které jsou ve funkčním stavu. Jako prostředek mu slouží vícesměrová adresa, která je ve tvaru 01-00-0c-cc-cc-cc. Z formátu adresy pak můžeme vyčíst, že protokol operuje na druhé vrstvě, je tedy označován jako linkový protokol. Princip vícesměrové adresy pak umožňuje participovat na komunikaci všem zařízením, které mají zájem. Další charakteristikou je nezávislost na konkrétním síťovém protokolu, proto je použití CDP možné uplatnit na poměrně širokou škálu technologií. Sem konkrétně patří Ethernet, ATM a Frame Relay.

Jak již bylo zmíněno výše, protokol jako takový byl uveden v zařízeních Cisco v roce 1994. Jako jeho vynálezci jsou označováni Keith McCloghrie a Dino Farinacci. Během let pak docházelo k vývoji tohoto protokolu, především v množství informací, které je schopen uchovávat a distribuovat. Pro ně se vžil termín atributy a jako reprezentace byla zvolena vhodná struktura s názvem TLV. Pokud prostudujeme zkratku blíže, objasní se nám princip funkce a samotná struktura, která je popsána trojicí typ, délka, hodnota(z anglického Type Length Value). Níže uvedeme krátký seznam schopností protokolu CDP podle roku kdy byly uvedeny. Ty pak jsou na implementační úrovni reprezentované právě strukturou TLV.[4]

- 1994
  - **Identifikace zařízení** - provádí se pomocí identifikátoru.
  - **Síťová adresa**
  - **Schopnosti prvku** - tento termín jednoduše signalizuje pomocí jednoho písmene, zda se jedná o směrovač, přepínač, nebo jiné zařízení.
  - **Verze řídícího programu**
  - **Identifikace platformy** - jako příklad pro vysvětlení pojmu pak může sloužit následující. Pokud se pohybujeme mezi Cisco produkty, tak jedna z identifikovaných platforem je Catalyst 6500.
- 1996
  - **Podpora protokolu ODR** - tato technologie představuje prostředek pro snadnou konfiguraci směrování na zařízeních, které mají takzvanou charakteristiku stub, tedy nejsou z principu architektury síťové topologie zahrnuty v směrování na vnitřní části topologie. Zjednodušeně řečeno slouží pouze jako přístupový bod prvků, které jsou připojeny za nimi, což obvykle bývají koncový klienti poskytovatele. Podpora na straně CDP zjednodušovala konfiguraci takovýchto zařízení, jelikož ve svých zprávách zahrnovala informaci o síťovém prefixu.
- 1997 - zde došlo k povýšení z verze 1 na verzi 2.
  - **Přidružení Hello paketů** - představuje možnost pro jiné protokoly přibalit svoje Hello pakety na prostředky CDP protokolu.
  - **VLAN technologie** - v rámci protokolu VTP došlo k doplnění záznamů reprezentující doménová jména. V tomto kontextu byla ještě doplněna informace o počtu nativních Virtuálních LAN.
  - **Duplex portu** - identifikace, zda daný port běží v plném, nebo polovičním duplexu.

- 1999
  - **Hlasová VLAN** - zavedla se indikace, která Virtuální LAN je zamýšlena pro podporu přenosu hlasu.
  - **Zaslání požadavku** - představuje možnost přinutit zařízení na vzdáleném konci k zaslání požadovaného CDP paketu.
  - **Vyjednávání hodnoty napájení** - používá se především při nasazování IP telefonie.
  - **Třída služeb** - jedná se o rozšířenou podporu této technologie, v originále označované jako CoS. Konkrétně se zde bavíme o možnostech určit, zda daný port je označen jako důvěryhodný a pokud není, tak jak specifikovat právě dříve zmíněnou třídu služeb, která se stará o prioritizaci provozu.
- 2000 - v tomto roce se vývoj zaměřoval především na technologii SNMP.
  - **Doménové jméno** - jedná se o takzvanou plnou specifikaci, v angličtině známou pod zkratkou FQDN, tedy Fully Qualified Domain Name.Záznam odpovídá specifikaci RFC 1907, která se zabývá upřesnění standardu SNMPv2.
  - **OID** - opět validní podle RFC 1907.
  - **IP adresa** - zde prvek přijímá zprávy protokolu SNMP
  - **Lokace** - položka popisující polohu našeho zařízení. V tomto případě se jedná o polohu fyzickou a popis je ve formě řetězce znaků.
- 2001
  - **Optické rozhraní** - představuje pole pro bližší identifikaci těchto rozhraní.
- 2003
  - **Správa napájení** - jedná se o rozšířenou podporu správy napájení, která je reprezentována dvojicí položek. První z nich popisuje požadované množství energie a druhá pak množství které je naopak dostupné. Jako závěrečný přídavek byla zavedena indikace jednosměrného rozhraní, tedy takového v kterém je akceptován provoz pouze v jednom směru.

Nyní když jsme představily schopnosti a stručně popsali historii protokolu je možné přikročit ke struktuře rámce. Ten je zobrazen níže spolu s popisem nejdůležitějších položek.



Obrázek 3.1: CDP Rámec.

- **TTL** - představuje délku platnosti informace v sekundách.

- **Type** - položka může nabývat několika hodnot, které indikují typ přenášené informace. Například identifikace zařízení, síťová adresa, platforma nebo schopnosti prvku. Speciálním typem je adresa, která se rozkládá na několik podpoložek. Ty určují druh protokolu, délku adresy a jiné. Detailněji popíši princip níže v pod bodu určenému pro položku Value.
- **Length** - reprezentuje délku záznamu.
- **Value** - obsahuje již konkrétní data. Podle políčka type to mohou být následující:
  - **Device ID** - označení prvku je rozpoznáno podle hodnoty 0x0001 v sekci Type. Standardně se využívá k asociaci různých adres, které ale náleží stejnému zařízení. Toho je dosaženo právě rozpoznáním popisovaného identifikátoru. Formát je realizován plně kvalifikovaným názvem zařízení, nebo případně řetězcem, jenž je vytvořen ze sériového čísla.
  - **Address** - tento typ je reprezentován pomocí číslice 0x0002. Záznam obsahuje nejdříve čislo, které identifkuje počet položek jež ho budou v rámci následovat. Tímto způsobem je tedy možné přenést více adres najednou. Pokud se zaměříme na jejich původ, tak se jedná o zástupce přiřazené k rozhraní, přes které CDP aktuálně posílá svoje rámce. Abychom tyto informace mohli adekvátně reprezentovat, je nutné vytvořit pro ně samostatnou strukturu, kterou popíšeme níže.
    - **Protocol type** - může nabývat hodnot 1 nebo 2, které identifikují NLPID nebo 802.2 formát.
    - **Length** - identifikuje délku následujícího elementu, jenž je závislá na předchozím poli. V případě, že se jedná o NLPID variantu, tak obsahuje hodnotu 1, v opačném případě 2.
    - **Protocol** - představuje již konkrétní protokol. Do podporované množiny patří například DECNET, AppleTalk, XNS, IP, IPv6 a další. Každý z nich má přiřazený adekvátní identifikátor.
    - **Address length** - informuje o délce následující položky, kterou je již z kontextu adresa.
    - **Address** - jedná se o označení přiřazené rozhraní, nebo systému jako celku. Pod tím si můžeme představit například přepínač, který má nastavenou VLAN pro jeho správu s adekvátní adresou.
  - **Port ID** - identifikátoru portu byla přiřazena hodnota 0x0003. Je složen z řetězce, který slouží k označení portu přes který jsou posílány CDP pakety.
  - **Capabilities** - představuje sumarizaci schopností prvku. Jelikož informaci reprezentuje pouze jedna hodnota a jednotlivé prvky mohou disponovat různými vlastnostmi, je nutné provést výpočet při jejím sestavování a následném extrahování. K tomuto účelu slouží logická operace nebo, která je prováděna po jednotlivých bitech dokud není složena kompletní informace. Níže uvádíme výčet definovaných schopností a k nim následně přiřazené hodnoty.
    - **0x01** - označuje směrovač, který pracuje nejméně s jedním sítovým protokolem.
    - **0x02** - jedná se o transparentní most.
    - **0x04** - reprezentuje Source-route mosty.

    - **0x08** - identifikuje opět přepínač, tedy podobně jako hodnota 0x02. Rozdíl je ovšem v tom, že u tohoto zařízení se nepočítá, že provozuje Spanning-Tree Protokol, tedy musí být nasazeno do topologie neobsahující smyčky.
    - **0x10** - zařízení pracuje s minimálně jedním sítovým protokolem, v kontextu se považuje za osobní počítač.
    - **0x20** - označuje most, který ale nepřeposílá IGMP pakety na porty, které nejsou směrované.
    - **0x40** - tento prvek pracuje na první vrstvě, identifikujeme ho tedy jako opakovač.
    - **0x80** - hodnota reprezentuje VoIP telefon.
    - **0x100** - konfigurace bitů značí zařízení, které je možné spravovat vzdáleně.
- **Version** - této položce byla přiřazena hodnota 0x0005. Jedná se o řetězec, který poskytuje informaci verzi programu, který operuje na zařízení.
- **Platform** - asociovaná hodnota je 0x0006. Informace je reprezentována řetězcem, který specifikuje platformu sledovaného zařízení.
- **IP Prefix** - poslednímu prvku v CDP zprávě bylo přiřazeno číslo 0x0007. Informace, která je pomocí nějž přenášena, obsahuje nula nebo více prefixů protokolu IP. Jejich počet je opět odvozen díky struktuře TLV. Reprezentace jednoho prefixu se skládá z pěti bajtů, kde první čtyři jsou určeny pro reprezentaci prefixu jako takového a poslední slouží pro definici použité masky. Ta je reprezentovaná v CIDR notaci, tedy číslem v rozsahu 0 až 32. Výše popsaná reprezentace je spjata s každou jednotlivou sítí, která je přímo připojená k směrovači. Zajímavostí je, že právě tato položka je použita při komunikaci mezi spoke a stub směrovačem v kontextu protokolu ODR.[3]

3.2 Link Layer Discovery Protocol

Tento zástupce rodiny protokolů, které jsou používány pro správu a sledování sítě, je na rozdíl od výše zmíněného CDP průmyslovým standardem, který je označován jako 802.1AB. Princip funkce je velmi podobný variantě od Cisca, tedy opět dochází k odesílání a kolekci informací o jednotlivých zařízeních. Tyto informace jsou velmi prakticky uloženy v MIB podle RFC2922. Což nám umožňuje k nim přistupovat pomocí technologie SNMP. Z názvu je pak patrné, že protokol operuje na druhé vrstvě, tedy je označován jako linkový. Komunikace probíhá stejně jako u CDP pomocí vícesměrové adresy, která má formát 01-80-c2-00-00-00. Pokud se podíváme na jednotlivé zprávy zasílané LLDP, tak zjistíme, že jsou složeny ze sekvence TLV položek. Což je opět velmi podobné variantě od Cisca. LLDP je vznikem mladší než CDP a jeho samotnou existenci si vynutila právě potřeba široce respektovaného průmyslového standardu. Je snadné si představit, jaké problémy museli správci síti řešit, pokud chtěli konzistentní správu a dohled nad sítí, která není složena pouze ze zařízení od jednoho výrobce. Tento fakt tedy přiměl Cisco i další společnosti se podílet na vzniku tohoto standardu a dá se říci, že dnes se těší široké podpoře.

Mezi základní informační položky, které LLDP podporuje, pak patří následující. V první řadě je to popis konkrétního systému a jeho identifikátor. Dále můžeme najít opět popis a identifikátor pro cílový port. Mezi další položky patří specifikace IP adresy rozhraní určeného pro správu prvku. Ze záznamů můžeme ještě zjistit schopnosti prvku, tedy jestli je určen pro přepínání, nebo směrování dat, případně jiné exotičtější schopnosti. Popsány

jsou také informace o konkrétních Virtuálních LAN, dále specifičtější popis lokální fyzické a MAC vrstvy. V neposlední řadě pak můžeme sbírat informace o napájení v rámci technologie PoE. Jelikož předchozí výčet položek nebyl pro některé aplikace dostatečný, přikročilo se k rozšíření standardu. Tato snaha vyústila ve vznik technologie LLDP-MED. Což ve zkratce znamená rozšíření protokolu o komunikaci mezi koncovými zařízeními a jejich protějšky, což může bývá nejčastěji přepínač. Těmito koncovými zařízeními se obvykle myslí IP telefony, kde je opět nutno řešit řadu problému s napájením, prioritizací provozu a přiřazováním do různých Virtuálních podsítí.

Cílem LLDP-MED rozšíření je tedy poskytnout tyto služby. První z nich je autodetekce politik na síti, sem patří především zmíněná prioritizace provozu, ať už se jedná o druhou, nebo třetí vrstvu. Tento princip by měl zaručit filosofii používání síťových prvků, kdy jej stačí pouze připojit a používat. Dále sem patří již u CDP zmiňovaná lokalizace zařízení. To usnadňuje přehled o topologii a také umožňuje vylepšenou funkci záchranných složek. Jak již bylo výše uvedeno u IP telefonie se také často z praktických důvodů řeší napájení přes technologii Ethernet, proto LLDP-MED přináší rozšíření těchto informačních položek. Mezi poslední pak patří podpora pro inventarizaci síťových zařízení, která se skládá z již zmíněné lokalizace a možnosti zjistit podrobnosti o prvku, tedy konkrétně identifikace výrobce, verze software a hardware, případně další věci jako jsou sériová čísla a jiné.

Nyní jakmile jsme objasnili obecnou funkci technologie je možné přikročit k popisu bližšího implementačního detailu na jednotlivých prvcích. Zařízení, které podporuje tento protokol implementuje takzvaného LLDP Agenta. Jeho funkčnost je velmi blízce spjata s principem SNMP MIB struktur, které slouží pro odkládání získaných a odesílaných informací. S tímto přístupem je pak možné přidat různá rozšíření, které operují s proprietárními informačními položkami.

Pokud prozkoumáme podobu architektury jednotlivých větví MIB uložiště, objevíme oproti technologii CDP podstatně bohatší strukturu. Z ní je možné vyčíst jak informace o lokálním systému, tak o systémech vzdálených. Tímto termínem je označován prvek, který je připojen přímo k systému na nějž se prostřednictvím SNMP dotazujeme.

Funkce výše zmíněného Agenta lze definovat v duchu tří rolí, v kterých může operovat.

- **Odesílací mod** - prvku je povoleno pouze šířit informace o schopnostech a stavu svého systému.
- **Přijímací mod** - tato varianta je totožná s předchozí s tím rozdílem, že jde pouze o operaci přijímání.
- **Přijímací a odesílací mod** - pokud je aktivována tato funkcionalita, tak se Agent chová podle vzoru sjednocení předchozích variant.

Tyto výše uvedené styly schování je nutné implementovat, proto byly zavedeny následující moduly.

- **Odesílací modul** - pokud je povolený, tak posílá v pravidelných intervalech informační struktury TLV. Jakmile ale dojde k jeho zakázání, pak jsou následně odeslány rámce, které obsahují TTL, v kontextu LLDPDU, s hodnotou 0. To umožní podle definice stavového automatu odebrat data z lokálních MIB vzdálených prvků o tomto zařízení.
- **Přijímací modul** - stará se o příjem informací o vzdálených prvcích, plus také obsluhuje logiku o již zmíněném TTL mechanizmu. Pokud dorazí rámec od souseda, tak

je time-to-live hodnota asociovaná s ním a pro něj platnými daty aktualizována právě z tohoto rámce. V kontextu také probíhá kontrola na nenulovost TTL, jinak by bylo nutné neaktuální informace vymazat z MIB.

Dále představíme strukturu LLDP rámce. Ten obsahuje čtyři povinné jednotky TLV a jednu doplňkovou. Které to jsou lze pak odvodit již z názvu položek.[9]

Obrázek 3.2: LLDP Rámec.

- **Chassis ID** - identifikuje sítový prvek.
- **Port ID** - reprezentuje zdrojový port.
- **TTL** - značí délku platnosti informace v sekundách.
- **End** - představuje konec LLDP rámce.
- **Optional** - slouží k doplnění dodatečných informací, které kompletují celou dostupnou množinu. Pod touto položkou se skrývají tři třídy, které definují jaká data budou tímto způsobem přenášena.
  - **Základní TLV množina** - obsahuje pět prvků.
    - **Port description** - jedná se o popis portu pomocí řetězce znaků. V kontextu SNMP je ekvivalentní k ifDescr objektu.
    - **System name** - definuje nastavený název systému. Stejnou informaci lze získat pomocí sysName v SNMP.
    - **System description** - představuje již kompletnější popis prvku, tedy nejen jméno ale i verzi programu a technického vybavení. V rámci SNMP lze získat stejnou informaci pomocí sysDescr.
    - **System capabilities** - značí schopnosti elementu. Strukturně se jedná o dva bajty, kde bity 0 až 7 jsou vyhrazeny konkrétní vlastnosti, dále 8 až 15 jsou rezervovány.
    - **Management address** - informuje o adrese na které odpovídá LLDP Agent obsluhující se o tento prvek.
  - **IEEE 802.1 TLV množina** - obsahuje čtyři prvky.
    - **Port VLANID** - indikuje příslušnost VLAN identifikátoru k příslušnému portu.
    - **PPVLAN ID** - slouží k stejnému účelu jako předchozí položka s tím rozdílem, že je asociován port s protokolovým VLAN ID. S tím souvisí i indikace v sekci Flags, kde se nachází informace, zda je tato technologie podporovaná a kolik identifikátoru bylo přiřazeno.
    - **VLAN name** - spojuje jmenné označení s VLAN ID. Stejná informace je dostupná přes dot1QVLANStaticName objekt.

    * **Protocol identity** - představuje množinu protokolů, které operují na daném portu.
  - **IEEE 802.3 TLV množina** - obsahuje čtyři prvky.
    * **MAC/PHY configuration/status** - značí schopnosti a aktuální nastavení odesílajícího zařízení. Mezi sledované hodnoty se řadí rychlost a duplex. Kromě těchto informací jsme schopni rozeznat, zda konfigurace byla získána auto negociací, nebo manuálním zásahem.
    * **Power via MDI** - informuje o napájecích schopnost tohoto prvku.
    * **Link aggregation** - rámec, který přišel přes tuto linku a obsahuje tuto informaci identifikuje, zda linka může být agregována a jestli již je nebo není. Pokud je, tak spolu nese ještě informaci o jejím agregačním identifikátoru.
    * **Maximum frame size** - indikuje maximální velikost rámce, který je možné přenést přes tento spoj.

3.3 Srovnání CDP a LLDP

Pro výsledky této práce je důležité stanovit technologii, která bude co možné nejšířeji podporovaná a bude poskytovat co nejvíce informací o síti jako takové. Proto jsme také zahrnuli tyto dva protokoly jako nejvýznamnější zástupce z této části správy a dohledu nad sítí. Níže provedeme jejich porovnání a rozhodneme, který bude primárně použit. Nicméně v širším pohledu na věc by nebylo moudré ten druhý cíleně zavrhnout už z možnosti, kdy by v některých případech mohl být jako jediný podporovaný. Proto porovnání níže slouží pouze pro prioritizaci použití těchto technologií.

- **Zjištění schopností prvku** - tuto operaci podporují oba dva protokoly. Rozdíl nastane, až pokud chceme zajít do větších detailů. Zde pak vítězí LLDP v rozšíření MED. V tomto případě je schopen navíc zaznamenat aktivní stav různých schopností, které sledované zařízení umožňuje aktivovat a deaktivovat.

- **Zjištění rychlosti a nastavení duplexu** - zde opět vítězi LLDP-MED, jelikož je schopen detekovat jak rychlost tak duplex, kde naproti tomu CDP umožňuje zjistit pouze duplex.

- **Detekce síťové politiky** - zde opět poskytují dostatečnou podporu oba protokoly, ale LLDP-MED umožňuje získat podrobnější informace, jelikož sledování dělí do dvou skupin. První sleduje služby určené pro signalizaci a druhá služby pro přenos dat. Nicméně kritické parametry technologií jsou pokryty v CDP i v LLDP-MED.

- **Detekce fyzické polohy prvku** - podpora je u obou protokolů, nicméně LLDP-MED poskytuje širší paletu formátů. Těmi je myšlen způsob uložení a zobrazení těchto informací.

- **Zjištění informací o napájení** - v tomto případě je situace složitější, proto shrnu stav na obou stranách. LLDP-MED obsahuje následující položky. První z nich je zobrazení způsobu, jakým je prvek napájen. Možnosti jsou například ze sítě, ze záložní baterie, případně úplně jiného zdroje. Další uchovávaná informace je priorita toho zařízení, tedy jak moc velkou škodu v celém systému způsobí fakt, že zařízení bude bez proudu. Posledním elementem je kolik daný prvek potřebuje energie k své činnosti.

CDP naproti tomu poskytuje pouze dva parametry. První značí množství energie požadované od zařízení. Tato hodnota se může měnit podle toho v jakém módu prvek pracuje. Následně druhý parametr reprezentuje dostupnou hodnotu napájení. Tyto dvě položky jsou v rámci CDP esenciální právě pro zabudovanou funkci vyjednávání výsledné hodnoty. Toto rozšíření se pak již v rámci standardu LLDP-MED nevyskytuje.

- **Funkce inventáře** - zde je situace u obou technologii stejná. Oba protokoly mají plnou podporu této funkcionality, která je velmi užitečná u zařízení, které neprovozují SNMP Agenta, ale podporují protokol CDP nebo LLDP-MED. Další výhodou takového přístupu je možnost dotazovat pouze prvek, který agreguje připojení těchto zařízení, což bývá obvykle přepínač.

- **Rozšíření důvěry v síti** - zde má pro změnu navrch protokol CDP, který umožňuje rozšířit důvěru v značení priority provozu až na hranici koncového zařízení. Hlavní výhoda je tedy v snížení zátěže agregujícího prvku. Tato funkce není v LLDP-MED pokryta.[5]

Po zhodnocení jednotlivých pozitiv a negativ obou technologií, tedy jak CDP tak LLDP, jsme došli k závěru, že vhodnější bude prioritizovat užití LLDP s rozšířením LLDP-MED. Klíčovou vlastností této technologie je především kvalitně popsaný průmyslový standard a dobrá podpora v produkčních zařízeních. Kromě těchto výhod je LLDP ve většině uchovávaných informací podrobnější než jeho konkurent. Co se týká jeho nedostatků v podobě několika nepodporovaných rozšíření, tak si myslím, že nejsou schopny ovlivnit činnost vytvářené aplikace. Proto jsme se tedy rozhodli upřednostnit standard LLDP před proprietárním CDP.

Kapitola 4

OMNeT++

Následující kapitola se zabývá popisem simulačního nástroje OMNeT++. Postupně popíši jeho jednotlivé charakteristiky a uplatnění, kde začnu nejdříve simulátorem jako takovým, přes který se dostanu k jeho rozšíření, které se jmenuje INET Framework a následně krátce shrnu stav ANSA projektu.

Nástroj popisovaný v této diplomové práci se zabývá právě přípravou dynamických dat pro tento simulátor a jeho rozšíření, proto je důležité se s tímto produktem a jeho rozšířeními seznámit.

4.1 Popis simulátoru

OMNeT++ je diskrétní objektově orientovaný simulátor, který poskytuje modulární prostředí pro reprezentaci různých systémů, tedy nejen počítačových sítí, ale i business modelů, nebo například validace hardware ale i velké množství jiných. Kromě těchto možností se během času prosadila i podpora pro paralelní simulace ve spolupráci s externím balíčkem MPI, který funguje pod různými platformami. Jako další technologie bylo zahrnuto rozšíření Akora. Projekt vznikl v roce 1992 a za jeho zakladatele je považován Andreas Varga, který jeho první vlastnosti implementoval na půdě univerzity v Budapešti. V současnosti jeho vývoj nadále pokračuje pod záštitou ústavu v Karlsruhe a aktuální verze v době psaní práce je označována jako 4.2. Alfou a omegou celého systému je princip hierarchie, který se aplikuje jak na základní komponenty systému, tak na jeho jednotlivá rozšíření. Základní stavební kámen každé simulace je pak jednotka modulu od níž se pak vše následující odvozuje. Odtud pak získáváme pomocí zanořování požadovanou komplexnost sledované sítě nebo jiného systému. Jednotlivé moduly pak vzájemně komunikují pomocí zasílání zpráv. Ty mohou být přenášeny primárně přes brány, které slouží jako propoje. Komunikace může probíhat jak mezi prvky na stejné úrovni komplexity a hierarchie, tak na úrovních různých. Z tohoto konceptu pak vyplývá kýžená univerzálnost software, který je možné využít pro různé účely. Modulárnost pak také napomáhá znovu použitelnosti jednotlivých komponent systému, což ulehčuje konstrukci složitějších modelů. Simulátor je možné stáhnout ve formě zdrojových kódů, které je třeba na platformě přeložit. Tento kód je kompatibilní s většinou běžně používaných operačních systémů. Co se týká licenčních podmínek, je OMNeT distribuován jako software s otevřeným kódem. Ovšem pouze pro akademické využití. Pokud bychom jej chtěli využít pro výdělečnou činnost, je třeba zvolit komerční variantu označovanou jako OMNEST.[22]

4.2 Popis prvku modulu

Jak již bylo zmíněno, modul a možnost zanoření je důležitá vlastnost celého systému, která zaručuje jeho univerzálnost a modulárnost. Pokud se podíváme na modul jako takový blíže, identifikujeme tři typy.

- **Jednoduchý typ** - jedná se o základní prvek v angličtině označovaný jako simple, s jehož pomocí jsou vytvářeny všechny prvky simulovaného systému. Tento modul představuje již nedělitelnou jednotku a také proto obsahuje již svoji vlastní implementaci chování v jazyce C++.

- **Složený typ** - je tvořen pomocí jednoduchých prvků a v angličtině nazýván jako compound. Typicky obsahuje jak několik základních modulů, které pomáhají dotvářet jejich funkcionalitu, ale také může obsahovat další složené, tedy stupeň zanoření není teoreticky omezen.

- **Systémový typ** - představuje speciální typ. Ten pak obsahuje všechny prvky simulace v sobě. Je možné si ho představit jako kořenový uzel v nějaké stromové struktuře.

Abychom byli schopni simulovat nějaký reálný proces je třeba aby jednotlivé entity v námi sledovaném systému měly možnost komunikace. Jak už bylo zmíněno výše, moduly mezi sebou komunikují pomocí bran. Nicméně je možné zvolit i alternativní variantu a to takovou, že jednotlivé elementy vystupující v komunikaci, řeší výměnu zpráv přímo, tedy bez asistence přidružených bran. Komunikace tedy obecně je možná i mezi různými úrovněmi hierarchie. Aby došlo k zachování žádoucích vlastností systému, je zde třeba udělat následující omezení. Pokud komunikujeme mezi různými komponentami modelu na různých hierarchických úrovních, tak je třeba využít služeb nadřazených, nebo rodičovských modulů. Pokud bychom propojili jednotlivé následnické moduly dvou rodičovských, tak by byla značně omezena modularita a znovu použitelnost tohoto návrhu, proto je tato architektura z principu v OMNeTu zakázána a komunikace musí probíhat přes rodiče. Jiná situace nastane, pokud by bylo třeba propojit dva prvky na stejné úrovni hierarchie v jednom nadřazeném modulu. V tom případě je konstrukce povolena a nazývá se connection. Samotné zprávy pak mohou nést určitá data, která mají libovolný typ. U těchto jednotlivých typů propojení pak specifikujeme jejich logický směr, tedy jestli jde o spojení vstupní nebo výstupní. Pokud se ale jedná o spojení typu connection, tak existuje i možnost vstupně výstupního propojení, které je přímo terminologicky popsané "inout". Blíže se pak jednotlivým variantám budeme věnovat v části práce popisující syntaxi modulů.

Když jsme popsali základní stavební kameny a principy nástroje OMNeT, tak se můžeme přesunout k bližšímu popisu jednotlivých komponent, které se starají o realizaci simulace. Mezi ně patří následující:

- **Sim** - představuje centrální prvek celé simulace, který obsahuje set knihovních funkcí, které jsou v jejím průběhu využívány. Dále se stará o korektní funkcí kalendáře událostí, tedy generování jednotlivých událostí a jejich správné provádění, kontroluje také běh samotných modulů. S tím je spojené také generování různých výjimek a také protokolování nejrůznějších událostí.

- **Envir** - nejdůležitější součástí tohoto prvku je právě hlavní smyčka simulačního programu, která se stará o průchod kalendáře událostí plus provádí inicializaci obsluhy jednotlivých položek. Představuje tedy základní vstupní a výstupní element celého

simulačního nástroje. Tyto kanály lze identifikovat následovně. Jako vstup slouží pro každou simulaci soubor omnetpp.ini, ten je před startem vždy načten právě touto komponentou. Nakonec pro výstup slouží rozhraní ev, pomocí něhož jsou vypisovány různé výjimky vzniklé při průběhu simulace.

- **CMDENV** - jedná se o uživatelské rozhraní ve smyslu přístupu k simulaci, tedy její kontrole, spouštění, nebo modifikaci jejích parametrů. Konkrétně CMDENV představuje rozhraní příkazového řádku, které jako takové nese sebou známé výhody a nevýhody tohoto přístupu. Je podstatně rychlejší než grafické rozhraní a můžeme v náš prospěch využít programově řízené spouštění pomocí skriptů. Mezi nevýhody patří především slabá vizualizace sledovaného procesu a celková vyšší náročnost na ovládání.

- **TKENV** - stejně jako u předešlé komponenty se jedná o reprezentanta výstupní části systému. V tomto případě je ale použito technologie Tcl\Tk, která je známá svoji multiplatformností. Oproti předešlému přístupu je tato varianta mnohem vhodnější pro vizualizaci problému, jelikož se jedná o grafické uživatelské rozhraní. Mezi další výhody pak patří možnost snadného ovlivňování běhu simulace, jako je například krokování, případně řízení rychlosti provádění jednotlivých událostí. Na druhou stranu je prostředí pomalejší a neposkytuje takovou flexibilitu při spouštění jako varianta založená na příkazovém řádku.

- **Model Component Library** - tato entita představuje virtuální knihovnu všech zahrnutých modulů. Tyto moduly jsou zde reprezentovány konkrétní podrobnou definicí, tedy jsou zde přítomny definice souborů popisujících topologii simulovaného systému. K tomu patří i popis jednotlivých propojů a zpráv, které putují mezi nimi. V neposlední řadě pak specifikace chování popisovaných modulů.

- **Executing Model** - představuje poslední nepopsanou komponentu v tomto systému, která reprezentuje již finální simulaci, proto obsahuje instance jednotlivých modulů, z nichž je sledovaný systém složen. Plus také již zmíněný systémový modul, který hraje roli předka všech modulů. Pomocí něj je pak umožněno k jednotlivým podřízeným modulům přistupovat.[11]

4.3 Uživatelské rozhraní OMNeTu

Mezi aktuálně propagované charakteristiky OMNeTu patří jeho grafické prostředí, které je rozděleno do dvou větví a to větev pro sledování samotné simulace a větev pro její vytváření, tedy pro tvorbu modulů a specifikování jeho jednotlivých vlastností. První část je založena na multiplatformní technologii Tcl\Tk, která je dostupná jak pod Windows tak i pod Linuxem. Ta umožňuje poměrné snadnou orientaci v průběhu simulace, nastavování jejich základních parametrů a také ovlivňování jejího diskrétního průběhu. Jedná se především o možnost krokování, případně nastavení rychlosti průběhu.

Větev v níž se pozornost zaměřuje na vytváření simulace je postavená na poměrně známém a volně dostupném vývojovém prostředí Eclipse. Toto prostředí je samo o sobě dost modulární a komplexní, proto nebyl problém jej přizpůsobit potřebám vývoje v kontextu OMNeTu. Klíčovou charakteristikou při výběru tohoto IDE byla pravděpodobně právě přítomnost prvku perspektivy a možnosti bližší specifikace jednotlivých spouštěcích konfigurací. Pokud se blíže zaměříme na perspektivu, kterou využívá OMNeT, tak zjistíme, že

v ní je možné prohlížet jednotlivé simulace, které jsou v čistém OMNeTu reprezentovány samostatnými projekty. Tedy blíže popsáno je tato modifikace schopna graficky interpretovat obsah samotných NED souborů, které specifikují v případě síťové varianty simulace topologii sledované sítě. Tato vlastnost je velmi výhodná pro přehlednost a kontrolu nad její konstrukcí. Co se týká specifikace spouštěcí konfigurace, tak zde můžeme nastavit konkrétní proměnné ovlivňující výsledný průběh zkoumaného procesu. Mezi ně patří například specifické jádro, které bude obsluhovat naši simulaci, počet vláken která budou využita, nebo nastavení různých dynamických parametrů.

4.4 Konfigurace simulace

Každá simulace před svým spuštěním potřebuje alespoň několik základních parametrů pro svůj smysluplný běh. To se dá zařídit v OMNeTu několika způsoby. První a nejsnazší je zakomponovat parametry přímo do definice jednotlivých modulů v NED souborech. Což už z principu se nejeví jako nejlepší cesta pokud tvoříme nějaký složitější model, nebo v případě ANSA projektu, kde cílem je vytvořit nástroj pro automatickou analýzu a verifikaci. Je třeba se tedy přesunout na další možnost, kterou je specifikace parametrů v souboru omnetpp.ini. Syntaxe tohoto souboru umožňuje zadat širokou škálu konfiguračních hodnot, kde nejzajímavější z nich je specifikace XML souboru, který dále nastavuje simulovaný systém. Struktura XML je velmi vhodná pro automatizované zpracování, proto již v předchozích pracích spojených s projektem ANSA byl kladen velký důraz na jeho využití a specifikaci jednotné struktury, pomocí niž budou inicializovány dynamické parametry simulace. Dále v textu specifikuji jednotlivé struktury, které jsou využívané již dokončenými, nebo rozpracovanými rozšířeními v rámci projektu ANSA. Přesné určení podoby těchto XML souborů je důležité vzhledem k tomu, že tato práce se také zabývá jejich naplněním a s tím spojenou inicializací simulace.

4.5 Popis topologie

Jakmile jsme popsali bližší principy a seznámili se s uživatelským rozhraním nástroje OMNeT, je možné postoupit k prozkoumání způsobu, jak se zapisují jednotlivé moduly a jak se tvoří celková topologie. Ta je uložena v souborech s příponou NED, které obsahují zápis kódu ve stejnojmenném jazyce.

4.5.1 Jazyk NED

Tento jazyk prakticky propojuje svůj definiční obsah, který reprezentuje právě podobu topologie simulované sítě, nebo jiného systému, s konkrétní implementací chování daného modulu. Propojení implementace a vizualizace je jednoduše zajištěno požadavkem na stejné pojmenování jak deklarovaného prvku topologie, tak třídy, která popisuje jeho chování v jazyce C++. Jakmile je zajištěna tato základní konzistence, je nutné popsat několik základních charakteristik zápisu v jazyce NED. Jelikož v celém systému existují tři konkrétní klíčové elementy, rozdělím tedy popis podle nich:

- **Definice jednoduchého modulu** - tento prvek se pozná podle klíčového slova simple, které se specifikuje před jeho názvem. Pak následuje již popis tohoto modulu. Ten je uzavřen ve složených závorkách a obsahuje dvě další klíčová slova, která upřesňují jeho vlastnosti. Obě dvě jsou nepovinná, tedy pro definici jednoduchého modulu stačí

pouze název. Pokud se rozhodneme zahrnout i zbývající dvě, tak můžeme specifikovat parametry modulu pomocí klíčového slova parameters. Zde se nabízí například úprava vizuálního prvku stránky, pomocí specifikace ikony, která se bude zobrazovat při vykreslení v Tcl\Tk. Dalším je pak sekce gates, která specifikuje jednotlivé brány, pomoci nichž se bude modul připojovat k jiným a komunikovat s nimi. Ty jsou definovány pomocí jednoznačného identifikátoru, který má podobu znakového řetězce a směru, v němž má být brána používána. Zde připadá v úvahu vstupní, výstupní, či kombinovaná varianta. Níže uvádíme jednoduchý příklad definice.

Příklad 4.1: Jednoduchý modul

```
simple Queue
{
    parameters:
        int capacity;
        @class(mylib::Queue);
        @display("i=block/queue");
    gates:
        input in;
        output out;
}
```

- **Definice složeného modulu** - složený modul je trochu komplikovanější než jednoduchá varianta. Je označen klíčovým slovem module, na rozdíl od simple v předchozím případě. S ním má však společné sekce parameters a gates. Z principu složeného modulu, ale muselo být doplněno klíčové slovo submodules, které specifikuje právě ty moduly, z kterých se tato složená varianta skládá. Jednotlivé položky v této sekci obsahují názvy dalších modulů, které mohou být složené, nebo jednoduché. Nicméně ve výsledku se OMNeT musí dostat až k jednoduché variantě, aby mohl specifikovat finální funkcionalitu pomocí souboru definujícího příslušnou C++ třídu. Jako poslední položka ve složeném modulu je klíčové slovo connections, kde se popisují jednotlivé propoje mezi následnickými moduly v tomto složeném. Pro korektní popis je nutné specifikovat jak rozhraní participujících pod modulů, tak směr v kterém probíhá komunikace. Zde jsou varianty stejné jako v případě sekce gates. Tedy existuje vstupní, výstupní, a kombinovaná možnost. Jako další možnost byla přidána dědičnost, kde potomek je schopen přidávat také nové moduly a nové propoje k těm již získaným od svého předka. Samozřejmostí je definice nových bran a parametrů. Kromě tohoto je mu umožněno se odkazovat na zděděné prvky předchůdce. Bližší představu je možné si udělat pomocí příkladu.

Příklad 4.2: Složený modul

```
module WirelessHostBase
{
   gates:
       input radioIn;
   submodules:
       tcp: TCP;
       ip: IP;
       wlan: Ieee80211;
   connections:
       tcp.ipOut --> ip.tcpIn;
       tcp.ipIn <-- ip.tcpOut;
       ip.nicOut++ --> wlan.ipIn;
       ip.nicIn++ <-- wlan.ipOut;
```

```
        wlan.radioIn <-- radioIn;
}

module WirelessHost extends WirelessHostBase
{
    submodules:
        webAgent: WebAgent;
    connections:
        webAgent.tcpOut --> tcp.appIn++;
        webAgent.tcpIn <-- tcp.appOut++;
}
```

- **Definice kompletní topologie** - tento popis je definován pomocí klíčového slova network, které se vyskytuje opět před názvem modulu. Ten samotný již představuje vrchol v hierarchii, proto obsahuje všechny elementy simulace a tedy je všem nadřazen. Mezi jeho sekce patří již známé submodules a connections, které jsme již popisovali u složeného modulu. Dále existuje sekce s názvem types, pomocí níž jsme schopni definovat nový typ, který je možné využívat v popisu tímto jazykem stejným způsobem jako v jiných programovacích jazycích. V oblasti počítačových sítí se využívá především k definici vlastností linek, které propojují jednotlivé prvky simulace. Můžeme tedy specifikovat chybovost, rychlost nebo zpoždění linky. Více ukazuje následující příklad.

Příklad 4.3: Kompletní topologie

```
network Network
{
    parameters:
        host*.ping.timeToLive = default(3);
        host{0..49}.ping.destAddress = default(50);
        host{50..}.ping.destAddress = default(0);

    submodules:
        host0: Host;
        host1: Host;
        host2: Host;
        ...
        host99: Host;
}
```

4.6 INET Framework

Jedná se o rozšíření simulačního nástroje OMNeT, kde cílem bylo zahrnout podporu především pro protokoly ze světa TCP/IP. Mezi konkrétní reprezentanty podporované tímto rozšířením patří IPv4, IPv6, TCP, UDP, Ethernet, PPP a další jako MPLS, nebo zástupci ze standardu 802.11, jež byli původně implementovány v bezdrátové variantě tohoto balíčku. Předtím než INET Framework dostal současnou podobu, musel projít řadou změn. Historicky první pojmenování bylo IPSuite a jeho vznik je přičítán univerzitě v Karlsruhe, kde kolem roku 2000 vznikl. Jeho vývoj pokračoval do roku 2003, kdy se ho ujmul tvůrce OMNeTu Andreas Varga a provedl v něm řadu změn, především doplnil potřebnou dokumentaci. Další změny pak nastaly v roce 2004, kdy došlo k přejmenování na současný název INET Framework, a byla doplněna funkcionalita protokolu TCP a OSPFv2. V současnosti

vývoj stále probíhá a aktuální verze nese název INET-20111118. V této verzi se autoři soustředí především na kompatibilitu s verzí OMNeTu 4.2. Zde je vidět, že projekt stále žije a autoři jsou aktivní. S tímto také souvisí činnost na naší fakultě, kde projekt ANSA na INET Framework navazuje a dále rozšiřuje jeho funkcionalitu. V tomto směru byli kontaktováni tvůrci INETu a bylo navrhnuto zakomponování výsledků práce projektu VUT FIT ANSA do oficiálního vydání balíku.[7] Princip fungování INET Frameworku je prakticky v duchu OMNeTu, tedy je majoritně složen z jednoduchých modulů, které ve vzájemné komunikaci využívají již výše popsané zprávy. Platí zde tedy stejné možnosti modularity jako v čistém OMNeTu. Můžeme jednoduše tyto moduly skládat a vytvářet složitější a reálnější prvky simulace. Několik těchto složených modulů na vyšší úrovni hierarchie je již hotovo v INET Frameworku. Veškerá simulace pak probíhá přes zkompilovaný spustitelný soubor INET, který pak tvoří simulační jádro celého frameworku. Pokud tedy chceme spustit jakoukoliv simulaci pod tímto rozšířením, je třeba využít tohoto souboru, a tím pádem mít zkompilovaný INET. Neméně důležité je, že INET využívá stejného principu konfigurace simulace jako samotný OMNeT, tedy je možné využít služeb souboru omnetpp.ini, kde opět můžeme specifikovat příslušný XML soubor.

4.7 ANSA projekt

Jak již bylo zmíněno výše, jedná se o projekt vyvíjený na Fakultě Informačních Technologií VUT FIT. Jako startovní bod v tomto případě padla volba na již dříve popsaný INET Framework. ANSA tedy staví právě na jeho rozšiřování a zdokonalování. Výstupem tohoto snažení by měla být automatická formální analýza nad reálnými počítačovými sítěmi. Pomocí tohoto nástroje by se tedy měly zkoumat nejrůznější parametry, jako například dostupnost, nebo bezpečnost konkrétní sítě. V současnosti je projekt ve stavu dotváření co možná nejvíce realistických komponent, z kterých by pak šla rekonstruovat cílená síť. Tato práce si klade za úkol ve výsledku rozšířit právě výše zmíněnou automatizaci zpracování. Tedy v reálné situaci po připojení simulačního programu k požadované síti by s pomocí externího nástroje byl uživatel schopen ihned provádět experimenty. Externí program proto musí naplnit požadované soubory dynamickými daty ze sledovaného systému. V rámci této snahy byla brána v potaz i užitečnost rekonstruovat automaticky topologii fyzické sítě. Tento přístup má ovšem několik limitních faktorů, které následně popíši v dalších kapitolách, kde budeme rozebírat použité technologie k dosažení vytyčeného cíle.[1]

4.7.1 Aktuální podoba konfiguračního souboru

Jak již bylo zmíněno výše, cíleným způsobem konfigurace simulace je využití vstupního XML souboru. Pro něj je následně důležité stanovit konkrétní formát, aby jeho podoba byla všude jednotná. V tomto směru bylo tedy nutné projít aktuální implementace a sepsat jeho souhrnnou podobu. Níže popisuji současnou podobu konfiguračního souboru, kde první část odpovídá konfiguraci směrovače a za ní následuje sekce popisující přepínač. Výpis formátu XML uvádíme níže v přílohách.

Routers - tento element se v konfiguraci vyskytuje pouze jednou a zahrnuje všechny směrovače v topologii. Jednotliví následníci jsou definováni vlastním elementem, který specifikuje jejich konfiguraci.

- **Router** - zde se nachází nastavení konkrétního směrovače. Značka obsahuje také

atribut definující identifikátor zařízení, což obvykle bývá vybraná IP adresa prvku. Element může obsahovat různé následníky.

- **Hostname** - určuje jméno přiřazené prvku.
- **Interfaces** - z principu může být rozhraní na zařízení více, proto jsou jednotlivé popisy uzavřeny v tomto tagu.
  - **Interface** - tato značka zapouzdřuje následníky, kteří již specifikují konkrétní hodnoty. Kromě této funkce lze nastavit také atribut, který určuje jméno rozhraní. Pokud bude v elementech za jejich názvem následovat číslice 6, tak se jedná o technologii v kontextu protokolu IPv6.
    - **IP Address** - nastavuje IPv4 adresu rozhraní.
    - **Mask** - popisuje masku IPv4 rozhraní v tečkové notaci.
    - **Duplex** - může nabývat hodnot half, full, nebo auto.
    - **Speed** - představuje rychlost rozhraní. Případně může být nastaven na auto.
    - **OspfNetworkType** - značí typ sítě, jak byla přímo určena správcem zařízení.
    - **OspfCost** - cena linky v rámci OSPF procesu.
    - **OspfPriority** - priorita v kontextu OSPF protokolu.
    - **OspfHelloInterval** - rozmezí zasílání Hello paketů
    - **OspfDeadInterval** - doba po které je sousední směrovač považován za neschopný komunikovat pomocí OSPF.
    - **OspfRetransmissionInterval** - čas v sekundách po kterém dojde k znovu zaslání LSA paketu.
    - **OspfInterfaceTransmissionDelay** - specifikuje dobu přenosu LSU paketu po lince.
    - **OspfAuthenticationType** - typ autentizace dvou různých OSPF procesů.
    - **OspfAuthenticationKey** - klíč sloužící pro potřeby autentizace, pokud ale tato služba není nastavena, tak obsahuje hodnotu nula.
    - **OspfProcess6** - značí identifikátor IPv6 varianty OSPF směrovacího procesu.
    - **OspfArea6** - nastavení OSPF oblasti na konkrétním rozhraní.
    - **OspfInstace6** - počet instancí OSPF na lince.
    - **OspfNetworkType6** - nastavený typ sítě, jak jej vnímá směrovací proces.
    - **OspfCost6** - cena linky.
    - **OspfPriority6** - priorita linky.
    - **OspfHelloInterval6** - rozmezí zasílání Hello paketů
    - **OspfDeadInterval6** - doba po které je sousední směrovač považován za nedostupný.
    - **OspfRetransmissionInterval6** - časové rozmezí po kterém dojde k znovu zaslání LSA paketu.
    - **OspfInterfaceTransmitDelay6** - doba přenosu LSU paketu po lince.

        - **shutdown** - reflektuje administrativní stav rozhraní, tedy jestli je nastaveno jako funkční, nebo nefunkční.
        - **IPv6Address** - představuje dvojici oddělenou znakem /. Kde první je adresa rozhraní v IPv6 formátu a druhá část značí délku prefixu.
        - **NdpAdvSendAdvertisment** - nastavuje zda bylo povoleno zasílání paketů NDP protokolu.
        - **NdpAdvPrefix** - určuje prefix v NDP paketech. Přesněji nastavuje jak jeho formát tak délku.
        - **NdpMaxRtrAdvInterval** - značí maximální interval po kterém proběhne znovu zaslání NDP paketu.
        - **NdpMinRtrAdvInterval** - hodnota stejného významu jako u předchozí položky, pouze s tím rozdílem, že představuje spodní hranici.
        - **IS-IS-Priority** - jedná se o prioritu rozhraní v rámci IS-IS. Rozmezí hodnot se pohybuje od 0 do 127. Položka není povinná a výchozí nastavení je 64.
        - **IS-IS-Metric** - představuje metriku rozhraní pro IS-IS. Element může obsahovat hodnoty od 0 do 63, ale není povinný. Výchozí nastavení je 10.
        - **Pim** - obsahuje konfiguraci protokolu PIM pro konkrétní rozhraní. Ta je složena ze dvou elementů. Mode identifikuje o jaký typ módu na tomto rozhraní se jedná. Border indikuje hraniční směrovač dvou PIM domén.
- **DefaultRouter** - obsahuje IPv6 adresu výchozího směrovače v topologii.
- **Routing** - tento element obsahuje nastavení pro směrování v rámci protokolu IPv4. Jsou zde zahrnuty nejrůznější přístupy od statické varianty až po technologii EIGRP.
    * **Static** - představuje kolekci statických směrovacích informací v zařízení.
        - **Route** - reflektuje již konkrétní nastavení jedné cesty. Obsahuje následující elementy. NetworkAdress a NetworkMask, které ukládají adresu a masku cílové sítě. ExitInterface značí výstupní rozhraní v kontextu této cesty.
    * **IS-IS** - obsahuje konfiguraci protokolu IS-IS a slouží k jeho aktivaci na směrovači.
        - **is-type** - značí typ instance IS-IS, je to nepovinná položka, která může nabývat hodnot "level-1", "level-2", "level-1-2". Posledně jmenovaná je také výchozí, pokud není nic zadáno.
        - **net** - net adresa směrovače, která je povinná. Musí být specifikována ve formátu XX.XXXX.XXXX.XXXX.XXXX.XX.
    * **Multicast** - představuje část konfigurace určené pro multicast. Ten je povolen nastavení hodnoty 1 do atributu enable, který je v elementu obsažen.
        - **Pim** - obsahuje konfiguraci protokolu PIM. Ta zahrnuje celkem tři zanořené elementy. RPAddress slouží pro statické nastavení Rendezvous Point. BSRCandidate a RPCandidate jsou určeny pro využití Bootstrap mechanismu.
    * **Rip** - obsahuje konfiguraci protokolu RIP.
        - **Network** - IP adresa sítě propagované RIP procesem.

· **Passive-interface** - identifikace rozhraní kam se nemají posílat pakety RIP protokolu.

· **Redistribute** - představuje subsekci starající se o redistribuci do tohoto RIP procesu. Ta obsahuje následující elementy. Z jaké technologie jsou cesty brány, to značí tag Protocol. Dále metrika, kterou se tyto cesty budou ohodnocovat, o to se stará element Metric.

* **Ospf** - zde se nachází nastavení OSPF protokolu

· **RFC1583Compatible** - pouze konstatuje, že se jedná o formát kompatibilní podle RFC 1583.

· **Areas** - zahrnuje všechny OSPF oblasti v kterých je směrovač aktivní. Každá z nich je reprezentovaná tagem Area a atributem id, který ji jednoznačně identifikuje. Následně pod takovou oblastí je výčet elementů nastavujících propagované sítě. Jejich parametry jsou uzavřeny v značce Network a ta je potomek Networks. Jednotlivé sítě jsou specifikovány dvojicí IPAdress a Wildcard, kde první z nich představuje IP adresu a druhý masku v invertovaném formátu.

* **Eigrp** - značka reflektuje nastavení směrování pomocí EIGRP.

· **AutonomousSystem** - formát je téměř totožný jako u OSPF varianty. Pouze zde došlo k záměně názvů, jelikož protokol EIGRP nepracuje s oblastmi, ale s autonomním systémem. Jejich výčet je uzavřen v tomto tagu, kde jeho následníci jsou označení zkratkou AS a atributem id. Pod každým z těchto systémů pak najdeme množinu sítí reprezentovanou v elementu Networks. Konkrétní konfigurace se nachází v tagu Network a obsahuje IP adresu a invertovanou masku ve stejném formátu jako u OSPF.

– **Routing6** - zde se kumulují nastavení pro směrování v kontextu IPv6.

* **Static** - opět jako u IPv4 varianty představuje kolekci statických cest.

· **Route** - obsahuje konfiguraci jedné cesty. Ta je složená z síťové adresy, která je reprezentována elementem NetworkAdress, který má formát prefixu sítě a jeho délky. Následně je specifikován příští směrovač pomocí NextHopAddress a jako poslední je definována administrativní vzdálenost cesty. K tomu slouží tag AdministrativeDistance.

* **Ospf** - zde je konfigurace krátká díky IPv6 přístupu rozložit nastavení protokolu na jednotlivá rozhraní.

· **Process** - specifikuje aktivní OSPF proces, který je identifikován pomocí atributu id. V tomto prvku je vnořen identifikátor směrovače, který je obalen tagem RouterId.

– **ACLs** - zahrnuje výčet všech ACL nastavených v prvku.

* **ACL** - nastavuje konkrétní sadu pravidel, která je identifikovaná atributem no.

· **entry** - představuje konkrétní položku v jedné sadě pravidel. Jejich rozlišení probíhá pomocí atributu seq_no. Následnické elementy pak tvoří samotné pravidlo. Mezi ně patří action, kde se specifikuje, zda se má provoz povolit pomocí hodnoty permit, nebo zakázat pomoci deny. Dále se identifikuje zdrojová a cílová adresa provozu a jejich masky v invertovaném tvaru. Tyto elementy mají formát IP_src, IP_dst, WC_src a

WC_dst. Po nich následuje operátor porovnávání hodnot portů a jejich rozmezí. Tyto položky jsou nastaveny pomocí port_op_src, port_op_dst, port_beg_src, port_end_src, port_beg_dst, ports_end_dst. Operace mohou nabývat hodnot eq, neq, lt a gt. Není nutné nastavovat všechny položky, konfigurace může být variabilní. Její formát následuje pravidla používaná v Cisco zařízeních. Jako poslední dva parametry sem patří určení protokolu a originální textový zápis pravidla.

· **interfaces** - popisuje rozhraní na které jsou pravidla aplikována. Jelikož jich může být více, jsou uzavřeny v tomto elementu. Jméno rozhraní je pak určeno tagem interface, který má atribut dir, jež nastavuje směr filtrování.

Switch - tato značka má za úkol definovat jednu instanci přepínače v simulaci. Kromě svého obsahu následníků, kteří určují konfiguraci, má i atribut id. Ten je unikátní a identifikuje přepínač. Zbylé prvky nastavení popíšeme níže.

- **Hostname** - představuje stejnou položku jako u směrovače tedy přiřazené jméno prvku.
- **Interfaces** - kumuluje všechny rozhraní přepínače.
  - **Interface** - obsahuje konkrétní konfiguraci. Kromě ní ale také identifikátor definovaný atributem id.
    * **Name** - řeší pojmenování daného rozhraní.
    * **VLAN** - specifikuje přiřazení přijatých a neoznačených rámců na tomto rozhraní k VLAN. Provádí tedy jejich značení.
  - **VLANs** - zahrnuje definice VLAN.
    * **VLAN** - ohraničuje nastavení konkrétní VLAN, plus definuje identifikátor atributem id.
      · **Name** - nastavuje pojmenování VLAN. Typicky je to určeno pouze pro lepší čitelnost.
      · **Tagged** - identifikuje na kterých portech je tato VLAN povolena. Aplikuje se v procesu odesílání paketů.
      · **Untagged** - určuje kde se nemá přiřazovat příslušnost rámce k této VLAN. Opět se této informace využívá při odesílání.
      · **Nountagged** - slouží především k odebírání implicitního nastavení.
  - **STP** - sdružuje konfiguraci STP protokolu.
    * **Instance** - popisuje jednotlivé instance, které jsou rozlišené pomocí atributu id.
      · **BridgePriority** - priorita pro daný přepínač v STP instanci.
      · **PortPriority** - to stejné jako předchozí element, pouze se aplikuje na port. Kromě tohoto obsahuje i atribut id specifikující index portu.
      · **ForwardDelay** - určuje rychlost přechodů mezi stavy portu.
      · **MaxAge** - specifikuje maximální stáří informací v rámci této instance.
      · **HelloTimer** - definuje periodu odesílání Hello paketů STP protokolu.

· **LinkCost** - nastavuje cenu linky v procesu STP, obsahuje také atribut id pro určení indexu portu.

Host - jedná se o označení konfigurace náležící stanici.

- **Interfaces** - ohraničuje množinu rozhraní stanice.
  - **Interface** - obsahuje konkrétní konfiguraci rozhraní plus také identifikátor definovaný atributem name.
    * **IPAddress** - obsahuje IPv4 adresu.
    * **Mask** - specifikuje masku platnou pro předchozí IPv4 adresu.
    * **IPv6Address** - zahrnuje adresu ve formátu IPv6 včetně masky. To je nutné vzhledem k možnosti existence více těchto adres na jednom rozhraní a tím vzniklé nutnosti s nimi provázat masku.
- **DefaultRouter** - značí výchozí bránu pro tento prvek.

Kapitola 5

Rekonstrukce topologie

Jednou z funkcí vyvíjeného nástroje má být schopnost rozpoznat a rekonstruovat topologii, proto se v následujících řádcích budeme zaobírat touto problematikou. Nejdříve shrneme aktuálně dostupné přístupy rekonstrukce topologie a pak uvedeme, pro kterou variantu jsme se rozhodli. Výsledný produkt této diplomové práce by měl být pokud možno co nejvíce kompatibilní s různorodým prostředím počítačových sítí, proto je tu tendence se spíše orientovat na otevřené standardy, než proprietární technologie.

5.1 Přístupy k rekonstrukci topologie

5.1.1 ICMP protokol

Tato technologie je velice známá v současném prostředí počítačových sítí, proto můžeme také očekávat její širokou podporu, což je jedna z podmínek úspěšného nasazení takového přístupu. Princip činnosti je postaven na schopnosti měřit čas podniknuté cesty odeslaného paketu. Problém ovšem nastává, když si uvědomíme, že toto zpoždění není jen závislé na délce linky a její rychlosti, ale také na vytížení sítě a zařízeních samotných. Sem patří především doba nutná ke zpracování paketu, které je opět odvislá jak od schopností prvku a od jeho vytížení. Kromě těchto vlastností je nevýhodou také prioritizace zpracovávání síťového provozu, kde mohou být obslouženy nejdříve důležitější data. Proto se také vždy nedá spolehnout na hodnoty naměřeného zpoždění. Pokud jsme se předtím zmínili o nutnosti podpory tohoto protokolu, vyvstává pak otázka zda nebude docházet k filtraci těchto paketů na jednotlivých prvcích. K tomu může docházet jednoduše z bezpečnostních důvodů a mnoho produkčních sítí to tímto způsobem má nastaveno. S přihlédnutím na tyto vlastnosti jsme se rozhodli tento přístup zavrhnout a věnovat se následujícím.

5.1.2 Spanning-tree protokol

Pokud se pohybujeme na úrovní linkové vrstvy, tak je nutné zajistit její bezsmyčkovost, což by mohlo vést k neúměrnému vytěžování linky pomocí všesměrových bouří a přeplnění MAC tabulek daných zařízení. Spannig-tree protokol tento problém řeší pomocí jednoznačné identifikace jednotlivých prvků v jednom síťovém segmentu na úrovni druhé vrstvy. Ke své činnosti využívá především unikátní identifikátorů zařízení, portů a také ceny přiřazené každému rozhraní. Jakmile se tedy síť dopracuje do stabilního stavu, tak jíž aktivně využívané linky v topologii netvoří graf, ale strom, kde je z principu zajištěna bezsmyčkovost. Z těchto faktů můžeme něco vytěžit pro disciplínu prozkoumávání topologie. Podmínkou

je ovšem podpora této technologie a přístupnost pracovní databáze protokolu. Nyní ale narážíme na hlavní nevýhodu tohoto přístupu, že jej lze aplikovat pouze na část sítě, která pracuje na druhé vrstvě. Nejsme tedy schopni monitorovat topologii jako celek. Na druhou stranu poměrně podstatná výhoda je značná aktuálnost obdržených dat, na kterou jsou kladeny vysoké požadavku už jen kvůli tomu, že na nich záleží provozuschopnost sítě. Nicméně vzhledem k nemožnosti použít tento přístup v globálním měřítku jsme pokračovali dále v hledání.

5.1.3 Forwarding databáze

Popisovaná technologie je oproti předchozí široce podporována a využívána. Jedná se jinými slovy o tabulku MAC adres, která je různě doplňována a aktualizována podle momentálního síťového provozu. Z tohoto ovšem vyplývá několik otázek. Za prvé, je možné že některé prvky nebudou v tabulce obsaženy jednoduše proto, že zatím neodeslaly žádný paket, nebo v opačném případě byly dostatečně dlouho neaktivní. Tento interval je standardně nastaven na 300 sekund. Po tuto dobu je pak možné, že data, která získáme, mohou být neaktuální. Jedna možnost jak se vyhnout těmto nepříjemnostem je generování ICMP provozu, který bude postupně data aktualizovat. Nevýhoda takového přístupu pak leží v nadměrném zatížení sítě, které je závislé na jejím návrhu. Dále není zajištěna opět kompletnost získaných informací, jelikož některé prvky mohou tyto pakety filtrovat, jak již bylo zmíněno výše. Obecně čím roste velikost sítě, tak se tyto problémy více projevují. Z těchto důvodů se zdá takovýto zdroj informací jako nespolehlivý, proto jsme se rozhodli jej nevyužít.

5.1.4 Physical Topology MIB

V průběhu let se objevily snahy standardizovat přístup k monitorování topologie. Výsledkem je také tato specializovaná MIB, která se přesně nazývá PTOPO-MIB podle RFC 2922. Standard byl vyvinut společnostmi Cisco a Nortel Networks v roce 2000. Cílem bylo definovat jak reprezentovat topologii v podstromu MIB. Bohužel v rámci těchto snah již nevznikl protokol pro výměnu těchto informací mezi prvky. Proto pravděpodobně nedošlo k rozšíření technologie a tím pádem je tato struktura pro naše účely nepoužitelná.[12]

5.1.5 Směrovací tabulka

Prozkoumávat síťovou topologii je možné i za pomocí tohoto přístupu. Struktura je standardně přístupná přes protokol SNMP a každá směrovaná síť obsahuje dodatečné informace, které jsou nápomocné k rekonstrukci. Mezi tyto data patří například zda je síť přímo připojená, případně zda byla naučena nějakým směrovacím protokolem. Tabulku můžeme aktuálně ve standardních verzích MIB nalézt ve dvou verzích, které jsou různě podporovány podle stáří monitorovaného zařízení. První z nich se nazývá ipRouteTable a je dostupná přes OID 1.3.6.1.2.1.4.21.1. Tato verze je považována podle dokumentace za již zastaralou a proto je doporučováno použít následující. Novější varianta je pojmenována jako ipAddressTable a přístup zajišťuje OID 1.3.6.1.2.1.4.34.1. Bohužel při testování dostupnosti těchto struktur jsme narazili na běžnější podporu starší verze, proto ji nelze opomenout.Hlavním problémem popisovaného přístupu je ale podpora pouze prvků na síťové úrovní topologie. Je jasné, že přepínače tímto způsobem nelze monitorovat, proto jsme se přesunuli k finálnímu způsobu rekonstrukce, který popíšeme níže.

5.1.6 Network Discovery protokoly

Tato třída protokolů, jejichž dva nejvýznačnější zástupci jsou popsání již v kapitole 2, slouží již od doby vzniku k prozkoumávání sítě jako takové. Nedílnou součástí jejich návrhu tedy často bývá dostupnost jejich dat přes některou managementovou technologii. Mezi další charakteristiky patří, že prvky si vyměňují informace mezi sebou, tedy přesněji řečeno jsou obeznámeni pouze o existenci svých sousedů. Komunikace tedy probíhá nejčastěji na druhé vrstvě a to vícesměrovou formou. Tyto vlastnosti tvoří z této sorty technologií ideální kandidáty na použití v našem algoritmu. Je tedy pravda, že mezi nevýhody lze počítat nutnost jejich podpory na cílových zařízeních. Nicméně z praktického pohledu je cílem našeho nástroje síť VUT, kde je standardně povolen protokol LLDP a jelikož se jedná o prvky od společnosti HP, lze očekávat i schopnost komunikovat s technologií CDP, kterou má licencovanou. Z těchto důvodů a vzhledem k volbě algoritmu, který popíši na následujících řádcích, jsme se rozhodli uplatnit čistě tento přístup.

Po proběhlé volbě přístupu je také potřeba připomenout nutné podmínky, které musí sledovaná síť splňovat, aby na ní bylo možné rozumným způsobem provádět zmíněnou rekonstrukci. První z nich je právě podpora SNMP protokolu a s tím související nutnost poskytnout validní množinu přístupových informací. Kromě těchto dat je nutné aby všechny prvky, které mají být nalezeny, byly dosažitelné ze sledujícího zařízení a měli konektivitu na úrovni třetí vrstvy, tedy nastavenou IP adresu alespoň pro účely správy.

5.2 Tvorba algoritmu

Nyní když byl vybrán technologický přístup pro rekonstrukci topologie je potřeba vytvořit také samotný algoritmus, který tuto technologii bude obsluhovat. Při jeho výběru jsme brali v potaz kritérium rozumné časové složitosti, která by vzhledem k realizované úloze měla být co nejmenší. V této části popíši zvolený algoritmus a důkaz jeho korektnosti. Následně uvedu moji modifikaci a její testování na reálné topologii.

Jako předloha pro princip algoritmu bylo použito prohledávání do šířky. Tato varianta se nabízí z principu pojetí programu, kdy jako vstup slouží prvek v cílové topologii. Od tohoto bodu se pak nástroj dotazuje na okolní zařízení pomocí SNMP technologie. Jelikož záznamy informací o okolí dosahují pouze k nejbližším sousedům je nutné se mezi uzly sítě přesouvat a tím tedy realizovat podstatu procházení grafu. Protože vstupní bod je pouze jeden a naším cílem je analyzovat celý graf, tak se jedná o variantu vyhledávání cest z jednoho bodu do všech ostatních. Časová složitost prohledávání do šířky je $O(m+n)$ kde m značí počet hran grafu a n počet uzlů. Nyní představím stručně princip prohledávání do šířky a použité datové struktury.

- **Pole** $Adj[u]$ - obsahuje všechny bezprostředně dosažitelné sousedy uzlu u.
- **Pole** $\pi[u]$ - reprezentuje předchůdce uzlu u.
- **Uzel** s - jedná se o startovní uzel prohledávání.
- **Pole** $d[u]$ - vrací vzdálenost ze startovního uzlu do u.
- **Fronta** Q - představuje hlavní prvek určující koncept algoritmu. Konkrétně pracujeme s typem FIFO. Fronta obsahuje uzly určené ke zpracování.

- **Pole** $color[u] \in \{White, Gray, Black\}$ - tyto tři barvy reflektují aktuální stav zpracování uzlu v rámci algoritmu. Pokud se jedná o bílou barvu, tak ještě nedošlo k navštívení, pokud o šedou tak uzel byl již navštíven, ale nebyli objeveni všichni jeho sousedi a černá značí uzavřenost, tedy algoritmus již dokončil zpracování uzlu a již se k němu nevrací.

Samotný průběh lze popsat následovně. Nejprve proběhne inicializace grafu. Zde se nastaví pro všechny uzly, vyjma startovního, hodnoty pole *color* na bílou, dále vzdálenosti d na ∞ a předchůdci π na NIL. Počáteční uzel z něhož startuje procházení je označen šedou barvou, vzdálenost nastavena na 0 a jeho předchůdce na NIL. V dalším kroku dojde k inicializaci fronty Q a vložení startovního uzlu do ní. Nyní vstupujeme do smyčky, která se opakuje dokud není fronta Q prázdná. V ní dojde k vyjmutí aktuálního prvku u z Q a pro všechny jeho sousedy v z pole Adj, pokud jsou bílé, dojde k změně barvy na šedou, nastavení vzdálenosti na $d[u] + 1$ a přiřazení předchůdce u. Jako poslední krok je uzel v vložen do fronty Q. Jakmile je takto zpracován každý soused v je obarvení jejich předchůdce u změněno na černou. Níže uvádíme algoritmus popsaný pomocí pseudokódu.

Algorithm 1 Prohledávání do šířky

```
function BFS(G, s)
    for ∀u ∈ V − s do
        color[u] ← White
        d[u] ← ∞
        π[u] ← NIL
    end for
    color[s] ← Gray
    d[s] ← 0
    π[s] ← NIL
    Q ← ∅
    Enqueue(Q, s)
    while Q ≠ ∅ do
        u ← Dequeue(Q)
        Adj[u] ← u
        for ∀v ∈ Adj[u] do
            if color[v] = White then
                color[v] ← Gray
                d[v] ← d[u] + 1
                π[v] ← u
                Enqueue(Q, v)
            end if
        end for
        color[u] ← Black
    end while
end function
```

5.2.1 Důkaz korektnosti prohledávání do šířky

Nyní po stručném uvedení do funkce algoritmu můžeme představit důkaz jeho korektnosti.[6]

Lemma 5.2.1. *Nechť $G = (V, E)$ je orientovaný nebo neorientovaný graf a nechť $s \in V$ představuje libovolný uzel. Pak pro každou hranu $(u, v) \in E$ platí následující nerovnost $\delta(s, v) \leq \delta(s, u) + 1$.*

Důkaz. Jestliže je uzel u dosažitelný z s, pak je rovněž uzel v dosažitelný z s.

- Odtud tedy dostaneme, že nejkratší cesta z s do v nemůže být delší než nejkratší cesta z s do u následovaná hranou (u, v). Nerovnost proto platí.
- Pokud uzel u není dosažitelný z s, pak $\delta(s, u) = \infty$ tedy nerovnost také platí.

□

Lemma 5.2.2. *Nechť $G = (V, E)$ je orientovaný nebo neorientovaný graf a prohledávání do šířky je spuštěno na G z počátečního uzlu $s \in V$. Pak následně po ukončení algoritmu platí, že $d[v] \geq \delta(s, v)$ pro každé $v \in V$.*

Důkaz. Dokážeme indukcí vzhledem k počtu Enqueue operací na frontě Q. $IPjed[v] \geq \delta(s, v)$ pro všechna $v \in V$.

- Báze: $d[s] = 0, \delta(s, s) = 0$ a $d[v] = \infty \geq \delta(s, v)$, $v \in V - s$.
- Nechť v je bílý uzel prozkoumaný během prohledávání z u. Z indukčního předpokladu získáme $d[u] \geq \delta(s, u)$.
- Dále dostaneme z algoritmu prohledávání do šířky, indukčního předpokladu a Lemma 5.2.1 : $d[v] = d[u] + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$.
- Uzel v je poté vložen do fronty a tako operace se na něm již neopakuje, jelikož má již šedou barvu a tedy nevyhovuje podmínce v algoritmu, proto hodnota $d[v]$ je již finální.

□

Lemma 5.2.3. *Nechť během průchodu algoritmu prohledávání do šířky grafem $G = (V, E)$ obsahuje fronta Q uzly $< v_1, v_2, ..., v_r >$, kde v_1 je první prvek Q a v_r je poslední prvek Q. Pak platí, že $d[v_r] \leq d[v_1] + 1ad[v_i] \leq d[v_{i+1}]$ pro $i = 1, 2, ..., r - 1$.*

Důkaz. Důkaz povedeme indukcí vzhledem k počtu operací fronty. Na počátku je stav fronty $Q =< s >$ a tvrzení tedy platí.

- Tvrzení následně platí i po obou provedených frontových operacích:
  - Pokud je v_1 odebráno, pak v_2 představuje nový první prvek. V případě že by došlo k vyprázdnění fronty, tvrzení by pak platilo triviálně. Z indukčního předpokladu pak získáme $d[v_1] \leq d[v_2]$.
  - Pak ale platí $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$, kde zbytek nerovností zůstane nezměněn.
  - Jakmile dojde k vložení v_{r+1} do fronty Q, pak v ten okamžik je již z Q vyjmut uzel u, jehož seznam procházíme. Z indukčního předpokladu následně dostaneme nerovnost $d[u] \leq d[v_1]$.
  - Tedy ve výsledku platí $d[v_{r+1}] = d[u] + 1 \leq d[v_1] + 1$, z čehož plyne $d[v_r] \leq_{IP} d[u] + 1 = d[v_{r+1}]$. Zbytek nerovností zůstane nezměněn.

□

Důsledek 5.2.4. *Nechť uzly v_i a v_j jsou vloženy do fronty během provádění algoritmu prohledávání do šířky a v_i je vloženo před v_j. Pak platí $d[v_i] \leq d[v_j]$ v okamžiku vložení v_j do fronty.*

Důkaz. Plyne z předchozího Lemmatu 5.2.3 a vlastnosti algoritmu, že každý uzel dostane finální hodnotu vzdálenosti d maximálně jednou během výpočtu. □

Teorém 5.2.5. *Nechť $G = (V, E)$ je orientovaný nebo neorientovaný graf a $s \in V$. Pak $BFS(G, s)$ prozkoumá všechny uzly $v \in V$ dosažitelné z s a po ukončení platí, že $d[v] = \delta(s, v)$ tedy nejkratší cestě v grafu G pro všechna $v \in V$. Pro každý uzel $v \neq s$ dosažitelný z s dále platí, že jedna z nejkratších cest z s do v je také nejkratší cesta z s do $\pi[v]$ následovaná hranou $(\pi[v], v)$.*

Důkaz. Dokážeme sporem. Nechť v je uzel s minimální $\delta(s, v)$ takový, že $d[v] \neq \delta(s, v)$. Pak $v \neq s$.

- Z Lemma 3 získáme, že $d[v] \geq \delta(s, v)$, proto $d[v] > \delta(s, v)$. Dále v musí být dosažitelné z s, jelikož jinak platí $\delta(s, v) = \infty \geq d[v]$.
- Nechť u je uzel bezprostředně předcházející v na nejkratší cestě z s do v, tedy platí $\delta(s, v) = \delta(s, u) + 1$.
- Vzhledem k platnosti $\delta(s, u) < \delta(s, v)$ a volbě v, pak platí $d[u] = \delta(s, u)$. Vyjde nám tedy následující rovnost $d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1$.
- Sledujeme-li BFS v době, kdy u je vybíráno z Q, tedy v se nachází ve stavu bud' bílý, šedý, nebo černý. Pak prozkoumáme následující možnosti.
- v je bílý, pak algoritmus nastavuje $d[v] = d[u] + 1$, čímž vznikne spor.
- v je černý, pak v již bylo vybráno z Q a podle důsledku 5.2.4 je $d[v] \leq d[u]$, čímž vznikne spor.
- v je šedý, pak v bylo obarveno při výběru nějakého uzlu w, jenž byl vybrán z Q dříve než u. Navíc, $d[v] = d[w] + 1$. Podle důsledku 5.2.4 je $d[w] \leq d[u]$, tj. $d[v] \leq d[u] + 1$, čímž vznikne spor.
- Tedy platí rovnost $d[v] = \delta(s, v)$ pro všechna $v \in V$, jelikož všechny uzly dosažitelné z s musí být prozkoumány, jinak by byla jejich vzdálenost $d = \infty$.
- Protože platí $\pi[v] = u$, pak tedy $d[v] = d[u] + 1$, ve výsledku můžeme získat nejkratší cestu z s do v přidáním hrany $(p[v], v)$ k nejkratší cestě z s do $p[v]$.

□

5.2.2 Modifikovaný algoritmus prohledávání do šířky

Nyní jsme shrnuli teoretickou podstatu algoritmu, je možné přikročit k jeho popisu. Princip činnosti je totožný s prohledáváním do šířky, nicméně při nasazení do reálného prostředí je nutné provést několik změn. Například není možné provést inicializaci každého uzlu v topologii, z jednoduchého principu že ji neznáme. Z tohoto důvodu jsme v implementaci

nezahrnuli bílou a šedou barvu. Tuto jejich činnost supluje unikátní identifikátor zařízení v síti. Hlavní změnou oproti prohledávání do šířky je také absence pole Adj, které se při běhu naopak vytváří. Níže popíšeme datové struktury nutné k běhu modifikovaného algoritmu. Zmíníme ovšem pouze změny. Je důležité uvést, že označení každého uzlu je realizováno unikátním identifikátorem.

- **Uzel** $root$ - unikátní identifikace počátečního uzlu.
- **Pole** $color[u] \in \{NIL, Black\}$ - má stejnou funkci jako u původního algoritmu.

Dále zobrazíme výslednou podobu algoritmu. Níže uvedené funkce $getUID$ a $getNeighbors$ jsou realizovány pomocí SNMP komunikace dotazující se na konkrétní prvek v síti. Dále $Enqueue$ a $Dequeue$ slouží pro vkládání a vybírání z fronty.

Algorithm 2 Modifikované prohledávání do šířky

```
function ModBFS(root)
    Adj ← getUID(root)
    Enqueue(Q, root)
    while Q ≠ ∅ do
        u ← Dequeue(Q)
        Adj ← u
        neighbors ← getNeighbors(u)
        for ∀v ∈ neighbors do
            neighbor ← getUID(v)
            if color[neighbor] = Black ∧ neighbor ≠ root then
                Adj[u] ← neighbor
                continue
            end if
            if neighbor ≠ root then
                Enqueue(Q, neighbor)
            end if
            Adj[u] ← neighbor
        end for
        color[u] ← Black
    end while
end function
```

Když se podíváme blíže na samotný průběh výpočtu, lze jej popsat následovně. Algoritmu je předložen startovní uzel, který je zde pojmenován jako $root$. Ten je v současnosti identifikován pomocí IP adresy, proto je potřeba získat jednoznačnější pojmenování, které by mělo být unikátní v rámci prozkoumávané sítě. Pro tento účel jsme zvolili kombinaci fyzických adres jednotlivých rozhraní na prvku, tento přístup funguje velmi dobře pokud se pohybujeme v technologii Ethernet. Problém ovšem nastane jakmile narazíme například na sériový interface, který takovouto informaci nenese. V tomto případě je pak nutné identifikátor doplnit další položkou. Zde jsme se rozhodli pro sériové číslo šasi prvku. Kombinace těchto elementu se jeví již jako dostatečně unikátní, další výhodou je, že všechny tyto informace jsou dosažitelné prostřednictvím standardizovaných MIB, tudíž je jejich dostupnost zaručena všemi prvky korektně implementujícími standard.

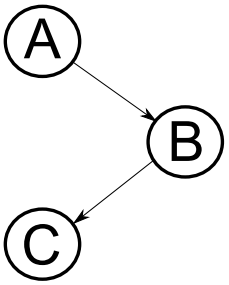
Nyní jakmile je počátečnímu uzlu přiřazen unikátní identifikátor, je možné jej přiřadit do dvoudimenzionálního pole sousedností, kde do korektně asociované dimenze budou přibývat

sousedé, kteří byli objeveni algoritmem. Protože počáteční uzel není v tomto bodě ještě kompletně prozkoumaný, je nutné jej přidat do fronty pro zpracování.

Po tomto úvodu vstupujeme do hlavní smyčky, kde probíhá celé jádro algoritmu. Cyklus se opakuje dokud není fronta Q prázdná. V každé iteraci dojde nejprve k vyjmutí nejstaršího prvku u ve dříve zmíněné frontě. Tento element reprezentovaný již unikátním identifikátorem je přiřazen do pole sousedností a je mu tedy vytvořena příslušná dimenze. Po této akci jsou prozkoumání sousedi tohoto prvku a jejich výčet je vložen do struktury $neighbors$.

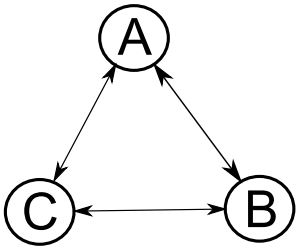
Zde jsme již dospěli do bodu kdy se začne vykonávat vnitřní smyčka For. Ta má za úkol projít postupně všechny sousedy prozkoumávaného uzlu u. V první řadě je každému z nich přirazen opět unikátní identifikátor. Následně dojde k testu zda tento element již byl dříve zpracován. Operace je realizována pomocí pole $color$, které je jako všechny struktury indexováno zmíněným ID elementu. Proběhne tedy test na černou barvu a současně se ověří zda jsme se zpětně nedostali k počátečnímu uzlu. To se může snadno stát, protože celá sít v této technologii je pojata jako neorientovaný graf.

Pokud jsou dříve zmíněné podmínky splněny, tak přiřadíme souseda do odpovídajícího pole $Adj[u]$ a zbytek iterace smyčky již přeskočíme, jelikož prvek má již svojí dimenzi v Adj přiřazenu a jediné, co je nutné, aby reprezentace grafu byla kompletní, tak dodat spojnici mezi těmito uzly. Popisovaná operace je nutné z principu funkce struktury, kterou interně reprezentujeme náš graf. Jak už bylo zmíněno jedná se o pole sousedností a to standardně popisuje orientované grafy. Nicméně pokud definujeme propoje oběma směry, můžeme získat i reprezentaci grafu neorientovaného. Pro snazší pochopení níže uvádíme několik příkladů a k nim odpovídající pole sousedností.



Tabulka 5.1: Příklad topologie 1

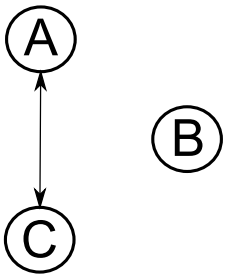
```
'A': ['B']
'B': ['C']
'C': []
```



Tabulka 5.2: Příklad topologie 2

```
'A': ['B','C']
'B': ['A','C']
'C': ['B','A']
```

Tabulka 5.3: Příklad topologie 3



'A': ['B']
'B': ['A']
'C': []

V opačné případě rozhodnutí struktury If dojde k dalšímu testu, kde se porovnává zda uzel není počáteční. V takovém případě je provedena operace vložení souseda u do fronty Q. Tentýž uzel je pak zpracován ve stejném kontextu, ale v rámci jiné struktury Adj. Jakmile jsou všechny elementy zpracovány, tak dojde k opuštění smyčky For a prozkoumávaný uzel u je označen černou barvou a tím již vyloučen z dalšího zpracovávání algoritmem.

Pokud prozkoumáme tělo smyček ještě jednou, můžeme identifikovat několik typů elementů a jejich specifickou cestu kódem.

- **počáteční uzel** - v průchodu nesplňuje ani jednu podmínku, proto je mu přiřazena pouze operace vložení do pole sousedností pro kompletaci neorientovaného grafu.
- **černý uzel** - pokud se současně nejedná o *root*, tak projde pouze první podmínkou, tedy dojde pouze ke kompletaci Adj.
- **neprozkoumaný uzel** - nesmí být černý, ani počáteční. V tomto případě je zařazen do fronty Q pro další zpracování a následně vložen do Adj.

5.3 Testování

Jakmile byl vytvořen teoretický návrh algoritmu, bylo možné přikročit k implementační části proof of concept aplikace, která měla za úkol ověřit jeho funkčnost. Nejdříve probíhalo ladění samotné implementace k čemuž jsme využili prostředí emulátoru GNS 3, který je schopen relativně věrně napodobit sítovou komunikaci na třetí úrovni a vyšších. Jelikož celou alfou a omegou tohoto nástroje je komunikace prostřednictvím protokolu SNMP, tak jsou tyto schopnosti dostačující. Kromě těchto věcí bylo ale třeba také zajistit podporu pro aplikační vrstvu, jelikož na ní se ve finále vytvářený program nachází, zde přišla vhod speciální edice tohoto emulátoru, která je schopná nativně zakomponovat podporu Virtual box klientů. Jinými slovy je možné připojit zařízení, které obsahuje libovolný operační systém podporovaný tímto prostředím.

V rámci ladění bylo nutné se také seznámit se způsobem nastavení protokolů nutných k běhu nástroje. Jelikož v prostředí simulátoru byla nejdostupnější platforma Cisco, soustředili jsme se právě na ni. Níže uvádíme použité konfigurační přístupy podle cílených protokolů.

5.3.1 Konfigurace CDP

V tomto případě není v drtivé většině nutné nastavovat nic, jelikož protokol je standardně povolený na majoritě Cisco zařízení. Co se týká jeho podpory, tak není překvapením, že je na mateřských prvcích implementován téměř všude, proto představuje poměrně kvalitní zdroj informací minimálně pod tímto výrobcem. Pro úplnost dále uvádíme nejdůležitější konfigurační příkazy s adekvátním komentářem.[2]

- **cdp run** - příkaz se zadává v konfiguračním modu prvku, slouží pro povolení protokolu.
- **show cdp neighbors** - lze použít v privilegovaném režimu a jedná se o nejpoužívanější výstup této technologie. V komprimované formě zobrazí detekované sousedy, kteří mají také povolené CDP.
- **show cdp neighbors detail** - je velmi podobný předchozí variantě s tím rozdílem, že zobrazuje všechny dostupné informace o svých sousedech.
- **show cdp entry** [] - pomocí tohoto příkazu je možné zobrazit data o konkrétním prvku v topologii, samozřejmě s omezení na dosah technologie jako takové.
- **no cdp enable** - konfigurace probíhá v rámci jednoho rozhraní, kde je možné zakázat příjem a vysílání rámců této technologie.

5.3.2 Konfigurace LLDP

Protokol LLDP je standardně na prvcích Cisco zakázán, nicméně pokud dojde k jeho povolení tak již základní konfiguraci obsahuje. Větší problém ale nastává pokud kontrolujeme jeho podporu napříč spektrem produktů firmy. Z produktových stránek společnosti lze zjistit, že podpora je zamýšlena pouze na přepínače, tedy prvky s označením Catalyst. Politika společnosti, je tedy pravděpodobně mířena na primární používání CDP a implementace LLDP by měla sloužit pouze tam, kde již CDP nedostačuje, tedy na místě přepínače a s ním spjatého koncového zařízení. Tohle nicméně produkuje následující problém, že není možné provádět emulaci těchto zařízení. Jejich specifikum je že princip je založený na obvodech s názvem ASIC, které představují jádro pro přepínací mechanizmy. Proto přístup tohoto algoritmu prostřednictvím technologie LLDP byl ověřen až v reálné topologii. Dále uvádíme několik nejpodstatnějších příkazů.[8]

- **lldp run** - příkaz se zadává v konfiguračním režimu a slouží pro povolení protokolu.
- **show lldp neighbors** - stejně jako u protějška CDP, slouží k výpisu stručných dat o aktuálně aktivních sousedech.
- **show lldp neighbors detail** - představuje detailnější variantu předchozího příkazu.
- **show lldp entry** [] - realizuje nejpodrobnější možný výpis informací o konkrétním prvku v topologii, který je znám tomuto zařízení.
- **show lldp traffic** - v rámci LLDP probíhá i monitorování provozu, který protokolem vzniká. Pomocí tohoto příkazu je možné si tyto statistiky zobrazit.
- **lldp transmit/receive** - nastavení probíhá na konkrétním síťovém rozhraní, lze zde velmi granulárně povolit, zda se mají data přijímat a odesílat, nebo nekomunikovat vůbec, případně jakákoliv kombinovaná varianta, stačí pouze doplnit klíčové slovo no.

5.3.3 Konfigurace SNMP

Nastavení protokolu je v rámci zařízení Cisco poměrně rozsáhle. Konkrétně je možné nastavit nejen SNMP Agenta, ale i druhou stranu, tedy Managera, který je schopen přijímat zprávy typu Trap. K našemu účelu bohatě stačí nastavení pro Agenta, proto se budeme soustředit pouze na něj. Pokud se zaměříme na podporu, tak se můžeme spolehnout i na třetí verzi technologie včetně nastavování různých pohledů na MIB a uživatelských skupin a účtů. Níže uvádíme konfigurace pro nejdůležitější verze protokolu.[14]

SNMP verze 1/2c

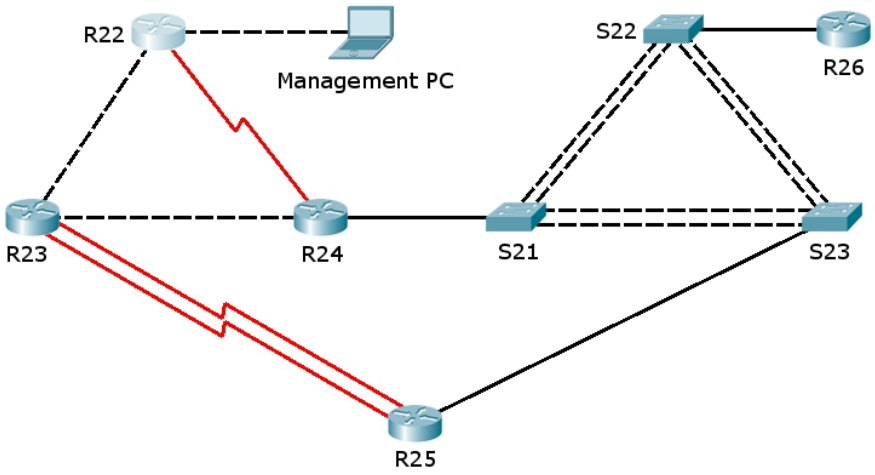
- **snmp-server community** - tento příkaz stačí pro zpřístupnění SNMP MIB okolnímu světu prostřednictvím znalosti community stringu. Toto zabezpečení je společné pro verzi 1 i 2c. Kromě tohoto můžeme nastavit i oprávnění pro konkrétní řetězec, tedy zda nám tato znalost umožní pouze číst, nebo i zapisovat. Dále můžeme doplnit omezení podle definovaného pohledu na MIB a v závěru také přiřadit konkrétní ACL.
- **snmp-server user** - je možné také definovat uživatelské účty, které nejsou aplikovatelné pouze na nejnovější verzi SNMP, ale lze je použít i v kontextu těch starších. Kromě jména uživatele ho můžeme přiřadit ke skupině a přiřadit ACL.
- **snmp-server group** - pro jednotlivé uživatele je možné vytvořit skupiny, které je sdružují a s tím ulehčují správu zabezpečení. To je realizováno mapováním skupin k pohledům na MIB.
- **show snmp** - příkaz slouží k vypsání statusu SNMP implementace na prvku.
- **show snmp user** - zobrazí informace o uživatelích definovaných na zařízení.
- **no snmp-server** - tímto způsobem je možné zakázat funkci SNMP Agenta.

SNMP verze 3

- **snmp-server user** - příkaz má stejnou funkci jako u starších verzí, rozdíl ale nastane v případě že umožníme autentizaci uživatelů, potom lze nastavit heslo a metodu, která se má při autentizaci použít.
- **snmp-server group** - opět příkaz disponuje stejnými schopnostmi jako u předešlých verzí, nová je možnost definice úrovně zabezpečení pomocí tří klíčových slov auth, noauth, priv. Zde jednotlivé pojmy definují použité technologie. Pro případ noauth se nepoužije nic, pro auth HMAC-MD5-96 nebo HMAC-SHA-96 a priv využívá CBC-DES. Pokud chceme použít tato rozšíření, které sebou nese třetí verze, tak je nutné uvést klíčové slovo v3. To platí i pro předcházející příkaz
- **show snmp** - pomocí tohoto příkazu je možné vypsat status SNMP implementace na prvku.
- **snmp-server view** - umožnuje definovat pohledy na MIB pomocí povolených podstromů. Budujeme tedy určitou množinu, která má funkci filtru a její chování lze rozdělit podle přepínačů included, excluded. První představuje inkluzivní přístup a druhý exkluzivní.
- **snmp-server engineID** - v rámci třetí verze protokolu je možné nastavit engineID, které je jinak generováno automaticky systémem.

5.3.4 Ověření funkce na laboratorní topologii

Za pomocí těchto nástrojů proběhlo prvotní úspěšné testování. Nyní jsme tedy mohli přikročit k ověření funkčnosti na reálně topologii. Zde jsme využili možností sítové laboratoře na fakultě VUT FIT, která se skládá převážně ze zařízení společnosti Cisco. Algoritmus byl testován na laboratorní topologii složené z několika Cisco směrovačů a přepínačů, které měly mezi sebou plnou konektivitu. Je třeba zmínit, že přepínače měli nastavené adresy třetí úrovně pomocí VLAN určených pro správu, protože je nutné splnit podmínku aby všechny prvky topologie měli IP konektivitu. Test proběhl úspěšně, níže uvádíme grafický nákres a textový výstup nástroje.



Obrázek 5.1: Testovaná topologie

Listing 5.1: Výsledky

```
'S22': ['R26', 'S23', 'S23', 'S21', 'S21']
'S23': ['R25', 'S21', 'S21', 'S22', 'S22']
'S21': ['R24', 'S23', 'S23', 'S22', 'S22']
'R26': ['S22']
'R25': ['R23', 'R23', 'S23']
'R24': ['R23', 'R22', 'S21']
'R23': ['R25', 'R25', 'R24', 'R22']
'R22': ['R23', 'R24']
```

Kapitola 6

Popis implementace

V této kapitole popíšeme techniky, které byly využity při vývoji nástroje. Nejdříve se zaměříme na volbu SNMP API, kde rozebereme různé dostupné implementace a postupně zhodnotíme jejich plusy a mínusy. Na závěr pak vybereme tu nejvhodnější. Dále se pak zaměříme na strukturu samotného nástroje, kde postupně popíšeme jednotlivé třídy tvořící výslednou funkcionalitu a jejich propojení. V tomto kontextu také zmíníme pouze podstatné implementační detaily.

6.1 Volba SNMP API

V produkčním prostředí existuje velké množství projektů, které se snaží různým způsobem ulehčit využívání technologie SNMP. Ta samotná je poměrně uspokojivě navržena, ale její implementace má různá specifika, které je nutné vzít v potaz. Pokud chceme vytvořit nástroj, který bude fungovat úspěšně v širokém spektru zařízení, které dnes podporují SNMP, je nezbytné se zabývat i velkým množstvím detailů. Z toho vyplývá, že implementace takového komplexního nástroje je poměrně komplikovaná a právě tato skutečnost dala vzniknout různým API popsaným níže. V současné době neexistuje ideální programovací rozhraní, které by bylo jasnou volbou, je tedy nutné zvážit jednotlivá pozitiva a negativa, jejichž analýza by měla vést k adekvátní volbě. Jelikož jsme se rozhodli psát aplikaci v jazyce Python, který sice neposkytuje takovou výkonovou podporu, ale je velmi efektivní a přehledný v zápisu kódu museli jsme náš výběr omezit na projekty, které jej podporují. V rámci této práce jsme ve finále vybírali z dvou implementací, kromě nich existuje i široká škála jiných, nicméně drtivá většina buď nabízí totožnou funkcionalitu jen hůře zpracovanou a dokumentovanou, nebo je ve stádiu vývoje a není téměř používána. Proto při seriozním použití se aktuálně prakticky ani jiné projekty neuvažují.

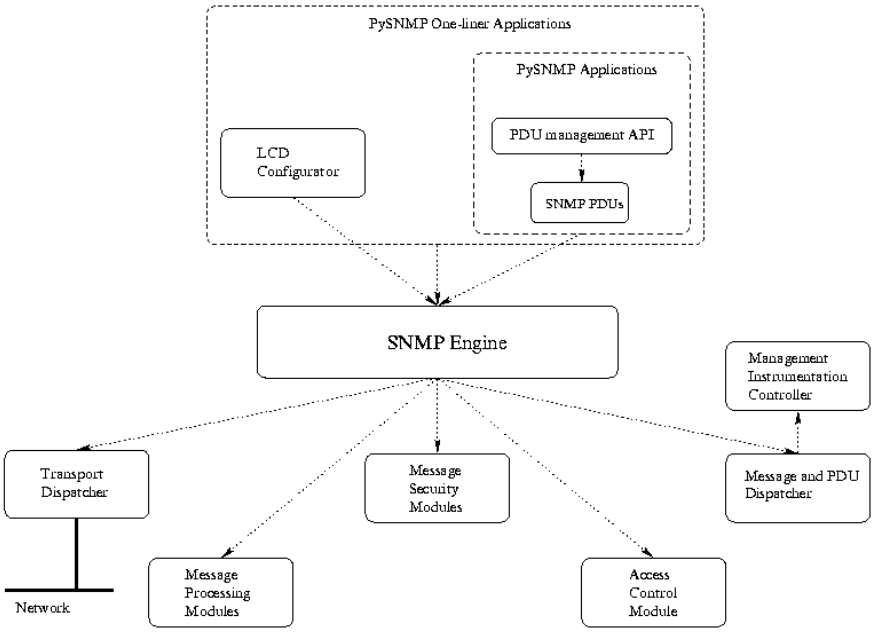
6.1.1 PySNMP

První popisovaný modul je čistě psaný v Pythonu. Za dobu svojí existence prošel už velkým množstvím změn. To přináší jak výhody, tak nevýhody. Při prvním pohledu zaujme poměrně slušně sepsaná dokumentace, která se sice nemůže rovnat velkým softwarovým projektům, ale jak později zjistíme je v tomto segmentu značně nadstandardní. Je k dispozici i velké množství tutoriálů. Problém ovšem je, že během vývoje došlo mnohokrát ke změně filosofie používání různých tříd, z kterých se nástroj skládá, proto lze narazit i na popisy, které jsou mystifikující, jelikož se vztahují ke starší verzi, která byla pojata úplně jinak.

Pokud se zaměříme na poslední verzi, která má označení 4.X, tak je dostupná podpora pro všechny majoritně využívané verze SNMP, tedy konkrétně verze 1, verze 2c a verze 3. Celý projekt se dělí do několika větví, ty lze odlišit bud za pomocí použité verze, nebo míry automatizace, kterou za vás systém provádí. V případě verzí má projekt následující charakteristiky.[13]

- API pro verze 1/2c - představuje nejvýkonnější část projektu, ale na programátora jsou kladeny vyšší nároky. Sám si musí stavět PDU a provádět parsování, také řešit chyby přenosu a jiné.
- API pro verzi 3 - zde už je úroveň automatizace dotažena výše. Na vývojáře jsou kladeny požadavky nastavovat strukturu Local Configuration Datastore a stavět, případně parsovat PDU.

Kromě těchto dvou existuje také nejvyšší úroveň API, která se nazývá One-liner, kde z názvu vyplývá, že zápis funkcionality by se měl vejít na jeden řádek. Pro lepší představu o systému níže uvádíme přehledné zobrazení architektury.



Obrázek 6.1: Architektura PySNMP

Z obrázku výše je vidět o které činnosti se stará která úroveň API a jak jsou jednotlivé moduly propojeny. Dále uvádíme jejich stručný popis.

- **LCD Configuration** - představuje specifický element v SNMPv3 definici. Slouží k uchování celé konfigurace SNMP Engine, tento přístup je výhodny i ze strany nastavování parametrů tohoto Engine přes technologii SNMP.
- **SNMP Engine** - jedná se o kompozitní třidu, která obsahuje odkazy na všechny komponenty nástroje. Jedná se o centrální prvek, který je sdílen více programy, jež využívají tuto implementaci.

- **Transport subsystem** - slouží pro přijímání a odesílání z připojení sítě. Struktura je složena z abstraktní komponenty Dispatcher a více transportních tříd. Ty pak již zajištují konkrétní implementaci pro specifickou technologii, která je zamýšlena k využívání. Tyto elementy jsou propojeny s dalšími prvky celé architektury. Konkrétně to jsou komponenty Message a PDU Dispatcher.

- **Message And PDU Dispatcher** - přebírají data od ostatních komponent systému a dále je předávají již zmíněnému elementu Transport Dispatcher, stejnou funkci plní pokud dorazí nějaká data opačným směrem ze sítě. Je zodpovědná také za udržování logického spojení s Management Instrumentation Controller, ten se stará o operace prováděné na jednotlivých objektech.

- **Message Processing Module** - obstarává operace na úrovni samotných zpráv. Modul je napsán obecně, tak aby mohla být později snadno doplnena množina podporovaných PDU, podle toho jak se bude protokol vyvíjet. Blíže zabezpečuje právě parsování a sestavování zpráv protokolu, k tomu také využivá moduly řešící zabezpečení technologie.

- **Message Security Module** - řeší autentizaci a šifrování komunikace mezi entitami. Implementace je zajištěna jak pro verze 1 a 2c, tak pro verzi 3. Všechny tyto přístupy mají standardizované API pro využití Message Processing modulem.

- **Access Control subsystem** - využívá LCD konfiguraci pro autorizaci vzdálených přístupů k spravovaným objektům. Tyto operace mohou probíhat jak na straně Agenta, tak Manažera.

Nyní, když jsou popsány základy architektury PySNMP projektu, je možné přikročit k praktické části a to sestavování jednotlivých dotazů. V tomto bodě je třeba rozhodnout jakou úroveň API využít. Z dokumentace je jasné, že všechny požadavky, které má strana Manažera je možno pokrýt nejvyšší úrovní, tedy One-liner API.

Tato část projektu se dělí do dvou větví, první je synchronní a druha asynchronní. Postupně popíši obě dvě.

- **Synchronní** - alfou a omegou celého přístupu je třída CommandGenerator, která jako přijímá volitelný parametr SNMP Engine ID a obsahuje několik metod, které již slouží k generování konkrétního dotazu.

  - **getCmd** - produkuje dotaz typu GET a přijímá následující parametry.
    - **authData** představuje objekt řešící zabezpečení protokolu. Typ obsahovaných parametrů se může lišit podle používané verze SNMP.
    - **transportTarget** - představuje objekt zajištující konfiguraci spojení. Povinně obsahuje dvojici IP adresa a port, volitelně je možné doplnit maximální dobu čekání na odezvu a počet pokusů.
    - **varNames** - obsahuje sekvenci jmen objektů, které chceme získat. V tomto případě tedy stačí pouze jméno.
  - Metoda vrací čtveřici errorIndication, errorStatus, errorIndex a varBinds. První z nich značí zda došlo při komunikaci k nějaké chybě, pokud ano je třeba přečíst i zbývající dvě, kde status identifikuje o jakou chybu šlo a index ukazuje do pole varBinds které položky se to konkrétně týká. Samotné varBinds představují sekvenci dvojic skládajících se z názvu objektu a jeho jména. Jednotlivé položky

si udržují reference k těm, které byli požadovány v dotazu, což je nutná vlastnost při indikaci chyby. Pro názornost níže uvádíme krátký příklad použití těchto nástrojů.

Listing 6.1: Příklad použití GET

```
>>> from pysnmp.entity.rfc3413.oneliner import cmdgen
>>> errorIndication, errorStatus, errorIndex, varBinds = cmdgen.
    CommandGenerator().getCmd(
... cmdgen.CommunityData('my-agent', 'public', 0),
... cmdgen.UdpTransportTarget(('localhost', 161)),
... '1.3.6.1.2.1.1.1.0',
... '1.3.6.1.2.1.1.2.0'
... )
>>> print(errorIndication)
None
>>> print(errorStatus)
0
>>> print(varBinds)
[(ObjectName(1.3.6.1.2.1.1.1.0), OctetString('Linux saturn
    2.6.37.6-smp
     #2 SMP Sat Apr 9 23:39:07 CDT 2011 i686')),
(ObjectName(1.3.6.1.2.1.1.2.0), ObjectIdentifier
    (1.3.6.1.4.1.8072.3.2.10))]
```

  - **nextCmd** - metoda implementuje dotaz typu GETNEXT. Tedy podle definice SNMP by měla vracet souseda, který následuje po dotazovaném prvku. Vstupní parametry jsou defakto stejné jako u varianty pro GET, stejně jako většina výstupních. Rozdíl nastává až v bodě varBindTable, což je struktura, která je schopna dosáhnout většího zanoření při reprezentaci více varBind objektů. To je výhodné chování když pochopíme způsob funkce této metody, které vrací všechny elementy MIB databáze, pro něž je OID dotazovaného objektu prefixem. Použití je prakticky téměř totožné, proto neuvádíme příklad.

- **Asynchronní** - přestavuje základ funkce pro synchronní variantu, proto mají některé charakteristiky společné. Použití tohoto přístupu je vhodné pro zvýšení výkonosti aplikace, jelikož během vykonávání SNMP dotazu uplyne poměrně dlouhá doba, je možné mezitím provádět jinou činnost, nebo dotazovat jiná zařízení. Jádrem této větve je AsynCommandGenerator, který přijímá jako volitelný parametr SNMP Engine ID a má následující metody.

  - **asyncGetCmd** - produkuje asynchronní dotaz GET a přijímá následující parametry. Je třeba zmínit, že authData, transportTarget a varNames mají stejný význam jako u synchronní varianty a proto je zde nebudu uvádět.
    * **dvojice cbFun a cbCtx** jedná se o volatelný objekt, často v Pythonu reprezentovaný jako funkce, který přijímá následující paramtery.
      · **sendRequestHandle** - spojuje jednotlivé dotazy a odpovědi, existují dvě inkarnace této hodnoty a to první vrácená při formulování dotazu a druhá při požadavku na vrácení odpovědi.
      · **errorIndication, errorStatus, errorIndex, varBinds** - hodnoty mají stejný význam jako při výstupu synchronního GET.
      · **cbCtx** - jedná se o doplňkový parametr, který slouží k podpoře skrytých stavů aplikace.

– Hlavní rozdíl oproti synchronní variantě tkví ve funkc cbFun, která se také stará o obdržení a zpracování dat, které přejímá od modulu Transport Dispatcher, jež je aktivován metodou runDispatcher. Pro ucelení získaných informací opět uvádíme příklad níže.

Listing 6.2: Příklad použití asynchronní GET

```
>>> from pysnmp.entity.rfc3413.oneliner import cmdgen
>>>
>>> def cbFun(sendRequestHandle, errorIndication, errorStatus,
    errorIndex, varBinds, cbCtx):
...     print('sendRequestHandle = %d' % sendRequestHandle)
...     print('errorIndication = %s' % errorIndication)
...     print('errorStatus = %s' % errorStatus)
...     print('varBinds = %s' % (varBinds,))
...     print('cbCtx = %s' % cbCtx)
...
>>> asynCommandGenerator = cmdgen.AsynCommandGenerator()
>>> # This is a non-blocking call
>>> sendRequestHandle = asynCommandGenerator.asyncGetCmd(
... cmdgen.UsmUserData('my-user', 'my-authkey', 'my-privkey'),
... cmdgen.UdpTransportTarget(('localhost', 161)),
... ((1,3,6,1,2,1,1,1,0),),
... (cbFun, None))
>>> print(sendRequestHandle)
1
>>> asynCommandGenerator.snmpEngine.transportDispatcher.
    runDispatcher()
sendRequestHandle = 1
errorIndication = None
errorStatus = 0
varBinds = [(ObjectName('1.3.6.1.2.1.1.1.0'),
             OctetString("'Linux saturn 2.6.21 #2 Mon Mar 19
                 17:07:18 MSD 2006 i686'"))]
cbCtx = None
>>>
```

– **asyncNextCmd** - v používání této metody neexistují oproti asynchronímu GET větší rozdíly, pouze při vkládání varNames se využívá způsob předání přes n-tice.

Projekt PySNMP se v tomto bodě jeví jako velmi perspektivní a užitečný nástroj, nicméně při testování jsme narazili na několik nestandardních chování, které nešlo odstranit. Například při pokusu implementace operace SNMP walk, která by měla být schopna procházet podstrom MIB, nástroj stále vracel více jak tento podstrom. Takovému chování se nepodařilo zamezit ani dodatečnou konfigurací modulů. Proto jsme se rozhodli pokračovat v hledání nového nástroje dále.

6.1.2 Net-SNMP

Net-SNMP představuje soubor aplikací, které podporují komunikaci přes protokol SNMP, ty lze rozdělit do dvou úrovní. První z nich je více nízko úrovňová a řeší spíše realizaci jednotlivých typů dotazů, tedy například GET, GETNEXT a další. Druhá vrstva již implementuje komplexnější aplikace, které již nemohou být realizovány jednoduchým dotazem, jak jsou popsány ve standardu technologie. Sem můžeme zařadit například snmptable, které slouží pro získání dat z tabulky reprezentované v MIB podstromu a její následné vypsání

ve formátované podobě. Případně existují další programy pro nastavování bezpečnostních parametrů a sledování různých provozních parametrů prvku.

Jakmile chceme využít API, zjistíme, že systém je psán kompletně v jazyce C. Nicméně existuje projekt, jenž se zabývá propojením jazyku Python a Net-SNMP. Nazývá se Net-SNMP Python bindings. Dále se již budu věnovat popisu tohoto propoje, jehož dokumentace je bohužel poměrně slabá, ale díky snadné čitelnosti, lze infomace poměrně snadno vyčíst i z kódu propoje. Výraznou výhodou je především to, že na pozadí jsou logicky využívány nastroje C implementace Net-SNMP, která jako taková je považována za aktuální standard implementace tohoto protokolu. Proto jsou výsledky těchto dotazu dobře interpretovány a komunikace zatím ve všech testovaných případech dopadla přesně podle očekávání.

Net-SNMP propoj je podporová na operačním systému Linux 2.x, současně by měl fungovat i na jiných UNIXových systémech a Microsoft Windows. Tyto poslední varianty ale nebyly testovány a náš nástroj byl vyvíjen pod Ubuntu, proto kompatibilitu nelze zaručit. Aktuálně není podporována kompletní funkcionalita, která je poskytována v originální C implementaci, nicméně pro potřeby Manažera je dostatečná. Podporována bohužel není ani asynchronní přistup, to se ovšem dá obejít vlákny, nebo více procesy. Posledním zmíněným způsobem tento problém řeší i rozšíření s názvem Multi-core SNMP, které je ale zatím ve fázi vývoje, nicméně autoři mají ambiciózní plány do budoucna.

Za nevýhodu nástroje se dá považovat nutnost vzájemné kompilace s Net-SNMP. Po jejím průběhu vše ale probíhá regulérně, dále uvádíme krátký postup instalace celého nástroje za pomocí příkazové řádky v systému Ubuntu.

Listing 6.3: Instalace Net-SNMP s Python bindings

```
apt-get install gcc
apt-get install libperl-dev
apt-get install python2.7-dev

wget http://downloads.sourceforge.net/project/net-snmp/net-snmp/5.7.1/net-
    snmp-5.7.1.tar.gz?r=http%3A%2F%2Fsourceforge.net%2Fprojects%2Fnet-snmp%2
    Ffiles%2Fnet-snmp%2F5.7.1%2F&ts=1318446955&use_mirror=surfnet

tar xzvf net-snmp-5.7.1.tar.gz
cd net-snmp-5.7.1

./configure --with-python-modules

make

make install

ldconfig
```

Nejdříve je nutné stáhnout základní nástroje na práci a překlad. Následně je nutné stáhnou aktuální verzi zdrojových kódů, provést konfiguraci překladu s parametrem přidružující python propoj. V závěru pak přeložit, nainstalovat a provést nastavení nástroje.

Pokud se zaměříme na způsob použití, existují tu dva způsoby. První z nich slouží pro verze 1 a 2c, druhý pak pro verzi 3. Jejich společnou charakteristikou je příprava dotazovaných dat. Zde je nutné cílené OID, které může být buď v kompletním numerickém tvaru, nebo pouze částečně zadané pomocí slovní interpretace. Tento řetězec je tedy nutné předat metodě Varbind, která vrací interní reprezentaci, jež se dá dále použít již v konkrétním dotazu. Níže uvádíme stručný popis pouze těch metod, jež jsou relevantní k našemu nástroji.[10]

- **metody pro verze 1 a 2c** - hlavní rozdíl spočívá jak již ve zmiňovaném zabezpečení, tak ve finální kompletaci množiny OID. V tomto případě stačí již popsané použití metody Varbind, výstupní data je již možné dát na vstup. Pro korektní funkci je třeba zprostředkovat pokaždé čtyři parametry.

  - **Množina OID** - může obsahovat buď jednu položku, nebo více. Pokud potřebujeme získat během jednoho dotazu více informací, je nutné výstupy Varbind předat do metody Varlist, která vytvoří seznam takovýchto dat, jež jsou pak zpracovány v rámci vytvořeného sezení. Z toho vyplývá, že na realizaci získání jedné položky od Agenta není nutné sezení zakládat.
  - **Verze** - indikuje jakou verzi SNMP má nástroj použít.
  - **DestHost** - představuje IP adresu cílového zařízení.
  - **Community** - jedná se o zabezpečení v rámci těchto verzí, tedy community string.

  Výše popsané parametry jsou vloženy na vstup následujícím metodám.

  - **snmpwalk** - realizuje průchod tabulkou, tedy automaticky buduje následující indexy. Vstupní OID tedy musí odkazovat pouze položku v tabulce, zbytek je dotvořen automaticky.
  - **snmpget** - z názvu je jasné, že provádí jednoduchý GET dotaz. Může mu být předloženo více položek v seznamu.
  - **snmpgetnext** - vytváří GETNEXT, tedy vrací element následující po dotazovaném.

- **metody pro verzi 3** - jelikož poslední verze vyžaduje složitější zabezpečení protokolu, je nutné ho promítnou i zde, proto při tvorbě dotazů je nutné použít i sezení o které se stará specializované metoda Session. Tím se následně mění i počet požadovaných parametrů, jejich výčet uvádíme níže.

  - **Version** - specifikuje verzi, v tomto případě bude hodnota vždy rovna třem.
  - **DestHost** - IP adresa cílového prvku.
  - **SecLevel** - definuje úroveň použitého zabezpečení. Zde jsou použity klasické názvy. Konkrétně jde o noAuthNoPriv, authNoPriv, authPriv.
  - **SecName** - zde se definuje jméno uživatele, kterého chceme autentizovat.
  - **PrivPass** - k uživateli volitelně podle úrovně zabezpečení přiřazeno heslo.
  - **AuthPass** - další heslo sloužící ale v kontextu autentizace.

Pokud je již vytvořeno sezení můžeme pomocí z něj získaného objektu volat metody formulující dotazy. Jejich funkce je de facto analogická vzhledem k předchozímu výčtu, proto je již nebudu popisovat. Jedná se o **walk**, **get**, **getnext**.

Vzhledem k uspokojivým výstupům tohoto nástroje jsme se rozhodli jej využít pro kompletaci budovaného programu. Funkcionalita byla v pořádku jak u starších verzí standardu, tak u nejnovější. Jedinou stinnou stránkou projektu je slabá dokumentace, kde programátor musí občas hledat ve zdrojovém kódu, aby zjistil korektní použití různých komponent. V celkovém souhrnu tedy převážily výhody nad nevýhodami a proto jsme zvolili tento projekt pro kompletaci našeho nástroje.

6.2 Popis struktury nástroje

V této části diplomové práce se dostanu již ke konkrétní podobě implementace. Při vývoji byl použit jazyk Python především díky snadné čitelnosti kódu a efektivnosti zápisu. Pokud se podíváme na nevýhodu výkonové stránky této platformy, tak zjistíme, že není příliš podstatná, jelikož majoritní prodleva ve výpočtu algoritmu tkví především v sítové komunikaci. Pro zakomponování funkcionality SNMP byl použit modul Net-SNMP s rozšířením SNMP bindings, jak již bylo popsáno v předchozí kapitole. Dále se již dostanu k popisu samotné architektury.

Celý nástroj je rozdělen do několika tříd, které jsou pro přehlednost distribuovány v různých souborech. Ty jsou pak připojeny v hlavním elementu, který nese název topoGet. Jak by se dalo předpokládat kromě této agregace zajišťuje také parsování vstupních parametrů, implementaci algoritmu prohledávání do šířky, který koordinuje samotný výpočet. Poslední část o kterou se stará je generování relevantního výstupu, jež musí být ve validním formátu vzhledem k definici jazyka ned. Níže uvedu výpis validní kombinace parametrů, které program přijímá.

- **Parametry pro přístup přes verze 1 a 2c**
  - topoGet v1/v2c community string host ip
- **Parametry pro přístup přes verzi 3**
  - topoGet v3 noAuthNoPriv user host ip
  - topoGet v3 authNoPriv user authPass host ip
  - topoGet v3 authPriv user authPass privPass host ip
- **Kombinovaná varianta**
  - topoGet v1/v2c community string v3 noAuthNoPriv user host ip
  - topoGet v1/v2c community string v3 authNoPriv user authPass host ip
  - topoGet v1/v2c community string v3 authPriv user authPass privPass host ip

Z předchozího seznamu je patrné, že je možné krom využití různých verzí aplikovat i kombinovanou variantu. V tomto případě dojde nejprve k pokusu o autentizaci pomocí starších verzí a pak teprve přes třetí. Přístup se následně liší podle úrovně zabezpečení. Tedy buď pouze vložení komunitního řetězce, nebo přímo určení úrovně v třetí verzi. Zde jsou používány následující pojmy, které lze napříč technologiemi považovat za již standardizované.

- **noAuthNoPriv** - v tomto případě není použita téměř žádná forma autentizace. nebo autorizace. Nicméně některé systémy požadují alespoň uživatelské jméno, proto je zde také zahrnuto.
- **authNoPriv** - zde je již aplikována autentizace, proto je nutné poskytnout nástroji kromě uživatelského jména také heslo.
- **authPriv** - tato varianta podporuje již důvěrnost, proto je nutné vložit i heslo sloužící k tomuto účelu.

Jakmile dojde k poskytnutí relevantních parametrů, je možné spustit algoritmus prohledávání do šířky, jenž koordinuje celou sítovou komunikaci a výpočet jako takový.

Nyní jakmile máme přehled o funkci centrálního prvku nástroje, přikročíme k popisu jednotlivých tříd, které podporují jeho funkcionalitu. Jedná se o snmp, node, netModel, genOut, které představují v uvedeném pořadí podporu pro následující operace. První se již podle názvu stará o zajištění komunikace pomocí protokolu SNMP, druhá pak slouží jako abstraktní struktura pro reprezentaci jednoho uzlu v rekonstruované topologii, třetí kompletuje již kompletní zobrazení zkoumané sítě, zde jsou aplikovány různé přístupy reprezentace, podle toho v jakém kontextu je vhodné je využít. Poslední se pak stará o implementaci metod pro generování relevantního výstupu. Po tomto krátkém úvodu přistoupíme k bližšímu popisu, který budu provádět formou výčtu poskytované funkcionality a na něj navazující seznam metod, které ji implementují.

6.2.1 Třída snmp

Tato část nástroje se stará jak již bylo řečeno o podporu komunikace pomocí protokolu SNMP. Samotná poskytuje následující funkce.

- **Konfigurace OID** - jedná se především o nastavení jednotlivých vstupních parametrů pro modul Net-SNMP bindings, který je schopen tato data přijímat ve dvou formátech. První je struktura VarBind, která obsahuje pouze jedno OID. Další představuje VarList, který je jich schopen obsáhnout více, ale kvůli omezením plynoucích ze současného stavu implementace je aktuálně využíván pro uskladnění pouze jedné. Důvod nasazení právě posledně jmenované struktury tkví v rozšířené množině informací, kterou lze dostat od Agenta. VarList totiž uchová kromě návratové hodnoty na SNMP dotaz také vrácené OID. Toho lze pak výhodně využít při prozkoumávání indexů jednotlivých tabulek. Následující metody se podílí na této funkcionalitě:
  - **setOID**
  - **setOIDSet**
  - **clearVar**
  - **setTech**
- **Nastavení parametrů SNMP komunikace** - v tomto případě rozlišujeme mezi komunikací dvou starších verzí, tedy 1 a 2c, nebo pomocí nejnovější. Pokud by cílová síť obsahovala oba přístupy, tak je zahrnuta i kombinovaná varianta. Tyto stavy jsou rozlišeny pomocí proměnné *mode*, která nabývá hodnot 1 až 3. 1 a 2 indikuje použití verzí 1, 2c nebo 3, přesně v tomto pořadí. Hodnota 3 značí pak kombinovanou variantu. Jakmile je tato hodnota určena ze vstupu aplikace, je nutné vložit adekvátní autentizační parametry. V nejjednodušším případě stačí komunitní řetězec a IP adresa. Ve složitějším se již nastavují úrovně autentizace a příslušná uživatelská jména a hesla. V tomto kontextu je vytvářeno sezení s cílovým prvkem. Níže uvádíme výčet zainteresovaných metod:
  - **setMode**
  - **setV3**
  - **setV3params**

  - **createSession**
  - **createV2Sess**
  - **setHost**
  - **setVersion**
  - **setCommunity**

- **Vytváření SNMP dotazů** - představuje hlavní prvek třídy. Formuje všechny používané dotazy, kde nejčastěji je nasazen typ SNMPWALK, což není dotaz ze standardu SNMP, ale množina zpracovaných GETNEXT požadavků. Každý SNMPWALK vrací n-tici hodnot pokud je aplikován na tabulku. Tyto jednotlivé metody jsou pak agregovány výše, tak aby adekvátně odpovídaly implementaci modifikovaného algoritmu prohledávání do šířky. Jako doplňující funkce jsou zahrnuty metody pro parsování indexů, které jsou součástí vrácených OID. Účelem takového přístupu je vlastnost návrhu LLDP MIB, který neumožnuje získat standardní cestou IP adresu sousedního prvku. Nicméně jednotlivé záznamy, reprezentující právě tyto sousedy, jsou indexovány také jejich IP adresou. Tedy za pomocí iterativně stylizovaného parseru je možné tuto informaci získat. V rámci třídy jsou také vytvářeny dotazy na jednotlivé tabulky obsahující dynamická data. Sem patří směrovací, MAC a ARP tabulka. Poslední částí implementace je sumarizace získaných informací, jelikož je možné, že v síti budou definovány přístupy k jednotlivým zařízením přes různé verze protokolu, je nutné s touto variantou počítat. Nástroj tedy vyzkouší přístup k prvkům pomocí všech poskytnutých dat. Výsledná n-tice výsledků je prozkoumána a navrácena je pouze neprázdná množina.

  - **getLocID**
  - **getIndexFromCurrentQuery**
  - **parseIndexFromCurrentQuery**
  - **queryWalk**
  - **queryWalkv3**
  - **queryGet**
  - **queryGetNext**
  - **querySet**
  - **querySetv3**
  - **queryNodes**
  - **setNode**
  - **getNodesInfo**
  - **combineInfo**
  - **getSingleNode**

- **Získání konfiguračního souboru** - jelikož v rámci projektu ANSA je i nástroj pro automatickou analýzu konfiguračních souborů jednotlivých zařízení, je vhodné tato data v průběhu prozkoumávání topologie také stáhnout. Jelikož monitorovaná sít VUT se skládá výhradně z prvků od společnosti HP, soustředili jsme se na metody aplikovatelné právě na tato zařízení. Cestou jak data získat je více, ale s výhodou lze

použít kombinaci SNMP a TFTP protokolu. V MIB struktuře existuje v proprietární části výrobce položka indikující dostupnost přes TFTP, ta po nastavení na adekvátní hodnotu zpřístupní hledaný konfigurační soubor. Pak je možné provést jednoduché spojení pomocí protokolu TFTP a požadovaná data stáhnout. K tomuto účelu jsme zamýšleli použít volně šiřitelnou implementaci TFTP s názvem Tftpy, která se ale ukázala jako funkčně nespolehlivá, proto jsme přešeli na alternativní způsob. Ten spočívá ve vytvoření subprocesu, jenž volá nástroj příkazové řádky konkrétně balík tftp, který se vyznačuje dobrou funkcionalitou. Skript zavolá tento program a počká na jeho dokončení, to je nutné provést vzhledem k tomu aby bylo možné uvést nastavení přístupu přes TFTP protokol k prvku zpět do původního stavu. V opačném případě by nástroj mohl vytvářet bezpečnostní díry v systému.

  - **getCfgFile**
  - **downloadCFGFile**

6.2.2 Třída node

V tomto bodě se staráme o reprezentaci jednoho uzlu sítě. Třída je logicky propojena s částmi *snmp* a *netModel*, kde v první jsou vytvářeny jednotlivé instance, podle toho jak je topologie prohledávána a prostřednictvím druhé dochází k jejich organizaci a zajištění vhodného přístupu. Tedy ošetření různých výjímek a zajištění implementačního detailu obecně. Kromě vlastní konstrukce objektů třída pak poskytuje následující funkce.

- **Konverze vstupních dat** - jelikož informace získané od Agentů nemusí být pokaždé ve formě řetězce, ale často jsou poskytnuty v binárním formátu, je nutné provést adekvátní konverzi do vhodné reprezentace. To se týká především získávání IP adres sousedů přes protokol CDP, plnění MAC tabulky a identifikace schopností jednotlivých prvků.

  - **convIP4**
  - **convPhyAdd**

- **Rozeznání schopností prvků** - nástroj podporuje přístup přes CDP i LLDP. Bohužel každý protokol má tuto informaci zakódovanou jiným způsobem, proto je vhodné implementovat separátní metody. Výhodou je, že alespoň princip je totožný, tedy množina schopností je zakódována v několika bajtové hodnotě. Rozdíl je tedy pouze v její interpretaci. O to se starají následující metody.

  - **convCapabCDP**
  - **convCapabLLDP**

- **Načtení dynamických dat** - v rámci této práce jsme také provedli extrakci několika nejdůležitějších prvků z množiny dynamických dat, které jednotlivé síťové prvky během své činnosti vytváří. Jako nejzásadnější jsme identifikovali tabulky MAC, ARP a směrovací tabulku. Každý tento prvek má svoji MIB strukturu a lze je tedy získat pomocí SNMP protokolu. Navíc je ale nutné provést interpretaci jednotlivých hodnot, které reprezentují například status položky MAC, nebo protokol přes který byla naučena určitá cesta v směrovací tabulce. Posledním prvkem komplikujícím kompletaci

těchto informací je provázanost s jednotlivými rozhraními prvku, to se týká všech zúčastněných, tedy směrovací, MAC a ARP tabulky. V MIB struktuře je obsažen pouze index odkazující do tabulky rozhraní, proto je nutné kompletovat i ji a získané informace doplnit do adekvátních položek.

  - **interpretStatus**
  - **getPortDesc**
  - **setRoute**
  - **setMAC**
  - **setARP**
  - **setIF**
  - **interpretProto**

6.2.3 Třída netModel

Prvek nástroje zajišťuje reprezentaci topologie jako celku. K tomu využívá několik datových struktur, které je výhodné využit v různých situacích. Cílem bylo především vyhnout se jakékoliv formě vyhledávání, a proto veškerá data jsou dostupná přes index, který musí být v rámci sítě unikátní. Pro tento účel jsme zvolili kombinaci fyzických adres rozhraní prvku a sériového čísla. Druhý element byl přidán kvůli zařízením, jejichž rozhraní žádnou adresu první úrovně nemají. Níže v textu rozepíši význam jednotlivých struktur a metody, které s nimi pracují.

- **Fronta pro modifikovaný algoritmus BFS** - tato datová struktura musí mít skutečně vlastnosti fronty jako takové, proto jsme zvolili interní reprezentaci v Pythonu pod názvem Queue, která již sama o sobě obsahuje všechny potřebné metody až na kontrolu duplicitních prvků. Tato vlastnost byla doplněna v přístupové metodě k ní přidružené.

  - **addBFS**

- **Pole sousedností** - jedná se o velmi efektivní způsob jak reprezentovat orientovaný, nebo neorientovaný graf. Příklady blíže popisující princip byly uvedeny již v části pojednávající o rekonstrukci topologie. Data poskytovaná tímto způsobem jsou pak využita při generování výstupu v jazyce NED a vznikají při samotném výpočtu BFS. Implementace probíhá datovou strukturou známou v Pythonu pod názvem Dictionary, k ní pak stejně jako u předchozího příkladu byla doplněna kontrola na duplicitu.

  - **addNode**
  - **addConn**

- **Kolekce prvků** - představuje množinu instancí třídy *node*. Tedy pokud je nutné získat konkrétní data o zařízení v topologii, tak se nacházejí zde, indexované pomocí unikátního identifikátoru. K implementaci byla použita struktura Dictionary.

  - **addNodeColInfo**
  - **isColInfo**

- **Pole color** - identifikuje již uzavřené uzly v rámci algoritmu a je implementované opět pomoci Dictionary. Přístup je realizován přes unikátní identifikátor.
  - **setBlack**
  - **getBlack**

6.2.4 Třída genOut

Princip funkce je založen na postupném budování řetězce, jehož metody jsou v průběhu využívány a který je ve finále jednorázově zapsán do souboru. Třidu lze tedy rozdělit do dvou částí. První se stará o práci se soubory a druhá o validní konstrukci řetězce.

- **Zápis do souboru** - zde se využívají standardní metody jazyka Python, tedy *open* která vrací objekt reprezentující otevřený soubor a *close* což je metoda tohoto objektu, jež slouží k uzavření souboru. Pokud byl již sestaven validní řetězec a je zavoláno toto uzavření, dojde ještě dříve k jednorázovému zápisu.
  - **OpenF**
  - **close**
- **Konstrukce výstupního řetězce** - každý soubor v jazyce NED má svoje specifika, pokud má reprezentovat topologii počítačové sítě. Mezi ně patří například definice kanálu pro propojení jednotlivých prvků a generování formátu souboru, tak aby byl strukturován syntakticky správně. Kromě tohoto je nutné zahrnout i podmoduly z nichž se síť skládá a pak již definovat samotné propoje mezi nimi. Jelikož v simulacích ANSA projektu se vyskytují definice formou neorientovaného grafu, je nutné provádět kontrolu na duplicitu jak v rámci přidávaných modulů, tak ve spojeních. K tomu jsou využity metody objektu řetězec. V rámci duplicit je ale také nutné ošetřit redundantní spoje v síti, ty se identifikují v kontextu jednoho uzlu, kde je jim udělena výjímka, a jsou zařazeny do souboru, který reprezentuje topologii. Jako poslední byla přidána podpora pro výstup do nástroje GraphViz, který je schopen generovat grafické zpracování topologie v různých formátech. Výstup se generuje společně s NED souborem.
  - **Write**
  - **WriteGV**
  - **addPack**
  - **addImp**
  - **setNetw**
  - **checkDupl**
  - **addRouter**
  - **addSwitch**
  - **setChan**
  - **addCon**

6.2.5 Získání dynamických dat

Jako poslední část byla zahrnuta množina několika funkcí a metod, které doplňují funkcionalitu nutnou k získání dynamických dat. Jelikož způsob jak tyto informace získat byl již popsán v rámci metod jednotlivých tříd, budeme se dále věnovat formátu výstupu. V tomto případě je jasnou volbou tvorba nových elementů XML struktury. Získaná data mají formu tabulky proto je návrh poměrně přímočarý. Níže uvádíme podobu struktury.

Listing 6.4: Příklad struktury pro dynamická data

```
<Devices>
        <Router id="11.0.0.1">
                <routingTable>
                        <routeEntry Destination="10.0.0.0" Interface="
                            FastEthernet0/0" Metric="0" NextHop="10.0.0.1"
                            Protocol="local"/>
                </routingTable>
                <ARPTable>
                        <ARPEntry NetAddress="10.0.0.1" PhyAddress="0xc8:0x1
                            :0x11:0x80:0x0:0x0" Port="FastEthernet0/0" Type
                            ="static"/>
                </ARPTable>
        </Router>
        <Switch id="11.0.0.2">
                <MACTable>
                        <MACEntry Address="0xc8:0x0:0x11:0x80:0x0:0x0" Port
                            ="FastEthernet0/0" Status="learned"/>
                </MACTable>
                <ARPTable>
                        <ARPEntry NetAddress="11.0.0.2" PhyAddress="0xc8:0x0
                            :0x11:0x80:0x0:0x0" Port="FastEthernet0/0" Type
                            ="static"/>
                </ARPTable>
        </Switch>
</Devices>
```

Generování provádí skupina funkcí zakomponovaná do hlavního souboru aplikace. Jako prostředek využívají implementaci MiniDom v rámci jazyku Python. Proces tvorby jsme rozdělili nejprve na klasifikaci typu zařízení.

- **buildXML**

Dále pak na navazující volání adekvátních funkcí, zde jsou implementováni tři reprezentanti.

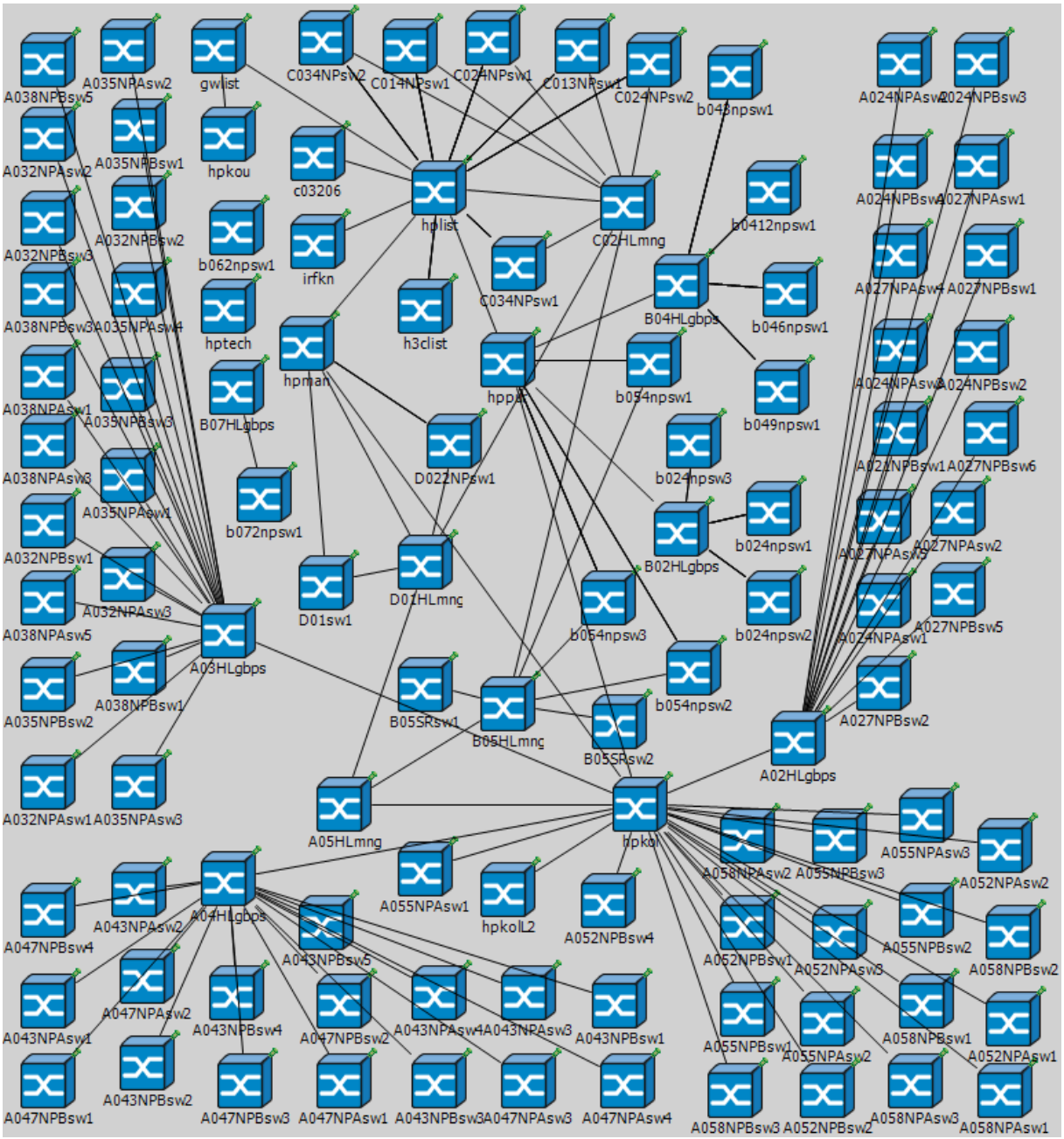
- **addXMLRouter**
- **addXMLSwitch**
- **addXMLL3SW**

Tento návrh se jeví jako vhodný především do budoucna, pokud bude rozšiřována funkcionalita.

Kapitola 7

Testování na síti VUT

V průběhu dokončování nástroje jsme se rozhodli jeho funkci otestovat také na produkční síti VUT. K tomuto kroku jsme přistoupili, jakmile proběhly všechny laboratorní testy v pořádku. Podle očekávání došlo během zkoušky k odhalení dalších skutečností, které bylo nutné zohlednit ve zdrojovém kódu programu. Jelikož se sledovaná topologie skládá výhradně z přepínačů, které mají přiřazené různé IP adresy, tak nelze přistupovat přes všechny tyto adresy k SNMP strukturám. Tímto způsobem vzniká inkonzistence mezi tabulkami v kontextu LLDP, s kterou je nutné v rámci kódu počítat. V síti VUT je možné taky aplikovat jiný přístup získávání IP adres a to pomocí DNS dotazů. V současnosti je hotová implementace pouze jako náhradní možnost za LLDP. Tedy pokud selže dotaz přes adresu získanou ze zmíněného LLDP, tak se realizuje druhá možnost pomocí DNS systému. V tomto případě se vezme jméno prvku a přidá se k němu zbytek doménového názvu. Výhoda toho přístupu je že takto kompletovaný dotaz vrací IP adresy z rozsahu správcovské VLAN, tedy přístup přes SNMP je garantovaný. V budoucnu je možné tento záložní mechanizmus aplikovat i na množinu sousedů prvku. Po těchto úpravách byl již nástroj schopen rozpoznat drtivou většinu topologie. Výstup generovaný pomocí IDE OMNeT++ uvádím kvůli prostorovým důvodům o stránku níže.



Obrázek 7.1: Výstup nástroje

Kapitola 8

Závěr

V první části diplomové práce jsme popsali protokol SNMP, který jsme se rozhodli použít jako hlavní prostředek pro získávání dat od reálných zařízení a o topologii jako takové. Prozkoumali jsme všechny aktuálně standardizované verze, kde nejvýhodněji se jeví 2c z důvodu nejširší podpory prakticky na drtivé většině síťových prvků. Její nevýhodou je slabá bezpečnost pro produkční prostředí. To ale neshledáváme jako velkou překážku, pokud se pohybujeme na akademické půdě pro účely testování vnitřní sítě, která povoluje tento provoz pouze v sobě samotné.

Následně jsme shrnuli základní vlastnosti protokolů pro prohledávání sítě. Zvolili jsme dva nejvýznamnější zástupce CDP a LLDP. Pak jsme provedli jejich porovnání, zde nejlépe vyšla varianta LLDP. Nicméně v nástroji je nastavena podpora pro obě dvě.

Dále bylo nutné se seznámit se samotným simulačním nástrojem OMNeT a jeho rozšířeními INET Framework a domácí částí ANSA. V příslušné části proběhlo popsání jeho principů a uvedení zamýšlené cesty konfigurace dynamických dat. Kromě toho jsme shrnuli také aktuální podobu konfiguračního souboru, po prozkoumání dostupných implementací.

Potom bylo možné přistoupit k rozboru přístupů k detekci topologie. Zde jsme shrnuli několik přístupů a zvolili ten nejvhodnější. V návaznosti na něj byl vybrán algoritmus na procházení grafu, který bylo nutné náležitě upravit pro reálné nasazení. Zde jsme se rozhodli pro modifikaci prohledávání do šířky, která byla následně úspěšně otestována v laboratoři.

Navazující kapitola byla věnována již konkrétní podobě implementace a s tím související popis využitých externích nástrojů pro samotnou realizaci SNMP dotazů.

Během dokončování nástroje jsme také přistoupili k testování na síti VUT, kde jsme ověřili dosaženou funkčnost.

Realizace této diplomové práce byla velmi přínosná pro poznání bližších detailů protokolů SNMP, CDP a LLDP. Kde jsme narazili na několik omezení, které byly ale úspěšně vyřešené s čímž souvisela také bližší analýza jednotlivých MIB struktur. Konkrétně především role jednotlivých indexů tabulek a jejich vzájemné propojení. Přínosná také byla zkušenost s praktickou aplikací teoretického algoritmu prohledávání do šířky a s tím spojené studium detekce topologie.

Jelikož je disciplína rozpoznávání topologie velmi komplexní a nabízí se velké množství přístupů, dá se nástroj rozšířit tímto směrem a pokusit se analyzovat i další prvky, které nepodporují protokoly CDP a LLDP. Kromě tohoto je možné rozšířit množinu získaných dynamických dat ze zařízení o další elementy.

Literatura

[1] ANSA Wiki. https://nes.fit.vutbr.cz/ansa/pmwiki.php?n=Main.HomePage.

[2] CDP configuration. http://www.cisco.com/en/US/docs/ios/12_2/configfun/configuration/guide/fcf015.html#1000952.

[3] CDP frame format. http://www.cisco.com/univercd/cc/td/doc/product/lan/trsrb/frames.htm#xtocid12.

[4] Cisco White Paper, LLDP-MED and CDP. http://www.cisco.com/en/US/technologies/tk652/tk701/technologies_white_paper0900aecd804cd46d.html.

[5] Configuring LLDP. http://www.cisco.com/en/US/docs/switches/lan/catalyst4500/12.2/46sg/configuration/guide/swlldp.html#wp1066710.

[6] GAL VUT FIT. https://www.fit.vutbr.cz/study/courses/index.php?id=7965.

[7] INET manual. http://inet.omnetpp.org/index.php?n=Main.Manual.

[8] LLDP configuration. http://www.cisco.com/en/US/docs/switches/lan/catalyst4500/12.2/46sg/configuration/guide/swlldp.html#wp1066710.

[9] LLDP frame format. http://www.eetimes.com/design/communications-design/4009357/Tutorial-on-the-Link-Layer-Discovery-Protocol.

[10] Net-SNMP Python bindings documentation. https://net-snmp.svn.sourceforge.net/svnroot/net-snmp/trunk/net-snmp/python/README.

[11] OMNeT online manual. http://www.omnetpp.org/doc/omnetpp/manual/usman.html.

[12] PTOPO MIB RFC2922. http://tools.ietf.org/html/rfc2922#page-7.

[13] PySNMP documentation. http://pysnmp.sourceforge.net/docs/4.x/index.html.

[14] SNMP configuration. http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a0080094aa4.shtml.

[15] SNMP seriál. http://www.hw.cz/ethernet/snmp/vyuziti_snmp.html, 9. Září 2003.

[16] Case, J.; Fedor, M.; Schoffstall, M.; aj.: A Simple Network Management Protocol. `http://www.ietf.org/rfc/rfc1157.txt`, Květen 1990.

[17] Case, J.; McCloghrie, K.; Rose, M.; aj.: Introduction to version 2 of the Internet-standard Network Management Framework. `http://tools.ietf.org/html/rfc1441`, Duben 1993.

[18] Case, J.; McCloghrie, K.; Rose, M.; aj.: Introduction to Community-based SNMPv2. `http://www.ietf.org/rfc/rfc1901.txt`, Leden 1996.

[19] Harrington, D.; Presuhn, R.; Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. `http://www.ietf.org/rfc/rfc3411.txt`, Listopad 2002.

[20] Klaška, L.: SNMP seriál. `http://www.svetsiti.cz/clanek.asp?cid=2331`, 11. června 2000.

[21] Klaška, L.: SNMP seriál. `http://www.svetsiti.cz/clanek.asp?cid=2333`, 14. června 2000.

[22] Černý Marek: *Modelování IPv6 v prostředí OMNeT++*. FIT VUT v Brně, 2011.

[23] McCloghrie, K.: An Administrative Infrastructure for SNMPv2. `http://www.ietf.org/rfc/rfc1909.txt`, Březen 1996.

[24] Zeltserman, D.: *A Practical Guide to SNMPv3 and Network Management*. PRENTICE HALL, iSBN 0-13-021453-1.

Příloha A

Obsah CD

Na CD jsou přiloženy následující položky.

- **zdrojové soubory aplikace**
- **dokumentace**
- **poster**
- **technická zpráva - pdf**
- **technická zpráva - zdrojové soubory**
- **výstupní NED soubor z testování na síti VUT**

Příloha B

Manual

Aby bylo možné nástroj úspěšně spustit je nutné mít funkční následující prvky.

- **Python 2.7**
- **Net-SNMP s Python Bindings** - zde je nutná separátní kompilace, více níže uvedený příklad.

Listing B.1: Instalace Net-SNMP s Python bindings

```
apt-get install gcc
apt-get install libperl-dev
apt-get install python2.7-dev

wget http://downloads.sourceforge.net/project/net-snmp/net-snmp/5.7.1/
    net-snmp-5.7.1.tar.gz?r=http%3A%2F%2Fsourceforge.net%2Fprojects%2
    Fnet-snmp%2Ffiles%2Fnet-snmp%2F5.7.1%2F&ts=1318446955&use_mirror=
    surfnet

tar xzvf net-snmp-5.7.1.tar.gz
cd net-snmp-5.7.1

./configure --with-python-modules

make

make install

ldconfig
```

- **balíček TFTP**

Program se pak používá následujícím způsobem.

- **Parametry pro přístup přes verze 1 a 2c**
  - topoGet v1/v2c community string host ip
- **Parametry pro přístup přes verzi 3**
  - topoGet v3 noAuthNoPriv user host ip
  - topoGet v3 authNoPriv user authPass host ip

  - topoGet v3 authPriv user authPass privPass host ip

- **Kombinovaná varianta**
  - topoGet v1/v2c community string v3 noAuthNoPriv user host ip
  - topoGet v1/v2c community string v3 authNoPriv user authPass host ip
  - topoGet v1/v2c community string v3 authPriv user authPass privPass host ip

Příloha C

Konfigrační XML soubor

```
<Routers>
 <Router id="192.168.3.3">
 <Hostname></Hostname>
   <Interfaces>
    <Interface name="eth0">
       <IPAddress>192.168.1.1</IPAddress>
       <Mask>255.255.255.0</Mask>
       <Bandwidth>1000</Bandwidth>
       <Duplex>auto</Duplex>
       <Speed>auto</Speed>
       <OspfNetworkType>broadcast</OspfNetworkType>
       <OspfCost>1</OspfCost>
       <OspfPriority>1</OspfPriority>
       <OspfHelloInterval>10</OspfHelloInterval>
       <OspfDeadInterval>40</OspfDeadInterval>
       <OspfRetransmissionInterval>
       5
       </OspfRetransmissionInterval>
       <OspfInterfaceTransmissionDelay>
       1
       </OspfInterfaceTransmissionDelay>
       <OspfAuthenticationType>
       Null
       </OspfAuthenticationType>
       <OspfAuthenticationKey>
       0x00
       </OspfAuthenticationKey>
       <OspfProcess6>(Process ID)</OspfProcess6>
       <OspfArea6>(Area ID)</OspfArea6>
       <OspfInstance6>(n)</OspfInstance6>
       <OspfNetworkType6></OspfNetworkType6>
       <OspfCost6>(n)</OspfCost6>
       <OspfPriority6>(n)</OspfPriority6>
       <OspfHelloInterval6>(n)</OspfHelloInterval6>
       <OspfRouterDeadInterval6>
       (n)
       </OspfRouterDeadInterval6>
       <OspfRetransmitInterval6>
       (n)
       </OspfRetransmitInterval6>
       <OspfTransmitDelay6>(n)</OspfTransmitDelay6>
       <IS-IS-Priority>127</IS-IS-Priority>
```

```
            <IS-IS-Metric>50</IS-IS-Metric>
            <Pim>
                    <Mode>dense-mode</Mode>
                    <Border />
            </Pim>
    <shutdown></shutdown>
 </Interface>
 <Interface name="(interface name)">
    <IPv6Address>
    (IPv6 address)/(Prefix Length)
    </IPv6Address>
    <NdpAdvSendAdvertisements>
    (yes/no)
    </NdpAdvSendAdvertisements>
    <NdpAdvPrefix>
    (IPv6 prefix)/(Prefix Length)
    </NdpAdvPrefix>
    <NdpMaxRtrAdvInterval>
    (4-1800)
    </NdpMaxRtrAdvInterval>
    <NdpMinRtrAdvInterval>
    (3-1350)
    </NdpMinRtrAdvInterval>
  </Interface>
</Interfaces>
<Routing6>
  <Ospf>
    <Process id="(Process ID)">
      <RouterId>(Router ID)</RouterId>
    </Process>
  </Ospf>
  <Static>
    <Route>
      <NetworkAddress>
      (IPv6 prefix)/(Prefix Length)
      </NetworkAddress>
      <NextHopAddress>(IPv6 address)</NextHopAddress>
      <AdministrativeDistance>
      (1-254)
      </AdministrativeDistance>
    </Route>
  </Static>
</Routing6>
<DefaultRouter>(IPv6 Address)</DefaultRouter>
<Routing>
  <Static>
    <Route>
      <NetworkAddress>172.16.200.0</NetworkAddress>
      <NetworkMask>255.255.255.0</NetworkMask>
      <ExitInterface>ppp0</ExitInterface>
    </Route>
    <Route>
      <NetworkAddress>172.16.100.0</NetworkAddress>
      <NetworkMask>255.255.255.0</NetworkMask>
      <NextHopAddress>192.168.3.1</NextHopAddress>
    </Route>
  </Static>
  <ISIS>
            <is-type>level-1</is-type>
```

```
            <net>49.0001.1921.6801.2001.00</net>
        </ISIS>
        <Multicast enable="1">
    <Pim>
            <RPAddress>
                    <IPAddress>192.168.1.2</IPAddress>
                    <Acl>1</Acl>
            </RPAddress>
            <BSRCandidate>
                    <Interface>eth0</Interface>
                    <Mask>30</Mask>
            </BSRCandidate>
            <RPCandidate>
                    <Interface>eth1</Interface>
                    <Ttl>2</Ttl>
                    <GroupList>1</GroupList>
            </RPCandidate>
        </Pim>
        </Multicast>
    <Rip>
      <Network>130.10.0.0</Network>
      <Passive-interface>eth1</Passive-interface>
      <Passive-interface>eth2</Passive-interface>
      <Redistribute>
        <Protocol>ospf</Protocol>
        <Metric>4</Metric>
      </Redistribute>
      </Rip>
    <Ospf>
      <RFC1583Compatible />
      <Areas>
        <Area id="0.0.0.0">
          <Networks>
            <Network>
              <IPAddress>0.0.0.0</IPAddress>
              <Wildcard>255.255.255.255</Wildcard>
            </Network>
          </Networks>
        </Area>
      </Areas>
    </Ospf>
    <Eigrp>
      <AutonomousSystems>
        <AS id="1">
          <Networks>
            <Network>
              <IPAddress>192.168.3.0</IPAddress>
              <Wildcard>0.0.0.255</Wildcard>
            </Network>
          </Networks>
        </AS>
      </AutonomousSystems>
    </Eigrp>
  </Routing>
  <ACLs>
    <ACL no="120">
       <entry seq_no="1">
              <action>deny</action>
              <IP_src>192.225.2.0</IP_src>
```

```
                <IP_dst>0.0.0.0</IP_dst>
                <WC_src>0.255.0.255</WC_src>
                <WC_dst>255.255.255.255</WC_dst>
                <port_op_src>eq</port_op_src>
                <port_op_dst></port_op_dst>
                <port_beg_src>25</port_beg_src>
                <port_end_src></port_end_src>
                <port_beg_dst></port_beg_dst>
                <port_end_dst></port_end_dst>
                <protocol>udp</protocol>
                <orig>
                </orig>
              </entry>
              <entry seq_no="10">
                <action>deny</action>
                <IP_src>0.0.0.0</IP_src>
                <IP_dst>0.0.0.0</IP_dst>
                <WC_src>255.255.255.255</WC_src>
                <WC_dst>255.255.255.255</WC_dst>
                <port_op_src></port_op_src>
                <port_op_dst></port_op_dst>
                <port_beg_src></port_beg_src>
                <port_end_src></port_end_src>
                <port_beg_dst></port_beg_dst>
                <port_end_dst></port_end_dst>
                <protocol>ospf</protocol>
                <orig>
                access-list 120 deny ospf any any
                </orig>
               </entry>
               <interfaces>
                 <interface dir="in">eth0</interface>
                 <interface dir="out">eth0</interface>
            <interface dir="in">eth1</interface>
              </interfaces>
          </ACL>
    </ACLs>
  </Router>
</Routers>

  <Switch id="Switch1">
    <Hostname>Foo</Hostname>
      <Interfaces>
        <Interface id="0">
          <Name>SW2</Name>
          <VLAN>1</VLAN>
        </Interface>
          <Interface id="1">
          <Name>host 1</Name>
          <VLAN>1</VLAN>
        </Interface>
        <Interface id="2">
          <Name>host 2</Name>
          <VLAN>1</VLAN>
          </Interface>
        <Interface id="3">
          <Name>host 3</Name>
          <VLAN>2</VLAN>
        </Interface>
```

```
    </Interfaces>
    <VLANs>
      <VLAN id="1">
        <Name>Study</Name>
        <Tagged>0</Tagged>
        <Nountagged>3</Nountagged>
      </VLAN>
      <VLAN id="2">
        <Name>Employee</Name>
        <Tagged>0</Tagged>
        <Untagged>3</Untagged>
      </VLAN>
    </VLANs>
  <STP>
    <Instance id="1">
      <BridgePriority>16000</BridgePriority>
      <PortPriority id="1"></PortPriority>
      <ForwardDelay>5</ForwardDelay>
      <MaxAge>6</MaxAge>
      <HelloTimer>1</HelloTimer>
      <LinkCost id="1"></LinkCost>
    </Instance>
    <Instance id="2">
      <BridgePriority>16000</BridgePriority>
    </Instance>
  </STP>
</Switch>

<Host id="192.168.0.2">
              <Interfaces>
                      <Interface name="eth0">
                              <IPAddress>192.168.0.2</IPAddress>
                              <Mask>255.255.255.0</Mask>
                              <IPv6Address>2001:db8:a::11/64</IPv6Address>
                      </Interface>
              </Interfaces>
              <DefaultRouter>2001:db8:a::1</DefaultRouter>
</Host>
```

Příloha D

Použité zkratky

LLDP - Link Layer Discovery Protocol
CDP - Cisco Discovery Protocol
SNMP - Simple Network Management Protocol
VUT FIT - Vysoké učení technické Fakulta informačních technologií
SGMP - Simple Gateway Management Protocol
CMIP - Common Management Information Protocol
RFC - Request For Comments
DES/AES - Data Encryption Standart/Advanced Encryption Standart
NMS - Network Management System
MIB - Management Information Base
UDP - User Datagram Protocol
ASN.1 - Abstract Syntax Notation One
SMI - Structure and Identification of Management Information
OID - Object Identifier
PDU - Protocol Data Unit
IP - Internet Protocol
BER - Basic Encoding Rules
HMAC-MD5 - Hashed Message Authentication Code-Message Diggest
SHA - Secure Hash Algorithm
CBC - Cipher Block Chaining
CLI - Command Line Interface
API - Application Programming Interface
ODR - On Demand Routing
VTP - VLAN Trunking Protocol
VoIP - Voice over IP
CoS - Class of Service
MAC - Medium Access Control
XML - Extensible Markup Language
OMNeT++ - Objective Modular Network Testbed in C++
MPI - Message Passing Interface
CMDENV - Command enviroment
TKENV - Toolkit enviroment
Tk - Tool Command Language\ Toolkit
ANSA - Automated Netwrok Simulation and Analysis
NED - Network Description
TCP/IP - Transmission Control Protocol\ Internet Protocol
PPP - Point-to-Point Protocol
MPLS - Multiprotocol Label Switching
OSPF - Open Shortest Path First
BFS - Breadth-First Search Algorithm
LSA - Link State Advertisement
LSU - Link State Update

NDP - Network Discovery Protocol
RIP - Routing Information Protocol
ACL - Access Control List
eq - equal
neq - not equal
lt - little
gt - greater
VLAN - Virtual LAN
LAN - Local Area Network
STP - Spanning Tree Protocol