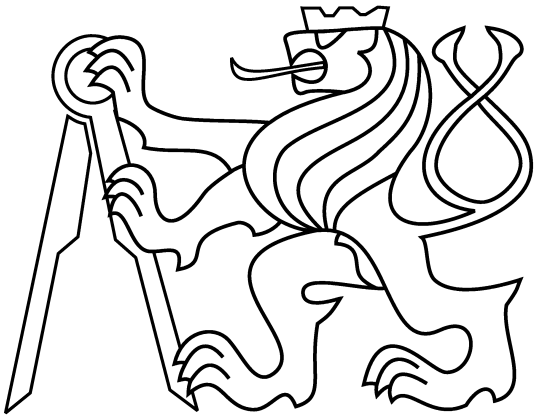
České vysoké učení technické v Praze
Fakulta elektrotechnická



DIPLOMOVÁ PRÁCE

Grafická administrace počítačové sítě

Praha, 2014 **Autor: Bc. Michal Hanzlík**

Prohlášení

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

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb. , o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne _____ _____

podpis

Poděkování

Děkuji především vedoucímu diplomové práce Ing. Pavlu Strnadovi, Ph.D. za pomoc s realizací této diplomové práce a také Ing. Viktorovi Černému. Děkuji také rodině a přátelům za podporu při práci.

Abstrakt

Cílem této práce je zjistit, jaké možnosti jsou pro vzdálenou správu zařízení MikroTik a jejich centrální správu s grafickou podporou. Hlavním cílem je navrhnout a vytvořit aplikaci, která by dynamicky vykreslila topologii počítačové sítě postavenou na zařízeních MirkoTik.

V této práci byly popsány neighbor discovery protokoly (NDP) a možnosti grafické správy počítačové sítě. Byly prostudovány možnosti vzdálené správy zařízení MikroTik a na tomto základě byla vytvořena aplikace pro dynamické vykreslení topologie počítačové sítě postavenou na zařízeních MirkoTik.

Abstract

The goal of this work is to present the existing solutions for remote management of MikroTik devices via graphical interface (GUI). The main goal is to develop a better application which is able to detect and present in GUI the network topology.

This work presents the class of Neighbour Discovery Protocols (NDP) and the ways a networks can be centrally managed via GUI. In this work the protocols for the remote management of MikroTik devices are presented and, based on those, the graphic tool for dynamically topology discover had been built.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V dnešní době, kdy téměř každý vlastní počítač, chytrý telefon nebo televizor, se všude vyskytují počítačové sítě ve větším nebo menším měřítku. V malých i velkých firmách to je nutností, doma to je spíše o pohodlí. V každém případě je důležitá dokumentace. Je důležitá při programování, při budování počítačové sítě nebo při jiných činnostech s určitým výsledkem. Bez správné dokumentace je poměrně těžké, ne-li nemožné, se v počítačové síti orientovat. Při předávání zodpovědnosti za stávající síť pak nový správce neví, který kabel kam vede a proč je na tom či onom směrovači (routeru) nějaké pravidlo.

O dokumentaci mluvím, protože se na ni často zapomíná. Absence dokumentace (nebo její neúplnost) je z velké většiny případů způsobena odkládáním jejího sepsání až na konec práce. Nebo nastanou chvíle, kdy se v topologii provede změna, která se následně nezanese do dokumentace (přidání prvku, spoje, nová služba, výměna zařízení, ...)

Problém nastává, když dokumentace není úplná a nemáme jasnou představu o tom, jak počítačová síť vypadá. Proto je potřeba takovou síť zrekonstruovat. Jednou z možností je začít u nějakého zařízení a ručně projít veškerou topologii počítačové sítě. To je poněkud pracné a zdlouhavé, zvláště pokud je počítačová síť rozlehlá. Lepší způsob je využít některých funkcí, kterými disponují různé aplikace, jako je například IP scan, route, tracert (traceroute) a ping.

Cílem mé práce je navrhnout a alespoň částečně realizovat aplikaci pro jednoduchou a centralizovanou správu sítě na zařízeních MikroTik. Aplikace by měla sama vyhledat tato zařízení a vykreslit je v co možná nejpřesnější topologii.

Kapitola 2

Současný stav

Následující kapitola obsahuje informace o "Neighbor Discovery Protokolu", který je stěžejní pro následující práci.

Další část je věnovaná softwaru, který se zabývá topologií sítě, monitoringem a dokumentací. Popisuje jejich základní vlastnosti, výhody a nevýhody.

Poslední část obsahuje způsoby vzdálené zprávy zařízení MikroTik. Jeden z těchto způsobů jsem musel zvolit pro připojení k zařízení, abych mohl vykreslit topologii.

2.1 Neighbor Discovery Protocol (NDP)

2.1.1 MikroTik Neighbor Discovery Protocol (MNDP)

MikroTik Neighbor Discovery Protocol (MNDP) (popsaný na [2]) je proprietární protokol společnosti MikroTik. Slouží k vyhledávání zařízení s RouterOS v síti a informací o něm. MikroTik RouterOS je schopné zjistit zařízení podporující MNDP, ale i CDP. Základní funkce MNDP je automatická konfigurace funkcí, které jsou dostupné mezi MikroTik směrovači. Momentálně je MNDP využíván pro funkci "Packet Packer". Tato funkce slouží k lepší propustnosti linky mezi MikroTik směrovači.

Vlastnosti MDNP:

- Je dostupný na rozhraních typu Ethernet a na rozhraních, které mají podporují IP protokol a mají alespoň jednu IP adresu.
- Ve výchozím stavu je MDNP zapnuto na všech rozhraních typu Ethernet - Ethernet, wireless, EoIP, IPIP tunnels, PPTP-server
- Používá UDP protokol na portu 5678.

- Router posílá UDP paket s informacemi o sobě broadcastem na L2 (linkové vrstvě ISO/OSI modelu) každých 60 vteřin.
- Router kontroluje, jestli jeho informace jsou aktuální každých 30 vteřin.
- Pokud nejsou přijaty žádné informace po dobu 180 vteřin, záznam o sousedovi se smaže.

2.1.2 Cisco Discovery Protocol (CDP)

Cisco Discovery Protocol (CDP) (popsaný na [22]) je proprietární protokol společnosti Cisco Systems. Je to protokol 2. síťové vrstvy ISO/OSI modelu nezávislý na mediu. Stejně jako MNDP je používán k sdílení informací o přímo připojených Cisco zařízeních. Tím, že poskytuje informace o PŘÍMO PŘIPOJENÝCH zařízení, je mnohem lepší pro dynamické vykreslení sítě. Dalším důvodem, proč je tento protokol lepší pro vykreslení sítě, je fakt, že Cisco zařízení nikdy nepřeposílá CDP pakety. Cisco zařízení posílá periodicky, každých 60 vteřin CDP zprávy na multicastovou cílovou MAC adresu 01:00:0C:CC:CC:CC. Konkrétní informace, které lze v CDP zprávě nalézt se liší podle zařízení, od kterého zpráva pochází.

Vlastnosti CDP:

- Cisco zařízení nikdy nepřeposílá CDP pakety.
- Cisco zařízení odesílá CDP paket s informacemi o sobě každých 60 vteřin.
- Router kontroluje, jestli jeho informace jsou aktuální každých 30 vteřin.
- Pokud nejsou přijaty žádné informace po dobu 180 vteřin (3 chybějící zprávy), záznam o sousedovi se smaže.

CDP zpráva obsahuje mimo jiné:

- Cisco IOS verzi na zařízení
- Platformu Hardwaru
- IP adresy na rozhraní zaízení
- Port-ID
- IP Network Prefix
- Jméno (hostname) zařízení

- Duplex setting
- Doménu VLAN Trunking Protokolu (VTP)
- Nativní VLAN

2.1.3 Link Layer Discovery Protocol (LLDP)

Link Layer Discovery Protocol (LLDP) je protokol 2. linkové vrstvy a je definován standardem IEEE 802.1AB [23]. Takže není závislý na prvcích od jednoho výrobce. Je používán síťovými prvky k sdílení informací o jejich identitě a funkcí s ostatními prvky v síti. LLDP je funkční na všech 802 mediích, typicky na Ethernetu. Všechny informace získané pomocí LLDP jsou uloženy v tabulce a je možné je vyčíst pomocí SNMP (Simple Network Management Protocol). Zařízení posílají zprávy (LLDPDU) periodicky nebo při změně stavu portu. Jako cílovou adresu se používá multicastová MAC adresa 01:80:c2:00:00:XX, kde XX je 03, 0E nebo 00.

LLDP zpráva obsahuje mimo jiné:

- Název a popis systému
- Název a popis portu
- VLAN
- IP adresu
- Schopnosti systému (přepínání, směrování,...)
- TTL = time to live

2.1.4 Neighbor Discovery Protocol (NDP) v IPv6

Neighbor Discovery Protocol (NDP) v IPv6 je definován v RFC 4861 [24]. Jeho vlastnosti popsal Pavel Satrapa v jeho publikaci [25]. Stejně jako všechny protokoly tohoto typu komunikuje na linkové vrstvě. Svým způsobem nahrazuje Address Resolution Protokol (ARP) v IPv4. Jedním z jeho úkolů je totiž zjišťování MAC adres uzlů ve lokální síti. Z IPv6 nahrazuje také ICMP Router Discovery (RDISC) a ICMP Redirect (ICMPv4). Navíc disponuje tzv. "detekce dosažitelnosti souseda". Nicméně tato dosažitelnost se využívá pouze v případě, kdy uzel komunikuje nebo chce komunikovat se sousedem.

2.2 Centrální správa sítě s grafickou podporou

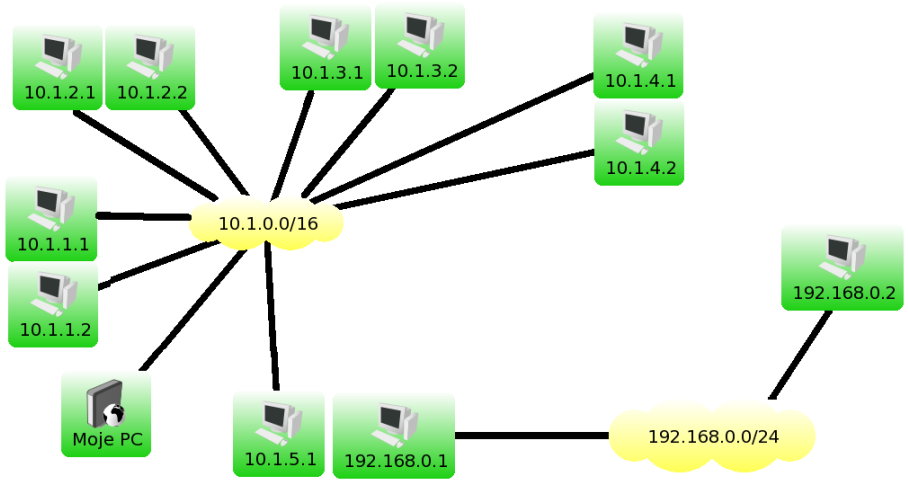
2.2.1 The Dude

The Dude je aplikace od společnosti MikroTik, která usnadňuje správu celé sítě a to především zařízení s RouterOS. Automaticky proskenuje síť na zadaném rozsahu, vykreslí nalezená zařízení do mapy a monitoruje jejich dostupnost. Dokáže monitorovat služby na zařízeních a upozornit, pokud některá služba nebo zařízení není dostupné. V tomto směru je velice podobný systému Nagios.

Funguje na síťové architektuře "klient-server". Dude existuje jako volitelný balíček do operačního systému RouterOS a jako aplikace do Windows. Po naskenování zařízení je možné vykreslenou mapu upravit a nastavit kontrolované služby na zařízení. Stejně tak je možné vytvořit vlastní zařízení, které třeba ještě není připojené do sítě. Umožňuje vkládání vlastních ikon (i vektorových) pro zařízení a vkládání obrázků na pozadí. Tím lze názorně zobrazit, kde jaké zařízení je umístěno.

Výhodou je také možnost nastavení "závislostí". Pokud selže síťový prvek A na cestě k zařízení B, C a D, pak Dude zahlásí nedostupnost pouze prvku A. Tím se nezahltí poštovní schránka zbytečnými zprávami. Pro získávání informací o prvku lze využít protokol SNMP a pokud jde o RouterOS, tak i další informace specifické pro takové zařízení. Je možné ho využít i jako vzdálený syslog.

Co se mi nelíbí na tomto programu je duplicita zařízení, které má více IP adres. Skenování jednotlivých IP adres nerozlišuje jedno zařízení s více IP adresami, což je typický směrovač. Tudíž směrovač mezi 5 sítěmi bude zanesen 5x a sítě budou odděleny. Výhodou je, že je zdarma.



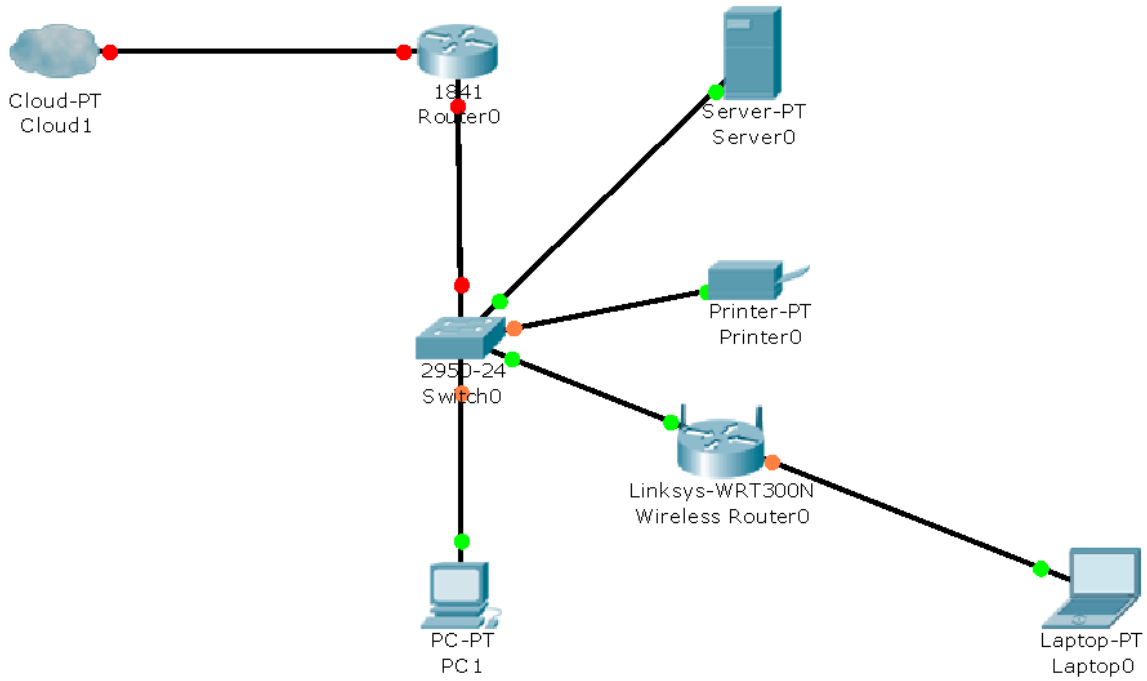
Obrázek 2.1: Ukázka programu The Dude

2.2.2 Cisco Packet Tracer

Cisco Packet Tracer je velice silný a rozsáhlý nástroj. Dokáže simulovat reálný provoz v počítačové síti. Umožňuje navrhnout a simulovat počítačovou síť složenou především ze síťových prvků značky Cisco. Mimo to nabízí propojení virtuální části sítě se skutečnou a tím usnadnit rozhodování při koupi nového zařízení do již stávající počítačové síě. Dokáže také sledovat provoz na síti a odchytávat pakety. Díky všem jeho vlastnostem dokáže tento software názorně ukázat možné problémy v počítačové síti a usnadnit tak hledání řešení.

Cisco Packet Tracer byl vytvořen společností Cisco za účelem podpory výuky v Cisco Networking Academy. Je volně dostupný jen pro registrované členy Networking Academy.

Nicméně tento program neumožňuje export do jiných formátů, tudíž není možné přenést v něm navrhnutou topologii počítačové sítě do jiného softwaru. Není ani možné vygenerovat/exportovat obrázek pro dokumentaci.



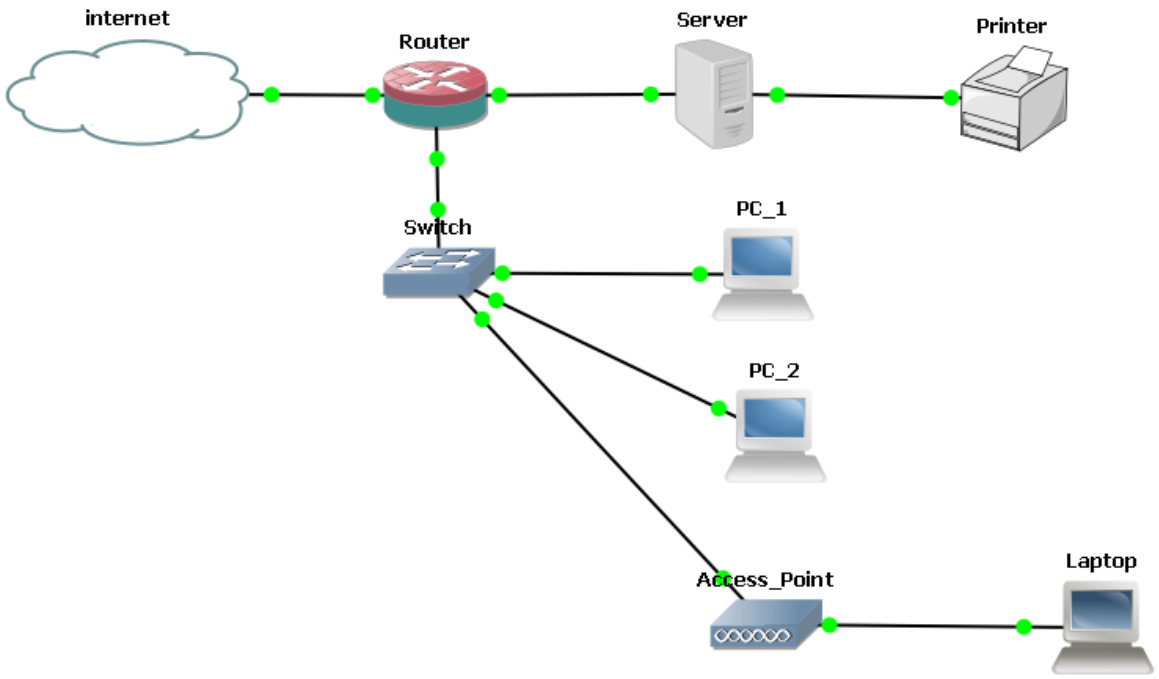
Obrázek 2.2: Ukázka programu Cisco Packet Tracer

2.2.3 Graphical Network Simulator (GNS3)

GNS3 je open source software, který dokáže simulovat komplexní počítačovou síť. Je vydávaný pod GPL v2 licencí a je dostupný v několika jazycích. Běží na všech známých operačních systémech (Windows, Linux a MacOS X). Na rozdíl od Packet Tracer-u k simulaci využívá více emulátorů: Qemu, Dynamips a VirtualBox. Je tedy výborným nástrojem pro výuku nebo testování. V případě použití VirtualBoxu má GNS3 velké nároky na CPU a na paměť hostovského počítače. Nároky na prostředí jsou závislé na konkrétním nastavení programu.

Ovládání je intuitivní a grafické zobrazení je podobné jako v Cico Packet Tracer. Ačkoli je více obecný, pro praktické použití vyžaduje IOS obrazy nebo virtualizované stroje.

Samozřejmostí je export obrázků do formátu PNG. Poskytuje zachytávání paketů a analýzu pomocí programu Wireshark [20]. Velkou výhodou je také možnost propojení virtuální sítě spolu s fyzickým zařízením (směrovač, přepínač, NAS, ...).



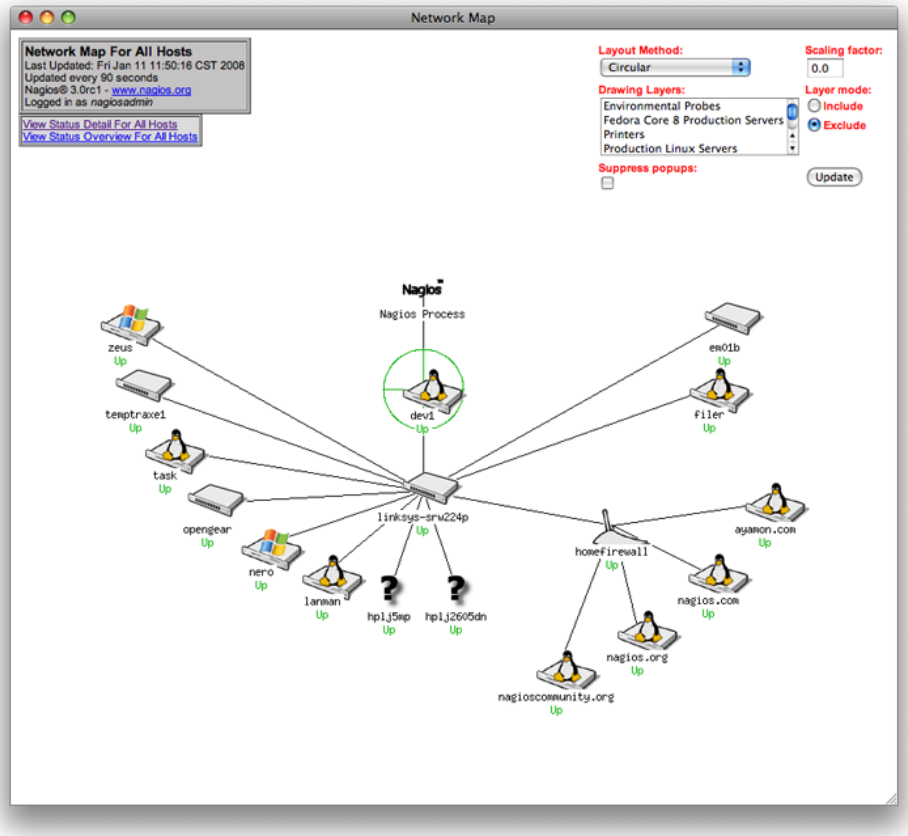
Obrázek 2.3: Ukázka programu GNS3

2.2.4 Nagios

Systémem Nagios je velmi používaným. Jedná se o monitorovací systém, který se musí bohužel ručně nastavit. Navíc konfigurace není nijak jednoduchá. Každé zařízení a služby,

které chceme monitorovat, se musí ručně zadat a ručně zakreslit topologii. Nagios podle topologie vykreslí mapu, na které lze přehledně vidět funkční a nefunkční zařízení. Orientuje se podle IP adres, z toho plyne, že pracuje na 3. vrstvě ISO/OSI modelu.

Při výpadku stroje, na kterém jsou spuštěny služby, pošle Nagios upozorňující zprávu o výpadku stroje a další zprávy o nedostupnosti všech služeb na tomto stroji. Na druhou stranu je přizpůsobitelný a rozšiřitelný. Je otestovaný i na velkých a složitých sítí. Disponuje velkým množství modulů.



Obrázek 2.4: Ukázka programu Nagios

2.3 Vzdálená správa zařízení

K zařízení s nainstalovaným RouterOS je možné se připojit několika způsoby. Velice známými způsoby jsou telnet, ssh nebo přes webové rozhraní (Webfig). Významnějším a praktičtějším způsobem připojení k tomuto operačním systému je pomocí API nebo pomocí jejich proprietární aplikace Winbox.

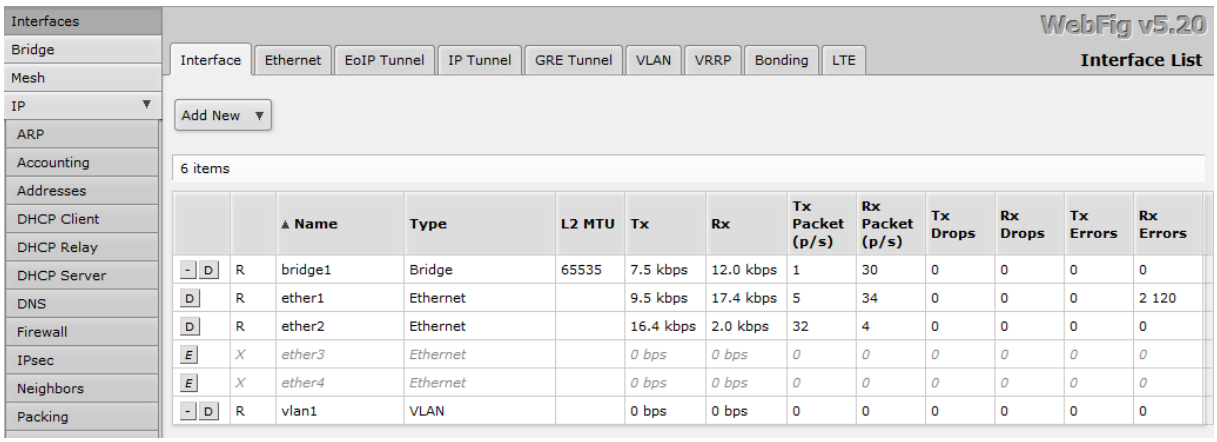
2.3.1 Telnet a SSH

Telnet je zkratkou Telecommunication Network. Je postaven nad protokolem TCP a používá spojení typu server-klient. Telnet standardně poslouchá na portu 23. Pro správnou komunikaci mezi oběma stranami je součástí tohoto protokolu také vzájemná domluva na některých dalších parametrech. Zásadním důvodem, proč není vhodné Telnet používat, je bezpečnost. Když byl Telnet vyvinut, nikdo nepředpokládal, že by mohl někdo tuto komunikaci odposlouchávat. Proto není Telnet nijak šifrován včetně hesla. Na druhou stranu je jednoduchý a rychlý. Proto se využívá v případě, kdy má správce přístup k zařízení například pomocí sériové linky.

Pro bezpečné připojení v síti byl vyvinut SSH (Secure Shell). SSH šifruje hesla a veškerou komunikaci, která je posílána. Využívá protokol TCP, stejně jako Telnet, a server standardně poslouchá na portu 22. Ani SSH není zcela bezpečné, pokud se nechováme bezpečně my. Proto je nutné, aby každý veřejný klíč vzdáleného zařízení byl řádně ověřen.

2.3.2 WebFig

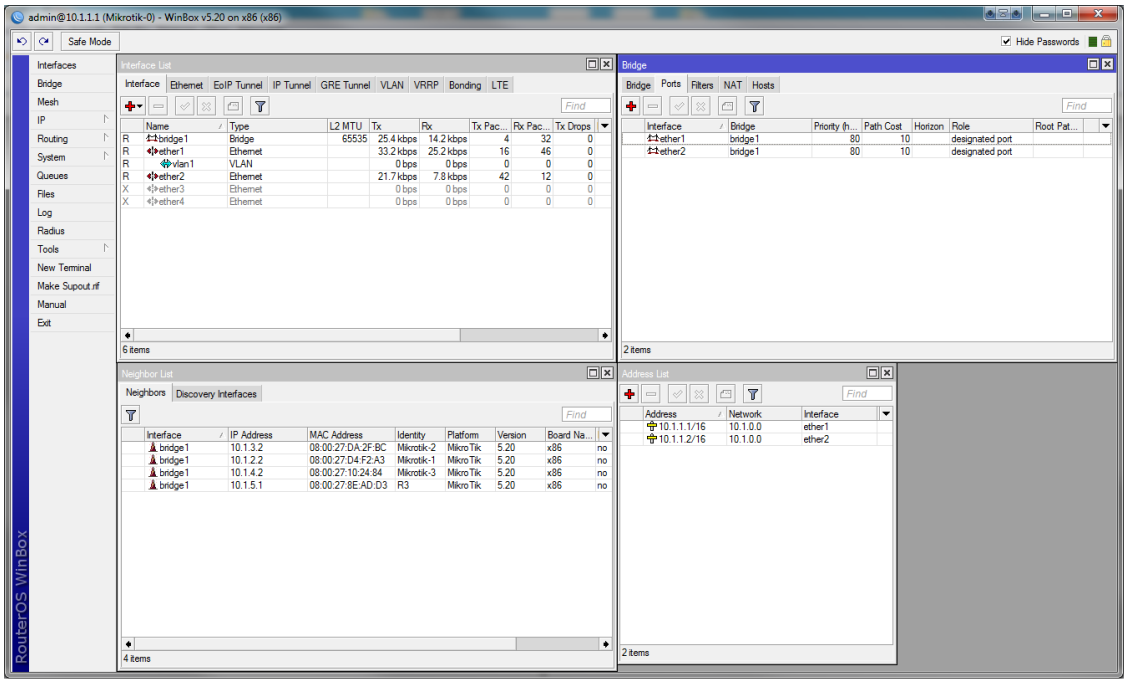
Dnešní uživatel často požaduje ovládat zařízení pomocí webové stránky. Má to hned několik výhod. Nemusí nic instalovat, má přístup odkudkoli a webový prohlížeč je pro něj známá věc, kterou používá denně. I pro zařízení s RouterOS existuje správa pomocí webového rozhraní zvaném WebFig. WebFig je doslova webovou kopii aplikace WinBox. K zařízení se přistupuje pomocí IP adresy zadané do adresního řádku webového prohlížeče.



Obrázek 2.5: Ukázka rozhraní WebFig

2.3.3 WinBox

WinBox je asi nejpoužívanější aplikací pro konfiguraci a monitoring těchto zařízení. Jedná se o malou aplikaci (jeden soubor) s jednoduchým grafickým rozhraní. Primárně je vytvořen pro operační systém Windows, ale funguje i na operačním systému Linux a MAC OSX pomocí Wine[21]. Wine je vrstva mezi aplikacemi pro OS Windows a Unix systémy. Rozhraní aplikace a konzolové rozhraní si jsou natolik podobné, že se každý uživatel, který umí v jednom, rychle zorientuje v druhém.



Obrázek 2.6: Ukázka rozhraní WinBox

2.3.4 API

API (Application Programmable Interface) dovoluje komukoli vytvořit aplikaci komunikující s RouterOS a zajišťující libovolné funkce, které vyžaduje. Syntaxe API je velice podobná syntaxi CLI (command line interface). API je dostupné až od RouterOS verze 3.x a vyšší. Důležitá je také poznámka, že služba API, poslouchající na portu 8728, je ve výchozím nastavení vypnutá. Princip komunikace s RouterOS je založena na zasílání tzv. "vět"(sentence) a přijímání jedné nebo více "odpovědí"(reply). Věta je sekvence "slov"(word). Každá věta musí jako první slovo obsahovat "příkaz"(command). Další slova mohou být v libovolném pořadí. Po přijetí celé věty ji RouterOS provede a vrátí

jednu nebo více odpovědí. Každé slovo musí být zakódováno určitým způsobem popsaným v manuálu na wiki [3].

Slova jsou rozdělena na:

- Command word
  - příkaz je totožný s CLI příkazem, který místo mezer obsahuje znak "/"
  - začíná znakem "/"
  - např:
    - /login
    - /ip/address/getall
    - /ip/neighbor/print
    - /interface/vlan/remove
- Attribute word
  - každý atribut začíná znakem "="
  - následuje jméno atributu
  - znak "=" a hodnota atributu
  - např:
    - =address=10.1.1.1
    - =name=moje jméno
- Query word
  - slouží k filtrování odpovědí. Analogií je WHERE v jazyce SQL
  - začíná znakem "?"
  - druhý znak je modifikátor: =, -, <, >
  - záleží na jejich pořadí!
  - např:
    - ?type=ether
    - ?>comment=
    - ?-comment

- Reply word
  - první slovo začíná znakem "!"
  - na každou větu RouterOS odpoví vždy alespoň jednou odpovědí
  - typy odpovědí:
    * !done - První slovo, poslední odpovědi na dotaz ("větu").
    * !re - vrácená data
    * !fatal - při ukončení spojení přes API, zašle router tuto zprávu spolu s důvodem uzavření spojení
    * !trap - problém a jeho kategorie
      1. missing item or command
      2. argument value failure
      3. execution of command interrupted
      4. scripting related failure
      5. general failure
      6. API related failure
      7. TTY related failure
      8. value generated with :return command

2.4 Výsledky a nedostatky existujících možností

V oblasti dostupného softwaru je jediná možnost pro centralizovanou správu a pro automatické vyhledávání zařízení MikrotTik. Je jím program The Dude od společnosti MikroiTik. Jak jsem psal v sekci o programu The Dude 2.2.1, je tento program velice propracovaným a užitečným nástrojem, který má mnoho funkcí. Nicméně má jednu zásadní věc, která je dle mého špatná. Je tím duplicita zařízení s více IP adresami. Další věc, která je podle mě zbytečná, je skenovat celý rozsah IP adres. Proto bych rád vytvořil aplikaci, která eliminuje duplicitu a nebude náhodně zkoušet IP adresy, které jsou dostupné v síti, ale využije NDP k systematickému vyhledávání zařízení. Výhodou mojí aplikace oproti programu The Dude bude, že nebude záležet, jaké podsítě zadám (pro skenování), ale nalezne všechna zařízení ve všech podsítí, která jsou dostupná.

Kapitola 3

Implementace

Pro vývoj aplikace potřebuji testovací prostředí, programovací jazyk a návrh rozdělení na podproblémy. Jako testovací prostředí byla zvolena simulace počítačové sítě 3.1 a pro implementaci programovací jazyk Java 3.2. Hlavní podproblémy jsou logická 3.3 a grafická 3.4 část aplikace.

Tato kapitola popisuje v první části simulační prostředí, které jsem potřeboval pro testování aplikace. Dále základní popis programovacího jazyka Java a důvody, proč jsem si tento jazyk zvolil pro implementaci mojí aplikace. Následuje implementace a problémy, se kterými jsem se potýkal v logické části. Poslední část je věnovaná grafickému zobrazení dat.

Ačkoli je zařízení Mikrotik a jeho operační systém RouterOS určen především ke směrování mezi sítěmi, není to tak vždy a jeho možnosti jsou rozsáhlé. Proto jsem při návrhu a následné implementaci nepředpokládal žádná speciální nastavení, která nejsou ve výchozím stavu. Tudíž jsem se nemohl spolehnout na dynamické směrování OSPF nebo BGP. Mým jediným předpokladem byla zapnutá služba API.

3.1 Simulační prostředí

K implementaci také patří vyzkoušení a otestování dané aplikace. Proto jsem potřeboval buď reálnou počítačovou síť postavenou na těchto zařízeních nebo si zřídit nějaké simulační prostředí s RouterOS. Protože získat přístup k takové reálné síti není lehké, využil jsem program GNS3 (Graphical Network Simulator) spolu s programem Oracle VM VirtualBox.

Jak jsem popsal výše, GNS3 je open source software, který dokáže simulovat komplexní počítačovou síť a je možné ho připojit i na fyzickou síť. S fyzickou sítí jsem propojení nepotřeboval, ale potřeboval jsem propojení s počítačem, na kterém aplikaci vyvíjím

(lokálním počítačem).

Oracle VM VirtualBox virtualizuje x86 hardware, takže cokoli lze spustit na této architektuře, lze spustit i ve virtualizačním nástroji VirtualBox. Stejně jako GNS3 tak i VirtualBox běží i na operačních systémech Windows, Linux a MacOS X.

Společnost Mikrotik poskytuje na svých webových stránkách instalační soubory operačního systému RouterOS pro různá hardwarová zařízení, mimo jiné i pro architekturu x86. Kombinací těchto 3 věcí jsem byl schopen simulovat malou virtuální síť, která mi byla nápomocna pro vývoj a testování mojí aplikace.

V programu VirtualBox jsem tedy vytvořil několik „počítačů“, na které jsem nainstaloval operační systém RouterOS. Minimální požadavky pro tento operační systém jsou: 32MB RAM a 64MB místa na disku. Následně jsem musel importovat virtuální počítače do programu GNS3, aby byly přístupné a mohl jsem tak ovládat tyto počítače a především je propojit mezi sebou, jak jsem potřeboval. Kdybych totiž použil pouze program VirtualBox, tak všechny počítače jsou v jedné podsíti.

3.2 Java

Rozhodl jsem se implementovat aplikaci v programovacím jazyku Java jako aplikaci běžící na lokálním počítači. Programovací jazyk Java je objektově-orientovaný programovací jazyk (tzv. OOP), který vyvinula společnost Sun Microsystem. Dnes je tento jazyk vyvíjen pod firmou Oracle známou především pro jejich databázové systémy. OOP je v dnešní době velice populární způsob jak psát aplikace, protože má oproti procedurálnímu programování spoustu výhod. Jmenuji jen ty, které považuji za nepodstatnější: znovupoužitelnost kódu, zapouzdření, dědičnost,... OOP se začalo využívat ve starších programovacích nebo skriptovacích jazycích jako je C++, PHP, JavaScript,...Výhodou jazyku Java je také jeho použitelnost na různých platformách. Aplikaci napsanou v jazyce Java je možné použít na všech zařízení, která mají nainstalovanou Java Runtime Enviroment (JRE), které je dostupné pro platformu Windows, Unix i Mac.

3.3 Logická část

3.3.1 Připojení k RouterOS

Pro připojení se zařízením MikroTik s operačním systémem RouterOS jsem se rozhodl využít službu API, která je nejvhodnější pro komunikace vlastní aplikace a RouterOS, a pro které jsou napsány knihovny pro různé programovací jazyky – Java, PHP, C,

Python, Perl, Delphi ... Chtěl jsem tedy použít knihovnu pro Javu dostupnou na stránkách společnosti MikroTik od autora „janisk“. Bohužel jsem při pokusu o první komunikaci zjistil, že tato knihovna neumožňuje spojení se zařízením. Naštěstí je na téže stránce zdrojový kód této knihovny a při hlubším prozkoumání jsem zjistil, že knihovna využívá metodu InetAddress.isReachable(int timeout), která mi při několika spuštění vždy vrátila “false”, tudíž se nikdy nespojí. Na tuto chybu jsem našel na internetu diskuze, ale žádná neměla vysvětlení proč se tak děje, natož jak ji opravit.

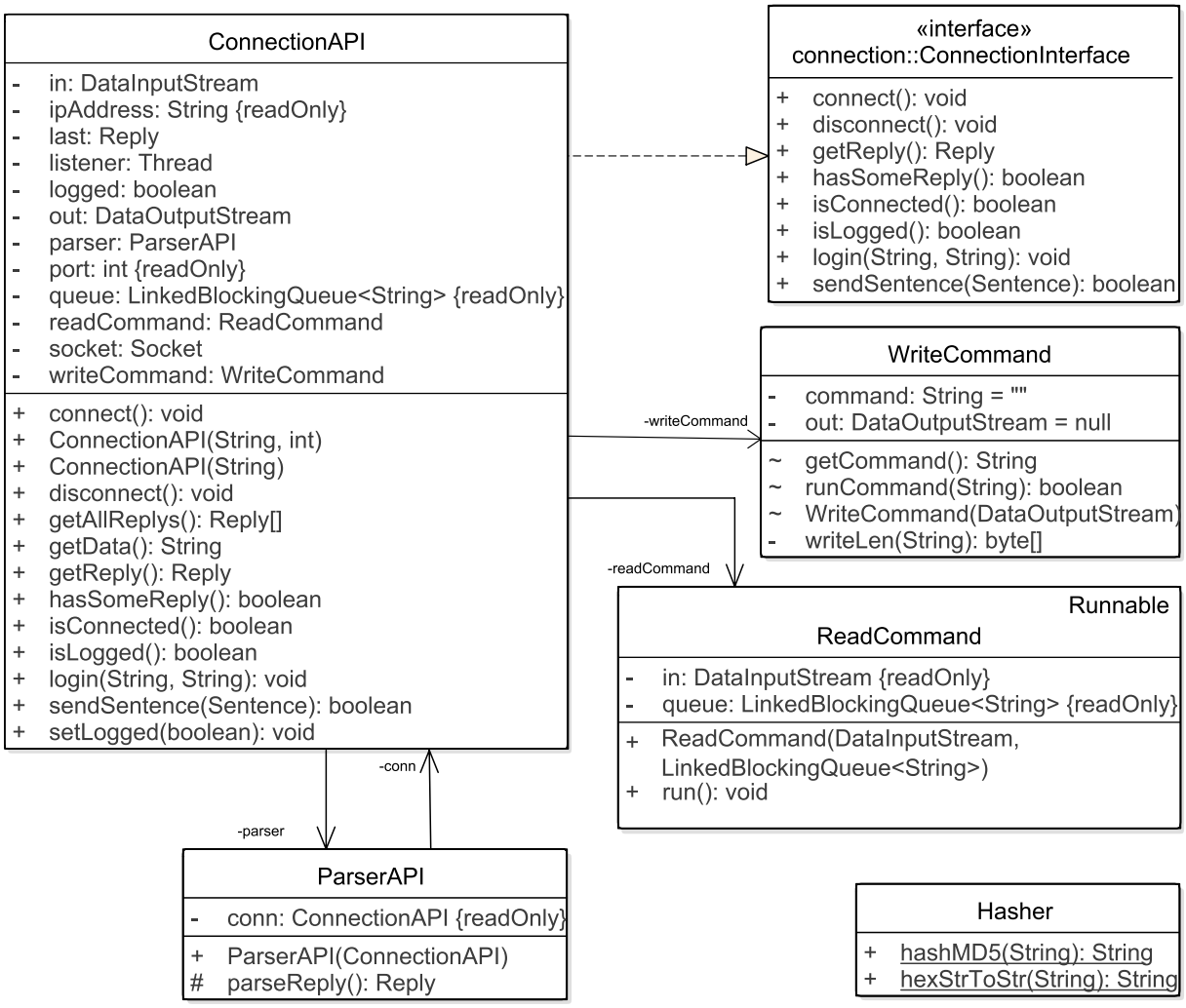
Knihovna rovněž obsahovala strukturu, kterou jsem shledal zbytečně složitou a nelogickou a obsahující zbytečné metody a proměnné. Tudíž jsem se rozhodl ji přepsat tak, aby byla jednodušší, přehlednější a vyhovovala mým potřebám. Ve výsledku jsem přepsal téměř celý kód při zachování myšlenky. Avšak namísto 2 speciálních vláken pro jedno spojení využívám pouze jedno (pro příjem dat) a strukturu jsem zjednodušil ze 4 souborů na 2. Jediná třída, kterou jsem využil beze změny, je *Hasher.java*, která obsahuje funkce pro hashování zpráv při přihlašování do zařízení. Důvodem sloučení těchto tříd je jejich vzájemný vztah a logické uspořádání. Každá instance spojení (*ConnectionAPI*) využije právě jednu instanci ostatních tříd a zároveň tyto třídy nebudou potřeba v jiné části aplikace. Proto je možné je zpřístupnit pouze pro třídu *ConnectionAPI* (oproti původnímu návrhu).

Na obrázku 3.1 je vidět struktura komunikace pomocí API. Třída *ConnectionAPI* vytváří při připojení socket (na zadanou IP adresu a port), instanci třídy *ParserAPI* (pro zpracování odpovědí) a vlákno („listener“), které vrácené odpovědi od RouterOS uloží do fronty. „Listener“ je instancí třídy *ReadCommand* a pro odesílání se používá třída *WriteCommand*.

Pro komunikaci jsem navrhl a vytvořil soubor tříd, která odpovídá popisu komunikace pomocí API. Jak zobrazuje obrázek 3.2, struktura obsahuje 2 základní třídy – *Reply* a *Sentence*. Reply je odpověď od zařízení s RouterOS a má svůj typ. Typ odpovědí je konečný a proto je implementován pomocí „enumeration“. Obsahuje seznam instancí třídy *Attribute*. *Sentence* je třída reprezentující „větu“, která se posílá zařízení na zpracování. Obsahuje „slova“, která jsou typu *Command*, *Attribute* nebo *Query*. *Třída Word* je abstraktní, protože obsahuje abstraktní metody getAPISyntax a getCLISyntax. Důvod je zřejmý, každý typ slova má jinou syntaxi.

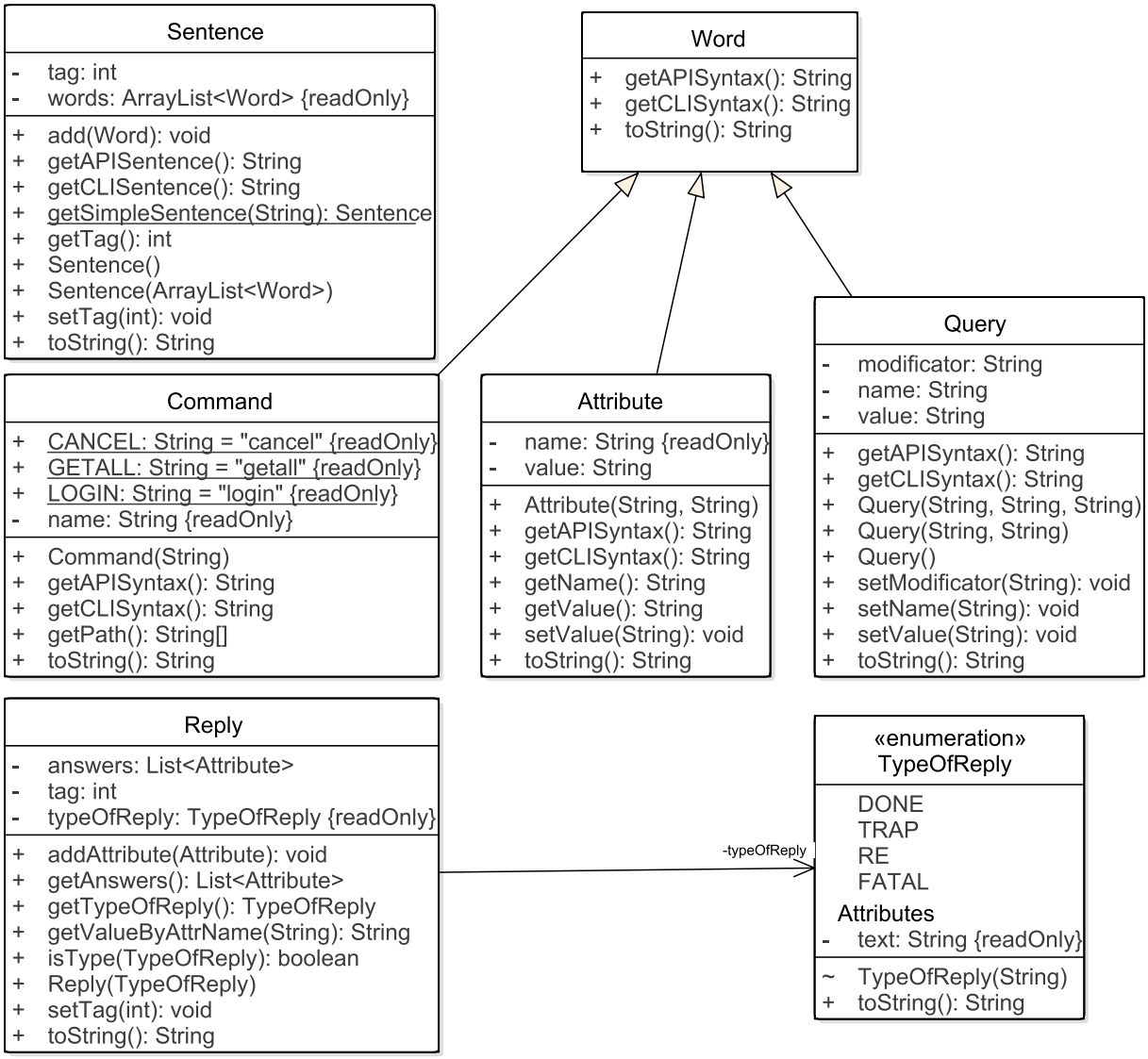
3.3.2 RouterOS a rozhraní

Implementovat celý operační systém RouterOS není cílem této práce, i když by to velice pomohlo. Nicméně jsem potřeboval vytvořit třídu, která bude reprezentovat zařízení



Obrázek 3.1: Struktura komunikace pomocí API

s RouterOS. K tomu slouží třída *RouterOsDevice*. Ta obsahuje základní informace o zařízení – identitu, (unikátní) software ID, platformu, verzi systému, router board model, přístupovou IP, přihlašovací údaje, instanci třídy *ConnectionInterface* pro připojení a HashMapu IP adres a rozhraní.



Obrázek 3.2: Struktura komunikace

Z metod zmíním jen ty nejdůležitější. Metoda equals(...), která porovnává klíčové software ID a identitu zařízení. Ta se využívá při přidávání do seznamu (přesněji množiny) všech zařízení. Metodu connect(...), která slouží k připojení, k získání instance rozhraní podle jména slouží metoda getInterfaceByName(...).

Velkým problémem, který jsem řešil, bylo vytvoření struktury rozhraní. Společných (využitelných) věcí mají totiž velice málo – jméno a typ. Pouze částečně jsem si ulehčil práci tím, že jsem se zaměřil na ty základní - bridge, ethernet, wlan, vlan. Opět jsem tedy musel vytvořit soubor tříd reprezentující tuto skutečnost. Základem je třída *Basicinterface* obsahující parametr *name* (název rozhraní), *type*, *device*, *MAC*, *comment* a *disabled*. Parametr *name* je název rozhraní, *device* je zařízení, ke kterému rozhraní patří, *MAC* obsahuje řetězec MAC adresy, *type* je instance *InterfaceType* říkající, o jaký typ rozhraní jde a *comment* spolu s *disabled* nejsou nijak podstatné, ale jsou jen pro úplnost. Struktura těchto tříd je zobrazena na obrázku "Struktura rozhraní"3.3.

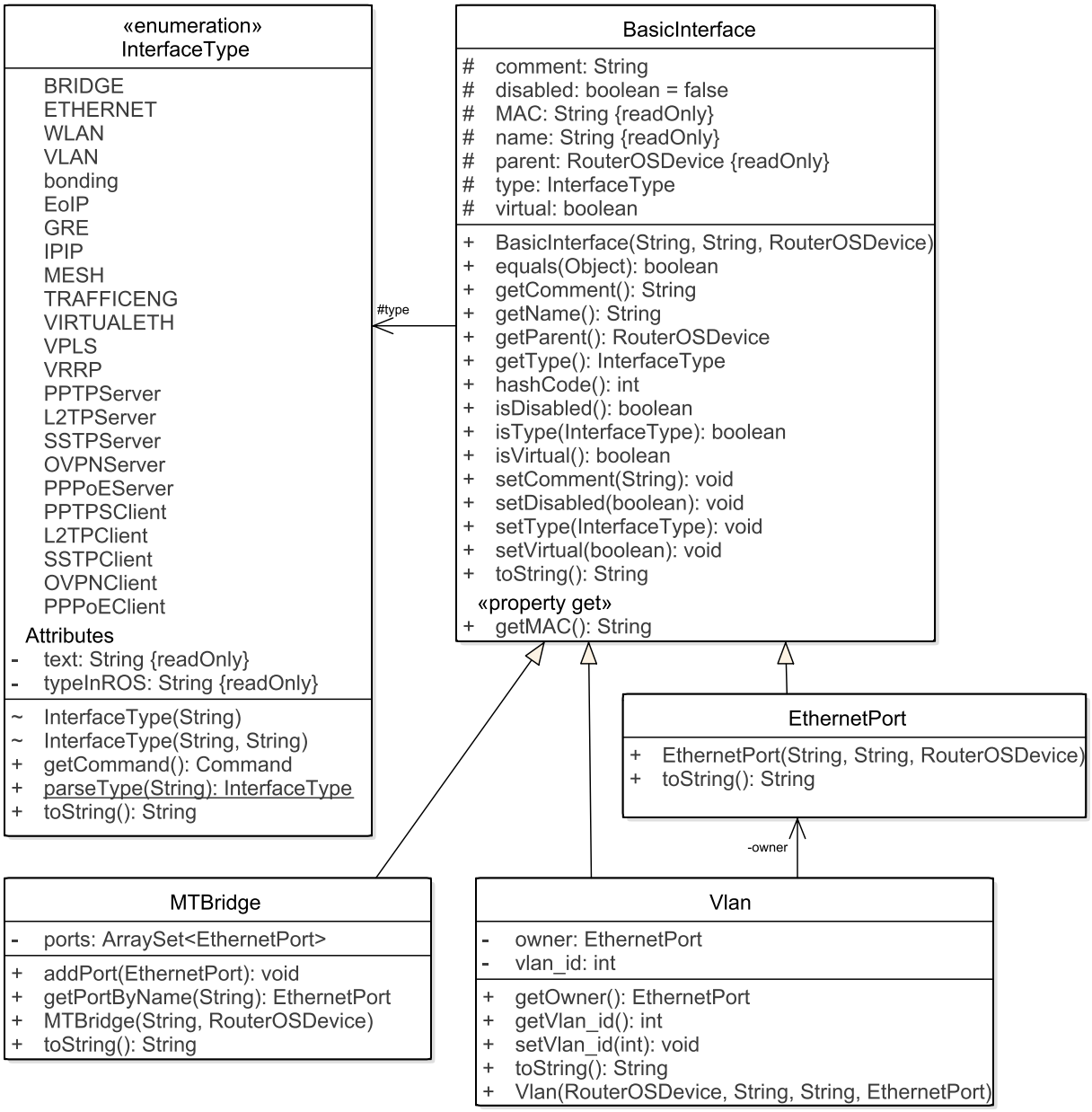
3.3.3 Načítání účtů

Abych se mohl připojit na zařízení s RouterOS, potřebuji k tomu přihlašovací údaje. Výchozím nastavením je přihlašovací jméno "admin" a prázdné heslo. V praxi samozřejmě nebude mít účet admin prázdné heslo, ale také ne každý bude mít přístup jako admin. V celé síti také nemusí mít jeden uživatel vždy stejné heslo, i když to je pravděpodobné. Zvláště v případě, kdy RouterOS může ověřovat uživatele na RADIUS serveru. Do souboru *accounts.csv* se mohou přidávat přihlašovací údaje oddělené středníkem, které se načítají do aplikace. K načítání využívám ověřenou a jednoduchou knihovnu CSVReader [4]. Tu také používám k zaznamenávání některých klíčových událostí do logu v adresáři "logs".

3.3.4 Prohledávání sítě

Nejdůležitější částí mojí aplikace je prohledávání sítě. Představa byla poměrně jednoduchá: "Zeptám se jednoho zařízení, jaké má sousedy, zapíšu a zeptám se dalšího." Tato zjednodušená představa má několik problémů.

Smyčky – v případě, že je v síti smyčka, ptal bych se stále dokola a nikdy bych se nezastavil. Stejně tak, když mi zařízení A vrátí souseda B, tak mi zařízení B vrátí souseda A. Tudíž potřebuji nějaký ukazatel, který mi řekne, že jsem se daného zařízení už ptal. Jedním z řešení je vytvořit si na zařízení soubor nebo nastavení, které nikdy nikdo nepoužije. Další způsob, který mě napadl byla kombinace parametrů, třeba IP adres. Takové zařízení bych si musel uložit, připojit se na něj, vyčíst informace a pak bych



Obrázek 3.3: Struktura rozhraní

zjistil, jestli jsem na tomto zařízení už byl. Po prozkoumání stránek firmy MikroTik jsem zjistil, že unikátním identifikátorem RouterOS je jeho tzv. „software ID“.

Důležitá je také informace, co nám vrací RouterOS na otázku „Jaké máš sousedy?“ Takto vypadá jedna odpověď:

```
!re
=.id=*2
=interface=bridge1
=address=10.1.2.2
```

```
=mac-address=08:00:27:D4:F2:A3
=identity=Mikrotik-1
=platform=MikroTik
=version=5.20
=unpack=none
=age=00:00:37
=uptime=00:00:19
=software-id=W5EY-LHT9
=board=x86
=ipv6=false
=interface-name=most1
```

- !re - označuje odpověď
- .id – číslo odpovědi
- interface – název lokálního rozhraní, ke kterému je toto zařízení připojeno
- address – IP adresa vzdáleného zařízení
- mac-address – MAC adresa vzdáleného zařízení
- interface-name – jméno rozhraní na vzdáleném zařízení
- software-id – software ID vzdáleného zařízení (pouze u RouterOS)
- identity – název vzdáleného zařízení
- platform – platforma vzdáleného zařízení (MikroTik, Cisco, AG5, ...)
- version – verze OS vzdáleného zařízení
- board - RouterBoard model (pouze u RouterOS)
- unpack – typ komprese discovery paketu
- age – čas od posledního přijetí discovery paketu
- uptime – doba spuštění vzdáleného zařízení (pouze u RouterOS)

Bohužel tyto informace jsou někdy zavádějící. Protože jsem celou simulovanou síť nastavil, vím, že zařízení Mikrotik-1 nemá na rozhraní s názvem most1 žádnou IP adresu.

Ale rozhraní most1 zahrnuje rozhraní ether1 a ether2 a právě jedno toto rozhraní má tuto IP adresu a MAC adresu.

Všechny implementace Neighbor Discovery Protokol vracejí sousedy na 2. linkové vrstvě ISO/OSI modelu. Je jedno, jestli jsou všechna zařízení zapojena do jednoho centrálního přepínače (switch), nebo za sebou a jejich porty nastavena do mostu (bridge) nebo libovolná kombinace. Při stejném nastavení zařízení s RouterOS a různém zapojení dávají dotazy na sousedy totožné výsledky na každém zařízení. Proto mi tyto informace k vykreslení topologie nestačí. K tomu potřebuji získat informace, jaká zařízení (a kolik jich je) jsou připojena na konkrétní fyzický port. Pokud RouterOS nemá zařazeny žádné porty do rozhraní bridge ani spojené porty do přepínače (switch-e), není to problém, ale mým cílem bylo nepředpokládat konkrétní nastavení ani omezení. Proto na tuto možnost nemohu spoléhat.

Jako možné řešení se jevilo získat informace z rozhraní "bridge". Při správném dotazu získám informaci, které MAC adresy jsou připojeny ke konkrétnímu fyzickému ethernet-portu. V mém malém simulovaném prostředí o několika málo zařízení se to zdálo jako možné řešení. V praxi však je počítačová síť daleko rozsáhlejší a seznam MAC adres je dlouhý. Z tohoto seznamu MAC adres navíc nezjistím, o jaké zařízení se jedná (RouterOS, Cicso, TP-LINK,...). Toto je tedy zásadní problém, proč jsem nevytvořil aplikaci, jakou jsem zamýšlel. Chtěl jsem vytvořit aplikaci, aby pravdivě zobrazila spojení mezi jednotlivými zařízeními podle toho, které fyzické porty jsou skutečně zapojeny. Tento problém je problém grafu, který lze vyřešit, ale není to obsahem této práce. Proto jsou v mé práci uložena spojení "každý s každým"v rámci jedné podsítě. Pro přehlednost je vykreslení zjednodušeno na spojení se "sítí".

Nalezená zařízení jsem zpočátku ukládal do struktury *ArrayList*. Následně jsem si uvědomil, že každé zařízení může být v systému pouze jednou. Tudíž je vhodnější struktura *Set* (množina). Bohužel přes struktury Set nebo Map nelze iterovat a zároveň do nich přidávat (*oproti ArrayList*). Tudíž jsem potřeboval strukturu s vlastnostmi jako má *ArrayList* a zároveň aby neumožňovala duplicity jako struktura Set – takový *„ArraySet“*. Naneštěstí Java SE neobsahuje takovou strukturu. Po kratším hledání jsme našel několik implementací ArraySet-u:

`edu.stanford.nlp.util.ArraySet<E>`, [8]

`com.ibm.wala.util.collections.ArraySet<T>`, [9]

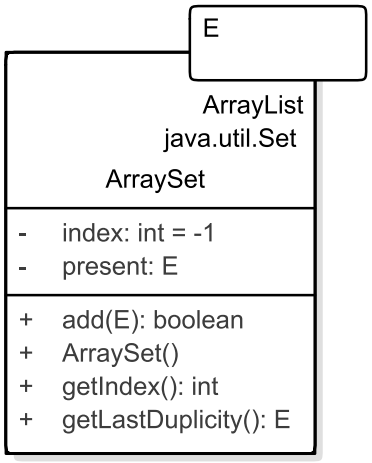
`soot.util.ArraySet<E>`,[10]

`org.slim3.util.ArraySet<E>`[11]

Nakonec jsem se rozhodl vytvořit vlastní implementaci, o které budu přesně vědět, jak se chová a mohu si jí v případě potřeby dále upravovat. Moje implementace třídy

ArraySet dědí třídu *java.util.ArrayList* a implementuje interface *java.util.Set*. (viz obr. 3.4)

- boolean add(E) - Přidává element E právě tehdy, když daný element ještě vložen nebyl.
- E getLastDuplicity() - vrací instanci posledního elementu, který byl vložen podruhé
- int getIndex() - vrací index posledního elementu, který byl vložen podruhé



Obrázek 3.4: Třída ArraySet<E >

Nyní popíšu celý algoritmus vyhledávání. Musím začít od známého zařízení. Vytvořím první instanci třídy *RouterOSDevice*, přiřadím mu přístupovou IP adresu a přidám ho do seznamu *devices* typu ArraySet. První, co musím zjistit, jestli je IP adresa zařízení dostupná. Pro kontrolu dostupnosti se nabízí použít metoda InetAddress.isReachable(timeout), ale (jak jsem psal výše) není spolehlivá. Proto jsem se uchýlil k volání systémové funkce Runtime.getRuntime().exec(cmd). Parametr *cmd* je samozřejmě upraven podle operačního systému na příslušný tvar. Pokud zařízení není dostupné, přeskočím ho a pokračuji v dalším. Poté vyzkouším možnosti přihlašovacích údajů načtené z CSV souboru *accounts.csv*. První přihlašovací údaje, se kterými se přihlásím do zařízení uložím k příslušné instanci zařízení pro pozdější použití.

Pokud aktuální zařízení nemá informaci o parametru software ID, pak zavolám funkci, která získá základní informace o zařízení a uloží je. Jako základní informace o zařízení považuji název (identitu), klíčové software ID, verzi systému platformu a tzv. board. To se týká především prvního zařízení, protože o něm z počátku vím jen IP adresu. U ostatních tyto informace získám pomocí MNDP z odpovědi na sousedy.

Abych získal potřebné informace k vykreslení topologie, volám postupně následující funkce: getInterfaceList(device), getAddresses(device) a getNeighbors(device).

3.3.5 Získání rozhraní

Získání informací o rozhraní je komplikovanější. Protože je poměrně velké množství typů rozhraní, nelze je moc dobře generalizovat a se získávání informací to není lehčí. Pro každý typ rozhraní se musím ptát zvlášť. Proto jsem zvolil následující postup:

1. Zeptám se, jaký typ rozhraní je zapnutý a je používán.
2. Zeptám se na jednotlivé typy rozhraní, získám o nich informace a uložím k zařízení.
3. Protože "bridge" obsahuje jiná rozhraní, přiřazuji každému existující a už uložená.

K získání všech rozhraní vytvořím instanci třídy Sentence s příkazem `"interface print"`. A protože mě zajímají pouze rozhraní, která se opravdu využívají (tedy ta co nejsou vypnutá), přidám slovo typu Query, parametry `name="disabled"` a `value="false"`. Tak jak je uvedeno v následující ukázce 3.1.

Listing 3.1: Věta pro získání rozhraní

```
Sentence sentence = Sentence.getSimpleSentence("interface print");
sentence.add(new Query("disabled", "false"));
```

3.3.6 Získání IP adres

Původně jsem chtěl každému rozhraní přiřadit IP adresu. Jenže každé rozhraní nemusí mít IP adresu. Proto jsem se rozhodl uchovávat IP adresy stejně jako to mají v RouterOS. Jednoduše seznam dvojic IP adresa a rozhraní. Současně ukládám IPv4 adresu, síť a masku, do které tato IP adresa patří. To vše je uloženo za pomocí třídy *IPv4*.

3.3.7 Získání sousedů

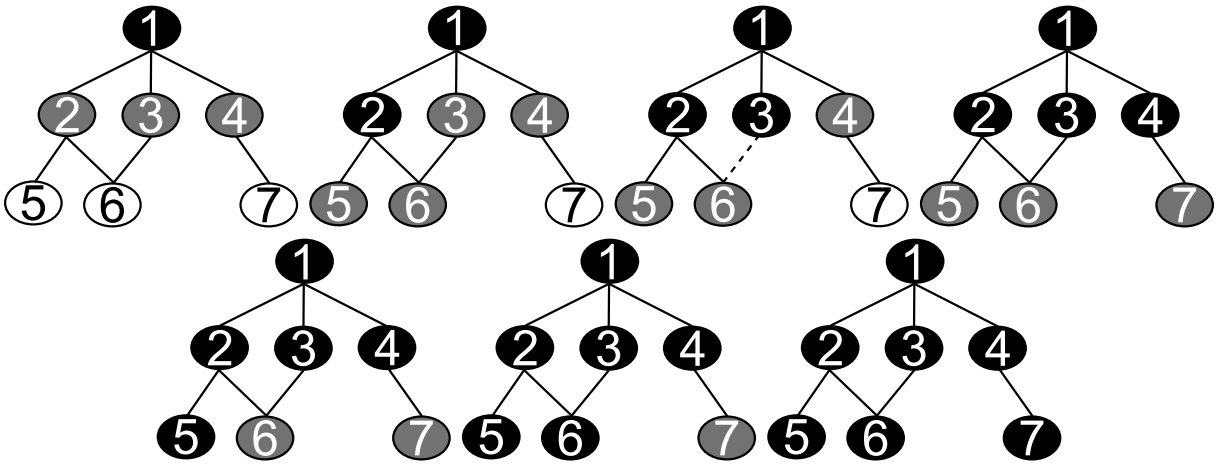
Základním kamenem výsledné aplikace je získávání sousedů. V momentě, kdy jsem připojen na zařízení s RouterOS zeptám se ho pomocí příkaz `ip neighbor print`. Tím získám všechna zařízení, která jsou kompatibilní s MNDP. Protože mě zajímají jenom zařízení s RouterOS, která se hlásí s parametrem `platform=MikroTik`, využiji opět Query slovo s tímto filtrem (viz. 3.2)

Listing 3.2: Věta pro získání sousedů

```
Sentence soused = Sentence.getSimpleSentence("ip neighbor print");
soused.add(new Query("platform", "MikroTik"));
```

Z odpovědi si vyberu ty části, které potřebuji a vytvořím z nich novou instanci třídy *RouterOSDevice*. Tuto novou instanci se pokusím přidat do seznamu všech zařízení. Pokud se přidá, jedná se o nové (dosud neznámé) zařízení, jinak toto zařízení už v systému mám a uložím spojení mezi těmito zařízeními.

Postup procházení je totožný s procházením stromu do šířky. K popisu použiji obrázek 3.5, Bílé prvky jsou neznámé, černé už prošlé a šedivé ty, co jsou další v seznamu. První prvek, který je přidán do seznamu, je kořenem stromu (číslo 1). Další prvky jsou ty, co sousedí s kořenem (vráceny v náhodném pořadí). Po kořenu se přejde na druhý prvek a jeho sousedi se přidají na konec seznamu. Ve třetím kroku se vytvoří pouze spojení mezi prvkem 3 a 6 (na obrázku znázorněné čárkovaně), protože prvek 6 je již v seznamu obsažen.



Obrázek 3.5: Prohledávání grafu do šířky

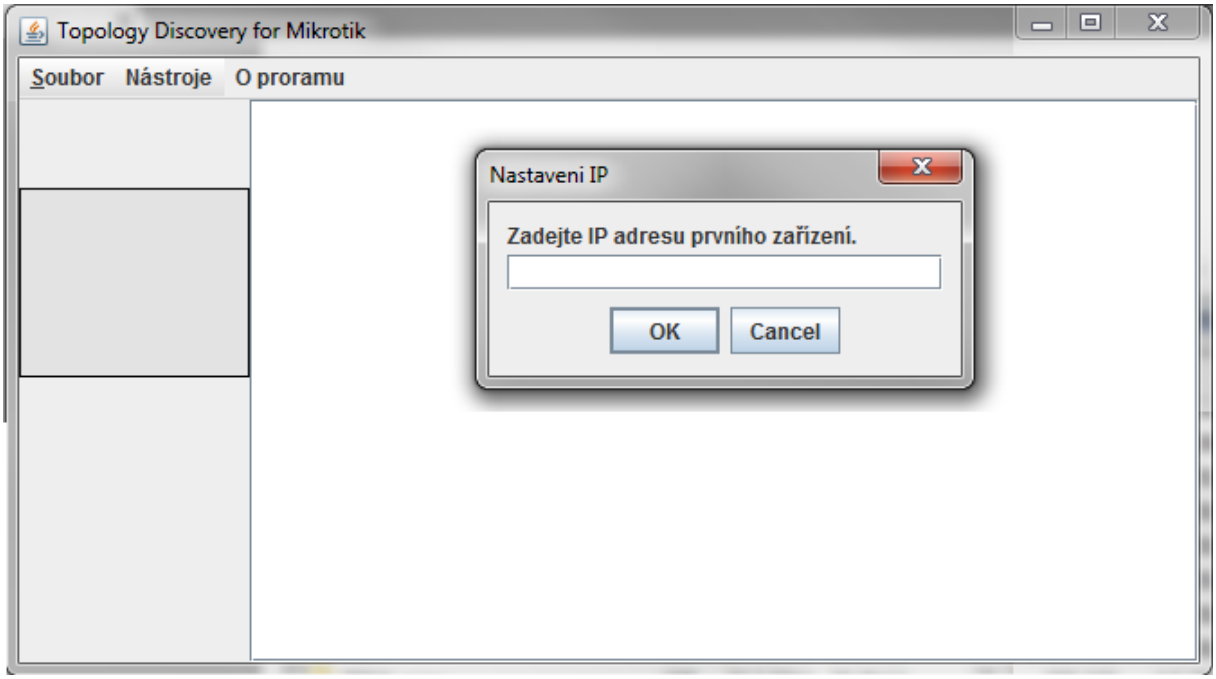
3.4 Grafická část

Použitelnost aplikace určuje rozhraní, které uživatel používá. To bývá nejčastěji grafické. Problém vytvoření GUI jsem si rozdělil na 2 části. Prvním je samotné okno, menu, ovládací prvky, nastavení atp. Druhá část je vykreslení topologie.

Pro grafickou část mojí aplikace jsem mohl využít tzv. "GUI Builder"(například vestavěný do IDE). Nebo si ho napsat sám ručně. Protože vím, jak pracuje takový "GUI

Builder", rozhodl jsem se spolehnout na vlastní zkušenosti. GUI Builder totiž vytváří spoustu zbytečného kódu, který je nepřehledný, zbytečně dlouhý a těžko pochopitelný.

Vytvořil jsem základní okno typu *JFrame*, tomu jsem přiřadil lištu s menu typu *JMenuBar* a contentPane typu *JPanel*. ContentPane obsahuje instanci třídy *JScrollPane*, aby se při velkém množství zobrazovaných dat mohly postupně zobrazit všechny. Celé to vypadá jako na obrázku 3.6.



Obrázek 3.6: Hlavní okno

Pro vykreslení topologie jsem usoudil, že vlastní implementace by sice možná byla, ale daleko výhodnější a efektivnější je využít nějakou již napsanou grafickou knihovnu. Tyto knihovny jsou totiž již otestované a prověřené uživateli. Navíc disponují nástroji a nastavením, které by mi v případě vlastní implementace zabraly zbytečně mnoho času.

3.4.1 Dostupné grafické knihovny

Lightweight Java Game Library (LWJGL) [12] je knihovna zaměřená na počítačové hry. Umožňuje programátorům přístup k jiným knihovnám dostupných na všech operačních systémech, jako OpenGL (Open Graphics Library), OpenCL (Open Computing Language) nebo OpenAL (Open Audio Library). Je vhodné ji použít i při 3D hrách. LWJGL je dostupná pod licencí BSD, což znamená, že je open source a volně použitelná jak pro další open source projekty, tak i pro komerční. Za zmínku stojí několik her, které tuto knihovnu

používají: Starfarer, Illarion, Sokobano, Cultris II a populární Minecraft.

Apache^{TM} Batik SVG Toolkit [13] je grafická knihovna pro aplikace nebo aplety napsané v programovacím jazyce Java, ve kterých je požadováno použití obrázků ve formátu SVG (Scalable Vector Graphics). Přesněji řečeno je tato knihovna určena pro manipulaci s SVG formáty. Obsahuje nástroje pro tvorbu a editaci SVG a export do jiných bitmapových formátů jako BMP, JPEG, PNG aj. Je tedy vhodná pro vytvoření grafického editoru jako je například Inkscape [18]. Proto není vhodná do mojí aplikace.

Graphical Editing Framework (GEF) [14] je open source framework poskytující grafické editovatelné prostředí pro aplikace na platformě eclipse. Momentálně je vyvíjen ve dvou verzích - hlavní a novější se nazývá GEF4. Starší verzi už nebudou nadále rozvíjet. GEF je vydáván pod licencí Eclipse Public License v1.0. Ukázky této knihovny jsou dobře použitelné a bylo ji možné použít. Avšak moje aplikace není na platformě Eclipse, proto jsem tuto knihovnu nevyužil.

OpenJGraph [15] je knihovna pro vytvoření grafu s možností manipulace a editace. Umožňuje vytvořit orientovaný i neorientovaný graf, manipulaci a upravovaní hran i uzlů. Obsahuje i některé algoritmy jako např. hledání nejkratší cesty, hledání kostry grafu, aj. Knihovna je vydávána pod licencí GNU Library nebo Lesser General Public License version 2.0 (LGPLv2).

Jung - Java Universal Network/Graph Framework [16] - je další z řady open source knihoven. Umožňuje vytvořit a zobrazit data, která reprezentují graf nebo počítačovou síť. Obsahuje sadu algoritmů pro rozložení (Kamada-Kawai, Fruchterman-Reingold, Meyer's "Self-Organizing Map"layout, náhodné rozložení po kružnici). Tato knihovna je vhodná i pro moji aplikaci. Výhodu je velké množství ukázek.

Vhodným kandidátem byla také knihovna yFiles [17]. Poskytuje algoritmy a komponenty pro analýzu, vizualizaci a automatické uspořádání grafu, diagramu a sítí. Má také celou sadu příkladů a ukázek, jak s touto knihovnou pracovat.

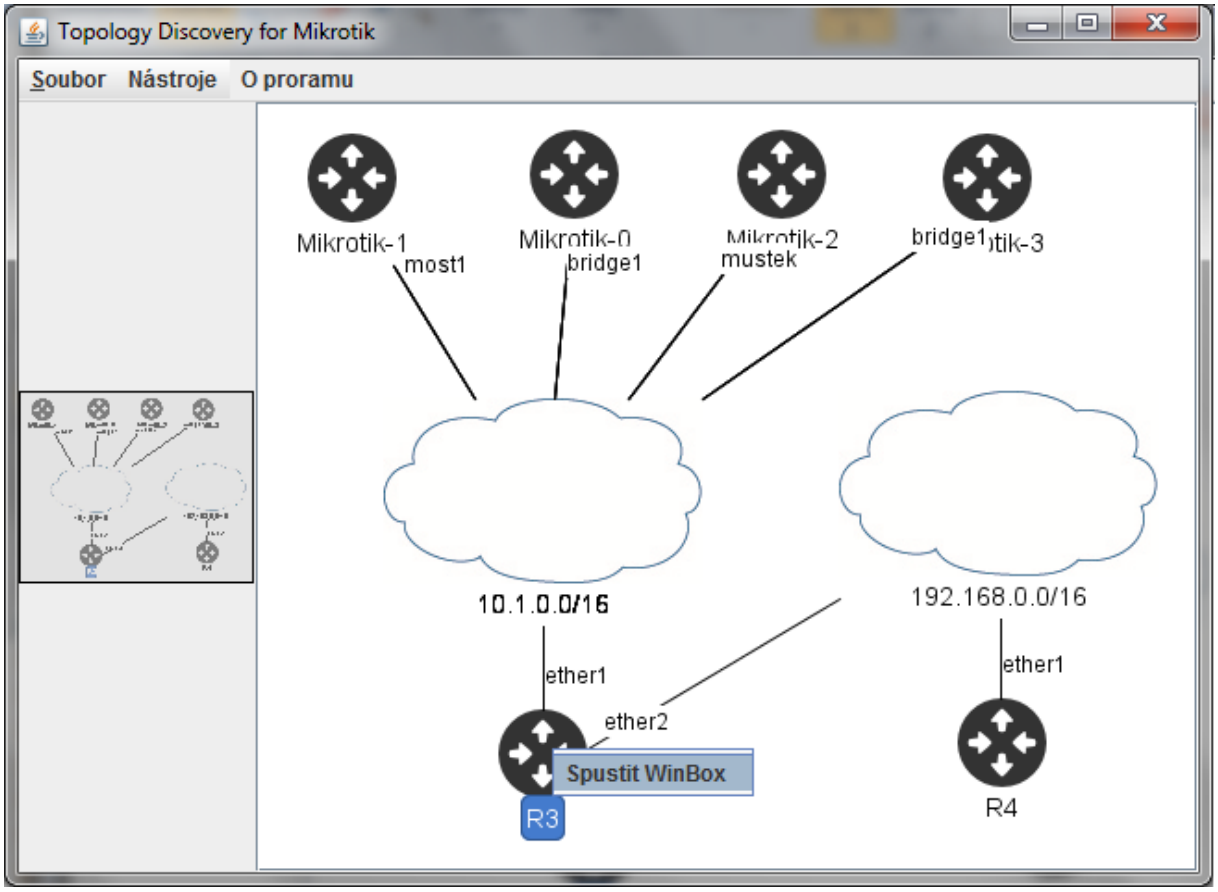
3.4.2 NetBeans Visual Library

Nakonec jsem se uchýlil ke knihovně, která možná nemá tak rozsáhlé možnosti jako některá předchozí, ale má z mého pohledu několik výhod. Je obsažena ve vývojovém prostředí NetBeans IDE [19], které jsem používal k implementaci. Již jsem měl tuto knihovnu nastudovanou a rozuměl jsem, jak je postavena a jak ji použít. Zároveň má jednoduchou demo aplikaci pro ukázky. Mohl jsem si tak projít ukázky a rozhodnout se, které vlastnosti chci použít. Obsahuje již předdefinované komponenty, které lze rozšířit a přizpůsobit.

Je postavena na objektech zvaných "Widget". *Widget* je jednoduchý visuální objekt podobný jako *JComponent*. Obsahuje informaci o svojí pozici, (preferované) velikosti, fontu, hranici objektu, aj. V aplikaci jsem využil zděděnou třídu *IconNodeWidget*, *ConnectionWidget* pro vytvoření spoje mezi instancemi třídy *Widget*, *WidgetAction* pro vytvoření "akce".

Základem pro zobrazení je jakési "plátno". V této knihovně se nazývá "scéna". Rozšiřující třídy jsou *ObjectScene*, *GraphScene*, *GraphPinScene*. Jako nejlepším kandidátem se z počátku jevila třída *GraphPinScene*. Ta totiž umožňuje spojení jen mezi tzv. "piny", které jsou přiřazené k uzlu. Jako uzel by bylo zařízení a jako pin by byl port na tomto zařízení. Bohužel, jak jsem psal výše, nebylo vždy možné získat informace, který spoj je od kterého portu zapojen. Proto jsem nakonec musel zvolit *GraphScene*. Další důležitá věc je rozložení - tzv. "layout". K dispozici jsou *GraphLayout*, *GridGraphLayout*, *TreeGraphLayout* a *UniversalGraph*. Prakticky je jedno, které rozvrhnutí se vybere. Jelikož je počítačová síť obvykle "strom", použil jsem *TreeGraphLayout*.

Vytvořil jsem třídu *RouterOSWidget*, pro kterou jsem zdědil *IconNodeWidget*. Ta reprezentuje zařízení s RouterOS. Každá instance obsahuje odkaz na datovou instanci třídy *RouterOSDevice*. A z každé instance *RouterOSWidget* lze spustit program WinBox s přihlašovacími údaji na konkrétní zařízení 3.7. Protože se v jedné podsíti vidí všechna zařízení jako vedle sebe, nemá smysl propojovat každé s každým (i když to tak je v datové částí uloženo). Proto jsem vytvořil další třídu *NetworkWidget*, která reprezentuje podsíť. K tomuto "widgetu"jsou připojeny jednotlivé *RouterOSWidget*, které do této podsítě patří.



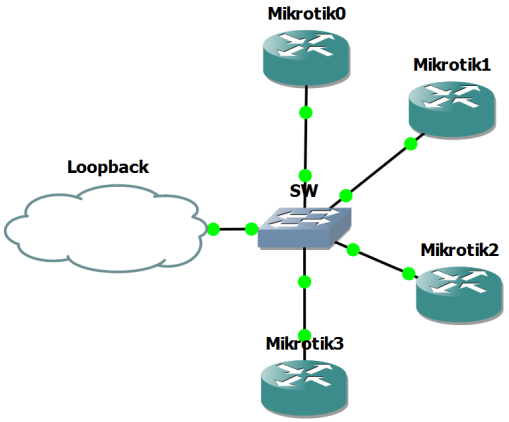
Obrázek 3.7: Ukázka aplikace

Kapitola 4

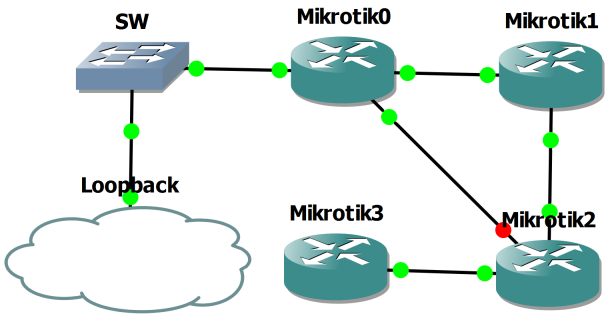
Testování

Z důvodu nedostupnosti reálné počítačové sítě (o více jak 2 zařízeních MikrotTik) probíhalo testování aplikace na simulované počítačové síti V programu GNS3 (viz 2.2.3). Tento program poskytuje reálnou simulaci s možností změny topologie.

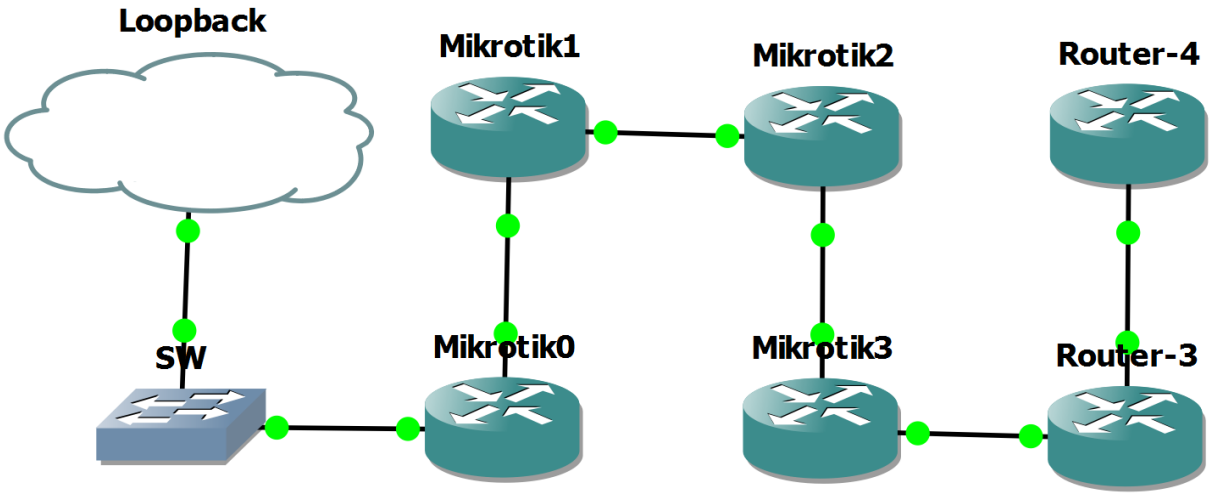
Topologie, na kterých jsem aplikaci testoval, jsem volil tak, aby odpovídaly reálné počítačové síti. V praxi je často používaná zapojení tzv. do hvězdy, zapojení za sebou a pro redundanci se využívá i kruh. Při zapojení do kruhu samozřejmě nesmí být všechna spojení propustná, aby síť fungovala tak, jak má. Nadbytečné spoje se může deaktivovat ručně (odpojením kabelu, vypnutím portu) nebo za pomocí Spaning Tree Protokolu (STP) nebo jinou jeho verzí - RSTP, MSTP, VSTP nebo PVST. Nezapomněl jsem ani na několik směrovačů mezi sítěmi.



Obrázek 4.1: Topologie hvězda



Obrázek 4.2: Topologie kruh



Obrázek 4.3: Topologie řetěz

Kapitola 5

Vysledky a rozšíření aplikace

Výsledná aplikace systematicky prochází počítačovou síť a dynamicky vykresluje zařízení s RouterOS v síti. Využívá k tomu proprietární protokol MNDP společnosti MikroTik. Po vykreslení je možné si prohlídnout částečnou strukturu sítě a připojit se pomocí programu WinBox na vyhledaná zařízení. Při připojení již není nutné zadávat přihlašovací údaje.

Největší prostor pro rozšíření aplikace je v grafickém rozhraní a podpůrných nástrojích. Důvodem, proč je grafické rozhraní jednoduché je skutečnost, že to nebylo cílem této práce. Aplikaci by bylo vhodné rozšířit o export dat do obecně používaných formátů jako jsou PDF nebo XML. Vhodný by byl formát, který podporuje program The Dude 2.2.1 a je možné ho do tohoto programu importovat.

Připojení k zařízení bylo implementováno pouze pomocí API. Vhodné by bylo implementovat připojení pomocí protokolu SSH, protože ten je (na rozdíl od API) ve výchozím stavu zapnutý. Konfigurace aplikace by bylo příjemnější provádět přímo z grafické nadstavby. Momentálně se nastavuje pouze IP adresa prvního zařízení. Seznam přihlašovacích údajů je uložen v CSV soubouru, který je nutné editovat ručně (v jiném editoru).

Kapitola 6

Závěr

Nastudoval jsem protokol NDP (Neighbor Discovery Protocol) a možnosti dynamického vykresleni topologie počítačové sítě. Prozkoumal jsem možnosti vzdálené správy zařízení Mikrotik a zjistil, jaké jsou klady a zápory. Zjistil jsem, jaké jsou dostupné možnosti pro centrální správu sítě s grafickou podporou a navrhl vlastní řešení.

Vlastní řešení oproti stávajícím je zaměřeno na zařízení MikroTik. Jediným konkurentem je program The Dude 2.2.1. Zásadní výhodou mé aplikace oproti stávajícím řešením, je v systematickém hledání sousedů v síti. Vyhledávání sousedů se obvykle provádí skenováním rozsahu sítě, tak jak to dělá program The Dude. Má aplikace nevytváří duplicitní znázornění zařízení, tak jako program The Dude.

Aplikace zatím není plnohodnotnou alternativou ke komerčním variantám. Nicméně jedná se o první verzi tohoto programu a při dalším rozvoji by se mohla stát konkurentem na trhu.

Literatura

[1] *Oracle, The Java Tutorials, 2011*.
Dostupné z: http://download.oracle.com/javase/tutorial/.
z 10.5.2014

[2] *MikroTik Neighbor Discovery Protocol*.
Dostupné z: http://www.mikrotik.com/testdocs/ros/3.0/system/mndp.php.
z 10.5.2014

[3] *Manual:API*.
Dostupné z: http://wiki.mikrotik.com/wiki/API.
z 10.5.2014

[4] *CSVReader*.
Dostupné z: http://www.csvreader.com/java_csv.php.
z 3.3.2014

[5] *GNS3*.
Dostupné z: http://www.gns3.net.
z 20.10.2013

[6] *Cisco Packet Tracer*.
Dostupné z: https://www.netacad.com/web/about-us/cisco-packet-tracer.
z 1.5.2014

[7] *Nagios*.
Dostupné z: http://www.nagios.com.
z 1.5.2014

[8] *edu.stanford.nlp.util.ArraySet*.
Dostupné z: http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/util/ArraySet.html.
z 7.4.2014

[9] *com.ibm.wala.util.collections.ArraySet*.
Dostupné z:
http://wala.sourceforge.net/javadocs/trunk/com/ibm/wala/util/collections/ArraySet.html.
z 7.4.2014

[10] *soot.util.ArraySet*.
Dostupné z: http://www.sable.mcgill.ca/soot/doc/soot/util/ArraySet.html. z 7.4.2014

[11] *org.slim3.util.ArraySet*.
Dostupné z:
http://slim3.googlecode.com/svn/tags/1.0.9/slim3/javadoc/org/slim3/util/ArraySet.html.
z 7.4.2014

[12] *Lightweight Java Game Library (LWJGL)*.
Dostupné z: http://lwjgl.org/.
z 20.4.2014

[13] *Apache^{TM} Batik SVG Toolkit*.
Dostupné z: http://xmlgraphics.apache.org/batik/.
z 20.4.2014

[14] *Graphical Editing Framework (GEF)*.
Dostupné z: http://www.eclipse.org/gef//.
z 20.4.2014

[15] *OpenJGraph*.
Dostupné z: http://freecode.com/projects/openjgraph/.
z 20.4.2014

[16] *Java Universal Network/Graph Framework (JUNG)*. http://jung.sourceforge.net/.
z 20.4.2014

[17] *yFiles for Java*.
Dostupné z: http://www.yworks.com/en/products_yfiles_about.html.
z 20.4.2014
yFiles for JavaFX.
Dostupné z: http://www.yworks.com/en/products_yfilesjavafx_about.html.
z 20.4.2014

[18] *Inksacpe*.
Dostupné z: http://www.inkscape.org/cs/.
z 10.4.2014

[19] *Netbeans IDE*.
Dostupné z: https://netbeans.org/.
z 1.10.2013

[20] *Wireshark*.
Dostupné z: http://www.wireshark.org/.
z 10.5.2014

[21] *Wine*.
Dostupné z: http://www.winehq.org/.
z 10.5.2014

[22] *Cisco Discovery Protocol Version 2*.
Dostupné z:
http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cdp/configuration/15-mt/cdp-15-mt-book/nm-cdp-discover.html. z 10.5.2014
Discovery Protocols.
Dostupné z: www.cisco.com/c/en/us/td/docs/net_mgmt/active_network_abstraction/3-7/reference/guide/ANARefGuide37/discover.html.
z 10.5.2014

[23] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE Std 802.1AB-2009 (Revision of IEEE Std 802.1AB-2005), IEEE Standard for Local and metropolitan area networks–Station and Media Access Control Connectivity Discovery*.
Dostupné z: https://standards.ieee.org/getieee802/download/802.1AB-2009.pdf.
z 10.5.2014

[24] Neighbor Discovery for IP version 6 Request for Comments: 4861
Dostupné z: http://tools.ietf.org/html/rfc4861.
z 10.5.2014

[25] Pavel Satrapa *Internet Protokol verze 6, třetí vydání.* Praha 2011, CZ.NIC, z.s.p.o.
Dostupné z: http://knihy.nic.cz/files/nic/edice/pavel_satrapa_ipv6_2012.pdf.
z 10.5.2014

Příloha A

Seznam zkratek, značek a symbolů

- NDP = Neighbor Discovery Protocol
- MNDP = MikroTik Neighbor Discovery Protocol
- API = Application Programmable Interface
- CLI = Command Line Interface
- MAC = Media Access Control
- IP = Internet Protocol
- TCP = Transmission Control Protocol
- GPL = GNU General Public License
- CPU = Central Processing Unit
- NAS = Network Attached Storage
- IOS = Internetwork Operating System (od Cisco)
- ISO = International Organization for Standardization
- OSI = Open Systems Interconnection
- SNMP = Simple Network Management Protocol
- SSH = Secure Shell
- SQL = Structured Query Language
- OSPF = Open Shortest Path First

- BGP = Border Gateway Protocol
- VM = Virtual Mashine
- GNS = Graphical Network Simulator
- RAM = Random-Access Memory
- OS = Operating System
- OOP = Object-oriented programming
- JRE = Java Runtime Enviroment
- PHP = Personal Home Page
- CSV = Comma-separated values
- GUI = Graphical User Interface
- SVG = Scalable Vector Graphics
- BMP = Bitmap Image File
- PNG = Portable Network Graphics
- IDE = Integrated Development Environment
- RSTP = Rapid Spanning Tree Protocol
- MSTP = Multiple Spanning Tree Protocol
- VSTP = VLAN Spanning Tree Protocol
- PVST = Per-VLAN Spanning Tree
- VLAN = Virtual Local Area Network

Příloha B

Třída ArraySet<E >

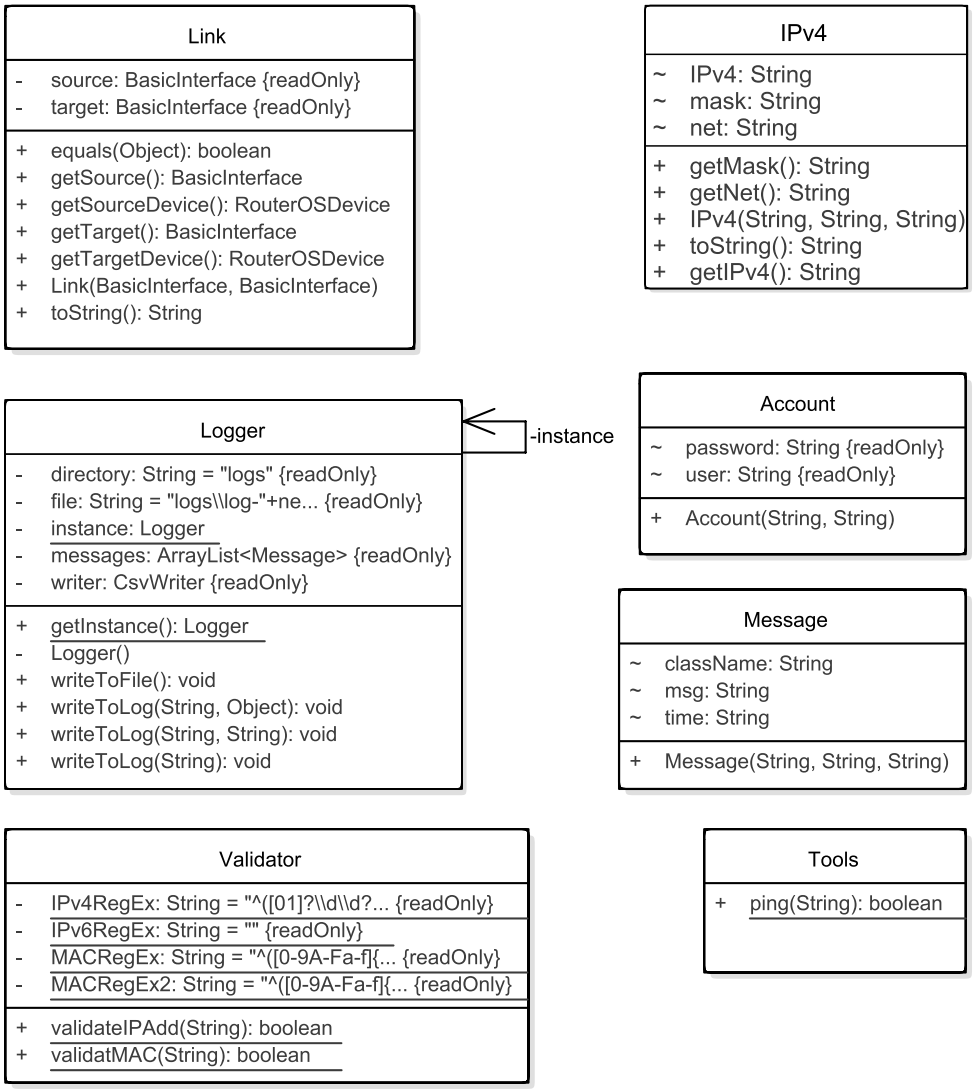
Listing B.1: Třída ArraySet<E >

```
/**
 * @author Bc. Michal Hanzlik
 * @param <E>
 */
public class ArraySet<E> extends ArrayList<E> implements java.util.
    Set<E> {

    private E present;
    private int index = -1;

    public ArraySet() {
        super();
    }

    /**
     * Prida prvek jen tehdy, kdyz dany prvek jeste v~mnozine neni.
     * Vyuziva metodu equals()
     * @param element Vkladany prvek
     * @return TRUE jestlize byl prvek pridan, jinak FALSE
     * @see getLastDuplicity()
     * @see getIndex()
     */
    @Override
    public boolean add(E element) {
        if (this.size() == 0) {
            super.add(element);
            return true;
        }
```

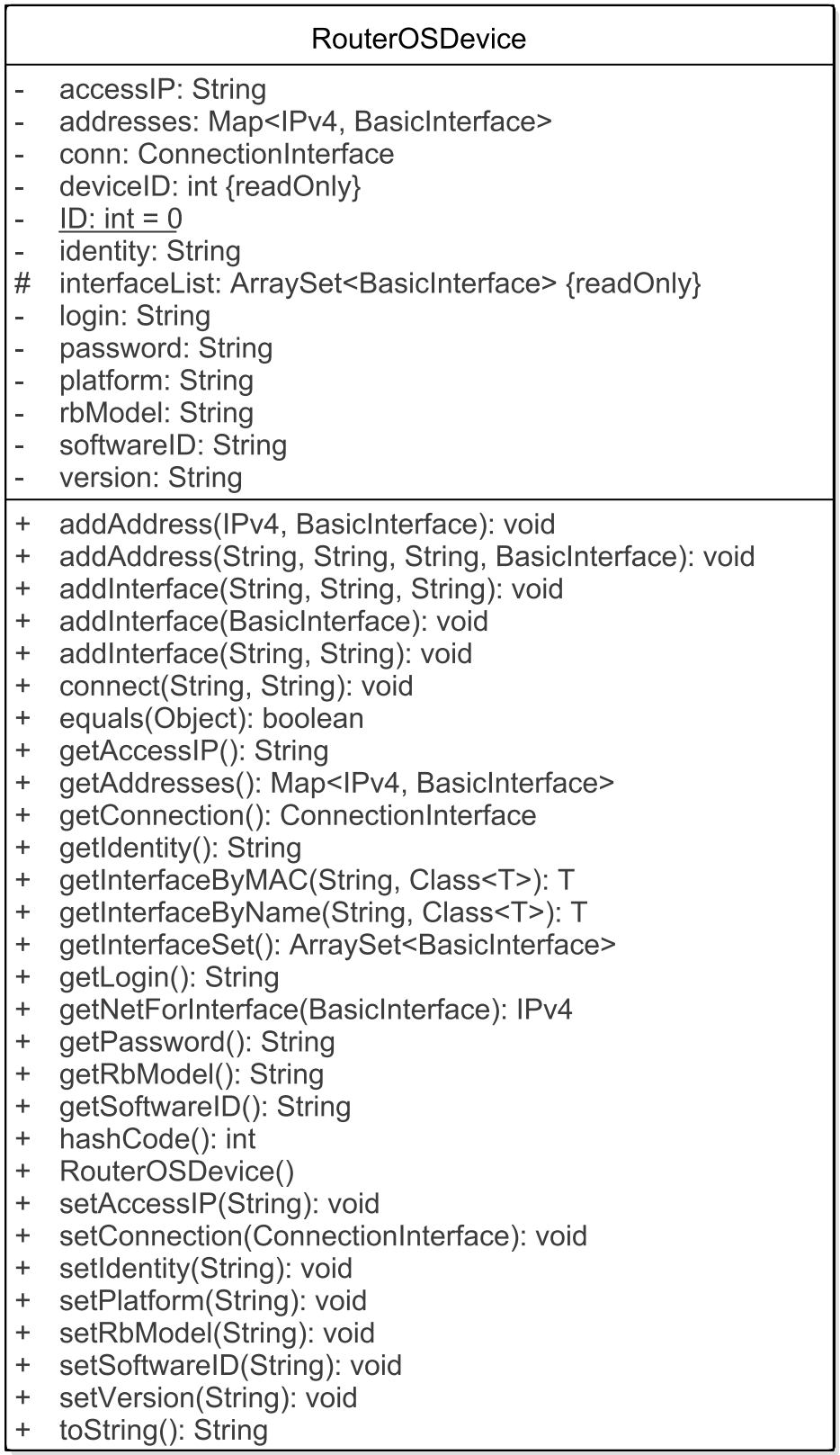
```
        for (int i = 0; i < this.size(); i++) {
            E e1 = this.get(i);
            if (element.equals(e1)) { // jestli uz v~seznamu je
                this.index = i;
                this.present = e1;
                return false;// vrat false
            } else if (i == (this.size() - 1)) {
                super.add(element);  // pridat a vratit true
                return true;
            }
        }
        return false;
    }

    /**
     * Vraci prvek posledniho vkladane duplicity.
     * @return element
     */
    public E getLastDuplicity() {
        return present;
    }

    /**
     * Vraci index posledniho vkladane duplicity.
     * @return index
     */
    public int getIndex() {
        return index;
    }
}
```

Příloha C

Diagramy



Obrázek C.1: Další využité třídy



Obrázek C.2: RouterOSDevice

Příloha D

Návod na propojení GNS3 s PC

Tento návod popisuje, jak propojit lokální počítač se simulovanou sítí v programu Graphical Network Simulator (GNS3). Návod je zaměřen na operační systém Microsoft Windows.

Použitá konfigurace:

- Microsoft Windows 7 64bit
- GNS3 verze 0.8.6

D.1 Nainstalování programů

První je potřeba stáhnout a nainstalovat potřebné programy. GNS3 je možné stáhnout na oficiálních stránkách http://www.gns3.net/.

GNS3 není třeba instalovat. Stačí stáhnout ZIP archiv pro příslušnou verzi operačního systému (32/64 bit) nebo balíček, který obsahuje obě verze. Po rozbalení obsahuje složka s programem spustitelný EXE soubor.

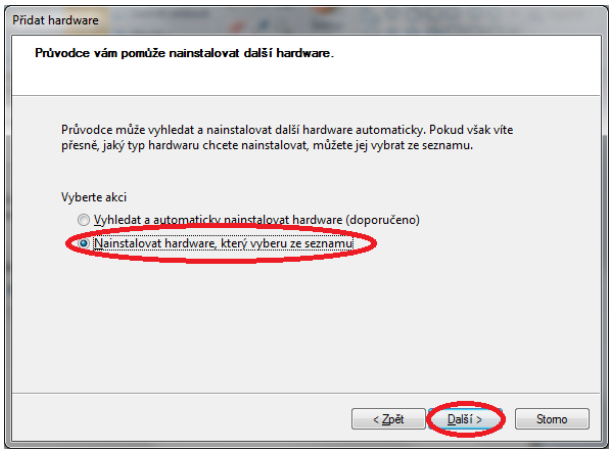
D.2 Vytvoření síťového připojení

V simulované síti musí být zástupce lokálního počítače. Ten se připojí na tzv. "loopback"síťový adaptér. Ten je potřeba vytvořit. Vytvoří se pomocí "Průvodce přidání hardwaru".

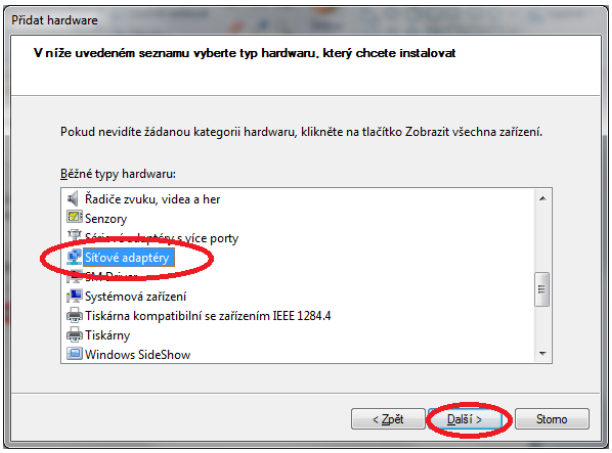
1. Kliknout na start
2. Vyhledat "cmd"
3. Spustit "cmd"jako administrátor

4. Spustit "hdwwiz.exe"

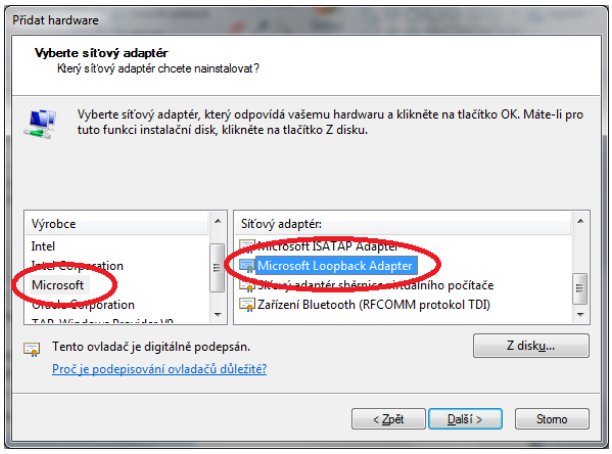
V druhém kroku průvodce zvolte položku "Nainstalovat hardware, který vyberu ze seznamu"a klepněte na tlačítko "Další". (viz obrázek D.1) Ve třetím kroku zvolte typ hardwaru "Síťové adaptéry"a klepněte na tlačítko "Další".(viz obrázek D.2) Ve čtvrtém kroku zvolte výrobce "Microsoft"a síťový adaptér "Microsoft Loopback Adapter"a klepněte na tlačítko "Další". (viz obrázek D.3) Pokračujte a dokončete instalaci. Toto síťové rozhraní si pro lepší identifikaci pojmenujeme "Loopback". (viz pbrázek D.4)



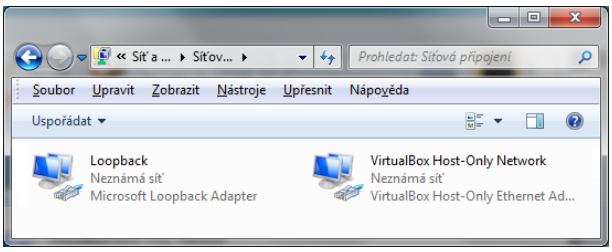
Obrázek D.1: Průvodce - Krok 2



Obrázek D.2: Průvodce - Krok 3



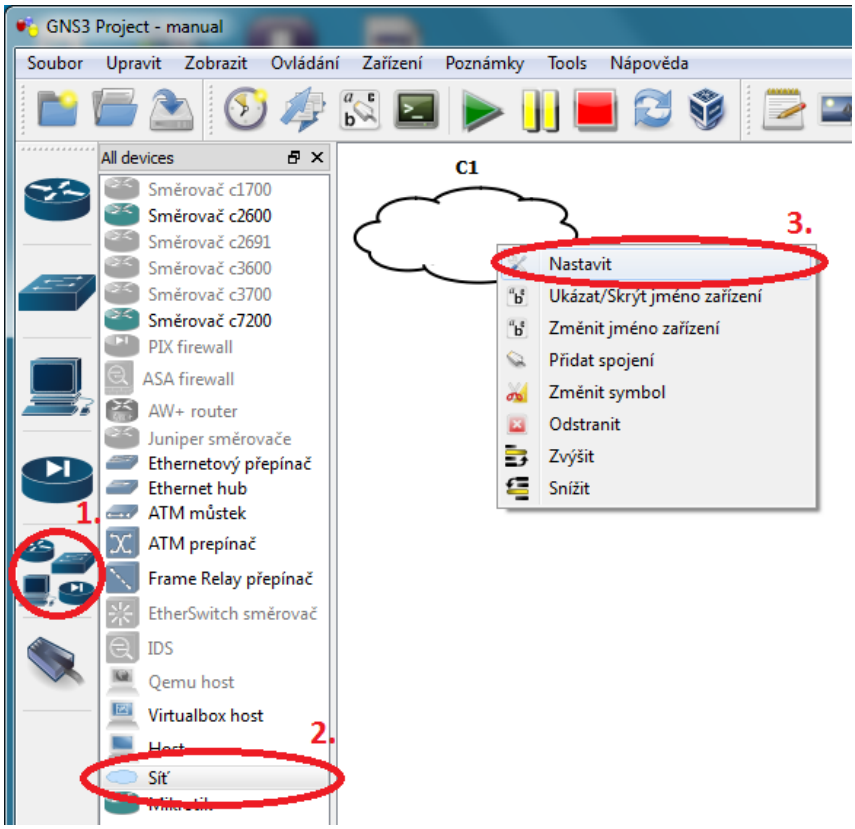
Obrázek D.3: Průvodce - Krok 4



Obrázek D.4: Seznam připojení k síti

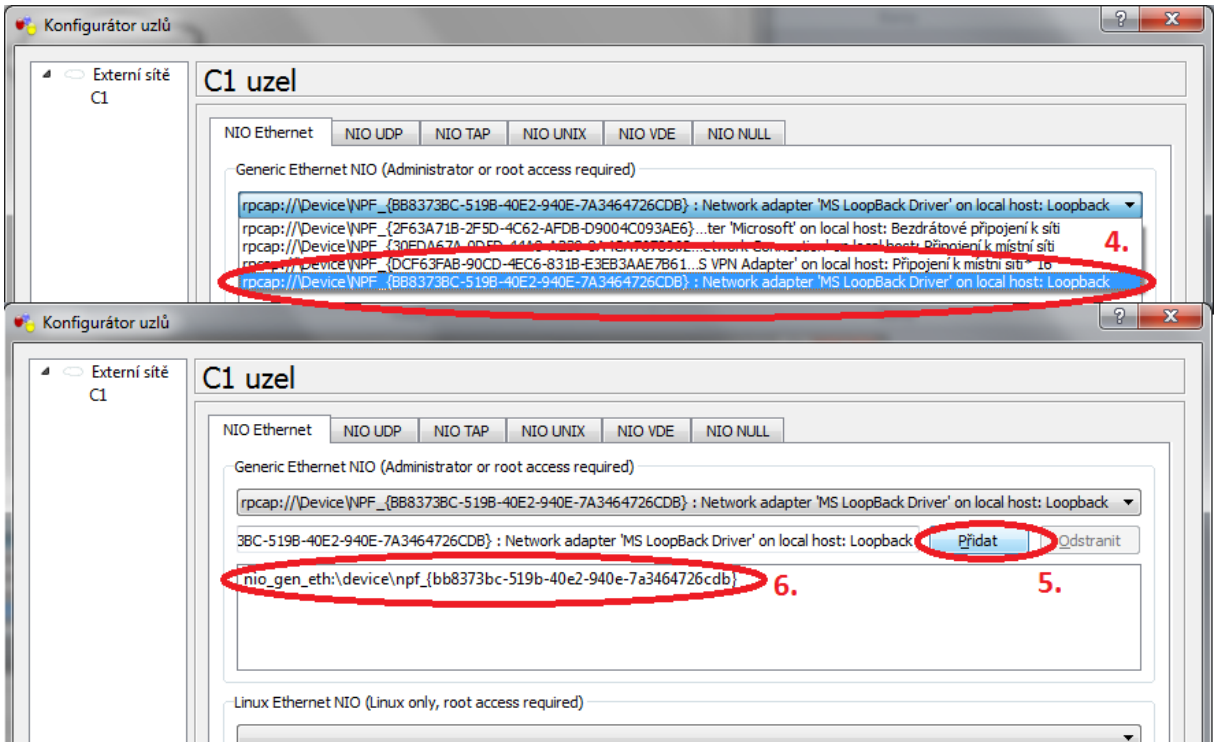
D.3 Nastavení projektu

V projektu vyberte z dostupných prvků (viz D.5 bod 1) prvek nazvaný "Síť"(Cloud) (bod 2) a myší ho přetáhněte na "plátno". Klikněte na tento prvek pravým tlačítkem myši a zvolte položku "Nastavit". Na kartě "NIO Ethernet"v sekci "Generic Ethernet NIO"vyberte síťový adaptér "MS LoopBack Driver"s názvem "Loopback".(viz D.6 bod 4.) a přidejte tlačítkem "Přidat"(bod 5.). V seznamu přiřazených adaptérů by se měl tento prvek objevit (bod 6.).



Obrázek D.5: Nastavení - Krok 1

Nyní máte propojené PC se simulovaném prostředí v programu GNS3.



Obrázek D.6: Nastavení - Krok 2

Příloha E

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové kódy a funkční spustitelná aplikace.

- Adresář *Aplikace*:
  - lib - Grafické knihovny
  - app - Obsahuje program winbox.exe
  - accunts.csv - Přihlašovací údaje
  - start.bat - Spustitelný soubor.
  - TDM.jar - Zkompilované zdrojové soubory
- Adresář *Diplomová práce*:
  - obr - Adresář s použitými obrázky.
  - hanzlmi4.pdf - Tato diplomová práce.
- Adresář *Src* - Zdrojové soubory aplikace
- Adresář *Zdroje*
  - 802.1AB-2009.pdf - IEEE 802.1AB standard (LLDP)
  - pavel_satrapa_ipv6_2012.pdf - Internetový protokol IPv6, Pavel Satrapa