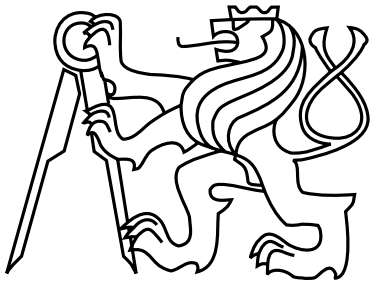
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra kybernetiky



Bakalářská práce

Automatické rozpoznávání změn v topologii monitorované sítě

Radek Holý

Vedoucí práce: Ing. Michal Medvecký

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Inteligentní systémy

27. května 2010

Poděkování

Rád bych poděkoval mému vedoucímu Ing. Michalu Medveckému za jeho cenné rady, náměty a zejména čas, který si na mne a mé dotazy při tvorbě této práce udělal. Dále bych poděkoval své rodině a přátelům za toleranci a podporu v průběhu vývoje bakalářské práce.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
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 27. 5. 2010 .....

Abstract

The aim was to study the protocols applicable to the network topology reconstruction and to propose a method for automatic detection of topology changes in monitored network. The result is a design and implementation of method, which identifies the changes and their causes, prints them and applies them to the monitoring system configuration.

Abstrakt

Cílem práce bylo prostudovat protokoly použitelné pro rekonstrukci topologie sítě a navrhnout metodu automatického rozpoznávání změn v topologii monitorované sítě. Výsledkem je návrh a implementace metody, která změny a jejich příčiny rozpoznává, vypisuje a aplikuje do konfigurace monitorovacího systému.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Aby správce jakékoliv počítačové sítě mohl vykonávat svou práci, musí nepochybně znát topologii této sítě. V případě, že nemá k dispozici vhodnou dokumentaci, nebo má pouze požadavek na převedení této dokumentace do elektronické podoby, jako nejjednodušší prostředek se nabízí aplikace, která pomocí dostupných síťových protokolů zmapuje zapojení sítových zařízení.

Jelikož se však topologie sítě neustále mění (ať už z technických důvodů, nebo kvůli zásahům uživatelů), bylo by velkým usnadněním práce, kdyby měl správce k dispozici také nástroj, který tyto změny umí automaticky rozpoznat. V okamžiku, kdy je změna objevena, je žádoucí, aby o ní (a její příčině) byla informována patřičná osoba a ideálně i další běžící aplikace, které vyžadují znalost topologie ke své práci.

Kapitola 2

Popis problému, specifikace cíle

Pokud aplikace, které rozpoznávají změny v topologii sítě, existují, dá se očekávat, že nebudou schopné diagnostikovat i příčiny nebo nebudou schopné spolupráce s jinými (a navíc jakýmikoliv) aplikacemi nebo nebudou dostupné pro ostatní platformy nebo správce s malými finančními prostředky. Tato skutečnost je důvodem vzniku této bakalářské práce.

Cílem této práce je:

- Prozkoumat protokoly 2. a vyšší vrstvy ISO/OSI modelu, které jsou použitelné pro rekonstrukci topologie sítí. Viz kapitola [3].
- Navrhnout metodu, která bude automaticky poznávat změny v topologii sítě s ohledem na monitorované prvky. Tyto změny bude metoda vhodně aplikovat do konfigurace monitorovacího systému. Viz kapitola [4].
- Navrženou metodu implementovat do některého open-source monitorovacího systému. Viz kapitola [5].

Kapitola 3

Teorie

V následující části jsou popsány různé druhy topologií, dvě nejznámější síťové architektury a protokoly, které jsou vhodné k rekonstrukci topologie sítě.

3.1 Topologie sítí

Pod pojmem zjišťování topologie sítě si můžeme představit zjišťování vzájemného propojení zařízení v dané síti. [12, 1, 77, 38]

3.1.1 Kategorie topologií sítí

Podle toho, jaký typ spojení sledujeme, můžeme topologii zařadit do jedné ze tří kategorií. Fyzické, logické, nebo signálové. Tu pak lze reprezentovat grafem, ve kterém uzly představují jednotlivá zařízení a hrany spojení mezi nimi. V jedné síti se grafy fyzické, logické i signálové topologie mohou lišit.

Sledujeme-li fyzické spoje mezi zařízeními (zpravidla síťové kabely), získáme *fyzickou topologii*. Sledujeme-li zdánlivou cestu, kterou vykonají data mezi jednotlivými zařízeními, získáme *logickou topologii*. Sledujeme-li skutečnou cestu, kterou vykoná signál mezi jednotlivými zařízeními, získáme *signálovou topologii*.

3.1.2 Struktura topologie sítě

Výsledné struktury pak můžeme rozdělit na dvoubodové, sběrnicové, hvězdicové, kruhové, síťové a stromové topologie.

Dvoubodová topologie je taková, ve které je cesta vždy mezi dvěma zařízeními. Pokud je tato cesta neměnná a dané zařízení vždy komunikuje s tím samým protějškem, mluvíme o *trvalé dvoubodové topologii*. Pokud se dvě zařízení spojují až dynamicky podle potřeby, mluvíme o *přepínané dvoubodové topologii*.

Sběrnicová topologie je taková, ve které jsou všechna zařízení propojena jedinou cestou a tedy i všechna data dostávají všechna zařízení (téměř) ve stejný okamžik. Má-li cesta právě dva konce, mluvíme o *lineární sběrnicové topologii* (nebo také *páteři* nebo *tepně*). Větví-li

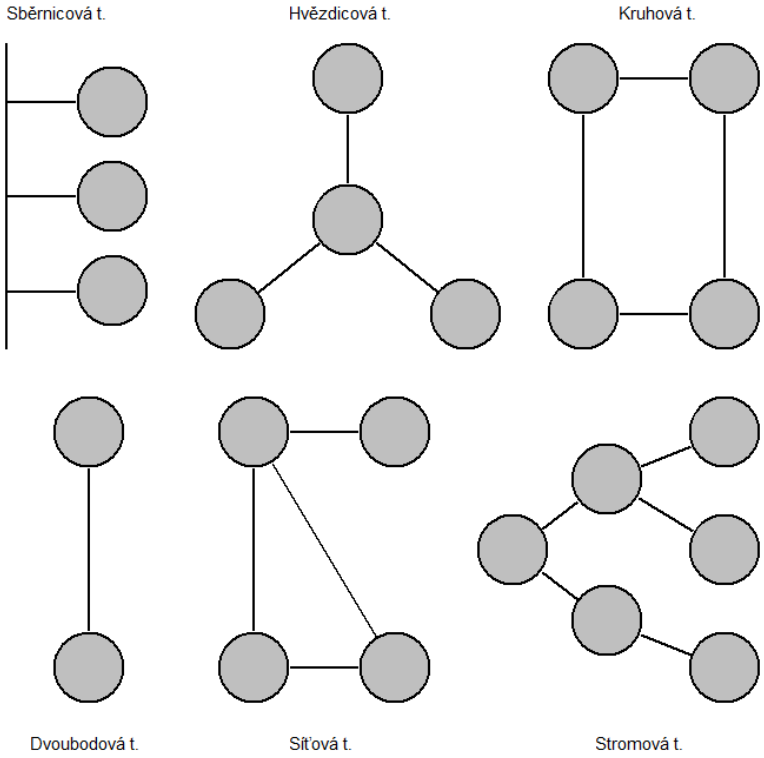
se v některých místech tato cesta (má tedy více konců), mluvíme o *distribuované sběrnicové topologii*. [44, 20]

Hvězdicová topologie je taková, ve které všechny cesty (jsou dvoubodového typu) ze zařízení vedou do jednoho centrálního prvku. Ten nemusí nutně být nějakým specializovaným zařízením. Obsahují-li některé cesty opakovač, mluvíme o *rozšířené hvězdicové topologii*. Obsahují-li některé cesty hub nebo rozbočovač, mluvíme o *hierarchické hvězdicové topologii*. Dojde-li k lineárnímu spojení (řetězení) více sítí s hvězdicovou topologií, mluvíme o *distribuované hvězdicové topologii*. [75, 24]

Kruhová topologie je taková, ve které vede cesta z prvního zařízení do druhého, z druhého do třetího a tak dále, přičemž z posledního zařízení vede cesta do prvního. Vznikne tak kruh. V tomto případě prochází signál všemi zařízeními v definovaném směru. [69, 27]

Síťová (smyčková) topologie je taková, ve které z různých zařízení vedou cesty (dvoubodového typu) do různých jiných zařízení. Vedou-li ze všech zařízení cesty do všech ostatních, mluvíme o *úplné síťové topologii*. Vedou-li jen z některých zařízení cesty do dvou a více jiných zařízení, mluvíme o *částečné síťové topologii*. [61]

Stromová (hierarchická) topologie je taková, ve které se nachází jedno (kořenové) zařízení na nejvyšší úrovni, k němuž jsou dvoubodově připojena zařízení na druhé úrovni, k nimž jsou připojena zařízení na třetí úrovni, k nimž mohou být připojena zařízení na čtvrté úrovni atd. Každý prvek má fixní počet následníků (alespoň dva), který se nazývá *větvící faktor*. [36]



Obrázek 3.1: Znázornění struktur topologie sítě.

3.2 Referenční model ISO/OSI

Referenční model ISO/OSI je abstrakce, jejíž implementací vzniká standardizované síťové prostředí. Definuje funkce služeb od hardwarové úrovně až po aplikační rozhraní. Popsán je v dokumentech série CCITT s označením X.200 [5] a normě ISO s číslem 7498 [4]. [6, 8, 9, 10, 3, 56, 26]

3.2.1 Vrstvy modelu

V modelu je definováno sedm vrstev (kolekcí služeb), přičemž každá má svou specifickou funkci. Komunikace začíná požadavkem v sedmé vrstvě. Ta jej předá šesté vrstvě a tak dále (vždy pomocí tzv. rozhraní) až po první vrstvu. Ta se postará o předání informace dalšímu zařízení. Zde pak informace prochází zpětně od první po sedmou vrstvu, která pak poskytuje původní informaci. Pokud příjemce nebyl adresátem požadavku, některá z vrstev zařídí, aby byl požadavek předán dál. Z aplikačního hlediska však tento přenos vypadá jako komunikace stejných vrstev mezi sebou (pomocí tzv. protokolů). Prostředkem pro jejich komunikaci jsou data (hlavičky), která přidají k původnímu požadavku (dochází k tzv. zapouzdřování). Jestliže informace přichází z vyšší vrstvy, aktuální vrstva přidá k datům z vyšší vrstvy své informace a celek předá nižší vrstvě. V opačném směru jsou pak potřebné informace vyextrahovány z dat od nižší vrstvy a není-li hlavičkou řečeno něco jiného, je zbytek předán vyšší vrstvě. Těchto sedm vrstev se nazývá fyzická, spojová, síťová, transportní, relační, prezentační a aplikační vrstva.

Sedmá *aplikační vrstva* umožňuje aplikacím přístup k přenosu dat po síti. Tyto společné aplikační části a podpůrné služby jsou reprezentovány virtuálními zařízeními nad konkrétními implementacemi. [41, 17]

Šestá *prezentační vrstva* převádí datové struktury a typy do jednotného formátu. Přizpůsobuje pořadí bajtů. Na této vrstvě také může docházet k šifrování a kompresi dat. [68, 31]

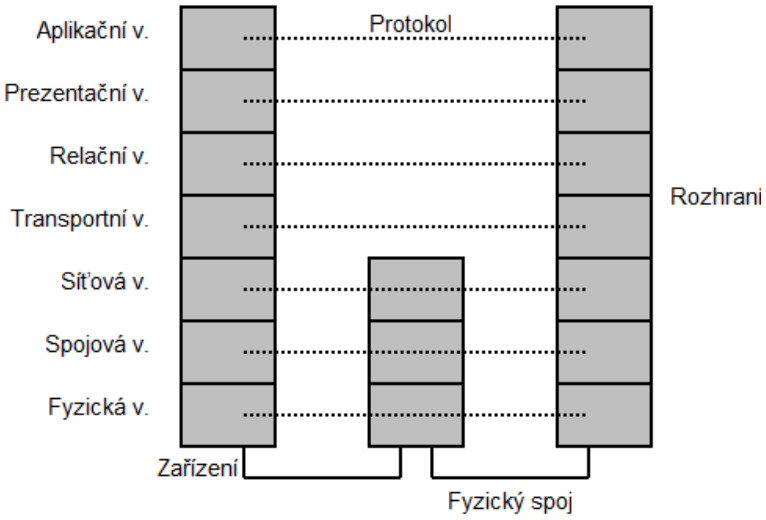
Pátá *relační vrstva* vytváří a ukončuje spojení mezi zařízeními. Předává pověření a zajišťuje synchronizaci přenosu dat. Často je implementována jako sada nástrojů, které obalují komunikaci na nižších vrstvách. [71, 32]

Čtvrtá *transportní vrstva* opatřuje data adresou odesílající aplikace, multiplexuje je a tyto celky dělí do segmentů (resp.datagramů) o velikosti přijatelné pro třetí vrstvu. Řídí tok dat (předchází přetečení zásobníku příjemce) a sleduje výskyt chyb (popř. je napravuje). Pracuje s potvrzovacími mechanismy. Zajišťuje buď spojovaný, nebo nespojovaný, buď spolehlivý, nebo nespolehlivý přenos dat. Umožňuje použití více implementací síťové vrstvy najednou. [78, 39]

Třetí *síťová vrstva* rozděluje segmenty na pakety takové velikosti, která je přijatelná pro druhou vrstvu, opatřuje je logickou (síťovou) adresou a umožňuje jejich přenos mezi nesousedícími zařízeními a sítěmi (včetně těch, co běží na různých technologiích). [64, 34]

Druhá *spojová (linková) vrstva* rozděluje pakety do rámců a řadí je do správného pořadí. Rámce opatřuje fyzickou adresou, která je jedinečná pro všechna zařízení. Zajišťuje pak jejich přenos mezi dvěma přímo propojenými zařízeními. Detekuje chyby při přenosu po fyzické vrstvě. Řídí přístup k médiu a datový tok. Pracuje s potvrzovacími a synchronizačními mechanismy. [57, 28]

První *fyzická vrstva* převádí rámce po bitech na signály a následně je přenáší protějšímu zařízení. Na druhé straně je pak přijímá. Definuje parametry přenosového kanálu, kódování, modulaci apod. [67, 23]



Obrázek 3.2: Znázornění referenčního modelu ISO/OSI.

3.3 Model TCP/IP

Model TCP/IP, někdy zvaný Internet model, na rozdíl od ISO/OSI modelu není referenční. Je tedy konkrétní implementací pravidel a protokolů umožňujících počítačům komunikaci po síti. Ty definují, jak mají být data formátovány, adresovány, směrovány a přeneseny do cíle. Další odlišností oproti ISO/OSI modelu je princip, který byl aplikován při návrhu modelu. Zatímco ISO/OSI model byl navrhován „maximalisticky“ (byla snaha do modelu zahrnout co možná nejvíce funkcí), TCP/IP model staví na „minimalistickém“ principu (tj. preferuje rychlost, implementuje jen nejnutnější funkce a dává tak prostor programátorům aplikací). TCP/IP model byl vytvořen agenturou DARPA v 70. letech 20. století. V tuto chvíli jej dále vyvíjí organizace Internet Engineering Task Force. [6, 8, 10, 7, 11, 76, 53, 37]

3.3.1 Vrstvy modelu

Stejně jako ISO/OSI model se i TCP/IP model dá na podobném principu rozdělit do několika vrstev (kde přenos dat mezi vrstvami probíhá stejně, jako bylo popsáno u modelu ISO/OSI). Ne však tak striktně už jen pro to, že původní podoba se používala ještě před ustanovením ISO/OSI modelu. Dělí se pouze do čtyř vrstev: vrstva síťového rozhraní, síťová vrstva, transportní vrstva a aplikační vrstva. První dvě vrstvy jsou implementovány ve všech zařízeních, zatímco zbylé dvě až v koncových.

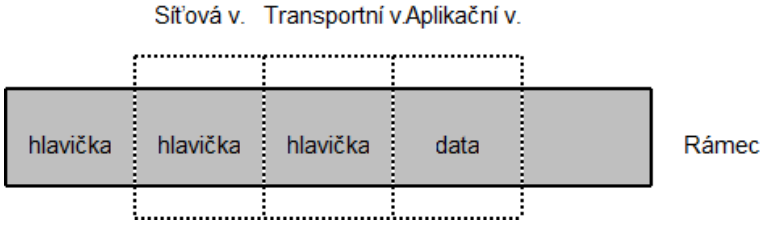
Čtvrtá *aplikační vrstva* (podobně jako ISO/OSI relační, prezentační a aplikační vrstva) reprezentuje samotné aplikace, případně některé osamostatněné funkční celky, jež bývají

dostupné pomocí API. Definuje rozhraní s transportní vrstvou nazývané port, které má svou adresu (identifikátor) a které je implementováno datovou strukturou fronta. Každá aplikace se musí sama postarat o prezentační a relační část komunikace.

Třetí *transportní vrstva* (podobně jako ISO/OSI relační a transportní vrstva) slouží k přizpůsobení chování komunikace různým aplikacím. Řídí tok dat (předchází přetečení zásobníku příjemce) a sleduje výskyt chyb (popř. je napravuje). Opatřuje data adresou odesílající aplikace, tzv. porty (jedinečné číselné označení), multiplexuje je a tyto celky dělí do segmentů. Poskytuje buď spojovaný, nebo nespojovaný, buď spolehlivý, nebo nespolehlivý přenos dat. Vytváří, udržuje a ukončuje spojení mezi zařízeními.

Druhá *síťová vrstva* (podobně jako ISO/OSI síťová vrstva) opatřuje segmenty síťovou adresou a umožňuje jejich nespolehlivý nespojovaný přenos mezi nesousedícími zařízeními a sítěmi (včetně těch, co běží na různých technologiích). [52]

První *vrstva síťového rozhraní* (podobně jako ISO/OSI fyzická, spojová a síťová vrstva) nebyla na počátku realizována. Místo toho zde byl poskytnut prostor pro použití již existujících řešení a protokolů. Tyto protokoly umožňují přístup k fyzickému přenosovému médiu. Rozdělují segmenty do rámců. Rámce opatřují fyzickou adresou. Převádí rámce po bitech na signály a následně je přenáší protějšímu zařízení. Na druhé straně je pak přijímají. Detekují chyby při přenosu. Jsou specifické pro každou síť v závislosti na její implementaci. Starají se o přenos dat v lokální síti mezi dvěma propojenými zařízeními. Definují parametry přenosového kanálu, kódování, modulaci apod. [60]



Obrázek 3.3: Znázornění rámců v modelu TCP/IP.

3.4 Síťové protokoly nad topologiemi

Automatická rekonstrukce topologie sítě se nejlépe provádí pomocí síťových protokolů, jakožto běžně se vyskytujících prostředků pro komunikaci v síti. Zajímavé jsou zejména protokoly objevující sousední zařízení, protokoly pro správu sítě, které umožňují mimo jiné získávání informací ze zařízení a směrovací protokoly pracující se směrovacími tabulkami (tabulky, které uvádějí, kam data odeslat, aby byla doručena odpovídajícímu příjemci). Jakékoliv z těchto protokolů je pak potřeba buď použít (imitovat zařízení, které běžně tento protokol používá, a vysílat obdobné požadavky) nebo komunikaci pomocí těchto protokolů odposlouchávat. V obou případech pak mohou nastat problémy s tím, že některé protokoly používají mechanizmy pro šifrování, autorizaci a další způsoby zabezpečení. [6, 8, 10, 2]

3.4.1 Protokoly objevující sousední zařízení

Cisco Discovery Protocol (CDP) je proprietární alternativa k LLDP. [45, 21]

ICMP Router Discovery Protocol (IRDP) slouží koncovým zařízením k vyhledávání směrovačů v lokální podsíti. [54]

Link Layer Discovery Protocol (LLDP) slouží zařízením k propagaci informací o sobě. Díky tomu jsou zařízení informována o svých sousedech. [58]

Link Layer Topology Discovery (LLTD) je proprietární protokol, který slouží zařízením ke zjišťování informací o svých sousedech. [59]

Neighbor Discovery Protocol (NDP) a jeho zabezpečená verze *Secure Neighbor Discovery Protocol* (SEND) jednak implementují funkce ARP a jednak slouží k vyhledávání dalších zařízení na společném fyzickém spoji včetně zařízení potřebných ke směrování dalších zpráv vzdáleným sítím. Dále umožňují informovat o použití neoptimální cesty do vzdálené sítě. [62, 73]

Nortel Discovery Protocol[1] (NDP) je proprietární alternativa k LLDP. [63]

3.4.2 Protokoly pro správu sítě

Common Management Information Protocol (CMIP) a jeho úprava *Common Management Interface Protocol Over Tcpip* (CMOT) jsou méně rozšířené protokoly pro správu sítě konkurující SNMP, které slouží k monitorování, spravování a získávání informací o zařízení. [46]

Simple Gateway Monitoring Protocol (SGMP) je dnes již zastaralý protokol pro správu sítě, který slouží ke správě bran. [72]

Simple Network Management Protocol (SNMP) je nejrozšířenější protokol pro správu sítě, který slouží k monitorování, spravování a získávání informací o zařízení. [74, 35]

3.4.3 Směrovací protokoly

Směrovací protokoly umožňují povolení, odepření nebo vynucení výměny směrovacích dat. Upozorňují také na změnu ve směrovacích datech.

Border Gateway Protocol (BGP) je protokol používaný externími branami. Tento protokol nahradil EGP, protože kompletní směrovací informace získává pouze při inicializaci a dále sleduje pouze změny. Protokol ukládá posloupnosti identifikátorů autonomních systému, přes které vede cesta do dané sítě. Výměna informací probíhá mezi sousedními směrovači. [14, 43, 19]

Enhanced Interior Gateway Routing Protocol (EIGRP) je proprietární protokol používaný v rámci autonomních systémů. Protokol si drží seznam sítí a délek cest, ke kterým má přístup. Informuje směrovače o změnách v sousedních směrovačích. Délky cest reprezentují různé volitelné parametry. [40]

[1] Dříve znám jako *SynOptics Network Management Protocol* (SONMP) nebo jako *Bay Network Management Protocol* (BNMP) nebo jako *Bay Topology Protocol* nebo jako *Bay Discovery Protocol* (BDP) nebo jako *Nortel Topology Discovery Protocol* (NTDP) nebo jako *Nortel Management MIB* (NMM).

Exterior Gateway Protocol (EGP) je dnes již zastaralý protokol používaný externími branami autonomních systémů. Protokol si drží a posílá kompletní seznam sítí, ke kterým má přístup, sousedním směrovačům. [48]

Gateway-to-Gateway Protocol (GGP) je dnes již zastaralý protokol používaný hlavními branami. Protokol si drží a pravidelně posílá kompletní seznam sítí a délek cest, ke kterým má přístup. Délka cesty reprezentuje počet mezisítí. [49]

Protokol *HELLO* je dnes již nepoužívaný protokol používaný v rámci autonomních systémů. Protokol si drží a pravidelně posílá kompletní seznam směrovačů a délek cest, ke kterým má přístup. Délka cesty reprezentuje zpoždění.

Interior Gateway Routing Protocol (IGRP) je proprietární dnes již zastaralý protokol používaný v rámci autonomních systémů. Protokol udržuje a pravidelně posílá celé směrovací tabulky včetně délek cest sousedním směrovačům. Délky cest reprezentují různé volitelné parametry. [50]

Protokol Intermediate system to intermediate system (IS-IS) je protokol používaný v rámci autonomních systémů. Protokol si drží seznam sítí a délek cest, ke kterým má přístup. Informuje směrovače o změnách v sousedních směrovačích. Délka cest reprezentuje počet mezisítí. [55]

Open Shortest Path First (OSPF) je protokol používaný v rámci autonomních systémů. Protokol si drží seznam všech směrovačů a délek cest v autonomním systému. Informuje všechny směrovače o změnách v sousedních směrovačích. Délky cest reprezentují různé volitelné parametry. [65, 29]

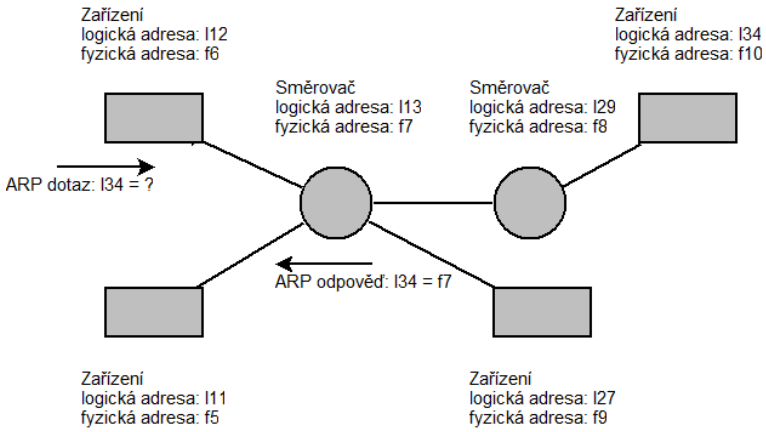
Routing Information Protocol (RIP) je protokol používaný v rámci autonomních systémů. Protokol udržuje a pravidelně posílá celé směrovací tabulky včetně délek cest sousedním směrovačům. Délka cest reprezentuje počet mezisítí. [70, 33]

3.4.4 Ostatní protokoly

Address Resolution Protocol (ARP) slouží zařízením k vyhledání fyzické adresy zařízení na základě znalosti jeho logické adresy. V případě, že hledané zařízení není v lokální podsíti, je vrácena fyzická adresa směrovače, který zajistí směrování k hledanému zařízení. [66, 42, 18, 30]

Internet Control Message Protocol (ICMP) slouží k předávání řídících informací a chybových zpráv. Pomocí tohoto protokolu dokáže brána upozornit zařízení, že nepoužívá optimální cestu k přenosu dat do koncové sítě. Dochází tak k učení směrovačů. Při odeslání paketu s postupně se zvyšujícím TTL se očekává ICMP odpověď, díky které lze sledovat cestu k danému uzlu. [51, 25]

Internet Protocol (IP) zajišťuje přenos dat na úrovni sítové vrstvy. Nepovinná část datagramu (základní datová jednotka tohoto protokolu) může obsahovat informace o tom, kterou cestou má být datagram směrován, nebo o tom, kterou cestou byl datagram směrován.



Obrázek 3.4: Znázornění ARP protokolu.

Kapitola 4

Analýza a návrh řešení

4.1 Analýza

4.1.1 Rozpoznávání změn

Změny v topologii sítě se dají detekovat několika způsoby.

Odposloucháváním na síti lze sledovat jaká zařízení a s využitím jakých prostředníků spolu komunikují. Stejně tak jsou důležitá data, která si posílají. Následně pak srovnáním s uloženým modelem topologie lze detekovat změny. Tato metoda ovšem nemusí detekovat všechny změny, jelikož všechna zařízení nemusí vždy komunikovat. Ze stejného důvodu je i samotná detekce velmi pomalá.

Opakovanou detekcí topologie a porovnáváním výsledků lze velmi dobře nalézat změny. Problémem však je to, že detekce topologie může trvat velmi dlouho. Tento nedostatek v kombinaci s předpokladem, že se topologie bude měnit pomaleji než je perioda detekce, povede k plýtvání systémovými zdroji a zbytečnému zatěžování systému.

Dalším způsobem je nasazení systému pro monitorování sítě. Takový systém zejména v pravidelných intervalech (ale lze použít i jiné mechanizmy) kontroluje dostupnost (ať už pomocí běžných síťových protokolů nebo procesů spuštěných přímo na zařízení) a parametry monitorovaných síťových zařízení (ať už vytížení procesoru nebo třeba síťovou adresu) a služeb na nich běžících. V případě detekce změny nějakého parametru je uživatel zvoleným způsobem upozorněn. Bohužel monitorovací systém často nesleduje propojení monitorovaných prvků. Z generovaných upozornění sice lze odvodit změny v topologii, ale ne všechny a tyto úvahy nemají zaručeně správný výsledek.

Jako nejlepší řešení se jeví kombinace výše zmíněných řešení. Tj. v případě, že monitorovací systém zaznamená změnu parametru nějakého zařízení, upozorní na tuto situaci událostí. Na tuto událost bude reagovat systém, který doplní informace o vzniklé situaci o stav aktuální topologie. Tato data se porovnají s uloženým modelem topologie sítě a výsledkem bude seznam změn v této topologii.

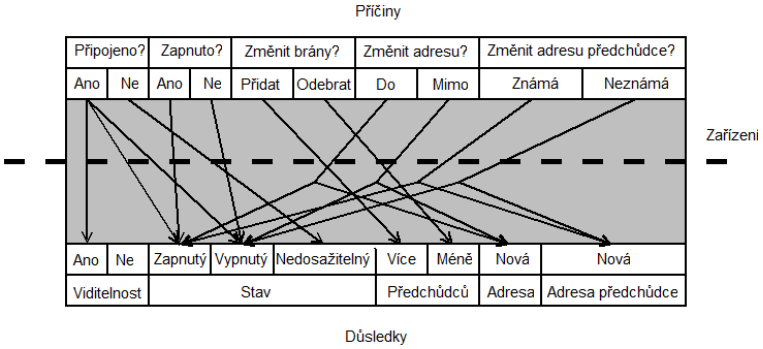
4.1.2 Rozpoznávání příčin změn

Správci sítě by usnadnil práci nejen systém, který umí změny detekovat, ale také diagnostikovat jejich příčiny. Níže je uveden výčet všech událostí a jejich projevů v síti.

- Dojde-li k připojení zařízení A svým vstupním rozhraním[1] do sítě, musí nastat jeden z následujících projevů:
  - V topologii přibude nové zařízení A.
  - Zařízení A změní svůj stav z nedosažitelné.
- Dojde-li k odpojení zařízení A svým vstupním rozhraním od sítě, musí nastat následující projev:
  - Zařízení A změní svůj stav na nedosažitelné.
- Dojde-li k zapnutí zařízení A, musí nastat následující projev:
  - Zařízení A změní svůj stav z vypnuté na zapnuté.
- Dojde-li k vypnutí zařízení A, musí nastat následující projev:
  - Zařízení A změní svůj stav ze zapnuté na vypnuté.
- Dojde-li k přidání bran zařízení A, musí nastat následující projev:
  - Zařízení A přibudou předchůdci.
- Dojde-li k odebrání bran zařízení A, musí nastat následující projev:
  - Zařízení A ubudou předchůdci.
- Dojde-li ke změně logické adresy vstupního rozhraní zařízení A z adresy mimo rozsah adres místní podsítě do rozsahu adres místní podsítě, musí nastat následující projevy:
  - Zařízení A změní svůj stav z vypnuté na zapnuté.
  - Zařízení A se změní logická adresa vstupního rozhraní.
- Dojde-li ke změně logické adresy vstupního rozhraní zařízení A z adresy v rozsahu adres místní podsítě mimo rozsah adres místní podsítě, musí nastat následující projevy:
  - Zařízení A změní svůj stav ze zapnuté na vypnuté.
  - Zařízení A se změní logická adresa vstupního rozhraní.
- Dojde-li ke změně logické adresy výstupního[2] rozhraní zařízení předcházejícího zařízení A, musí nastat následující projevy:
  - Zařízení A změní svůj stav z vypnuté na zapnuté (pokud se adresa změnila na jemu známou), nebo ze zapnuté na vypnuté (pokud se adresa změnila na jemu neznámou).
  - Zařízení předcházejícímu zařízení A se změní logická adresa výstupního rozhraní.

[1] Vstupním rozhraním se myslí rozhraní mezi zařízením, jemuž rozhraní patří, a zařízením, které je blíže k monitorovacímu systému (předchůdce).

[2] Výstupním rozhraním se myslí rozhraní mezi zařízením, jemuž rozhraní patří, a zařízením, které je dále od monitorovacího systému (následník).



Obrázek 4.1: Znázornění vztahů příčin a důsledků.

4.2 Návrh řešení

Algoritmus je nejprve nutné inicializovat počátečním stavem topologie. To lze provést jak spuštěním *detekce topologie*, tak *získáním z monitorovacího systému*. Následně se algoritmus přepne do režimu, kdy čeká na nějakou událost. V tomto případě *události generuje monitorovací systém*. Jakmile dojde k *zachycení události*, *detekuje se aktuální stav topologie*. Ten se doplní o informace ze zachycené události a o zařízení, která nebyla detekována. Porovnáním aktuální topologie s topologií uloženou se *získá seznam změn a jejich příčin*. Jestliže se detekují nějaké změny, jsou *uloženy do monitorovacího systému* a zároveň *vypsány na* zvolený *výstup*. Následně se vše opakuje bez počáteční inicializace.

Ke vzájemné komunikaci mezi procesy lze použít několik prostředků: soubor, standardní vstup a výstup, sockety, signály atd. Nejuniverzálnějším způsobem jsou soubory. Pokud totiž proces nepoužívá soubory, zpravidla lze jím zvolený prostředek do souboru přesměrovat, pokud není souborem přímo realizován. Informace ze souboru se zpravidla získávají pomocí *parseru*. Jeden ze způsobů jak by mohl parser fungovat je takový, že se ze souboru načtou znaky (množství odpovídá definované podmínce - často to bývá množství znaků od začátku do konce řádku) do bufferu. Na buffer se následně vyzkouší množina regulárních výrazů, které odpovídají aktuálnímu stavu parseru (parser bude tedy fungovat jako stavový automat). Odpovídá-li buffer nějakým výrazů, jsou vyvolány funkce přiřazené těmto výrazům. Tyto funkce pak například mohou změnit stav parseru, uložit nějaké hodnoty do paměti apod. Následně se celý proces opakuje do splnění ukončovací podmínky (tou je zpravidla konec souboru).

Detekci topologie lze jednak implementovat pomocí protokolů popsaných výše a jednak je možné použít již existující program, který vstup přijímá z příkazové řádky a pro výstup používá soubory, nebo jiné prostředky, které lze přesměrovat do souboru. V obou případech se pak soubor s výstupem předloží odpovídajícímu parseru.

Získávání a ukládání informací do monitorovacího systému může probíhat také pomocí parseru, který rozumí syntaxi konfiguračních souborů daného monitorovacího systému. Daný monitorovací systém samozřejmě musí mít nastavení uloženo v konfiguračních souborech.

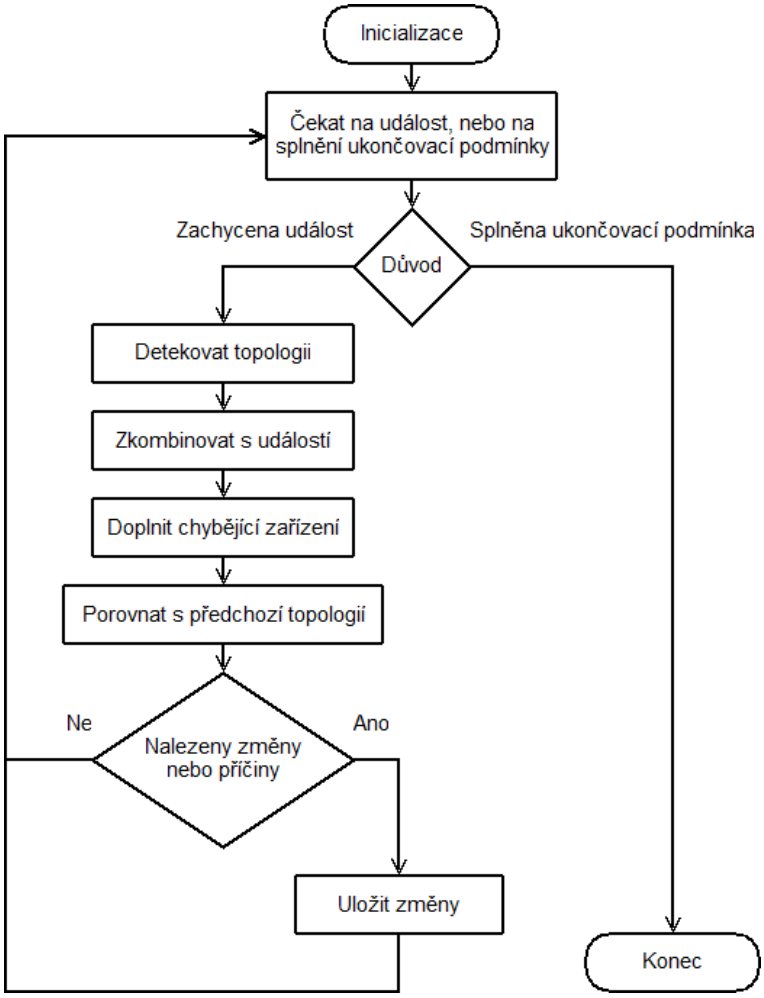
Monitorovací systém opět musí své *události generovat* do souboru, nebo pomocí jiných prostředků, které lze přesměrovat do souboru. Tento soubor je zpracováván odpovídajícím parserem.

Jsou-li události ukládány do souboru, existuje několik variant jak *zjišťovat změny v souboru*, které mohou a nemusí být podporovány daným operačním systémem a použitým programovacím jazykem a jeho prostředím. Lze například sledovat datum poslední změny souboru (zpravidla držený jako atribut daného souboru), lze porovnávat obsah souboru, lze porovnávat hash otisk souboru, lze porovnávat počet řádků souboru, lze porovnávat velikost souboru, lze zaregistrovat zvolenou funkci jako pozorovatel (observer) změn v daném souboru (tímto se starost o sledování změn předá operačnímu systému, programovacímu jazyku a jeho prostředí) nebo si také lze pamatovat poslední pozici ukazatele v souboru a po zvolené periodě pokračovat ve čtení souboru od poslední pozice.

Získávání změn v topologii a jejich příčin probíhá tak, že se u každého zařízení zjistí, zda došlo k nějakým změnám. Pokud ano, jsou uloženy do paměti a jejich příčiny se diagnostikují pomocí obrácení[3] definice v sekci [4.1.2]. Jestliže dané zařízení má nějaké následníky, jsou příčiny propagovány i těmto zařízením.

Výpis změn a jejich příčin na výstup pak probíhá do zvoleného souboru pomocí parseru odpovídajícímu zvolenému formátu souboru.

[3]Obrácení znamená, že projevy budou implikovat jejich příčiny.



Obrázek 4.2: Znázornění navržené metody.

4.3 Volba implementačního prostředí

Navržený program je možné implementovat v jakémkoliv jazyce, který podporuje práci se soubory a spouštění externích programů. Tyto vlastnosti mají všechny nejrozšířenější běžně používané programovací jazyky. Pro implementaci byl nakonec zvolen jazyk *Python*, jelikož filozofie vývojářů tohoto jazyka je autorovi nejbližší. Jazyk Python je vyvíjen pod svobodnou licencí komunitou programátorů Python Software Foundation, v jejímž čele stojí Guido van Rossum, nizozemský programátor hrající zásadní roli v šíření myšlenky *svobodného softwaru*. Tento jazyk je interpretovaný, velmi orientovaný na objektové programování, má srozumitelnou a přehlednou syntaxi, umožňuje velmi rychlý vývoj jak skriptů, tak rozsáhlých projektů, má velmi dobrou dokumentaci [16], dostupnou podporu a velké množství publikovaných knihoven. Rovněž množství existující aplikací poskytuje API pro programy psané v jazyce Python. Interpret jazyka je dostupný nejen pro všechny nejrozšířenější operační

systémy (Microsoft Windows, operační systémy nad jádrem Linux, Apple Mac OS), což je jeden z požadavků na implementaci navržené metody.

Byla zvolena verze *2.5.4*, která zaručuje kompatibilitu s poslední verzí mutace Jython (implementace jazyka Python interpretovaná Java Virtual Machine). Použitím mutace Jython se následně dá docílit kompatibility s operačními systémy, pro něž není dostupný Python interpret, ale pro něž je dostupný Java Virtual Machine.

Aplikace je vyvinuta v prostředí operačního systému *Microsoft Windows 7 Enterprise*, jakožto jediného operačního systému používaného autorem. Ovšem při vývoji byl kladen důraz na kompatibilitu s ostatními operačními systémy, což velmi usnadnil zvolený programovací jazyk.

Zdrojové kódy jsou napsány ve vývojovém prostředí *Eclipse 3.5.2* vyvíjeném komunitou The Eclipse Foundation s doplňkem *Aptana PyDev 1.5.6*, jakožto poslední (autorem nainstalované) verze nejčastěji používaného svobodného multiplatformního prostředí pro vývoj aplikací v jazyce Python.

Kapitola 5

Realizace

Cílem práce bylo navrženou metodu implementovat do některého open-source monitorovacího systému. Program vzniklý touto implementací byl nazván *TopologyWoo*.

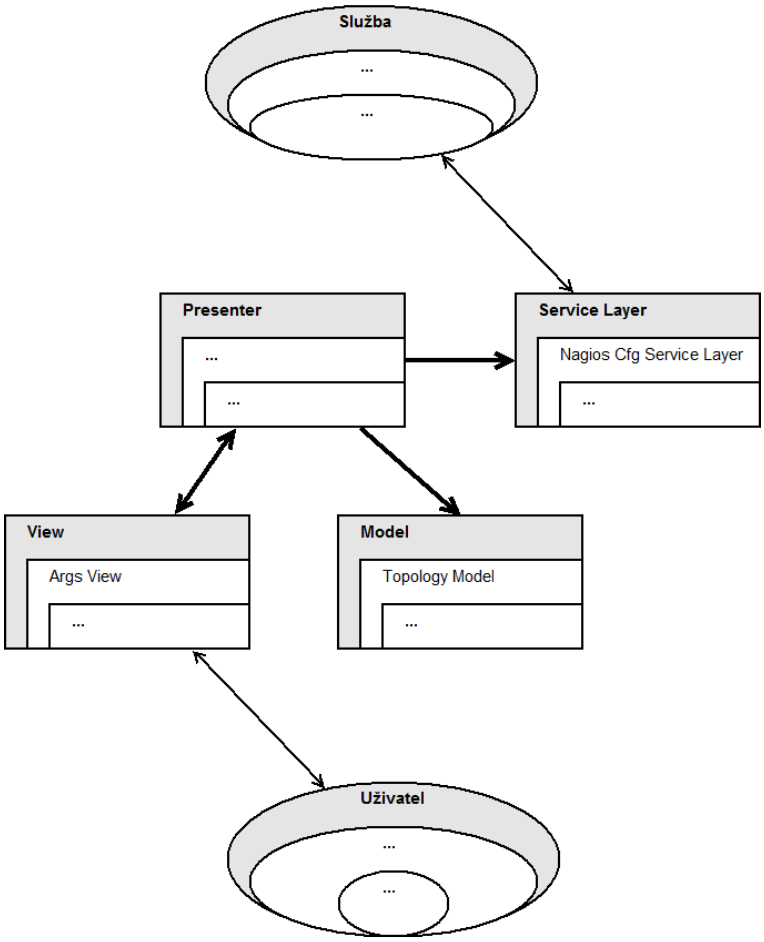
Program se člení na čtyři základní druhy komponent. *Model* reprezentuje data a funkce, které s nimi zachází. *View* reprezentuje rozhraní mezi uživatelem a programem. *Presenter* reprezentuje funkce aplikace a propojuje komponenty model, view a service layer. *Service layer* reprezentuje rozhraní mezi programem a externími aplikacemi a službami.

Při implementaci byl kladen důraz na možnost program libovolně konfigurovat a v budoucnu rozšířit nebo *znovu-použít*[1]. Program podporuje následující funkce:

- Načtení počátečního stavu topologie z monitorovacího systému Nagios.
- Vyžádání aktuálního stavu topologie z falešného programu.
- Načtení výstupu z programu zjišťujícího topologii ve formátu souborů systému Nagios.
- Uložení změn v topologii do souboru ve formátu XML.
- Uložení změn v topologii do souborů ve formátu souborů systému Nagios.
- Nastavení frekvence kontroly souborů s událostmi.
- Sledování souboru s událostmi ve formátu souborů systému Nagios.

Pro dosažení maximální míry kompatibility s ostatními aplikacemi bylo uživatelské rozhraní realizováno pomocí parametrů příkazového řádku, standardního výstupu (zpravidla konzole) a souborů.

[1] Výraz znovu-použít byl odvozen od slova znovupoužitelnost (anglicky reusability).



Obrázek 5.1: Znázornění rozdělení programu na čtyři druhy komponent.

5.1 Nagios

Pro realizaci byl zvolen open-source monitorovací systém Nagios vyvíjený společností Nagios Enterprises, protože je to jeden z nejpoužívanějších monitorovacích systémů a autor měl možnost získat data ze systému, který běží v reálné síti.

Soubory systému Nagios, které jsou užitečné pro program TopologyWoo, mají dva různé formáty. Ty lze od sebe rozlišit podle přípony názvu souboru.

První nejdůležitější formát systému Nagios je formát souborů s příponou *cfg* [13]. Tyto soubory jsou určeny ke konfiguraci tohoto systému. Z pohledu parseru, který je implementován v programu TopologyWoo, je důležité, že konfigurace je rozdělena na sekce, přičemž každý parametr dané sekce se nachází na zvláštním řádku. Pro tento parser jsou podstatné pouze sekce, které definují zařízení v síti. Proto parser do bufferu vždy načítá celý řádek a v tomto řádku hledá klíčové řetězce znaků „`address`“, „`alias`“, „`define`“, „`display_name`“, „`host_name`“, „`name`“, „`parents`“, „`register`“, „`statusmap_image`“, „`use`“a „}“. V konfiguračních souborech může být definována i dědičnost vlastností z jiných zařízení, jejíž pod-

pora je v parseru také zahrnuta. Implementovaný parser podporuje čtení a editaci (změna hodnoty daného řádku při zachování zbytku souboru) těchto souborů.

```
define host{
    use                     host,host-pnpng
    host_name               workstation.domain.com
    address                 192.12.13.14
    alias                   Workstation
    parents                 192.12.13.1
    statusmap_image         debian.png
    check_command           check-alive-tcpng!22
    contact_groups          default
    notifications_enabled 1
    check_interval          5
    notes_url               http://wiki.domain.com/
    _QUEUE                  x5
}
define service{
    use                     generic_service,srv-pnpng
    host_name               workstation.domain.com
    service_description     host-availability
    notifications_enabled 0
    contact_groups          default
    normal_check_interval 10
    retry_check_interval  2
    max_check_attempts    5
    check_command           check-alive-tcpng!22
    _QUEUE                  x5
}
```

Ukázka 1: Ukázka *.cfg souboru systému Nagios.

Druhý formát systému Nagios je formát souborů s příponou *log*. Tyto soubory jsou určeny k zaznamenávání událostí tohoto systému. Z pohledu parseru, který je implementován v programu TopologyWoo, je důležité, že každá událost je zaznamenána na jednom řádku, přičemž každý parametr události je oddělen středníkem. Pro tento parser jsou podstatné pouze události, které se týkají zařízení v síti. Proto parser do bufferu vždy načítá pouze sekvenci znaků do prvního výskytu středníku a v této sekvenci hledá klíčové řetězce znaků „0“, „1“, „2“, „CURRENT HOST STATE“, „DOWN“, „EXTERNAL COMMAND“, „GLOBAL HOST EVENT HANDLER“, „HOST ALERT“, „HOST NOTIFICATION“, „PASSIVE HOST CHECK“, „PROCESS_HOST_CHECK_RESULT“, „UNREACHABLE“a „UP“. Implementovaný parser podporuje čtení těchto souborů.

```
[1273788000] CURRENT HOST STATE: domain.com;UP;HARD;1;ALWAYS UP OK
[1273803531] HOST ALERT: domain.com;DOWN;SOFT;1;CRITICAL - Host Unreachable
```

Ukázka 2: Ukázka *.log souboru systému Nagios.

5.2 Falešný program na zjišťování topologie sítě

Autor se rozhodl preferovat snadnou možnost ověření všech myslitelný situací v síti, proto byla do programu implementována funkce, využívající falešný program, který ve skutečnosti nedělá nic. Do adresáře, jehož cesta byla programu TopologyWoo předána jako cesta k adresáři obsahující výstup programu na zjišťování topologie sítě, je třeba umístit výstup, který by vrátil skutečný program. V programu je implementována funkce, která tyto vstupy čte ve formátu *.cfg souborů systému Nagios.

5.3 Model

Model je druh komponenty implementovaný třídou, která reprezentuje data a funkce, které s nimi zachází. Celou topologii spravuje jeden model, který drží seznam dalších modelů představujících zařízení v síti. Tato zařízení pak obsahují seznam jmen zařízení, se kterými sousedí. Tato struktura se obecně nazývá strom. V tomto konkrétním případě pak uzly stromu odpovídají jednotlivým zařízením a hrany stromu odpovídají sousednosti těchto zařízení.

Změny jsou také uloženy v modelu společně s příčinami. Změny jsou implementovány jako asociativní pole (v jazyku Python nazýván slovník, v jazyku Java mapa), kde klíčem je `hostname` zařízení a hodnotou seznam referencí na modely tohoto zařízení v původní topologii a v aktuální topologii. Příčiny jsou implementovány jako asociativní pole, kde klíčem je `hostname` zařízení a hodnotou seznam konstant reprezentujících dané příčiny.

5.4 XML výstup

Za účelem vypisování výstupu (a umožnění jiným aplikacím tento výstup zpracovávat) byl pro tento program definován vlastní formát souboru používající jazyk XML. První sekce tohoto formátu obsahuje definice všech příčin změn v topologii. Druhá sekce obsahuje seznam změn v topologii a odkazů na jejich příčiny.

Popis jednotlivých značek je následující:

- Značka `changes` reprezentuje seznam změn a jejich příčin. Může obsahovat značku `causes` a značku `effects`.
- Značka `causes` reprezentuje seznam příčin změn. Může obsahovat značky `hostcause`.
- Značka `hostcause` reprezentuje zařízení jakožto příčinu. Musí obsahovat atributy `id`, `hostname` a `action`. Atribut `id` reprezentuje identifikátor příčiny. Atribut `hostname` reprezentuje `hostname` zařízení. Atribut `action` reprezentuje akci zařízení.

- Značka `effects` reprezentuje seznam změn v topologii. Může obsahovat značky `host`.
- Značka `host` reprezentuje změnu v zařízení. Musí obsahovat atribut `hostname` a `new`. Může obsahovat značky `state`, `address`, `parent` a `references`. Atribut `hostname` reprezentuje `hostname` zařízení. Hodnota `True` atributu `new` znamená, že zařízení je v topologii nové.
- Značka `state` reprezentuje změnu ve stavu zařízení. Může obsahovat atribut `old` a musí obsahovat atribut `new`. Atribut `old` reprezentuje původní stav. Atribut `new` reprezentuje nový stav.
- Značka `address` reprezentuje změnu v adrese zařízení. Může obsahovat atribut `old` a musí obsahovat atribut `new`. Atribut `old` reprezentuje původní adresu. Atribut `new` reprezentuje novou adresu.
- Značka `parent` reprezentuje změnu v předchůdcích zařízení. Může obsahovat atributy `old` a `new`. Atribut `old` reprezentuje `hostname` předchůdce, který ubyl. Atribut `new` reprezentuje `hostname` předchůdce, který přibyl.
- Značka `references` reprezentuje odkazy na příčiny těchto změn. Může obsahovat značku `or`.
- Značka `or` reprezentuje operátor logického nebo. Může obsahovat značky `causeref`.
- Značka `causeref` reprezentuje odkaz na příčinu těchto změn. Musí obsahovat atribut `id`. Atribut `id` reprezentuje identifikátor příčiny.

V programu je implementován parser, který zapisuje soubory v tomto formátu.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT changes (causes?,effects?)>

<!ELEMENT causes (hostcause*)>

<!ELEMENT hostcause EMPTY>
<!ATTLIST hostcause id CDATA #REQUIRED>
<!ATTLIST hostcause hostname CDATA #REQUIRED>
<!ATTLIST hostcause action CDATA #REQUIRED>

<!ELEMENT effects (host*)>

<!ELEMENT host (state?,address?,parent*,references?)>
<!ATTLIST host hostname CDATA #REQUIRED>
<!ATTLIST host new (True|False) #REQUIRED>

<!ELEMENT state EMPTY>
<!ATTLIST state old CDATA #IMPLIED>
<!ATTLIST state new CDATA #REQUIRED>

<!ELEMENT address EMPTY>
<!ATTLIST address old CDATA #IMPLIED>
<!ATTLIST address new CDATA #REQUIRED>

<!ELEMENT parent EMPTY>
<!ATTLIST parent old CDATA #IMPLIED>
<!ATTLIST parent new CDATA #IMPLIED>

<!ELEMENT references (or?)>

<!ELEMENT or (causeref*)>

<!ELEMENT causeref EMPTY>
<!ATTLIST causeref id CDATA #REQUIRED>
```

Ukázka 3: Popis struktury (DTD) nového formátu.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE changes SYSTEM "http://pydeq.name/My/topologywoo.dtd">
<changes>
    <causes>
        <hostcause id="0" hostname="r16" action="turned on" />
        <hostcause id="1" hostname="r16" action="in address in range" />
        <hostcause id="2" hostname="r16" action="prev out address in range" />
        <hostcause id="3" hostname="r16" action="gateway added" />
        <hostcause id="8" hostname="w17" action="in interface connected" />
        <hostcause id="9" hostname="w17" action="gateway added" />
    </causes>
    <effects>
        <host hostname="r16" new="False">
            <state old="down" new="up" />
            <parent new="r1" />
            <references>
                <or>
                    <causeref id="0" />
                    <causeref id="1" />
                    <causeref id="2" />
                    <causeref id="3" />
                </or>
            </references>
        </host>
        <host hostname="w17" new="False">
            <state old="unreachable" new="up" />
            <parent new="r16" />
            <references>
                <or>
                    <causeref id="0" />
                    <causeref id="1" />
                    <causeref id="2" />
                    <causeref id="3" />
                    <causeref id="8" />
                    <causeref id="9" />
                </or>
            </references>
        </host>
    </effects>
</changes>
```

Ukázka 4: Ukázka souboru s novým formátem.

5.5 Sledování událostí

Jazyk Python podporuje sledování změn v souborech, ale jen na některých operačních systémech. Proto bylo implementováno sledování událostí v souboru pomocí periodického čtení tohoto souboru s tím, že se uloží poslední pozice kurzoru v souboru a při příštím čtení se pokračuje od této pozice. Tento mechanizmus nebude fungovat, pokud monitorovací systém bude do souboru zapisovat jinam, než na konec. Proto je podmínkou, aby zapisoval pouze na konec souboru.

5.6 Získávání změn v topologii a jejich příčin

Získávání změn v topologii a jejich příčin probíhá tak, jak je popsáno v návrhu řešení. Procházení všech zařízení a propagace příčin dalším zařízením využívá algoritmu prohledávání do hloubky nad stromem uloženým v modelu topologie s kořenem v zařízení s monitorovacím systémem. Algoritmus *prohledávání do hloubky* [47, 22] je rekurzivní (dá se implementovat i iterativně) grafový algoritmus, který vždy v aktuálním vrcholu provede danou akci a následně je postupně zavolán na všechny následníky aktuálního uzlu, kteří ještě nebyli navštíveni.

5.7 Zajímavé části zdrojového kódu

Jelikož jazyk Python umožňuje předávání ukazatele na funkci, podařilo se implementovat algoritmus prohledávání do hloubky obecně. Bylo tedy možné jej v programu volat vícekrát tak, aby pokaždé prováděl jinou akci.

```
class _TopologyModel(_Model):
    def _depthfirstsearch(self, processor, roothostname, topology,
                          *processorargs):
        expanded = set()
        stack = [roothostname]
        while len(stack) > 0:
            hostname = stack.pop()
            if processor(hostname, topology, *processorargs):
                for child in topology.gethost(hostname).children:
                    if child.hostname not in expanded:
                        stack.append(child.hostname)
                        expanded.add(hostname)

    def detecttopologychanges(self, newtopology):
        ...
        self._depthfirstsearch(self.__processtopologychangesdetection,
                               roothostname, newtopology, changes)
        ...

    def __processimpliedtopologychanges(self, hostname, newtopology,
                                        direction, changes, *args):
        ...

    def __processtopologychangesdetection(self, hostname, newtopology,
                                          changes, *args):
        ...
        self._depthfirstsearch(self.__processimpliedtopologychanges,
                               newchild.hostname, newtopology, +1,
                               changes)
        ...
```

Ukázka 5: Ukázka zdrojového kódu implementace algoritmu prohledávání do hloubky.

Ze stejného důvodu se podařilo implementovat velmi přehledné volání funkcí odpovídajících daným regulárním výrazům v parseru.

```
class _NagiosLogServiceLayer(_ServiceLayer):
    def __init__(self, file=None, directory=None):
        ...
        self._processors = {...
                            "EXTERNAL COMMAND":
                            self.__processtopologydoubledirective,
                            ...
                            "HOST NOTIFICATION":
                            self.__processtopologyskippingdirective,
                            ...}
        ...

    def _loadfile(self):
        ...
        key = match.groupdict()[self.KEY_REGEX_GROUP_NAME]
        value = match.groupdict()[self.VALUE_REGEX_GROUP_NAME]
        if key in self._processors:
            self._processors[key](value)
        ...

    def __processtopologydoubledirective(self, value):
        self._state = self.DOUBLE_DIRECTIVE_STATE

    def __processtopologyskippingdirective(self, value):
        self._state = self.SKIPPING_DIRECTIVE_STATE
```

Ukázka 6: Ukázka zdrojového kódu implementace funkce parseru.

5.8 Rozšíření programu

Program byl psán s důrazem na možnost rozšířit jej o podporu dalších formátů. Přidání podpory dalšího formátu spočívá v implementaci další service layer komponenty, která drží parser pro daný formát. Přizpůsobení obecného parseru děděného od nadtřídy spočívá jen v deklaraci možných stavů parseru, regulárních výrazů pro dané stavy, metod přiřazených jednotlivým výrazům a abstraktních metod (metody pro načtení řetězce znaků do bufferu, metody vracející výsledek parseru, případně metody na ukládání do souboru, nebo editaci souboru).

Kapitola 6

Testování

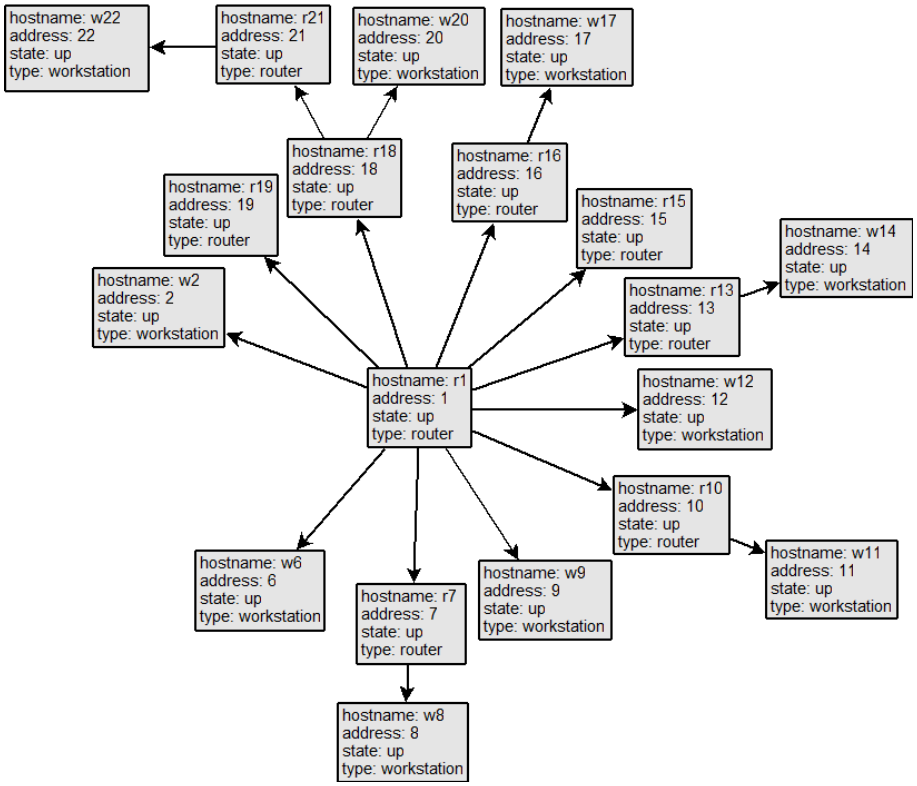
Program byl spouštěn interpretem Python 2.5.4 na operačním systému Microsoft Windows 7 Enterprise. Z výše uvedených důvodů je testování poměrně komplikované. Pro každou iteraci programu, která testuje stav topologie v daný okamžik, je nutné simulovat výstup programu na zjišťování topologie sítě. Tento výstup je pak nutné ve správný okamžik umístit do daného adresáře.

6.1 Testovací data

Jelikož však autor neměl k dispozici dostatečně velkou monitorovanou počítačovou síť, bylo nutné simulovat i vstup a výstup monitorovacího systému. Zato bylo možné otestovat většinu případů, které mohou nastat.

Před spuštěním programu je nutné simulovat počáteční stav topologie.

- Je vytvořen prázdný soubor `test.log` - ten odpovídá situaci, kdy monitorovací systém ještě nevygeneroval žádné události.
- Je vytvořen prázdný adresář `request` - ten odpovídá nedostupnému výsledku programu na zjišťování topologie sítě.
- Je vytvořen prázdný adresář `output` určený na výstup programu.
- Do adresáře `input` je umístěn soubor `test.cfg` - ten odpovídá konfiguraci monitorovacího systému v tento okamžik.



Obrázek 6.1: Znázornění počátečního stavu testovací topologie.

Program je spuštěn pomocí následujícího příkazu:

```
topologywoo\__main__.py --loadnagiostopology input \
  --requestfaketopologycommand --requestnagiostopologyoutput request \
  --savexmltopologychanges output --savenagiostopology input \
  --watchingrate 45 --watchnagiostopology test.log
```

Ten odpovídá požadavku na monitorování změn topologie sítě s využitím monitorovacího systému Nagios, který jednak poskytuje počáteční stav topologie (adresář `input`), jednak generuje události (soubor `test.log`) a jednak do něho jsou zpětně ukládány změny v topologii (také adresář `input`). Další použité parametry příkazového řádku říkají, že má být použit falešný program na detekci topologie sítě (jehož výstup bude v adresáři `request` ve formátu souborů systému Nagios) a že se změny v topologii mají ukládat ve vlastním formátu definovaném pomocí jazyka XML (adresář `output`). Soubor `test.log` bude testován s periodou 45 sekund.

V dalším kroku je simulováno připojení počítače $w3$, připojení směrovače $r4$, odpojení počítače $w6$, odpojení směrovače $r7$, vypnutí počítače $w12$ a vypnutí směrovače $r13$.

- Do souboru `test.log` je přidán obsah - ten odpovídá událostem monitorovacího systému v tento okamžik.
- Do adresáře `request` je umístěn soubor `test.cfg` - ten odpovídá výstupu programu na zjišťování topologie sítě v tento okamžik.

V dalším kroku je simulováno připojení počítače $w6$, připojení směrovače $r7$, vypnutí počítače $w9$, vypnutí směrovače $r10$, zapnutí počítače $w12$, zapnutí směrovače $r13$, nastavení adresy vstupního rozhraní směrovače $r15$ mimo rozsah adres místní podsítě a nastavení adresy vstupního rozhraní směrovače $r16$ mimo rozsah adres místní podsítě.

- Do souboru `test.log` je přidán obsah - ten odpovídá událostem monitorovacího systému v tento okamžik.
- V adresáři `request` je nahrazen soubor `test.cfg` - ten odpovídá výstupu programu na zjišťování topologie sítě v tento okamžik.

V posledním kroku je simulováno nastavení adresy vstupního rozhraní směrovače $r15$ do rozsahu adres místní podsítě, nastavení adresy vstupního rozhraní směrovače $r16$ do rozsahu adres místní podsítě, vypnutí počítače $r18$ a nahrazení nyní nefunkční cesty k směrovače $r21$ změnou jeho brány na směrovač $r19$ a změnou adresy jeho vstupního rozhraní.

- Do souboru `test.log` je přidán obsah - ten odpovídá událostem monitorovacího systému v tento okamžik.
- V adresáři `request` je nahrazen soubor `test.cfg` - ten odpovídá výstupu programu na zjišťování topologie sítě v tento okamžik.

```
...
+       define host{
+           host_name             w17
+           address               17
+           statusmap_image       linux.png
+           parents               r16
+       }
+
...
-       define host{
-           host_name             w20
-           address               20
-           statusmap_image       linux.png
-           parents               r18
-       }
-
.       define host{
*           host_name             r21
.           address               211
.           statusmap_image       router.png
*           parents               r19
.       }
```

Ukázka 7: Ukázka části rozdílu nově nahrávaného souboru `request/test.cfg` oproti souboru z předchozí iterace.

Značky `...`, `+`, `-`, `*` a `.` na začátku řádku v ukázce [8] nejsou součástí souboru, nýbrž označují řádek, který v původním souboru nebyl (`+`), řádek, který není v novém souboru (`-`), nezměněný řádek (`.`), změněný řádek (`*`) a vynechané řádky (`...`).

```
[1273788000] CURRENT HOST STATE: r15;UP;HARD;1;ALWAYS UP OK
[1273788000] CURRENT HOST STATE: r16;UP;HARD;1;ALWAYS UP OK
[1273788000] CURRENT HOST STATE: w17;UP;HARD;1;ALWAYS UP OK
[1273788000] CURRENT HOST STATE: r18;DOWN;HARD;1;ALWAYS UP OK
[1273788000] CURRENT HOST STATE: w20;UNREACHABLE;HARD;1;ALWAYS UP OK
```

Ukázka 8: Ukázka obsahu přidaného do souboru `test.log` v aktuální iteraci.

6.1.1 Výsledky

Po prvním kroku došlo k odpovídajícím změnám v souboru `input/test.cfg`, přibyl soubor `input/new.cfg` s novými zařízeními a přibyl nový soubor `output/100326024704.xml`, který říká, že:

- Přibylo nové zařízení $w3$, protože
  - *zařízení* $w3$ *bylo připojeno vstupním rozhraním* nebo
  - zařízení $w3$ přibyla brána.
- Přibylo nové zařízení $r4$, protože
  - *zařízení* $r4$ *bylo připojeno vstupním rozhraním* nebo
  - zařízení $r4$ přibyla brána.
- Přibylo nové zařízení $w5$, protože
  - *zařízení* $r4$ *bylo připojeno vstupním rozhraním* nebo
  - zařízení $r4$ přibyla brána nebo
  - zařízení $w5$ bylo připojeno vstupním rozhraním nebo
  - zařízení $w5$ přibyla brána.
- Zařízení $w6$ se změnil stav ze zapnuté na nedosažitelné a ubyl předchůdce $r1$, protože
  - *zařízení* $w6$ *bylo odpojeno vstupním rozhraním* nebo
  - zařízení $w6$ ubyla brána.
- Zařízení $r7$ se změnil stav ze zapnuté na nedosažitelné a ubyl předchůdce $r1$, protože
  - *zařízení* $r7$ *bylo odpojeno vstupním rozhraním* nebo
  - zařízení $r7$ ubyla brána.
- Zařízení $w8$ se změnil stav ze zapnuté na nedosažitelné a ubyl předchůdce $r7$, protože
  - *zařízení* $r7$ *bylo odpojeno vstupním rozhraním* nebo
  - zařízení $r7$ ubyla brána nebo
  - zařízení $w8$ bylo odpojeno vstupním rozhraním nebo
  - zařízení $w8$ ubyla brána.
- Zařízení $w12$ se změnil stav ze zapnuté na vypnuté a ubyl předchůdce $r1$, protože
  - *zařízení* $w12$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $w12$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $w12$ se změnila mimo rozsah adres místní podsítě nebo

  - zařízení $w12$ ubyla brána.

- Zařízení $r13$ se změnil stav ze zapnuté na vypnuté a ubyl předchůdce $r1$, protože
  - *zařízení* $r13$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r13$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r13$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r13$ ubyla brána.

- Zařízení $w14$ se změnil stav ze zapnuté na nedosažitelné a ubyl předchůdce $r13$, protože
  - *zařízení* $r13$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r13$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r13$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r13$ ubyla brána nebo
  - zařízení $w14$ bylo odpojeno vstupním rozhraním nebo
  - zařízení $w14$ ubyla brána.

Po druhém kroku došlo k odpovídajícím změnám v souboru `input/test.cfg` a přibyl nový soubor `output/100326024749.xml`, který říká, že:

- Zařízení $w6$ se změnil stav z nedosažitelné na zapnuté a přibyl předchůdce $r1$, protože
  - *zařízení* $w6$ *bylo připojeno vstupním rozhraním* nebo
  - zařízení $w6$ přibyla brána.

- Zařízení $r7$ se změnil stav z nedosažitelné na zapnuté a přibyl předchůdce $r1$, protože
  - *zařízení* $r7$ *bylo připojeno vstupním rozhraním* nebo
  - zařízení $r7$ přibyla brána.

- Zařízení $w8$ se změnil stav z nedosažitelné na vypnuté, protože
  - zařízení $w8$ bylo připojeno vstupním rozhraním.

- Zařízení $w9$ se změnil stav ze zapnuté na vypnuté a ubyl předchůdce $r1$, protože
  - *zařízení* $w9$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $w9$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $w9$ se změnila mimo rozsah adres místní podsítě nebo

  - zařízení $w9$ ubyla brána.

- Zařízení $r10$ se změnil stav ze zapnuté na vypnuté a ubyl předchůdce $r1$, protože
  - *zařízení* $r10$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r10$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r10$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r10$ ubyla brána.

- Zařízení $w11$ se změnil stav ze zapnuté na nedosažitelné a ubyl předchůdce $r10$, protože
  - *zařízení* $r10$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r10$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r10$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r10$ ubyla brána nebo
  - zařízení $w11$ bylo odpojeno vstupním rozhraním nebo
  - zařízení $w11$ ubyla brána.

- Zařízení $w12$ se změnil stav z vypnuté na zapnuté a přibyl předchůdce $r1$, protože
  - *zařízení* $w12$ *bylo zapnuto* nebo
  - adresa vstupního rozhraní zařízení $w12$ se změnila do rozsahu adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $w12$ se změnila do rozsahu adres místní podsítě nebo
  - zařízení $w12$ přibyla brána.

- Zařízení $r13$ se změnil stav z vypnuté na zapnuté a přibyl předchůdce $r1$, protože
  - *zařízení* $r13$ *bylo zapnuto* nebo
  - adresa vstupního rozhraní zařízení $r13$ se změnila do rozsahu adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r13$ se změnila do rozsahu adres místní podsítě nebo
  - zařízení $r13$ přibyla brána.

- Zařízení $w14$ se změnil stav z nedosažitelné na zapnuté a přibyl předchůdce $r13$, protože
  - *zařízení* $r13$ *bylo zapnuto* nebo
  - adresa vstupního rozhraní zařízení $r13$ se změnila do rozsahu adres místní podsítě nebo

  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r13$ se změnila do rozsahu adres místní podsítě nebo
  - zařízení $r13$ přibyla brána nebo
  - zařízení $w14$ bylo připojeno vstupním rozhraním nebo
  - zařízení $w14$ přibyla brána.

- Zařízení $r15$ se změnil stav ze zapnuté na vypnuté a ubyl předchůdce $r1$, protože
  - *zařízení* $r15$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r10$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r15$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r15$ ubyla brána.

- Zařízení $r16$ se změnil stav ze zapnuté na vypnuté a ubyl předchůdce $r1$, protože
  - *zařízení* $r16$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r16$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r16$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r16$ ubyla brána.

- Zařízení $w17$ se změnil stav ze zapnuté na nedosažitelné a ubyl předchůdce $r16$, protože
  - *zařízení* $r16$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r16$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r16$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r16$ ubyla brána nebo
  - zařízení $w17$ bylo odpojeno vstupním rozhraním nebo
  - zařízení $w17$ ubyla brána.

Po třetím kroku došlo k odpovídajícím změnám v souboru `input/test.cfg` a přibyl nový soubor `output/100326024834.xml`, který říká, že:

- Zařízení $r15$ se změnil stav z vypnuté na zapnuté a přibyl předchůdce $r1$, protože
  - zařízení $r15$ bylo zapnuto nebo
  - *adresa vstupního rozhraní zařízení* $r15$ *se změnila do rozsahu adres místní podsítě* nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r15$ se změnila do rozsahu adres místní podsítě nebo

  - zařízení $r15$ přibyla brána.

- Zařízení $r16$ se změnil stav z vypnuté na zapnuté a přibyl předchůdce $r1$, protože
  - zařízení $r16$ bylo zapnuto nebo
  - *adresa vstupního rozhraní zařízení* $r16$ *se změnila do rozsahu adres místní podsítě* nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r16$ se změnila do rozsahu adres místní podsítě nebo
  - zařízení $r16$ přibyla brána.

- Zařízení $w17$ se změnil stav z nedosažitelné na zapnuté a přibyl předchůdce $r16$, protože
  - zařízení $r16$ bylo zapnuto nebo
  - *adresa vstupního rozhraní zařízení* $r16$ *se změnila do rozsahu adres místní podsítě* nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r16$ se změnila do rozsahu adres místní podsítě nebo
  - zařízení $r16$ přibyla brána nebo
  - *zařízení* $w17$ *bylo připojeno vstupním rozhraním* nebo
  - zařízení $w17$ přibyla brána.

- Zařízení $r18$ se změnil stav ze zapnuté na vypnuté a ubyl předchůdce $r1$, protože
  - *zařízení* $r18$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r18$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r18$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r18$ ubyla brána.

- Zařízení $w20$ se změnil stav ze zapnuté na nedosažitelné a ubyl předchůdce $r18$, protože
  - *zařízení* $r18$ *bylo vypnuto* nebo
  - adresa vstupního rozhraní zařízení $r18$ se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího zařízení $r18$ se změnila mimo rozsah adres místní podsítě nebo
  - zařízení $r18$ ubyla brána nebo
  - zařízení $w20$ bylo odpojeno vstupním rozhraním nebo
  - zařízení $w20$ ubyla brána.

- Zařízení $r21$ se změnila adresa vstupního rozhraní z 21 na 211, ubyl předchůdce $r18$ a přibyl předchůdce $r19$, protože

- *zařízení* $r18$ *bylo vypnuto* nebo
- adresa vstupního rozhraní zařízení $r18$ se změnila mimo rozsah adres místní podsítě nebo
- adresa výstupního rozhraní zařízení předcházejícího zařízení $r18$ se změnila mimo rozsah adres místní podsítě nebo
- zařízení $r18$ ubyla brána nebo
- *zařízení* $r21$ *ubyla brána* nebo
- *zařízení* $r21$ *přibyla brána.*

6.2 Reálná data

Jelikož autor měl k dispozici výstup z monitorovacího systému, který sleduje skutečnou rozsáhlou počítačovou síť, bylo možné otestovat program i na skutečných datech. Jelikož však tato data odpovídají pouze jednomu okamžiku a není k dispozici stav sítě po provedených změnách, je výsledek velmi nepřesný.

Před spuštěním programu je třeba nastavit počáteční stav topologie.

- Je zkopírován soubor `nagios.log` - ten odpovídá událostem monitorovacího systému v tento okamžik.
- Je vytvořen prázdný adresář `request` - ten odpovídá nedostupnému výsledku programu na zjišťování topologie sítě.
- Je vytvořen prázdný adresář `output` určený na výstup programu.
- Do adresáře `input` jsou umístěny *.cfg soubory - ty odpovídají konfiguraci monitorovacího systému na počátku.

Program je spuštěn pomocí následujícího příkazu:

```
topologywoo\__main__.py --loadnagiostopology input \
  --requestfaketopologycommand --requestnagiostopologyoutput request \
  --savexmltopologychanges output --savenagiostopology input \
  --watchnagiostopology nagios.log
```

Ten odpovídá požadavku na monitorování změn topologie sítě s využitím monitorovacího systému Nagios, který jednak poskytuje počáteční stav topologie (adresář `input`), jednak generuje události (soubor `nagios.log`) a jednak do něho jsou zpětně ukládány změny v topologii (také adresář `input`). Další použité parametry příkazového řádku říkají, že má být použit falešný program na detekci topologie sítě (jehož výstup bude v adresáři `request` ve formátu souborů systému Nagios) a že se změny v topologii mají ukládat ve vlastním formátu definovaném pomocí jazyka XML (adresář `output`).

6.2.1 Výsledky

Přibyl nový soubor `100326124525.xml` v adresáři `output` se změnami, který říká, že:

- 185 zařízení ubyl předchůdce, protože
  - zařízením ubyla brána.
- 2 zařízením se změnil stav ze zapnuté na vypnuté a ubyl předchůdce, protože
  - zařízení byla vypnuta nebo
  - adresa vstupního rozhraní zařízení se změnila mimo rozsah adres místní podsítě nebo
  - adresa výstupního rozhraní zařízení předcházejícího danému zařízení se změnila mimo rozsah adres místní podsítě nebo
  - zařízením ubyla brána.

Kapitola 7

Závěr

Prvním cílem této práce bylo prozkoumat protokoly 2. a vyšší vrstvy ISO/OSI modelu, které jsou použitelné pro rekonstrukci topologie sítí. Ukázalo se, že síťové technologie používané v současnosti poskytují velké množství protokolů, které se dají použít pro detekci topologie. Tyto protokoly by se daly zařadit do skupin: *protokoly, které slouží k nalezení sousedních zařízení*, *protokoly umožňující správu sítě* a *směrovací protokoly*. Za tímto účelem se dají použít i některé další běžně používané protokoly, u nichž je tato schopnost spíše *vedlejším efektem*. Jakékoliv z těchto protokolů je pak potřeba buď *použít* (imitovat zařízení, které běžně tento protokol používá, a vysílat obdobné požadavky) nebo komunikaci pomocí těchto protokolů *odposlouchávat*. V obou případech pak mohou nastat problémy s tím, že některé protokoly používají mechanizmy pro šifrování, autorizaci a další způsoby zabezpečení. Další překážkou může být skutečnost, že některé protokoly jsou proprietární. Nejpodstatnější překážkou však zůstává to, že zdaleka *ne všechny protokoly jsou implementovány ve všech zařízeních*.

Druhým cílem práce bylo navrhnout metodu, která bude automaticky poznávat změny v topologii sítě s ohledem na monitorované prvky a která tyto změny bude vhodně aplikovat do konfigurace monitorovacího systému. Po diskuzi několika možných řešení byla navržena metoda, která *sleduje události* generované monitorovacím systémem, na něž *reaguje detekcí* aktuálního stavu topologie. Následně jsou předchozí a aktuální stav topologie (včetně informací z událostí) porovnány, čímž se získá seznam změn v topologii sítě. Na tyto změny jsou aplikována uvedená pravidla, která slouží k *diagnostice příčin* těchto změn pouze na základě již získaných informací. Seznam změn a jejich příčin následně poslouží volitelné množině parserů, které tyto informace zaznamenají. *Záznam* lze provést jak do konfiguračních souborů volitelných monitorovacích systému, tak do souborů určených k budoucímu zpracovaní uživatelem nebo jinou aplikací. Parsery slouží nejen k zápisu informací, ale zejména obrácením jejich funkce k získávání informací. Tímto způsobem je řešena inicializace metody. Aktuální stav topologie lze získávat buď vlastní metodou, nebo je vhodné využít externí program, jehož výstup je také zpracován odpovídajícím parserem.

Třetím cílem bylo navrženou metodu implementovat do některého open-source monitorovacího systému. Metoda byla implementována v jazyce *Python* pro systém *Nagios*, jeden z nejpoužívanějších monitorovacích systému. Při implementaci byl kladen důraz na možnost

program libovolně konfigurovat a v budoucnu rozšířit nebo znovu-použít[1]. Z toho důvodu byl rozčleněn na čtyři základní druhy komponent lišící se svou logickou funkcí. To umožnilo do programu vkládat *libovolné množství parserů* (držených v jednom druhu komponent), jež může uživatel v budoucnu - až bude implementováno více parserů - *libovolně kombinovat*. Program obsahuje dva parsery pro práci se dvěma různými formáty souborů systému Nagios. Třetí obsažený parser pracuje s novým speciálním formátem souboru (využívající syntaxi jazyka XML) nesoucím informace o změnách a jejich příčinách. Pro detekci aktuálního stavu topologie sítě se předpokládá použití *externího programu*, místo něhož však byl implementován způsob, jak předstírat existenci tohoto programu. Po diskuzi možností, jak sledovat události ukládané do souboru, byl zvolen způsob, který periodicky kontroluje změny v souboru od poslední pozice kurzoru. Z výsledků testování vychází, že návrh i implementace metody jsou *správné*.

Díky rozdělení programu na několik typů komponent a obecné implementaci parseru může být program *dále rozvíjen a doplněn* o podporu dalších monitorovacích systému (resp. formátů jejich souborů). Přidání parseru spočívá v implementaci několika abstraktních metod, které definují použití a funkce jinak obecného parseru, a jeho zpřístupnění pomocí uživatelského rozhraní. Uživatelské rozhraní je realizováno pomocí parametrů příkazového řádku, standardního výstupu (zpravidla konzole) a souborů, což představuje maximální míru *kompatibility* s dalšími aplikacemi.

Skutečnost, že implementace *nepoužívá* žádný existující program na detekci aktuálního stavu topologie sítě, brání nasazení tohoto programu v reálné síti. Výhodou však byla možnost program otestovat na *všech myslitelných situacích*. Slabinou implementace je pak právě tato závislost na externím detekčním programu, který navíc určuje rychlost celé metody detekce změn. Je proto nezbytné, aby detekční program byl dobře optimalizovaný.

Další náměty na zlepšení jsou:

- *Zpřesnění diagnostiky* příčin změn pomocí sledování dalších parametrů zařízení. Některé parametry však nelze sledovat ze vzdáleného počítače a proto by bylo vhodné *implementovat agenta*, který by je sledoval přímo na daném zařízení.
- Pravděpodobně existují lepší způsoby monitorování *změn v souborech* nebo obecné implementace *parseru*. Ty však nejsou autorovi známy.
- Parsery obsažené v programu při zápisu do souborů nevyužívají stejné funkce jako při čtení souborů. Pokud by se podařilo dosáhnout *vyšší míry abstrakce*, vedlo by to k ještě snazší rozšiřitelnosti.

Tato práce autorovi rozšířila znalosti v oblasti *počítačových sítí*, zlepšila schopnost programování v jazyce *Python* a dala zkušenosti s realizací *odborných prací* od získávání teoretických znalostí po implementaci aplikace.

[1] Výraz znovu-použít byl odvozen od slova znovupoužitelnost (anglicky reusability).

Literatura

[1] ATIS committee PRQC. *ATIS Telecom Glossary 2007: Network topology* [online]. 2007. [cit. 21. 11. 2009]. Dostupné z: http://www.atis.org/glossary/definition.aspx?id=3516.

[2] Charles M. Kozierok. *TCP/IP Lower-Layer (Interface, Internet and Transport) Protocols (OSI Layers 2, 3 and 4)* [online]. 2005. [cit. 22. 5. 2010]. Dostupné z: http://www.tcpipguide.com/free/t_TCPIPLowerLayerInterfaceInternetandTransportProtoc.htm.

[3] Charles M. Kozierok. *The Open System Interconnection (OSI) Reference Model* [online]. 2005. [cit. 22. 11. 2009]. Dostupné z: http://www.tcpipguide.com/free/t_TheOpenSystemInterconnectionOSIReferenceModel.htm.

[4] ISO. Publicly Available Standards. http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html.

[5] ITU-T. X-Series Recommendations. http://www.itu.int/rec/T-REC-X/en/.

[6] Jiří Peterka. *Co je čím ... v počítačových sítích* [online]. [cit. 21. 5. 2010]. Dostupné z: http://www.earchiv.cz/i_coje.php3.

[7] Jiří Peterka. *Interoperabilita* [online]. [cit. 22. 5. 2010]. Dostupné z: http://www.earchiv.cz/i_inter.php3.

[8] Jiří Peterka. *Principy počítačových sítí* [online]. [cit. 23. 11. 2009]. Dostupné z: http://www.earchiv.cz/i_pri.php3.

[9] Jiří Peterka. *Referenční model ISO/OSI* [online]. [cit. 23. 11. 2009]. Dostupné z: http://www.earchiv.cz/anovinky/ai1552.php3.

[10] Jiří Peterka. *Počítačové sítě, verze 3.1* [online]. [cit. 23. 11. 2009]. Dostupné z: http://www.earchiv.cz/l214/index.php3.

[11] Learn-Networking.com. *TCP/IP* [online]. 2007. [cit. 3. 1. 2010]. Dostupné z: http://learn-networking.com/category/tcp-ip.

[12] Learn-Networking.com. *A Guide to Network Topology* [online]. 2007. [cit. 21. 11. 2009]. Dostupné z: http://learn-networking.com/network-design/a-guide-to-network-topology.

[13] Nagios Enterprises. Nagios Core 3.x Documentation. http://nagios.sourceforge.net/docs/3_0/toc.html.

[14] Petr Grygárek. *Autonomní systémy a směrování mezi nimi. Border Gateway Protocol (BGP). Externí a interní BGP. Spolupráce BGP s protokoly třídy IGP.* [online]. [cit. 22. 5. 2010]. Dostupné z: http://www.cs.vsb.cz/grygarek/SPS/lect/BGP/BGP.html.

[15] Python Software Foundation. Download Python, . http://python.org/download/.

[16] Python Software Foundation. Python Documentation, . http://docs.python.org/.

[17] Přispěvatelé Wikipedie. *Aplikační vrstva* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Aplika%C4%8Dn%C3%AD_vrstva&oldid=4413504.

[18] Přispěvatelé Wikipedie. *Address Resolution Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Address_Resolution_Protocol&oldid=5236175.

[19] Přispěvatelé Wikipedie. *Border Gateway Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Border_Gateway_Protocol&oldid=5283731.

[20] Přispěvatelé Wikipedie. *Sběrnicová topologie* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Sb%C4%9Brnicov%C3%A1_topologie&oldid=3665185.

[21] Přispěvatelé Wikipedie. *Cisco Discovery Protocol* [online]. 2009. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Cisco_Discovery_Protocol&oldid=4375468.

[22] Přispěvatelé Wikipedie. *Prohledávání do hloubky* [online]. 2010. [cit. 20. 3. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Prohled%C3%A1v%C3%A1n%C3%AD_do_hloubky&oldid=5113346.

[23] Přispěvatelé Wikipedie. *Hvězdicová topologie* [online]. 2009. [cit. 22. 12. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Fyzick%C3%A1_vrstva&oldid=4617961.

[24] Přispěvatelé Wikipedie. *Hvězdicová topologie* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Hv%C4%9Bzdicov%C3%A1_topologie&oldid=4524177.

[25] Přispěvatelé Wikipedie. *ICMP* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=ICMP&oldid=5243370.

[26] Přispěvatelé Wikipedie. *Referenční model ISO/OSI* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Referen%C4%8Dn%C3%AD_model_ISO/OSI&oldid=4505714.

[27] Přispěvatelé Wikipedie. *Kruhová topologie* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Kruhov%C3%A1_topologie&oldid=4504005.

[28] Přispěvatelé Wikipedie. *Linková vrstva* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Linkov%C3%A1_vrstva&oldid=4408423.

[29] Přispěvatelé Wikipedie. *Open Shortest Path First* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Open_Shortest_Path_First&oldid=5310016.

[30] Přispěvatelé Wikipedie. *Proxy ARP* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Proxy_ARP&oldid=5138449.

[31] Přispěvatelé Wikipedie. *Prezentační vrstva* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Prezenta%C4%8Dn%C3%AD_vrstva&oldid=4382914.

[32] Přispěvatelé Wikipedie. *Relační vrstva* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Rela%C4%8Dn%C3%AD_vrstva%&oldid=4382904.

[33] Přispěvatelé Wikipedie. *Routing Information Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Routing_Information_Protocol%&oldid=5352272.

[34] Přispěvatelé Wikipedie. *Síťová vrstva* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=S%C3%AD%C5%A5ov%C3%A1_vrstva&oldid=4408408.

[35] Přispěvatelé Wikipedie. *Simple Network Management Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Simple_Network_Management_Protocol&oldid=5245969.

[36] Přispěvatelé Wikipedie. *Stromová topologie* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Stromov%C3%A1_topologie&oldid=3665190.

[37] Přispěvatelé Wikipedie. *TCP/IP* [online]. 2009. [cit. 3. 1. 2010]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=TCP/IP&oldid=4750149.

[38] Přispěvatelé Wikipedie. *Topologie sítí* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Topologie_s%C3%ADt%C3%AD%&oldid=4532121.

[39] Přispěvatelé Wikipedie. *Transportní vrstva* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Transportn%C3%AD_vrstva&oldid=4009544.

[40] Wikipedia contributors. *Enhanced Interior Gateway Routing Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Enhanced_Interior_Gateway_Routing_Protocol&oldid=363181042.

[41] Wikipedia contributors. *Application Layer* [online]. 2009. [cit. 3. 1. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Application_Layer&oldid=332459220.

[42] Wikipedia contributors. *Address Resolution Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Address_Resolution_Protocol&oldid=363327800.

[43] Wikipedia contributors. *Border Gateway Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Border_Gateway_Protocol&oldid=361288589.

[44] Wikipedia contributors. *Bus network* [online]. 2009. [cit. 21. 11. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Bus_network&oldid=323902637.

[45] Wikipedia contributors. *Cisco Discovery Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Cisco_Discovery_Protocol&oldid=340280388.

[46] Wikipedia contributors. *Common management information protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Common_management_information_protocol&oldid=340837615.

[47] Wikipedia contributors. *Depth-first search* [online]. 2010. [cit. 20. 3. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Depth-first_search&oldid=351033904.

[48] Wikipedia contributors. *Exterior Gateway Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Exterior_Gateway_Protocol&oldid=340569811.

[49] Wikipedia contributors. *Gateway-to-Gateway Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Gateway-to-Gateway_Protocol&oldid=363319664.

[50] Wikipedia contributors. *Interior Gateway Routing Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Interior_Gateway_Routing_Protocol&oldid=347357357.

[51] Wikipedia contributors. *Internet Control Message Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Internet_Control_Message_Protocol&oldid=357177625.

[52] Wikipedia contributors. *Internet Layer* [online]. 2009. [cit. 3. 1. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Internet_Layer&oldid=332868421.

[53] Wikipedia contributors. *Internet Protocol Suite* [online]. 2009. [cit. 3. 1. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Internet_Protocol_Suite&oldid=334190717.

[54] Wikipedia contributors. *ICMP Router Discovery Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=ICMP_Router_Discovery_Protocol&oldid=353481603.

[55] Wikipedia contributors. *IS-IS* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=IS-IS&oldid=360777572.

[56] Wikipedia contributors. *OSI model* [online]. 2010. [cit. 3. 1. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=OSI_model&oldid=335316800.

[57] Wikipedia contributors. *Data Link Layer* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Data_Link_Layer&oldid=324913145.

[58] Wikipedia contributors. *Link Layer Discovery Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Link_Layer_Discovery_Protocol&oldid=357276259.

[59] Wikipedia contributors. *Link Layer Topology Discovery* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Link_Layer_Topology_Discovery&oldid=350893755.

[60] Wikipedia contributors. *Link Layer* [online]. 2009. [cit. 3. 1. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Link_Layer&oldid=317293487.

[61] Wikipedia contributors. *Mesh networking* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Mesh_networking&oldid=319052147.

[62] Wikipedia contributors. *Neighbor Discovery Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Neighbor_Discovery_Protocol&oldid=362762023.

[63] Wikipedia contributors. *Nortel Discovery Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Nortel_Discovery_Protocol&oldid=351704125.

[64] Wikipedia contributors. *Network Layer* [online]. 2009. [cit. 21. 10. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Network_Layer&oldid=321225387.

[65] Wikipedia contributors. *Open Shortest Path First* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Open_Shortest_Path_First&oldid=363316741.

[66] Wikipedia contributors. *Proxy ARP* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Proxy_ARP&oldid=351853955.

[67] Wikipedia contributors. *Physical Layer* [online]. 209. [cit. 22. 11. 209]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Physical_Layer&oldid=326522880.

[68] Wikipedia contributors. *Presentation Layer* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Presentation_Layer&oldid=327226144.

[69] Wikipedia contributors. *Ring network* [online]. 2009. [cit. 21. 11. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Ring_network&oldid=324859231.

[70] Wikipedia contributors. *Routing Information Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Routing_Information_Protocol%&oldid=362824382.

[71] Wikipedia contributors. *Session Layer* [online]. 2009. [cit. 22. 11. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Session_Layer&oldid=331737134.

[72] Wikipedia contributors. *Simple Gateway Monitoring Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Simple_Gateway_Monitoring_Protocol&oldid=357260841.

[73] Wikipedia contributors. *Secure Neighbor Discovery Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Secure_Neighbor_Discovery_Protocol&oldid=362832893.

[74] Wikipedia contributors. *Simple Network Management Protocol* [online]. 2010. [cit. 22. 5. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Simple_Network_Management_Protocol&oldid=362577442.

[75] Wikipedia contributors. *Star network* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Star_network&oldid=318669927.

[76] Wikipedia contributors. *TCP/IP model* [online]. 2009. [cit. 3. 1. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=TCP/IP_model&oldid=331893915.

[77] Wikipedia contributors. *Network topology* [online]. 2009. [cit. 23. 10. 2009]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Network_topology&oldid=320929653.

[78] Wikipedia contributors. *Transport Layer* [online]. 2010. [cit. 3. 1. 2010]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Transport_Layer&oldid=335654753.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

ARP Address Resolution Protocol

BDP Bay Discovery Protocol

BGP Border Gateway Protocol

BNMP Bay Network Management Protocol

CCITT International Telegraph and Telephone Consultative Committee

CDP Cisco Discovery Protocol

CMIP Common Management Information Protocol

CMOT Common Management Interface Protocol Over Tcpip

DARPA Defense Advanced Research Projects Agency

DFS Depth-first search

DTD Document Type Definition

EGP Exterior Gateway Protocol

EIGRP Enhanced Interior Gateway Routing Protocol

GGP Gateway-to-Gateway Protocol

ICMP Internet Control Message Protocol

IGRP Interior Gateway Routing Protocol

IP Internet Protocol

IRDP ICMP Router Discovery Protocol

IS-IS Protokol Intermediate system to intermediate system

ISO International Organization for Standardization

ITU-T International Telecommunication Union Telecommunication Standardization Sector

LLDP Link Layer Discovery Protocol

LLTD Link Layer Topology Discovery

MIB Management Information Base

NDP Neighbor Discovery Protocol

NMM Nortel Management MIB

NTDP Nortel Topology Discovery Protocol

OS Operating system

OSI Open Systems Interconnection

OSPF Open Shortest Path First

RIP Routing Information Protocol

SEND Secure Neighbor Discovery Protocol

SGMP Simple Gateway Monitoring Protocol

SNMP Simple Network Management Protocol

SONMP SynOptics Network Management Protocol

TCP Transmission Control Protocol

TTL Time to live

Příloha B

Česko anglický slovníček

Aplikační vrstva Application layer

Autonomní systém Autonomous system

Částečná ... topologie Partially connected ... topology

Datagram Datagram

Distribuovaná ... topologie Distributed ... topology

Dvoubodová topologie Point-to-point topology

Externí brána Exterior gateway

Fyzická adresa Physical address

Fyzická topologie Physical topology

Fyzická vrstva Physical layer

Hierarchická ... topologie Hierarchical ... topology

Hlavičky Headers

Hlavní brána Core gateway

Hvězdicová topologie Star topology

Kruhová topologie Ring topology

Lineární ... topologie Linear ... topology

Logická (síťová) adresa Logical (Network Layer) address

Logická topologie Logical topology

Nespojovaná síť Connectionless network

Nespolehlivá služba Unreliable service

Paket Packet

Páteř Backbone

Potvrzení Acknowledgment

Prezentační vrstva Presentation layer

Prohledávání do hloubky Depth-first search

Protokol Protocol

Přepínaná ... topologie Switched ... topology

Rámce Frames

Relační vrstva Session layer

Rozhraní Interface

Rozšířená ... topologie Extended ... topology

Řízení přístupu k médiu Media Access Control

Řízení toku Flow control

Sběrnicová topologie Bus topology

Segment Segment

Signálová topologie Signal topology

Síťová (smyčková) topologie Mesh topology

Síťová vrstva (TCP/IP) Internet layer

Síťová vrstva (ISO/OSI) Network layer

Směrovací protokol Routing protocol

Směrovací tabulka Routing table

Směrovač Router

Spojovaná síť Connection oriented network

Spojová (linková) vrstva Data Link layer

Spolehlivá služba Reliable service

Systém pro monitorování sítě Network monitoring system

Tepna Trunk

Topologie sítě Network topology

Transportní vrstva Transport layer

Trvalá ... topologie Permanent ... topology, dedicated ... topology

Úplná ... topologie Fully connected ... topology

Vrstva Layer

Vrstva síťového rozhraní Link layer

Zapouzdřování Encapsulation

Příloha C

Instalační a uživatelská příručka

C.1 Účel programu

Program slouží k automatické detekci změn v topologii počítačové sítě. Pro úplnou funkčnost programu jsou vyžadovány programy třetích stran a jejich vstupy a výstupy:

- Výstup programu, který získá počáteční stav topologie sítě.
- Program, který získá aktuální stav topologie sítě, a jeho výstup.
- Vstup programu, který vyžaduje informace o změnách v topologii sítě.
- Výstup programu, který monitoruje počítačovou síť.

V aktuální verzi program podporuje vstup a výstup monitorovacího systému Nagios. Rovněž poskytuje výstup v XML souborech. Informace o aktuálním stavu topologie je třeba nastavovat manuálně.

C.2 Instalace

Program TopologyWoo je multiplatformní (běží na většině operačních systémů – podpora systému se odvíjí od dostupnosti interpretu jazyka Python pro tento systém) a přenosný (z anglického portable), tj. nevyžaduje instalaci a nezanechává po sobě nevyžádané stopy.

Instalace spočívá ve zkopírování adresáře `topologywoo` (uloženého na přiloženém CD v adresáři `src`) do libovolného adresáře. Dále je nutné nainstalovat interpret jazyka Python [15].

C.3 Spuštění

Uživatelské rozhraní je realizováno pomocí parametrů příkazového řádku, standardního výstupu (zpravidla konzole) a souborů.

Spuštěním programu s parametrem `-h` je vypsána následující nápověda:

```
Usage: __main__.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit

  Input Options:
    --loadnagiostopology=DIR
                        load topology from Nagios configuration directory DIR
                        (example: "path/to/Nagios/config")

  Requesting Options:
    --requestfaketopologycommand
                        fake command executed to request topology
    --requestnagiostopologyoutput=DIR
                        Nagios configuration directory DIR containing output
                        of request topology command (example:
                        "path/to/Nagios/config")

  Output Options:
    --savexmltopologychanges=DIR
                        save topology changes to XML files in directory DIR
                        (example: "path/to/changes/XML")
    --savenagiostopology=DIR
                        save topology to Nagios configuration directory DIR
                        (example: "path/to/Nagios/config")

  Watching Options:
    --watchingrate=SEC  set watching refresh rate SEC in seconds (default:
                        60.0)
    --watchnagiostopology=FILE
                        watch topology via Nagios log file FILE (example:
                        "path/to/Nagios/log.ext")
```

Ukázka 9: Nápověda programu.

Na pořadí parametrů záleží. Díky množství parametrů lze dosáhnout velké variability programu. Parametr `--loadnagiostopology` definuje adresář, v němž se nachází počáteční stav topologie ve formátu konfiguračních souborů programu Nagios. Parametr `--requestfaketopologycommand` definuje, že jako program na získávání aktuální topologie sítě má být použít falešný (nic neprovádějící) příkaz. Parametr `--requestnagiostopologyoutput` definuje adresář, v němž se nachází výstup programu zjišťujícího aktuální stav topologie v souborech formátu konfiguračních souborů programu Nagios. Parametr `--savexmltopologychanges` definuje adresář, do něhož se budou ukládat XML soubory se změnami v topologii. Parametr `--savenagiostopology` definuje adresář s konfiguračními soubory programu Nagios, do něhož se budou ukládat aktuální stavy topologie. Parametr `--watchingrate` definuje v sekundách periodu kontroly změn v topologii sítě. Parametr `--watchnagiostopology` definuje log soubor programu Nagios, který má být sledován za účelem získávání změn v topologii.

Typický příkaz pro spuštění programu z adresáře nacházejícího se o úroveň výš vypadá takto:

```
topologywoo/__main__.py --loadnagiostopology "path/to/Nagios/config" \
  --requestfaketopologycommand --requestnagiostopologyoutput \
  "path/to/command/nagios" --savexmltopologychanges "path/to/changes/XML" \
  --savenagiostopology "path/to/Nagios/config" --watchingrate 45
  --watchnagiostopology "path/to/Nagios/log.ext"
```

C.4 Ukončení

Program se ukončuje signálem *SIGINT*, který se na většině systémů vyvolává stiskem kombinace kláves `ctrl` a `c`.

Příloha D

Obsah přiloženého CD

```
\- src                      adresář se zdrojovými kódy
|   \- topologywoo          adresář se zdrojovými kódy programu
|       \- __init__.py      pomocný soubor jazyka Python
|       \- __main__.py      zdrojový kód programu
\- text                     adresář s textem práce
|   \- bp.pdf               text bakalářské práce
\- readme.txt               uživatelská příručka
```