VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

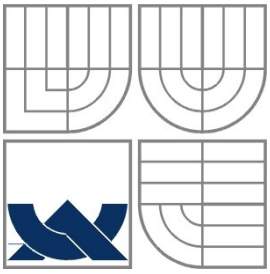
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SIMULACE CHOVÁNÍ SÍTĚ NA ZÁKLADĚ ANALÝZY KONFIGURAČNÍCH SOUBORŮ AKTIVNÍCH SÍŤOVÝCH ZAŘÍZENÍ

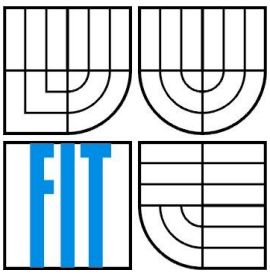
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JIŘÍ MACKŮ
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SIMULACE CHOVÁNÍ SÍTĚ NA ZÁKLADĚ ANALÝZY KONFIGURAČNÍCH SOUBORŮ AKTIVNÍCH SÍŤOVÝCH ZAŘÍZENÍ

SIMULATION OF NETWORK BEHAVIOUR BASED ON ANALYSIS OF CONFIGURATION OF ACTIVE NETWORK DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ MACKŮ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MATOUŠEK, Ph.D.

BRNO 2008

Zadání diplomové práce/5947/2007/xmacku06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání diplomové práce

Řešitel: **Macků Jiří, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Simulace chování sítě na základě analýzy konfiguračních souborů aktivních síťových zařízení**
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s možnostmi diskrétní simulace počítačových sítí pomocí nástroje ns2.
2. Proveďte analýzu konfiguračních souborů vybraných Cisco zařízení (přepínače, směrovače). Definujte data, která jsou důležitá pro simulační model. Inspirujte se nástrojem nipper.
3. Definujte způsob překladu vybraných konfiguračních souborů do simulačního jazyka nástroje ns2.
4. Vytvořte nástroj na automatický překlad vybraných informací z konfiguračních souborů do nástroje ns2.
5. Na příkladu počítačové sítě ukažte použitelnost vytvořeného nástroje. Proveďte simulaci dané sítě. Zobrazte výsledky simulace v nástroji vhodném pro analýzu chování sítě.
6. Porovnejte výsledky simulace s reálnou komunikací otestovanou v školní síťové laboratoři.
7. Zhodnoťte použitelnost uvedených prostředků pro různé třídy síťové komunikace.

Literatura:

- Manuály k systému NS dostupné na http://www.isi.edu/nsnam
- nipper - network infrastructure parser, dokumentace na http://sourceforge.net/projects/nipper

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1-4.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVR-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Matoušek Petr, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 24. září 2007
Datum odevzdání: 19. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Jiří Macků**
Id studenta: 89491
Bytem: Líšeňská 5, 636 00 Brno
Narozen: 09. 12. 1983, Brno
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Simulace chování sítě na základě analýzy konfiguračních souborů aktivních síťových zařízení
Vedoucí/školitel VŠKP: Matoušek Petr, Ing., Ph.D.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

V diplomové práci se zabýváme simulací sítě v nástroji Network Simulator. Model sítě i popis simulace vytváříme na základě analýzy konfiguračních souborů a jejich následného překladu vytvořeným převodním nástrojem, jehož návrh i implementace je zde popsána. Pro Network Simulator vytváříme IPv4 adresaci, která není v simulátoru podporována. Network Simulator je rozšířen o filtrování paketů pomocí seznamů ACL. Praktické využití implementovaných součástí je ukázáno na simulaci reálné sítě, kde se také zabýváme jejich ekvivalentním chováním v porovnání se skutečným prostředím.

Klíčová slova

Analýza konfiguračního souboru, aktivní síťové zařízení, ns-2, diskrétní simulace, simulace sítě, cisco, IPv4 adresace, access control list.

Abstract

This master's thesis describes simulation of network using Network Simulator. Model of network and description of simulation is extracted from the analysis of configuration files of active network devices, and translated by a parser. Concept and implementation of the parser is described here. Because IPv4 addressing is not supported in Network Simulator, it was added as a new module. The Network Simulator is also extended by filtering properties of packets using access control lists. Practical usage of the implemented modules is demonstrated on a simulation of real network.

Keywords

Analysis of configuration, active network device, ns-2, discrete simulation, network simulation, cisco, IPv4 addressing, access control list.

Citace

Macků Jiří: Simulace chování sítě na základě analýzy konfiguračních souborů aktivních síťových zařízení. Brno, 2008, diplomová práce, FIT VUT v Brně.

Simulace chování sítě na základě analýzy konfiguračních souborů aktivních síťových zařízení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením
Ing. Petra Matouška, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Macků
14.5.2008

Poděkování

Děkuji vedoucímu mé diplomové práce Ing. Petru Matouškovi, Ph.D., za jeho odborné rady a veškerou pomoc při vytváření diplomové práce a za permanentní časovou dostupnost k řešení průběžných problémů.

© Jiří Macků, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

Při návrhu, rozšiřování, zavádění nových služeb nebo i zjišťování současného stavu datových sítí se musíme stále zabývat otázkou, zdali je nebo bude datová síť schopna poskytovat potřebné parametry, jimiž jsou především rychlost odezvy, datová propustnost, stabilita a bezpečnost v závislosti na počtu uživatelů a služeb, které síť využívají.

Při rozšiřování stávajících sítí a v nich implementací nových služeb nebo zabezpečovacích politik není často možné podrobit síť testům, protože by se narušil její provoz. Vzhledem k již nedostupným starším zařízením a nemožnosti „držet skladem“ všechny typy aktivních prvků nejsme schopni síť postavit na jiném místě a na ní provádět potřebné zkoušky. Často jsme pouze odkázáni na výrobcem udávané parametry.

V případě návrhu nových sítí máme většinou k dispozici více variant, jak novou topologii realizovat. Zde se setkáváme především s finančními problémy, kdy je neekonomické koupit všechny možnosti realizace a na základě testů vybrat tu nejvhodnější.

Pokud vezmeme v úvahu i rozlehlost sítí a jedinečnost jednotlivých spojů (např. Wi-fi a prostředí, kterým se šíří signál), nejsme schopni ani síť dopředu otestovat a často se toto děje až přímo na místě instalace.

Vlivem uvedených nedostatků je snaha používat simulační nástroje, které by na základě dostupných informací byly schopny vytvořit topologii sítě, a po ručním vložení nebo automatickém získání parametrů pro jednotlivé uzly a spoje mezi nimi zprostředkovat simulaci chování sítě. Tento přístup je především velice laciný, rovněž umožňuje vlivem grafických aplikací sledovat více aspektů sítě zároveň, také jsme schopni podrobit síť daleko většímu a rozmanitějšímu počtu testů i použít verifikační nástroje. Na základě těchto možností můžeme vytvořit stabilnější, výkonnější síť a lépe ji optimalizovat do budoucnosti.

1.1 Cíl práce

V diplomové práci se zabývám dvěma problematikami. První je vytvoření simulačního modelu sítě na základě analýzy a následného překladu konfiguračních souborů aktivních prvků pro nástroj Network Simulator – ns-2 (dále jen NS), který svým chováním odpovídá realitě. Vlivem různého stylu a rozmanitosti ukládaných informací a různých struktur v konfiguračních souborech aktivních prvků odlišných výrobců uvažuji pouze použití Cisco aktivních prvků.

Druhou problematikou je implementace Cisco access control listů do modelu sítě a vytvoření IPv4 adresace uzlů pro NS.

Uvedené dva směry zakomponovávám do simulace, která ukazuje ekvivalenci vytvořeného modelu v rozsahu nově implementovaných modulů a funkcí se skutečným prostředím.

1.2 Od konfiguračních souborů až k simulaci

Po seznámení se s nástrojem nipper, který pro svou práci také potřebuje transformaci konfiguračních souborů, jsem se rozhodl použít pro překlad na simulaci v NS PHP.

Prvním krokem k vytvoření modelu je získání konfiguračních souborů buď přímo ze zařízení, nebo exportem do souboru.

V dalším kroku jsou analyzovány a získány potřebné parametry z konfiguračních souborů (např. nastavení interface, parametry linek, routovaní, access control list) převodním nástrojem. Na základě zjištěných údajů se vytvoří topologie sítě, přiřadí se parametry jednotlivým linkám (např. zpoždění, šířka pásma), vytvoří se routování a definují se access control listy. Výstupem převodu je soubor v jazyce OTcl, který celkově popisuje model sítě i samotnou simulaci.

NS umí identifikovat jednotlivé uzly pouze podle jejich jmen, což je problém, se kterým se musí překlad zejména při vytváření topologie nebo routování vyrovnat.

Samotná simulace pak probíhá tak, že na vstupu převodního nástroje jsou k dispozici konfigurační soubory aktivních prvků tvořících simulovanou topologii a je zadán výstupní *.tcl soubor. Vše se odehrává na příkazové řádce. Po vygenerování simulace ji můžeme ihned spustit. Protože v této práci budu prezentovat jednoduché topologie sítí, je do simulace zahrnut i popis pro grafické rozhraní NAM, které po dokončení výpočtu celou simulaci graficky zobrazí.

1.3 Použité prostředky

Vytváření modelu, samotnou simulaci i rozšiřování NS jsem prováděl na operačním systému Windows XP SP2 s použitím Cygwin s nainstalovaným balíkem ns-allinone-2.32. Výsledný produkt je tedy kompatibilní s verzí NS ns-2.32.

Pro samotný překladový nástroj konfiguračních souborů jsem zvolil skriptovací jazyk PHP 4.4.2.

Jako zdroj vstupních dat jsem použil vygenerované konfigurační soubory z programu Cisco Packet Tracer 4.1 na základě v něm vytvořené síti nebo přímo konfigurace získané z reálného zapojení aktivních prvků.

Příkazovou sadu jsem porovnával na směrovači Cisco 2801 (revision 7.0) s verzí IOS 12.4 (13r).

Paralelně s mou diplomovou prací vytvářel svou bakalářskou práci student Jan Šafář, který použil mé IPv4 adresování a rozšířil NS tak, že je schopný adresace paketů na základě IP adres. Dále pak upravil statické směrování pro IPv4 a vytvořil směrování pomocí RIP v1 i v2 protokolu pro použití s Cisco routery. Na jeho práci navazuji v rozšíření převodního nástroje o příkazy z Cisco routerů, které podporují jeho vylepšení v NS. Vzhledem k tomu, že principielní změny Jana Šafáře provedené v NS jsou veliké a není možné použít jeden simulační model pro obě verze, převodní

nástroj umožňuje vytvořit simulační model jak pro oficiální verzi ns-2.32, tak i rozšířenou Janem Šafářem.

1.4 Typografické konvence a zkratky

Za účelem zkrácení textu užívám místo frekventovaných dlouhých názvů zkratky s následujícím významem:

- NS – Network Simulator.
- NSJS – Rozšířená verze Network Simulatoru Janem Šafářem.
- ACL – Access control list na Cisco směrovačích.

Pro zpřehlednění textu a lepší výklad užívám následující typy písem ve významu:

- Základní typ písma – podstatný rozsah textu práce.
- *Kurzíva* – zvýraznění různých fragmentů v textu.
- `Fontem Courier New` a `jeho menší velikostí` – ukázkové příklady zdrojových kódů.

1.5 Struktura práce

V Úvodu práce popisuji důvody použití simulačních technik v oblastech návrhu, rozšiřování, zavádění nových služeb nebo i zjišťování současného stavu datových sítí. Stručně se zmiňuji o cíli této práce a způsobu jeho realizace. Uvádím zde prostředky, které byly použity při vytváření modelu a simulace.

V druhé kapitole zařazuji simulace sítí do obecné problematiky simulace a vytváření modelu simulace a představují některé v současnosti používané simulační nástroje (NS, OPNET Modeler, QualNet, OMNET++).

Ve třetí kapitole podrobněji popisuji nástroj NS. Zaměřuji se na jeho historii a hlavní rysy, probírám vytváření modelu, výstup simulace a v poslední podkapitole uvádím používané grafické nástroje pro zobrazení výsledku simulace.

Ve čtvrté kapitole vysvětluji princip IPv4 adresace implementovaného v NS.

V páté kapitole rozebírám analýzu konfiguračních souborů po stránce jejich struktury i převoditelnosti potřebných informací pro NS i NSJS.

V šesté kapitole podrobně popisuji převod konfiguračních souborů do NS a NSJS. Postupně uvádím jednotlivé fáze převodu v pořadí, ve kterém musí být vykonány: získání potřebných informací z konfiguračních souborů, vytvoření uzlů a linek (topologie), aktivace a konfigurace směrování, definice ACL a vytvoření kontrolní simulace.

V sedmé kapitole ukazuji význam jednotlivých zdrojových souborů převodního nástroje, definuji způsob jeho použití a přehledně vypisuji Cisco příkazy a jejich úroveň podpory převodním nástrojem, NS i NSJS.

V osmé kapitole teoreticky představuji ACL, zařazuji je do světa bezpečnosti sítí, uvádím jejich princip a způsob používání. Dále rozebírám jednotlivé typy ACL.

V deváté kapitole popisuji problémy a nedostatky NS, které musíme vyřešit, než začneme se samotnou implementací ACL. Poté rozebírám zpracování vybraných příkazů, úroveň podpory jednotlivých implementovaných typů ACL a v poslední části ukazuji příklady použití některých příkazů.

V desáté kapitole prakticky ukazuji způsob hledání chyb v konfiguraci ACL s použitím nově implementovaných součástí a dále tuto problematiku obecně rozvádím a rozšiřuji i o kontrolu konfigurace směrování. Vše ukazuji na příkladu a rozebírám možnosti a problémy při zpracování výsledků testů.

V jedenácté kapitole demonstruji použití převodního nástroje a implementovaných součastí na příkladu simulace reálné sítě, kde také ukazují ekvivalenci simulačního modelu se skutečnou sítí.

V Závěru práce hodnotím dosažené výsledky a uvádím další možný vývoj projektu.

V Přílohách prezentuji obsah přiloženého CD, seznam příkazů podporovaných převodním nástrojem s popisem úrovně podpory pro NS a NSJS i s příklady použití, seznam souborů zdrojových kódů NS, kde jsem provedl rozšíření, a nově přidané zdrojové soubory, konfigurační soubory směrovačů a OTcl skripty simulací z příkladů v desáté a jedenácté kapitole.

2 Simulace sítí

V následující podkapitole se seznámíme obecně s pojmem simulace sítí a zařadíme ji do kontextu modelování a simulací. Dále uvedeme nejčastější způsob vytváření modelu sítě, který se svými jednotlivými kroky přibližuje vrstvám ISO/OSI modelu. Poté rozdělíme simulaci podle metody simulace a představíme typy výstupů ze simulace.

Rovněž se seznámíme s některými představiteli simulačních a modelových nástrojů, které se v současnosti využívají.

2.1 Úvod do simulace sítí

Simulace sítí vychází z obecné simulace, kde pracujeme s termíny jako systém, modelování, model, simulace, které mají stejný význam a principy i v simulaci sítí. Setkáváme se zde se stejnými problémy jako např. validita modelu, náročnost na výkon počítačů při simulaci velkých sítí, někdy i vysoké úsilí při vytváření modelů.

Modely sítí jsou diskrétní, událostmi řízené. Některé simulační nástroje podporují real-time simulace.

Postup při vytváření modelu sítě zpravidla probíhá po úrovních abstrakce, kde se více či méně přibližuje k jednotlivým vrstvám ISO/OSI modelu. Každý simulovaný prvek (router, switch, pracovní stanice) je v modelu sítě tvořen uzlem. Fyzickou a linkovou vrstvu pak představuje linka, která propojuje dva uzly. Ve velké většině případů je síťová vrstva řešena na principu IP protokolu. Zde některé simulační jazyky neumějí pracovat přímo s IP adresami, ale pouze s unikátními názvy uzlů. Princip adresace, směrování i směrovacích protokolů je však zachován. Parametry jako např. velikost a typ front, zpoždění, rychlost, vlastnosti protokolů se nastavují na příslušných úrovních abstrakce modelu. Dále se definuje transportní vrstva typicky protokoly TCP a UDP, na které se pak navazují protokoly vyšších vrstev. Některé simulační prostředky, např. NS zbývající tři vrstvy (tři úrovně abstrakce) spojuje v jednu, a to aplikační vrstvu. Pro aplikační vrstvu se pak modelují protokoly jako FTP nebo je možné definovat určité typy datových toků typicky CBR (Constant Bit Rate).

Takto vytvořený model je pak oživen implementací paketu a na základě předdefinovaných parametrů na všech úrovních abstrakce tvoří simulaci a ovlivňuje chování modelu.

Podle schopností modelovacích nástrojů jednotlivých simulačních programů je možné vytvářet model pomocí grafického rozhraní nebo jen skrze textový soubor, který má definovaný syntax (programovací jazyk).

Samotnou simulaci pak můžeme rozdělit podle metody simulace do následujících skupin:

- **Událostmi řízená** – v jádru simulace je vytvořen kalendář, který plánuje provádění akcí na úrovni paketů na základě nadefinovaných vlastností prvků v modelu. Kalendář také

reaguje na předem neznámé události (např. nefunkční linka a reakce směrovacích protokolů). Tato metoda simulace vykazuje přesné výsledky, je však ale pomalejší a náročnější na výpočet.

- **Analytická** – model sítě je popsán matematickou reprezentací. Tento princip je rychlejší než předchozí, ale výsledky simulace nejsou tak podrobné. Navíc tato metoda je vhodná pouze pro menší a zjednodušené modely.
- **Hybridní** – kombinuje předchozí dva přístupy, kde se např. pro jistou oblast použije analytické zpracování a tam, kde je potřeba přesnějších výsledků, je použito událostmi řízené zpracování.

Výstup simulace se zpravidla zaznamenává do souboru s definovanou strukturou. Pomocí tohoto souboru jsou pak k dispozici grafické nástroje, které animují chování sítě nebo zobrazují charakteristiky určité vlastnosti v modelu. Některé simulační nástroje umožňují zobrazení simulace již za jejího chodu.

2.2 Některé současné simulační nástroje

V této části práce uvádíme některé v současnosti používané simulační nástroje a charakterizujeme jejich základní principy a možnosti.

2.2.1 Network Simulator – ns-2

NS je objektově orientovaný, událostmi řízený síťový simulátor (real-time simulace jsou ve stádiu vývoje) napsaný v jazycích C++ a OTcl. C++ tvoří výpočetní jádro simulátoru a OTcl je použit pro popis modelu sítě i simulace. Jeho zdrojové kódy jsou kompletně volně dostupné. Je primárně určen pro modelování IP sítí LAN a WAN drátových i bezdrátových. Podporuje statické a dynamické směrování pro unicast i multicast sítě. NS nemá žádné grafické rozhraní, jeho výstupem je soubor s přesně definovanou strukturou, který dále zpracovávají externí programy. Pro animaci simulace se používá NAM, s nástrojem Huginn je dokonce možné provádět 3D animace simulace, charakteristiky lze zobrazovat v programu Xgraph. Podporované platformy jsou Linux/Unix a Windows (Cygwin). Zdrojové kódy jsou volně dostupné k dalšímu rozšiřování. NS disponuje rozsáhlou dokumentací, která však svým stylem výkladu může zbrzdit pochopení NS i práci v něm nebo dokonce odradit začínajícího uživatele [3].

Protože NS je simulační prostředí použité pro modelování a simulace sítí v této práci, jeho podrobný popis je rozebrán v následující kapitole.

2.2.2 OPNET Modeler

OPNET Modeler je jednou ze součástí OPNET Solutions konkrétně Network R&D. Používá se jak pro vývojové účely, tak také pro návrh a analýzu. Jedná se o hierarchický, objektově orientovaný, diskrétní simulátor, který podporuje událostmi řízené, hybridní nebo analytické simulace. Má v sobě zaimplementovány stovky protokolů a otevřené rozhraní pro integraci externích objektů, knihoven a simulátorů. Podporuje techniku system-in-the-loop a 32bit i 64bit paralelní simulaci.

OPNET Modeler se opírá o grafické prostředí, které je vytvořeno z jazyků C a C++. Samotné vytváření modelu můžeme rozdělit na tři úrovně abstrakce:

- **Síťová** - nejvyšší úroveň popisující topologie sítí i podsítí (uzly a linky mezi nimi) s možností udání geografické polohy a mobility.
- **Uzly v síti** - představuje architekturu jednotlivých uzlů (např. směrovačů, pracovních stanic), která je popsaná pomocí modulů a datových toků mezi nimi. Moduly typicky zahrnují protokoly nebo algoritmy, která určují chování uzlu. Struktura modulů je standardně stejná jako jednotlivé úrovně ISO/OSI modelu.
- **Procesy** - popisuje pomocí konečných automatů chování jednotlivých modulů v uzlu. Konečný automat se programuje jazykem C a C++.

OPNET Modeler umožňuje do svých modelů zapojit i reálnou síť nebo aplikace. Pakety z fyzického prostředí jsou zkonvertovány do simulátoru, kde se mohou dále upravovat. To umožňuje efektivnější možnosti simulace a testování chování celého modelu ve skutečných podmínkách.

Pro podrobné zobrazení výsledků simulace slouží OPNET's ACE. Tento produkt je komerční, ale pro univerzitní účely je poskytován zdarma. Podporuje platformy Windows a Linux/Unix [7].

2.2.3 QualNet

QualNet je software pro modelování sítí, který umožňuje simulace a emulace protokolů a sítí a zjišťuje jejich výkon. Podporuje diskrétní a real-time simulace, kde je možné použít techniky software-in-the-loop, síťové emulace, harware-in-the-loop a human-in-the-loop. Umí pracovat s tisíci uzly zároveň, což se může využít pro paralelní simulace architektur sítí a tím i rychlejšího zpracování modelů sítě. QualNet nabízí hlubokou a detailní analýzu modelu. Je možné ho provozovat na všech běžných platformách (Linux, Solaris, Windows a Mac OS).

IDE rozhraní je rozděleno do čtyř bloků dohromady tvořící kompletní nástroj pro modelování a analýzu síťové komunikace. Jsou to:

- **Scenario Designer** - vytváření a modifikace síťové topologie na úrovni uzlů a jejich typů, linek, prostředí, mobility a protokolů linkové až aplikační vrstvy.
- **Animator** - vizualizace modelu s ovládáním simulace (např. rychlost, zobrazovací filtry).

- **Analyzer** - grafická analýza statistik získaných během simulace z pohledu jednotlivého uzlu nebo protokolu.
- **Packet Tracer** - paketová analýza simulace, která zobrazuje informace o hlavičkách a jednotlivých polích paketu organizovaným způsobem.

Samotný simulátor může být použit z příkazové řádky, což umožňuje použití skriptů k lepší a pružnější modifikaci a změně parametrů simulace. Výpočet simulace je rychlejší, protože se nevynakládají prostředky pro grafické zobrazení jejího průběhu v Animatoru. Na zpracování výsledků může být opět použit již zmíněný Analyzer a Packet Tracer [4].

QualNet je komerční produkt. Pro výukové použití je nabízen ve výhodných cenových balících s možností se zapojit do QUP (QualNet University Program) viz [5].

Pro QualNet jsem nenašel žádnou zmínku o možnosti uživatelského programování na jakékoliv úrovni. Proto předpokládám, že QualNet umí pouze to, co je možné nastavit v prostředí programu.

2.2.4 OMNET++

OMNET++ je objektově orientovaný diskrétní simulátor vytvořený pro modelování sítí s otevřeným zdrojovým kódem. Dá se také využít pro simulaci telekomunikačních sítí, protokolů, sítí s frontami, modelovaní multiprocesorů a jiných distribuovaných systémů a také k validaci hardwarové architektury. Podporuje také paralelní distribuovanou simulaci (např. MPI).

V OMNET++ je simulační model složený z hierarchie modulů, které mají svoje parametry. Hloubka zanoření není limitovaná, aby bylo možné popsat jakoukoliv strukturu. Na nejnižší úrovni moduly popisují chování a programují se v C++.

Moduly mezi sebou komunikují zasíláním zpráv, které mohou obsahovat komplexní datové struktury. Zprávy se mohou posílat buď přímo k cíli, nebo po předdefinovaných cestách skrze brány a spojení. Topologie sítě je popsána jazykem NED, který používá modulární přístup a lze ho replikovat do jiných specifikací sítě.

OMNET++ má své grafické rozhraní, které poskytuje pokročilou formu správy simulace. Celé prostředí OMNET++ je možné provozovat jak na platformě Windows, tak i Linux/Unix, kde je nainstalován nějaký C++ překladač.

Komerční verze nese název OMNEST. OMNET++ je volně k dispozici k nekomerčním účelům [6].

2.3 Shrnutí

V této kapitole jsme zařadili síťové simulace do oblasti modelování a simulací. Uvedli jsme, že model systému síťové simulace je diskrétní, událostmi řízený, někde s podporou real-time simulace. Dále jsme rozdělili simulaci podle její metody na událostmi řízenou, analytickou a hybridní. Představili

jsme typické výstupy simulací (datový soubor s definovanou strukturou, animační nástroje a nástroje pro zobrazování charakteristik). Poté jsme uvedli některé ze současných simulačních a modelových nástrojů (NS, OPNET Modeler, QualNet, OMNET++).

3 Network Simulator – ns-2

V této kapitole se podrobněji seznámíme se simulačním nástrojem NS. Po uvedení historie a základní charakteristiky NS sdělíme, jakým způsobem jsou v NS tvořeny jednotlivé základní prvky (uzly, linky, směrování, agenti a protokoly aplikační vrstvy), které dohromady tvoří model sítě. Poté probereme výstupy, které produkuje NS jako výsledek simulace, a navážeme použitím výstupů pro nástroje, které je umožňují graficky reprezentovat.

3.1 Historie a hlavní rysy NS

NS vznikl jako varianta REAL network simulator v roce 1989 a v posledních letech se značně vyvinul. V roce 1995 začala vývoj NS podporovat DARPA skrze VINT project. V současnosti se na vývoji podílí kromě DARPA také NSF. Na dalším rozvoji se účastní jak jednotlivci, tak komerční i nekomerční organizace [3].

Jak bylo uvedeno výše, NS je objektově orientovaný, diskrétní (real-time simulace jsou ve stádiu vývoje), událostmi řízený síťový simulátor. Provozovat ho můžeme na platformách Linux/Unix, Windows (Cygwin). Je založen na dvou jazycích, které jsou vzájemně propojeny pomocí tclCL:

- **C++** je použitý pro jádro simulátoru, kterému dává výkonnost a rychlost provádění simulace. Jsou v něm vytvořeny veškeré algoritmy protokolů NS.
- **OTcl** (objektový Tcl) slouží pro popis a programovaní simulace (topologie sítě, specifikace protokolů, aplikací, jednotlivých prvků v síti a formy výstupu simulace) [2].

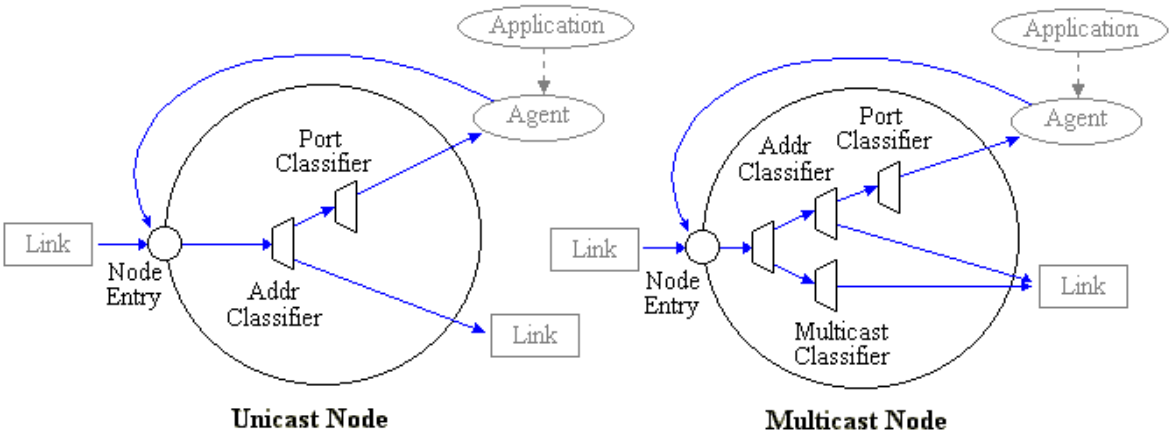
3.2 Vytváření modelu

Základní stavební jednotkou modelu síťové topologie v NS je uzel (node) a linka (link), která propojuje dva uzly. Pro vytvořenou topologii je potřeba zvolit směrovaní (statické nebo dynamické) a v případě směrování dynamického také směrovací protokol (síťová vrstva). Komunikace v síti je řešena pomocí agentů např. TCP a UDP (transportní vrstva). Na agenty se navazují protokoly aplikační vrstvy nebo různé typy datových toků.

3.2.1 Uzly

V NS jsou vytvořeny dva typy uzlů v závislosti na přenosu unicastových a multicastových paketů. Na obrázku 3.1 je znázorněna vnitřní struktura obou typů uzlů. Každý uzel má na vstupu objekt Node Entry, objekty Addr Classifier (pro unicast směrování) a Port Classifier (rozděluje pakety na základě

portů). Multicast uzel má navíc Classifier pro rozdělení unicast a multicast komunikace a Multicast Classifier (pro multicast směrování) [1].

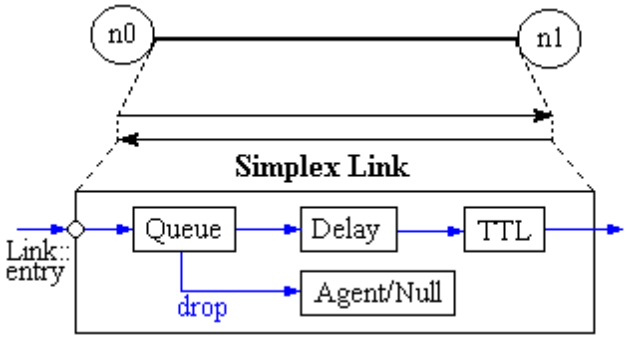


Obrázek 3.1: Implementace uzlů v NS [1]

Každý uzel je určený pouze svým jednoznačným ID datového typu u_int32_t (proměnnou v OTcl skriptu), která ho identifikuje. To je nepříjemná vlastnost z hlediska automatického vytváření topologie a identifikace pomocí IP adres. Tato problematika je rozvedena v šesté kapitole. Pro grafický výstup simulace je možné uzlům přiřadit tvar, název a barvu [2].

3.2.2 Linky

V NS existují hlavní dva typu linek: jednosměrná (directional link) a obousměrná linka (bi-derectianal), která je složená ze dvou jednosměrných linek. Jednosměrná linka má svou orientaci (počáteční uzel a koncový uzel). Implementace linky je znázorněna na obrázku 3.2, kde n0 je počáteční a n1 koncový uzel, objekty queue (fronta), delay (zpoždění), TTL představují parametry linky a do objektu Agent/Null se zahazují pakety z přeplněné fronty (queue) [2].



Obrázek 3.2: Implementace jednosměrné linky v NS2 [1]

Pro každou linku můžeme nastavit velikost fronty (defaultní hodnota je 50) a způsob, jakým bude zahazovat pakety v případě jejího přetečení. Mezi dva hlavní způsoby patří DropTail (zahazuje

naposled přijatý paket) a RED (Random Early Discart – zahozený paket je náhodně vybrán z fronty). Mezi další možnosti patří např. FQ (Fair Queueing), DDR (Deficit Round Robin). Odstraněné pakety se posílají do Agent/Null.

Zpoždění (delay) se určuje v jednotkách ms a v objektu TTL se počítá časový parametr přijatého paketu, který určuje, jak dlouho daný paket může ještě být v síti, než dosáhne cíle [2].

3.2.3 Směrování

NS podporuje unicast i multicast směrování.

3.2.3.1 Unicast

NS podporuje čtyři typy unicast směrování: Manual, Static, Session a DV. Pokud explicitně nedefinujeme směrovací techniku, je defaultně použitá Static.

Manual směrování umožňuje vytvořit ručně cesty. Zde je potřeba zdůraznit, že NS nezná své sousedy po stránce komunikace. To znamená, že při použití Manual směrování musíme vytvořit cesty i pro přímé sousedy uzlu!

Vytvářet cesty můžeme dvěma způsoby. Prvním je klasický případ, který známe z konfigurace statického směrování v jakémkoliv systému nebo zařízení (cílová IP a maska + next-hop rozhraní nebo IP adresa). Dalším je způsob, kdy pouze simulátoru řekneme, že určitá linka umožňuje v jednom směru komunikace statické směrování a poté NS sám dopočítá potřebné směrování. Navíc druhý způsob lze použít k vytvoření defaultní cesty. Oba dva způsoby lze navzájem kombinovat.

Z vlastní zkušenosti doporučuji použít pro všechny varianty Manual směrování první možnost (i na spoje mezi přímými sousedy) a pouze pro defaultní cesty použít variantu druhou. První variantou lze totiž přesně definovat jednotlivé cesty, zatímco u druhého typu taková možnost není.

Static směrování je svým principem naprosto totožné jako Manual s tím rozdílem, že jednotlivé cesty jsou vypočítány na začátku simulace a je využit Dijkstra's all-pairs SPF algoritmus.

Session směrování používá stejný algoritmus jako Static směrování. Je však rozdílné v tom, že je schopné reagovat na změny v topologii sítě za běhu simulace (např. výpadek linky). V tom případě se opět použije Dijkstra's all-pairs SPF algoritmus, který vypočte znovu směrování v síti. Tento typ směrování představuje implementaci link-state protokolů.

Posledním z typů směrování je DV, které je zástupcem distance-vector směrovacích protokolů. Používá Bellman-Ford algoritmus pro výpočet jednotlivých cest s použitím split horizon s poisoned reverse mechanizmu. Při změně topologie je opět schopný reagovat na změny v topologii. DV je možné spustit jen na určitých uzlech, nezvolené uzly pak nevysílají, nezpracovávají ani nepřeposílají aktualizace [3].

Můžeme také způsoby směrování kombinovat, ale jen omezeně: (Manual + DV, Static + DV, Session + DV). Velkým nedostatkem je, že není možné kombinovat Manual + Session! Za této varianty skončí NS vlastní chybou "Segmentation fault (core dumped)".

Za další nedostatek považuji případ, kdy NS neví, kam směrovat paket. V tomto případě dojde ke skončení simulace, místo toho, aby byl paket zahozen s odpovídající obslužnou operací a simulace pokračovala dál.

3.2.3.2 Multicast

V NS jsou k dispozici 3 typy multicast směrování: centralised, dense mode (DM) a shared tree mode (ST). Centralised směrování odpovídá PIM-SM. DM představuje implementaci protokolu dense-mode-like, který umí pracovat ve dvou režimech (PIM-DM-like a DVRMP). ST v sobě implementuje Sharp-tree multicast protokol [3].

3.2.4 Agenti a protokoly aplikační vrstvy

Agenti spojují dva uzly v síti na úrovni transportní vrstvy. Pro TCP agenta musí být definované obousměrné linky (potvrzování doručování zdroji). V NS je implementováno několik typů TCP (Tahoe, Reno, Newreno, Vegas). Pro TCP je možné nastavit množství parametrů, defaultní hodnota velikosti jednoho paketu je 1000B. Na nakonfigurovaného agenta je pak možné navázat nějakou službu, např. FTP.

Dalším typickým agentem je UDP, na který jsou spíše navázány typy datových toků, než přímo nějaké služby aplikační vrstvy. Mezi používané datové toky patří CBR (constant bit rate), u kterého můžeme nastavit velikost datového toku a pevný nebo náhodný interval jeho zasílání. Dalším představitelem datových toků je Exponential on-off a Pareto on-off [2].

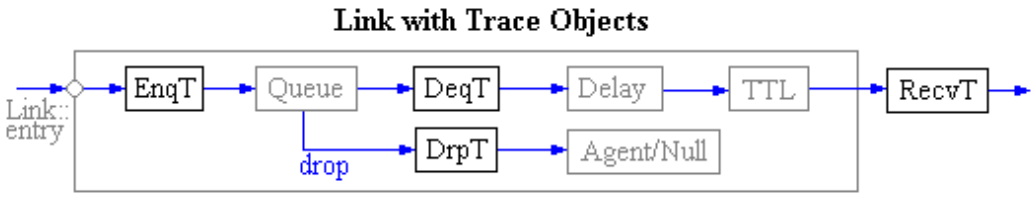
3.3 Výstup simulace

V NS rozlišujeme dva typy výstupů simulace. První typ zaznamenává jakékoliv události, které se stanou v modelu sítě během simulace. Druhý typ umožňuje sledovat fronty jednotlivý linek.

3.3.1 Událostmi řízený výstup simulace

Jedním z výstupu simulace v NS je datový soubor s definovanou strukturou, který je tvořen buď pro NAM (animátor simulace), nebo se jedná o ASCII soubor, v němž jsou uloženy veškeré události zaznamenané během simulace.

Oba dva typy souborů se tvoří na základě aktivity vyskytující se v jednosměrné (obousměrné) lince. Mezi základní objekty linky jsou vloženy čtyři další, pomocí nichž je zaznamenáván síťový provoz, viz obrázek 3.3 [2].



Obrázek 3.3: Objekty v lince zaznamenávající aktivitu [1]

EnqT registruje informace o vstupních paketech, které jsou zařazeny do fronty. DeqT uchovává informace o odchozích paketech z fronty. Pokud je paket z fronty zahozen, je tato událost zaznamenána v DrpT. RecvT pak shromažďuje informace o paketech, které úspěšně prošly linkou.

Sledovanou oblast lze definovat pomocí zadání zdrojového a cílového uzlu. Nejnižší úroveň zaznamenávání je tedy linka mezi dvěma uzly [2].

Příklad struktury výstupního ASCII souboru ukazuje následující obrázek 3.4. Bližší vysvětlení jednotlivých částí jsou uvedeny v [2].

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

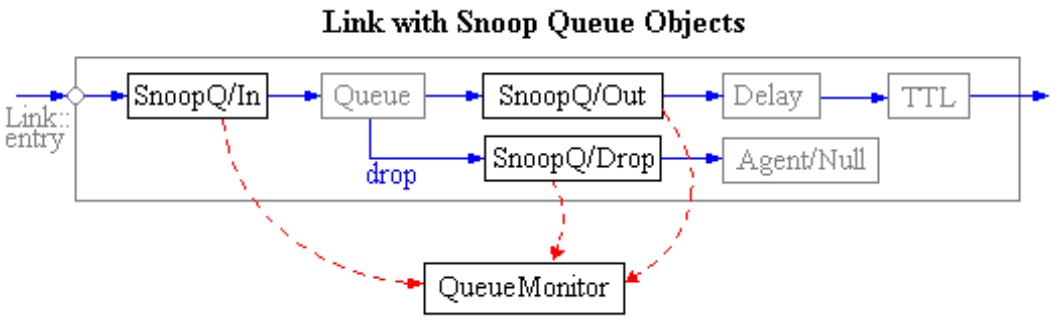
```
r : receive  (at to_node)                  src_addr : node.port (3.0)
+ : enqueue  (at queue)                    dst_addr : node.port (0.0)
- : dequeue  (at queue)
d : drop     (at queue)
```

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Obrázek 3.4: Struktura výstupního ASCII souboru a jeho příklad [1]

3.3.2 Sledování front

Předcházející případ výstupu simulace pouze vyhodnocoval tok paketů a události jimi vyvolané. Je však užitečné také vědět, co se děje s jednotlivými frontami na linkách, např. průměrná nebo aktuální velikost fronty. Jako v předcházejícím případě se i zde k objektům linky přidávají další. Vše názorně ukazuje obrázek 3.5.



Obrázek 3.5: Objekty v lince sledující frontu [1]

O všech událostech ve frontě informují SnoopQ objekty (In, Out, Drop) objekt QueueMonitor, který získané informace zpracovává.

3.4 Vizualizace simulace

Pro výstup v NS existují jak animační prostředky chování sítě, tak i nástroje pro zobrazování charakteristik. Všechny nástroje uvedené v této podkapitole je možné provozovat v prostředí Linux/Unix i Windows (Cygwin).

3.4.1 NAM

NAM (Network Animator) je 2D animační nástroj chování sítě založený na jazyku Tcl/TK. V prostředí NAM je přímo vidět topologie sítě popsaná v modelu simulace na úrovni linek a uzlů. Animovaným elementem je paket, který je vhodně barevně a velikostně zobrazován na základě rychlosti a zpoždění linky a jeho datové velikosti. NAM umožňuje u jednotlivých uzlů zobrazovat fronty a graficky reprezentovat jejich zaplnění nebo zahazování paketů.

Protože NAM neumožňuje detailnější pohled na jednotlivé součásti v síti, je vhodný pouze pro menší sítě. S narůstající velikostí modelu sítě, se animace může stát nepřehledná. Je však velmi dobrý pro základní analýzu fungování modelu sítě, např. směrování nebo ověření si, že vytvořená topologie sítě odpovídá realitě.

3.4.2 Xgraph a Gnuplot

Xgraph i Gnuplot slouží k zobrazení charakteristik. Pro každý z těchto dvou nástrojů je potřeba výstup simulace z NS přefiltrovat.

3.5 Shrnutí

Ve třetí kapitole jsme se podrobněji seznámili se simulačním nástrojem NS. Jeho základní charakteristika je, že se jedná o diskrétní, událostmi řízený simulátor založený na dvou jazycích C++ (výpočetní jádro) a OTcl (popis modelu a simulace). Dále jsme uvedli jednotlivé prvky (uzly, linky, unicast i multicast směrování, agenti, protokoly a datové toky aplikační vrstvy) a jejich složení, které se používají při vytváření modelu a jak fungují navzájem mezi sebou.

Poté jsme se seznámili se dvěma možnými výstupy simulace, kde každý z nich sleduje jistou oblast v průběhu simulace. První jsou veškeré události v síti vyvolané pakety a druhou je monitorování front na linkách.

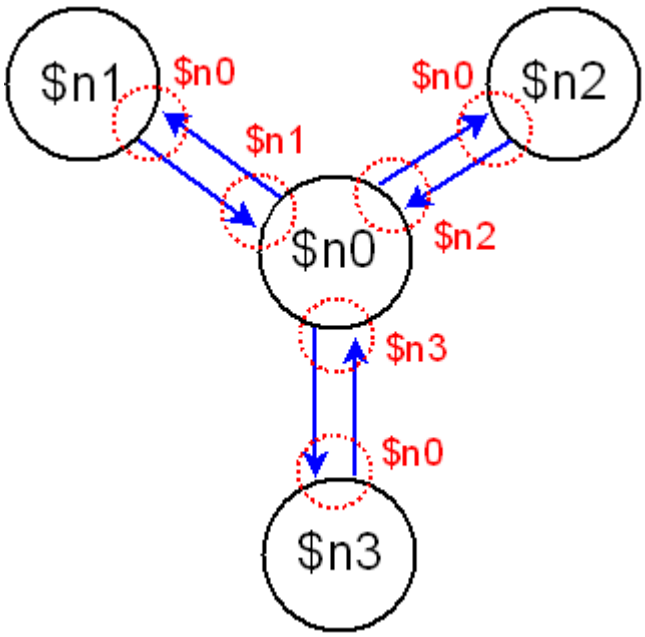
V poslední části jsme představili nástroje pro grafickou reprezentaci výstupu simulace: NAM jako animátor chování sítě a nástroje Xgraph, Gnuplot pro zobrazení charakteristik.

4 Implementace IPv4 adresace

V této části popíšeme princip implementace IPv4 adresace, který jsem vytvořil pro NS, a navážeme tak na teoretický úvod NS ve třetí kapitole. Jeho používání nijak neovlivňuje oficiální verzi NS. Má spíše informativní charakter, který pak mohou využít přidávané moduly, jako např. statické směrování nebo směrování RIP protokolem implementované v NSJS.

4.1 Princip

Princip IPv4 vysvětlíme na následujícím obrázku 4.1. Černá kolečka představují uzly. $*nx* je identifikace uzlu, kde *$nx* je název proměnné, kterou bychom našli v OTcl skriptu. Modré čáry označují jednosměrné linky a šipky jejich orientaci. Červená tečkovaná kolečka ukazují na rozhraní, která budeme adresovat.



Obrázek 4.1: Princip IPv4 adresace

V každém uzlu je pak vytvořena datová struktura rozhraní a informací o nich. Jednotlivá rozhraní ve struktuře jsou adresovatelná podle uzlu představující přímého souseda rozhraní (červeně označená identifikace uzlu *$nx* v obrázku 4.1). Datová struktura má následující obsah:

```
ifs_addr_[$next_hop_node] => ip/prefix
ifs_addr_[$next_hop_node] => přímý soused rozhraní (uzel)
ifs_addr_[$next_hop_node][network] => mask
```

Topologie sítě znázorněná na obrázku 4.1 i s definovanou IPv4 adresací popsaná v OTcl skriptu má tento tvar:

```
$n0 interface $n1 192.168.1.1 255.255.255.0
$n0 interface $n2 192.168.2.1 255.255.255.0
$n0 interface $n3 192.168.3.1 255.255.255.0
$n1 interface $n0 192.168.1.2 255.255.255.0
$n2 interface $n0 192.168.2.2 255.255.255.0
$n3 interface $n0 192.168.3.2 255.255.255.0
```

Nevýhoda tohoto přístupu k IPv4 adresaci je v tom, že je použitelný jen pro topologie, kde v jedné síti existují pouze 2 zařízení (point-to-point spoje). V síti, kde by např. switch spojoval 3 a více směrovačů dohromady, není totiž přímý soused rozhraní jednoznačný.

5 Analýza konfiguračních souborů

V tomto bloku práce se budeme zabývat rozborem konfiguračních souborů Cisco routerů po stránce převoditelnosti do NS. Podrobněji rozebereme části potřebné pro vytvoření modelu sítě v NS. Na začátku je potřeba uvést, že budeme analyzovat pouze informace, které jsou významné pro tuto práci. Jsou to nastavení pro hostname, interface, statické směrování, směrování pomocí protokolů RIP v1 a v2, EIGRP, OSPF, zabezpečení protokolu RIP a konfigurace ACL.

Každá zde analyzovaná informace bude také hodnocena podle převoditelnosti pro model sítě oficiální verze NS a rozšířené verze NSJS.

5.1 Příklad struktury konfiguračního souboru

Tuto podkapitolu začneme pro názornost ukázkou defaultního konfiguračního souboru Cisco routeru se čtyřmi fast ethernet rozhraními, kde je zahrnut popis statického směrování a dalších několik řádků navíc v konfiguraci rozhraní pro lepší pochopení problematiky:

```
!
version 12.2
no service password-encryption
!
hostname Router1
!
!
interface FastEthernet0/0
 bandwidth 1024
 ip address 192.168.102.1 255.255.255.0
 delay 5000
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.101.1 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet1/0
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet1/1
 no ip address
 duplex auto
 speed auto
 shutdown
!
ip classless
ip route 0.0.0.0 0.0.0.0 192.168.101.3
ip route 192.168.3.0 255.255.255.0 192.168.101.2 200
ip route 192.168.2.0 255.255.255.0 FastEthernet0/0
!
!
line con 0
line vty 0 4
 login
!
!
end
```

Vzhledem k tomu, že konfigurační soubor si tvoří každý router sám, a také vzhledem k selektivnímu výběru informací při zpracování pro NS převodním nástrojem (viz šestá kapitola) se nemusíme zabývat pořadím jednotlivých příkazů.

Znak *!* tvoří uvození jednořádkové poznámky. Příkazy v konfiguračním souboru můžeme rozdělit na jednořádkové a víceřádkové. Víceřádkový příkaz je složený z několika jednořádkových příkazů, kde první řádek je hlavní příkaz a zbylé řádky kromě posledního ho parametrizují. Poslední řádek je tvořen pouze znakem *!*. Příklad víceřádkového příkazu:

```
interface FastEthernet1/1
 no ip address
 duplex auto
 speed auto
 shutdown
!
```

Pokud známe tyto závislosti, můžeme je použít pro překlad konfiguračního souboru. Tuto problematiku podrobně popisuje šestá kapitola.

5.2 Potřebné informace pro NS

V následujících podkapitolách uvedeme důležité části pro vytvoření fungující topologie v modelu sítě a také parametry jednotlivých prvků, které umějí použít i NS. Ve většině případů Cisco směrovače pracují na základě defaultních hodnot. To znamená, že ač by se nám mohl zdát uvedený příklad konfiguračního souboru krátký, velká část možností je nastavena z výchozích hodnot. Explicitním uplatněním příkazu můžeme tyto parametry změnit. A právě až toto nové nastavení se projeví výskytem příkazu v konfiguračním souboru. Proto je potřeba v analýze počítat i s výchozími hodnotami, které nejsou v konfiguračním souboru vidět (např. rozdíl v konfiguraci pro *interface FastEthernet0/0* a *interface FastEthernet0/1* ve výše uvedeném příkladu).

Pro názornost bude na začátku každé podkapitoly jako příklad uvedena část z konfiguračního souboru odpovídající dané problematice.

5.2.1 Hostname

```
hostname Router1
```

Jedná se o nejjednodušší příkaz zahrnutý do překladu. Je do něj započítán především kvůli pořádku v rozlehlejších sítích, kde by se jinak mohlo použít generování jmen za cenu nižší přehlednosti. Problém, který vzniká při zachování jmen routerů, je jejich nejedinečnost, což vyřešíme tak, že k získanému názvu routeru přidáme pořadové číslo a tím unikátnost zajistíme. Formát jména bude potom vypadat: *hostname_x* , kde *x* tvoří pořadové číslo.

Tedy *hostname_x* pak bude použito pro vytvoření názvu proměnné uzlu v popisu modelu i jako popisek uzlu v grafické reprezentaci simulace.

Informace o hostname je převoditelné pro NS i NSJS.

5.2.2 Interface

Všechny parametry pro oba dva typy interface uvedené v této podkapitole jsou převoditelné pro NS i NSJS.

5.2.2.1 Ethernet

```
interface FastEthernet0/0
 bandwidth 1024
 ip address 192.168.102.1 255.255.255.0
 delay 5000
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.101.1 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet1/1
 no ip address
 duplex auto
 speed auto
 shutdown
!
```

Do této podkapitoly spadají základní typy ethernet rozhraní (Ethernet, FastEthernet, GigabitEthernet), samozřejmě s rychlostním omezením parametru speed. Všechny typy parametrů, které se mohou nastavit na Cisco routerech pro ethernet rozhraní, je možné převést do NS (viz tabulka 5.1).

Parametr	**Výchozí hodnota**	**Možné hodnoty**	**Implicitně viditelná**
bandwidth	-	1 – 10000000 [kb]	ne
delay	-	1 – 16777215 [us]	ne
duplex	auto	auto/full/half	ano
speed	auto	auto/10/100/1000	ano
shutdown	-	-	ano

Tabulka 5.1: Typy parametrů Ethernet, FastEthernet, GigabitEthernet rozhraní

Zpoždění a rychlost se v NS nedefinuje na úrovni uzlů, ale na úrovni linek. Proto je potřeba zjistit parametry z routerů, které spojuje linka.

Vzhledem k tomu, že linka v NS má pouze jeden parametr pro rychlost, vybere se z bandwidth, duplex a speed nejnižší hodnota, která se pak porovná s nejnižší hodnotou těchto tří parametrů druhého routeru. Výsledná nejmenší hodnota se aplikuje jako rychlost pro linku.

Parametr delay se opět porovnává na obou stranách. Vyšší hodnota je pak zvolena jako zpoždění linky.

Vzniká otázka, jakým způsobem porovnáváme nedefinované hodnoty? Na tuto otázku předkládá odpověď šestá kapitola.

5.2.2.2 Serial

```
interface Serial1/0
 no ip address
 shutdown
!
interface Serial1/1
 bandwidth 32000
 ip address 10.0.0.1 255.0.0.0
 delay 5000
 clock rate 56000
!
```

U Serial rozhraní můžeme opět použít veškeré parametry (viz tabulka 5.2).

Parametr	Výchozí hodnota	Možné hodnoty	Implicitně viditelná
bandwidth	-	1 – 10000000 [kb]	ne
delay	-	1 – 16777215 [us]	ne
clock rate	-	300 – 4000000 [b/s]	ne
shutdown	-	-	ano

Tabulka 5.2: Typy parametrů Serial rozhraní

Problém s nemožností parametrizovat uzel, ale linku v NS, pokud se týká vlastností pro Serial rozhraní, je i zde. Řešení je stejné jako u ethernet rozhraní, s tím rozdílem, že nejmenší hodnota pro rychlost linky se vypočítává pouze z parametrů bandwidth a clock rate.

5.2.3 Směrování

Přestože v dostupných materiálech [3] je napsáno, že v NS můžeme přiřazovat prioritu jednotlivých cest (ekvivalent pro Cisco administrative distance), ani po mnoha testováních a dlouhé době hledání východiska na internetu se mi nepodařilo prioritizaci zprovoznit. V NSJS je však priorita cest funkční.

5.2.3.1 Statické směrování a výchozí brána

```
ip route 0.0.0.0 0.0.0.0 192.168.101.3
ip route 192.168.3.0 255.255.255.0 192.168.101.2 200
ip route 192.168.2.0 255.255.255.0 FastEthernet0/0
```

Statické cesty popsané v konfiguračním souboru jsou kompletně převoditelné do NS i NSJS. Problémem při vytváření modelu sítě pro NS je adresace uzlů podle IP adres nebo názvu rozhraní (transformace IP adres na ID uzlu). Řešení této problematiky je popsáno v šesté kapitole. V NSJS takovýto problém není, protože pro definici statického směrování můžeme použít přímo IP adresy. Pro obě varianty však platí nutnost převodu názvu rozhraní na ID uzlu v případě NS, respektive IP adresy v případě NSJS.

5.2.3.2 RIP v1 a v2 v NS

```
router rip
 version 2
 network 10.0.0.0
!
```

RIP v1 i v2 je představitelem routovacích protokolů typu distance-vector. Tomu v NS odpovídá směrování DV. Jak již bylo zmíněno ve třetí kapitole, DV je možné ovládat pouze na úrovni povolení/zakázání používání DV v uzlu. I tak není možné protokoly RIP dostatečně parametrizovat, protože by bylo potřeba specifikovat komunikaci protokolu RIP na úrovni rozhraní uzlu (příkaz *network*). Ostatní parametry a nastavení, které nabízí Cisco routery, není možné v NS uplatnit.

Vzhledem k omezujícím možnostem NS je v modelu sítě pro NS směrování DV v uzlu povoleno jen tehdy, pokud v konfiguračním souboru existuje hlavní příkaz *router rip* obsahující minimálně jeden parametrizující příkaz *network x.x.x.x*

5.2.3.3 RIP v1 a v2 v NSJS

```
interface FastEthernet1/0
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
 ip rip receive version 1
 ip rip send version 2
 ip split-horizon
 ip rip authentication key-chain flinstone
 ip rip authentication mode md5
!
!
key chain flintstone
 key 1
 key-string fred
 key 2
 key-string barney
 accept-lifetime 00:00:00 Dec 5 2008 23:59:59 Dec 5 2008
 send-lifetime 06:00:00 Dec 5 2008 18:00:00 Dec 5 2008
!
```

Oproti NS, NSJS přímo podporuje směrovací protokol RIP v1 i v2 i jeho zabezpečení komunikace mezi směrovači. Všechny příkazy zde uvedené jsou převoditelné do NSJS i pokud se týká funkčního významu, proto si uvedeme jen jejich výčet.

Podporované příkazy pro konfiguraci protokolu RIP v hlavním příkazu *router rip*: *network, version, distance, auto-summary, no auto-summary, timers basic, passive-interface.*

Podporované příkazy pro konfiguraci protokolu RIP v hlavním příkazu *interface*: *ip rip receive version, ip rip send version, ip split-horizon, no ip split-horizon.*

Podporované příkazy pro vytvoření klíčů zabezpečené komunikace: hlavní příkaz *key chain* a jeho parametrizující příkazy: *key, key-string*.

Podporované příkazy pro aplikování klíčů na interface v hlavním příkazu interface: *ip rip authentication key-chain, ip rip authentication mode.*

5.2.3.4 EIGRP

```
router eigrp 1
 network 10.0.10.0
 network 10.0.20.0
 auto-summary
!
router eigrp 2
 network 192.168.1.0
 network 192.168.2.0
 auto-summary
!
```

EIGRP je v základu typem distance-vector směrovacích protokolů. Proto jako jeho reprezentaci v NS použijeme DV. Platí však pro něj stejné omezení a způsob použití jako v případě RIP.

V EIGRP existují tzv. autonomní systémy, které rozdělují síť na menší části, jež pak mohou tvořit hierarchii (např. pobočkové sítě a páteřní síť). Směrování se pak děje jen uvnitř autonomních systémů. Tento princip také není podporován v NS.

V NSJS není EIGRP podporován.

5.2.3.5 OSPF

```
router ospf 1
 log-adjacency-changes
 network 10.0.0.0 0.0.0.255 area 0
!
router ospf 2
 log-adjacency-changes
 network 192.168.1.0 0.0.0.255 area 0
 network 192.168.2.0 0.0.0.255 area 0
!
```

OSPF řadíme mezi směrovací protokoly typu link-state. V NS je tato rodina routovacích protokolů zastoupena typem směrování Session, ve kterém nelze nastavovat ani používané uzly, jak tomu je aspoň u DV. Tento typ směrování můžeme pouze zapnout nebo vypnout na úrovni celého simulačního modelu. Odpadá také možnost konfigurace veškerých parametrů dostupných v Cisco routerech nebo použití více *area* v jednom modelu sítě.

Tento typ směrování je uplatněn v simulaci na základě stejného principu jako u RIP. Pokud hlavní příkaz *router ospf x* obsahuje aspoň jednu konfiguraci sítě *network x.x.x.x y.y.y.y area z,* je v simulaci tento typ směrování spuštěn.

V NSJS není OSPF podporován.

5.2.4 Access control list

Teoretickému výkladu, podrobnému popisu způsobu převodu, míře převoditelnosti a realizaci ACL v NS se věnuje osmá a devátá kapitola práce. Zde uvedeme pouze informace související s tématem této kapitoly.

Podporované ACL v této práci jsou: standard, extended, ip named a dynamic. Kde standard ACL jsou převoditelné v celém rozsahu a zbylé tři typy z velké části. Míra převoditelnosti i funkčnost je pro model sítě v NS i pro NSJS stejná.

Standard, extended a dynamic ACL lze definovat pomocí jednořádkového příkazu (*access-list*) např.:

```
access-list 102 permit ip 192.168.3.0 0.0.0.255 192.168.1.0 0.0.0.255
access-list 102 permit tcp 192.168.3.0 0.0.0.255 192.168.1.0 0.0.0.255 eq www
```

IP named ACL představují přehlednější definici pravidel ACL. Můžeme však vytvářet pouze standard a extended ACL. Definici IP named ACL v konfiguračním souboru tvoří víceřádkový příkaz (*ip access-list*) např.:

```
ip access-list extended rdp
 permit tcp host 192.168.2.25 host 192.168.1.2 eq 3389
 permit tcp host 192.168.2.31 host 192.168.1.2 eq 3389
 permit tcp host 192.168.2.31 host 192.168.1.3 eq 3389
!
```

5.3 Shrnutí

V páté kapitole jsme si ukázali lehce změněný defaultní konfigurační soubor Cisco routeru. Uvedli jsme, že příkazy v konfiguračním souboru můžeme rozdělit na jednořádkové a víceřádkové a že příkaz *!* uvozuje jednořádkovou poznámku.

Dále jsme analyzovali konfigurační soubor a zjistili jsme, do jaké míry jsou převoditelné vybrané informace do NS a NSJS.

Uvedli jsme, že do modelu sítě v NS můžeme bez problémů přenést informace o hostname routeru a veškeré údaje o ethernet a serial rozhraní. Převod směrování do NS má však velké omezení. Zatímco statické směrování jsme schopni kromě priorit kompletně převést, dynamické směrování (RIP, EIGRP i OSPF) jen velmi povrchně.

Do modelu sítě pro NSJS můžeme také přenést veškeré informace o hostname a nastavení serial a ethernet rozhraní. Oproti NS můžeme statické směrování definovat pomocí IP adres a je i funkční prioritizace cest. Směrování pomocí RIP protokolu v1 i v2 je přímo v NSJS podporováno i s možností nastavení zabezpečení komunikace mezi routery. Není však možné převést směrování pomocí EIGRP a OSPF protokolu.

Míra převoditelnosti ACL je pro NS i NSJS stejná. Podporovány jsou standard, extended, ip named a dynamic ACL.

6 Převod konfigurace do NS a NSJS

V této části podrobně popíšeme způsob převodu konfiguračních souborů Cisco routerů do NS a NSJS. Pomocí získaných informací vytvoříme model sítě a jednoduchou simulaci.

Samotný překlad postupuje v následných krocích, které budou postupně popisovat následující podkapitoly: získání potřebných informací z konfiguračních souborů, vytvoření uzlů a linek (topologie), aktivace a konfigurace směrování, definice ACL a vytvoření kontrolní simulace.

6.1 Získání informací z konfiguračních souborů

Pro dočasné uložení informací z Cisco routerů je v překladu vytvořeno dynamické pole. Jeho struktura je následující:

```
$a_routers[$i]
        [hostname]
        [interfaces][$j]
                [type]
                [port]
                [ip]
                [ip_bin]
                [mask]
                [mask_bin]
                [acl][$k]
                        [number]
                        [direction]
                [ip_rip][$k]
                        [rs]
                        [version]
                [split_horizon]
                [no_split_horizon]
                [key_chain_name]
                [key_chain_mode]
                [duplex]
                [speed]
                [delay]
                [bandwidth]
                [clock_rate]
                [shutdown]
        [acl][$j]
        [ip_acl][$j]
                [line][$k]
        [routing][static][$j]
                [dst]
                [mask]
                [next_hop]
                [distance]
        [routing][rip][$j]
                [version]
                [network][$k]
                [distance]
                [auto_summary]
                [no_auto_aummary]
                [timers_basic]
                [passive_interface][$k]
                        [type]
                        [port]
        [routing][eigrp][$j]
                [as]
                [network][$k]
        [routing][ospf][$j]
                [proc]
                [network][$k]
```

```
                [network]
                [wc_mask]
                [area]
        [key_chain][$j]
                [name]
                [keys][$k]
                [key]
                [string]
```

U překladu využíváme toho, že dopředu známe syntax jednotlivých příkazů. Musíme si však dát pozor na jednořádkové a víceřádkové příkazy. Každý konfigurační soubor procházíme řádek po řádku a hledáme textové řetězce, které uvozují námi hledanou informaci. Navíc také víme, že tyto uvozovací řetězce jsou svým obsahem jedinečné.

6.1.1 Jednořádkový příkaz

Jednořádkové příkazy se skládají vždy z dvojice <uvození> <informace>, jak je možné také vidět v následujícím příkladu:

```
ip route 192.168.3.0 255.255.255.0 192.168.101.2 200
```

Na základě uvození *ip route* víme, že se jedná o definici statické cesty a tudíž víme, že jako informace budou následovat tři povinné parametry a jeden volitelný parametr oddělené mezerou (<dst network> <mask> <next_hop> <distance>). Zde *192.168.3.0 255.255.255 192.168.101.2 200*. Zjištěné hodnoty uložíme do pole. Tímto způsobem vyhodnotíme všechny ostatní jednořádkové příkazy.

6.1.2 Víceřádkový příkaz

Víceřádkové příkazy jsou složeny z několika jednořádkových, kde první příkaz je hlavní a zbylé ho parametrizují. Opět zde platí zákonitost pro hlavní i jednořádkový příkaz, tedy dvojice <uvození> <informace>. Víceřádkový příkaz je vždy ukončen *!*. Typický víceřádkový příkaz je např. definice rozhraní:

```
interface FastEthernet1/1
 no ip address
 duplex auto
 speed auto
 shutdown
!
```

Uvození hlavního příkazu zde tvoří *interface* a informaci *FastEthernet1/1*. Parametrizující příkazy se vyhodnotí jako jednořádkové. Vše probíhá do doby, kdy je pro překlad načten řádek obsahující *!*. Takto budeme postupovat u všech ostatních víceřádkových příkazů. Zjištěné hodnoty opět uložíme do pole.

6.1.3 Jednotky

V Cisco směrovačích jsou určité parametry rozhraní zadávány v různých jednotkách, které nesouhlasí s jednotkami v NS (viz tabulka 6.1):

Parametr	**Jednotka**
[delay]	µs
[bandwidth]	kb
[clock_rate]	b/s

Tabulka 6.1: Jednotky parametrů rozhraní na Cisco směrovačích

Zpoždění (delay) je v NS definováno v ms. Při získávání informací se proto delay dělí 1000. NS dále podporuje všechny používané jednotky rychlosti. Avšak vlivem porovnávání hodnot kvůli zjištění rychlosti linky jsou všechny hodnoty převedeny na b nebo b/s.

6.2 Uzly a linky

Na začátku vytvoříme seřazené dynamické pole podle binární hodnoty masky sítě (viz následující příklad). V poli jsou zahrnuta pouze aktivní rozhraní. Tedy ta, která v hlavním příkazu *interface* neměla příkaz *shutdown*.

```
$a_network[$i]
        [mask_bin]
        [ip_bin]
        [hostname]
        [idr]
        [idi]
        [linked]
```

[idr] a [idi] představuje zpětnou vazbu do pole $a_routers. [linked] bude zaznamenávat stav svázání rozhraní s nějakou linkou.

Následně budeme ve dvojnásobném cyklu procházet pole $a_network a na základě [mask_bin] a [ip_bin] budeme hledat IP adresy, které patří do stejné sítě. V tomto bodě využíváme seřazeného pole od nejvyšší masky (bráno jako binární číslo) po nejnižší a zabraňujeme tak chybnému přiřazení IP adresy do jiné sítě.

Při prohledávání pole mohou nastat dva případy nalezení lišící se v počtu IP adres spadající do jedné sítě:

- **2 IP adresy** – spojení typu point-to-point.
- **3 a více IP adres** – spojení typu hvězda.

6.2.1 Spojení typu point-to-point

Pro dvě IP adresy je situace jednoduchá:

1. Vytvoříme linku mezi dvěma uzly (bude popsáno dále).

2. Změníme parametr [linked] u rozhraní routerů v poli $a_network.
3. V případě vytváření modelu sítě pro NS definujeme statické směrování mezi uzly (bude popsáno v následující podkapitole). Pokud však model sítě generujeme pro NSJS, tento krok přeskočíme.

6.2.2 Spojení typu hvězda

Před popisem vytvoření hvězdy v NS je důležité si uvědomit, že NS podporuje pouze spoje typu point-to-point. Je tedy zapotřebí místo (např. switch), které všechny routery ve hvězdě spojuje, nahradit v NS novým uzlem, do něhož povedou linky ze všech routerů tvořící hvězdu. Musíme se však vypořádat se statickým směrováním, protože jsme vytvořili nový uzel, který porušuje jeho konfiguraci. Takto nově vytvořený uzel budeme nazývat *super_node_x* a tento název bude vystupovat jak jako název proměnné v modelu simulace, tak i jako název uzlu v grafických nástrojích zobrazujících simulaci, *x* potom představuje počítadlo (tvoří unikátní název).

Pro super_node je vytvořeno následující pole:

```
$a_super_nodes[$i]
        [hostname]
        [interfaces][$j]
                [type]
                [ip]
                [ip_bin]
                [mask]
                [mask_bin]
        [routing][static][$j]
                [dst]
                [mask]
                [next_hop]
```

Super_node musíme adresovat pomocí IP adresy a masky. Abychom nevytvořili kolizi v síti, přidělíme super_node IP adresu sítě, tím zajistíme i správnou náležitost do sítě routerů tvořících hvězdu. Vzhledem k adresování uzlů v NS podle jejich ID, není přidělená adresa sítě jako adresa rozhraní chybou. Jedná se pouze o pomůcku převodního nástroje. Hvězdu v NS pak vytvoříme následovně:

1. Vytvoříme nový super_node a nadefinujeme mu [interfaces].
2. Budeme procházet všechny statické cesty u směrovačů tvořících hvězdu a pokud najdeme [next_hop] patřící do hvězdy, změníme hodnotu v [next_hop] na IP adresu super_node a v super_node vytvoříme nový záznam statického routování odpovídající původnímu záznamu v routeru.
3. Vytvoříme linky mezi super_node a všemi routery ve hvězdě.
4. Nastavíme parametr [linked] pro příslušné rozhraní ve směrovačích.
5. Vytvoříme statické směrování mezi super_node a routery tvořící hvězdu (pouze sousední routery).

Zde je důležitá odlišnost mezi vytvářením modelu sítě pro NS a NSJS. V NSJS jsou pro adresování uzlů použity klasické IP adresy. Jak již však bylo uvedeno ve čtvrté kapitole, použitý princip adresace umožňuje pouze spojení typu point-to-point v topologii sítě. Proto pokud převodní nástroj rozpozná spojení typu hvězda a bude se vytvářet model sítě pro NSJS, převodní nástroj skončí a nedojde k vytvoření modelu ani simulace.

6.2.3 Nespojené aktivní rozhraní

Po skončení hledání IP adres patřících do jedné sítě mohou zůstat nepřipojena některá aktivní rozhraní (nejsou svázána se žádnou linkou). To může být způsobeno tím, že např. v reálném zapojení je toto aktivní rozhraní připojeno ke switchi a zaštiťuje konektivitu určitého segmentu sítě. V těchto případech se vytvoří nový uzel, který bude představovat tyto části topologie sítě. Novému uzlu budeme přiřazovat název *auto_node*. Pole, v němž jsou uloženy informace o auto_node uzlech, je následující:

```
$a_auto_nodes[$i]
        [hostname]
        [interfaces][$j]
                [type]
                [ip]
                [ip_bin]
                [mask]
                [mask_bin]
        [routing][static][$j]
                [dst]
                [mask]
                [next_hop]
```

Tak jako v případě uzlů super_node i auto_node uzel bude mít na konci svého názvu počítadlo, které bude zajišťovat jeho unikátní jméno. Výsledkem tedy bude název: *auto_node_x* kde *x* je počítadlo.

Proces vytvoření auto_node je následující:

1. Vytvoříme nový auto_node.
2. Na základě IP adresy a masky sítě aktivního rozhraní ke kterému auto_node připojujeme, vygenerujeme novou IP adresu pro příslušnou síť. Poté nadefinujeme hodnoty v [interfaces].
3. Vytvoříme linku mezi auto_node a routerem.
4. Změníme parametr [linked] u rozhraní routeru.
5. V případě generování modelu sítě pro NS vytvoříme statické směrování mezi auto_node a routerem. V NSJS tato operace není potřebná.
6. V auto_node nastavíme výchozí bránu na router.

Pokud v druhém bodě nenalezneme volnou IP adresu, převodní nástroj skončí a nedojde k vytvoření modelu sítě a simulace.

6.2.4 Tvoření linek

Jak již bylo uvedeno, výsledná rychlost linky a zpoždění se určují na základě porovnávání hodnot jednotlivých parametrů rozhraní routerů. Některé hodnoty nejsou však z konfiguračních souborů zjistitelné, protože se použije jejich defaultní hodnota. Proto je vytvořen soubor (variables.php) obsahující defaultní a parametrizující hodnoty, které se použijí v případě nedefinovaného obsahu.

6.3 Směrování

V této podkapitole ukážeme, jak je v převodním nástroji implementováno statické i dynamické směrování pro NS a NSJS.

6.3.1 Statické pro NS

Než začneme zpracovávat získané informace o statickém směrování, musíme routovací záznamy v $a_routers prověřit, zdali [next_hop] není určen rozhraním routeru. Pokud ano, musíme tuto hodnotu nejdříve převést na IP adresu. Poté můžeme přikročit k vytváření statického směrování.

I když NS podporuje statické směrování, klasický parametr cílové sítě, který postačí standardně zadat, aby bylo dostupné jakékoliv zařízení v síti, zde takto nefunguje. Cílová síť v NS představuje pouze jeden uzel, proto je potřeba vytvořit statické cesty pro všechny uzly v sítí zvlášť.

Jak již bylo v této práci zmíněno, NS umí adresovat uzly pouze podle jejich ID (názvů proměnných). Proto celý proces tvoření statického směrování doprovází vyhledávání hostname routerů (budoucích názvů proměnných uzlů v NS) na základě IP adres nebo příslušnosti do určité sítě.

Po vytvoření topologie sítě máme tři pole uzlů: $a_routers, $a_super_node a $a_auto_node. Pole $a_auto_node pro vytváření směrování brát v úvahu nemusíme, protože víme, že se jedná pouze o koncové uzly a směrování pro tyto uzly již bylo vytvořeno. Tvoření statického směrování pro $a_routers a $a_auto_node je založeno na stejném principu:

1. Procházíme pole směrovačů ve dvojnásobném cyklu.
2. Ve vnějším cyklu najdeme [hostname] zdrojového routeru.
3. Ve vnitřním cyklu najdeme [hostname] [next_hop] routeru.
4. Ve vnitřním cyklu najdeme [hostname] všech routerů v [dst] síti.
5. Vytvoříme příslušné cesty.

6.3.2 Statické pro NSJS

Pro model sítě v NSJS musíme také převést [next_hop] na IP adresu, pokud je určen rozhraním. Další operace však nejsou potřebné. Vlivem použití IPv4 adresace jsme schopni přímo tvořit definice statického směrování v podobě, které jsou potřeba pro OTcl skript.

6.3.3 Dynamické pro NS

Dynamické směrování se bohužel musí řídit omezeními NS, která jsou popsána ve třetí kapitole. Proto u protokolů RIP, EIGRP a OSPF zkoumáme pouze výskyt hodnoty v [network] jednotlivého protokolu. Podle zjištěného stavu se dále rozhodujeme, jakou kombinaci směrovacích protokolů použijeme v simulaci.

Problém kombinace Manual + Session směrování je vyřešeno prioritou Session. Tedy pokud použijeme Session směrování v modelu sítě, nepoužijeme ani jednu statickou cestu.

Další nemožnou kombinací je použití více DV v jedné simulaci (např. RIP + EIGRP). S tímto nedostatkem se vypořádáme tak, že v případě existence obou dvou protokolů budeme upřednostňovat RIP.

6.3.4 Dynamické pro NSJS

NSJS podporuje pouze jeden typ dynamického směrování. Tím je RIP v1 i v2. Je možné ho kombinovat se statickým směrováním. Opět vlivem IPv4 adresace a přímé podpory Cisco příkazů již jen generujeme příslušný kód do OTcl skriptu, který popisuje jak aktivaci protokolu RIP na jednotlivých uzlech tak i konfiguraci parametrů.

6.4 Access control listy

Pro ACL platí v NS i NSJS stejné zákonitosti. ACL jsou implementovány tak, že pokud je chceme v simulačním modelu používat, musíme je nejdříve pro celou simulaci povolit. Povolení však převodní nástroj provede jen na základě zjištění, že aspoň jeden router obsahuje definici ACL na rozhraní.

Dále je již převodní nástroj pro ACL čistě jen překladač, který transformuje Cisco definici ACL na definici použitou v OTcl skriptu. Veškeré zpracování se děje až za běhu simulace.

6.5 Model sítě a jednoduchá simulace

Jakmile máme k dispozici vytvořený model sítě, můžeme jej vložit do simulace. Ta je v převodním nástroji realizována tak, že z modelu sítě zjistíme dostupné *auto_node_x* a v sekundových intervalech budeme simulovat FTP komunikaci mezi prvním *auto_node_x* a všemi ostatními *auto_node_x*.

Vzhledem k tomu, že při použití statického směrování a modelu sítě tvořeného pěti routery, pěti auto_node a dvěma super_node trvá při takto zvolené FTP komunikaci vyhodnocení simulace desítky sekund, nejsou v testované komunikaci zahrnuty i další kombinace dvojic uzlů auto_node.

V generovaném popisu simulace je i zahrnuto spuštění animačního nástroje NAM, který zjištěné chování sítě zobrazí.

6.6 Shrnutí

V této části práce jsme teoreticky popsali postup, kterým z konfiguračních souborů Cisco routerů vytvoříme model sítě a následně i jednoduchou simulaci.

Prvním krokem v modelování sítě je získání potřebných informací z konfiguračních souborů. Zde jsme uvedli dva typy příkazů (jednořádkový a víceřádkový), které musíme zpracovat. Rovněž jsme se seznámili s datovou strukturou, která ukládá zjištěné informace. Při zpracovávání konfiguračních souborů také musíme dát pozor na jednotky používané v Cisco zařízeních a v NS.

Dalším krokem je vytvoření uzlů a linek (topologie sítě). Bylo uvedeno, že NS umí pracovat pouze se spojením typu point-to-point. Možné typy zapojení ve fyzické topologii jsme si proto rozdělili na point-to-point a hvězdu, které musíme na systém point-to-point převést. Při převodu hvězdy se jako centrální prvek (např. náhrada za switch) vytváří nový uzel *super_node_x*. Zde musíme upravit statické směrování vlivem přidání dalšího uzlu. K aktivním rozhraním, ke kterým ještě v tomto bodě vytváření topologie není připojena linka, se vytvoří nový uzel *auto_node_x* (např. jeden koncový segment fyzické sítě) a s aktivním rozhraním routeru se vytvoří spojení pomocí linky. Tímto je model topologie sítě vytvořen.

U NSJS musíme počítat s omezením vlivem použitého principu IPv4 adresace, jenž umožňuje simulovat jen ty reálné topologie sítí, ve kterých se vyskytují pouze spojení typu point-to-point.

V modelu sítě musíme dále popsat směrování. Statické směrování pro NS vytvoříme ze zjištěných a upravených hodnot, kde se musíme vypořádat s adresací uzlů podle ID a také s neschopností NS použít jednu definici statického směrování pro všechna zařízení v cílové síti. U dynamického směrování musíme postupovat velmi omezeně (nemožnost kombinování některých směrovacích technik, nedostatečná úroveň správy a ovládání routovacího protokolu). Proto se omezujeme pouze na výskyty příkazů v konfiguračních souborech a podle nich směrovaní v modelu sítě buď povolíme, nebo zakážeme.

Definování statického i dynamického směrování je při vytváření modelu sítě pro NSJS velmi jednoduché a přímé, protože používáme přímo IP adres ke konfiguraci směrování. V podstatě řešíme pouze překlad Cisco příkazů na příkazy v OTcl skriptu.

V předposlední části jsme uvedli, že převod ACL do NS i NSJS je stejného charakteru a vlivem zpracování ACL až na straně simulačního programu řeší převodní nástroj pouze transformaci příkazů z Cisco routerů do OTcl skriptu.

Na závěr této kapitoly jsme uvedli obsah generované simulace a testovaní modelu sítě pomocí FTP komunikace.

7 Převodní program

V této kapitole popíšeme převodní program z uživatelského pohledu. Seznámíme se s jeho zdrojovými kódy a ukážeme způsob používání. Příkazy z Cisco konfiguračních souborů, které podporuje převodní nástroj i s mírou převoditelnosti do NS i NSJS a s příklady použití, jsou uvedeny v příloze této práce.

7.1 Zdrojové kódy

Převodní nástroj se skládá ze sedmi PHP souborů:

- **cisco.php** – hlavní soubor
  - **func_parser.php** – získání informací z konfiguračních souborů routerů
  - **func_addressing.php** – převodní a zjišťovací algoritmy pro IP adresy
  - **func_string.php** – funkce pro práci s textovými řetězci
  - **func_ns.php** – vytváření uzlů, linek a statického směrování
  - **func_create_ns.php** – vytváření modelu sítě a simulace
  - **variables.php** – výchozí a volitelné parametry

7.2 Použití

Převodní program se spouští z příkazové řádky. Syntax příkazu (pro Windows) je následující:

```
php –q cisco.php [-ipv4 enable] [-r|-s r_file|s_file]+ [-o o_file]
```

- -r|-s – rozlišení použitého konfiguračního souboru buď z routeru, nebo switche
- r_file|s_file – konfigurační soubor routeru nebo switche
- -o o_file – vygenerovaný tcl soubor
- -ipv4 enable – v případě použití tohoto parametru se bude vytvářet simulační model pro NSJS, pokud tento parametr nebude zadán, model bude vytvořen pro NS
- -help|-h|-? – zobrazí nápovědu

Simulaci poté můžeme spustit přikazem:

```
ns o_file
```

Samotný proces načítání parametrů z konfiguračních souborů kontroluje nepodporované příkazy. Pokud na takový narazí, je zobrazena např. následující informace do konzole:

```
Unsupported cmd in file rr2.txt at line 3: no service password-encryption
```

8 Filtrování paketů pomocí access control list

V této kapitole se teoreticky seznámíme s ACL. Budeme pouze rozebírat IP ACL, které jsou předmětem této práce. Nejříve ACL zařadíme do světa bezpečnosti sítí, poté se podrobně seznámíme s možnostmi a zásadami jejich použití a principem činnosti. Také popíšeme způsob porovnávání IP adres pomocí wildcart masky. V poslední části uvedeme způsob aktivování ACL na rozhraní routerů a rozebereme typy ACL, které jsou implementovány v NS i NSJS.

8.1 Úvod

Bezpečnost v síti v dnešní době představuje široký pojem, do kterého spadají nejrůznější fyzické, hardwarové i softwarové oblasti počítačového světa. Jedním z hlavních bezpečnostních prostředků jsou firewally.

Firewall bývá zpravidla jednoúčelové zařízení, jeden z modulů zařízení nebo program běžící na nějakém počítači. Jeho funkcí je kontrola síťového provozu přes něj procházející. Na základě definovaných pravidel je schopný určitý typ komunikace buď povolit, nebo zakázat. Firewally můžeme rozdělovat na:

- **Paketové filtry** – jsou první generací firewallů vyvinuté v roce 1988. Pracují na principu inspekce jednotlivých paketů, které zpravidla zkoumají na 3. a 4. úrovni ISO/OSI modelu. Nejčastěji kontrolované hodnoty bývají zdrojová a cílová adresa a port a také typ protokolu. Výhodou tohoto řešení je vysoká rychlost zpracování paketů, nevýhodou je jejich jednoduchost a buď přílišná komplikace definování pravidel, nebo přímo nemožnost použití na složitější protokoly. To, zdali paket bude přeposlán nebo zahozen, závisí na pravidlech, podle kterých se firewall řídí.
- **Stavové paketové filtry** – jsou vylepšenou variantou předchozího typu. Umožňují kromě funkcí paketových filtrů detekovat navázané spojení. To přináší výhodu v tom, že pokud kontrolovaný paket patří do již navázaného spojení, je automaticky přeposlán. Tímto je tento typ firewallů ještě rychlejší, protože nedochází ke kompletní kontrole jednotlivých paketů. Navíc stačí u známých protokolů např. FTP pouze definovat pravidla pro navazující směr komunikace a firewall sám vytvoří zbylá pravidla pro navázání celého přenosu v obou směrech. Tímto se také zvětšuje přehlednost v definicích pravidel.
- **Aplikační brány (proxy firewally)** – pracují a provádějí kontrolu na aplikační úrovni ISO/OSI modelu a zcela oddělují sítě, mezi které jsou postaveny. Veškerá komunikace

přes aplikační bránu probíhá ve dvouch spojeních. Klient nejdříve naváže spojení s aplikační bránou, ta zpracuje požadavek klienta a vytvoří nové spojení s požadovaným cílem klienta. Odpověď od cíle je pak přenesena přes původní spojení mezi klientem a aplikační bránou. Aplikační brány zpravidla provozují NAT, jsou schopny vytvářet šifrované spojení mezi sebou a požadovaným cílem, poskytují důkladnou inspekci komunikace, ale vyhodnocování zakázání nebo povolení spojení může být pomalejší, protože se provádějí až na aplikační úrovni [8].

8.2 ACL podrobněji

ACL řadíme mezi paketové filtry. I když jsou směrovače obecně zařízení 3. vrstvy ISO/OSI modelu, umějí ACL zpracovávat informace z paketu nejen z 3. ,ale i ze 4. vrstvy. ACL se skládá ze sady pravidel a na základě jejich porovnání s informacemi v paketu se router rozhoduje, zdali paket přepošle nebo zahodí. Mezi hlavní oblasti kontroly patří:

- Zdrojová/cílová adresa zařízení nebo sítě
- Zdrojový/cílový port
- Typ protokolu
- Typ ICMP zprávy

ACL může být v Cisco směrovači aplikovatelný podle:

- Protokolu – kontrola se provádí na rozhraní pro každý povolený (aktivní) protokol. Jeden ACL musí být definovaný pro každý jeden typ protokolu.
- Směru – komunikace je kontrolována podle směru komunikace (příchozí a odchozí). Pro každý směr musí být definován jeden ACL.
- Rozhraní – ACL se aplikují pro jednotlivá rozhraní.

Pokud bychom měli např. směrovač se dvěma rozhraními, na kterých by byly povoleny protokoly IP, AppleTalk a IPX, můžeme použít až 12 jednotlivých ACL (3 pro jednotlivé protokoly * 2 pro směry * 2 pro rozhraní) [9].

8.2.1 Princip činnosti

Kontrola paketů procházejících směrovačem pracuje následujícím principem. Pokud existuje nějaký ACL pro příchozí paket na rozhraní, je paket zkontrolován a na základě výsledku je buď puštěn do směrovače, nebo zahozen. Pokud neexistuje ACL pro příchozí paket, je paket do směrovače vpuštěn. Ve směrovači se na paket aplikují routovací mechanismy. Pokud je nalezena cesta, je paket přeposlán na odchozí rozhraní. Pokud není nalezena, je paket zahozen. Na odchozím rozhraní se opět kontroluje, zdali pro paket existuje nějaký ACL. Pokud ano, je paket zkontrolován a na základě výsledku je buď poslán, nebo zahozen. Pokud na odchozím rozhraní neexistuje ACL pro paket, je

paket poslán. Směrovač neuplatňuje ACL pravidla na pakety vytvořené jím samotným (odchozí směr) [9]!

Bylo uvedeno že, ACL se skládá z pravidel. Při kontrole paketu se prochází jednotlivá pravidla v pořadí od prvního zadaného až po posledního zadaného. Pokud je v tomto procesu nalezeno pravidlo, je aplikováno na paket a následující pravidla již nejsou použita. Pokud není nalezeno žádné pravidlo, je použito implicitní pravidlo – zahození paketu [9].

8.2.2 Wildcard mask

Při kontrole paketů s pravidly ACL se porovnávají především statické hodnoty (např. hodnoty portů, typy ICMP zpráv). Náležitost paketu do určité sítě nebo přímo porovnání IP adres není přímočará záležitost. Používá se k tomu wildcard mask, která pracuje na podobném principu jako určování adresy sítě pomocí masky sítě.

Stejně jako bity s hodnotou 1 v masce sítě určují, které bity z IP adresy jsou adresou sítě, tak i bity s hodnotou 0 u wildcard mask určují, které bity v IP adresách paketu se budou porovnávat. Pokud bychom měli např. paket se zdrojovou adresou 192.168.1.8 a kontrolovali bychom tuto IP adresu pomocí zápisu v ACL 192.168.1.0 0.0.254.255, tak jsme dali pokyn porovnávacímu systému ACL, že úspěšné porovnání nastane, pokud porovnávaná IP adresa bude ve tvaru 192.168.1.x, kde x je libovolná hodnota. V našem případě by bylo porovnání úspěšné. Celou situaci můžeme ukázat pomocí bitového zápisu (červeně označené bity jsou bity pro porovnání):

IP adresa v paketu (klasický zápis):	192.168.1.8
IP adresa v paketu (binární zápis):	11000000.10101000.00000001.00001000
ACL IP adresa (klasický zápis):	192.168.1.0
ACL IP adresa (binární zápis):	11000000.10101000.00000001.00000000
Wildcard mask (klasický zápis):	0.0.254.255
Wildcard mask (binární zápis):	00000000.00000000.11111110.11111111

Jak vidíme, porovnání IP adresy je úspěšné.

8.3 Typy ACL

V této části popíšeme ACL, které jsou implementovány do NS i NSJS. Nebudeme zde rozebírat úroveň podpory jednotlivých typů (to popisuje následující kapitola), ale spíše se seznámíme s jejich možnostmi kontroly paketu a také si uvedeme tvar příkazů, kterými jsou konfigurovány na Cisco směrovačích.

V ukázkových tvarech příkazů bude ohraničení příkazů znaky „{ }“ znamenat povinný příkaz, „[ ]“ volitelný příkaz a znak „|“ představuje výraz *nebo*.

8.3.1 Standard ACL

Standard ACL poskytují pouze kontrolu podle zdrojové IP adresy zařízení nebo sítě. Příkaz pro konfiguraci tohoto typu ACL má následující tvar [9]:

```
access-list access-list-number {permit|deny|remark}
source [source-wildcard] [log]
```

8.3.2 Extended ACL

Tyto ACL poskytují větší škálu možností pro kontrolu paketů. Mezi nejdůležitější patří: zdrojová nebo cílová adresa zařízení nebo sítě, typ protokolu, zdrojový nebo cílový port. Pro různé typy protokolů jsou pak poskytovány rozšířené možnosti (např. pro ICMP typy zpráv nebo pro TCP rozpoznání navázaného spojení), které tvoří podtypy extended ACL. Tvar příkazu bez rozšířených možností vypadá následovně [9]:

```
access-list access-list-number {deny|permit|remark}
protocol source [source-wildcard] [operator [port]]
destination [destination-wildcard] [operator [port]]
```

8.3.2.1 Lock and Key (Dynamic ACLs)

Tento typ ACL pracuje na principu dynamického povolování komunikace s nějakým zařízením (typicky počítačem). Zařízení, které chce komunikovat skrze směrovač, je blokováno pomocí nějakého extended ACL. Až na základě ověření uživatele pomocí telnetu na směrovači je spojení přes směrovač povoleno. Dobu povolení komunikace lze řídit pomocí časového intervalu, absolutní hodnoty nebo lze použít možnost idle. Dynamic ACL nelze definovat pomocí Named ACL. Příkaz definující tento ACL má následující tvar [10]:

```
access-list access-list-number dynamic name [timeout time]
{permit|deny} [protocol] source [source-wildcard] destination
[destination-wildcard] [operator [port]]
```

8.3.2.2 Reflexive ACL

I když zde popsaný typ ACL nebude přímo Reflexive, dá se svým principem jako Reflexice použít. Extended ACL nabízejí parametr established, který v TCP paketech kontroluje bity ACK a RST a pomocí nich je schopný detekovat navázané spojení. Toho se, stejně jako v případě použití Reflexive ACL, dá využít k tomu, že směrovači se povolí přijímat jen započaté (navázané) spojení. Příkaz pro konfiguraci popsaného typu ACL je ve tvaru [11]:

```
access-list access-list-number {deny|permit|remark}
protocol source [source-wildcard] [operator [port]]
destination [destination-wildcard] [operator [port]]
[established]
```

8.3.2.3 Další typy ACL

Další typy zde uvedených ACL již nejsou součástí implementace v NS a NSJS. Jsou to:

- Time-Based ACL - ke klasickým extended ACL je přidána možnost definovat kontrolu paketů na základě časových informací, kterými jsou periodicky se opakující časový interval ve dnech týdne nebo absolutní hodnota vyjadřující délku trvání.
- Context-Based Access Control – ke své činnosti potřebují Cisco IOS Firewall feature set. Principem jsou to stavové paketové filtry pro TCP a UDP protokoly.
- Authentication Proxy - ke své činnosti potřebují Cisco IOS Firewall feature set. Principem mohou připomínat Dynamic ACL. Zde se však ověřování uživatelů děje pomocí proxy serveru. Pro autentizaci uživatelů lze také využít TACACS+ nebo RADIUS serveru.
- Turbo ACL – jsou vyvinuty pouze pro nejvyšší řady Cisco zařízení (např. řada produktů 7200 nebo 7500) . Představují urychlení zpracování ACL.
- Distributed Time-Based ACL – principem jsou naprosto stejné jako Time-Based ACL, řeší však problém Time-Based ACL na směrovačích VPN-enabled 7500 series s použitím line card.
- Receive ACL – využívá se na Cisco routerech řady 12000, kde chrání směrovací gigabitový procesor před přílišnou zátěží.
- Infrastructure Protection ACL – ochraňuje před útoky pomocí autorizované komunikace.
- Transit ACL – další ACL, které pomáhá řešit bezpečnost v sítí tím, že komunikace je povolena až explicitním zadáním povolení pro požadovaný přenos [10].

8.3.3 Named ACL

Named ACL představují zjednodušení ve srovnání s číselným označeným standard nebo extended ACL. Je totiž možné je znakově pojmenovat. Dalším zpřehledněním je možnost zadávat více ACL v jednom bloku. Příkaz pro vytváření named ACL je ve tvaru:

```
ip access-list {extended|standard} name
```

Jednotlivá pravidla pak konfigurujeme ve stejném tvaru jak pro standard, tak i extended ACL, přičemž s příkazem začneme až od části *{permit|deny|remark}* [9].

8.3.4 Aktivování ACL na rozhraní

Nyní ukážeme, jakým způsobem se aktivují jednotlivé ACL na rozhraní. Po definici ACL máme k dispozici předpis pro jeden typ protokolu, podle kterého můžeme provádět kontrolu paketů tohoto typu protokolu. Abychom ACL použili pro kontrolu paketů, musíme ACL v konfiguraci nějakého rozhraní přiřadit k aktivnímu protokolu (definovaném v ACL) a směru komunikace. Příkaz má pak následující tvar [9]:

```
ip access-group {number|name} {in|out}
```

8.4 Shrnutí

V této kapitole jsme se seznámili s IP ACL. V úvodu jsme je zařadili do oblasti bezpečnosti sítí a vyhodnotili jsme, že svým principem činnosti jsou to firewally typu paketových filtrů.

Dále jsme se blíže seznámili s ACL. Uvedli jsme, že ACL se skládá z jednotlivých pravidel, podle kterých porovnává jednotlivé pakety. Mezi nejčastěji porovnávané hodnoty patří zdrojová a cílová adresa a port a také typ protokolu. Poté jsme se seznámili s možnostmi přiřazení ACL podle rozhraní, směru komunikace a typu protokolu. Také jsme ukázali princip vyhodnocování paketů pomocí ACL od prvního kontaktu se směrovačem až po odeslání paketu do sítě. Rovněž jsme vysvětlili použití wildcard masky při porovnávání IP adres.

V poslední části této kapitoly jsme se seznámili s typy ACL. Základní rozřazení jsou standard a extended ACL. Kde standard ACL kontrolují pakety pouze podle zdrojové IP adresy, kdežto extended ACL poskytují rozsáhlé možnoti kontroly (základní oblasti inspekce jsou zdrojová a cílová adresa a port, typ protokolu). Pomocí rozšířených extended ACL pak můžeme kontrolovat pakety např. pomocí časových měřítek, autentizace uživatele nebo detekce navázaného spojení. Také jsme se zmínili o Named ACL, které svým principem nepřinášejí nic nového, ale spíše poskytují přehlednost v konfiguraci ACL a možnost přiřadit jednotlivým sadám ACL jméno oproti standardnímu číselnému označení. Závěrem této části jsme ukázali způsob aktivace ACL na rozhraní směrovače.

9 Implementace ACL do NS a NSJS

V této kapitole se budeme zabývat způsobem implementace ACL do NS i NSJS. V první části představíme problémy a některé neznámé, jež musíme v prvé řadě vyřešit, abychom mohli rozšířit NS i NSJS o ACL.

V další části rozebereme příkazy, které tvoří pojítko mezi konfiguracemi ACL na Cisco směrovačích a v NS (*access-list*, *ip-access-group*).

Poté uvedeme úrovně podpory pro jednotlivé typy implementovaných ACL. Seznámíme se s principem kontroly paktů v NS a srovnáme ho s principem používaným na Cisco směrovačích.

V poslední části ukážeme příklady použití příkazů v NS, které poskytují uživateli přehled nad výskytem a funkcí ACL v modelu sítě.

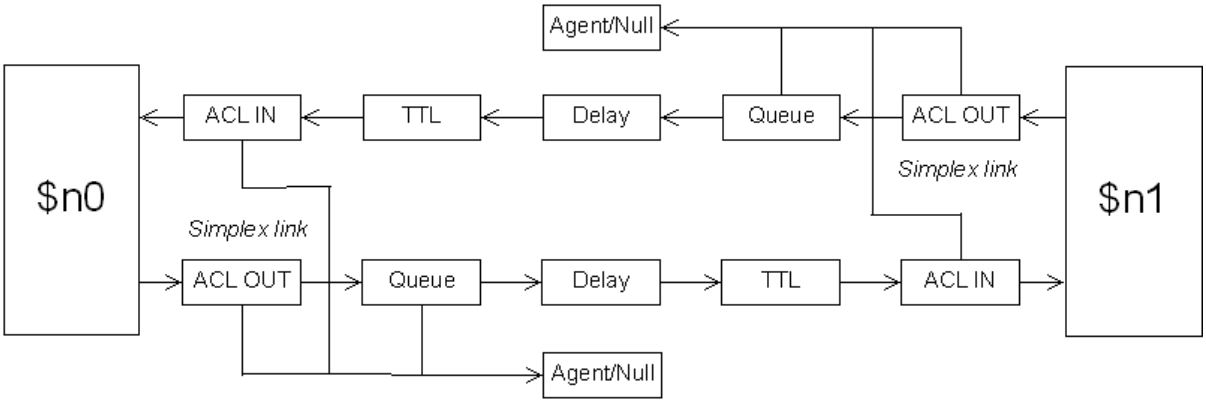
9.1 Návrh

V předcházející kapitole jsme se teoreticky seznámili s ACL. Před započetím implementace musíme vyřešit některé neznámé, jež by bránily v rozšíření NS či NSJS o používaní ACL:

- Kde vytvoříme nový prvek pro ACL na cestě paketu do/z směrovače, tak aby realizace odpovídala reálnému stavu?
- Jakým způsobem se budou pakety zahazovat nebo přeposílat?
- Kde budou uloženy informace a v jakém formátu o definovaných ACL?
- Jakým způsobem se budou aktivovat ACL na rozhraních, které v NS neexistují?
- Jakým způsobem je v NS implementován paket a jaké informace jsou použitelné pro kontrolu s ACL? Jakým způsobem se bude detekovat typ paketu?
- Jakým způsobem bude možné aplikovat ACL na protokoly již implementované v NS?
- Problém adresace podle ID (pouze NS), ale potřebná je IPv4.
- Jaká bude příkazová sada pro ovládání ACL v NS i NSJS?

9.1.1 ACL jako nový prvek v cestě paketu

Z předchozího textu již může být jasné, že NS nemá žádnou implementaci rozhraní v uzlu. Proto se např. fronty (Queue) řeší až na úrovni linek a ne na rozhraní, jak by se mohlo očekávat. Pro simulační procesy je však toto řešení naprosto shodné s reálným provedením. Stejně tak budeme řešit i přidání nového bloku, který bude provádět inspekci paketů. Situaci názorně ukazuje následující obrázek 9.1.



Obrázek 9.1: ACL vytvořené na lince

Na obrázku 9.1 můžeme vidět dvě simplex linky, které propojují dva uzly $n0 a $n1. Ve srovnání s normálním provedením (viz obrázek 3.2) přibyly pro každou linku dva nové prvky ACL IN a ACL OUT, které budou řešit kontrolu příchozích (ACL IN) a odchozích (ACL OUT) paketů z uzlů. Pakety, zahazované bloky ACL IN, nebudou posílány do bloků Agent/Null na svých simplex linkách, ale budou posílány do sousedního bloku Agent/Null. Kdybychom tak neučinili, zahazované pakety by se graficky zobrazovaly na opačném uzlu, než by ve skutečnosti zahazovány byly. Nevýhodou tohoto řešení je, že v případě záznamu toku paketů do výstupních souborů simulace se bude zahazovaný paket tvářit, jako by byl zahozen na sousední simplex lince, i když tomu tak není. Pokud by nám tato skutečnost vadila, můžeme explicitní přenastavení spojení bloků ACL IN se sousedními bloky Agent/Null změnit zpět na očekávaný spoj.

Stejně jako bloky Queue nebo Delay se i bloky ACL IN/OUT budou vytvářet automaticky při definování linek mezi uzly. Vzhledem k tomu, že ACL IN/OUT budou z programového hlediska stejné, musíme je od sebe nějak odlišit. To se děje již při vytvářecím procesu, kde definujeme typ prvku ACL. Další potřebnou součástí je, aby každý blok ACL měl znalost o svém zdrojovém a cílovém uzlu. Tím jsme schopni adresovat simplex linku, uzel, kde budou uloženy informace s ACL pravidly, a také můžeme zjistit přesnou polohu paketu v topologii sítě. Zdrojový a cílový uzel pro ACL se volí podle orientace simplex linky. Uzel, kde linka začíná, je pro ACL OUT i ACL IN prvky na stejné simplex lince zdrojovým uzlem. Uzel, kde simplex linka končí, je potom cílovým uzlem jak pro ACL OUT, tak i ACL IN na stejné simplex lince.

Pro takto definované dva nové prvky na lince můžeme využit již vytvořeného prvku Agent/Null, který obstarává zahozené pakety. Tedy v případě zahození paketu v jakémkoliv bloku ACL IN/OUT dojde k poslání paketu do bloku Agent/Null. V případě přeposlání paketu se paket pošle do následujícího prvku v cestě (Queue pro ACL OUT, uzel pro ACL IN). Vlivem využití prvku Agent/Null je vidět i zahazování paketu v prostředí NAM.

9.1.2 Datová struktura pro ACL

Jednotlivé ACL i jejich pravidla budeme ukládat do dynamického pole, které bude součástí každého uzlu. Pole bude v následujícím formátu:

```
$acl[$i]
        [number]
        [type]
        [used]
        [srule][$j]
                [ip_dec]
                [ip_bin]
                [mask_dec]
                [mask_bin]
                [wild_dec]
                [wild_bin]
                [action]
                [row]
                [used]
        [erule][$j]
                [src_ip_dec]
                [src_ip_bin]
                [src_mask_dec]
                [src_mask_bin]
                [src_wild_dec]
                [src_wild_bin]
                [src_port_from]
                [src_port_to]
                [src_port_exclude]
                [dst_ip_dec]
                [dst_ip_bin]
                [dst_mask_dec]
                [dst_mask_bin]
                [dst_wild_dec]
                [dst_wild_bin]
                [dst_port_from]
                [dst_port_to]
                [dst_port_exclude]
                [protocol]
                [icmp_type]
                [icmp_code]
                [action]
                [established]
                [row]
                [used]
```

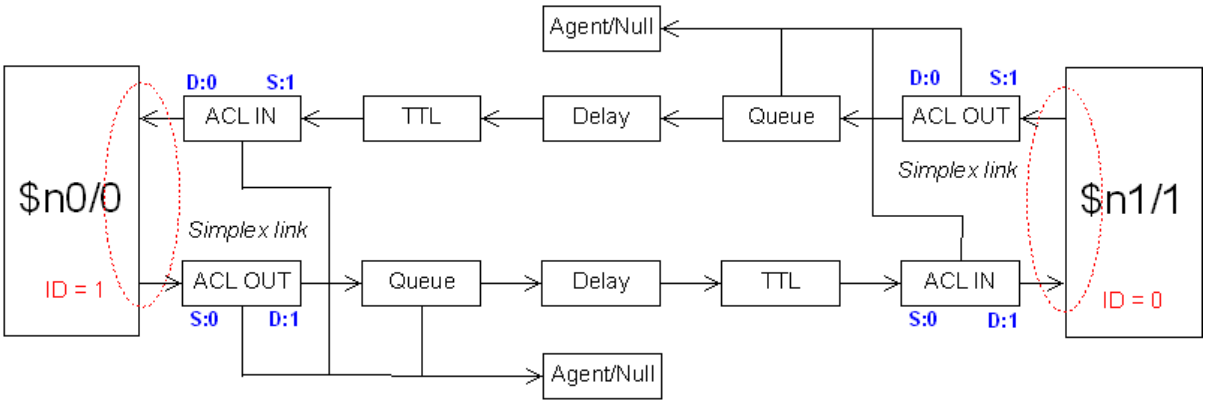
Prvek [number] představuje číselné i jmenné označení ACL. [type] určuje, zdali se jedná o extended nebo standard ACL (pokud bychom použili Named ACL, tak pouze z názvu ACL není typ jasný). [used] v první úrovni je počítadlo použití defaultního pravidla deny any.

Dále se pole větví pro uložení informací o standard ACL pravidlech [srule] a extended ACL pravidlech [erule]. Pro oba dva typy platí, že [action] obsahuje informaci o povolení nebo zakázání paketu, [row] v sobě ukládá celý řetězec definující jedno pravidlo a [used] je počítadlo použití jednotlivého pravidla.

Ostatní prvky již svým názvem určují, co bude jejich obsahem, proto je dále nemusíme rozebírat.

9.1.3 Aktivace ACL na rozhraní

V NS neexistují rozhraní. Proto, abychom mohli nějakým způsobem určit příslušnost určitých ACL k určitému směru komunikace v uzlu, musíme vytvořit virtuální rozhraní, které bude definované dynamickým polem v každém uzlu. Pro snažší pochopení slouží následující obrázek 9.2:



Obrázek 9.2: Virtuální rozhraní

Obrázek 9.2 je rozšířeným obrázkem 9.1, kde červené čárkované vertikální elipsy představují rozhraní směrovače a červeně napsaný text (např. ID = 1) jejich identifikaci. Identifikace je založena na principu ID přímého souseda rozhraní (stejně jako u IPv4 adresace). U bloků ACL IN/OUT jsou modrým písmem znázorněny hodnoty pro zdrojový (S:a) a cílový (D:b) uzel, kde *a,b* jsou ID nějakého uzlu. Označení uzlů $nx/y pak představuje označení uzlu (proměnné - $nx) a hodnota y je ID uzlu.

Pokud zvolíme takovou strategii označení jednotlivých součástí, jsme schopni z každého bloku ACL IN/OUT adresovat příslušné aktivované ACL pro určité rozhraní.

Použití ukážeme na příkladu, kde budeme chtít adresovat ACL pro příchozí i odchozí pakety pro uzel $n0 a rozhraní ID = 1. Z bloku ACL IN (D:0, S:1) pomocí znalosti ID cílového uzlu můžeme přistoupit k uzlu a tím i ke všem definovaným ACL. Pomocí znalosti ID zdrojového uzlu jsme schopni adresovat rozhraní a zjistit, zdali je aktivované nějaké ACL pro toto rozhraní a směr komunikace. Pro blok ACL OUT (S:0, D:1) je situace podobná. Z hodnoty ID zdrojového uzlu můžeme adresovat uzel $n0 a z hodnoty ID cílového uzlu příslušné rozhraní.

Dynamické pole, v němž jsou uloženy informace o aktivovaných ACL pro určitá rozhraní, má následující formát:

```
$acl_bind[$i]
        [interface_id]
        [acl_binded][$j]
                [number]
                [direction]
```

Prvek [number] má stejný význam jako v dynamickém poli $acl. Představuje identifikátor pro spojení těchto dvou polí. Význam zbylých dvou prvků ([interface_id] a [direction]) charakterizuje jejich název.

9.1.4 Dostupné informace z paketu pro ACL

Jak již bylo v této práci uvedeno, paket je pro NS přenositelem informace a také oživuje celý model sítě a simulaci. Nyní zaměříme na zjištění, zdali paket obsahuje veškeré potřebné informace pro ACL a pokud ne, pokusíme informace do paketu doplnit.

9.1.4.1 Zdrojová a cílová IP adresa

Pro NS musíme veškerou vytvářenou komunikaci doplnit o informace o zdrojové a cílové IP adrese, protože adresace probíhá na základě ID uzlů. V NSJS je již adresace pomocí IP adres implementována a je součástí paketu.

9.1.4.2 Zdrojový a cílový port

Jak v NS, tak i v NSJS jsou porty implementovány, ale představují spíše interní rozřazení pro port classifier v uzlu a nelze tyto hodnoty ve všech případech kombinovat. Proto pokud chceme pro ACL využívat i kontrolu portů, musíme jejich hodnoty explicitně zadat do nových proměnných v paketu.

9.1.4.3 Typ protokolu

V NS i NSJS spojuje paket všechny informace z potřebných vrstev ISO/OSI modelu do jednoho celku. Proto se typ paketu řídí podle typu aplikace, kterou simuluje. Zde je však pro použití s ACL problém. Pokud totiž chceme např. vytvořit a kontrolovat FTP komunikaci v ACL, kontrolujeme typ protokolu TCP a standardně port 21 popřípadě 20. V NS však vytváření FTP komunikace probíhá tak, že nejdříve vytvoříme agenta, který simuluje TCP protokol, na který se naváže FTP aplikace. Ve výsledku je však typ paketu FTP a ne TCP, jak by bylo potřeba. Proto musíme v paketu vytvořit další proměnnou, která v sobě uchovává typ agenta představující typ protokolu.

9.1.4.4 Podpora ACL pro dostupné protokoly v NS

NS je svým principem simulátor IP sítí. Proto můžeme na všechny jeho protokoly použít standard ACL. U extended ACL, kde definujeme i typ protokolu, musí typ protokolu odpovídat názvu protokolu agenta, který v NS komunikaci zprostředkovává.

9.1.4.5 Detekce navázaného spojení

Při použití agenta typu TCP/FullTcp jsme schopni z paketu zjistit pouze informaci o ACK. Pokud má ACK hodnotu 0, jedná se o paket, který spojení navazuje. Pokud je hodnota větší než 0, jedná se o paket s již navázaným spojením. Na základě této hodnoty můžeme extended ACL kontrolovat paket v závislosti na parametru established.

9.1.4.6 Typy ICMP zpráv

ICMP protokol není v NS ani NSJS implementován.

9.1.5 Příkazová sada pro konfiguraci ACL

Vzhledem k tomu, že vyhodnocování ACL představuje další zátěž pro běh simulace, i když nejsou definovány žádné ACL (musí se to totiž v každém bloku ACL IN/OUT pro každý paket prověřit), vytvoříme pro NS možnost globálně zapnout používaní ACL nebo globálně vypnout (defaultní stav). Příklad příkazu pro povolení ACL:

```
$ns enable-acl
```

Při použití číselného označení standard i extended ACL jsme schopni jednotlivá pravidla konfigurovat na jednom řádku. V tomto případě pro jednoduchost převodu příkazu z konfiguračního souboru bude mít příkaz stejný syntax jako v Cisco směrovačích. Například:

```
$n0 access-list "150 permit udp 192.168.1.0 0.0.0.255 host 192.168.2.2 range 5 500"
```

Pokud však pro definici ACL budeme používat Named ACL, musíme se vypořádat s několikařádkovým příkazem. To vyřešíme tak, že definici Named ACL přidáme ke každému řádku definující toto ACL. Tím budeme schopni Named ACL konfigurovat v NS pomocí jednoho řádku. Příklad příkazu je následující:

```
$n0 access-list "extended test permit tcp host 192.168.3.2 host 192.168.4.1"
$n0 access-list "extended test permit tcp host 192.168.3.3 host 192.168.4.1"
```

Musíme také zajistit aktivování nějakého ACL na rozhraní uzlu. To bude provádět příkaz:

```
$n0 ip-access-group [$n2 id] test out
```

Dále také bude možné zobrazovat namapované ACL na rozhraní pro jednotlivé uzly. To bude řešit příkaz s příkladem zápisu:

```
$n2 show-acl-bind
```

Také budeme mít možnost zobrazit jednotlivé ACL konfigurované v uzlech a to ve dvou režimech. V prvním se zobrazí kompletní výpis jednotlivých pravidel (všechny prvky obsažené v poli $acl) pro jednotlivé ACL ve druhé variantě se zobrazí pouze počet použití pravidla (prvek [used]) a obsah prvku [row]. Pro kompletní výpis použijeme např. příkaz:

```
$n2 show-acl full
```

A pro zkrácenou variantu může příkaz vypadat následovně:

```
$n2 show-acl short
```

Abychom měli přehled o používání ACL na úrovni jednotlivých pravidel, vytvoříme možnost, která bude zobrazovat každé kladné porovnání paketu s libovolným pravidlem jakéhokoliv ACL pro jednotlivé uzly i s informací o simulačním čase porovnání. Příklad příkazu, který aktivuje tuto funkci:

```
$n1 debug-acl
```

9.2 Zpracování příkazu access-list

V této části popíšeme způsob zpracování příkazu access-list a jemu předcházející procesy.

9.2.1 Volitelné parametry

Každý jeden řádek příkazu access-list použitého v NS je obsahem jednoho záznamu v dynamickém poli $acl. Musíme však vyřešit situace, kdy nejsou zadány veškeré parametry obsažené v poli. U standard ACL to muže být např. wildcard maska, u extended ACL je těchto variant podstatně více, např. nezadání zdrojového nebo cílového portu nebo parametru established.

Proto k celému vyhodnocení přistoupíme tak, že těm prvkům, o kterých víme, že nemusí být zadány, již dopředu vložíme nějaké hodnoty, které nijak neovlivní definici pravidla ACL, ale přitom umožní korektní kontrolu paketu. Pokud budou existovat nějaké volitelné parametry, přepíšeme jimi výchozí hodnoty.

Výchozí hodnoty ukážeme přímo v poli $acl, kde popíšeme jen prvky, kterých se to týká:

```
$acl[$i]
        [used] = 0
        [srule][$j]
                [row] = "příkaz obsažený v uvozovkách"
                [used] = 0
        [erule][$j]
                [src_port_from] = -1
                [src_port_to] = 65535
                [src_port_exclude] = 65536
                [dst_port_from] = -1
                [dst_port_to] = 65535
                [dst_port_exclude] = 65536
                [icmp_type] = -1
                [icmp_code] = -1
                [established] = false
                [row] = "příkaz obsažený v uvozovkách"
                [used] = 0
```

9.2.2 Převody parametrů

Další úpravy, které musíme provést při zpracování příkazu access-list, jsou různé převody především textových řetězců na číselné hodnoty nebo IP adresy, aby je bylo možné rozumně porovnávat. V následujících tabulkách ukážeme, kterých parametrů se to týká a také jakých hodnot pak nabývají ovlivněné prvky v poli $acl.

9.2.2.1 IP adresace

Parametr v příkazu	IP adresa	Wildcard mask	Network mask
samostatně zadaná IP adresa x.x.x.x	x.x.x.x	0.0.0.0	255.255.255.255
IP adresa x.x.x.x + wildcard mask y.y.y.y	x.x.x.x	y.y.y.y	z.z.z.z kde se jednotlivé části vypočítají z wildcard mask
příkaz any	0.0.0.0	255.255.255.255	0.0.0.0
příkaz host + IP adresa x.x.x.x	x.x.x.x	0.0.0.0	255.255.255.255

Tabulka 9.1: Získání parametrů pro IP adresaci

9.2.2.2 Porty

Samotné porty mohou být zadány číslem nebo názvem. Pokud jsou zadány názvem, jejich hodnoty jsou převedeny na čísla. Nebude-li název portu podporován, je číslo portu nastaveno na hodnotu -1. Tím v podstatě znemožníme úspěšné porovnání s jakýmkoliv paketem a pravidlo se v podstatě neuplatní. Není však problém přidat další název pro hodnotu portu do zdrojových kódů. V následující tabulce 9.2 bude znak "~" představovat žádnou změnu v prvku.

Parametr v příkazu	**Dolní hranice rozsahu portů**	**Horní hranice rozsahu portů**	**Port nepatřící do intervalu**
eq + port	port	port	~
gt + port	port + 1	~	~
lt + port	~	port - 1	~
neq + port	~	~	port
range + port1 + port2	port1	port2	~

Tabulka 9.2: Získání hodnot portů

Důvod, proč v případě parametru gt přičítáme hodnotu 1 a u parametru lt odečítáme hodnotu 1, je ten, že výsledná příslušnost portu do nějakého pravidla v ACL je zjišťována podle výrazu:

```
acl_port_from <= paket_port and
paket_port <= acl_port_to and
paket_port != acl_port_exclude
```

9.2.2.3 ICMP zprávy

V definici ACL pravidla pro protokol ICMP můžeme přesně určit typ a kód ICMP zprávy. Máme k tomu tři možnosti. V první ICMP typ i kód charakterizuje nějaký název, v druhé můžeme číselně definovat pouze ICMP typ a třetí variantou je přesné číselné určení typu i kódu ICMP.

Pokud bude tedy typ i kód ICMP zprávy zadán textovou formou, převedeme jej na příslušné číselné hodnoty. Pokud ICMP zpráva bude definována textem, který charakterizuje pouze její typ (např. unreachable), změníme hodnotu jen v ICMP typu zprávy a v kódu necháme výchozí hodnotu -1, která charakterizuje veškeré kódy pro příslušný typ ICMP.

V případě zadání ICMP typu a kódu číselnou formou je situace jasná.

9.2.3 Postup při zpracování příkazu access-list

Zpracování příkazu access-list si můžeme rozdělit na dvě části. V první zjišťujeme typ ACL (standard, extended) a jeho číslo popřípadě název při použití Named ACL. Ve druhé části podle zjištěné příslušnosti k typu ACL analyzujeme zbylý řetězec vyjadřující definici parvidla.

Při analýze definice pravidla vycházíme ze znalosti možných variant příkazu. Také můžeme rozdělit příkaz na slova, která odděluje znak " " (mezera). Tato slova pak tvoří definici parametru

nebo jeho hodnotu. Pravidla tedy analyzujeme slovo za slovem a na základě zpracované hodnoty vytváříme interní reprezentaci pravidla, které pak uložíme do pole $acl.

9.3 Zpracování příkazu ip-access-group

Příkaz ip-access-group je ve srovnání s příkazem access-list velmi jednoduchý, protože má pevný počet parametrů (ID rozhraní, identifikaci ACL a směr, ve kterém se budou na rozhraní kontrolovat pakety). Všechny tyto tři parametry můžeme přímo uložit do pole $acl_bind.

9.4 Úroveň podpory implementovaných ACL

V následujících podkapitolách rozebereme jednotlivé implementované typy ACL a popíšeme jejich úroveň podpory. Úroveň podpory jednotlivých ACL je jak pro NS tak i NSJS stejná.

Pro všechny typy ACL platí omezení nemožnosti používání přímého názvu zařízení (hostname) místo IP adresy. V NS ani NSJS neexistuje mechanismus, kterým by takováto adresace byla možná. Pokud se při zpracování ACL pravidla na takovouto definici narazí, není toto pravidlo použito.

Dále pro všechny typy ACL platí, že se nijak nezpracovávají poznámky (příkaz remarak).

Pro názornost budeme definice jednotlivých typů ACL rozkládat do bloků, které budou tvořit celý příkaz používaný na Cisco směrovačích. Text uzavřený v závorkách “[ ]“ bude mít pouze významový charakter. Text, který nebude v těchto závorkách uzavřen, bude přímo příkazem v definici ACL. Znak “|“ bude představovat význam *nebo*.

9.4.1 Standard ACL

```
access-list [number] [action] [source] [optional parameters]
```

Kompletní příkazová i funční podpora je pro bloky [number], [action], [source] (v bloku [action] není zpracováván příkaz remark). Naopak žádná podpora není pro blok [optional parameters].

9.4.2 Extended ACL

Pro [protocol] TCP a UDP:

```
access-list [number] [action] [protocol] [source] [source port]
[destination] [destination port] [optional parameters]
```

Kompletně jsou podporovány bloky [number], [action], [protocol], [source], [source port], [destination], [destination port]. Vůbec není podporován blok [optional parameters].

Pro [protocol] ICMP:

```
access-list [number] [action] [protocol] [source] [destination]
[ICMP message] [optional parameters]
```

Kompletně jsou podporovány bloky [number], [action], [protocol], [source], [destination], [ICMP message]. Vůbec není podporován blok [optional parameters]. I když není v NS ani NSJS protokol ICMP implementován, je vytvořená podpora pro ACL schopná tento typ pasivně zpracovat. Případný zájemce, který by chtěl ICMP protokol vytvořit, může lehce svou práci spojit s ACL.

Pro [protocol] IP, AHP, EIGRP, ESP, GRE, IGMP, IPINIP, NOS, OSPF, PCP, PIM:

```
access-list [number] [action] [protocol] [source] [destination]
[optional parameters]
```

Kompletně jsou podporovány bloky [number], [action], [protocol], [source], [destination]. Vůbec není podporován blok [optional parameters]. I zde platí podobné, co pro protocol ICMP. Protokoly AHP až PIM nejsou v NS ani NSJS vytvořeny, ale zpracování ACL je dokáže pasivně zpracovat pro případné budoucí použití.

Dále rozebereme podporu pro dva typy extended ACL. Pro oba dva platí vše, co jsme uvedli pro protokoly v této podkapitole.

9.4.2.1 Lock and Key (Dynamic ACL)

```
access-list [number] [dynamic definitions] [action] [protocol]
[src, src port, dst, dst port depend on type of protocol]
[optional parameters]
```

Tento typ ACL je implementován tak, že ignoruje blok [dynamic definitions] a tím tak v podstatě vytvoří klasický extended ACL. Abychom mohli využívat plně funkci tohoto ACL, museli bychom v NS vytvořit možnost zadaní reálného času pro start simulace. Sám běh simulace by pak určoval reálný čas, který by se mohl použít pro Dynamic ACL.

Kompletně jsou podporovány všechny bloky, kromě [dynamic definitions] (viz předcházející odstavec) a [optional parameters], který není podporovaný vůbec.

9.4.2.2 Reflexive ACL

```
access-list [number] [action] [protocol]
[src, src port, dst, dst port depend on type of protocol]
[[established] | [optional parameters]]
```

Pro tento typ ACL jsou podporovány všechny bloky příkazu až na [optional parameters], který není podporován vůbec.

9.4.3 Named ACL

```
ip access-list [type] [name]
      [action] …
```

Pro blok [type] je možné zpracovat příkazy standard a extended. Zbylé příkazy nejsou podporovány. Blok [name] má plnou podporu. Ale v bloku [action] jsou podporovány pouze příkazy deny a permit. V případě, že je v bloku [type] zjištěn nepodporovaný příkaz, celý víceřádkový příkaz se neuplatní. Podobně, pokud je v bloku [action] zjištěn nepodporovaný příkaz, definice pravidla se neuplatní.

9.5 Kontrola paketů ACL

Pokud jsou ACL povoleny, probíhá kontrola paketů v každém bloku ACL IN nebo ACL OUT na lince mezi uzly. Prověřuje se zde každý průchozí paket. Samotný princip kontroly je naprosto shodný s principem na Cisco směrovačích. To znamená, že nejdříve se hledá, zdali existuje nějaké ACL pro tento typ komunikace a směr. Pokud ne, je paket v bloku přeposlán dále. Pokud ano, začne se paket porovnávat s ACL.

Pro standard ACL se se porovnává pouze zdrojová adresa.

Pro extended se porovnává nejdříve zdrojová a cílová adresa, poté příslušnost k určitému protokolu, následně se kontrolují informace podle typu protokolu (např. pro TCP a UDP zdrojový a cílový port), pokud je pravidlo rozšířeným extended ACL např. Reflexive, jsou kontrolovány příslušné hodnoty.

Samotná kontrola probíhá po jednotlivých pravidlech v ACL, dokud se nenajde nějaké pravidlo, které svou definicí kompletně odpovídá informacím v kontrolovaném paketu. Pokud je takové pravidlo nalezeno, je paket podle pravidla přeposlán nebo zahozen. Další pravidla se již neprověřují. Pokud nebylo nalezeno žádné pravidlo, uplatní se defaultní akce – zahození paketu.

Samozřejmě se počítá s pakety vytvořenými směrovačem, které procházejí bloky ACL OUT tohoto směrovače. V tomto případě jsou pakety automaticky přeposlány.

9.6 Ukázky použití

V následujících podkapitolách ukážeme příklady výstupů příkazů, které jsou schopny zobrazovat pasivní i aktivní informace o ACL uživateli. Pro názornost každou ukázku doplníme o příkaz, který jednotlivé typy informací zobrazuje. Všechny výpisy jsou prováděny na jednom uzlu $n0, který má definovány ACL následovně:

```
$n0 access-list "extended cbr permit udp host 192.168.4.1 eq 1500 host 192.168.2.2 eq 500"
$n0 access-list "extended ftp deny tcp 192.168.3.0 0.0.0.255 192.168.4.0 0.0.0.255 eq 21"
$n0 ip-access-group [$n2 id] cbr out
$n0 ip-access-group [$n1 id] ftp out
```

9.6.1 Výpis aktivních ACL na rozhraní

```
$n0 show-acl-bind
```

```
~/ns
Administrator@oem-988747c24f6 ~/ns
$ ns32 ex4.tcl

* * * * * BINDED ACL for NODE: Node_0 * * * * *

interface id: 2
acl number: cbr
direction: out

interface id: 1
acl number: ftp
direction: out

Administrator@oem-988747c24f6 ~/ns
$
```

Obrázek 9.3: Výpis aktivních ACL na rozhraní

9.6.2 Zkrácený výpis ACL v uzlu

```
$n0 show-acl short
```

```
~/ns
Administrator@oem-988747c24f6 ~/ns
$ ns32 ex4.tcl

* * * * * ACL for NODE: Node_0 * * * * *

ACL number: cbr (extended)
Deny any (any) used: 0x

Row: extended cbr permit udp host 192.168.4.1 eq 1500 host 192.168.2.2 eq 500
Used: 20x

ACL number: ftp (extended)
Deny any (any) used: 0x

Row: extended ftp deny tcp 192.168.3.0 0.0.0.255 192.168.4.0 0.0.0.255 eq 21
Used: 1x

Administrator@oem-988747c24f6 ~/ns
$
```

Obrázek 9.4: Zkrácený výpis ACL v uzlu

9.6.3 Úplný výpis ACL v uzlu

```
$n0 show-acl full
```

```
~/ns
Administrator@oem-988747c24f6 ~/ns
$ ns32 ex4.tcl

* * * * * ACL for NODE: Node_0 * * * * *

ACL number: cbr (extended)
Deny any (any) used: 0x

action: permit
established: false
Protocol: udp
Src IP: 192.168.4.1 11000000101010000000010000000001
Src mask: 255.255.255.255 11111111111111111111111111111111
Src wild: 0.0.0.0 00000000000000000000000000000000
Src ports (from,to,exclude): 1500 1500 65536
Dst IP: 192.168.2.2 11000000101010000000001000000010
Dst mask: 255.255.255.255 11111111111111111111111111111111
Dst wild: 0.0.0.0 00000000000000000000000000000000
Dst ports (from,to,exclude): 500 500 65536
ICMP (type,code): -1 -1
Row: extended cbr permit udp host 192.168.4.1 eq 1500 host 192.168.2.2 eq 500
Used: 20x

ACL number: ftp (extended)
Deny any (any) used: 0x

action: deny
established: false
Protocol: tcp
Src IP: 192.168.3.0 11000000101010000000001100000000
Src mask: 255.255.255.0 11111111111111111111111100000000
Src wild: 0.0.0.255 00000000000000000000000011111111
Src ports (from,to,exclude): -1 65535 65536
Dst IP: 192.168.4.0 11000000101010000000010000000000
Dst mask: 255.255.255.0 11111111111111111111111100000000
Dst wild: 0.0.0.255 00000000000000000000000011111111
Dst ports (from,to,exclude): 21 21 65536
ICMP (type,code): -1 -1
Row: extended ftp deny tcp 192.168.3.0 0.0.0.255 192.168.4.0 0.0.0.255 eq 21
Used: 1x

Administrator@oem-988747c24f6 ~/ns
$
```

Obrázek 9.5: Úplný výpis ACL v uzlu

9.6.4 Sledování používání jednotlivých pravidel ACL

```
$n0 debug-acl
```

```
~/ns
Administrator@oem-988747c24f6 ~/ns
$ ns32 ex4.tcl
*** ACL debug. Node: Node_0 OUT-boud pkt id: 27, protocol: udp,
src IP: 192.168.4.1, dst IP: 192.168.2.2 was permited
by rule: extended cbr permit udp host 192.168.4.1 eq 1500 host 192.168.2.2 eq 500
at: 0.142240
*** ACL debug. Node: Node_0 OUT-boud pkt id: 28, protocol: udp,
src IP: 192.168.4.1, dst IP: 192.168.2.2 was permited
by rule: extended cbr permit udp host 192.168.4.1 eq 1500 host 192.168.2.2 eq 500
at: 0.159040
*** ACL debug. Node: Node_0 OUT-boud pkt id: 29, protocol: tcp,
src IP: 192.168.3.2, dst IP: 192.168.4.1 was rejected
by rule: extended ftp deny tcp 192.168.3.0 0.0.0.255 192.168.4.0 0.0.0.255 eq 21
at: 0.160213

Administrator@oem-988747c24f6 ~/ns
$
```

Obrázek 9.6: Sledování používání jednotlivých pravidel ACL

9.7 Shrnutí

V této části práce jsme se seznámili se způsobem implementace ACL do NS i NSJS. Na začátku jsme museli vyřešit několik problémů, které nám v řešení bránily. Mezi nejdůležitější patřilo: umístění bloku v cestě paketu, který by vykonával práci ACL, kde budou uloženy a v jakém formátu informace o ACL, vyřešení problému s IPv4 adresací, která neexistuje v NS, rozebrání možnosti používání paketu a informací v něm pro kontrolu s ACL.

Dále jsme ukázali rozbor příkazů *access-list* a *ip-access-group*, které jsou stěžejní pro konfiguraci ACL v NS i NSJS. Také jsme popsali úrovně podpory pro jednotlivé typy implementovaných ACL.

V předposlední části jsme definovali princip kontroly paketů pomocí ACL a srovnali jsme jej s principem na Cisco směrovačích. V poslední části jsme ukázali výstupy různých příkazů pro použití s ACL, které usnadňují uživateli práci s jejich konfigurací.

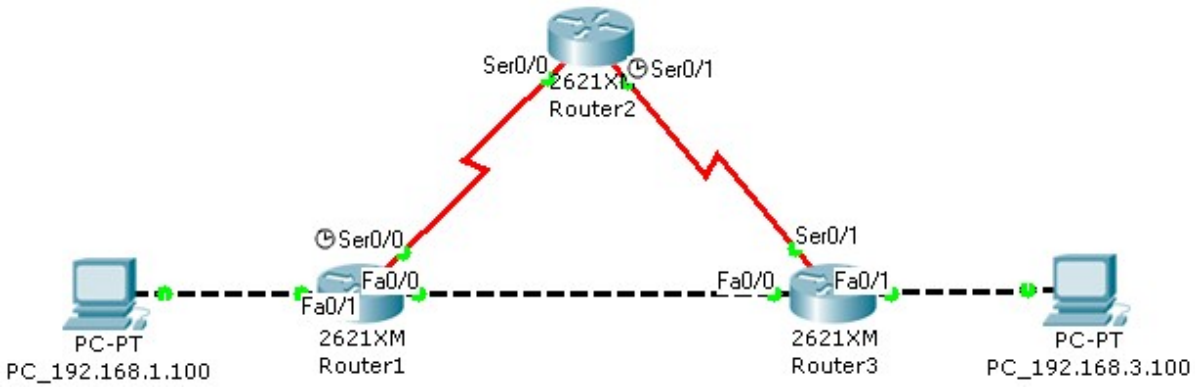
10 Ověření správnosti konfigurace ACL a směrování

V této kapitole prakticky nastíníme jednu z variant, jakým způsobem je možné provádět ověření správnosti konfigurace ACL pomocí převodního nástroje a NS i NSJS a tím také zjišťovat případné chyby nebo nedostatky v nastavení směrovačů. Vše ukážeme na jednoduchém zapojení.

V následujících částech této kapitoly rozebereme možnosti, jakým způsobem ověření konfigurace ACL i směrování provádět automaticky, a celou problematiku se pokusíme teoreticky zobecnit.

10.1 Schéma a popis zapojení

Na obrázku 10.1 je znázorněno schéma zapojení. Černé čárkované spoje mezi zařízeními jsou tvořeny pomocí UTP kříženého kabelu a červené spoje jsou realizovány sériovou linkou. Mezi směrovači Router1 a Router3 je nastaveno dynamické směrování pomocí protokolu RIP v1. Sériová linka zde simuluje záložní cestu v případě výpadku hlavní linky mezi směrovači Router1 a Router3. Směrování po této lince je řešeno statickým směrováním s explicitně zvýšenou administrative distance tak, aby v případě funkčnosti veškeré pakety procházely hlavní linkou.



Obrázek 10.1: Schéma zapojení

Pro odchozí směr rozhraní Fa0/0 směrovače Router1 je definován ACL s následujícím pravidlem:

```
access list 101 permit tcp 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255
```

Jednotlivé konfigurační soubory směrovačů a výsledný OTcl skript je obsažen v příloze této práce.

10.2 Ověření konfigurace

V popisovaném zapojení nás zajimá, zdali jsme provedli konfiguraci ACL správně tak, aby za všech případů toků paketů v síti se chovalo zabezpečení ACL podle našich představ. Podle výše uvedeného pravidla to znamená, že ze sítě 192.168.1.0 do sítě 192.168.3.0 chceme povolit jen komunikaci pomocí TCP protokolu. Ostatní komunikace není povolena. Samotný test provedeme tak, že po překladu konfiguračních souborů v NS doplníme UDP komunikaci s CBR a TCP komunikaci s FTP službou a budeme zkoumat chování sítě v případě, kdy hlavní linka mezi směrovači Router1 a Router3 bude, či nebude funkční.

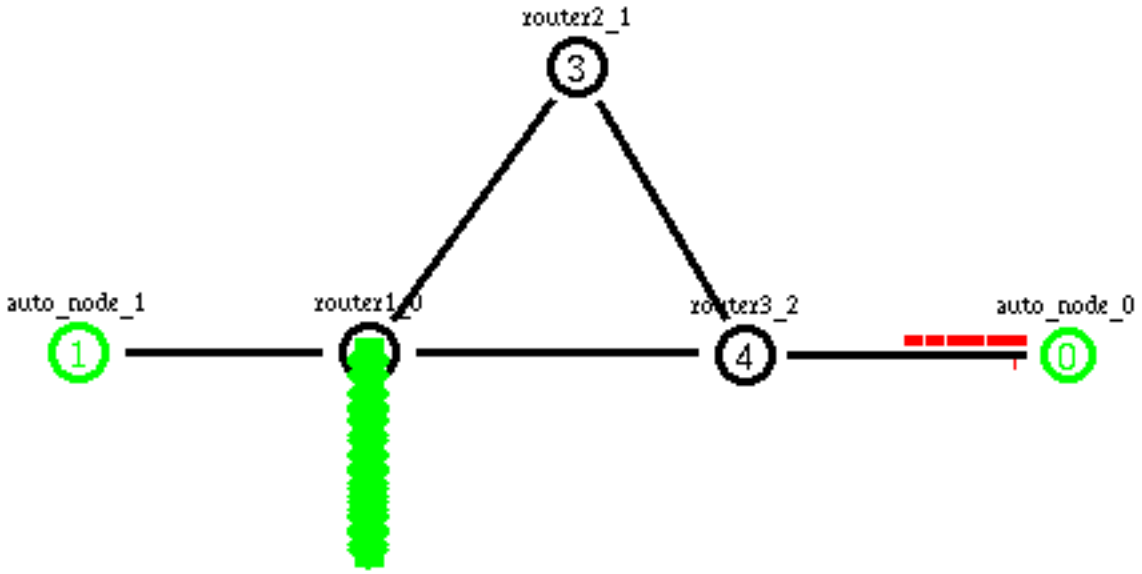
Nyní ukážeme výstupy ze simulátoru, podle kterých určíme správnost konfigurace ACL. V následujících obrázcích bude zeleně vyznačena UDP komunikace a červeně TCP komunikace. Nefunkční linka je znázorněna červenou barvou linky (oproti černé). Auto_node_1 představuje počítač PC_192.168.1.100 a auto_node_0 je počítač PC_192.168.3.100 na obrázku 10.1.

10.2.1 Funkční hlavní linka

Pomocí příkazu *$router1_0 debug-acl* jsme schopni zjistit činnost ACL pro každý kladně ověřený paket (výňatek z výpisu):

```
*** ACL debug. Node: router1_0 OUT-boud pkt id: 12, protocol: tcp,
src IP: 192.168.1.2, dst IP: 192.168.3.2 was permited
by rule: 101 permit tcp 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255
at: 0.102003
*** ACL debug. Node: router1_0 OUT-boud pkt id: 13, protocol: udp,
src IP: 192.168.1.2, dst IP: 192.168.3.2 was rejected
by rule: deny any any
at: 0.102019
```

Dále pomocí programu NAM můžeme graficky zobrazit průběh simulace:



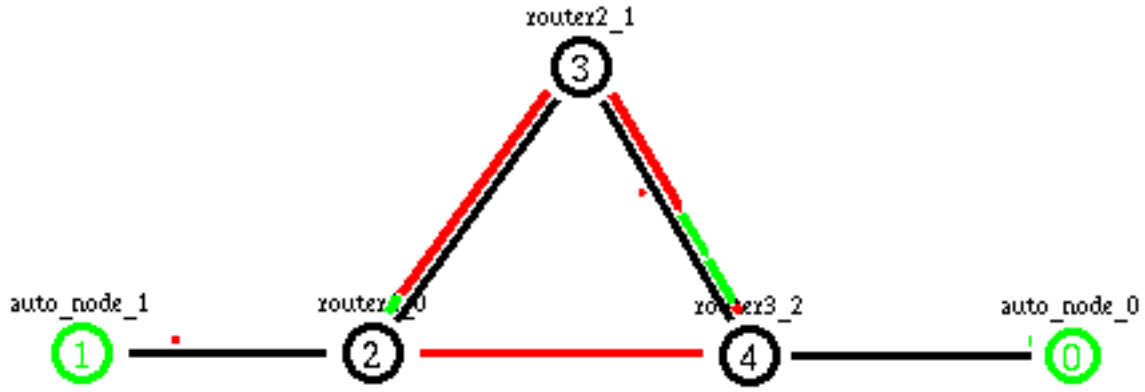
Obrázek 10.2: Tok paketů při funkční hlavní lince

Z výpisu používání ACL vidíme, že komunikace je filtrována tak, jak požadujeme (TCP pakety jsou propouštěny, ale UDP jsou zahazovány). I z výstupu programu NAM (obrázek 10.2) je korektní funkčnost zřejmá.

10.2.2 Nefunkční hlavní linka

Pomocí příkazu *$router1_0 debug-acl* se nám však v tomto případě nic nevypíše, protože komunikace neprochází rozhraním, kde je nakonfigurované ACL.

Grafické zobrazení pomocí programu NAM je následující:



Obrázek 10.3: Tok paketů při nefunkční hlavní lince

Na základě žádných výpisů z ACL zjišťujeme, že ACL se neuplatňuje. Pomocí programu NAM přímo vidíme, že veškerá komunikace je propouštěna až do sítě 192.168.3.0 - chyba.

10.2.3 Vyhodnocení

Z předcházejících dvou kapitol je zřejmé, že konfigurace ACL nebyla provedena korektně. Řešením této situace je umístění ACL na rozhraní Fa0/1 směrovače Router1 v příchozím směru. Takto zajistíme uplatnění ACL i v případě, že je hlavní linka nefunkční.

Prezentovaný příklad je jednoduchou ukázkou možného řešení při testování konfigurací. Pro větší topologie sítí je tento manuální přístup vhodný pouze v případě kontroly malého počtu změn v síti, protože veškeré změny a následné provedení simulace musí uživatel provést sám. V dalších podkapitolách bude tento problém více rozebrán.

10.3 Automatické ověřování konfigurace ACL

Automatické ověření konfigurace by mělo fungovat tak, že i neznalému uživateli NS bude stačit použití převodního nástroje, a dále jen spuštění simulačního skriptu, který přímo do konzole nebo souboru zobrazí potřebné informace a pro vizualizaci může pomoci i program NAM. Vzhledem k tomu, že generátorem modelu sítě i simulace je převodní nástroj, veškeré požadavky pro vytvoření

kontroly by byly kladeny na něj. Pro řešení této problematiky můžeme vycházet z následujících znalostí o převodním nástroji:

- V poli $a_routers, $a_super_node, $a_auto_node jsou uloženy veškeré podporované informace o simulovaných směrovačích a uzlech.
- Převodní nástroj vytváří topologii sítě modelu, to znamená, že existuje nějaký mechanizmus, kterým můžeme zjišťovat potřebné informace z topologie, např. příslušnost uzlu do nějaké sítě, vyhledat sousedy uzlu a další.
- Převodní nástroj si uchovává informace o jednotlivých linkách, tj. zná uzly, které linka spojuje.
- Pokud bychom rozšířili převodní nástroj o rozbor ACL příkazů, které definují jednotlivá pravidla (*access-list, ip access-list*), mohli bychom například zjišťovat počáteční uzel (síť) a koncový uzel (síť) nebo také typ protokolu, který ACL kontroluje, a další parametry.

Správná funkčnost ACL závisí zajisté na korektní definici jednotlivých pravidel i jejich pořadí, ale také i na aktivování ACL pro správný směr komunikace na správném rozhraní. Dále také ACL mohou ovlivňovat změny v topologii sítě v průběhu času (např. nefukčnost některých linek nebo chyba v konfiguraci směrování). V následujících podkapitolách si popíšemé možná řešení kontroly v závislosti na uvedených možnostech, kde ACL nemusí vždy pracovat korektně.

10.3.1 Ověření zabezpečení ACL

V případě, že máme dostupné jednotlivé informace z pravidel ACL, jsme schopni vytvořit testovou komunikaci. Způsob, jakým toho dosáhneme, si popíšeme na příkladu.

Nějaký směrovač v topologii sítě má definovaný ACL pro nějaké rozhraní v jistém směru. ACL obsahuje jedno pravidlo:

```
access list 101 permit tcp 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255
```

Z uvedeného pravidla můžeme zjistit, že se mají zahazovat veškeré pakety kromě paketů protokolu TCP ve směru ze sítě 192.168.1.0 do sítě 192.168.3.0. Na základě těchto informací vyhledáme v topologii sítě, zdali existuje nějaký uzel ve zmíněných sítích a pokud ano, definujeme komunikaci, která odpovídá zjištěným parametrům. Další komunikaci vytvoříme s použitím jiného typu protokolu, abychom zachovali kompletní, ale opačný význam pravidla. Obě dvě komunikace spustíme nejlépe paralelně a pro větší rychlost ověřování se budeme snažit každou komunikaci testovat v co nejmenším čase. Pomocí nějaké funkce přímo v NS by se pak mohly vypisovat úspěšné či neúspěšné průchody paketů do vybraného uzlu v síti 192.168.3.0.

Pro různé typy pravidel by se postupovalo podobně. Vždy by se vytvořila přesně odpovídající komunikace popsaná v pravidle a následně další jedna nebo více komunikací vyjadřující nějaké případy, kdy se pravidlo neuplatní.

10.3.2 Závislost funkčnosti linek

Celou předcházející podkapitolu bychom mohli dále rozšířit o testování zabezpečení ACL v případě nefunkčnosti linek. Zde je velký prostor pro možné konfigurace. Pokud bychom k celé věci přistoupili obecně a řekli bychom, že *n* vyjadřuje počet všech linek v topologii sítě, tak v případě simulování všech možností nefunkčností linek až na *n* nefunkčních (což by postrádalo smysl) je výsledný počet kombinací právě

$$2^n - 1 \tag{10.1}$$

Kdybychom do výrazu započítali i počet testovaných pravidel ACL a jejich kontrolních variant, byl by výsledný počet kombinací

$$\left(2^n - 1\right).pv \tag{10.2}$$

kde *p* je počet testovaných pravidel ACL a $v \in \mathrm{N}$ je průměrný počet kontrolních komunikací pro jedno pravidlo *p*.

10.4 Automatické ověření konfigurace směrování

Pro ověření funkčnosti směrování se nabízí řada pohledů, kterými se může testování ubírat. Mohli bychom například kontrolovat směrování na základě typů směrovacích protokolů, statického nebo dynamického směrování, funkčnosti a nefunkčnosti linek, dostupnosti koncových zařízení nebo sítí a další.

Odrazovým můstkem pro všechny typy testů nejspíše budou parametry směrování v konfiguračních souborech, a to zejména:

- Statické směrování – cílová síť (uzel).
- Dynamické směrování – síť definovaná příkazem *network*.

Zkusíme teoreticky popsat ověření dostupnosti koncových zařízení (sítí) navzájem v jednom směru komunikace podle zjištěných konfigurací statického a dynamického směrovaní na směrovačích, kde všechna koncová zařízení budou mít pouze jedno rozhraní. Jako testovou komunikaci spojení bychom mohli použít nějakou implementaci pingu v NS.

10.4.1 Popis řešení

Zjištěné adresy sítí uvedených v definicích cílových sítí statického směrování a v příkazu *network* u dynamického směrování si uložíme do množiny. Označme ji *A*. V poli $a_auto_node jsou uloženy pouze koncové uzly. Také víme, že každý uzel v tomto poli je pouze jediným uzlem ve své síti (pokud nepočítáme směrovač jako výchozí bránu). Na základě IP adres a masek sítí rozhraní koncových zařízení obsažených v poli $a_auto_node určíme jejich adresy sítí. Tyto sítě koncových zařízení si také uložíme do množiny. Označme ji *B*. Rozdílem množin $B - A = C$ získáme množinu,

která bude obsahovat koncové sítě, pro něž neexistuje definice směrování ze strany směrovačů. Následným porovnáním obsahu množiny *C* s IP adresami jednotlivých rozhraní koncových uzlů zjistíme přímo nedostupné koncové zařízení ze strany sítě směrovačů.

Dále budeme testovat konektivitu koncových zařízení, které mohou být dostupné ze strany sítě směrovačů. Na základě porovnání IP adres rozhraní pro jednotlivé zařízení obsažené v poli $a_auto_node s adresami sítí obsažených v množině *A* určíme koncové uzly, jejichž vzájemnou dostupnost budeme testovat. Všechny takovéto uzly uložíme do množiny *D*. Jejich počet označíme písmenem *d*. Kompletní ověření konektivity těchto zařízení můžeme rozdělit do dvou fází. Tou první je ověření dostupnosti výchozí brány pro každý koncový uzel v množině *D* a druhou je testování samotné sítě mezi směrovači.

Při testování dostupnosti výchozí brány nás zajímá pouze její dostupnost a ne chování směrování v případě výpadku linky. Proto počet testů první fáze je roven *d*. Počet neúspěšných testů dosažitelnosti výchozí brány si označíme písmenem *e*.

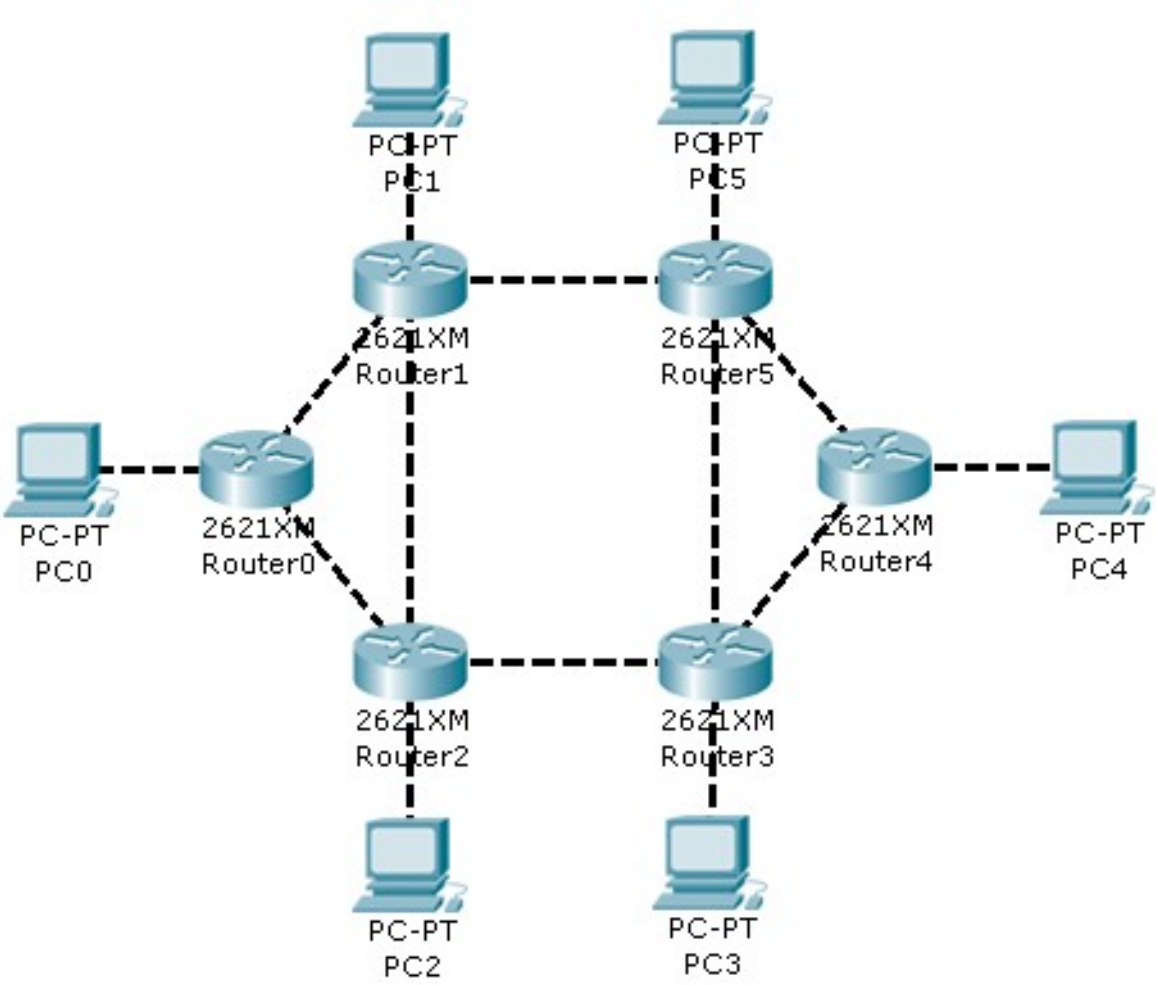
Ve druhé fázi nás již bude zajímat chování směrování v závislosti na výpadcích linek mezi směrovači. Testy budeme provádět tak, že budeme zkoumat dostupnost všech koncových zařízení navzájem, ale jen těch, která v první fázi dosáhla své výchozí brány. Počet takových zařízení označíme písmenem *z*, kde $z = d - e$. S použitím vzorce 10.1 popisujícího počet testových kombinací výpadků linek můžeme pro náš způsob ověřování směrování vytvořit obecný vztah popisující počet testů:

$$\frac{z!}{(z-2)!2!}\left[\left(2^{m}\right)-1\right]+d \qquad (10.3)$$

kde *m* je počet linek v síti směrovačů.

10.5 Příklad výpočtu

Nyní si zkusíme vypočítat množství testů potřebných pro ověření správnosti konfigurace ACL i směrování a celkový čas provádění kontrolovaní. Testování směrování budeme zkoumat při aktivních i neaktivních ACL na rozhraních směrovačů. Topologie sítě je zobrazena na následujícím obrázku 10.4.



Obrázek 10.4: Topologie testované sítě

Z obrázku můžeme vyčíst tyto hodnoty:

n = 14

m = 8

d = 6

Následující hodnoty si zvolíme:

p = 20

v = 3

e = 1, tedy z = 5

Dále si určíme, že jeden test bude trvat v průměru 0,025s.

Po dosazení do vzorce 10.2 dospějeme k počtu: 982 980 testů pro ACL. Pokud do kontroly nezahrneme testování linek, bude výsledný počet: 60.

Dále dosadíme do vzorce 10.3 pro výpočet počtu testů pro směrování a dospějeme k číslu: 2556. Tuto hodnotu musíme vynásobit 2, protože směrování testujeme při aktivních i neaktivních ACL. Výsledná hodnota bude: 5 112.

Dohromady tedy budeme vykonávat 988 092 testů respektive 5 172 bez testování funkčnosti linek. Při výpočtu celkové doby trvání ověřování vycházíme z toho, že testové komunikace pro jedno pravidlo ACL byly uskutečněny paralelně, proto výsledné počty testů ACL musíme podělit hodnotou *v* a až poté vynásobit délkou jednoho testu (0,025s). Takže výsledná doba testování bude: 2,311h respektive 128,3s. Pokud vezmeme v úvahu počet testů s kontrolou linek, je výsledná doba zanedbatelná, když si představíme, jak nám toto prověření může usnadnit práci při konfiguraci aktivních prvků. Kdybychom do výpočtu jednoho provedení testu nezapočítávali parametry linek, ale soustředili bychom se jen na samotné směrování a ACL, výsledný čas by mohl být daleko menší, závisel by pouze na výkonnosti počítače.

10.5.1 Možné problémy při vyhodnocování testů

Otázkou však zůstává, jakým způsobem zpracovat pro uživatele takové množství testů. Pokud bude v síti mnoho chyb, bude zavalen nekonečným výpisem problémů. I samotné nefunkčnosti mnoha linek v testu zajisté chybu vyvolají. Proto by bylo vhodné vymyslet nějaký způsob agregace chyb, nebo porovnání s očekávaným chováním.

Pokud bychom uplatnili nějaké vylepšení detekce chyb a prováděli testování bez parametrů linek, mohlo by se testování zastavit po např. prvních deseti nalezených problémech. Až by je administrátor odstranil, opakoval by testování tak dlouho, dokud by celá kontrola neskončila bezchybným stavem. Tento způsob odstraňování chyb v každém jednom kroku může vyřešit i problémy, které souvisí s těmito chybami, a další spuštěný test by pak zjistil o to menší počet dalších chyb než jen oněch deset v předcházejícím kroku opravených.

Také by se mohl testovat pouze určitý počet nefunkčních linek zároveň (např. maximálně 2) nebo aby si uživatel mohl zvolit, kterých linek se možný výpadek týká a ne zkoumání všech variant, které v reálném světě nemohou s téměř jistou pravděpodobností nastat (pokud nás nezajímají všechny možnosti).

Dalším možným problémem může být samotná velikost OTcl souboru, který popisuje testování a jeho zpracování v NS. Zkusme si tuto problematiku ukázat na uvedeném příkladu. Zaměříme se pouze na velikost popisu komunikací jednotlivých testů pro ověření dustupnosti koncových zařízení a konfigurací ACL a dále na popis výpadků linek (definice topologie sítě, ACL, linek a další je v porovnání s velikostí a počtem definic komunikací a výpadků linek zanedbatelná). Pro názornost výpočtu budeme předpokládat, že všechny jednotlivé popisy komunikací budou jak pro ověření dotupnosti koncových uzlů, tak i ACL budou v každém kroku stejně obsáhlé.

Definice komunikace pro ACL může vypadat následovně:

```
set tcp1 [new Agent/TCP]
$ns attach-agent $auto_node_1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $auto_node_0 $sink1
$ns connect $tcp1 $sink1
$tcp1 set acl_dst_port_ 3389
$tcp1 set src_addr_ [$tcp1 quad-to-value 192.168.2.2]
```

```
$tcp1 set dst_addr_ [$tcp1 quad-to-value 192.168.5.2]
$sink1 set src_addr_ [$sink1 quad-to-value 192.168.5.2]
$sink1 set dst_addr_ [$sink1 quad-to-value 192.168.2.2]
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
$ns at 0.0 "$ftp1 start"
$ns at 0.025 "$ftp1 stop"
```

Počet řádků: 15, počet viditelných znaků: 514, tedy 514B. Pro ověření dostupnosti koncových uzlů bychom mohli komunikaci definovat takto (uvedený zdrojový kód nemusí fungovat, slouží jen jako příklad):

```
set echo1 [new Agent/ECHO]
$ns attach-agent $auto_node_1 $echo1
$echo1 set src_addr_ [$echo1 quad-to-value 192.168.2.2]
$echo1 set dst_addr_ [$echo1 quad-to-value 192.168.5.2]
$ns at 0.0 "$echo1 start"
$ns at 0.025 "$echo1 stop"
```

Zde je počet řádků: 6 a počet viditelných viditelných znaků: 223, tedy 223B. Definice linek pro každý jeden test pravidla ACL může vypadat následovně (zde musíme uvést definici vždy pro všechny linky, i když se stav linky nezmění):

```
$ns rtmodel-at 0.0 down $router0 $router1
$ns rtmodel-at 0.0 up $router0 $router2
$ns rtmodel-at 0.0 up $router1 $router2
$ns rtmodel-at 0.0 up $router1 $router5
$ns rtmodel-at 0.0 up $router2 $router3
$ns rtmodel-at 0.0 up $router4 $router5
$ns rtmodel-at 0.0 up $router4 $router3
$ns rtmodel-at 0.0 up $router5 $router3
$ns rtmodel-at 0.0 up $router0 $pc0
$ns rtmodel-at 0.0 up $router1 $pc1
$ns rtmodel-at 0.0 up $router2 $pc2
$ns rtmodel-at 0.0 up $router3 $pc3
$ns rtmodel-at 0.0 up $router4 $pc4
$ns rtmodel-at 0.0 up $router5 $pc5
```

Počet řádků: 12, počet viditelných znaků: 524, tedy 524B. Definice linek pro každý jeden test dostupnosti koncových zařízení může vypadat následovně:

```
$ns rtmodel-at 0.0 down $router0 $router1
$ns rtmodel-at 0.0 up $router0 $router2
$ns rtmodel-at 0.0 up $router1 $router2
$ns rtmodel-at 0.0 up $router1 $router5
$ns rtmodel-at 0.0 up $router2 $router3
$ns rtmodel-at 0.0 up $router4 $router5
$ns rtmodel-at 0.0 up $router4 $router3
$ns rtmodel-at 0.0 up $router5 $router3
```

Počet řádků: 8, počet viditelných znaků: 314, tedy 314B.

Při počtu 982 980 ACL testů budou jednotlivé komunikace definovány pomocí zhruba $505{,}251.10^6$ znaků ($14{,}744.10^6$ řádků). Pro výpočet velikosti definic stavů linek musíme počet 982 980 vydělit hodnotou *v*, protože při jednom nastavení linek proběhne právě *v* testů. Změny stavů linek si tedy vyžádají použití okolo $171{,}693.10^6$ znaků ($3{,}931.10^6$ řádků). Dohromady zjištěný počet znaků představuje datovou velikost zhruba 645,584MB.

Při počtu 5112 testů dostupnosti koncových zařízení budou jednotlivé komunikace definovány pomocí zhruba $1{,}139.10^6$ znaků (30672 řádků). Změny stavů linek vyjádříme s použitím okolo $1{,}605.10^6$ znaků (40896 řádků). Dohromady zjištěný počet znaků představuje datovou velikost zhruba 2,62MB.

Výsledná velikost souboru popisující testování by byla okolo 648,204MB (zhruba $18{,}746.10^6$ řádků). Zvládne však takto veliký soubor zpracovat NS? Při výpočtu nebyly použity žádné optimalizační techniky, které by velikost kódu snížily. Vzhledem k tomu, že dopředu známe posloupnost kombinací aktivních/neaktivních linek, mohli bychom v každém kroku měnit pouze stav linek, u kterých dojde ke změně, a ne u všech, kde se často jejich stav nemění. Také bychom se mohli pokusit o vícenásobné využití jedné komunikace.

10.6 Shrnutí

V této části práce jsme ukázali jednoduchý příklad, který prezentoval manualní kontrolu správnosti konfigurace ACL. Ukázali jsme na něm možný princip testování a představili jsme dva nástroje, jakým chyby ACL odhalit (příkaz *debug-acl* a program NAM). Dále jsme teoreticky popsali, jakým způsobem bychom mohli kontrolu provádět v případě jiných typů ACL.

Na praktickou ukázku jsme navázali teoretickou úvahou, ve které jsme se pokusili obecně popsat automatické testovaní konfigurací směrování a ACL pro celou topologii sítě. Pro směrování i ACL jsme definovali obecné vzorce, které určují počet testů námi zvoleného pohledu kontroly v závisloti zejména na funkčnosti jednotlivých linek.

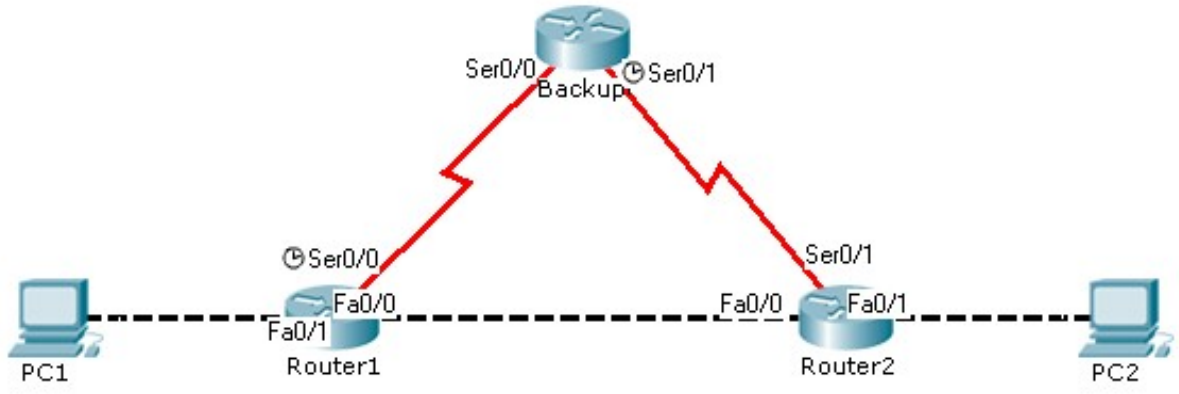
Samotný výpočet jsme ukázali na příkladu, kde jsme zjistili výhody i nevýhody provádění tolika testů. Velké množství kontrol může téměř eliminovat chyby v konfiguracích, ale také může být obtížné na vyhodnocování. Proto jsme uvedli několik přístupů, které by mohly zpracování výsledků zjednodušit. Dále jsme se pokusili o výpočet velikosti souboru, který by popisoval veškeré testování v NS pro prezentovaný příklad.

11 Příklad simulace reálné sítě

V této kapitole ukážeme využití převodního nástroje a NS nebo NSJS na příkladu, který bude simulovat malou reálnou síť. Budeme také zkoumat, zdali chování simulace odpovídá chování zjištěnému v laboratoři. Veškeré konfigurační soubory směrovačů i výsledný OTcl skript jsou obsaženy v příloze této práce.

11.1 Popis simulované sítě

Topologie sítě je znázorněna na následujícím obrázku 11.1.



Obrázek 11.1: Topologie simulované sítě

Zapojení je podobné, jako tomu bylo v desáté kapitole: Černé čárkované spoje mezi zařízeními představují UTP křížený kabel a spoje tvořené klikatou červenou čarou jsou tvořeny sériovou linkou. Všechny směrovače v síti (Router1, Router2, Backup) jsou Cisco směrovače modelu 2801 s verzí IOS 12.4 (17).

UTP spojení mezi Router1 a Router2 představuje hlavní linku a v případě jejího výpadku je komunikace vedena přes záložní sériovou linku. Směrování v síti tvořené UTP kabely zajišťuje protokol RIP v2, kde na rozhraních Fa0/0 směrovačů Router1 a Router2 je nastaveno zabezpečení protokolu RIP. Směrování v síti sériových linek je řešeno staticky. Samotné přesměrování komunikace při výpadku hlavní linky je provedeno pomocí statického směrování s explicitně zvětšenou administrative distance.

Na směrovači Router2 na rozhraní Fa0/1 pro příchozí směr je nakonfigurován ACL s následující definicí pravidel:

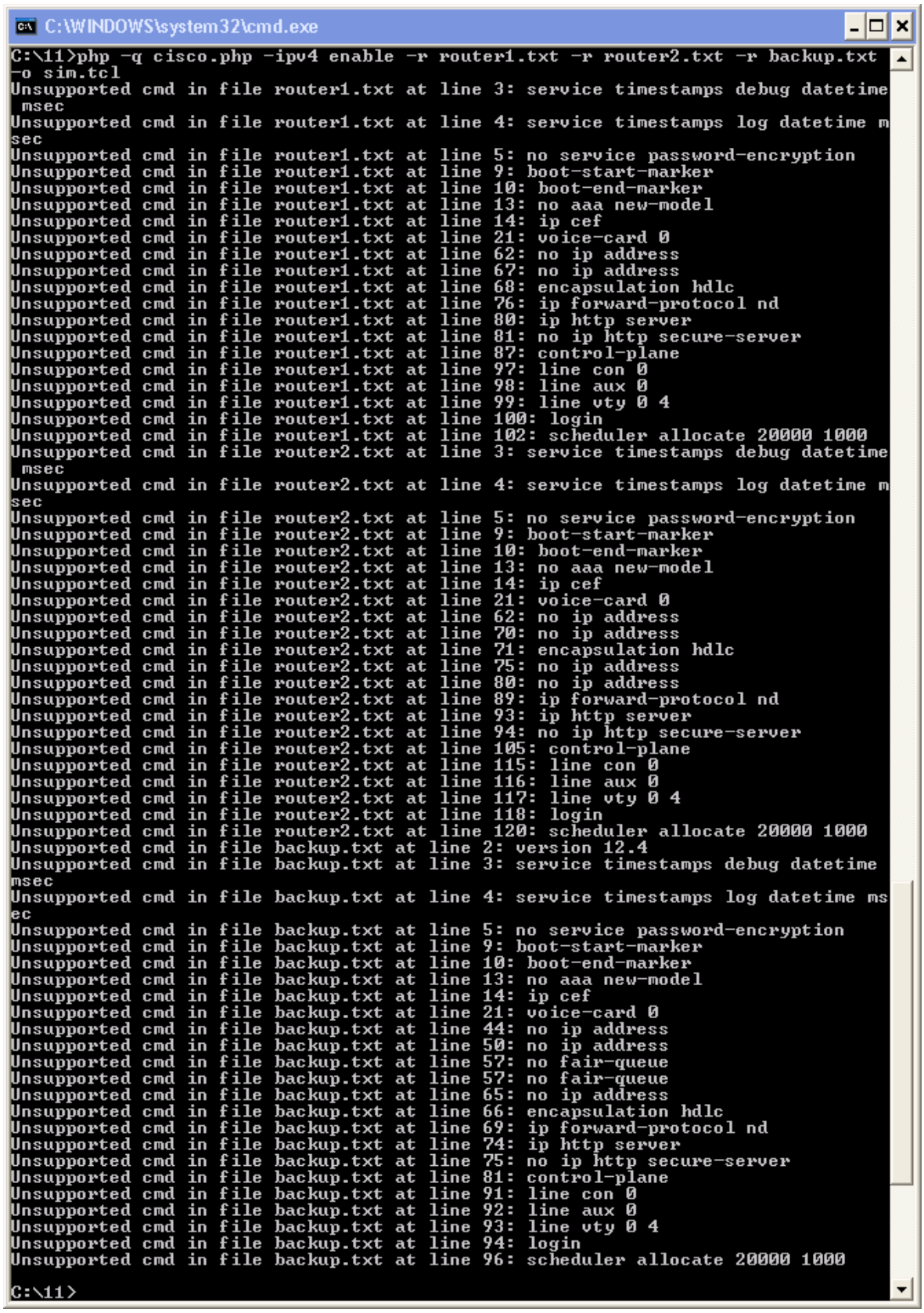
```
access-list 101 permit tcp 192.168.2.0 0.0.0.255 192.168.5.0 0.0.0.255 eq 3389
access-list 101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.1.1 eq www
access-list 101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.4.2 eq www
access-list 101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.2.1 eq www
access-list 101 permit icmp any any
```

Z uvedených pravidel je patrné, že z PC2 je blokována veškerá komunikace až na protokol TCP, kterým se může buď přes terminálového klienta přípojovat k PC1, nebo přistupovat k webovým stránkám na uvedených IP adresách (tyto IP adresy jsou definovány na rozhraních tří směrovačů). Dále mu je povoleno posílání kamkoliv veškerých typů ICMP paketů.

Samotnou simulaci si uzpůsobíme tak, že vytvoříme tři komunikace, které budou simulovat TCP komunikaci z PC2 do PC1 s cílovým portem 3389, dále UDP přenos z PC2 do PC1 a UDP komunikaci z PC1 do PC2. Při testování těchto přenosů v laboratoři zjistíme, že i v případě výpadku hlavní linky TCP komunikace dosáhne PC1 a je stále korektně povolována ACL, UDP přenos z PC2 do PC1 je stále blokován a komunikace z PC2 do PC1 je stále funkční.

11.2 Vytvoření simulace

Nyní popíšeme jednotlivé kroky potřebné k vytvoření simulace. Ze směrovačů nejdříve získáme konfigurační soubory např. zkopírováním na tftp server nebo exportem z jejich webového rozhraní. Soubory pojmenujeme podle názvů směrovačů (Router1 – router1.txt, Router2 – router2.txt, Backup – backup.txt). Na tyto soubory použijeme převodní nástroj. Tuto část postupu názorně ukazuje následující obrázek 11.2.

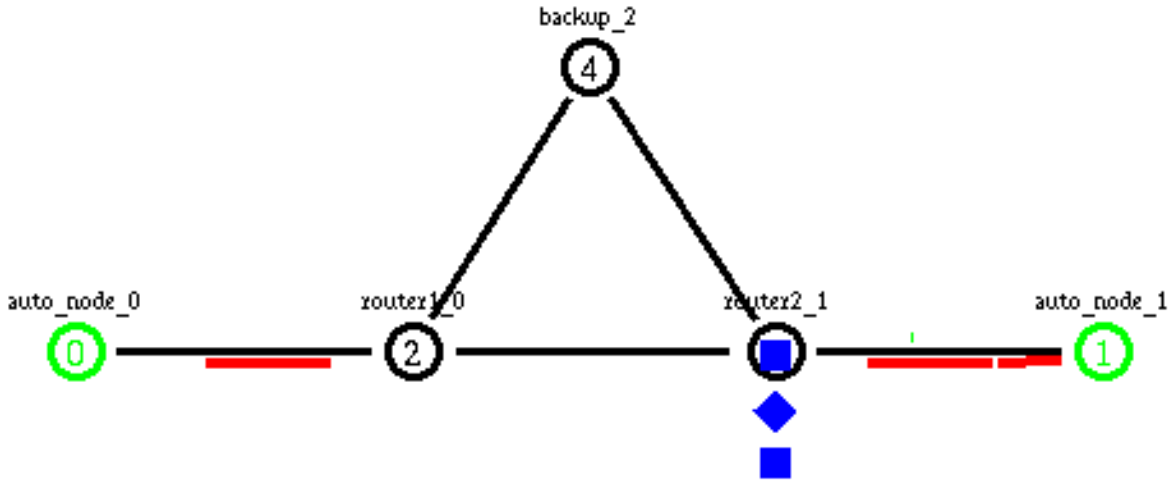


Obrázek 11.2: Použití převodního nástroje

Na obrázku 11.2 vidíme příkaz, kterým překlad spustíme, ale také i výpisy jednotlivých nepodporovaných příkazů převodním nástrojem. Po skončení překladu máme k dispozici soubor sim.tcl, který popisuje model sítě i určitou testovou komunikaci. Samotný skript musíme upravit definováním tří typů komunikací uvedených výše a také nastavením časových intervalů popisujících výpadek hlavní linky, a to tak, že první sekundu bude linka aktivní, druhou sekundu bude nefunkční a třetí a čtvrtou sekundu bude opět funkční. Pak již můžeme simulaci spustit.

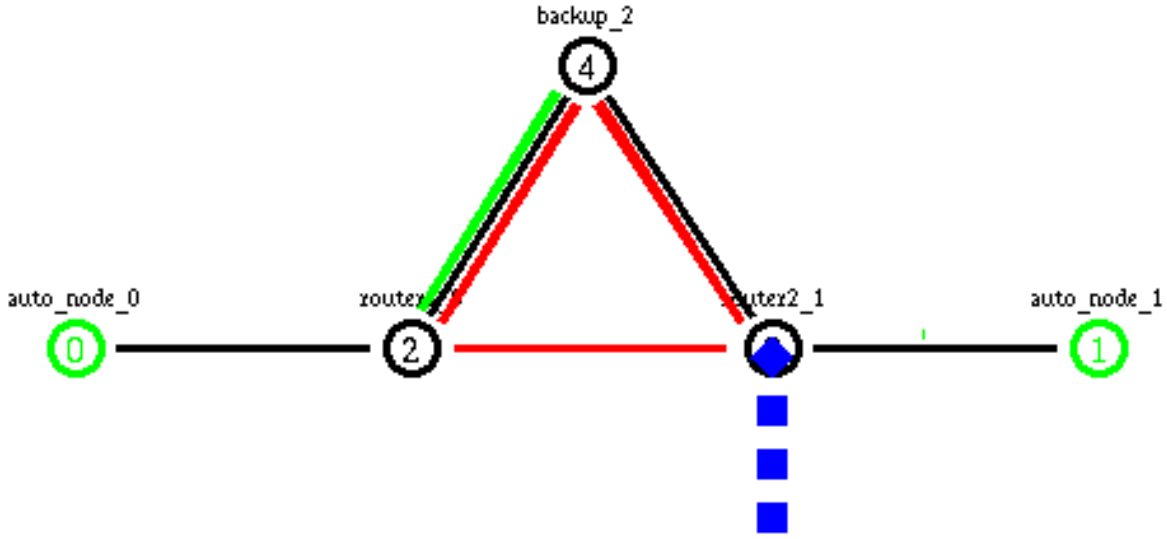
11.3 Vyhodnocení simulace

V intervalu první sekundy je stav sítě normální a s použitím programu NAM můžeme vizuelně sledovat její chování. Situaci ukazuje následující obrázek 11.3, kde červeně znázorněné pakety představují TCP komunikaci, modrá je UDP přenos z PC2 (auto_node_1) do PC1 (auto_node_0) a zelená je UDP komunikace z PC1 (auto_node_0) do PC2 (auto_node_1).



Obrázek 11.3: Chování sítě v intervalu první sekundy simulace

Ve druhém intervalu, tj. hlavní linka je nefukční, je v grafickém výstupu programu NAM (obrázek 11.4) nefunkční hlavní linka znázorněna červenou plnou čarou mezi směrovači router1_0 a router2_1. Dále z obrázku vidíme, že tok paketů jednotlivých komunikací odpovídá v laboratoři zjištěnému chování při výpadku hlavní linky.



Obrázek 11.4: Chování sítě v intervalu druhé sekundy simulace

V posledním intervalu (třetí a čtvrtá sekunda) odpovídá chování sítě po ustálení stavu prvnímu časovému intervalu simulace.

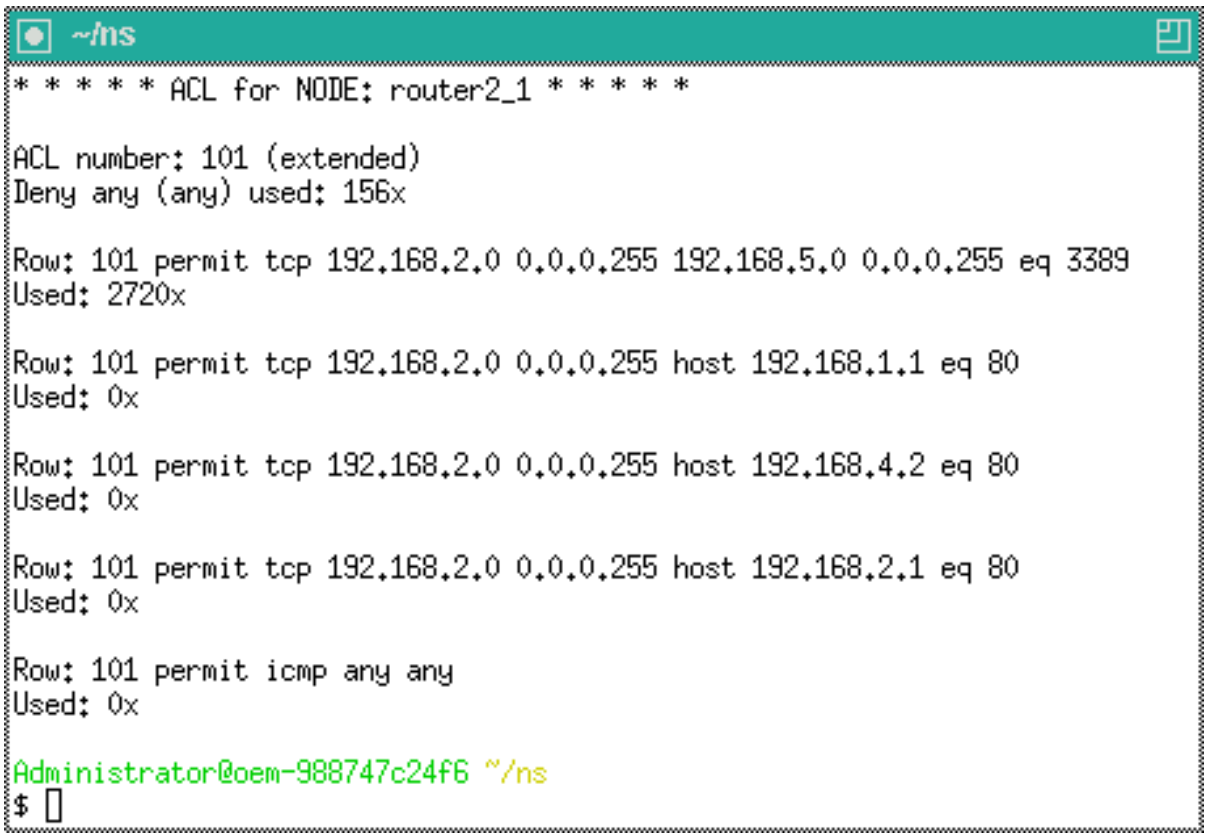
Pro zjištění správné funkčnosti ACL můžeme použít výpisy používaných pravidel. Následující obrázek 11.5 ukazuje část z výpisu reakcí pravidel ACL na TCP a UDP komunikaci směřovanou z PC2 do PC1.

```
~/ns
*** ACL debug. Node: router2_1 IN-boud pkt id: 3135, protocol: udp,
src IP: 192.168.2.2, dst IP: 192.168.5.2 was rejected
by rule: deny any any
at: 2.779516
*** ACL debug. Node: router2_1 IN-boud pkt id: 3137, protocol: tcp,
src IP: 192.168.2.2, dst IP: 192.168.5.2 was permited
by rule: 101 permit tcp 192.168.2.0 0.0.0.255 192.168.5.0 0.0.0.255 eq 3389
at: 2.784307
*** ACL debug. Node: router2_1 IN-boud pkt id: 3139, protocol: udp,
src IP: 192.168.2.2, dst IP: 192.168.5.2 was rejected
by rule: deny any any
at: 2.805766
*** ACL debug. Node: router2_1 IN-boud pkt id: 3141, protocol: udp,
src IP: 192.168.2.2, dst IP: 192.168.5.2 was rejected
by rule: deny any any
at: 2.832016
*** ACL debug. Node: router2_1 IN-boud pkt id: 3144, protocol: tcp,
src IP: 192.168.2.2, dst IP: 192.168.5.2 was permited
by rule: 101 permit tcp 192.168.2.0 0.0.0.255 192.168.5.0 0.0.0.255 eq 3389
at: 2.849307
```

Obrázek 11.5: Výpis uplatněných pravidel ACL pro TCP a UDP komunikaci

Pokud bychom prozkoumali celý výpis použitých pravidel, zjistíme, že filtrování paketů pomocí ACL bylo v průběhu celé simulace stejné jako při reálném zapojení.

Na závěr simulace si můžeme zobrazit zjednodušený výpis definovaných ACL na směrovači Router2, kde vidíme statistické hodnoty použitelnosti jednotlivých pravidel (obrázek 11.6).



Obrázek 11.6: Statistické hodnoty použitelnosti pravidel ACL

11.4 Shrnutí

V této příkladové kapitole jsme prakticky ukázali, jakým způsobem provádět simulaci s pomocí převodního nástroje a NS nebo NSJS. V první řadě musíme získat konfigurační soubory z Cisco směrovačů. Dále použijeme převodní nástroj a vytvoříme simulační model. Vygenerovaný popis simulace můžeme uplatnit nebo ho změnit podle našich simulačních požadavků. Pak již jen simulaci spustíme a na základě různých výstupů z chování sítě v průběhu simulace můžeme provádět rozbor simulace.

Následující příklad také demonstroval ekvivalenci implementovaných součásti jak v převodním nástroji (parametry linek, topologie sítě), tak i v NS (ACL, směrování, IPv4 adresace) s reálnou síti.

12 Závěr

Cílem diplomové práce bylo vytvoření nástroje, jenž by umožňoval automatický překlad konfiguračních souborů Cisco routerů, ze kterého by následně vytvořil simulační model a simulaci pro NS a NSJS. Při překladu jsem se zabýval pouze nastavením pro hostname, interface, statické směrování, směrování pomocí protokolů RIP v1 a v2, EIGRP, OSPF, zabezpečení protokolu RIP a konfigurace ACL. Také jsem navrhl způsob IPv4 adresace uzlů, která není v NS podporována. Dalším cílem bylo vytvoření podpory pro filtrování paketů pomocí ACL.

12.1 Dosažené výsledky

Dosažené výsledky rozdělím do dvou ucelených bloků.

12.1.1 Převodní nástroj

Prvním krokem bylo získání informací z konfiguračních souborů aktivních prvků. Tento proces je bezproblémový a informace lze dobře strukturovat pro další zpracování. Dále jsem analyzoval konfigurační soubory teoreticky a prakticky zkoušením v mnoha simulacích, kde jsem zjišťoval, které ze zkoumaných parametrů lze použít v NS i NSJS.

Veškeré možnosti nastavení, jež nabízí Cisco routery pro rozhraní je převoditelné i do NS. Musel jsem se však vyrovnat s problémem v NS, kde nelze adresovat uzly v síti podle IP adres, ale jen na základě názvů uzlů (ID). Tento nedostatek jsem vyřešil na základě jedinečnosti IP adresy v modelované topologii sítě, podle níž jsem hledal hostname routeru (název uzlu).

Musel jsem si také poradit se zapojením typu hvězda (NS umí pouze tvořit spoje typu point-to-point), kde centrálním prvkem může například být switch. Centrální prvek jsem nahradil novým uzlem a vyřešil problém statického směrování vlivem přidání nového uzlu. Samo vytvoření topologie modelu sítě je pak bezproblémové.

Statické směrování je kompletně přenositelné do NS. Daleko horší je však situace u dynamického směrování, ve kterém nelze téměř vůbec nic parametrizovat. U směrování DV (distance-vector) je aspoň možné určit, které uzly budou tento typ routování používat. Pro Session směrování (link-state) tato volba nelze nastavit vůbec. Není tedy možné ovládat směrovací protokoly na úrovni rozhraní, tak jak je tomu v konfiguračních možnostech Cisco routerů.

Dalším nedostatkem směrování v NS je nemožnost kombinovat některé typy směrování, např. Session + Manual (u tohoto případu dokonce skončí NS vlastní chybou jádra simulátoru “Segmentation fault (core dumped)“). Nelze také použít stejnou směrovací techniku vícekrát v jednom modelu sítě (např. RIP + EIGRP).

Použití dynamického směrování jsem implementoval aspoň na základě výskytu v konfiguračních souborech tak, že pokud se v parametru některého z dynamických směrovacích protokolů objevuje definice sítě (příkaz *network*), směrovací protokol jsem v modelu sítě (Session směrování) nebo v uzlu (DV směrování) povolil.

I přes uvedené nedostatky NS jsem vytvořil nástroj, který je použitelný pro jakoukoliv topologii sítě. Pokud se nebude hlavně jeho používání zaměřovat na zjišťování chování směrování, ale spíše na protokoly a služby vyšších vrstev ISO/OSI modelu, je tento nástroj užitečný.

NSJS podporuje IPv4 adresaci uzlů, statické směrování a dynamické směrování pomocí protokolu RIP v1 a v2, které lze konfigurovat přímo pomocí IP adres s použitím stejných příkazů jako na Cisco směrovačích. Použití převodního nástroje s NSJS potom řeší všechny uvedené nedostatky NS při vytváření modelu simulace a rozšiřuje tím možnost využití např. pro testování konfigurací směrování.

12.1.2 Filtrování paketů pomocí access control list

Při implementaci ACL jsem vytvářel zcela nový modul pro NS i NSJS. Proto jsem nebyl limitovaný uvedenými nedostatky v NS a mohl jsem tak vytvořit principielně ekvivalentní systém filtrace paketů pomocí ACL, který najdeme na Cisco směrovačích. Jediným omezením byla nedostatečná podpora NS pro některé protokoly (např. ICMP) a vedlejší paramery v ACL (např. definice skutečných časů), které sice snížily úroveň podpory některých typů ACL, ale nijak nebránily inspekci paketů podle jejich zásadních parametrů. Implementace ACL je tedy schopná kompletně filtrovat pakety na základě:

- zdrojové a cílové adresy zařízení nebo sítě
- typu protokolu
- zdrojového a cílového portu.

Podporované typy ACL jsou následující:

- standard ACL
- extended ACL
  - dynamic ACL
  - reflexive ACL
- named ACL.

12.2 Network Simulator

Nástroj NS mě velice zaujal. Jeho nespornou výhodou jsou volně dostupné zdrojové kódy. Nejedná se však o nástroj typu „naistaluj a pracuj“. Pokud chce uživatel používat NS pod systémem Windows,

musí vyřešit netriviální instalaci pomocí Cygwin nebo musí použít starší verzi, která je pro Windows již zkompilována.

Pro NS existují hlavní dva prameny informací [2] a [3]. Zde bych chtěl podotknout, že se, podle mého názoru, mohli autoři dokumentace více zaměřit právě na popis směrování. Rovněž pak v některých částech mohli pro vysvětlení použít více slov než pouze výpisu zdrojových kódů, algoritmů a odkazů na soubory zdrojových kódů. Např. příkaz pro použití statického směrování *add-route*, který je pro něj v NS klíčový, je v dokumentaci uveden i s příkladem použití, ale pokud jej tak použijeme, simulace nefunguje. Uživatel je pak nucen dlouhou dobu bádat, jak problém vyřešit. Přitom by stačila jedna ukázka nebo lepší vysvětlení principu použití.

Pro začínající uživatele NS a zejména pak ty, kteří chtějí NS dále rozšiřovat, může být proniknutí do jeho systému zdlouhavější. Pomoci mu mohou tutoriály na internetu, např. [1], kde se případný uživatel může rychleji uvést do problematiky.

12.3 Další vývoj projektu

Další vývoj projektu by se mohl zabývat ověřováním konfigurací směrování a ACL a také automatickou analýzou chování sítě. Možný princip, obecný postup i problémy při vyhodnocování výsledků testů byl nastíněn v desáté kapitole. Výsledkem by pak mohl být nástroj nejlépe s grafickým uživatelským rozhraním, který by sám odhaloval chyby v topologii sítě i nedostatky v konfiguracích aktivních zařízení. Grafické uživatelské rozhraní bychom využili především na daleko přehlednější ovládání a vyhodnocování testů.

Také bychom mohli rozšířit implementovanou IPv4 adresaci o možnost adresování nejenom topologie sítě složené pouze z point-to-point spojů, ale i o možnost spojení typu hvězda.

Dále bychom mohli navázat na NSJS a vylepšit ho o další směrovací protokoly a aplikace, které by přispěly k větší podpoře různých typů konfigurací směrovačů a služeb, jež v síti pracují.

Z pohledu ACL by se NS i NSJS mohly dále vylepšit o podporu reálného času, abychom mohli plně využívat některé typy ACL a také rozšířit současně implementované protokoly nebo vytvořit nové, tak abychom mohli využít veškerých možností filtrování paketů, jež ACL nabízí.

Literatura

[1] Chung, J., Claypool, M.: *NS by Example*, dokument dostupný na URL: http://nile.wpi.edu/NS (květen 2008)

[2] Altman, E., Jiménez, T.: *SNS Simulator for beginners*,2003, dokument dostupný na URL: http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf (květen 2008)

[3] *The ns Manual*, dokument dostupný na URL: http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf (květen 2008)

[4] *QualNet 4.0 Product Tour*, dokument dostupný na URL: http://www.scalable-networks.com/distributions/documentation/4.0/QualNet-4.0-ProductTour.pdf (květen 2008)

[5] *QualNet University Program*, dokument dostupný na URL: http://www.scalable-networks.com/customers/qup/index.php (květen 2008)

[6] *OMNeT++ User Manual version 3.2*, dokument dostupný na URL: http://www.omnetpp.org/doc/manual/usman.html (květen 2008)

[7] *OPNET Modeler*, dokument dostupný na URL: http://www.opnet.com/solutions/brochures/Modeler.pdf (květen 2008)

[8] *Firewall*, dokument dostupný na URL: http://en.wikipedia.org/wiki/Firewall (květen 2008)

[9] *Cisco CCNA Exploration 4*, dokument dostupný na URL: https://netacad.fit.vutbr.cz/ccna/ccna4/v40/index.html (květen 2008)

[10] *Configuring IP Access Lists*, dokument dostupný na URL: http://www.cisco.com/en/US/products/sw/secursw/ps1018/products_tech_note09186a00800a5b9a.shtml (květen 2008)

[11] *Configuring Commonly Used IP ACLs*, dokument dostupný na URL: http://www.cisco.com/en/US/tech/tk648/tk361/technologies_configuration_example09186a0080100548.shtml (květen 2008)

Seznam příloh

Příloha 1

V následující tabulce popíšeme obsah přiloženého CD.

Umístění	**Popis**
examples\10	Konfigurační soubory směrovačů a výsledný OTcl skript simulace z příkladu v desáté kapitole
examples\11	Konfigurační soubory směrovačů a výsledný OTcl skript simulace z příkladu v jedenácté kapitole
ns\bin	Binární soubor NS vytvořený v Cygwin obsahující rozšíření NSJS i veškeré implementace popsané v této práci. A také binární soubor pro NAM vytvořený v Cygwin. Není tedy nutná přímá instalace ns-allinone-2.32 a následná aplikace patche.
ns\patch	Patch pro rozšíření oficiální verze ns-2.32 o NSJS i veškeré implementace popsané v této práci.
ns\ ns-allinone-2.32.tar.gz	Oficiální distribuce ns-2.32 s dalšími balíky potřebnými pro práci s NS (např. NAM, Tcl).
php\bin	Obsahuje binární soubor php.exe s jednou knihovnou, který postačí ke spuštění a práci s převodním nástrojem pod Windows.
php	Zdrojové kódy převodního nástroje.
text	Text této práce ve formátu pdf.

Příloha 2

V následující tabulce jsou uvedeny příkazy z konfiguračních souborů Cisco routerů, se kterými umí pracovat vytvořený převodní nástroj, a následně NS (NSJS). Sloupce „NS“ a „NSJS” charakterizují, do jaké míry je příkaz podporován svým významem (funkcí) v NS respektive NSJS. Příkazy, které nejsou v tabulce uvedeny, nejsou podporovány vůbec.

Jednořádkový / Hlavní příkaz	**Parametrizující příkaz**	**NS**	**NSJS**	**Příklad v OTcl skriptu**	**Poznámka**
hostname	-	úplná	úplná	$auto_node_1 hostname auto_node_1	
interface ethernet[type][port] interface fastethernet[type][port] interface gigabitethernet[type][port]	ip address	žádná	úplná	$n0 interface $n1 192.168.4.2 255.255.255.0	
	duplex	úplná	úplná	$ns duplex-link $n1 $n0 104857600b 2ms DropTail	104857600b = min(duplex,speed, bandwidth) z $n1 a $n0
	speed	úplná	úplná		
	bandwidth	úplná	úplná		
	delay	úplná	úplná	$ns duplex-link $n1 $n0 104857600b 2ms DropTail	2ms = delay
	shutdown	úplná	úplná	-	
	ip access-group	úplná	úplná	$n0 ip-access-group [$n1 id] 101 out	
	ip rip receive version	žádná	úplná	$ns at 0.0 "$n0 RIP version receive 1 $n4"	
	ip rip send version	žádná	úplná	$ns at 0.0 "$n0 RIP version send 1 $n4"	
	ip rip authentication key-chain	žádná	úplná	$ns at 0.0 "$n0 RIP authentication key-chain family $n4"	
	ip rip authentication mode	žádná	úplná	$ns at 0.0 "$n0 RIP authentication mode md5 $n4"	
	ip split-horizon	žádná	úplná	$ns at 0.0 "$n0 RIP split-horizon $n4"	
	no ip split-horizon	žádná	úplná	$ns at 0.0 "$n0 RIP no split-horizon $n4"	

Jednořádkový / Hlavní příkaz	Parametrizující příkaz	NS	NSJS	Příklad v OTcl skriptu	Poznámka
interface serial[type][port]	ip address	úplná	úplná	$n0 interface $n1 192.168.4.2 255.255.255.0	
	clock rate	úplná	úplná	$ns duplex-link $n1 $n0 56000b 20ms DropTail	56000b = min(clock rate, delay) z $n1 a $n0
	bandwidth	úplná	úplná		
	delay	úplná	úplná	$ns duplex-link $n1 $n0 56000b 20ms DropTail	20ms = delay
	shutdown	úplná	úplná	-	
	ip access-group	úplná	úplná	$n0 ip-access-group [$n1 id] 101 out	
	ip rip receive version	žádná	úplná	$ns at 0.0 "$n0 RIP version receive 1 $n4"	
	ip rip send version	žádná	úplná	$ns at 0.0 "$n0 RIP version send 1 $n4"	
	ip rip authentication key-chain	žádná	úplná	$ns at 0.0 "$n0 RIP authentication key-chain barney $n4"	
	ip rip authentication mode	žádná	úplná	$ns at 0.0 "$n0 RIP authentication mode md5 $n4"	
	ip split-horizon	žádná	úplná	$ns at 0.0 "$n0 RIP split-horizon $n4"	
	no ip split-horizon	žádná	úplná	$ns at 0.0 "$n0 RIP no split-horizon $n4"	
ip route	-	částečná	úplná	NS: [$n0 get-module "Manual"] add-route-to-adj-node -default $n1 NS: [$n0 get-module "Manual"] add-route [$n3 set address_] [[$ns link $n0 $n1] head] NSJS: $ns at 0.0 "$n0 Static route 10.0.0.0 255.0.0.0 192.168.1.2 200"	NS: nepodporuje prioritu cest
router rip	network	částečná	úplná	NS: $ns rtproto DV "$n0 $n1 $n5" NSJS: $ns at 0.0 "$n3 RIP network 192.168.100.0"	NS: umí pouze povolit DV směrování na úrovni uzlů
	version	žádná	úplná	$ns at 0.0 "$n3 RIP version 2"	
	distance	žádná	úplná	$ns at 0.0 "$n3 RIP distance 60"	
	auto-summary	žádná	úplná	$ns at 0.0 "$n3 RIP auto-summary"	
	no auto-summary	žádná	úplná	$ns at 0.0 "$n3 RIP no auto-summary"	
	timers basic	žádná	úplná	$ns at 0.0 "$n2 RIP timers_basic 5 10 15 20"	
	passive-interface	žádná	úplná	$ns at 0.0 "$n3 RIP passive-interface $n4"	
router eigrp	[as]	žádná	žádná	-	
	network	částečná	žádná	NS: $ns rtproto DV "$n0 $n1 $n5"	NS: umí pouze povolit DV směrování na úrovni uzlů

Jednořádkový / Hlavní příkaz	**Parametrizující příkaz**	**NS**	**NSJS**	**Příklad v OTcl skriptu**	**Poznámka**
router ospf	[process id]	žádná	žádná	-	
	network	částečná	žádná	NS: $ns rtproto Session	NS: umí pouze povolit Session směrování na úrovni celé simulace
ip access-list [type] [number|name]	-	částečná	částečná	*definice hlavního příkazu je přenesena do každého řádku popisující jedno pravidlo*	v [type] jsou podporovány pouze hodnoty extended a standard
	[line]	částečná	částečná	$n0 access-list "extended rdp permit tcp 192.168.1.0 0.0.0.255 host 192.168.2.2 eq 3389"	pouze řádky začínající příkazem deny, permit, remark
access-list	-	částečná	částečná	$n0 access-list "105 permit tcp host 192.168.3.2 host 192.168.4.1"	nejsou podporovány všechny typy ACL a také není podporována celá příkazová sada podporovaných typů ACL
key chain	[name]	žádná	úplná	$n1 keychain family	
	key	žádná	úplná	$n1 key family 1	
	key-string	žádná	úplná	$n1 key-string family 1 jirka	

Příloha 3

V této části uvedeme jednotlivé zdrojové kódy NS, ve kterých jsou provedeny změny. Pro lepší orientaci v kódech jsou místa jejich rozšíření vyznačena komentařem ***begin Jiri Macku*** (pro začátek) a ***end Jiri Macku*** (pro konec). Jednotlivá pole ve sloupci „umístění“ v následující tabulce jsou brána z pohledu adresáře ns-2.32.

Umístění	**Popis**
acl	Nově přidaný adresář do struktury obsahující implementaci bloků ACL IN/OUT (acl.cc, acl.h) a funkce pro práci s ACL a definice polí pro aplikování ACL na rozhraní a uložení jednotlivých pravidel ACL (acl-functions.cc, acl-functions.h).
common\agent.cc common\agent.h	Rozšíření informací, které agent vkládá každému vygenerovanému paketu.
common\node.cc common\node.h	Rozšíření funkcí uzlu o práci s ACL.
common\packet.h	Rozšíření hlavičky paketu o potřebné informace pro ACL.
tcl\lib\ns-agent.tcl	Funkce pro práci z IP adresami.
tcl\lib\ns-default.tcl	Nastavení výchozích hodnot.
tcl\lib\ns-lib.tcl	Povolení ACL pro celou simulaci.
tcl\lib\ns-link.tcl	Vytvoření bloků ACL IN/OUT při vytváření jednotlivých simplex linek a rozšíření funkcí pro adresování jednotlivých bloků na lince.
tcl\lib\ns-node.tcl	IPv4 adresování a pojmenování uzlů pomocí hostname.

Příloha 4

Konfigurační soubory směrovačů z příkladu v desáté kapitole a OTcl skript simulace. Veškeré zde vypsané soubory jsou obsaženy v přiloženém CD v adresáři examples\10.

Router1_0 / r1.txt

```
!
version 12.2
no service password-encryption
!
hostname Router1
!
!
!
!
interface FastEthernet0/0
 ip address 192.168.103.1 255.255.255.0
 ip access-group 101 out
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
!
interface Serial0/0
 ip address 192.168.101.1 255.255.255.0
 clock rate 4000000
!
interface Serial0/1
 no ip address
!
router rip
 network 192.168.1.0
 network 192.168.103.0
!
ip classless
ip route 192.168.3.0 255.255.255.0 192.168.101.2 200
!
access-list 101 permit tcp 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255
!
!
!
line con 0
line vty 0 4
 login
!
!
end
```

Router2_1 / r2.txt

```
!
version 12.2
no service password-encryption
!
hostname Router2
!
!
!
!
interface FastEthernet0/0
 no ip address
 duplex auto
 speed auto
 shutdown
!
interface FastEthernet0/1
 no ip address
 duplex auto
 speed auto
 shutdown
!
interface Serial0/0
 ip address 192.168.101.2 255.255.255.0
!
interface Serial0/1
 ip address 192.168.102.2 255.255.255.0
 clock rate 4000000
!
ip classless
ip route 192.168.1.0 255.255.255.0 192.168.101.1
ip route 192.168.3.0 255.255.255.0 192.168.102.1
!
!
!
!
line con 0
line vty 0 4
 login
!
!
end
```

Router3_2 / r3.txt

```
!
version 12.2
no service password-encryption
!
hostname Router3
!
!
!
!
interface FastEthernet0/0
 ip address 192.168.103.2 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.3.1 255.255.255.0
 duplex auto
 speed auto
!
interface Serial0/0
 no ip address
!
interface Serial0/1
 ip address 192.168.102.1 255.255.255.0
!
router rip
 network 192.168.3.0
 network 192.168.103.0
!
ip classless
ip route 192.168.1.0 255.255.255.0 192.168.102.2 200
!
!
!
!
line con 0
line vty 0 4
 login
!
!
end
```

Vygenerovaný OTcl skript s úpravou testové komunikace / test.tcl

```
set ns [new Simulator]
$ns enable-acl
$ns IPv4 ON

$ns color 1 Red
$ns color 2 Green


set file1 [open out.nam w]
$ns namtrace-all $file1

proc finish {} {
   global ns file1
   $ns flush-trace
   close $file1
   exec nam out.nam &
   exit 0
}

set auto_node_0 [$ns node]
$auto_node_0 label auto_node_0
$auto_node_0 hostname auto_node_0
$ns at 0.0 "$auto_node_0 color green"
set auto_node_1 [$ns node]
$auto_node_1 label auto_node_1
$auto_node_1 hostname auto_node_1
$ns at 0.0 "$auto_node_1 color green"


set router1_0 [$ns node]
$router1_0 label router1_0
$router1_0 hostname router1_0
$ns at 0.0 "$router1_0 color black"
set router2_1 [$ns node]
$router2_1 label router2_1
$router2_1 hostname router2_1
$ns at 0.0 "$router2_1 color black"
set router3_2 [$ns node]
$router3_2 label router3_2
$router3_2 hostname router3_2
$ns at 0.0 "$router3_2 color black"

$ns rtproto RIP $router1_0 $router3_2
$ns rtproto StaticIPv4

$ns duplex-link $router3_2 $router1_0 104857600b 2ms DropTail
$ns duplex-link $router2_1 $router3_2 4000000b 2ms DropTail
$ns duplex-link $router2_1 $router1_0 4000000b 2ms DropTail
$ns duplex-link $router3_2 $auto_node_0 104857600b 2ms DropTail
$ns duplex-link $router1_0 $auto_node_1 104857600b 2ms DropTail

$router1_0 interface $router3_2 192.168.103.1 255.255.255.0
$router1_0 interface $auto_node_1 192.168.1.1 255.255.255.0
$router1_0 interface $router2_1 192.168.101.1 255.255.255.0
$router2_1 interface $router1_0 192.168.101.2 255.255.255.0
$router2_1 interface $router3_2 192.168.102.2 255.255.255.0
$router3_2 interface $router1_0 192.168.103.2 255.255.255.0
$router3_2 interface $auto_node_0 192.168.3.1 255.255.255.0
$router3_2 interface $router2_1 192.168.102.1 255.255.255.0
$auto_node_0 interface $router3_2 192.168.3.2 255.255.255.0
$auto_node_1 interface $router1_0 192.168.1.2 255.255.255.0

$router1_0 access-list "101 permit tcp 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255"

$router1_0 ip-access-group [$router3_2 id] 101 out

$router1_0 debug-acl

$ns at 0.0 "$router1_0 Static route 192.168.3.0 255.255.255.0 192.168.101.2 200"
```

```
$ns at 0.0 "$router2_1 Static route 192.168.1.0 255.255.255.0 192.168.101.1"
$ns at 0.0 "$router2_1 Static route 192.168.3.0 255.255.255.0 192.168.102.1"
$ns at 0.0 "$router3_2 Static route 192.168.1.0 255.255.255.0 192.168.102.2 200"

$ns at 0.0 "$auto_node_0 Static route 0.0.0.0 0.0.0.0 192.168.3.1"
$ns at 0.0 "$auto_node_1 Static route 0.0.0.0 0.0.0.0 192.168.1.1"

$ns at 0.0 "$router1_0 RIP network 192.168.1.0"
$ns at 0.0 "$router1_0 RIP network 192.168.103.0"
$ns at 0.0 "$router3_2 RIP network 192.168.3.0"
$ns at 0.0 "$router3_2 RIP network 192.168.103.0"

set tcp1 [new Agent/TCP]
$ns attach-agent $auto_node_1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $auto_node_0 $sink1
$ns connect $tcp1 $sink1
$tcp1 set src_addr_ [$tcp1 quad-to-value 192.168.1.2]
$tcp1 set dst_addr_ [$tcp1 quad-to-value 192.168.3.2]
$sink1 set src_addr_ [$sink1 quad-to-value 192.168.3.2]
$sink1 set dst_addr_ [$sink1 quad-to-value 192.168.1.2]
$tcp1 set fid_ 1
$sink1 set fid_ 1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

set udp [new Agent/UDP]
$ns attach-agent $auto_node_1 $udp
set null [new Agent/Null]
$ns attach-agent $auto_node_0 $null
$ns connect $udp $null
$udp set fid_ 2
$udp set src_addr_ [$udp quad-to-value 192.168.1.2]
$udp set dst_addr_ [$udp quad-to-value 192.168.3.2]

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetSizet_ 500000
$cbr set rate_ 1Mb
$cbr set random_ false

$ns at 0.1 "$ftp1 start"
$ns at 0.1 "$cbr start"
$ns at 1.0 "$cbr stop"
$ns at 1.0 "$ftp1 stop"

$ns rtmodel-at 1.0 down $router1_0 $router3_2

$ns at 1.0 "$ftp1 start"
$ns at 1.0 "$cbr start"
$ns at 2.0 "$cbr stop"
$ns at 2.0 "$ftp1 stop"

$ns rtmodel-at 2.0 up $router1_0 $router3_2

$ns at 2.01 "finish"
$ns run
```

Příloha 5

Konfigurační soubory směrovačů z příkladu v jedenácté kapitole a OTcl skript simulace. Veškeré zde vypsané soubory jsou obsaženy v přiloženém CD v adresáři examples\11.

Router1 / router1.txt

```
!
!version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Router1
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
ip cef
!
!
!
voice-card 0
!
!
key chain testing
 key 1
   key-string test1
!
!
!
interface FastEthernet0/0
 ip address 192.168.1.1 255.255.255.0
 ip rip authentication key-chain testing
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.5.1 255.255.255.0
 duplex auto
 speed auto
!
interface Serial0/1/0
 ip address 192.168.3.1 255.255.255.0
!
interface Serial0/1/1
 no ip address
 shutdown
 clock rate 125000
!
interface BRI0/2/0
 no ip address
 encapsulation hdlc
 shutdown
!
router rip
 version 2
 network 192.168.1.0
 network 192.168.5.0
!
ip forward-protocol nd
ip route 192.168.2.0 255.255.255.0 192.168.3.2 200
!
!
ip http server
no ip http secure-server
!
!
```

```
!
!
!
control-plane
!
!
!
line con 0
line aux 0
line vty 0 4
 login
!
scheduler allocate 20000 1000
end
```

Router2 / router2.txt

```
!
!version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Router2
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
ip cef
!
!
!
voice-card 0
!
!
key chain testing
 key 1
   key-string test1
!
!
!
interface FastEthernet0/0
 ip address 192.168.1.2 255.255.255.0
 ip rip authentication key-chain testing
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.2.1 255.255.255.0
 ip access-group 101 in
 duplex auto
 speed auto
!
interface Serial0/1/0
 no ip address
 shutdown
 clock rate 125000
!
interface Serial0/1/1
 ip address 192.168.4.1 255.255.255.0
!
interface BRI0/2/0
 no ip address
 encapsulation hdlc
 shutdown
!
interface Serial0/3/0
 no ip address
 shutdown
```

```
 clock rate 125000
!
interface Serial0/3/1
 no ip address
 shutdown
 clock rate 125000
!
router rip
 version 2
 network 192.168.1.0
 network 192.168.2.0
!
ip forward-protocol nd
ip route 192.168.5.0 255.255.255.0 192.168.4.2 200
!
!
ip http server
no ip http secure-server
!
access-list 101 permit tcp 192.168.2.0 0.0.0.255 192.168.5.0 0.0.0.255 eq 3389
access-list 101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.1.1 eq www
access-list 101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.4.2 eq www
access-list 101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.2.1 eq www
access-list 101 permit icmp any any
!
!
!
control-plane
!
!
!
line con 0
line aux 0
line vty 0 4
 login
!
scheduler allocate 20000 1000
end
```

Backup / backup.txt

```
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Backup
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
ip cef
!
!
!
voice-card 0
!
!
!
interface FastEthernet0/0
 no ip address
 shutdown
 duplex auto
 speed auto
!
interface FastEthernet0/1
 no ip address
 shutdown
```

```
 duplex auto
 speed auto
!
interface Serial0/1/0
 ip address 192.168.3.2 255.255.255.0
 no fair-queue
 clock rate 128000
!
interface Serial0/1/1
 ip address 192.168.4.2 255.255.255.0
 clock rate 128000
!
interface BRI0/2/0
 no ip address
 encapsulation hdlc
 shutdown
!
ip forward-protocol nd
ip route 0.0.0.0 0.0.0.0 192.168.3.1
ip route 192.168.2.0 255.255.255.0 192.168.4.1
!
!
ip http server
no ip http secure-server
!
!
!
control-plane
!
!
!
line con 0
line aux 0
line vty 0 4
 login
!
scheduler allocate 20000 1000
end
```

Vygenerovaný OTcl skript s úpravou testové komunikace / sim.tcl

```
set ns [new Simulator]
$ns enable-acl
$ns IPv4 ON

$ns color 1 Red
$ns color 2 Blue
$ns color 3 Green


set file1 [open out.nam w]
$ns namtrace-all $file1

proc finish {} {
   global ns file1
   $ns flush-trace
   close $file1
   exec nam out.nam &
   exit 0
}

set auto_node_0 [$ns node]
$auto_node_0 label auto_node_0
$auto_node_0 hostname auto_node_0
$ns at 0.0 "$auto_node_0 color green"
set auto_node_1 [$ns node]
$auto_node_1 label auto_node_1
$auto_node_1 hostname auto_node_1
$ns at 0.0 "$auto_node_1 color green"


set router1_0 [$ns node]
$router1_0 label router1_0
$router1_0 hostname router1_0
$ns at 0.0 "$router1_0 color black"
set router2_1 [$ns node]
$router2_1 label router2_1
$router2_1 hostname router2_1
$ns at 0.0 "$router2_1 color black"
set backup_2 [$ns node]
$backup_2 label backup_2
$backup_2 hostname backup_2
$ns at 0.0 "$backup_2 color black"

$ns rtproto RIP $router1_0 $router2_1
$ns rtproto StaticIPv4

$ns duplex-link $backup_2 $router2_1 128000b 2ms DropTail
[[$ns link $backup_2 $router2_1] acl_in] drop-target [[$ns link $router2_1 $backup_2] drophead]
[[$ns link $router2_1 $backup_2] acl_in] drop-target [[$ns link $backup_2 $router2_1] drophead]
$ns duplex-link $backup_2 $router1_0 128000b 2ms DropTail
[[$ns link $backup_2 $router1_0] acl_in] drop-target [[$ns link $router1_0 $backup_2] drophead]
[[$ns link $router1_0 $backup_2] acl_in] drop-target [[$ns link $backup_2 $router1_0] drophead]
$ns duplex-link $router2_1 $router1_0 104857600b 2ms DropTail
[[$ns link $router2_1 $router1_0] acl_in] drop-target [[$ns link $router1_0 $router2_1] drophead]
[[$ns link $router1_0 $router2_1] acl_in] drop-target [[$ns link $router2_1 $router1_0] drophead]
$ns duplex-link $router1_0 $auto_node_0 104857600b 2ms DropTail
[[$ns link $router1_0 $auto_node_0] acl_in] drop-target [[$ns link $auto_node_0 $router1_0] drophead]
[[$ns link $auto_node_0 $router1_0] acl_in] drop-target [[$ns link $router1_0 $auto_node_0] drophead]
$ns duplex-link $router2_1 $auto_node_1 104857600b 2ms DropTail
[[$ns link $router2_1 $auto_node_1] acl_in] drop-target [[$ns link $auto_node_1 $router2_1] drophead]
[[$ns link $auto_node_1 $router2_1] acl_in] drop-target [[$ns link $router2_1 $auto_node_1] drophead]

$router1_0 interface $router2_1 192.168.1.1 255.255.255.0
$router1_0 interface $auto_node_0 192.168.5.1 255.255.255.0
$router1_0 interface $backup_2 192.168.3.1 255.255.255.0
$router2_1 interface $router1_0 192.168.1.2 255.255.255.0
$router2_1 interface $auto_node_1 192.168.2.1 255.255.255.0
$router2_1 interface $backup_2 192.168.4.1 255.255.255.0
$backup_2 interface $router1_0 192.168.3.2 255.255.255.0
```

```
$backup_2 interface $router2_1 192.168.4.2 255.255.255.0
$auto_node_0 interface $router1_0 192.168.5.2 255.255.255.0
$auto_node_1 interface $router2_1 192.168.2.2 255.255.255.0

$router2_1 access-list "101 permit tcp 192.168.2.0 0.0.0.255 192.168.5.0 0.0.0.255 eq 3389"
$router2_1 access-list "101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.1.1 eq www"
$router2_1 access-list "101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.4.2 eq www"
$router2_1 access-list "101 permit tcp 192.168.2.0 0.0.0.255 host 192.168.2.1 eq www"
$router2_1 access-list "101 permit icmp any any"

$router2_1 ip-access-group [$auto_node_1 id] 101 in
$router2_1 debug-acl
$ns at 4.0 "$router2_1 show-acl short"

$ns at 0.0 "$router1_0 Static route 192.168.2.0 255.255.255.0 192.168.3.2 200"
$ns at 0.0 "$router2_1 Static route 192.168.5.0 255.255.255.0 192.168.4.2 200"
$ns at 0.0 "$backup_2 Static route 0.0.0.0 0.0.0.0 192.168.3.1"
$ns at 0.0 "$backup_2 Static route 192.168.2.0 255.255.255.0 192.168.4.1"

$ns at 0.0 "$auto_node_0 Static route 0.0.0.0 0.0.0.0 192.168.5.1"
$ns at 0.0 "$auto_node_1 Static route 0.0.0.0 0.0.0.0 192.168.2.1"

$ns at 0.0 "$router1_0 RIP network 192.168.1.0"
$ns at 0.0 "$router1_0 RIP network 192.168.5.0"
$ns at 0.0 "$router1_0 RIP version 2"
$ns at 0.0 "$router1_0 RIP authentication key-chain testing $router2_1"
$router1_0 keychain testing
$router1_0 key testing 1
$router1_0 key-string testing 1 test1
$ns at 0.0 "$router2_1 RIP network 192.168.1.0"
$ns at 0.0 "$router2_1 RIP network 192.168.2.0"
$ns at 0.0 "$router2_1 RIP version 2"
$ns at 0.0 "$router2_1 RIP authentication key-chain testing $router1_0"
$router2_1 keychain testing
$router2_1 key testing 1
$router2_1 key-string testing 1 test1

set tcp1 [new Agent/TCP]
$ns attach-agent $auto_node_1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $auto_node_0 $sink1
$ns connect $tcp1 $sink1
$tcp1 set acl_dst_port_ 3389
$tcp1 set src_addr_ [$tcp1 quad-to-value 192.168.2.2]
$tcp1 set dst_addr_ [$tcp1 quad-to-value 192.168.5.2]
$sink1 set src_addr_ [$sink1 quad-to-value 192.168.5.2]
$sink1 set dst_addr_ [$sink1 quad-to-value 192.168.2.2]
$tcp1 set fid_ 1
$sink1 set fid_ 1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

set udp1 [new Agent/UDP]
$ns attach-agent $auto_node_1 $udp1
set null1 [new Agent/Null]
$ns attach-agent $auto_node_0 $null1
$ns connect $udp1 $null1
$udp1 set fid_ 2
$udp1 set src_addr_ [$udp1 quad-to-value 192.168.2.2]
$udp1 set dst_addr_ [$udp1 quad-to-value 192.168.5.2]

set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSizet_ 500
$cbr1 set rate_ 64000b
$cbr1 set random_ false

set udp2 [new Agent/UDP]
$ns attach-agent $auto_node_0 $udp2
```

```
set null2 [new Agent/Null]
$ns attach-agent $auto_node_1 $null2
$ns connect $udp2 $null2
$udp2 set fid_ 3
$udp2 set src_addr_ [$udp2 quad-to-value 192.168.5.2]
$udp2 set dst_addr_ [$udp2 quad-to-value 192.168.2.2]

set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
$cbr2 set packetSizet_ 500
$cbr2 set rate_ 64000b
$cbr2 set random_ false

$ns at 0.1 "$ftp1 start"
$ns at 0.1 "$cbr1 start"
$ns at 0.1 "$cbr2 start"

$ns rtmodel-at 1.0 down $router1_0 $router2_1
$ns rtmodel-at 2.0 up $router1_0 $router2_1

$ns at 4.0 "$ftp1 stop"
$ns at 4.0 "$cbr1 stop"
$ns at 4.0 "$cbr2 stop"

$ns at 4.0 "finish"
$ns run
```