VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÍ SIMULÁTORU OMNET++ O FILTROVACÍ PRAVIDLA ACL

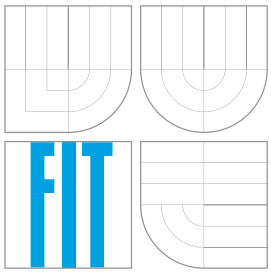
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ SUCHOMEL

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÍ SIMULÁTORU OMNET++ O FILTROVACÍ PRAVIDLA ACL

OMNET++ EXTENSION WITH ACL FILTERING MODULE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ SUCHOMEL
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR MATOUŠEK, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

V bakalářské práci se zabýváme diskrétní simulací sítě v simulačním nástroji OMNeT++. Prozkoumáváme možnosti efektivní reprezentace a vyhodnocování ACL pomocí pokročilých struktur na bázi intervalových rozhodovacích diagramů. Simulátor OMNeT++ rozšiřujeme o modul filtrování paketů pomocí seznamů ACL, jehož návrh i implementace je zde popsána. Praktické využití implementovaného rozšíření je ukázáno na simulaci reálné netriviální sítě, kde ověřujeme výsledky simulace a porovnáváme je s s chováním skutečného prostředí.

Abstract

This bachelor's thesis describes discrete simulation of network in OMNeT++. We are exploring effective representation and evaluation of ACL rules by advanced data structures based on interval decision diagrams. OMNeT++ is extended by filtering properties of packets using access control lists. Because ACL filtering is not supported in OMNeT++, it was added as a brand-new module, whose concept and implementation is described here. Practical usage of the implemented module is demonstrated on a simulation of real nontrivial network. We also analyse results of the simulation and verify them by comparison with real network behaviour.

Klíčová slova

Access control list, ACL, OMNeT++, INET Framework, diskrétní simulace, bezpečnost sítě, IDD

Keywords

Access control list, ACL, OMNeT++, INET Framework, discrete simulation, network security, IDD

Citace

Tomáš Suchomel: Rozšíření simulátoru OMNeT++ o filtrovací pravidla ACL, bakalářská práce, Brno, FIT VUT v Brně, 2009

Rozšíření simulátoru OMNeT++ o filtrovací pravidla ACL

Prohlášení

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

.....
Tomáš Suchomel
14. května 2009

Poděkování

Děkuji Ing. Petru Matouškovi, Ph.D. za neocenitelné rady při psaní mé práce, ochotu a permanentní časovou dostupnost. Dále bych rád poděkoval kolegům z výzkumného projektu ANSA za mnohé významné rady a nápady, ale zejména za to, že tvořili skvělý kolektiv nejen na četných konzultacích. V neposlední řadě děkuji své rodině, přátelům a přítelkyni za podporu v těžkých chvílích, kdy jsem nevěděl, kudy dál jít. Ing. Petru Novotnému a ostatním spolupracovníkům za toleranci.

Obsah

Úvod

V dnešní době při budování nových a zejména pak upravování stávajících datových sítí je nutno dbát na zvyšující se požadavky a nároky na stabilitu, přenosovou rychlost, šířku průtokového pásma (throughput) a v neposlední řadě také na bezpečnost. Ač si to mnozí neuvědomují, bezpečnost počítačové sítě je v mnoha ohledech nedostatečná.

Při návrhu počítačové sítě obvykle předem neznáme všechna kritéria, která bude třeba dodržet, ani všechny případné překážky, které mohou nastat jak při zavádění sítě, tak až při samotné konfigurace vytvořené sítě. Mnohdy zjistíme, že požadavky na síť po prvotní analýze nejsou dostatečné pro reálný provoz sítě. Je neekonomické, a někdy dokonce i nemožné (například vzhledem k podmínkám prostředí), testovat nově budovanou síť na reálném síťovém hardwaru, a stejně tak rozšiřování současných sítí je často nežádoucí, protože by se nějakým způsobem muselo zasahovat do existující sítě za běhu. Je tedy nasnadě se poohlédnout po kvalitním řešení, které by tyto problémy s analýzou vyřešilo za nás a přitom neohrozilo či nežádoucím způsobem neovlivnilo (výpadky při zkoušení nových technologií, chyby zabezpečení při rekonfiguraci firewallu, apod.) síť, kterou chceme modifikovat.

Z výše uvedených důvodů vidíme, že je velmi vhodné, ne-li přímo nutné, se bezprostředně po analýze a návrhu počítačové sítě zaměřit na její simulaci.

Cílem této bakalářské práce je přidat do síťového simulátoru OMNeT++ modul pro filtrování paketů ACL, abychom se při simulaci přiblížili vlastnostem reálné sítě a byli schopni na základě konfigurace firewallu vyhodnocovat data proudící sítí.

Kapitola 1 uvede simulaci do kontextu počítačových sítí, seznámí nás s existujícími simulačními nástroji a na základě jejich analýzy pak vybere ten nejvhodnější. Kapitola 2 má za cíl představit nám síťový simulátor OMNeT++, do něhož budeme integrovat filtrovací modul. Konzultuje jeho klady i nedostatky a sděluje zkušenosti s instalací a zprovozněním. Dále nás seznámí s prostředím nástroje, simulacemi a nakonec nás provede jeho konfigurací. V kapitole 3 se teoreticky seznámíme s ACL v kontextu bezpečnosti počítačových sítí, blíže se podíváme na princip jejich fungování a možnostmi jejich použití. Uděláme si také přehled různých typů ACL, přičemž se zaměříme zejména na IP ACL, které jsou předmětem našeho zkoumání. Dále zmíníme také wildcard adresy a způsob, jakým se pomocí nich porovnává IP adresa. Následně v kapitole 4 budeme zkoumat efektivní metodu klasifikace paketů, podíváme se na způsob reprezentace filtrovacích pravidel pomocí rozhodovacích diagramů, konkrétně IDD. Dozvíme se zde, jak se dá seznam filtrovacích pravidel transformovat na predikátovou logiku a také proč jsou IDD optimální datovou strukturou pro reprezentaci ACL. Kapitola 5 popisuje převodní nástroj a strukturu globálního konfiguračního XML souboru. Vlastní řešení implementace modulu je potom podrobně popsáno v kapitole 6, stejně jako řešení problémů s implementací spjatých. V poslední kapitole 7 testujeme implementaci a zaměříme se na výsledky simulace. Také zde porovnáme výsledky simulace s reálným stavem. Na závěr diskutujeme dosažené výsledky, hodnotíme přínos práce a zvažujeme její možné rozšíření do budoucna.

Kapitola 1

Simulace počítačové sítě

Kapitola se zabývá simulacemi v oblasti počítačových sítí. Nejprve definujeme diskrétní simulaci, potom si stanovíme kritéria pro výběr vhodného simulačního nástroje, následně se seznámíme s několika dostupnými simulačními nástroji. Tyto simulátory srovnáme, uvedeme jejich výhody a nevýhody.

1.1 Diskrétní simulace

Simulace má za cíl analyzovat chování systému podle dostupných hodnot parametrů, které získáme popisem systému. Obecně platí, že nejprve musíme vytvořit abstraktní model systému. Implementací abstraktního modelu získáme simulační model, se kterým už můžeme experimentovat. Experimentování s modelem je velice důležité, simulace bez vhodně zvolených experimentů nemá valný význam. Simulace nemůže nikdy obsáhnout chování celého reálného systému, proto se musíme rozhodnout, které prvky můžeme zanedbat. Obecně platí, že simulace je tím bližší reálnému stavu, čím důkladnější a složitější je vytvořený model.

Diskrétní simulace jsou charakteristické tím, že se proměnné v modelu mění skokově (nespojitě) pouze tehdy, nastala-li určitá událost. Mohou mít diskrétní i spojitý čas, avšak v počítačové simulaci se spojitý čas převádí na diskrétní a vhodně se zvolí malý simulační krok. Simulace počítačové sítě řadíme mezi diskrétní simulace.

1.2 Výběr simulačního nástroje

Simulace počítačové sítě musí splňovat určité nároky. Cílem námi vytvořené simulace je sledovat v čase cesty jednotlivých paketů skrze počítačovou síť. Do kategorie požadavků na simulaci spadá přesné vytvoření modelu topologie sítě (např. počet a typy aktivních prvků, tj. směrovačů, rozbočovačů apod.), možnost konfigurace těchto zařízení co nejpodobnější reálnému stavu a možnostem, dále pak možnost nastavení obecně reakcí na určité situace - nástroj musí být schopen odsimulovat například pád linky, ztrátu paketu, filtrování paketů (firewall), atp. Simulátor by měl být schopen do vytvořeného modelu sítě ručně vložit nebo automaticky načíst parametry simulace pro jednotlivé uzly, rozhraní a pakety, a na základě těchto informací zprostředkovat simulaci chování podobnou reálné síti.

1.3 Simulační nástroje

Z pohledu síťových simulátorů se seznámíme nejen s nástrojem OMNeT++, ale i s dalšími, přičemž je vzájemně porovnáme a uvedeme si jejich výhody a nevýhody. Na OMNeT++ bude kladen největší důraz, je mu také vyhrazena celá příští kapitola 2.

1.3.1 Network Simulator - NS2

NS2 je diskrétní simulátor počítačových sítí. Vznikl už v roce 1989 jako projekt REAL NS a podstatně se vyvinul. Na rozvoji se podílí jak komerční i nekomerční organizace, tak jednotlivci. NS2 je objektově orientovaný diskrétní simulátor řízený událostmi, jehož jádro je naprogramováno v jazyce C++. Jazyk C++ slouží také k manipulaci s daty. Jazyk OTcl slouží pro popis a programování simulačního modelu, obecně pro řízení celé simulace. NS2 umožňuje mimo jiné vystavět model sítě, generovat provoz v síti a také chyby. Základní stavební jednotkou síťové topologie je uzel (node) a linka (link), která dva uzly vzájemně propojuje. Na linkách se dají nastavit vlastnosti jako zpoždění (delay), šířka pásma (bandwidth), fronty. Podpora zahrnuje mimo jiné unicast i multicast směrování, drátové i bezdrátové sítě.[10]

Výhody a nevýhody NS2

Mezi výhody můžeme rozhodně zařadit možnost přidávat vlastní moduly, dále NS2 je volně šiřitelný software. Nevýhodami jsou: absence IPv4 adresace a směrování, slabá podpora multicastu, NS2 nemá vlastní vizuální výstup (nutnost vizualizace přes NAM, který je navíc nepřehledný), celková složitost.

1.3.2 CiscoPT: Packet Tracer 5.0

Packet Tracer je poměrně kvalitní simulátor sítě od společnosti Cisco Systems, který slouží hlavně pro výuku a experimentování. Má kvalitní grafický výstup, propracované ovládání (GUI), které se graficky podobá výukovým materiálům. Topologie sítě a vůbec vytváření simulačního modelu se provádí metodou drag and drop, kde uživatel má k dispozici různá síťová zařízení firmy Cisco. PT5 obsahuje i emulaci konzole, pomocí níž se jednotlivá zařízení konfigurují. Vzhledem k tomu, že jako absolvent akademie CCNA s ním mám zkušenosti, rozhodně ho mohu doporučit, je jednoduchý a praktický.

Výhody a nevýhody PT5

U Packet Traceru začneme jednoznačnou nevýhodou, a totiž, že se jedná o uzavřený nástroj společnosti Cisco Systems, tudíž není možné rozšiřovat funkčnost přidáním vlastních modulů, program ani není volně šiřitelný. Hlavní výhodou je potom především jednoduchost (a to jak používání, tak i instalace) a intuitivní ovládání, které nemají problém pochopit ani začátečníci či nováčci v CCNA akademii. Potěší i elegantní grafické rozhraní a velkým plus je emulace příkazové konzole podporující příkazy CISCO zařízení.

1.3.3 OMNeT++

Za vhodný simulační nástroj jsme si vybrali OMNeT++. Ten má celkem solidní uživatelskou základnu (community wiki, forum, bugtracker) a velmi obsáhlou, a troufnu si říct i užitečnou, dokumentaci. Zjistili jsme, že budeme potřebovat ještě nástavbu snad OMNeTem, a

tou je rozhraní INET Framework. Ten je vhodný především pro simulování drátových, bezdrátových a ad-hoc sítí. Kromě IP (podporuje dokonce i IPv6) a TCP/UDP obsahuje také simulaci 802.11, PPP, OSPF, RIP a jiné standardy a protokoly. Více k systému OMNeT++ najdete v kapitole 3.

Kapitola 2

OMNeT++

Tato kapitola se věnuje síťovému simulačnímu software OMNeT++. Tento nástroj máme rozšířit o filtrovací modul ACL, proto je vhodné se s ním krátce seznámit. Nejprve si charakterizujeme hlavní rysy simulátoru a shrneme jeho výhody a nevýhody. Dále zde uvedeme naše zkušenosti s instalací, seznámíme se s prostředím OMNeT++, popíšeme konfiguraci simulace. Také si zde zdůvodníme výběr konkrétní verze nástroje a budeme diskutovat vývojovou platformu.

2.1 Úvod

Požadavky na simulační nástroj z kapitoly 1.2 splňuje simulační systém diskrétních událostí - OMNeT++. Je to velmi obecný a flexibilní simulační nástroj, my se však zabýváme simulací počítačových sítí, takže v dalším textu budu OMNeT++ popisovat pouze jako síťový simulační nástroj - to je také jeho primární účel. Lze jej však použít pro simulaci obecně libovolného distribuovaného systému (komplexní IT sítě, sítě na bázi front) a klidně také pro simulování hardwarové architektury. Ne nadarmo se díky jeho univerzálnosti a širokým možnostem konfigurace stal velice oblíbeným a rozšířeným simulačním nástrojem síťařské komunity.

2.2 Charakteristika OMNeT++

OMNeT++ je objektově orientovaný simulátor diskrétních systémů, jehož hierarchická struktura je složena z jednotlivých komponent (modulů) - jedná se tedy o systém modulární. S velkou hloubkou zanoření modulů lze popsat téměř libovolný simulační model. Jednotlivé komponenty (objekty) jsou naprogramovány v C++, celková struktura vznikne skládáním modulů do jednoho zapouzdřeného celku. Pro tento účel se používá jazyk vysoké úrovně, NED. Nástroj disponuje silným GUI (grafickým rozhraním) a je poměrně rychlý.[12]

2.2.1 Výhody a nevýhody

Z výhod budeme jmenovat zejména obecnost, modulárnost, kvalitní vizualizaci, dále OMNeT++ je volně šiřitelný software, takže není problém do něj přidat další moduly. Nevýhodou oproti například NS2 je obtížné řízení již probíhající simulace.

2.3 Vývoj v rámci projektu ANSA

Náš modul vyvíjíme v rámci projektu NeSim (Network Simulation and Analysis) pod hlavičkou projektu ANSA[2]. Projekt se zabývá analýzou a odhadováním chování sítě užitím simulací vyvíjených v simulátoru OMNeT++. Topologie sítě je postavena na uzlech (např. směrovačích) a linkách. Uzly obsahují rozhraní s IP adresami, filtrovacími pravidly v seznamech ACL, směrovacími procesy apod. Na základě změn v konfiguraci sítě (padající linky aj.) zkoumáme a analyzujeme dynamické chování takových sítí. Cílem je odhalit slabá místa v konfiguraci a designu sítě.

2.4 Potřeba zavedení standardů

Vzhledem k množství dostupných verzí nástroje bylo potřebné stanovit standard, který bude používat celý náš vývojový tým ANSA[2]. Se souhlasem vedoucího práce jsme se shodli, že budeme používat OMNeT++ ve verzi 3.3 a INET Framework, verze 20061020. Vývojovou platformu jsme zatím ponechali volnou, avšak rozhodli jsme se náš projekt (modul) vyvíjet pod systémem Linux, konkrétně Kubuntu verze 9.04. Nutno dodat, že neúspěšně skončil můj pokus rozchodit OMNeT++ pod systémem MS Windows Vista x64 i MS Windows XP SP3.

Po další společné diskuzi jsme ovšem naznali, že je nový OMNeT++ ve verzi 4.0 mnohem více propracovanější, a tak proběhl pokus nově vytvořit vývojové prostředí stejné pro celý náš ANSA[2] projekt. Použili jsme tedy OMNeT++ 4.0 a k němu i novou verzi INET-MANET Framework. Velkou výhodou je, že do OMNeTu je integrováno dle mého názoru výborné vývojové prostředí IDE Eclipse, které má mezi jinými například dobrou podporu pro týmovou práci (SVN repository, možnost sdílet jeden projekt v rámci vývojového týmu). Odladění a přechod na verzi 4.0 a nový INET-MANET se všem zdařil a práce mohla začít.

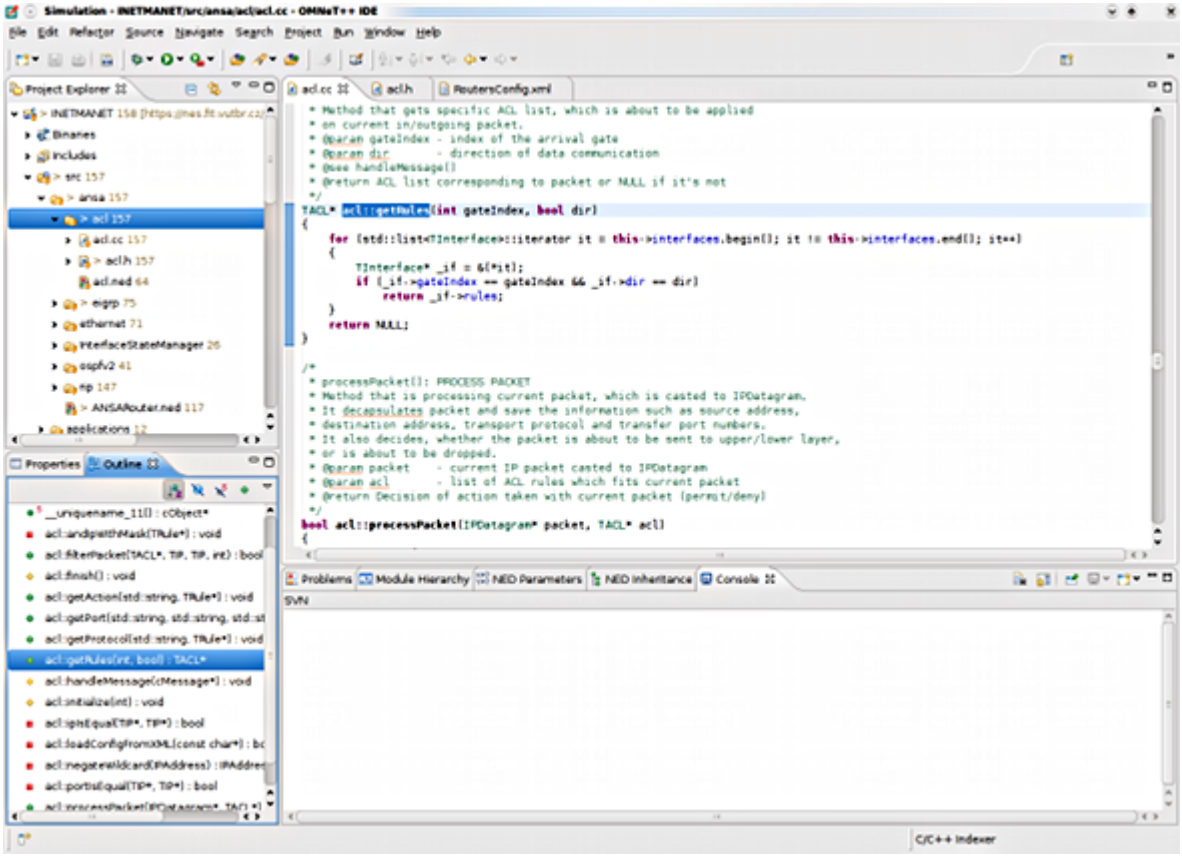
2.5 Překlad

Po stažení a úspěšném překladu zdrojových souborů OMNeT++ 4.0 bylo potřeba nastavit v PATH cestu k prostředkům OMNeT++ (../omnetpp/bin) pomocí příkazu export PATH. Dále bylo nutné zkompilovat rozhraní INET-MANET Framework. Postupovali jsme podle návodu v dokumentaci, a i přesto, že celá procedura trvala 25 minut, nezaznamenali jsme žádné problémy a zkušební odsimulování dvojice OSPFRouterů proběhlo napoprvé v pořádku. Nutno podotknout, že celý OMNeT++ se nainstaloval včetně vývojového prostředí IDE Eclipse, které dokonce podporuje i přímé spouštění simulací, debugování apod.

2.6 Seznámení s prostředím, vyzkoušení simulací

Po úspěšné instalaci jsme se seznámili s prostředím OMNeT++, odsimulovali zajímavě řešené demo aplikace a interaktivní příklady k procvičení a seznámení se s jazykem NED i s platformou C++. Následně jsme si vyzkoušeli rozsáhlejší simulace v INETu -> Examples, díky nimž jsme pronikli do tajů tohoto mocného simulačního nástroje, zjistili jak fungují jednotlivé linky/interface - tedy například to, že hodnotu „connected“ u linky routovací proces nečte, pouze ji zpracuje na samotném začátku simulace, k zjišťování stavu linky up/down je tedy třeba použít jiný vhodný způsob - zamyšlení nad Scenario Managerem. Ten se ukázal být vhodným, fungování bude na principu zasílání zpráv o změně stavu linky

s identifikací linky (router/interface) a stavu (up/down). Jak routovací proces, tak interface table i samotný interface musí být schopny rozpoznat tuto změnu a reagovat na ni. Dále jsme se seznámili s IDE Eclipse, integrovali do něj náš nově vytvořený ANSA projekt OpenRouter přes subversion systém SVN, který jsme za tímto účelem vytvořili. Odzkoušeli jsme si spouštění simulací, a to jak z příkazové řádky, tak přímo přes prostředí Eclipse. Druhou variantu můžeme doporučit. Na obrázku 2.1 vidíme ukázku prostředí Eclipse v simulátoru OMNeT++.



Obrázek 2.1: Prostředí IDE Eclipse v OMNeT++

2.6.1 Konfigurace simulace v OMNeT++

Základem pro úspěšné nakonfigurování parametrů simulace je soubor *omnetpp.ini*, bez jeho přítomnosti nelze simulaci zahájit. Konfigurační soubor je v textovém formátu, v němž jsou bloky příkazů rozděleny do logických sekcí. Ze souboru se načítají nastavení simulace, například které sítě se mají odsimulovat, nastavení daných sítí, `.ned` soubory, které se mají načíst atd.

Syntax konfiguračního souboru

Jak již bylo řečeno, konfigurační soubor je rozdělen do jednotlivých sekcí, které jsou označeny hranatými závorkami. Komentáře začínají na řádku znakem `#`. Celková velikost souboru není omezená, je však omezená délka jednoho řádku v konfiguraci, a to je 1024 znaků

na řádek, přičemž řádek může být rozdělený znakem \. Ukázka konfiguračního souboru *omnetpp.ini* pro fungující simulaci je na obrázku 2.2.

```
# OSPFv2 + RIP with ACL: test network
# ZAKLADNI NASTAVENI SIMULACE
[General]
description = "Simple test"
network = SimpleTest
tkenv-plugin-path = ../../../etc/plugins
# NASTAVENI pro smerovani RIP
**.rip.localPort = 520
**.rip.destPort = 520
# NASTAVENI GLOBALNIHO KONFIGURACNIHO SOUBORU XML PRO MODULY SIMULACE
**.rip.configFile = "RoutersConfig.xml"
**.ospf.configFile = "RoutersConfig.xml"
**.acl.configFile = "RoutersConfig.xml"
# SCENARIO MANAGER PRO SIMULACI
**.scenarioManager.script = xmldoc("scenario.xml")
# KONFIGURACE SMEROVACU
**.R1.routingFile = "R1.irt"
**.R2.routingFile = "R2.irt"
**.R3.routingFile = "R3.irt"
# KONFIGURACE POCITACU
**.H1.routingFile = "H1.irt"
**.H2.routingFile = "H2.irt"
# GENEROVANI ICMP PAKETU Z H1 na H2 a naopak v intervalu 3s
**.pingApp.packetSize = 256B
**.pingApp.interval = 3s
**.pingApp.printPing = true
**.H1.pingApp.destAddr = "192.168.2.2"
**.H2.pingApp.destAddr = "192.168.1.2"
# GENEROVANI UDP PAKETU z H1 a H2, zdrojovy port 25, cilovy port 6112
# frekvence posilani UDP paketu je 1 sekunda
**.numUdpApps = 1
**.udpAppType = "UDPBasicApp"
**.udpApp[0].localPort = 25
**.udpApp[0].destPort = 6112
**.udpApp[0].messageLength = 32 bytes
**.udpApp[0].messageFreq = 1s
**.H2.udpApp[0].destAddresses = "192.168.1.2"
**.H1.udpApp[0].destAddresses = "192.168.2.2"
```

Obrázek 2.2: Konfigurační soubor *omnetpp.ini*

Kapitola 3

Access control list, ACL

V této kapitole se teoreticky seznámíme s ACL v kontextu bezpečnosti počítačových sítí, blíže se podíváme na princip jejich fungování a možnostmi jejich použití. Uděláme si také přehled různých typů ACL, přičemž se zaměříme zejména na IP ACL, které jsou předmětem našeho zkoumání. Dále zmíníme také wildcard adresy a způsob, jakým se pomocí nich porovnává IP adresa. Na závěr kapitoly si řekneme, jak aktivujeme ACL na rozhraní routerů. ACL budeme načítat do simulátoru a dále je používat pro účely simulace.

3.1 Bezpečnost počítačových sítí obecně, firewall

Zabezpečení počítačové sítě je dnes - v době, kdy se snažíme o integrování stále více služeb v rámci datové sítě - jedna z nejvyšších priorit. Do oblasti bezpečnosti spadají fyzické, hardwarové i softwarové prostředky. Mezi nejpoužívanější bezpečnostní prostředky patří firewall.

Firewall je jednoúčelové zařízení, které slouží k řízení a zabezpečování veškerého síťového provozu. Jedná se v podstatě o kontrolní bod, který definuje pravidla pro komunikaci mezi sítěmi, které od sebe odděluje. Na základě těchto pravidel je potom schopen určitý typ komunikace povolit, nebo zakázat. Firewally můžeme rozdělit do několika kategorií:

- **Paketové filtry** - jedná se o nejstarší a také nejjednodušší typ firewallu (vyvinuto 1988), jehož funkce spočívá v tom, že má přesně definované z jaké adresy a portu, na jakou adresu a port může být doručen procházející paket. Paketová kontrola se tedy provádí na L3 a L4 vrstvě modelu síťové komunikace OSI. Kromě adresy a portu se vyhodnocuje typ protokolu. Výhodou paketových filtrů je především vysoká rychlost zpracování procházejících paketů, avšak nevýhodou je nízká úroveň vyhodnocování procházejících spojení a také přílišná jednoduchost formulace pravidel. V případě složitějších pravidel budou jejich definice příliš krkolomné nebo filtrování vyspělejších protokolů nebude možné vůbec (FTP, RPC, A/V streaming). V praxi se tento typ stále uplatňuje v místech, kde není potřeba taková přesnost nebo důkladnější analýza zpracovávaných dat, to znamená zejména vysokorychlostní přenosy velkého množství dat. K typickým představitelům paketových filtrů patří zejména tzv. ACL (Access Control Lists) ve starších verzích operačního systému IOS na Cisco routerech.
- **Stavové paketové filtry** - jsou rozšířenou variantou paketového filtru. Provádějí stejnou kontrolu, ale navíc si uchovávají informace o povolených spojeních (tzv. stav),

podle kterých se mohou rozhodovat, zda procházející pakety už patří mezi jednou povolená spojení, nebo zda musí projít rozhodovacím procesem. Výhoda spočívá v urychlení kontroly paketů již povolených spojení, dále stačí v pravidlech uvádět jen směr navázání spojení, a firewall sám povolí i pakety s odpovědí a u známých protokolů (FTP) spojení na dynamicky dohodnutých portech. Zásadním vylepšením pak je možnost vytvoření virtuálního stavu spojení u bezstavových protokolů (ICMP, UDP).

- **Aplikační brány** - někdy se také můžeme setkat s názvem proxy firewally. Byly zavedeny krátce po paketových filtrech a oproti nim zcela oddělily sítě, mezi které byly postaveny. Veškerá komunikace potom probíhá přes aplikační bránu pomocí dvou spojení - klient, který zahajuje spojení, se připojí na proxy, která příchozí spojení zpracuje a vytváří nové datové spojení k serveru na základě požadavku od klienta, novým klientem se potom stává aplikační brána. Veškerá výměna dat probíhá tedy výhradně na trase Klient - Proxy - Server. Server pošle data proxy, který je následně vrátí klientovi v rámci původního spojení. Aplikační brány fungují na sedmé (aplikační) vrstvě modelu OSI, automaticky provozují NAT (překlad adres). Výhodou je poměrně vysoká úroveň zabezpečení, nevýhodou oproti paketovým filtrům je vysoká náročnost na použitý hardware.[1]

3.2 ACL pod lupou

Jak jsme již naznačili v předchozí kapitole, ACL řadíme mezi paketové filtry. Směrovače jsou schopny zpracovávat ACL z paketů L3 i L4 vrstvy modelu OSI. ACL sestává ze sady pravidel, podle nichž jsou kontrolovány veškeré pakety procházející směrovačem. Právě na základě jejich porovnání s informacemi, které nese paket, se směrovač rozhodne, zda-li se paket pošle dále, nebo ho zahodí. Obecně se nejčastěji kontroluje:

- Zdrojová/cílová IP adresa sítě/zařízení v síti
- Zdrojový/cílový port
- typ protokolu
- typ ICMP požadavku

V routeru společnosti Cisco systems pak je ACL použitelný podle:

- rozhraní - ACL se aktivují pro jednotlivá rozhraní směrovače.
- směru komunikace (příchozí/odchozí), pravidla musí být definována zvlášť pro každý směr.
- typu protokolu - jeden ACL musí být definován pro každý typ protokolu, jelikož kontrola je prováděna na každém rozhraní pro každý povolený (aktivní) protokol.

Je tedy zřejmé, že pravidel je potřebné mít v ACL definováno skutečně velké množství, například pro směrovač o dvou aktivních rozhraních, na kterém jsou povoleny čtyři protokoly, bude nutné pro zakázání komunikace s určitou sítí mít definováno šestnáct ACL - dvě pro každé rozhraní zvlášť * dvě pro každý směr zvlášť * čtyři povolené protokoly.

3.2.1 Jak fungují ACL na směrovačích

Princip fungování ACL ve směrovači je následovný: V případě, že existuje pravidlo pro příchozí paket, paket se zkontroluje. Pokud je výsledek pozitivní, paket je vpuštěn do směrovače. V opačném případě je paket zahozen. Pokud ACL pro paket neexistuje, pak je implicitně vpuštěn. Jestliže je pro daný paket nalezena cesta, router ho přepošle na své odchozí rozhraní, jinak ho zahodí. Na odchozím rozhraní se potom paket znovu kontroluje stejnou metodikou jako u příchozího rozhraní. Směrovač však neuplatňuje ACL pravidla na pakety v odchozím směru, které sám vytvořil. Řekli jsme si, že ACL je složen ze sady pravidel. Při kontrole paketu se tato pravidla prochází od prvního zadaného až do posledního. V případě shody s prvním pravidlem se provede akce (poslání/zahození) a další pravidla se již neaplikují. Tím se šetří prostředky routeru a tím pádem také čas. Pokud se nenajde shoda s žádným z pravidel, provede se implicitní pravidlo - paket je zahozen.[5]

3.2.2 Wildcard maska

Cílem této podkapitoly je vysvětlit význam a princip fungování wildcard masky. Když kontrolujeme pakety ACL pravidly, jsou porovnány zejména statické hodnoty. Avšak to, jestli paket náleží určité síti, není možné určit jen na základě obyčejného porovnání. To samé platí i pro samotné porovnání IP adres. Pro tyto účely se využívá `wildcard maska`, která funguje na podobném principu jako síťová maska. Wildcard maska se od obyčejné masky sítě liší opačným uspořádáním jedniček a nul. Pokud ve wildcard masce nese bit hodnotu 0, je při porovnávání použit. Pokud nese hodnotu 1, je při porovnávání ignorován. Pokud chceme, aby se zdrojová nebo cílová adresa nekontrolovala vůbec, můžeme uvést libovolnou adresu a masku 255.255.255.255. Namísto toho je však vhodnější použít klíčové slovo `any`. Pokud naopak chceme, aby byla povolena `právě jedna` IP adresa, wildcard masku uvedeme 0.0.0.0, podle které se musí shodovat všechny bity adresy, zatímco efektivnější řešení je použít klíčové slovo `host`. Vše si ukažme na příkladech:
Pokud chceme, aby byly povoleny všechny IP pakety ze zdrojových adres s prefixem 192.168.100.0/24 na libovolnou cílovou adresu, musíme přidat do ACL následující záznam: `permit ip 192.168.100.0 0.0.0.255 any`, kde wilcard maska 0.0.0.255 znamená v bitovém zápisu:
00000000 00000000 00000000 11111111 - červeně jsou označeny bity masky, ve kterých se ip adresa může lišit. Tedy znamená to, že prvních 24 bitů IP je při porovnání kontrolováno, zatímco posledních 8 je ignorováno. Poslední čtvrtý oktet může nést jakékoli bity, tzn. hodnotu od 0 do 255.
Jestliže chceme, aby byly zakázány veškeré TCP spojení z 1jedné zdrojové adresy 192.168.1.33 na cílovou síť 10.60.20.128/25, musíme přidat do ACL tento záznam:
`deny tcp host 192.168.1.33 10.60.20.128 0.0.0.127}`.[6]

3.3 Typy ACL

Zde si ukážeme, jaké různé typy ACL existují, jaké jsou stěžejní z hlediska jejich uplatnění v reálné praxi a jaké jsou zajímavé pro implementaci do simulátoru OMNeT++. Seznámíme se s možnostmi kontroly paketu a uvedeme si přesné tvary příkazů, které bude potřeba zpracovat, což je nezbytné pro návrh i implementaci. Syntax příkazů bude taková, že cokoli mezi `{}` znamená povinný příkaz, mezi `[]` potom volitelný příkaz, a dělící znak | představuje logický výraz *nebo* - znamená to tedy, že na místě daného příkazu musí být právě jeden

z uvedených. Nejpoužívanější je dělení ACL na dva typy, a to:

- Standardní (Standard) ACL - starší a jednodušší verze ACL s méně možnostmi konfigurace
- Rozšířené (Extended) ACL - novější a složitější ACL s více možnostmi

Dále existují různé speciální ACL, které jsou často odvozeny z těchto dvou, jako je dynamické ACL, kontextové ACL, reflexivní ACL nebo pojmenované ACL.

3.3.1 Standardní ACL

- používá čísla 1 - 99 a 1300 - 1999 (unikátní číslo značí konkrétní seznam pravidel, použito jako `number` v syntaxi příkazu)
- je jednoduché na konfiguraci
- filtruje pouze podle zdrojové adresy
- obvykle se umísťuje na rozhraní co nejblíže destinaci

```
access-list number {deny|permit|remark} {host|source source-wildcard|any}
[log]
access-list 10 permit host 192.168.1.100
```

Pokud použijeme klíčové slovo `remark`, můžeme za něj vložit komentář (popis) použitého pravidla. Volitelný parametr `log` zajistí, že na konzoli a do logu budou posílány informace o paketech, které splní daná kritéria určená pravidlem. Toto je rozhodně dobré pro ladění/debugging, ale v reálném provozu příliš zatěžuje provoz na zařízení.[5]

3.3.2 Rozšířené ACL

- používá čísla 100 - 199 a 2000 - 2699 (unikátní číslo značí konkrétní seznam pravidel, použito jako `number` v syntaxi příkazu)
- dívá se na IP adresu zdroje i cíle
- kontroluje řadu položek v hlavičce vrstvy L3 a L4 (protokol, port apod.)
- může blokovat provoz kdekoliv (na rozhraní se obvykle umísťuje blízko zdroje)

Rozšířené ACL je schopno kontrolovat parametry jak v L3 vrstvě ISO/OSI (v IP hlavičce tedy IP adresu, protokol, údaje z ToS), tak v L4 vrstvě (v TCP hlavičce porty a protokoly, v UDP hlavičce porty, v ICMP hlavičce pak typy ICMP zpráv).

```
access-list number deny|permit|remark protokol
host|source source-wildcard|any [port]
host|destination destination-wildcard|any [port]

access-list 101 deny tcp host 192.168.1.100 eq 80 any
```

Jako protokol pak můžeme použít IP, TCP, UDP, ICMP, ale i mnohé další. V závislosti na použitém protokolu se mění parametry, například parametr port se používá pouze u TCP a UDP. Když chceme konkrétní pravidlo přidružit všem protokolům, použijeme IP, protože ostatní jsou mu podřízené.

V případě, že potřebujeme kontrolovat rozsah určitých portů, jsou k tomu přímo určené operátory. Pro porty existují operátory `eq` (je roven), `neq` (není roven), `gt` (je větší než), `lt` (je menší než) a `range` (rozsah). Operátory včetně čísla portu se pak zadávají za zdrojovou nebo cílovou adresu, kontrola portu se potom použije u zdroje nebo u cíle.[5]

Reflexivní ACL

Rozšířené ACL umožňují obsáhnout i parametr `established`, který prověřuje u TCP paketů ACK a RST bity, takže dokáže detekovat navázané spojení. Směrovači se pak povolí přijímat jen toto již započaté spojení. Toto chování je totožné s chováním `reflexivních ACL`. Definice příkazu dle Cisco IOS [5]:

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

Dynamické ACL

Tento typ ACL funguje na principu dynamického povolování komunikace s nějakým zařízením v síti. Komunikace je blokována pomocí rozšířených ACL na směrovači. Až poté, co se přes Telnet ověří uživatel na směrovači, je komunikace povolena. Dobu, po kterou je spojení povoleno, je možno nastavit. Lze nastavit časový interval nebo absolutní hodnotu, nebo použít přepínač „idle“. Dynamické ACL není možné definovat pomocí pojmenovaných ACL. Definice příkazu dle Cisco IOS [5]:

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

3.4 Pojmenované ACL

Pojmenované ACL jsou zjednodušením oproti označení číslem u standardních/rozšířených ACL, protože mohou nést jméno. Zpřehlední to tedy celý seznam, dále je možné zadávat více ACL v jednom bloku. K vytvoření pojmenovaných ACL slouží příkaz:

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

Jednotlivá pravidla pak konfigurujeme ve stejném tvaru jak pro standardní, tak i rozšířená ACL, přičemž s příkazem začneme až od části `{permit|deny|remark}`.[5]

3.5 Aktivace ACL na rozhraní

Nyní si ukážeme, jakým způsobem se aktivují jednotlivé ACL na rozhraní směrovačů. Po definici ACL máme k dispozici předpis pro určitý typ protokolu, na základě kterého pak můžeme provádět kontrolu paketů tohoto konkrétního typu protokolu. Abychom však mohli

ACL použít pro kontrolu paketů, je zapotřebí ACL v konfiguraci nějakého rozhraní přiřadit k aktivnímu protokolu a směru komunikace. Příkaz má tvar:

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

3.6 Shrnutí

V této kapitole jsme se seznámili s IP ACL. Na úvod jsme je zařadili do sféry bezpečnosti počítačových sítí a určili jsme, že svou činností se řadí mezi firewally typu paketových filtrů.

Dále jsme se blíže seznámili s ACL. Uvedli jsme, že ACL se skládá z jednotlivých pravidel, se kterými porovnává informace z příchozích paketů. Mezi nejčastěji porovnávané hodnoty patří typ protokolu, potom zdrojové a cílové IP adresy a porty. Uvedli jsme si také, jak se přiřazuje ACL na konkrétní rozhraní podle směru komunikace a typu protokolu. Také jsme si vysvětlili princip vyhodnocování paketu směrovačem od samotného vstupu na rozhraní až po odeslání paketu do sítě. V neposlední řadě jsme objasnili roli wildcard masky při porovnávání IP adres.

V předposlední části kapitoly jsme se seznámili s jednotlivými typy ACL. Základní rozdělení je na standardní a rozšířené ACL, přičemž standardní jsou schopny kontrolovat pouze zdrojovou IP adresu, zatímco rozšířené ACL umožňují mnohem pokročilejší možnosti kontroly. Základními oblastmi inspekce paketu jsou jak zdrojová, tak i cílová adresa a port, typ protokolu. Také jsme si uvedli pojmenovaná ACL, které nepřinášejí ani tak novou funkčnost, jako spíš zpřehlednění konfigurace ACL a možnost přiřadit jednotlivým sadám ACL jméno oproti běžnému číselnému označení.

Závěrem této části jsme ukázali způsob aktivace ACL na rozhraní routeru.

Kapitola 4

Interval Decision Diagrams

Cílem této kapitoly je vysvětlit reprezentaci filtrovacích pravidel pomocí Interval Decision Diagrams (dále jen IDD). Ve snaze najít optimální strukturu pro reprezentaci seznamu ACL, jsme se pokusili navrhnout vnitřní reprezentaci dat pro pozdější analýzu, která je založena na principu rozhodovacích stromů.

4.1 Vyhodnocení ACL jako formule predikátové logiky

Důležitou částí formální analýzy je, jakým způsobem budou vyjádřena data určená k analýze. V našem případě bylo zapotřebí najít způsob, jakým lze reprezentovat access-control listy (ACLs), které se používají pro filtrování provozu na směrovačích. Jejich reprezentace by měla být vhodná pro pozdější použití, např. přidávání dalších pravidel, vyhledávání (test na shodnost příchozího paketu s podmínkami). Uvažujeme ACL podle implementace na Cisco směrovačích [5].

Ve stručnosti, ACL je sekvence jednotlivých filtrovacích pravidel, které buď zakazují, nebo povolují specifický provoz z/do daných uzlů/sítí. Následující příklad povoluje pouze HTTP komunikaci pocházející ze sítě 192.168.1.0/24 v prvním pravidle, druhé pravidlo zakazuje jakoukoli jinou komunikaci z téže sítě. Jakékoli jiné zdrojové sítě vyjma 192.168.1.0/24 mohou komunikovat bez omezení [8].

```
permit tcp 192.168.1.0 0.0.0.255 any 80
deny ip 192.168.1.0 0.0.0.255 any
permit ip any any
```

Naším cílem je tyto uspořádané seznamy filtrovacích pravidel dokázat reprezentovat takovým způsobem, který je efektivní pro analýzu provozu na síti. Christiansen a Fleury v [4] dokázali, že filtrovací pravidla ve firewallech mohou být vnímany a reprezentovány jako stromy logických formulí, vhodně zanořených.

Filtrovací funkce na hlavičce h může být vyjádřena jako predikát v konjunktivní formě, viz.[4]. Například funkce f_1, f_2 a f_3 představují ACL pravidla výše.
$f_1(h) = (h.protocol \in TCP) \wedge (h.srcIP \in 192.168.1.0/24) \wedge (h.dstPort = 80)$
$f_2(h) = (h.protocol \in IP) \wedge (h.srcIP \in 192.168.1.0/24)$
$f_3(h) = (h.protocol \in IP)$

Co se protokolů týče, definujeme si IP = `{ip, udp, tcp}`, protože každý TCP či UDP paket může být klasifikován jako provoz na IP vrstvě, a proto je legitimní takový paket

posuzovat pravidlem pro IP. Pro uniformní reprezentaci si podobně definujeme TCP = `{tcp}`, UDP = `{udp}` a ICMP = `{icmp}`.

Jedno pravidlo ACL je dáno následovně:

```
RULE = <action: {permit, deny},
        match: IPheader -> BOOLEAN>
```

Paketový filtr ACL φ je uspořádaná sekvence pravidel r_i. Každé pravidlo r má přiřazenou akci a bude hledat shodu podle odpovídající filtrovací funkce f. Pro seznam ACL z příkladu 4.1 je dán odpovídající filtr φ_1:
$\varphi_1 = <r1, r2, r3>$
$r_1 = <action = permit, match = f_1>$
$r_2 = <action = deny, match = f_2>$
$r_3 = <action = permit, match = f_3>$

ACL je seznam vlastních filtrovacích pravidel, množina seznamů ACLs je definována následovně: $ACL = \{<r_1, ...r_n>, r_i \in RULE, 1 \leq i \leq n\}$ Tato forma reprezentace vyžaduje přesnou vyhodnocovací metodu. Pořadí pravidel je zde samozřejmě velice důležité vzhledem k tomu, že pokud je nalezena shoda s ve funkci pravidla f_i, provede se odpovídající akce a další pravidla se už neporovnávají. V případě, že f_i se neshoduje s příchozím paketem, jsou testována další pravidla. Buď je nalezena shoda s jedním z pravidel, nebo je dokončeno porovnání paketu s posledním pravidlem v ACL seznamu. Pokud není nalezena žádná shoda, není aplikována žádná akce a paketu je umožněn vstup. Avšak reálná implementace ACL v Cisco směrovačích (které jsme si vzali za vzor) přidává implicitní pravidlo `deny any` na konec ACL seznamu. Toto pravidlo $r_\infty = <action = deny, match = (h \rightarrow TRUE)>$ zahodí jakýkoli paket. Dále budeme předpokládat, že každé ACL vždy obsahuje toto implicitní pravidlo r_∞ [8].

Způsob vyhodnocení ACL: Algoritmus pro vyhodnocení ACL je rekurzivní funkce $AclEval : IPHeader \times ACL \rightarrow \{Permit, Deny\}$, která je definována následovně:

- AclEval(h,a) = Deny, pokud a je prázdný list,
- AclEval(h,a) = **if** r.Match(h) **then** r.Action **else** AclEval(h,t) pokud head(a,r) a tail(a,t).

Ukážeme si, jak lze reprezentovat souhrnně ACL pravidla jako jednu predikátovou formuli bez kvantifikátorů, což je efektivnější z hlediska využití paměti i vyhodnocování, při použití struktur na bázi binárních rozhodovacích diagramů (BDD) [3].

Procedura transformace vytvoří predikát z množiny všech filtrovacích pravidel v ACL seznamu tak, že iterativně bere jedno pravidlo po druhém od začátku do konce seznamu ACL v daném pořadí (což je velice důležité). Při zpracování každého z pravidel přidá nové pravidlo k logické formuli podle následující definice:

- Pokud je akce v daném pravidle `permit`, znamená to, že je dané pravidlo připojeno operátorem disjunkce ($\vee$) k logické formuli.
- Pokud je akce v daném pravidle `deny`, je přidána negace pravidla a je připojena operátorem konjunkce ($\wedge$) k logické formuli.

Až se dosáhne konce ACL seznamu, stejným způsobem je do vytvářené formule přidáno i implicitní pravidlo *deny any*. Princip dělení seznamu na head a tail je následující: Když vezmeme první pravidlo ze seznamu ACL, nazveme jej head, zbytek (tedy druhé až n-té pravidlo) nazveme tail. Tento postup se aplikuje rekurzivně vždy na každé pravidlo, přičemž s počtem pravidel roste i míra zanoření. Po zpracování aktuálního pravidla se toto pravidlo ze seznamu odstraní, a zbytek seznamu se stane novým seznamem, na který zase aplikujeme dělení na head a tail. Pokud dojdeme k poslednímu pravidlu, zbude nám prázdná množina, kterou můžeme brát jako FALSE. Tímto způsobem jsme schopni převést celý seznam ACL pravidel na jedinou formuli predikátové logiky.

Celý postup si nyní po krocích ukážeme na ACL seznamu z příkladu 4.1:
Filtrovací predikát φ vyvíjející se při zpracování v krocích:
$\varphi_1 = f_1(h)$
$\varphi_2 = \varphi_1 \vee (\neg f_2(h)$
$\varphi_3 = \varphi_2 \wedge (f_3(h) \vee FALSE)$
ACL seznam: zpracován, výsledkem logického součtu $(f_3(h) \vee FALSE)$ bude vždy $f_3(h)$.
Výsledný filtrovací predikát tedy dostaneme v podobě:

$$\varphi = f_1(h) \vee (\neg f_2(h) \wedge f_3(h))$$

Jak tedy lze vidět z tohoto postupu, pořadí jednotlivých pravidel de facto určuje prioritu ve filtrovací formuli. První pravidlo provádí logický součet s celým zbytkem všech filtrovacích pravidel bez ohledu na to, jestli zbytek je 2 nebo 150 pravidel. Vyhodnocování logických operací nad pravidly probíhá postupně zprava od nejvíce zanořených závorek doleva.

4.2 IDD jako efektivní datová struktura pro ACL

Jak jsme si ukázali v předchozí kapitole, ACL můžeme vyjádřit pomocí jedné logické formule. Pro výpočet operací konjunkce, disjunkce apod. můžeme tuto formuli reprezentovat pomocí speciálních datových struktur - Binary Decision Diagrams (BDDs)[3].

Operacemi definovanými nad touto datovou strukturou můžeme lehce manipulovat s logickou formulí. Filtrovací pravidla obvykle pracují s určitým rozsahem IP adres a čísel portů. Pro reprezentaci rozsahu tohoto typu se jeví jako nejvhodnější řešení použít intervaly, které jsou velmi efektivní co se týče výpočetního času a místa v paměti. Struktura Interval Decision Diagrams (IDD) [11] nám umožňuje provádět jednodušší klasifikaci přirozených čísel v rámci konečné domény.

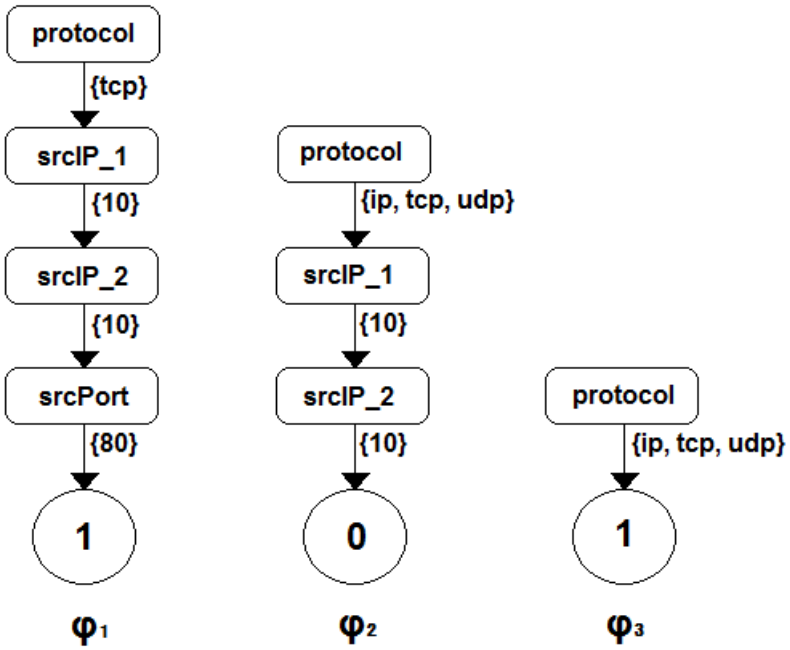
IDD je kořenový, orientovaný acyklický graf se dvěma typy uzlů (terminály a nonterminály) s tím, že jeden nebo dva terminální uzly jsou označeny 0 (FALSE) nebo 1 (TRUE).

Ukážeme si, jak výhodně využít IDD pro uložení rozsahu IP adres. Interval IP adres v pravidle ACL je určen wildcard maskou. Intervaly jsou pak vypočteny na základě těchto masek. Představme si IP adresu jako čtyři oddělená přirozená čísla v rozsahu 0-255. Potom například IP adresu 10.0.0.0/16 můžeme převést na intervaly `{10}.{0}.[0-255].[0-255]`, zatímco 10.1.0.0/16 můžeme zařadit jako `{10}.{1}.[0-255].[0-255]`. Každý oktet je reprezentován pomocí intervalové množiny, `{}` představuje jediné číslo, `[]` představuje množinu čísel.

Jasná výhoda použití reprezentace pomocí IDD pak spočívá v rychlém vyhodnocením seznamu. Pokud nějaké části IP adresy (oktety) obsahují stejné hodnoty, mohou se spojit.

Oktety, jejichž hodnoty tvoří po překrytí s jinými navazující interval, pak mohou být spojeny a reprezentovány jedním intervalem. V následujícím příkladě vytvoříme interval, který kombinuje oba intervaly výše zmíněné. `{10}.[0-1].[0-255].[0-255]` odpovídá IP adrese 10.0.0.0/15.

Filtrovací funkce hledá shodu s paketem na základě několika informací - zdrojové a cílové adresy, typu protokolu, zdrojového a cílového portu. Pro reprezentaci pomocí IDD je nezbytně nutné, abychom byli schopni definovat interval pokrývající celou doménu (rozsah) pro každé pole z hlavičky paketu. To nám umožní rozdělit doménu na intervaly, které můžeme užít v IDD grafu.



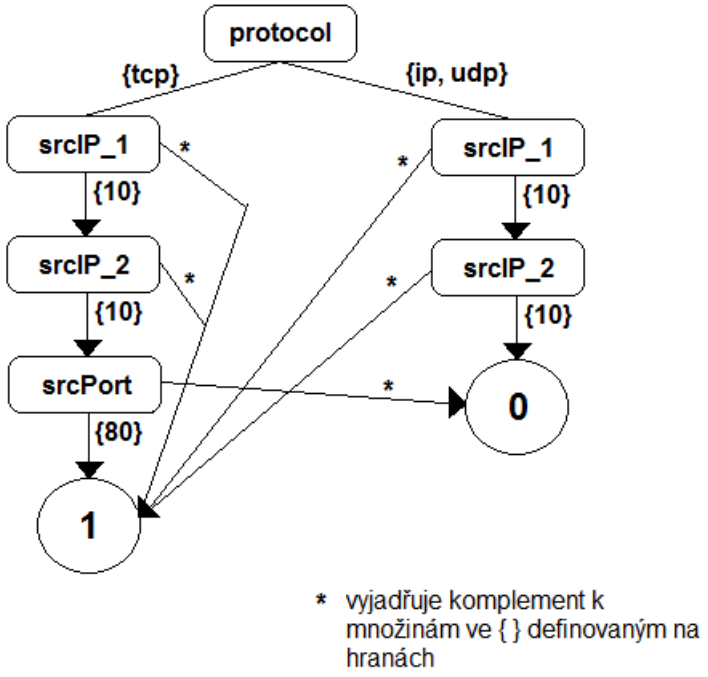
Obrázek 4.1: Příklad reprezentace ACL pravidel pomocí IDD

Každá správně vytvořená funkce hledající shodu může být vyjádřena pomocí IDD. Například, na obrázku 4.1 jsou vyjádřeny takové funkce φ_1, φ_2 a φ_3. Podobně jako jsme výše konstruovali celkovou filtrovací funkci φ, odpovídající IDD struktura může být sestavena použitím základních logických operátorů, které aplikujeme na IDD grafy sestavené pro jednotlivá pravidla. Všechny použité logické operátory ($\vee$, $\wedge$, $\neg$) musí být striktně definovány nad strukturou IDD a musí zachovávat zamýšlenou sémantiku.

Je obecně známo, že struktura IDD diagramů záleží na pořadí proměnných. Z příkladu 4.2 je zřejmé, že užití jiného pořadí porovnávaných proměnných (parametrů) by mohlo vést k jiné, potenciálně efektivnější reprezentaci. Nakolik se ale efektivita může zlepšit/zhoršit, není předmětem této bakalářské práce. Nicméně, i v [4] je použito sestavování IDD v pořadí protokol, IP adresa po oktetech od prvního k poslednímu, port.

Obrázek 4.2 ukazuje příklad vytvoření filtrovací funkce složené ze tří pravidel z příkladu 4.1, a to pomocí IDD. Konstrukce používá uspořádání proměnných $x = <protocol, srcIP, dstIP, srcPort, dstPort>$. IDD, které vznikne, je výsledek kombinace tří funkcí φ_1, φ_2, φ_3

a optimalizačního kroku, který odstraňuje duplicity, a tím zlepšuje efektivitu.



Obrázek 4.2: Příklad reprezentace celkové filtrovací funkce pomocí IDD

4.3 Principy a výhody použití IDD v paketovém filtru

Firewall je jedna z klíčových technologií síťové bezpečnosti, protože umožňuje síťovým administrátorům kontrolovat přístup do sítě. Princip spočívá v tom, že v jednom centralizovaném místě se rozhoduje, zda paket vstupující nebo vystupující z firewallem chráněné sítě bude filtrován. Tento centrální mechanismus se nazývá paketový filtr, který má za funkci rozhodovat o tom, které pakety projdou skrze firewall na základě specifikace filtru. Tato specifikace se sestavuje z množiny pravidel, která nám říká, jaký postup máme použít na který paket na základě informací z hlavičky paketu.

Základním aspektem filtrování paketů je vyhodnocování paketu, tzv. klasifikace. Je předmětem mnoha diskuzí, jaký způsob pro klasifikaci paketů zvolit. Požadavky na klasifikaci se liší, záleží na použití. Ve většině případů se však jedná zejména o rychlost, velikost reprezentace množiny pravidel a složitost algoritmu vytvářejícího reprezentaci množiny pravidel. Firewall sice nemění množinu svých pravidel často, zase je však potřeba klasifikovat paket pomocí více proměnných v hlavičce paketu.

Díky informacím čerpaných z [4] si nyní shrneme hlavní rysy IDD, které by se daly charakterizovat následovně:

- Přístup k filtrovacím pravidlům přes Booleovu algebru. To zjednodušuje popis algoritmů použitých ve schématu, protože jsme schopni používat dobře pochopitelné

operace jako porovnání, průnik a sjednocení. Tyto operace jsou samozřejmě definovány nad intervaly.

- Kompaktní reprezentace a snadná rozšiřitelnost. Hloubka rozhodovacích diagramů nezávisí na počtu položek z hlavičky paketu, je konstantní.
- Efektivní složitost klasifikace paketu. Při reprezentaci pomocí IDD se složitost dá vyjádřit jako $O(m \cdot \log r)$, kde m je počet proměnných v hlavičce paketu a r je maximální rozsah těchto proměnných.

Kapitola 5

Převodní nástroj pro ACL

Tato kapitola popisuje způsob převodu konfigurace ze směrovačů do simulátoru OMNeT++. Pro funkčnost filtrování paketů je naprosto zásadní získat pro každý do simulace zapojený router data z jeho access-listů. O převod konfiguračních Cisco IOS souborů se stará týmový kolega z projektu ANSA[9]. Výstupem jeho práce[9] je globální XML soubor, z něhož si potom jednotlivé moduly zapojené v simulaci načítají konfigurační data.

5.1 Konfigurace XML

Globální konfigurace pro všechny routery je převedena do souboru formátu XML, z něhož se pak dají jednoduše získávat data pomocí XML knihovny vestavěné v OMNeT++. Náš modul si potom pro potřeby simulace dokáže načíst data pro každý router, který je v dané simulaci zapojen. ID routeru je zjištěno pomocí funkce `getElementByPath()`. Další funkce pro procházení zanořených elementů využívané z XML knihovny jsou `getFirstChildWithTag`, `getNextSibling()` a `getNextSiblingWithTag()`. Dále jsme použili funkce pro výběr hodnot z elementů, a to `getAttribute()`, `getTagName()` a `getNodeValue()`. Nejprve si tedy zjistíme ID routeru v aktuální instanci ACL modulu a poté už procházíme jednotlivé elementy pod blokem `<ACLs>`, kde získáváme potřebná data a ukládáme si je do předpřipravených datových struktur, o kterých si však povíme více až v podkapitole 6.2.

Pro lepší znázornění jistě poslouží komentovaná ukázka potřebné části XML dokumentu:

```
<?xml version=,,1.0‘‘ encoding=,,windows-1250‘‘?>
<Router id=,,192.168.3.1‘‘>                  # ID SMEROVACE
 <ACLs>
  <ACL no=,,101‘‘>                           # CISLO SEZNAMU ACL
    <entry seq_no=,,10‘‘>                    # CISLO PRAVIDLA
      ## JEDNOTLIVE CASTI PRAVIDLA PREVEDENA Z~CISCO KONFIGURACE ##
      ## access-list 101 permit tcp 192.168.1.100 0.0.0.255 eq 80
            10.60.0.0 0.0.0.127 range 40000 40100 ##
      <action>permit</action>
      <IP_src>192.168.1.0</IP_src>
      <IP_dst>10.60.0.0</IP_dst>
      <WC_src>0.0.0.255</WC_src>
      <WC_dst>0.0.0.127</WC_dst>
      <port_op_src>eq</port_op_src>
      <port_op_dst>range</port_op_dst>
```

```
      <port_beg_src>80</port_beg_src>
      <port_end_src></port_end_src>
      <port_beg_dst>40000</port_beg_dst>
      <port_end_dst>40100</port_end_dst>
      <protocol>tcp</protocol>
      <orig>access-list 101 permit tcp 192.168.1.100 0.0.0.255 eq 80
            10.60.0.0 0.0.0.127 range 40000 40100</orig>
    ## ROZHRANI A~SMERY PRO KTERA JE DANY ACL SEZNAM AKTIVNI ##
    <interfaces>
      <interface dir=,,in‘‘>eth0</interface>
      <interface dir=,,out‘‘>eth1</interface>
    </interfaces>
  </ACL>
 </ACLs>
</Router>
```

XML je navrženo s ohledem na efektivní umístění našeho ACL modulu, které bude popsáno v dalším odstavci. Pro každý směrovač může být definováno více ACL, v sekci ACLs jsou definovány všechny aktivní seznamy. V každém ACL může být libovolné množství konkrétních záznamů obsahující filtrovací pravidlo. Na konci každého ACL je definováno v prvku `<interfaces>`, na kterých rozhraních a ve kterém směru je daný ACL aktivován. Můžeme si také všimnout, že směr je definován jako atribut prvku `<interface>`, zatímco samotná identifikace rozhraní je hodnotou tohoto prvku. Určení toho, který seznam je navázán na které rozhraní a směr komunikace, je velmi důležité s ohledem na další implementaci. Za účelem přiřazení seznamu na určité rozhraní jsme si vytvořili strukturu

```
struct TInterface
{
   int gateIndex;
   bool dir;
   TACL* rules;
};
```

Tato struktura nám slouží jednak k uložení si indexu simulovaného rozhraní (např. eth0), které je nadefinováno v XML prvku `<interface>`, dále k uložení si směru pro který je seznam přiřazen. Struktura také obsahuje ukazatel na konkrétní seznam pravidel, který se přiřadí později metodou `acl::getRules()`. Ta porovnává informace načtené z XML pro konkrétní rozhraní a směr s informacemi z příchozího paketu. Pokud se shoduje jak brána, tak i směr, je vše v pořádku a příchozímu paketu je přiřazen konkrétní seznam, podle kterého se bude provádět filtrování. V opačném případě není paketu přiřazen žádný seznam (hodnota *NULL*) a paket je automaticky přeposlán dále do/ze směrovače.

Prvky zanořené pod uzlem `<entry>` nesou informace o konkrétním pravidle. Jsou zde přítomny všechny potřebné hodnoty pro filtrování pomocí rozšířených ACL, ne všechny však musí být nutně vyplněny. Z tohoto důvodu se na parametry, které nenesou žádné informace z definice záznamu, aplikují implicitní hodnoty, které jsme si pro takové případy vhodně zvolili. Například, když nebude definován operátor pro zdrojový port a číslo zdrojového portu, potom v `port_beg_src` bude hodnota 0 a v `port_end_src` bude hodnota 65535. Tím je zajištěno, že se dané pravidlo aplikuje na všechny porty.

Následující tabulka 5.1 názorně předvádí transformaci originálních příkazů z aktivních

zařízení firmy Cisco na XML konfiguraci až po finální krok převedení na vhodnou reprezentaci do datové struktury. Ta je potom použita v implementaci filtrovacího modulu.

Cisco zařízení	Konfigurace v XML	Reprezentace ve struktuře pro ACL
access-list 101 permit tcp host 10.60.20.1 eq www any	<ACL no="101"> <entry> <action>permit</action> <protocol>tcp</protocol> <IP_src>10.60.20.1</IP_src> <WC_src>0.0.0.0</WC_src> <IP_dst>0.0.0.0</IP_dst> <WC_dst>255.255.255.255</WC_dst> <port_op_src>eq</port_op_src> <port_src_beg>www</port_src_beg> </entry> </ACL>	TRule action = A_PERMIT Trule protocol = PROT_TCP TIP source.ipAddr = 10.60.20.1 TIP source.netmask = 255.255.255.255 TIP dest.ipAddr = 0.0.0.0 TIP dest.netmask = 0.0.0.0 TIP source.port_op = PORT_EQ TIP source.portBeg = 80 TIP dest.port_op = PORT_RNG TIP dest.port_beg = 0 TIP dest.port_end = 65535

Obrázek 5.1: Přehledová tabulka převodního nástroje

Kapitola 6

Implementace ACL do OMNeT++

Tahle kapitola se zabývá samotnou implementací modulu na filtrování paketů pomocí ACL. Nejprve si ukážeme návrh implementace, poté se obecně seznámíme s umístěním modulu v simulačním schématu, zvolíme vhodné struktury pro manipulaci s daty, diskutujeme problémy a zvláštnosti, se kterými se v průběhu implementační fáze bude třeba vypořádat. Na závěr kapitoly budeme konzultovat efektivitu a vhodnost zvolené implementace a ověříme si její funkčnost.

6.1 Návrh modulu

Základem kvalitní implementace je dobře zpracovaný návrh. Vzhledem k povaze nástroje OMNeT++ bylo od počátku jasné, že programovacím jazykem modulu bude C++. V kapitole 3 jsme si teoreticky popsali funkci a zápis ACL. Před započetím vlastní implementace však musíme vyřešit problémy, které by mohly bránit rozšíření simulátoru o filtrovací modul. Je důležité uvážit tyto body a dokázat na ně odpovědět:

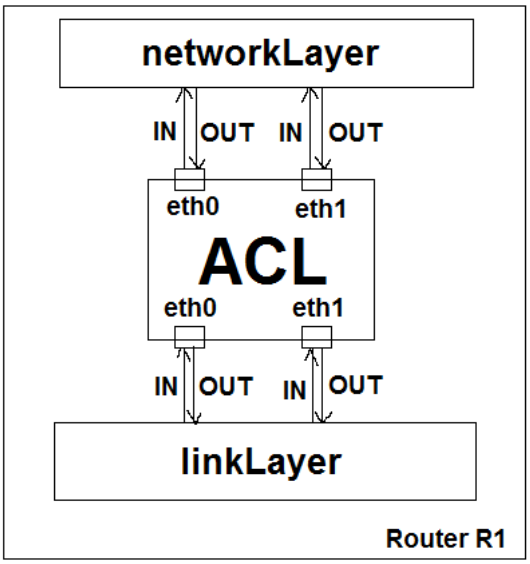
- Kde vezmeme data pro ACL z konfiguračního souboru routeru?
- Kam umístíme nový prvek - modul pro filtrování paketů pomocí ACL - na cestě paketu z/do směrovače tak, aby simulace odpovídala co nejvíce reálnému stavu?
- Jakým způsobem je implementován paket v OMNeT++ a jak rozeznáme jednotlivé typy paketů ? Jak budeme porovnávat všechny informace potřebné k filtrování? Jakým způsobem budeme paket zahazovat?
- V jaké datové struktuře a v jakém formátu budou uložena data vytvořená z ACL seznamů, aby bylo možné co nejefektivnější vyhledávání?
- Jak poznáme, ze kterého rozhraní paket dorazil a v jakém směru?

V následujících podkapitolách zdůvodníme, jak jsme se rozhodli tyto problémy řešit. Převodní nástroj a načítání konfigurace z XML je popsáno v kapitole 5.

6.1.1 ACL jako nový prvek v cestě paketu

Aby bylo dosaženo optimální efektivity a snadné implementace do simulačního schématu, rozhodli jsme se zvolit umístění nového modulu následovně. Ve směrovači bude obsažen filtrovací modul ACL, který bude propojený z jedné strany s linkovou vrstvou a ze strany

druhé s vrstvou síťovou. Filtrovací modul bude v simulačním schématu pouze jeden pro každý směrovač, jednotlivá rozhraní budou zakomponována napojením na modul jako dvě brány, jedna pro vstupní směr a druhá pro směr výstupní pro každou z vrstev (síťovou i linkovou), tak jak je vyznačeno na obrázku 6.1.



Obrázek 6.1: Umístění modulu ACL v simulačním schématu směrovače

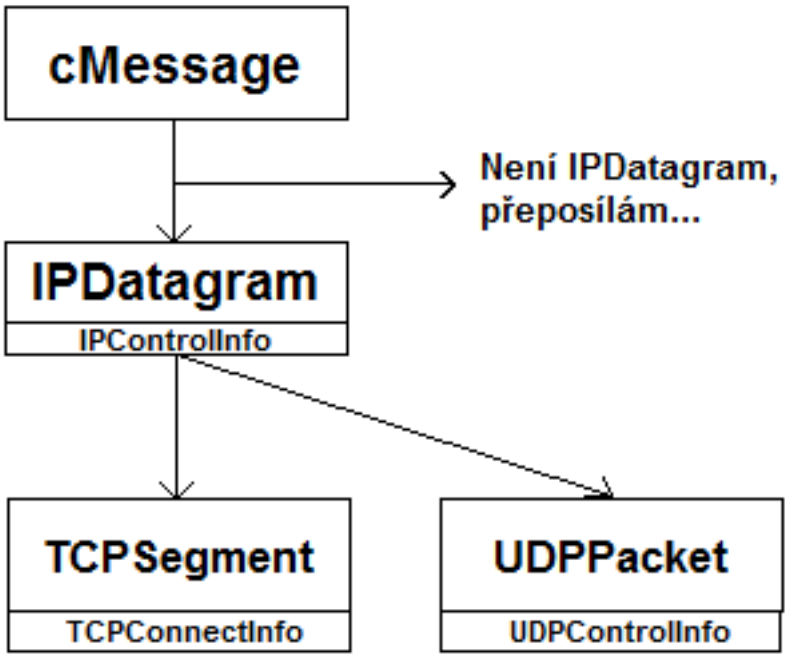
Vzhledem k tomu, že jsme zvolili takové zapojení, stačí načíst jen jedenkrát konfiguraci z XML, které formou odpovídá návrhu umístění filtrovacího modulu. Není ani potřeba v rámci jednoho směrovače klonovat více ACL modulů pro každé rozhraní. Jednotlivá rozhraní směrovače budou napojena na modul pomocí čtyř bran (ifIn, ifOut, ToNetworkLayerIn, ToNetworkLayerOut), což je nadefinováno v .ned souboru filtrovacího modulu.

Každý příchozí paket bude zaobalen ještě další hlavičkou, to znamená, že kromě samotných dat o paketu bude navíc obsahovat informace o rozhraní, na kterém přišel do směrovače (potažmo do ACL modulu). Podobným způsobem se zjistí i směr paketu (in/out), a to podle konkrétní brány z obrázku 6.1.

6.1.2 Struktura paketu a práce s ním v OMNeT++

V simulačním nástroji OMNeT++, jak jsme se dozvěděli v kapitole 2, funguje komunikace na principu zasílání zpráv mezi jednotlivými moduly a branami. Proto každý paket, který přijde na nějaké rozhraní nebo který je přeposílán mezi moduly v rámci simulace, přijde zabalen jako datový typ „zpráva“, resp. je typu `cMessage`. Takto přijatá zpráva se dá samozřejmě přetypovat na paket typu `IPDatagram`, který se dá dále zkoumat a mohou z něj být extrahována určitá data, například typ protokolu a IP adresy. Pro zjištění, ze kterého portu, příp. na který port je paket posílán, musíme nejprve zjistit typ protokolu; pokud se jedná o protokol TCP nebo UDP, paket dále rozbalíme - získáme buď `TCPSegment`, nebo `UDPPacket`, odkud získáme informace z TCP/UDP hlavičky, např. port. Podobně se dá z ICMP paketu zjistit typ ICMP zprávy. Přetypování na paket jiné třídy probíhá přímo za běhu simulace pomocí operátoru `dynamic_cast`, resp. `check_and_cast`.

Náš modul je schopen filtrovat veškerý IP provoz, tedy pakety, které jsou třídy „IPDatagram“. Na brány našeho modulu mohou být však zasílány i různé jiné pakety, např. z nižších ISO/OSI vrstev - linková vrstva, aj. Jiné než IP pakety by ACL modul neměl filtrovat, proto jsme v implementaci zahrnuli testování, že pokud příchozí paket není typu „IPDatagram“, pouze si zjistíme jeho příchozí bránu a index brány, na kterou je určen, a celou zprávu bez jakýchkoliv úprav pošleme dále. Tímto způsobem jsou ošetřeny například ARP pakety, které si posílají samotná rozhraní s dotazy na MAC adresy apod. Teprve ve chvíli, kdy bezpečně víme, že příchozí zpráva je IP paket, paket dále zpracováváme. Princip práce s paketem je znázorněn na obrázku 6.2.



Obrázek 6.2: Schéma práce s paketem v OMNeT++

6.2 Datová struktura pro ACL

V úvodu kapitoly jsme si pověděli něco o tom, že se budeme muset rozhodnout, kam ukládat data z filtrovacích seznamů. Nejdříve si řekneme něco o dvou možných způsobech, srovnáme je a popíšeme, uvedeme si klady a zápory obou metod.

6.2.1 Implementace ACL seznamem lineárních seznamů

Podobně jako u definice ACL od firmy Cisco, je nasnadě přemýšlet o ACL jako o seznamu pravidel. Lineární seznam je pokročilá datová struktura, která se skládá ze záznamu (položky) a ukazatele na následující položku seznamu. Samotný řádek pak může být další

netriviální struktura. V našem případě při implementaci ACL se pod obecným pojmem řádek skrývá konkrétní jedno aplikovatelné pravidlo ACL seznamu. Pravidlo je složitá datová struktura záznam (položka), která nese tyto informace: protokol, IP adresu zdroje a cíle, počáteční port zdroje a cíle, koncový port zdroje a cíle, operátor portu zdroje a cíle, typ akce (povolit/zakázat) a ukazatel na čítač použití daného pravidla. V lineárním seznamu může být teoreticky libovolný počet jednotlivých záznamů, limitováni jsme pouze velikostí paměti.

Pro každý ACL existuje jeden lineární seznam. Protože však už z definice ACL víme, že seznam existuje pro každé rozhraní a každý směr, je logické, že seznamů budeme potřebovat uložit více, ne však používat všechny zároveň. Toho dosáhneme tak, že při načítání ACL z XML souboru budeme vytvářet seznam struktur lineárních seznamů.

Lineární seznam implementujeme pomocí datového typu `std::list`, který je standardní datovou strukturou v C++. Procházíme jej po prvcích v cyklu pomocí vytvořeného iterátoru.

Ukázka použitých struktur v pořadí:

- Struktura `TIP` obsahuje IP adresu a masku, počáteční a koncové číslo portu z rozsahu a operátor portu.
- Struktura `TRule` definuje jedno konkrétní pravidlo - ACL záznam. Obsahuje akci, protokol a zdrojovou a cílovou strukturu `TIP`, viz výše. Dále obsahuje ukazatel na čítač `used`, abychom dokázali později určit, kolikrát bylo konkrétní pravidlo aplikováno.
- Třída pro statistiky o činnosti ACL modulu.
- Definujeme nový datový typ `TACL`, což je `std::list` (seznam) struktur `TRule`.
- Vytvoříme iterátor k procházení seznamu pravidel.
- Vytvoříme seznam seznamů - `acls` je `std::list` (seznam) seznamů `TACL`.
- Vytvoříme seznam statistik - `std::list` (seznam) statistik třídy `Stat`.

```
struct TIP
{
   IPAddress ipAddr, netmask;
   int portBeg, portEnd;
   TPortOP port_op;
};
```

Pokud je v seznamu ACL IP adresa definována klíčovým slovem `host` nebo `any`, přetransformuje se vždy na číselně vyjádřenou IP adresu a wildcard masku. Tuto práci zajišťuje modul pro načítání konfigurace z aktivních síťových zařízení[9].

```
struct TRule
{
   bool action;
   TProtocol protocol;
   TIP source, dest;
   int* used;
};
```

```
class Stat
{
public:
   std::string text;
   int used;
};

typedef std::list<TRule> TACL;
typedef std::list<TRule>::iterator TACL_it;
std::list<TACL> acls;
std::list<Stat> stats;
```

Mezi výhody použití lineárního seznamu jakožto implementačního prvku patří především relativní jednoduchost samotné implementace. Také se v něm lépe orientuje, snadněji se prochází (stačí použití cyklu). Nevýhodou lineárního seznamu je vyšší náročnost s přibývajícím počtem položek při jeho procházení. Pokud hledáme pravidlo, které se na paket má aplikovat, nejhorší možný případ co se týče složitosti nastává, jestliže paket neodpovídá žádnému z pravidel v ACL seznamu. Je totiž zapotřebí projít všechny řádky (n) struktury, přičemž v každém řádku je ještě 5 sloupců ($m = 5$). Moje implementace je založena na hledání neshody v jednotlivých sloupcích (částech pravidla) v každém řádku (pravidle). Jakmile je nalezena neshoda, aktuálně procházený řádek se ihned přeskočí a hledá se až v dalším řádku. To je určitá optimalizace, protože nemusíme provádět vždy porovnání všech proměnných v pravidle, např. pokud už víme, že uvedený protokol v pravidle se neshoduje s informací o protokolu z paketu, rovnou se přejde k porovnání s dalším pravidlem.

I přesto teoretická nejhorší možná složitost je $O((m-1) \cdot n)$, kde m je počet kontrolovaných položek v hlavičce paketu a n je počet pravidel. Při $m = 5$ tedy složitost vychází lineární, $O(4 \cdot n)$. Při velké délce seznamu ACL by bylo procházení seznamu náročnější, avšak pro potřeby naší simulace je implementace lineárním seznamem dostačující.

6.2.2 ACL jako Interval Decision Diagrams

My si však ukážeme další možnou metodu uložení informací z ACL, která je sice mnohem náročnější na implementaci, avšak její složitost je o poznání nižší. Informace k Interval Decision Diagrams (dále jen IDD) je popsána v kapitole 4.

6.3 Filtrování ve směrovacích protokolech

Kromě tradičního filtrování se práce zabývá také zkoumáním, jak bude síť reagovat při povolených/zakázaných směrovacích protokolech. Například, zakážeme-li protokol OSPF (89), přestanou chodit Hello pakety OSPF a výměna směrovacích informací mezi sousedními směrovači nebude provedena. Díky filtrování ACL tak můžeme zkoumat chování sítě při změně její topologie (např. směrování už dále nebude OSPF, ale RIP) apod.

Jak napovídá definice ACL záznamů (viz kapitola 3), jako protokol může být uveden libovolný typ protokolu dle RFC 791[7], a to buď jeho názvem, (tcp, udp, icmp, ospf aj.) anebo číslem (6, 17, 1, 89 aj.) Pokud na směrovač dorazí IP paket, je klasicky rozbalen a dále zjištěn jeho protokol. Můžeme v podstatě i filtrovat směrování, tedy přesněji řečeno pakety, jež jsou posílány navzájem mezi směrovači, které se tím informují o změnách ve

směrovacích tabulkách, nebo zjišťují další nové sousedy, cesty, apod. Můžeme například zakázat RIP pakety z určité sítě (protokol UDP, port 520), přičemž ostatní data z dané sítě mohou procházet bez problému.

Implementace je ošetřena tak, aby výše popsané umožňovala. Vždy se zjistí nejprve protokol z IP paketu a posléze se na paket aplikuje nějaké konkrétní pravidlo, existuje-li takové v seznamu ACL. Filtrovací komponenta je schopna reagovat na protokoly ip, tcp, udp, icmp, igmp, eigrp, ospf a sctp. Není však problém do zdrojových kódů případně přidat další protokoly dle naší potřeby.

6.4 Filtrování IP adres s využitím wildcard masky

Jak jsme uvedli v kapitole 3, rozsah síťových adres určených k filtrování je v ACL vždy určen dvojicí IP adresa a wildcard maska. V implementaci filtrovacího modulu bychom mohli toto hledisko rozdělit na dvě části. Nejprve při samotném zpracování dat z XML konfiguračního souboru si uložíme IP adresu, reprezentujeme ji strukturou `IPAddress` z knihovny OMNeT++. Dále wildcard masku si převedeme na síťovou masku, a to pomocí operace bitové negace. Druhou částí je potom samotné porovnávání. Nad IP adresou v ACL pravidle provedeme operaci logického součinu se síťovou maskou metodou `andIpWithMask()`. Z paketu nejprve získáme jeho zdrojovou či cílovou IP adresu. Poté metodou `ipIsEqual()` zpracováváme strukturu, ve které máme uloženu jak IP adresu, tak i masku. Na základě následujícího porovnání zjistíme, zda IP adresa z právě zpracovávaného paketu patří/nepatří do rozsahu možných IP adres obsažených v ACL pravidle. To docílíme operací bitového součinu, která je prováděna nad IP adresou a maskou metodou `doAnd()`:

```
if (ip->ipAddr != (packet->ipAddr.doAnd(ip->netmask)))
```

Například, pokud se v ACL pravidle nachází zdrojová síť 10.10.10.128 s wildcard maskou 0.0.0.3, nejprve provedeme bitovou negaci a získáme tak masku 255.255.255.252. Poté provede operaci AND nad adresou a maskou z ACL pravidla. Jelikož `128 & 252` vrátí 128, záznam zůstane v podobě 10.10.10.128. Následně pokud přijde paket pocházející z IP adresy 10.10.10.130, tak provedeme bitový součin `130 & 252`, což nám dá výsledek 128. Vidíme tedy, že poslední oktet adresy sítě se shoduje, čili příchozí paket patří do sítě uvedené v daném ACL pravidle. Porovnání posledního čtvrtého oktetu rozepsáno bitově:

```
10000010 &          130
11111100            252
```

Vidíme tedy, že jelikož poslední dva bity masky nenesou hodnotu 1, operací bitového součinu vznikne hodnota 1 pouze v nejlevějším bitu, a tedy výsledek této operace bude 10000000_b, což je 128_d.

V seznamu uložená IP adresa je konstantní, protože se síťovou maskou se neguje již při načítání z XML. Při kontrole každého paketu se vždy musí provést operace bitového součinu nad IP adresou z hlavičky paketu a maskou sítě z ACL pravidla.

Modul dokáže zpracovat IP adresy zadané v ACL všemi třemi způsoby. První možností je kombinace klíčových slov `host IP_ADDR`, kterou převádíme na IP adresu `IP_ADDR` a wildcard masku 0.0.0.0, potažmo masku podsítě 255.255.255.255, což nám v podstatě říká, že není možné změnit ani jediný bit v IP adrese, tj. že se bude porovnávat všech 32 bitů IPv4 adresy. Druhou možností je klíčové slovo `any`, které si ihned převedeme na IP adresu 0.0.0.0 (může být v podstatě jakákoli) a wildcard masku 255.255.255.255, tedy masku podsítě

0.0.0.0. To nám říká, že se nebude kontrolovat ani jeden bit v IP adrese, což znamená, že dané pravidlo splní libovolná 32-bitová IPv4 adresa. Poslední a také nejčastěji používanou možností je kombinace `IP_ADDR WILDCARD_MASK`, kde je IP adresa sítě a příslušná wildcard maska. Převod se provede postupem popsaným na začátku této podkapitoly.

6.5 Filtrování portů

Samotné porty mohou být zadány buď číslem anebo názvem. Pokud je port zadán jeho názvem (dle RFC 1700), je převeden na číselný formát. V reprezentaci podle názvu jsou podporovány porty ftp, ftp-data, smtp, telnet, www, http-www, tftp, pop3, snmp, irc, ipx, ldap a https. Tyto porty jsou statisticky nejčastěji používané. Není však problém do zdrojových kódu v případě potřeby přidat další názvy portů. Načítání konfigurace z XML pro porty probíhá následovně: Pokud je přítomen operátor portu z množiny (*eq*, *neq*, *lt*, *gt*), načte se jedno číslo portu do *portBeg*. Pokud je přítomen operátor *range*, načte se první číslo za ním do *portBeg* a druhé do *portEnd*. Pokud není detekován žádný operátor portu, znamená to, že porty nejsou zadané (dle kapitoly 3 víme, že příkaz port je volitelný příkaz v ACL záznamu). Pokud se tedy nemá dle portů filtrovat, do *portBeg* uložíme 0 a do *portEnd* uložíme 65535, čímž zajistíme pokrytí celého možného rozsahu portů. Tím v podstatě zajistíme, že ať je port v příchozím paketu jakýkoliv, podmínka na vyhodnocení portu bude vždy splněna.

6.6 Úroveň podpory implementovaných ACL

V následujících podkapitolách uvedeme jednotlivé typy ACL, které jsou podporovány v naší implementaci, a popíšeme si úroveň jejich podpory.

Pro všechny typy ACL platí, že se nezpracovávají poznámky (blok `[remark]`) ani blok `[optional parameters]`.

6.6.1 Standardní ACL

`access-list [number] [action] [source]`
Kompletní funkčnost pro všechny bloky standardních ACL.

6.6.2 Rozšířené ACL

Protokol TCP a UDP

`access-list [number] [action] [protocol] [source] [source port] [destination] [destination port]`
Kompletní funkčnost pro všechny bloky rozšířených ACL pro uvedené protokoly TCP a UDP.

Protokol ICMP

`access-list [number] [action] [protocol] [source] [destination] [ICMP message]`
Funkčnost pro akci, číslo seznamu, protokol, zdrojové a cílové adresy. Blok `[ICMP message]` zatím není podporován.

Protokoly IP, IGMP, EIGRP, OSPF, SCTP

`access-list [number] [action] [protocol] [source] [destination]`
Kompletní funkčnost pro všechny bloky rozšířených ACL pro uvedené protokoly. Obecně platí, že jakýkoli protokol, byť není v OMNeT++ zatím implementován, je schopen ACL modul pasivně zpracovat, je tak připraven na budoucí použití.

6.6.3 Pojmenovaná ACL

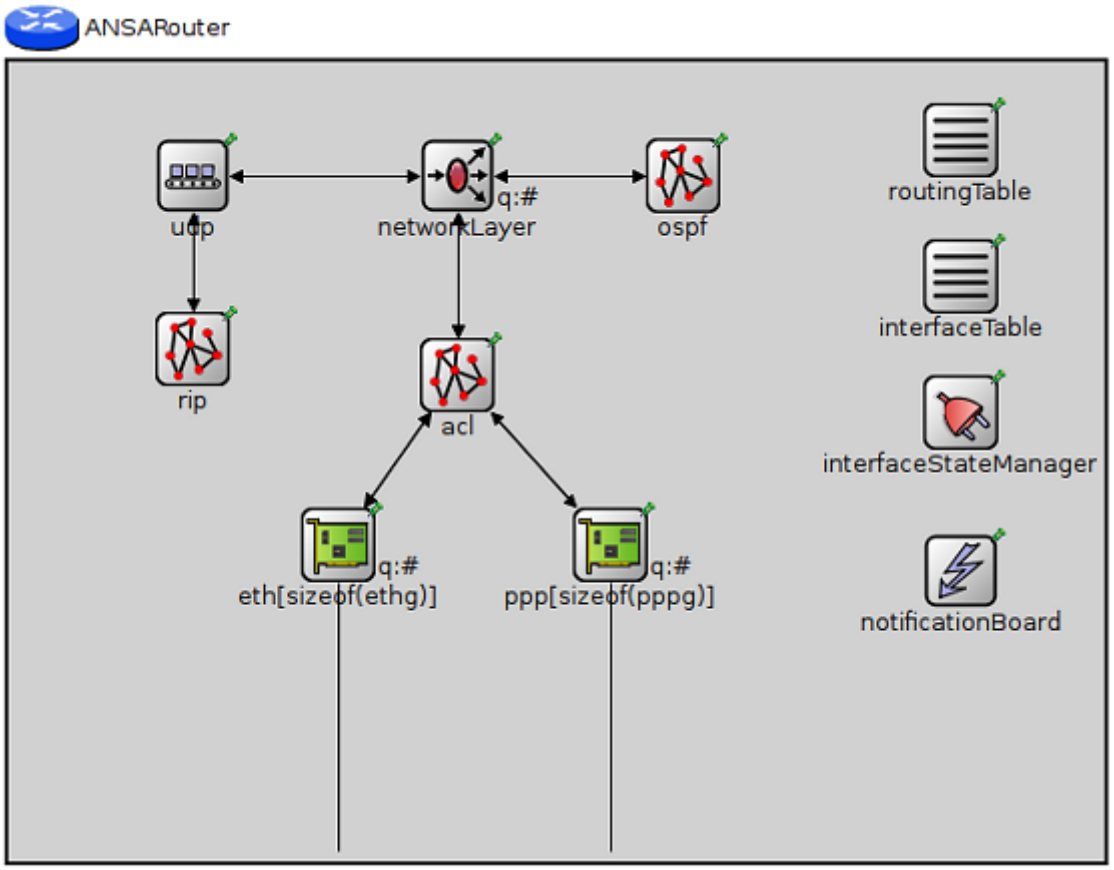
```
ip access-list [standard|extended] [name] [action] [protocol] [source]
[source port] [destination] [destination port]
```

Kompletní funkčnost pro všechny bloky pojmenovaných ACL pro všechny protokoly podobně jako u rozšířených ACL.

6.7 Aktivace ACL na rozhraní

Filtrovací modul ACL je zakomponován do mnohem obecnějšího modulu ANSARouter, jak ukazuje obrázek 6.3. Vlastní ACL je tedy, jak již víme, vloženo mezi linkovou a síťovou vrstvu. Pokud v konfiguračním souboru chybí části `<ACLs>` nebo `<ACL>`, je nastaven příznak `ACLEnabled` na hodnotu *false*. Pokud poté přichází na ACL modul zpráva (paket), nejprve se kontroluje, je-li `ACLEnabled` nastaveno na hodnotu *true*. Pokud ne, zpráva se jen přepošle bez kontroly dále. Pokud je konfigurace pro ACL obsažena v konfiguračním XML souboru, ACL je automaticky aktivováno a povoleno. Pokud však rozhraní, ze kterého přichází zpráva, nemá definováno ve směru komunikace žádný ACL seznam, potom se zpráva také pouze přepošle dále bez vykonání akce. Poslední možností je, že rozhraní má v požadovaném směru přiřazeno ACL seznam. V takovém případě je na něj daný seznam aplikován a paket se následně přepošle dále, nebo se zahodí.

Protože ACL modul dostává data pro simulaci v podobě jakou potřebuje od překladače Cisco konfigurace, neměl by se vyskytnout případ, že mu ke zpracování přijdou data, kterým nerozumí. Na vstupu však může přijít např. chybně pojmenovaný port či protokol. Pokud se tak stane, ACL modul zahlásí chybu při načítání konfigurace a simulace se ukončí.



Obrázek 6.3: Umístění ACL modulu v komponentě ANSARouter

6.8 Generování výsledků simulace, statistiky

Do naší implementace jsme zahrnuli i třídu, která umožňuje generovat statistiky po ukončení simulace. Po ukončení každé simulace vidíme, kolik IP datagramů bylo zpracováno ACL modulem, kolik z nich bylo přeposláno bez činnosti ACL filtru, kolik z nich bylo ACL filtrem povoleno a kolik zahozeno. Mimo jiné nám také umožňuje vidět, kolikrát bylo které pravidlo uplatněno na pakety proudící sítí. Pro implementaci těchto statistických funkcí jsme zavedli několik čítačů přímo ve třídě ACL, které inkrementujeme na základě provedené činnosti:

```
int numPackets;       // IPDatagrams arrived into ACL filtering module
int packetsDropped;   // packets dropped by an ACL action ,,deny‘‘
int packetsPermitted; // packets permitted by an ACL action ,,permit‘‘
int packetsAllowed;   // without ACL action (e.g. no ACL bound for packet)
```

Navíc jsme implementovali výbornou a zejména názornou funkci, která nám umožňuje sledovat statistiku aplikování jednotlivých pravidel z ACL seznamů na pakety, a to všechno v grafickém módu pro každý směrovač zvlášť. Díky tomu jsme schopni získat přehled o činnosti filtrovacího modulu v simulačním nástroji ve kterémkoli okamžiku simulačního času. Pro tyto účely využíváme vestavěných OMNeT++ metod, díky kterým dokážeme sledovat stav STL datových kontajnerů v C++. V inicializační fázi ACL modulu si definujeme nové

sledování `WATCH_LIST(stats)`, které nám bude bdít nad lineárním seznamem našich statistik. Vždy, když inkrementujeme čítač četnosti použití nějakého pravidla, tato změna se ihned projeví v našem statistickém modulu a jsme tak schopni přímo v průběhu simulace vidět počet použití všech pravidel ze všech seznamů na každém směrovači zapojeném do simulace.

6.9 Integrace modulu do OMNeT++, shrnutí

V této kapitole jsme si popsali veškeré náležitosti týkající se implementace filtrovacího modulu do prostředí simulačního nástroje OMNeT++. Počínaje návrhem modulu, volbou jeho umístění v komponentě CiscoRouter, přes problémy a jejich řešení, až po samotnou fázi implementace v jazyce C++. Demonstrovali jsme si některé použité datové struktury a proměnné, popsali si použité funkce a metody. Vysvětlili jsme si princip porovnávání IP adres za pomoci wildcard masky, popsali si filtrování portů a úroveň podpory implementovaných typů ACL. Naimplementovaný modul využívá knihovny, struktury, třídy, funkce a metody ze simulačního nástroje OMNeT++ a INET Frameworku, které usnadňují práci zejména s datovými typy jako jsou IP Adresa, maska, paket, zpráva a operace nad nimi. Modul je zaintegrován do prostředí simulačního nástroje OMNeT++ se všemi patřičnými náležitostmi. Veškeré komentáře zdrojových kódů jsou psány v anglickém jazyce kvůli případnému pozdějšímu využívání komunitou, což koresponduje se samotným nástojem OMNeT++.

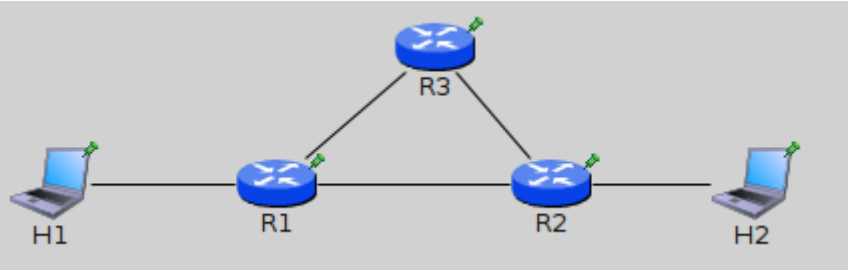
Kapitola 7

Testování na netriviální síťové konfiguraci

Tato kapitola se věnuje testování našeho modulu. Dále se zabývá ověřením korektní funkcionality ACL pravidel, to je zajištěno vytvořením simulace reálné sítě. Síť je záměrně vytvořena tak, aby se dalo jednoduše vypozorovat a zaznamenat její chování se zakomponovaným ACL filtrovacím modulem. Budeme sledovat pakety proudící sítí a budeme vyhodnocovat, kolik paketů bylo ACL modulem povoleno a kolik zahozeno. Také sledujeme, která konkrétní pravidla se v simulaci kolikrát uplatnila. Dosažené výsledky poté zaznamenáme do grafů pomocí nástroje integrovaného v simulátoru OMNeT++. Po celkovém vyhodnocení provedeme odpovídající test na reálném síťovém hardwaru v laboratoři a jeho výsledky porovnáme s výsledky dosaženými v námi vytvořené simulaci. Následující text je vhodně doplněn obrázky, které lépe znázorňují situaci v probíhající simulaci.

7.1 Popis vytvořené simulace

Síť, kterou budeme simulovat, je složena ze dvou počítačů a tří navzájem v kruhu propojených směrovačů s běžícím OSPF procesem. Na počítačích simulujeme TCP aplikaci (telnet) a UDP aplikaci. Dále mezi sebou komunikují i ICMP protokolem, pokud nějaký z poslaných paketů je z určitého důvodu nedoručitelný. Mezi směrovači jsou v rámci vytváření síťové topologie posílány OSPF pakety pomocí multicastu. Díky nim si udržují aktuální směrovací tabulku a mají přehled o sousedních směrovačích a jejich metrikách. Schéma simulace popisuje obrázek 7.1.



Obrázek 7.1: Simulační schéma testované sítě

Vzhledem k tomu, že jsme záměrně zvolili kruhové zapojení se 3 směrovači, můžeme sledovat chování sítě, pokud ACL pravidlem zakážeme komunikaci směrovačů R1 a R2, tedy těch s lepší metrikou. Poté se pakety budou posílat po trase R1 - R3 - R2.

7.2 Testovací konfigurace „Povolit vše“

7.2.1 Parametry simulace

Jako první testovací konfiguraci zvolím takové nastavení ACL, kde bude na každém ze tří směrovačů zadána následující série příkazů:

```
access-list 101 permit ip any any
interface eth0
ip access-group 101 in
ip access-group 101 out
interface eth1
ip access-group 101 in
ip access-group 101 out
interface eth2
ip access-group 101 in
ip access-group 101 out
```

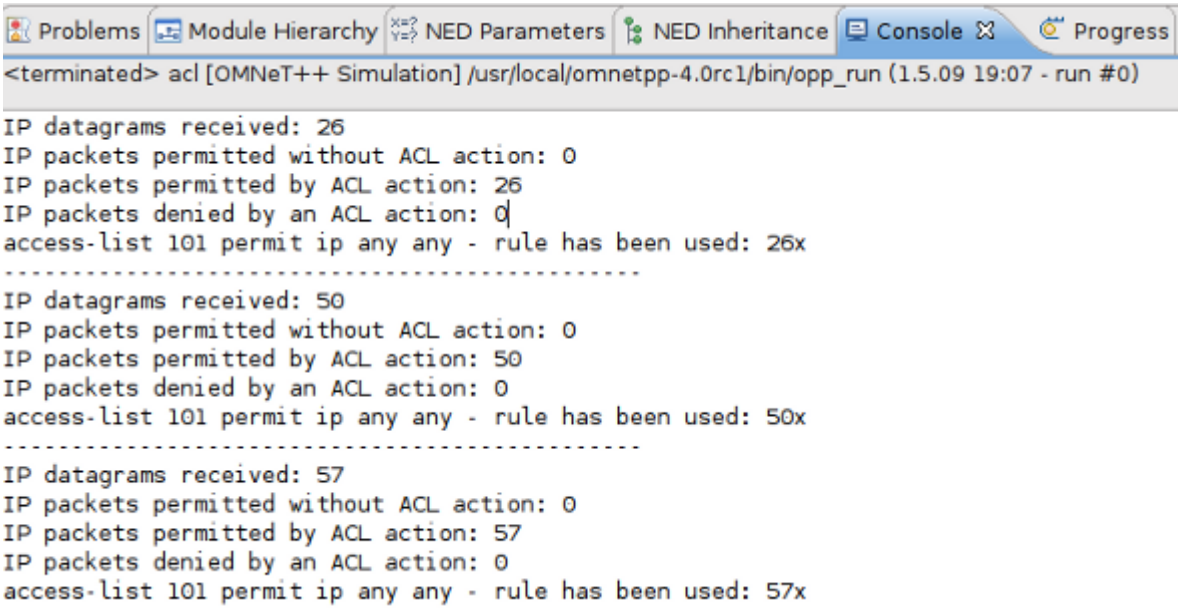
Vidíme tedy, že ACL seznam 101 bude přiřazen pro všechna rozhraní směrovače v obou směrech. Tím zajistíme, že bude povolen jakýkoli paket protokolu IP z libovolné zdrojové adresy na libovolnou cílovou adresu, tedy všechny pakety budou povoleny. Následuje ukázka transformace příkazů do XML konfiguračního souboru pro směrovač R1 tak, aby odpovídal implementaci ACL modulu. U ostatních směrovačů R2 a R3 je uplatněn stejný postup.

```
<Routers>
  <Router id=,,192.168.3.3‘‘> <!-- R1 -->
<ACLs>
    <ACL no=,,101‘‘>
      <entry seq_no=,,10‘‘>
        <action>permit</action>
        <IP_src>0.0.0.0</IP_src>
        <IP_dst>0.0.0.0</IP_dst>
        <WC_src>255.255.255.255</WC_src>
        <WC_dst>255.255.255.255</WC_dst>
        <port_op_src></port_op_src>
        <port_op_dst></port_op_dst>
        <port_beg_src></port_beg_src>
        <port_end_src></port_end_src>
        <port_beg_dst></port_beg_dst>
        <port_end_dst></port_end_dst>
        <protocol>ip</protocol>
      </entry>
      <interfaces>
        <interface dir=,,in‘‘>eth0</interface>
        <interface dir=,,out‘‘>eth0</interface>
```

```
        <interface dir=,,in‘‘>eth1</interface>
        <interface dir=,,out‘‘>eth1</interface>
        <interface dir=,,in‘‘>eth2</interface>
        <interface dir=,,out‘‘>eth2</interface>
      </interfaces>
    </ACL>
</ACLs>
  </Router>
<Routers>
```

7.2.2 Výsledky simulace

Po skončení simulace můžeme vidět na programovém výstupu, kolik IP paketů prošlo ACL filtrovacím modulem, kolik z nich bylo povoleno, nebo zahozeno, nebo nefiltrováno. U alespoň jedenkrát použitých pravidel vidíme, které bylo kolikrát v rámci simulace na pakety uplatněno. Síťová komunikace zahrnuje protokoly OSPF, TCP, UDP i ICMP. Výstup do konzole po skončení simulace v nástroji OMNeT++ je znázorněn na obrázku 7.2.



Obrázek 7.2: Výstup simulace #1 do konzole

Vidíme, že na všech třech směrovačích byly veškeré příchozí i odchozí IP datagramy povoleny ACL pravidlem. Můžeme i vyčíst, kterým pravidlem a kolikrát bylo dané pravidlo použito. V našem případě bylo na veškerý IP provoz aplikováno pravidlo `permit ip any any`. Z výstupu také vidíme, že žádný paket ze neztratil. Dokázali jsme, že tato simulace dle zadané konfigurace měla stejné chování jako reálná síť, což jsme si ověřili pokusem v laboratoři.
Pravidlo `permit ip any any`, pokud je uvedeno na začátku seznamu, skutečně veškerý provoz povoluje.

7.3 Testovací konfigurace „Zakázat vše“

7.3.1 Parametry simulace

Při druhé testovací konfiguraci bude na každém ze tří směrovačů zadána následující série příkazů:

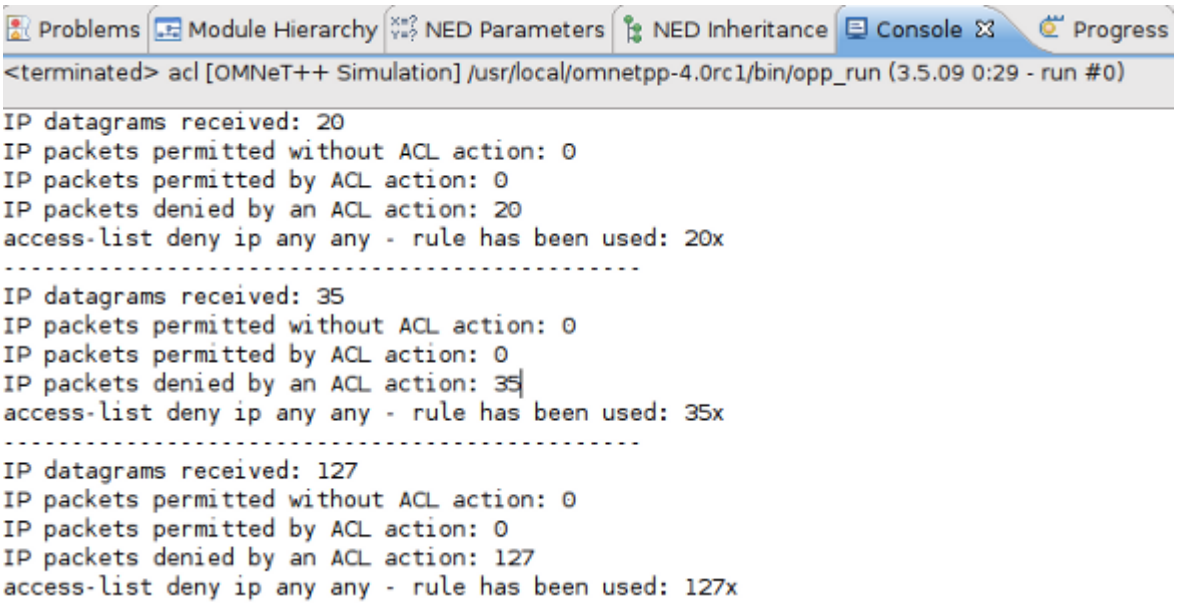
```
access-list 102 permit tcp host 192.168.10.100 eq 80 any
interface eth0
ip access-group 102 in
interface eth1
ip access-group 102 out
interface eth2
ip access-group 102 in
ip access-group 102 out
```

Tento příkaz jsme zvolili záměrně, protože takovou IP adresu nemá žádné zařízení v námi vytvořené simulaci. V praxi to znamená, že se toto pravidlo nikdy nepoužije a na řadu tak přijde implicitní pravidlo `deny ip any any`, které blokuje veškerý IP provoz. Můžeme tedy říci, že tento test jsme vybrali pro ověření korektní funkce blokování paketů pomocí ACL a zároveň na otestování aplikace implicitního pravidla.

Sérii příkazů transformujeme do XML stejným způsobem jako při konfiguraci v sekci 7.2.1.

7.3.2 Výsledky simulace

Po skončení simulace můžeme opět vidět na programovém výstupu, jak bylo ACL na pakety používáno. Výstup do konzole po skončení této simulace v nástroji OMNeT++ je znázorněn na obrázku 7.3.



Obrázek 7.3: Výstup simulace #2 do konzole

Vidíme, že veškeré pakety byly zahozeny v důsledku uplatnění implicitního pravidla `deny ip any any`. Ke stejnému závěru jsme dospěli i ověřením testu v síťové laboratoři.

7.4 Testovací konfigurace „Kombinace více pravidel“

7.4.1 Parametry simulace

V poslední simulaci se pokusíme simulovat stav co nejbližší běžně používanému filtrování v praxi. Proto na každém ze směrovačů nastavíme příkazy, které povolují, nebo blokují provoz pouze na požadovaném rozhraní v určitém směru tak, aby konfigurace dávala smysl a byla o něčem vypovídající.

Síť, kterou simulujeme, vypadá tak jako na obrázku 7.1. Na směrovači R1 zadáme následující sérii příkazů:

```
access-list 110 permit udp host 192.168.1.2 192.168.2.0 0.0.0.255
access-list 110 deny icmp host 192.168.1.2 192.168.2.0 0.0.0.255
interface eth0
ip access-group 110 in
```

Těmito příkazy povolíme pouze UDP pakety, které přichází na směrovač R1 a které pochází z počítače H1 a jsou určeny pro síť 192.168.2.0/24. V cílové síti se nachází i druhý počítač, H2.

Následně na směrovači R2 úplně zakážeme výměnu OSPF paketů se směrovačem R1, čímž docílíme stavu, kdy bude veškerý provoz chodit přes směrovač R3. Také povolíme libovolné TCP a ICMP pakety, naopak zakážeme UDP pakety příchozí na rozhraní směrovače R2 z počítače H2 pocházející ze sítě 192.225.2.0 s wildcard maskou 0.255.0.255. V pravidle o zákazu UDP paketů je záměrně v druhém oktetu IP adresy číslo 225, jelikož je totiž ve wildcard masce v odpovídajícím oktetu číslo 255, víme, že se nebude kontrolovat ani 1 bit z druhého oktetu, jinými slovy ve druhém oktetu může být libovolné číslo 0-255. Tímto pravidlem si zároveň ověříme korektní činnost při zpracování wildcard masek. Zdrojový port zakázaných UDP paketů je 25. Za tímto účelem transformujeme všechny následující příkazy na směrovači R2 do konfiguračního souboru:

```
access-list 120 deny ospf any any
access-list 130 deny udp 192.225.2.0 0.255.0.255 eq 25 any
access-list 130 permit tcp 192.168.2.0 0.0.0.255 any
access-list 130 permit icmp 192.168.2.0 0.0.0.255 any
interface eth0
ip access-group 120 in
ip access-group 120 out
interface eth2
ip access-group 130 in
```

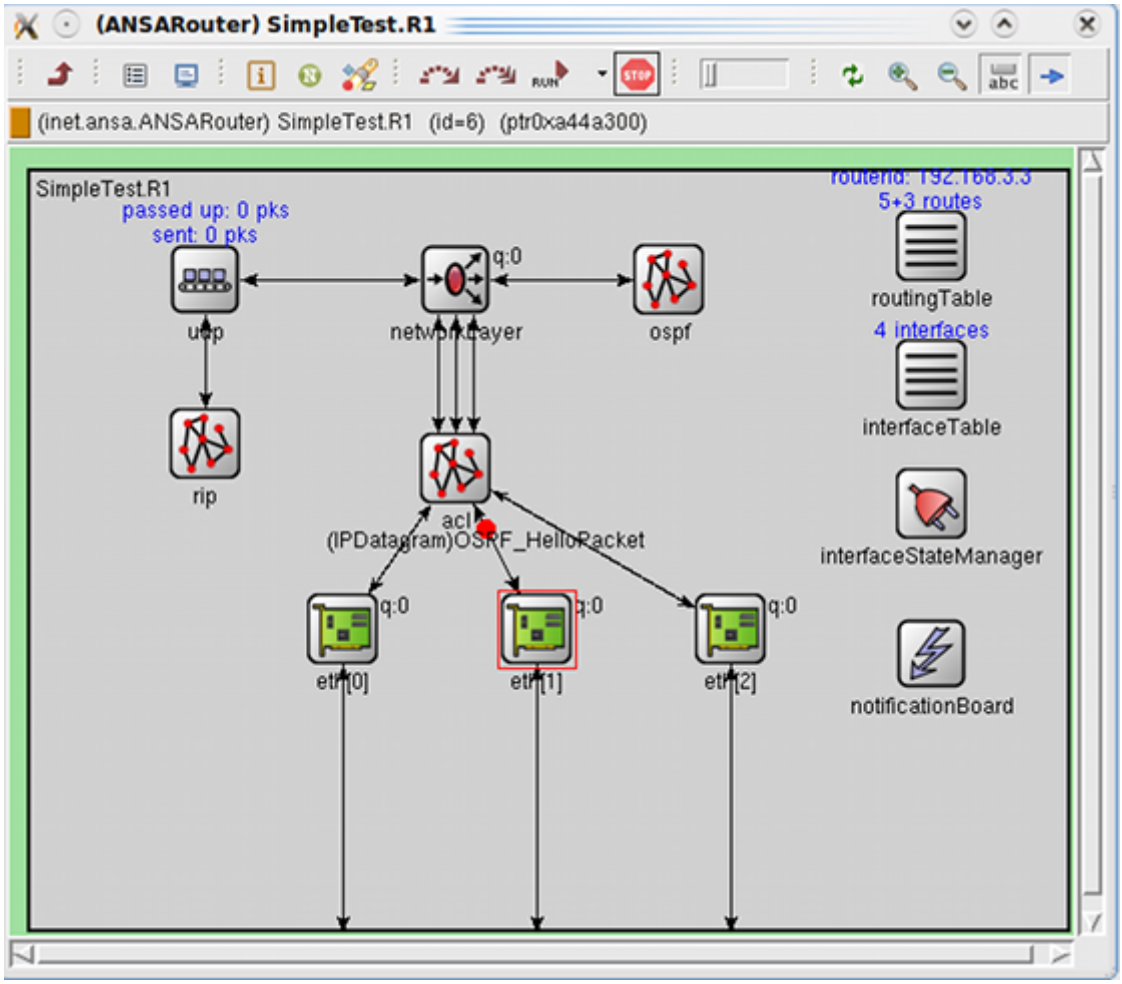
Na rozhraních a ve směrech, kde není definován žádný ACL seznam, jsou pakety automaticky povoleny, tedy přeposlány dále, bez činnosti ACL filtru. Takové chování je standardní podle Cisco ACL, viz. kapitola 3.

Na směrovači R3 chceme povolit veškerou komunikaci, proto mu přiřadíme na všechna jeho vstupní rozhraní pravidlo `permit any`.

```
access-list 10 permit any
interface eth0
ip access-group 10 in
interface eth1
ip access-group 10 in
```

7.4.2 Průběh simulace

Po spuštění simulace se nejprve inicializuje celá síť modulů včetně ACL filtrování. ACL modul si načte data z konfiguračního XML souboru a aktivuje se jako komponenta v simulačním schématu. ACL je přítomno na každém směrovači zapojeném v simulaci, v našem případě tedy na R1, R2 i R3. Na obrázku 7.4 vidíme, jak je OSPF Hello paket po povolení ACL modulem přeposílán na rozhraní *eth*1 routeru R1. Je zde také vidět, že ACL tvoří mezistupeň mezi linkovou a síťovou vrstvou.



Obrázek 7.4: ACL modul jako aktivní komponenta v simulaci

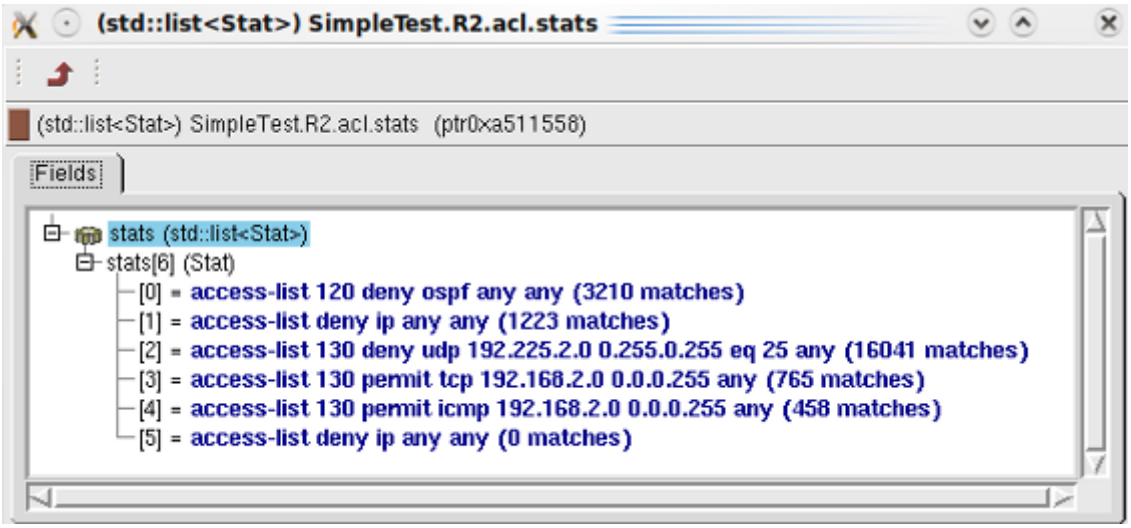
Na obrázku 7.5 můžeme vidět zpracování paketu ACL modulem přímo v probíhající simulaci. Konkrétně se jedná o UDP paket z adresy 192.168.2.2 směrující na 192.168.1.2. Zdrojový port je 25, cílový potom 6112. Protokol je UDP (17). Můžeme vidět jednotlivá

porovnání oktetů IP adres a konečný výsledek - permit. Také zde je vidět, že paket přišel ze spodní vrsty a je určen pro vrstvu síťovou, kam je také dále přeposlán. Takový výstup přímo na event log simulátoru je součástí naší implementace.

```
** Event #2181  T=27.000007939999  SimpleTest.R2.acl (acl, id=77), on `UDPBasicAppData-26' (IPDatagram, id=820)
acl::handleMessage
acl::getRules
An ACL list bound correctly for this router/interface.
acl::processPacket
Packet Source IP Address: 192.168.2.2
Packet Dest.  IP Address: 192.168.1.2
Packet Protocol ID      : 17
UDP packet, source port: 25
UDP packet, dest.  port: 6112
acl::filterPacket
acl::filterPacket: PROTOCOL MATCH
ACL ipAddr(0): 0
pkt ipAddr(0): 192
ACL netmask(0): 0
ACL ipAddr(1): 0
pkt ipAddr(1): 168
ACL netmask(1): 0
ACL ipAddr(2): 0
pkt ipAddr(2): 2
ACL netmask(2): 0
ACL ipAddr(3): 0
pkt ipAddr(3): 2
ACL netmask(3): 0
IP is all OK.
acl::filterPacket: SOURCE IP MATCH
ACL ipAddr(0): 0
pkt ipAddr(0): 192
ACL netmask(0): 0
ACL ipAddr(1): 0
pkt ipAddr(1): 168
ACL netmask(1): 0
ACL ipAddr(2): 0
pkt ipAddr(2): 1
ACL netmask(2): 0
ACL ipAddr(3): 0
pkt ipAddr(3): 2
ACL netmask(3): 0
IP is all OK.
acl::filterPacket: DESTINATION IP MATCH
+ACL ACTION:  PERMIT +++ from interface out to networkLayer
```

Obrázek 7.5: Zachycená událost paketu povoleného ACL filtrem

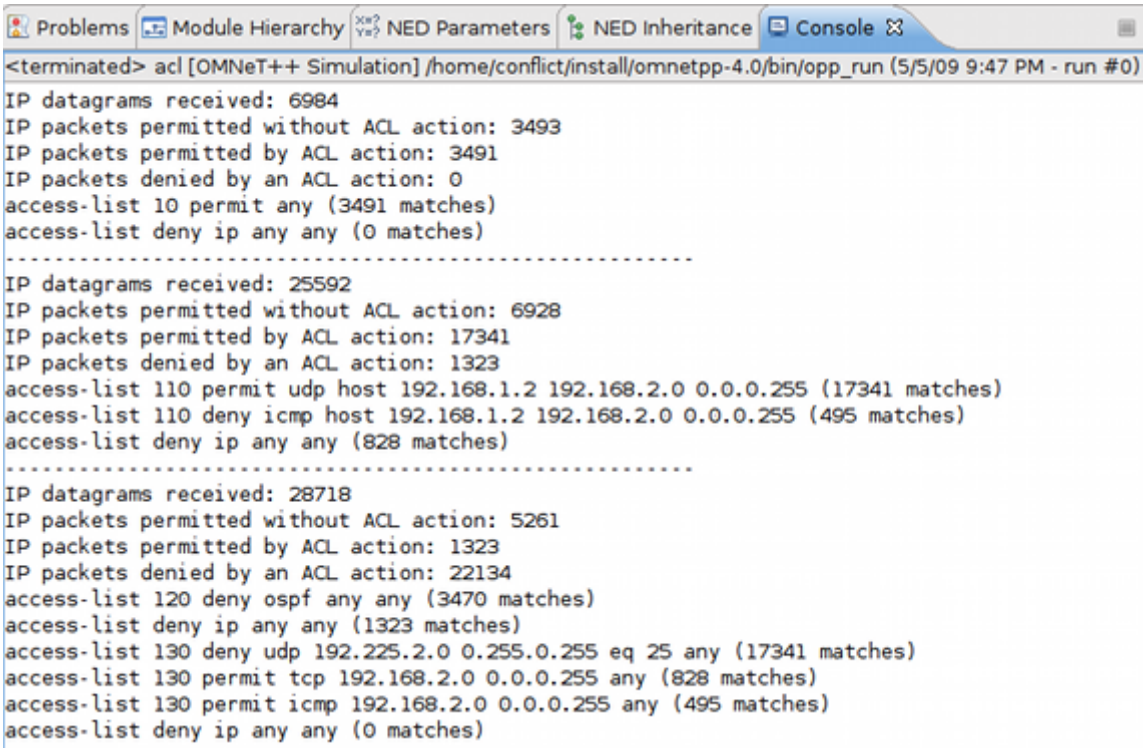
Přímo v průběhu simulace také můžeme vidět na grafickém výstupu četnost aktuálně aplikovaných ACL pravidel na každém ze směrovačů. Na obrázku 7.6 vidíme průběžnou statistiku ACL filtrování na směrovači R2 a aplikovatelnost všech pravidel v aktuálně běžící simulaci. Pokud si necháme stastické okno otevřeno v GUI samotné simulace a přitom necháme simulaci běžet, jsme schopni pozorovat zvyšování četnosti použití jednotlivých pravidel se zvyšujícím se počtem filtrovaných paketů. Tuto metodu sledování činnosti ACL za běhu simulace považujeme za velice přínosnou. Způsob výpisu počtu použití ACL pravidel odpovídá výpisu na Cisco směrovačích.



Obrázek 7.6: Četnost aplikovaných ACL pravidel za běhu simulace

7.4.3 Výsledky simulace

Po skončení simulace můžeme opět vidět na programovém výstupu, jak bylo ACL na pakety používáno. Výstup do konzole po skončení této simulace v nástroji OMNeT++ je znázorněn na obrázku 7.7.



Obrázek 7.7: Výstup simulace #3 do konzole

Skutečně se správně uplatnila všechna pravidla obsažená v konfiguraci všech tří směrovačů. Na těch rozhraních, kde v nějakém směru nebyl přiřazen žádný ACL seznam, se pakety propustily bez zásahu ACL filtrování. V nastavení simulace jsme si zvolili nejčastější posílání UDP paketů, proto i jejich zahazování bylo nejčastější na směrovači R2. Ověřili jsme si, že jsme schopni odsimulovat reálnou síť s konfigurací ACL co nejbližší reálému stavu, tedy kombinací více různých seznamů s více pravidly přiřazených k rozličným rozhraním. Modul byl schopen filtrovat všechny pakety účinně dle zadané konfigurace.

Také jsme zjistili, že při zákazu OSPF protokolu mezi sousedními směrovači R1 a R2 budou data sítí skutečně proudit „oklikou“ přes směrovač R3 a až poté na R2.

7.5 Shrnutí

V této příkladové kapitole jsme prakticky ukázali, jakým způsobem provést simulaci pouze na základě Cisco IOS příkazů. Příkazy nejprve transforujeme do XML konfiguračního souboru, který poté dále použijeme při získávání parametrů pro jednotlivé simulační komponenty. Následně vytvoříme simulační model, v prostředí OMNeT++ graficky. Vygenerovaný popis simulace můžeme případně změnit editací souboru `omnetpp.ini`. Poté simulaci spustíme a na základě výstupů z chování sítě můžeme provést rozbor dané simulace.

Závěr

Cílem bakalářské práce bylo rozšíření simulačního nástroje OMNeT++ o modul, který dokáže filtrovat provoz v síti pomocí ACL pravidel. To zahrnovalo vytvoření zcela nového modulu, který se dále musel zaintegrovat do obecnějšího modulu CiscoRouter.

Filtrování paketů pomocí ACL

Prvním nutným krokem bylo zorientovat se v prostředí simulačního nástroje OMNeT++. Po prozkoumání knihoven simulátoru a Frameworku INET jsme navrhli strukturu modulu pro zpracování filtrovacích pravidel ACL. Na základě tohoto návrhu jsme ACL filtrování naimplementovali jako komponentu v jazyce C++. Vypořádali jsme se s problémem, kam filtrovací komponentu v simulačním schématu umístit. Modul je schopen si sám načítat transformované Cisco ACL příkazy z globálního konfiguračního souboru. Pro ACL seznamy jsme diskutovali výběr vhodné datové struktury pro reprezentaci filtrovacích pravidel. Teoreticky jsme popsali princip použití intervalových rozhodovacích diagramů a naznačili možnost budoucí implementace IDD strukturami, přičemž úlohu reprezentace dat splnil spojový lineární seznam. Naimplementovali jsme principielně ekvivalentní systém filtrace paketů pomocí ACL, který najdeme na Cisco směrovačích. Jediným omezením je nedostatečná podpora simulátoru pro některé vedlejší parametry v ACL (např. definice skutečných časů), které sice snížily úroveň podpory některých typů ACL, nicméně nijak nebránily inspekci paketů podle jejich zásadních parametrů. Naše implementace ACL je tedy schopna kompletně filtrovat pakety na základě:

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

Podporované typy ACL jsou následující:

- Standardní ACL
- Rozšířené ACL
- Pojmenované ACL

Využití výsledků

Námi dosažené výsledky se dají považovat za úspěch, chování simulace se zakomponovaným filtrovacím modulem koresponduje s chováním reálné sítě. V průběhu testu nedocházelo

k chybám, implementace ACL modulu je tedy korektní. Kromě grafické animace paketů proudící sítí přímo za běhu simulace, požadovaným výsledkem je přehledný textový výpis zobrazený na konzolovém výstupu po ukončení simulace. Na tomto výstupu je možno vidět, kolikrát bylo které ACL pravidlo aplikováno, kolik paketů bylo propuštěno (a kolik z toho bylo díky ACL filtrování) a kolik zahozeno. Tyto informace lze navíc sledovat v grafickém uživatelském rozhraní přímo za běhu simulace s aktuálností odpovídající simulačnímu času.

Pokud provedeme analýzu těchto výsledků z bezpečnostního hlediska, můžeme vidět, jaký síťový provoz je povolen a jaký naopak blokován. To nám může napovědět o stavu sítě; je možné například vysledovat, že sítí jsou propouštěny nebezpečné pakety, na což můžeme pružně reagovat změnou v konfiguraci simulace a opětovným testem.

Další vývoj projektu

Další vývoj projektu by se mohl zabývat, jak již bylo naznačeno, vytvořením efektivnější reprezentace ACL pravidel pomocí intervalových rozhodovacích diagramů (IDD). Z pohledu ACL bychom mohli rozšířit podporu implementovaných typů access-listů i na ty méně používanější. Také by nebylo na škodu rozšířit množinu současně implementovaných protokolů nebo vytvořit nové, abychom mohli využívat veškerých možností, co ACL nabízí.

Dále by bylo možné v rámci společného ANSA projektu rozšířit simulační nástroj OMNeT++ o další potřebné moduly, například o BGP směrování v rozsáhlých sítích, o modul překladu adres NAT, aj.

Literatura

[1] Firewall [online]. [cit. 2009-01-07].
URL http://www.cisco.com/univercd/cc/td/doc/product/iaabu/centri4/user/scf4ch3.htm

[2] Project ANSA[online]. February 2009 [cit. 2009-05-09].
URL http://nes.fit.vutbr.cz/ansa

[3] Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, vol. C-35-8, pp. 677-691, 1986.

[4] Christiansen, M.; Fleury, E.: An Interval Decision Diagram Based Firewall. In 3rd International Conference on Networking (ICN'04), IEEE, Feb. 2004.

[5] Cisco: Configuring IP Access Lists [online]. 2007-12-27 [cit. 2009-01-24].
URL
http://www.cisco.com/application/pdf/paws/23602/confaccesslists.pdf

[6] Grygárek, P.: Bezpečnost počítačových sítí [online]. 2004-11-02 [cit. 2009-01-22].
URL http://www.cs.vsb.cz/grygarek/LAN/lect/bezpecnost.pdf

[7] IANA: Assigned Internet Protocol Numbers [online]. March 2009 [cit. 2009-04-27].
URL http://www.iana.org/assignments/protocol-numbers/

[8] Matoušek, P.; Ráb, J.; Ryšavý, O.; aj.: A Formal Model for Network-wide Security Analysis. 2007.

[9] Scherfel, P.: Simulace chování sítě na základě analýzy konfiguračních souborů aktivních síťových zařízení. Bakalářská práce, FIT VUT Brno, 2009.

[10] nsnam Sourceforge project: The Network Simulator - ns-2 [online]. 2009-01-15 [cit. 2009-01-31].
URL http://isi.edu/nsnam/ns/

[11] Strehl, K.; Thiele, L.: Symbolic Model Checking using interval diagram techniques. Technical Report 40, Computer Engineering and Networks Lab (TIK), ETH Zurich, Feb. 1998.

[12] Varga, A.: What is OMNeT++? [online]. 2009-01-01 [cit. 2009-01-31].
URL http://www.omnetpp.org/external/whatis.php

Dodatek A

Obsah CD

- `\omnetpp-4.0-src.tgz` - instalační balík simulátoru OMNeT++ pro Linux
- `\INETMANET-20080920.tbz2` - instalační balík INET-MANET Framework pro OMNeT++
- `\projekt.pdf` - vlastní text technické zprávy v PDF (verze odevzdaná do WISu)
- `\src\ansa\` - zdrojové kódy všech potřebných modulů v ANSA projektu (včetně ACL modulu)
- `\examples\AnsaExamples\` - funkční vytvořené simulace pro ukázku