VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELY SMĚROVACÍCH PROTOKOLŮ PRO SÍŤOVÝ SIMULÁTOR A ANALÝZU CHOVÁNÍ SÍTĚ

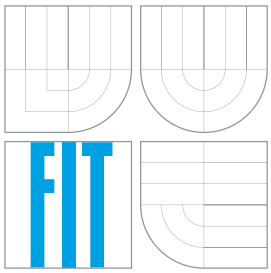
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN ŠAFÁŘ

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELY SMĚROVACÍCH PROTOKOLŮ PRO SÍŤOVÝ SIMULÁTOR A ANALÝZU CHOVÁNÍ SÍTĚ

MODELS OF ROUTING PROTOCOLS FOR NETWORK SIMULATION AND ANALYZE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN ŠAFÁŘ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV RÁB
SUPERVISOR

BRNO 2008

Abstrakt

Práce se zabývá způsobem implementace směrovacího protokolu RIP do simulátoru sítě ns-2

Klíčová slova

NS2, RIP, simulace, implementace směrovacího protokolu

Abstract

Thesis describes the way of implementation of new routing protocol for network simulator ns-2

Keywords

NS2, RIP, simulation, routing protocol implementation

Citace

Jan Šafář: Modely směrovacích protokolů pro síťový simulátor a analýzu chování sítě, bakalářská práce, Brno, FIT VUT v Brně, 2008

Modely směrovacích protokolů pro síťový simulátor a analýzu chování sítě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Jaroslava Rába

.....
Jan Šafář
9. května 2008

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

Obsah

Kapitola 1

Úvod

Motivace

Tématem této práce bylo zdokumentovat současné možnosti síťového simulátoru ns-2 simulovat směrování v IPv4 sítích. Tyto možnosti měli být rozšířeny o skutečný směrovací protokol RIP společně s uživatelským rozhraním podobnému tomu na síťových prvcích společnosti Cisco. Společně s implementací by tato práce měla být návodem pro další rozšiřovaní síťového simulátoru.

Na tuto práci navazuje implementace přístupových seznamů řešených v rámci diplomové práce Bc. Jiřím Macků. Tato práce společně s přístupovými seznamy by měli být použity v rámci projektu ANSA, Automated Network-wide Security Analysis, kde by měly sloužit jako součást procedury analýzy bezpečnosti sítě.

Struktura dokumentu

Kapitola 'Směrovací protokoly' slouží jako představení různých směrovacích protokolů a jejich chování. Důraz je kladen na distance vector protokoly a konkrétně na RIP, z důvodu, že byl předmětem implementace. Dále je zde uveden způsob konfigurace RIPu na Cisco zařízeních.

V kapitole 'Simulátor sítě ns-2' je popsán simulátor ns-2, jeho základní prvky, popis implementace směrování a rozdíl vůči IPv4 prostředí.

'Návrh a implementace' obsahuje definici vlastností, které by měla implementace obsahovat. Dále pak popisuje samotnou implementaci směrovacího protokolu společně s jeho integrací do stávající struktury síťového simulátoru ns-2.

Kapitola 'Testy' diskutuje výsledky testů implementace vůči reálnému prostředí.

Kapitola 2

Směrovací protokoly

Úvod

Před samotnou implementací směrovacího protokolu je nutné pochopit k čemu vlastně směrovací protokoly slouží a jakým způsobem protokoly pracují. Stejně tak je nutné prostudovat jejich konfiguraci na Cisco zařízeních, jelikož uživatelské rozhraní, které se bude implementovat má být podobné rozhraní na Cisco zařízení.

2.1 Směrování

Směrování je proces výběru cesty od zdroje k zadanému cíli skrz obecnou síť. Tento dokument je ale konkrétně zaměřen na sítě počítačové. V počítačových sítích se procesu směrování účastní všechny prvky a předávají si informace ve formě paketů. Prvky rozumíme směrovače, přepínače, PC atd. Prvky však většinou nehledají celou cestu k cíli z důvodu, že může být až příliš složitá, ale pouze se rozhodnout na základě směrovací tabulky, kterému dalšímu prvku paket předají.

Tabulka 2.1: Příklad směrovací tabulky

Identifikátor cíle	Cena	Další uzel
10.0.0.0/8	1	192.168.3.6
10.3.0.0/16	2	192.168.2.1

Směrovací tabulka je datová struktura jejíž záznamy obsahují identifikátor (ip adresu) cíle, cenu cesty k dosažení tohoto cíle a další uzel na cestě k jeho dosažení. Při vyhledávání ve směrovací tabulce mají přednost záznamy s větším síťovým prefixem, tzn. specifičtější záznamy. Síťový prefix je zkrácený zápis síťové masky, přesněji součet jedniček v ní. Síťová maska vymezuje, která část z adresy je adresa sítě a která adresa prvku, pomocí bitové operace AND. V tabulce 2.2 je vidět příklad použití těchto pojmů. Zkráceně zapisujeme záznamy ve tvaru síť/prefix.

Pokud se pomocí směrovací tabulky z obrázku 2.1 bude vyhledávat další skok pro paket, s cílovou adresou prvku 10.3.1.4, bude vybrán záznam 10.3.0.0/16, i když by mohl být poslán do sítě 10.0.0.0/8, na základě vyššího prefixu

Otázka je, na základě čeho si každý prvek vytvoří svojí směrovací tabulku? Může být samozřejmě vytvořena ručně, u osobních počítačů s jedním síťovým je to poměrně jedno-

Tabulka 2.2: Aplikace síťové masky

IP adresa	00001100.00000010.00000101.00100101
Síťová maska	11111111.11111111.11111100.00000000
Síť	00001100.00000010.00000100.00000000
Prefix	22

duché, stačí definovat cestu k bráně, na kterou mají směrovat všechny pakety, které nejsou určené pro připojenou síť. U zařízení s více rozhraními není situace tak snadná, protože administrátor musí definovat cesty ke všem sítím do kterých chce mít přístup. A pokud velikost směrovací tabulky přesáhne určitou velikost, může se stát její správa náročnou. Tomuto způsobu se říká manuální směrovaní, protože může změnit pouze lidským zásahem. Je možné se setkat i s pojmem statické směrování.

Východiskem pro prvky, kde by ruční definování tabulky bylo příliš zdlouhavé nebo nevýhodné, leží ve směrovacích protokolech. Směrovací protokoly slouží k výměně směrovacích informací mezi uzly v sítí a tedy k propagaci cesty k cílům. Jejich výhodou oproti manuálnímu směrování je, že mohou aktivně reagovat na změny v sítí a měnit směrovací tabulku na základě těchto změn, například výpadek prvku nebo linky. Z toho důvodu se směrování pomocí směrovacích protokolů nazývá dynamické.

Směrovací protokoly v prostředí internetu se dají rozdělit, na základě parametrů, do různých skupin. Nejčastěji se rozdělují podle místa použití a to do 2 skupin. EGP, exterior gateway protocol nebo-li vnější směrovací protokol, slouží ke směrovaní mezi autonomními systémy, zkráceně AS. AS je část sítě s jednotnou směrovací politikou vůči zbytku internetu, který je vlastně sítí autonomních systémů. Typicky tuto politiku řídí nějaká organizace, například poskytovatel internetu, ISP.

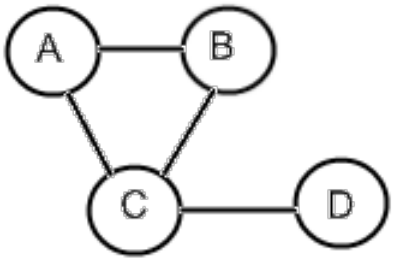
Druhou skupinou jsou IGP, interior gateway protocol nebo-li vnitřní směrovací protokol, který se používá pro směrování uvnitř autonomního systému.

Také se dají rozdělit podle způsobu jakým jednotlivé uzly vyměňují směrovací informace.

2.2 Distance vector protokoly

Distance vektor protokoly využívají Bellman-Fordův algoritmus pro výpočet směrovací tabulky. Protokoly pracují na jednoduchém principu, prvek zná nejprve pouze připojené sítě a cenu cesty k nim. Tyto informace periodicky, po určitém definovaném intervalu, rozesílá svým sousedům, na kterých také běží instance tohoto protokolu. Prvky si uchovávají informaci o nejlepší cestě k danému cíli nebo cenu ke všem cílům skrz všechny své sousedy, oba přístupy jsou možné. Prvku, kterému dojde zpráva od procesu na jiném prvku si zaktualizuje informace, které má ve své databázi sítí, které se dovídá prostřednictvím daného souseda. Pokud si uchovává pouze nejlepší cestu, zkontroluje zda cesta ve zprávě má lepší, menší, metriku. Takto naučené informace propaguje dál, ale pouze nejlepší variantu pro daný cíl. Směrovací tabulka obsahuje pouze nejlepší varianty.

Kvůli takovému způsobu učení sítí, nazýváme tento typ směrování jako 'směrování pomocí zvěsti' (routing by rumor), protože uzel důvěřuje informacím, které zaslechl od svého souseda. Bohužel to vede k problému počítání do nekonečna (count-to-infinity problem).



Obrázek 2.1: Počítání do nekonečna

Počítání do nekonečna

Problém počítání do nekonečna bude vysvětlen pomocí příkladu, v kterém použijeme topologii z obrázku 2.1.

Cestu k D, popřípadě do sítě mezi C a D, zná z počátku pouze C. Ta propaguje tuto informaci ke svým sousedům A a B. Předpokládejme, že zvýšení ceny za přechod přes jednu linku je 1. A a B tedy ví, že cesta k D je přes C s cenou 2. Tuto informaci dále propagují. Výsledná tabulka informací pro uzel C je vidět v tabulce 2.2. Co se ale stane pří výpadku linky mezi C a D? Uzel C si smaže informace o cestách skrz D. Od uzlu A nebo B se dozví, že nejlepší cena do D je 2 skrz C, protože oni se neměli jak dozvědět, že linka mezi C a D je nefunkční. C si aktualizuje směrovací tabulku, že D je k dispozici skrz A, popřípadě B, a tuto metriku, 3 (2 + 1 za cenu linky), odesílá dál. A a B si poté aktualizují záznamy o D skrz A, ta stoupne ze 2 na 4 (3 + 1 za cenu linky), C dostane zase tuto informace zvýší cenu na 5 a propaguje dál až do nekonečna. Pakety určené pro D mezitím kolují mezi těmito 3 uzly a zbytečně vytěžují zdroje.

	A	B	D
skrz A	1	2	2
srkz B	2	1	2
srkz D	3	3	1

Z důvodu, že se s tímto problémem setkávali distance vector protokoly již od svého vzniku, a toto chování bylo značně nežádoucí, vzniklo několik technik, které se snaží zabránit tomuto škodlivému chování.

Maximální metrika - maximum count

Pokud cena cesta k sítí přesáhne určitou mez, je považována za nedostupnou a pakety určené pro ní se zahazují. Více jak maximální metrika se nikdy nepropaguje.

Rozdělený horizont - split horizont

Technika, která zabraňuje odesílání informací o cestách k sítím zpět uzlu od kterého byli získaný. Pokud se tedy vrátíme k obrázku 2.1, A a B nebudou informovat C o cestě k D, protože tuto informaci získali od C.

Zneplatnění zpětné cesty - poison reverse

Tato technika se používá společně s technikou rozděleného horizontu. Informace o sítích se znovu propagují zpět k prvku od kterého byly získány, ale s metrikou, která v daném protokolu značí nedostupnost.

Aktualizace nezávisle na časovači - triggered updates

Kdykoliv se prvku změní metrika do určité sítě, informuje okamžitě o této změně své sousedy, bez toho aby čekal na dobu kdy má posílat pravidelnou aktualizaci.

Další časovače - timers

Časovač zneplatnění cesty (invalid timer), vyprší pokud síť, nebyla dlouho uvedena v žádné aktualizaci. Po vypršení se její metrika nastaví na hodnotu která označuje nekonečno. Ale záznam zůstává stále ve směrovací tabulce. V případě že cesta je zneplatněna, ať už časovačem nebo aktualizací s nekonečnou metrikou od prvku, od kterého jsme informaci o síti získali, může být aktivován časovač typu holddown, během kterého nejsou přijímány žádné aktualizace týkajícího se tohoto cíle od jiných zdrojů než brány (prvku, který nám informace o této sítí zasílá.

Ještě se používá časovač expirace (flush timer), kdy je cesta úplně odstraněna ze směrovací tabulky.

Při použití výše uvedených technik se stabilita směrovacího protokolu značně zvyšuje, ale i tak byli zdokumentovány případy, kdy tento problém může nastat.

2.2.1 RIP

RIP, routing information protocol, se používá jako IGP. Komunikuje pomocí UDP paketů na portu 520 Jako metriku využívá počet skoků, počet uzlu, které jsou mezi ním a cílovou sítí. Maximální hodnota, kterou může metrika nabít je 15, 16 je již považováno za nekonečno. Přímo připojené sítě mají metriku 0. Aktualizace si mezi sebou uzly vyměňují periodicky každých 30 sekund a pokud síť není uveden v aktualizacích více jak 180 vteřin, 6 krát aktualizační čas, je zneplatněna popřípadě vyhozena ze směrovací tabulky. Používá většinu výše uvedených způsobů k zabránění počítání do nekonečna.

RIP existuje ve dvou verzích, verze 1 a 2, které se mezi sebou liší v několika bodech:

- RIPv1 nepodporuje VLSM, RIPv2 ano
- RIPv1 nepodporuje autentizaci, RIPv2 ano
- RIPv1 posílá aktualizace na adresu 255.255.255.255, RIPv2 na adresu 224.0.0.9
- RIPv1 nepodporuje značení záznamů

Podpora VLSM, variable length subnetmask, tedy masek s různou délkou prefixu, znamená, že v rámci 1 třídní sítě mohou existovat prefixy různé délky. Podpora VLSM je v RIPv2 docílena tím, že v aktualizacích jsou uvedeny i prefixy jednotlivých sítí.

Tabulka 2.3: Paket RIPu verze 1

1 octet	1 octet	2 octet	2 octet	2 octet	4 octet	4 octet	4 octet	4 octet
command	version	–	AFI	–	IP address	–	–	metric

Tabulka 2.4: Paket RIPu verze 2

1 octet	1 octet	2 octet	2 octet	2 octet	4 octet	4 octet	4 octet	4 octet
command	version	–	AFI	tag	IP address	subnet mask	next hop	metric

Formát paketů

V tabulkách 2.3 a 2.4 je vidět struktura paketů obou verzí RIPu.

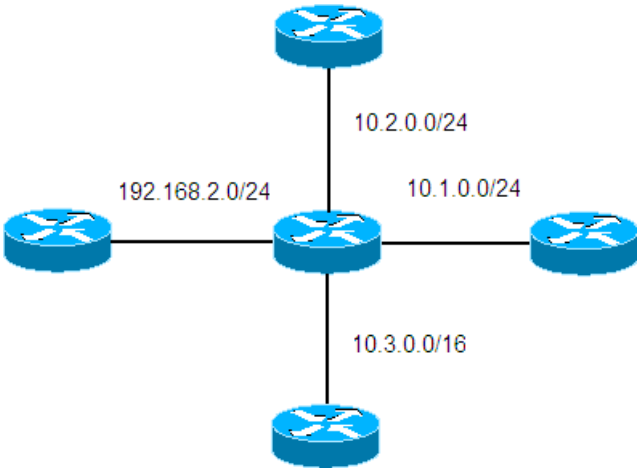
Vysvětlení jednotlivých polí v paketů:

- Command - příkaz - určuje typ paketu, zda se jedná o dotaz nebo odpověď
- Version - verze - určuje verzi procesu RIPu, který daný paket poslal
- AFI - rodina adres - RIP ma podporu i jak pro Token Ring sítě tak pro Ethernet sítě
- tag - pouze RIPv2, slouží pro rozlišení sítí naučených skrz RIP nebo externích zdrojů
- IP address - adresa sítě
- subnet mask - maska podsítě - pouze RIPv2, toto pole zajišťuje podporu VLSM
- next hop - adresa dalšího skoku směrem k dané sítí - pouze RIPv2
- metric - metrika - cena cesty k síti

Kromě prvních 3 buněk, jsou následující buňky vlastně záznamem směrovací tabulky. Tato část, od AFI pole včetně, se může až 25 krát opakovat. Jeden paket tak může nést informaci o více sítích. Hodnota AFI pro ethernetové sítě je 0x2.

Pakety jsou v podstatě dvou typu, dotaz, hodnota v poli command je 1, a odpověď, hodnota 2. Pakety typu dotaz jsou používány pro vyžádání celé směrovací tabulky, popřípadě konkretní sítě. Vysílá se většinou pokud se některé ze síťových rozhraní prvku stalo aktivní. Paket typu odpověď se posílá na vyžádání, pomocí paketu dotaz, ale jeho hlavní použití spočívá v tom, že je pravidelně zasílán sousedům s informacemi o sítích, které uzel má ve směrovací tabulce.

Pokud je aktivována autentizace, místo prvního záznamu sítě jsou uvedeny autentizační informace. AFI hodnota takového záznamu je 0xFFFF.



Obrázek 2.2: Příklad topologie

Způsob vytváření odpovědi

RIP implementuje několik technik, které zabraňují počítání do nekonečna. Ale jejich aktivaci komplikuje rozesílání aktualizací. Pokud by nebyly tyto techniky aktivovány, mohly by se ze všech rozhraní, ke všem sousedům posílat stejné aktualizace. Pokud jsou ale aktivní je nutné vytvářet aktualizace pro každé rozhraní zvlášť s ohledem na aktivované techniky.

Dalším problémem je semi-podpora VLSM v RIPv1, alespoň na směrovačích Cisco. Ta způsobuje, že je možné odeslat informaci o sítí na rozhraní ležící ve stejné třídní sítí a má stejný prefix jako propagovaná síť.

Průběh rozhodování je zachycen níže.

- Síť podléhá rozdělenému horizontu
  - tuto síť nepřidávej
  - přidej tuto síť, ale změn metriku na nekonečnou (pokud je zapnuté otrávení cesty)
- Síť nepodléhá rozdělenému horizontu
  - Síť bude propagována do sítě spadající do různé třídní sítě
    - převeď síť na třídní (u RIPv2 se muže, ale nemusí převádět)
    - přidej síť do aktualizace
  - Síť bude propagována do sítě spadající do stejné třídní sítě
    - Síť má stejný prefix, jako síť do které se bude posílat aktualizace
      - přidej síť do aktualizace
    - Síť má různý prefix než síť do které se bude posílat aktualizace
      - RIPv1 - nepřidávej síť do aktualizace
      - RIPv2 - přidej síť do aktualizace

Posledním krokem je zvýšení metriky všech záznamů uvedených v odpovědi.

Příklad

Jednotlivá pravidla si blíž vysvětlíme na obrázku 2.2 s příkladem topologie. Směrovač ve středu chce zaslat pravidelnou aktualizace do všech připojených sítí. Rozebereme vytvoření aktualizace pro horní směrovač.

Horní směrovač leží na sítí 10.2.0.0/24.

- 10.1.0.0/24 - Bude v aktualizaci, je ve stejné třídní síti, ale má stejný prefix jako síť na kterou je zasílána aktualizace
- 10.2.0.0/24 - Nebude v aktualizaci nebo bude, ale s nekonečnou metrikou, protože podléhá rozdělenému horizontu
- 10.3.0.0/16 - Nebude v aktualizaci, sice je ve stejné třídní síti, ale má rozdílný prefix
- 192.168.2.128/27 - Bude v aktualizaci, ale bude převedena na síť 192.168.2.0/24

U takto vytvořené odpovědi se už pouze zvýší metrika u všech záznamů a odpověď se odešle.

Způsob kontroly autentizace paketů

Po aktivací autentizace na rozhraní směrovač přijímá pouze takové směrovací informace, které se prokáží platným heslem nebo hashem.

Autentizace není podporována RIPem verze 1 což může způsobit nestabilitu v sítí. RIPv1 první záznam, ten který obsahuje autentizační údaje, ignoruje a zpracovává další záznamy.

Pokud je aktivní RIP verze 2 bez aktivní autentizace na rozhraní a přijde paket s autentizačním záznamem, je paket okamžitě zahozen. Ostatní pakety, verze 1 a verze 2 bez autentizace, jsou přijaty

Při aktivované autentizaci jsou veškeré neautentizované pakety zahozeny u ostatních je kontrolována autentizační hlavička dle nastavení. Zda jsou hesla nebo vypočítané hashe totožné.

Způsob zpracování odpovědi

Paket, který projde autentizační kontrolou, je dále zpracováván. Jednotlivé záznamy se porovnávají s aktuálním stavem databáze a pokud jsou nutné změny je databáze aktualizována. Záznamy jsou zpracovávány postupně, jak jsou uvedeny v paketu. Jako další skok je použita zdrojová adresa v ip hlavičce paketu.

Kontrola s databází probíhá následujícím způsobem:

- Záznam je již ve směrovací tabulce
  - Stejná brána
    - V paketu je jiná metrika
      - změn metriku ve směrovací tabulce, aby odpovídala metrice v odpovědi
      - pošli aktualizaci sousedům
    - resetuj časovač zneplatnění
    - přejdi na další záznam

  - Nestejná brána
    * Metrika v paketu je lepší
      · změn informace ve směrovací tabulce
      · resetuj časovač zneplatnění
      · pošli aktualizaci sousedům
    * přejdi na další záznam

- Záznam není ve směrovací tabulce
  - Metrika je nekonečná
    * přejdi na další záznam
  - Metrika není nekonečná
    * přidej novou síť do směrovací tabulky
    * aktivuj čítač zneplatnění
    * přejdi na nový záznam

Způsob zpracování dotazu

Pokud dotaz obsahuje pouze 1 záznam s hodnotou v poli AFI 0 a s nekonečnou metrikou, je zaslána zpět celá směrovací tabulka. Tento způsob dotazu je v praxi nejpoužívanější.

V případě, že záznamů je víc, zpracovávání probíhá tím způsobem, že se vytvoří odpověď, která obsahuje záznamy z dotazu. Pokud záznam z dotazu je ve směrovací tabulce, tak je v odpovědi uveden s metrikou ze směrovací tabulky. Pokud ve směrovací tabulce není, je u něho uvedená nekonečná metrika.

2.3 Protokoly stavu linky

Koncept protokolů stavu linky spočívá v tom, že si všechny prvky v síti vytvoří stejný graf sítě, z kterého si vypočítají počítají cestu ke všem ostatním prvkům/sítím. Rozdíl mezi protokoly stavu linky a distance vektor protokoly, spočívá v tom, že distance vektor protokoly si vyměňují směrovací tabulky. Protokoly stavu linky si vyměňují pouze informace potřebné k vytvoření onoho grafu sítě.

Výhoda těchto protokolů je, že jsou použitelné i do větších sítích, na rozdíl od distance vektor protokolů, kterým trvá až příliš dlouho než celá síť zkonverguje, ustálí se do jednotného stavu. U větších sítích může být i velkou nevýhodou omezení maximálního počtu skoků u distance vektor protokolů, kdy síť bude větší než tento maximální počet skoků.

Obecný protokol stavu linky

Obecný protokol stavu linky funguje na následujícím principu:

- Prvek si nejprve zjistí pomocí nějakého protokolu, které prvky leží na přímo připojených sítích.
- Prvek rozesílá informace o přímo přípojených prvcích do celé sítě.
- Prvek přijímá informace, které posílají ostatní prvky.

- Až celá síť zkonverguje, všechny uzly mají stejný graf, vypočítají si prvky nejkratší cestu ke všem prvkům pomocí Dijkstrova algoritmu. Výsledky si umístí do směrovací tabulky

V případě nějaké změny v sítí, prvek, který tuto změnu zaznamená, si změní svůj graf sítě a rozešle informaci o změně do celé sítě. Prvky si poté přepočítají svoji směrovací tabulku.

2.3.1 OSPF

Open shortest path first protokol se používá jako IGP. Protokol komunikuje na unicastových IP adresách a multicastových adresách 224.0.0.5 a 224.0.0.6, nevyužívá však služeb TCP ani UDP, ale přímo IP, skrz IP protokol 89. Díky tomu si musí, ale zajistit zpracování chyb a přeposílání nedoručených zpráv.

Obecný protokol stavu linky rozšiřuje o oblasti, síť poté není plochá, tzn. všechny uzly vidí všechny ostatní, ale je hierarchicky strukturovaná. Graf sítě se poté vytváří pro každou oblast zvlášť. Na hranicích mezi oblastmi jsou takzvané hraniční prvky, které zajišťují výměnu informací mezi oblastmi. Existuje několik speciálních typu oblastí:

- Páteřní oblast - všechny ostatní oblasti musí být připojeny k této oblasti. Zajišťuje výměnu směrovacích informacích mezi ostatními oblastmi.
- Pahýlní oblast - Stub oblast - nedostává směrovací informace o sítích, které se ospf proces dozvěděl od ostatních směrovacích protokolů, ale směrovací informace z ostatních ospf oblastí se do této oblasti dostanou.
- Uplný pahýl - Totally stubby oblast - Pouze implicitní cesta, směr kudy se budou posílat všechny pakety, pro které není cílová síť ve směrovací tabulce, se může propagovat do této oblasti
- Ne tak pahýlní oblast - Not-so-stubby oblast - Skrz oblast se mohou exportovat do páteřní oblasti externí cesty, ale ne z páteřní sítě do ní.

Jako metrika se používá součet cen linek mezi zdrojem a cílem. Cena linky je rovna podílu 100Mbps děleno rychlost linky.

2.4 Path vector protokoly

Path vector protokoly, neboli protokoly vektoru cesty, se používají zejména jako EGP protokoly. Princip má některé vlastnosti společné s distance vektor protokoly. V autonomním systému existuje prvek, mluvčí, který vytváří a inzeruje směrovací tabulku ostatním prvkům. Rozdíl je v tom, že pouze mluvčí mají právo mezi sebou komunikovat. U tohoto typu protokolů se nepoužívá metrika, namísto ní je použit vektor cesty, vektor autonomních systému, který popisuje cestu k dané koncové síti. Preferovanou cestou do sítě se stává ta, na níž se musí projít nejmenším množství autonomních systémů.

2.4.1 BGP

BGP, border gateway protokol, se používá jako směrovací protokol v internetu, mezi ISP. Komunikuje na unicastových adresách na portu 179 a využívá služeb TCP. U tohoto protokolu je výběr cesty k síti, řízen na bázi politik, spíše než na bázi metriky, nejkratší cesty.

Vzhledem k tomu, že se dané směrovací politiky aplikují v každém autonomním systému, může být výsledná cesta poměrně vzdálená cestě nejkratší.

BGP je poměrně robustní protokol a ne každý prvek ho bude schopný provozovat, i když také záleží na velikostí směrovací tabulky. Pro představu, směrovací tabulka internetu má přes 245,000 záznamů. U velkých provozovatelů internetu toto číslo může stoupnout i o 50%, kvůli směrování ve vnitřní sítí a mezi zákazníky.

2.5 Konfigurace na zařízeních Cisco

Konfigurace na zařízeních Cisco se provádí přes konzoly, dá se použít í webové rozhraní, ale konfigurace přes příkazový řádek je stále nejpoužívanější. Na počítači, z kterého se bude provádět konfigurace síťového prvku musí běžet program, který umí emulovat VT100 terminál. Například HyperTerminal nebo PuTty. Program musí nastavit parametry sériového portu, skrz který se bude komunikovat se zařízením na následující hodnoty:

- Rychlost v baudech - 9600
- Počet datových bitů - 8
- Parita - žádná
- Počet stop bitů - 1
- Kontrola toku - žádná

Propojení prvku se poté provede kabelem s koncovkou RJ-45 na straně prvku a DB-9 na straně PC.

Uživatel se poté octne v uživatelském modu, indikovaným znakem '>' za názvem prvku. V tomto modu má uživatel k dispozici pouze omezenou sadu příkazů k základní diagnostice směrovače.

```
Router>
```

Pro vstup do privilegovaného režimu, je nutné zadat příkaz `enable`. Po zadání může být uživatel vyzván k zadaní hesla. Oproti uživatelskému módu je k dispozici detailnější diagnostika směrovače a možnost přejít do konfiguračního režimu, odkud se dá měnit konfigurace prvku. Privilegovaný mód má indikační znak '#'. Pro opuštění privilegovaného režimu můžeme použít příkaz `disable`

```
Router> enable
Password:
Router#
Router# disable
Router>
```

Do konfiguračního režimu, přejdeme příkazem `configure terminal`. Přítomnost v konfiguračním režimu lze tím, že mezi názvem prvku a znakem '#' je přítomna sekvence znaku `(config)`. Z globálního konfiguračního režimu se poté přechází do jednotlivých specifických módů, např do konfigurace směrovacích protokolů, rozhraní atd. Pro opuštění konfiguračního režimu slouží 2 příkazy `exit` a `end`. `Exit` vrátí uživatele o jednu úroveň zpět, např. ze konfigurace směrovacího protokolu zpět do globálního konfiguračního režimu nebo z globálního konfiguračního režimu zpět do privilegovaného. Příkaz `end` vrátí uživatele okamžitě zpět do režimu privilegovaného

```
Router# configure terminal
Router(config)#
Router(config)# exit
Router#
```

Konfigurační možnosti Cisco zařízení jsou velice rozsáhlé a podrobný popis všech by byl až příliš rozsáhlý. Následující část se proto bude věnovat pouze konfiguraci směrovacího protokolu RIP a několika možnostem, které jsou použitelné všemi směrovacími protokoly nebo mají významný vliv na směrování.

2.5.1 Konfigurace bez závislosti na protokolu

Statické směrování

Cisco zařízení nabízejí podporu pro manuální specifikování směrovací tabulky. Konfigurace statických cest se provádí příkazem následujícím způsobem¨

```
ip route network net_mask { next_hop_address | interface } [distance] [tag] [permament]
```

Běžně se však poslední 2 volitelné parametry nepoužívají. Parametr distance označuje administrativní vzdálenost. Ta jako moc je daný zdroj směrovacích informací preferován. Čím nižší vzdálenost tím je zdroj kvalitnější. Každý směrovací protokol má přiřazenou administrativní vzdálenost, kterou je však možné měnit. Implicitní nastavení administrativních vzdáleností některých protokolů je v tabulce 2.5. Maximální hodnota je 255.

Tabulka 2.5: Administrativní vzdálenosti protokolů

Protokol	Implicitní adm. vzdálenost
Připojené sítě	0
Statické cesty	1
BGP	20
OSPF	110
RIP	120

Příklad:

```
Router(config)# ip route 192.168.5.0 255.255.255.0 10.3.4.2 150
```

Možnost specifikovat administrativní vzdálenost (distance) se využívá hlavně při použití záložních linek, kdy se vytvoří statická cesta do určité sítě s vyšší vzdáleností než je používaný dynamický směrovací protokol. Pokud se důsledkem nedostupnosti hlavní linky, smaže cesta do sítě ze směrovací tabulky, je použita ta s vyšší vzdálenosti skrz záložní linku.

Existuje speciální síť `0.0.0.0` s prefixem 0, na kterou, pokud je uvedena ve směrovací tabulce, se odesílají veškeré pakety jejíž cílová síť není ve směrovací tabulce.

Klíčenky

Klíčenky jsou struktury, které jsou využívány na uskladňování hesel, používaných při autentizaci směrovacích informací. Definice klíčenky se provádí v globálním konfiguračním režimu příkazem:

```
Router(config)# key chain keychain_name
Router(config-keychain)#
```

Po definici klíčenky je nutné vytvořit jednotlivé klíče pomocí identifikátoru, čísla. Každý klíč ma vlastní konfigurační režim.

```
Router(config-keychain)# key id
Router(config-keychain-key)#
```

Definováním klíče prvek přešel do konfiguračního režimu klíče, kde je možné definovat jeho vlastnosti. Hodnotu pomocí příkazu `key-string string` a časy platnosti pomocí dvou příkazu `accept-lifetime` a `send-lifetime`.

Ukazka konfigurace:

```
Router(config)# key chain RIPhesla
Router(config-keychain)# key 1
Router(config-keychain-key)# key-string rip
Router(config-keychain-key)# accept-lifetime 12:00:00 Jan 25 2007 infinite
```

2.5.2 RIP

Konfigurace RIPu se provádí v jednom z módů konfiguračního režimu Cisco zařízení. K přechodu do tohoto módu je nutné zadat příkaz `router rip`.

Spuštění procesu

```
Router(config)# router rip
Router(config-router)#
```

Pro spuštění RIP procesu, je nutné zadat sítě, které budou zahrnuty v aktualizacích a zároveň do kterých se budou vysílat aktualizace. Síť se přidává pomocí příkazu `network x.x.x.x`, kde `x.x.x.x` je adresa sítě. Pokud je nutné odebrat některou ze sítí, slouží k tomu z negovaný příkaz `no network x.x.x.x`. Přidání všech okolních sítí z obrázku 2.2 do procesu RIPu by vypadalo následovně:

```
Router(config-router)# network 10.1.0.0
Router(config-router)# network 10.2.0.0
Router(config-router)# network 10.3.0.0
Router(config-router)# network 192.168.2.0
```

Nastavení verze

Po přidání sítí už RIP proces posílá aktualizace verze 1 do zadaných sítí a přijímá aktualizace ve verzi 1 i 2. Pro změnu globální verze je možné zadat příkaz `version číslo_verze`. Změna verze ale mění původní nastavení a proces přijímá pouze zprávy zadané verze. K základnímu nastavení je možné se vrátit zadaním příkazu `no version`

```
Router(config-router)# version 2
Router(config-router)# no version
```

Verzi je možné měnit i na jednotlivých rozhraních, kde je k dispozici detailnější konfigurace. Je možné specificky nastavit, které verze se mají přijímat i odesílat. Nejprve je však nutné přejít do konfiguračního režimu rozhraní. To se provede následujícím způsobem:

```
Router(config-router)# exit
Router(config)# interface FastEthernet 0/0
Router(config-if)#
```

Příkaz `interface název_rozhraní číslo_slotu/číslo_portu` zpřístupnil konfiguraci 0. rozhraní FastEthernetu a kartě s pořadovým číslem 0. Nyní lze zadat příkazy pro změnu vysílaných a přijímaných verzí. K tomu slouží příkazy `ip rip send version číslo_verze` a `ip rip receive version číslo_verze`, s tím že mohou být uvedeny obě verze oddělené mezerou. Obrácené verze příkazů vrací zpět implicitní nastavení. Příklad demonstruje způsob jakým by se zapsalo implicitní nastavení.

```
Router(config-if)# ip rip send version 1
Router(config-if)# ip rip receive version 1 2
```

Rozdělený horizont

Další poměrně důležitý příkaz, který se konfiguruje v režimu rozhraní je `ip split-horizont`, který aktivuje rozdělený horizont, naopak příkaz `no ip split-horizont` ho deaktivuje. Zda je rozdělený horizont implicitně aktivní/neaktivní je dáno typem zapouzdření na rozhraní, pro sítě typu Ethernet je aktivní.

```
Router(config-if)# no ip split-horizont
```

Autentizace

Autentizace zajišťuje, že v rámci sítě bude prvek dostávat směrovací informace pouze od prvků se kterými sdílí klíč. Zabraňuje se tak případným neautorizovaným změnám do směrovacích tabulek prvků, které by mohly být využity při nějakém typu útoku.

Aby bylo možné nakonfigurovat autentizaci pro RIP je nutné nejprve definovat klíčenku a alespoň jeden klíč. Způsob vytvoření byl již uveden. Po vytvoření je nutné klíčenku propojit s RIPem a rozhraním pomoci příkazu v konfiguračním módu rozhraní `ip rip authentication key-chain keychain_name`.

RIP také nabízí podporu více typů autentizaci. Heslo je buď v paketech zasílané v textové podobě nebo je namísto něho použit hash, vytvořený ze zprávy a hesla, z důvodu větší bezpečnosti. Způsob zabezpečení se konfiguruje příkazem `ip rip authentication mode {text|mode}`. Pokud není mód autentizace specifikován, uvažuje se implicitně textové podoba hesla.

```
Router(config-if)# ip rip authentication key-chain RIPkeys
Router(config-if)# ip rip authentication mode md5
```

Podpora pro autentizaci je pouze u paketu RIPv2. Proto pokud přijde směrovací paket verze 1 na rozhraní, kde je zapnutá autentizace, je automaticky zahozen.

Pasivní rozhraní

Je nutné ale zmínit ještě několik příkazů, které se zadávají v konfiguračním režimu RIPu. Pokud na připojené sítí není žádný další aktivní prvek, který by potřeboval aktualizace směrovacích informací, lze vypnout zasílání zpráv na rozhraní pomocí příkazu `passive-interface název_rozhraní číslo_slotu/číslo_portu`. Zprávy ale budou nadále přijímány. Pokud je místo názvu rozhraní zadáno klíčové slovo `default` jsou všechna rozhraní nastavena jako

pasivní. Jednotlivá rozhraní, která chceme mít aktivní poté aktivujeme pomocí obráceného příkazu `no passive-interface název_rozhraní číslo_slotu/číslo_portu`.

```
Router(config-router)# passive-interface FastEthernet 0/1
```

Automatická sumarizace

V kapitole o RIPu bylo zmíněno, že RIPu převádí adresy sítí na třídní, pokud se překračuje hranice mezi třídními sítěmi, a také že u RIPv2 lze tuto funkci vypnout. K tomu slouží funkce `no auto-summary`. RIPv2 poté propaguje sítě s nezměněnými prefixy i přes třídní hranice. Na verzi 1 tato funkce nemá vliv, ta sumarizuje vždy.

```
Router(config-router)# no auto-summary
```

Časovače

Poslední důležitá funkce se týká manipulace s časovači. V původním nastavení mají časovače tyto hodnoty:

- čas aktualizací - 30 sekund
- čas zneplatnění - 180 sekund
- čas potlačení - 180 sekund
- čas vyplavení - 240 sekund

Příkaz pro změnu těchto hodnot je `timers basic a z p v` kde a, z, p, v jsou nové hodnoty jednotlivých časů. Příkazem `no timers basic` je obnoveno původní nastavení. Podle doporučení by čas zneplatnění a potlačení měl být alespoň 3x větší než čas mezi aktualizacemi a čas vyplavení by měl alespoň stejně dlouhý jako součet časů aktualizaci a potlačení.

```
Router(config-router)# timers basic 20 120 120 160
```

Shrnutí

V kapitole byly představeny jednotlivé rodiny směrovacích protokolů. Důkladněji byl představen směrovací protokol RIP, protože je předmětem implementace, a způsob jeho konfigurace na Cisco prvcích.

Kapitola 3

Simulátor sítě ns-2

Úvod

Implementovaný směrovací protokol by měl být rozšířením síťového simulátoru. Je nutné se tedy seznámit s jeho strukturou, jakým způsobem je řešeno směrování, odesílání paketů atd.

Představení simulátoru

NS-2, network simulator, je diskrétní simulátor zaměřený na výzkum sítí. NS by měl nabízet značnou podporu pro simulaci TCP, UDP, směrování a multicastové protokoly a to jak pro klasické drátové i bezdrátové sítě. Jeho nedílnou součástí je NAM, network animator, který slouží ke zobrazení výsledků simulace v grafickém režimu. Podporuje změny rozmístění prvků, animování jednotlivých paketů atd. Díky spojení těchto dvou nástrojů, se můžeme občas setkat s názvem NSNAM.

Simulátor je napsán ve dvou jazycích, OTcl, což je Tcl rozšířené o objekty, a C++. C++ slouží jako výpočetní část a OTcl jako uživatelské rozhraní. Vzhledem k tomu, že oba jazyky jsou objektové existují zde 2 hierarchie objektů, které v některých místech korespondují mezi sebou.

Scénář simulace se vytváří ve formě OTcl skriptu. Je možné ovlivňovat dobu simulace, chovaní jednotlivých objektů, stav linek atd.

3.1 Základní prvky

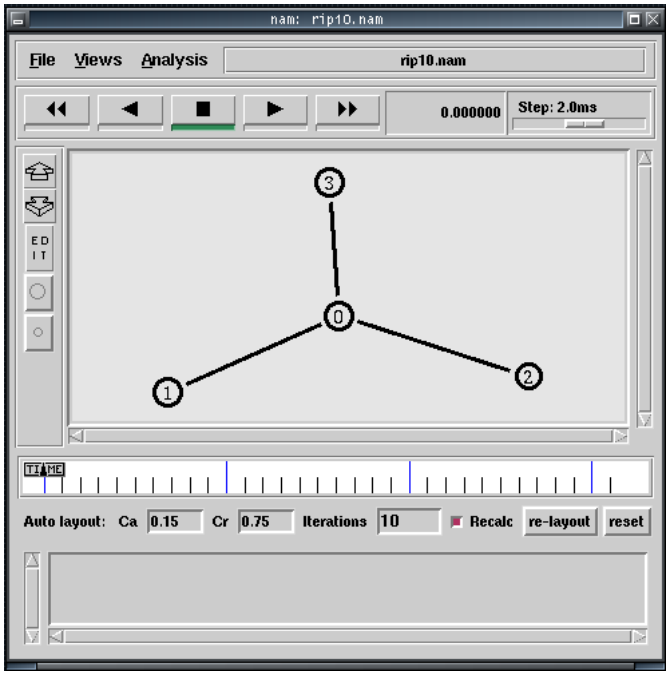
NS poskytuje základní stavební kameny pro vytváření topologii a to prvky, linky, které je spojují a procesy, které na nich běží.

Uzly

Uzel je základní stavební prvek topologie, může reprezentovat každý síťový prvek. Při vytvoření je mu přiděleno jedinečné ID a adresa. Na uzly jsou navázány další komponenty, klasifikátory a agenti.

Linky

Zajišťují spojení mezi jednotlivými uzly. Skládá se z komponent poskládaných do řetězu, kde každá komponenta má jinou funkci, např. fronta, zpoždění atd.



Obrázek 3.1: Network animator

Agenti

Agent reprezentuje procesy, které běží na jednotlivých uzlech. Možná ještě obecněji se dá říct, že se jedná o místo kde vznikají, jsou zpracovávány a zanikají pakety. Může zastupovat funkce různých aplikací, směrovacích i transportních protokolů

Pakety

Zprávy, které si mezi sebou vyměňují jednotlivý agenti. Specifikum ns-2 je, že paket obsahuje hlavičky všech nadefinovaných protokolů. Agent, kterému takový paket přijde muže si rovnou přečíst svoji hlavičku, a na jejím základě zkoumat zda je to paket, který přijímá. Agent tedy nemusí zkoumat bit po bitu daný paket.

Nevýhodou tohoto přístupu je, že pakety jsou velké a při velkých simulacích způsobují velké vytížení zdrojů. Hlavičky se dají, ale selektivně vypnout a ponechat si pouze ty, které pro konkretní simulaci potřebujeme.

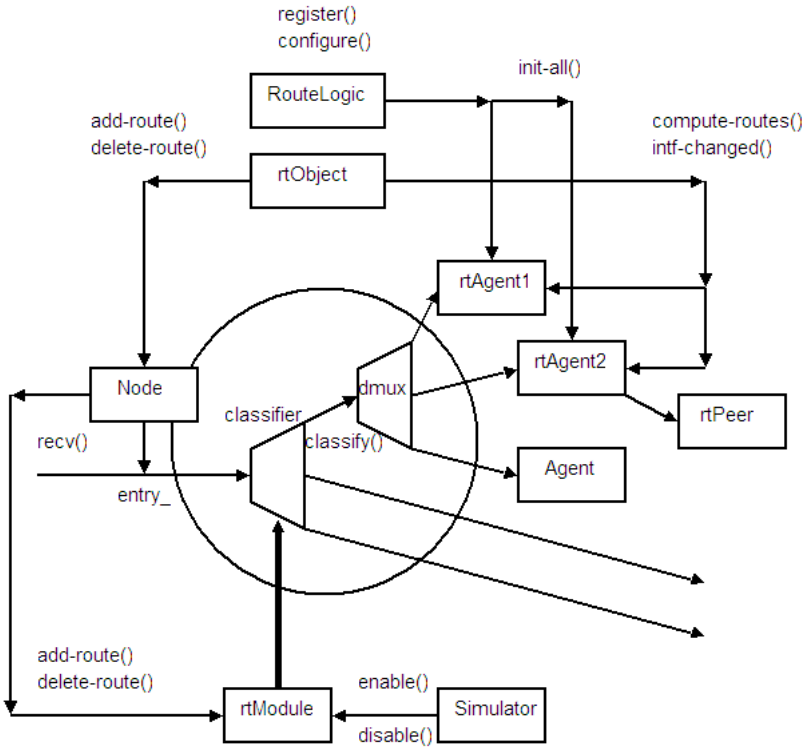
3.2 Směrovací struktura

Směrovací struktura v ns-2 se skládá z několika částí.

Klasifikátor

Klasifikuje pakety na základě nejrůznějších informaci a poté předává, na základě těchto informací, pakety dál. Lze si ho představit jako demultiplexor.

Uzel obsahuje většinou minimálně 2, jeden který klasifikuje na základě adresy a plní v podstatě funkci směrovací tabulky, a druhý, který klasifikuje podle cílového portu, pokud



Obrázek 3.2: Směrovací struktura ns-2

je paket určen pro tento uzel. Krom těchto 2 může být například přítomen klasifikátor multicastových paketů. Klasifikátory poté tvoří stromovou strukuru.

rtModule - směrovací modul

Směrovací modul řídí klasifikátor, a to jak jeho operace, přidávaní a odebírání cest, tak jeho vytváření a instalaci do uzlu. Uzel může obsahovat více modulů, stejně jako klasifikátoru. Ty potom vytvářejí řetěz, po kterém se propagují informace ke všem směrovacím modulům.

Směrovací protokoly

Jedná se vlastně o agenty, kteří si vyměňují zprávy s ostatními instancemi protokolu.

rtObject

Dává dohromady databáze směrovacích protokolů a vytváří z nich jedinou směrovací tabulku. Upozorní poté uzel, že si má změnit cestu k určitému cíli. Uzel poté upozornění pošle poté zprávu na řetěz směrovacích modulů a ty změní informace v klasifikátoru.

RouteLogic

Zajišťuje přidávání směrovacích protokolů k jednotlivým uzlům dle specifikací uživatele.

3.3 Dostupné unicastové směrovací protokoly

Static

Využívá Dijkstrův algoritmus, cesty k jednotlivým uzlům se vypočítají při spuštění simulace a každý uzel si je přidá do klasifikátoru.

Session

Pracuje na podobném principu jako Statické směrování, ale s tím rozdílem, že při každé změně v topologii si uzly změní cesty k jednotlivým uzlům. Šíření této informace není nijak zpracováno, předpokládá se, že všechny uzly zaznamenají jakoukoliv změnu v topologii.

DV

Forma distance vektor protokolu. Uzly si pravidelně zasílají svoje směrovací tabulky. Změny se v topologií nešíří okamžitě, jednotlivé uzly si musí tyto změny přeposlat. Protokol, ale zjednodušuje posílání zpráv mezi jednotlivými subjekty. Existuje 'globální' databáze zpráv, a uzly si mezi sebou pouze vyměňují informace o tom, kterou zprávu si máji vyzvednout.

3.4 Příklad skriptu

V příkladu je uvedena jednoduchá kruhová topologie, se zapnutým distance vektor směrováním.

První řádek vytvoří novou instanci simulátoru. Ve druhém a třetím je určeno, které výstupy ze simulace se budou ukládat a kam.

```
set ns [new Simulator]          #vytvoreni simulatoru
set nf [open DV.nam w]          #otevreni trace souboru pro NAM
$ns namtrace-all $nf
```

Jak již název procedury napovídá, spouští se na konci simulace a jejím úkolem je spustit NAM s nově získanými výsledky.

```
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam DV.nam &
        exit 0
}
```

Následuje vytvoření uzlů, linek mezi nimi a agentů. Pořadí parametrů při vytváření linek je rychlost, zpoždění a typ fronty. Null agent pouze zahazuje veškeré pakety, které mu přijdou.

```
#Vytvoreni uzlu
for {set i 0} {$i < 5} {incr i} {
        set n($i) [$ns node]
}
```

```
#Vytvoreni linek
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1Mb 10ms DropTail
$ns duplex-link $n(4) $n(0) 1Mb 10ms DropTail

#Vytvoreni UDP agentu
for {set i 0} {$i < 5} {incr i} {
        set udp($i) [new Agent/UDP]
        $ns attach-agent $n($i) $udp($i)
}

#Vytvoreni Null agentu
for {set i 0} {$i < 5} {incr i} {
        set null($i) [new Agent/Null]
        $ns attach-agent $n($i) $null($i)
}

#Pripojeni aplikace k agentovi
set cbr(13) [new Application/Traffic/CBR]
$cbr(13) set packetSize_ 500
$cbr(13) set interval_ 0.005
$cbr(13) attach-agent $udp(1)

#Propojeni dvou agentu
$ns connect $udp(1) $null(3)

#Aktivace distance vektor smerovani pro vsechny uzly
$ns rtproto DV
```

Poslední blok příkazů definuje chování simulace. Kdy se mají začít posílat data z jednotlivych aplikaci, kdy mají být linky neaktivní a kdy ma simulace skončit atd. Nakonec se celá simulace příkazem `run` instance simulátoru spustí.

```
#zacate behu aplikace
$ns at 0.5 "$cbr(13) start"
$ns at 4.5 "$cbr(13) stop"

#zmena stavu linky
$ns at 1.5 down $n(1) $n(2)
$ns at 2.5 up $n(1) $n(2)

#delka simulace
$ns at 5.0 "finish"

#spusteni simulace
$ns run
```

3.5 Nedostatky vůči IPv4

Ns příliš nekoresponduje s IPv4 prostředím. Následuje výčet nedostatků, které má ns-2 bez provedení jakýchkoliv změn:

- Pouze uzly mají adresy, rozhraní jsou neadresovatelná.
- Pakety vcházejí do uzly pouze 1 vstupem, rozhraní lze rozlišit pouze výstupu.
- Rozhraní na výstupu má formu pouhé linky.
- Směrování neprobíhá nad sítěmi, ale na uzly. Ve směrovací tabulce jsou uvedeny uzly a cesty k nim, namísto sítí. Toto také plyne z toho že pouze uzly mohou mít adresu

Shrnutí

V této kapitole byl představen simulátor sítě ns-2 a byli adresovány jeho nedostatky vůči IPv4 prostředí, které budou muset být před implementaci napraveny.

Kapitola 4

Návrh a implementace

Úvod

V této kapitole jsou uvedeny požadavky na aplikaci společně s návrhem možné implementace. Dále je tu popsán způsob implementace jednotlivých vlastností.

4.1 Koncept rozšíření

Návrh

Záměrem této práce je rozšíření stávajících možností ns-2 a ne úplná změna ns-2 a proto by měla být implementace koncipována tak, aby ji bylo možné selektivně aktivovat či deaktivovat. A v případě, že bude deaktivována nijak neovlivňovala běh simulace.

Implementace

Soubor: ns-lib.tcl, ns-route.tcl

Aby byla dodržena podmínka rozšíření, tedy že implementace nebude ovlivňovat původní strukturu ns-2, byla zavedena proměnná simulátoru `IPv4_`. Na základě jeho hodnoty se aktivuje nová IPv4 struktura nebo zůstane původní. Aktivace nové struktury se provádí voláním metody simulátoru `IPv4` s parametry `ON`. Metoda deaktivuje základní, Base, směrovací modul a aktivuje nový IPv4 modul. Pro zjištění zda je rozšíření aktivováno slouží metoda simuáltor `IPv4?`, která vrací hodnotu `ON`, pokud je rozšíření aktivní.

Příklad zadání do skriptu:

```
$ns IPv4 ON
```

Bylo nutné zasáhnout do stávajících kódů ns-2 a změnit chování směrovací logiky, RouteLogic, tak aby při spuštění IPv4 rozšíření vytvořila instance nových rtObjectů na všech uzlech.

4.2 Podpůrné prvky a struktury

Protože ns-2 nemá pro IPv4 prostředí dostatečnou podporu, bylo nutné implementovat nové prvky směrovací struktury, které již IPv4 podporují.

4.2.1 IPv4 adresy

Návrh

Uzly by měli být adresovatelné více adresami, ne pouze jednou jako tomu je v původní ns-2. Uzly by měli mít strukturu pro uchovávání adres rozhraní a také by měli přijímat pakety, které jsou určeny pro tyto adresy. Stejně tak by měli podporovat přijímání broadcastových zpráv.

Implementace

Soubor: ns-nodeIPv4.tcl

Adresy jednotlivých rozhraní jsou uloženy v asociativním poli, kde je jako klíč použít sousední uzel. Nevýhodou tohoto přístupu je možnost mít pouze jedno rozhraní k jakémukoliv uzlu, ale vzhledem k uložení linek v samotném simulátoru, pomocí id počátečního a koncového uzlu, neumožňuje ani samotná ns-2 podporu pro více rozhraní mezi uzly. Výhodou je snadná orientace v linkách a při vytváření skriptů.

Po zadáni adresy rozhraní je záznam převeden na tvar adresa/maska a uložen do asociativního pole. Navíc je přidána informace o sítí na které rozhraní leží, z důvodu že se tato informace často využívá. Odpadá tak nutnost ji znovu počítat při každém jejím použití.

4.2.2 Klasifikátor

Návrh

Klasifikátor s podporou IPv4 musí, pokud mu přijde paket, ze své směrovací tabulky vybrat síť, ve které se nachází uzel, pro který je paket určen. Klasifikátor musí udržovat cesty k jednotlivým sítím, ne uzlům jako to je v implicitním klasifikátoru v ns-2. Dále by měl vyhledávat ve své směrovací tabulce od nejspecifičtějších záznamu, záznamu s nejdelším prefixem.

Implicitní klasifikátor v ns-2 měl ještě jeden nedostatek, pokud nenašel cíl pro daný paket, volal proceduru `no-route`, která zastavila simulaci a vypsala na standardní výstup zprávu o tom, že daná cesta neexistuje. Toto chování ale není vhodné pro IPv4 sítě a je nutné ho odstranit. Zařízení v reálných sítích by v tomto případě odesilateli zaslalo zprávu o tom, že daná síť není k dispozici.

Implementace

Soubory: classifier-ipv4.{h, cc}, classifier-port.{h, cc}

Klasifikátor, který směruje na základě IP adres do sítí byl již implementován při vytváření modelu BGP pro tento simulátor. Autory tohoto modelu a i klasifikátoru jsou Rob Ballatyne a Tony Dongliang Feng. Klasifikátor však nepodporuje rozesílání paketu více cílům naráz, multicast a broadcast, ale tuto vlastnost nebude potřeba implementovat. Klasifikátor se ale choval podobně jako implicitní klasifikátor v případě, že přijal paket jehož cílovou adresu, nebo alespoň síť do které spadá, neměl ve směrovací tabulce. Tuto vlastnost bylo třeba změnit.

Pokud do klasifikátoru přijde paket je na něj volána metoda `classify` jejíž návratová hodnota určuje na který slot v klasifikátoru se daný paket odešle. Jakým způsobem se hodnota tohoto slotu získá je čistě na autorovi klasifikátoru.

Metoda `classify` je volána z metody `find`, která na základě slotu vrací objekt na který se má paket poslat dále. Pokud však metoda `classify` vrátila hodnotu, která značila, že žádný slot nebyl nalezen, byla volána právě ona zmiňovaná funkce `no-route`. Tato vlastnost však neodpovídala návrhu, proto byla změněna metoda `find` zděděná z rodičovské třídy `Classifier`. A to u obou používaných klasifikátorů, jak adres tak portů.

4.2.3 Směrovací modul

Návrh

Jak již bylo zmíněno, úlohou směrovacího modulu je řízení klasifikátoru. Měl by proto poskytovat rozhraní uzlu pro přidávání cest do jednotlivých sítích a zároveň řídit správu klasifikátoru.

Implementace

Soubory: rtmodule.{h,cc}

Vzheldem k tomu, že klasifikátor od Ballatyna a Fenga měl k sobě již vytvořený směrovací modul, nebyl implementován nový modul, pouze byl použitý, ten již naimplementovaný. V dalším textu však bude popsán způsob jeho intergrace.

Definice nového směrovacího modulu - rtmodule.h:

```
class IPv4RoutingModule : public RoutingModule {
public:
        IPv4RoutingModule() : RoutingModule() {}
        virtual const char* module_name() const {return "IPv4";}
        virtual int command (int argc, const char* const * argv);
protected:
        IPv4Classifier *classifier_;
};
```

Je zde definováno zařazení do hieararchie tříd a rozhraní s OTcl, pomocí funkce `command`, její podrobnější rozbor je dále v textu až později. Protože používáme jinou třídu klasifikátoru je nutné zde tuto skutečnost uvést. Vzhledem k tomu, že je tato třída implementována v obou jazycích ns-2, je nutné tyto dvě prostředí propojit. V následujícím bloku kódu v souboru rtmodule.cc definujeme, že pokud se v OTcl vytvoří objekt třídy `RtModule/IPv4` bude vytvořen odpovídající objekt třídy `IPv4RoutingModule`, pomocí volání funkce `create`.

Propojení OTcl/C++ - rtmodule.h:

```
static class IPv4RoutingModuleClass : public TclClass {
public:
        IPv4RoutingModuleClass() : TclClass("RtModule/IPv4") {}
        TclObject* create(int, const char*const*) {
                return ( new IPv4RoutingModule );
        }
} class_ipv4_module;
```

V témže souboru je ještě potřeba implementovat funkci `command`, nejsou nutné zvláštní funkce, takže ji můžeme implementovat podle vzoru `BaseRoutingModule`.

OTcl

Soubor: ns-rtModuleIPv4.tcl

Ostatní součásti ns-2 využívají několik metod směrovacího modulu skrz Otcl. Metodu `register`, která definuje, které klasifikátory se mají přidat do uzlu, pokud je daný modul aktivní. Uzly potřebují rozhraní pro přidávání a odebírání cest z klasifikátoru, toto rozhraní je zajištěno metodami `add-route` a `delete-route`. V těchto metodách je poté již voláno rozhraní klasifikátoru.

4.2.4 Nový rtObject

Návrh

Původní rtObject byl implementován čistě v OTcl a skládal z několika asociativních polí, kde jako klíč byli použity jednotlivé uzly. Toto řešení bylo možné vzhledem k tomu, že počet uzlů byl předem znám. Implementací IPv4 prostředí a směrování na bázi sítí se stalo toto řešení nepoužitelným. Musela by se udržovat nějaká centrální databáze všech použitých sítí, ke které by měli přístup všechny prvky.

Další podstatnou věcí bylo, že všechny směrovací protokoly museli mít jednotnou strukturu asociativních polí v OTcl, ze kterých si rtObject bral informace cestách k různým uzlům. Pokud si tedy směrovací protokol udržoval svoji směrovací tabulku v C++, musel mít nutně duplicitní v OTcl, kvůli rtObjectu.

Také chyběla možnost ovlivnit, ne který port bude směrovací protokol připojen. V původní implementaci si směrovací protokoly museli nejprve zjistit na kterém portu sousedského uzlu běží instance tohoto protokolu, protože bylo možné, že jednotliví agenti směrovacích protokolů byli na různých portech.

Nový rtObject by měl poskytnout podporu pro směrování na základě sítí, a také odstranit nutnost mít pevně specifikovanou směrovací tabulku v OTcl pro směrovací protokoly. Jako vhodný model se zdál být model typu dotaz/odpověd, kdy směrovací protokoly informují rtObject o změnách ve svých směrovacích tabulkách. On si na základě těchto upozornění mění vlastní směrovací tabulku a informuje uzel, aby si změnil informace v klasifikátoru. Pokud je nutné smazat cestu do některé sítě, měl by se zeptat ostatních protokolů zda nemají cestu do dané sítě. Měl by také umět připojit směrovací protokol na konkrétní port.

Z důvodu podobnosti s výpisy z Cisco zařízení by bylo vhodné přidat také možnost specifikovat identifikátor směrovacího protokolu, např. C pro připojené sítě, R pro RIP atd.

Implementace

Soubor: ns-rtObjectIPv4.tcl

Nový model rtObjectu je založen na komunikaci formou dotaz/odpověď mezi rtObjectem a směrovacími protokoly, kde odpovědi od směrovacích protokolů nemusí být vždy vyžádané. Směrovací protokoly pomocí odpovědi informují rtObject o změnách svých databází. Vzhledem k tomu, že rtObject nepracuje s pakety, ale pouze se směrovacími informacemi, nebyl důvod implementovat jeho část v C++, která je ve většině případu určena pro zpracovávání jednotlivých paketů.

Sítě jsou uloženy v asociativním poli kde jako klíč slouží síť/prefix. V dalších polích jsou uloženy poté informace o dalším skoku, preference protokolu a jeho metrika.

Metoda směrovacího protokolu, která se používá jako dotaz na cestu k sítí a kterou musí implementovat každý směrovací protokol, má tvar volaní `Protocol notify-about-route síť prefix`. Jako odpověď směrovacích protokolů rtObjectu se používá stejnojmenná metoda, ale v tomto případě se jedná o metodu rtObjectu. Její parametry jsou však rozšířené. Její volání má tvar `rtObject notify-about-route identifikátor síť prefix adresa_dalšího skoku preference metrika`. Pokud je v odpovědi hodnota metriku -1 je to signál pro smazaní cesty do určené sítě.

Další metodou, kterou musí implementovat směrovací protokoly je `Protocol clear-route síť prefix`, kterou žádá rtObject o smazaní určité cesty ze směrovací tabulky. Pokud je hodnota parametru `síť` rovna `all`, je to žádost o vymazání celé databáze. O smazaní již nemusí být rtObject již informován.

Metoda `intf-changed` zůstala v podstatě nezměněna, pouze v ní není explicitně voláno `compute-routes` všech protokolů. Nové protokoly by měli reagovat na tuto změnu, a ne na volání funkce `compute-routes`

Implementace musela obsahovat funkce, které by sloužili jako rozhraní pro ostatní komponenty simulátoru. Ve většině případů se shodují se starým rtObjectem, pouze metoda `add-proto` byla rozšířena o možnost připojit směrovací protokol na konkrétní port. Pro úplnou podporu připojení agenta na konkretní port v demultiplexoru portů bylo nutné ještě změnit metodu simulátoru `attach-agent`, definovanou v ns-lib.tcl. Metodě byl přidán volitelný parametr, který určuje, na který port má být agent připojen. Není-li jeho hodnota specifikována, agent se připojí na první možný port.

Pokud se smaže cesta do některé sítě rtObject pošle dotaz všem protokolům na novou cestu do této sítě. Je nutné si, ale uvědomit že toto nastává okamžitě po obdržení zprávy o změně, a že rtObject se dotazuje všech směrovacích protokolů, i toho který zprávu o smazaní cesty odeslal. Může totiž nastat situace, že směrovací protokol má tuto cestu ještě v databázi a při dotazu na tuto síť odpoví právě mazanou cestou. To vede k nejrůznějším problémům.

Přehled funkcí, které je nutné implementovat do nových směrovacích protokolů:

- `intf-changed`
- `notify-about-route`
- `clear-route`

4.2.5 Připojené sítě

Návrh

Každý uzel by měl znát sítě, které má k sobě přímo připojené. V původním ns-2 znal uzel svoje sousedské uzly. To bylo realizováno 'směrovacím' protokolem Direct.

Podobný princip je nutné zařídit i pro přímo připojené sítě a proto je nutné implementovat modifikaci směrovacího protokolu Direct. Tento protokol by měl umět nejenom přidat připojené sítě do klasifikátoru, ale také přidat adresy rozhraní, tak aby pakety, které jsou směrovány na tyto adresy byly předány klasifikátoru portů. Dále by měl přidat i broadcastové adresy jednotlivých sítí.

Implementace

Soubor: ns-rtProtoDirectIPv4.tcl

Správu připojených sítí zajišťuje nová verze protokolu Direct s názvem DirectIPv4. Instance tohoto 'směrovacího' protokolu vzniká společně se vznikem rtObjectu, aby bylo zajištěno, že všechny uzly budou znát alespoň připojené sítě. Při inicializaci také podle návrhu přídává broadcastové adresy a adresy rozhraní do klasifikátoru, aby uzel přijímal pakety jejíž cílová adresa je rovna adrese rozhraní.

Pracuje na jednoduchém principu, kde na základě stavu rozhraní určuje zda daná síť má být přidána, případně odebrána z rtObjectu. O adresách rozhraní předpokládáme že se nemění. Na `clear-route` nereaguje, připojené sítě jsou známy i při smazání směrovacích databází.

Protokol Direct je implementován pouze v OTcl, protože nevyužívá zasílání paketů. Implementovány jsou všechny funkce vyžadované pro interakci s rtObjectem.

4.2.6 Statické směrování

Návrh

Statické směrování je oproti RIPu jednoduší. Její instance si nevyměňují pakety a pouze uživatel může změnit jeho cesty k sítím. Přidání statické cesty k sítí by mělo být co nejpodobnější příkazu Cisco IOS `ip route x.x.x.x m.m.m.m n.n.n.n p`, kde x značí adresu sítě, m její masku, n další skok ve směru k sítí a p volitelně preferenci. Protokol by mě pružně reagovat na změny linek vedoucí k uzlu a nepropagovat cesty k sítím skrz linky, které jsou nedostupné.

Implementace

Soubor: ns-rtProtoStaticIPv4.tcl

Podobně jako Direct ani instance statického směrování si mezi sebou nevyměňují zprávy o stavu směrovacích tabulek a proto je možné i statické směrování implementovat jako OTcl třídu. Musí implementovat, stejně jako Direct a další směrovací protokoly, rozhraní s rtObjectem, konkrétně funkce `notify-about-route`, `clear-route` a `intf-changed`.

Databáze cest se dá implementovat pomocí asociativních polí s vhodně zvoleným klíčem. Klíč v podobně síť/prefix není bohužel dostatečný, protože u statických cest je běžnou praxí mít 2 do stejné sítě, ale s různou preferencí. Pokud linka ve směru s cesty s nižší preferencí není funkční, zvolí se druhá s preferencí vyšší.

Klíč je tedy nutné zvolit jako kombinaci sítě prefixu a adresy dalšího skoku.

Protokol musí sledovat změny rozhraní, aby protokol oznamoval rtObjectu správné cesty skrz aktivní rozhraní, ne skrz ty, které jsou nefunkční. Na metodu `clear-route` protokol nereaguje, staticky přidané cesty, podobně jako přímo připojené se ze směrovací tabulky nemažou, pouze dynamický naučené cesty.

Přidání směrovacího protokolu do simulace

Na rozdíl od protokolu Direct, který je přidán vždy, statické směrování je možné přidat selektivně na určité uzly pomocí příkazu:

```
$ns rtproto StaticIPv4 $n(1) $n(2)
```

Pokud není definován seznam uzlů, je protokol přidán na všechny uzly.

Pojmenování je zvoleno tak aby se protokol odlišil od původního protokolu Static. Kvůli způsobu přidání protokolu, skrz volání funkce rtproto, jsou jeho instance vytvořeny až po spuštění simulace a i jeho konfigurace je možná až v simulačním čase.

4.3 Směrovací protokol RIP

Návrh

Model směrovacího protokolu by měl odpovídat funkčně RFC dokumentům [2], [4] a také chování protokolu na Cisco prvcích. Implementace nemusí odražet chování úplně přesně, z důvodu že se jedná o model protokolu, z toho důvodu je možné si dovolit některá zjednodušení.

Implementace

soubory: rtProtoRIP.h, cc, rtProtoRIP_packets.h

Implementace RIPu je napsána jak v OTcl tak v C++, z důvodu manipulace s pakety. Z toho plyne, podobně jako v případe směrovacího modulu, nutnost tyto 2 oddělené části propojit. Také bylo nutné nadefinovat novou hlavičku paketu a zpřístupnit nastavení procesu uživatelům.

4.3.1 Nový typ paketů

Návrh

Aby spolu mohly instance protokolu komunikovat je nutné nadefinovat formát zprávy, RIP hlavičku paketu. Z tabulek 2.3 a 2.4 je patrné, že většinu polí mají protokoly společné. Je tedy možné nadefinovat pouze 1 hlavičku, kterou budou využívat obě verze protokolu. Pole verze protokolu poskytuje dostatečné rozlišení jednotlivých verzí. Specifická pole verze 2, může verze 1 ignorovat.

Implementace

Obsah paketu je stejný jako obsah paketu RIPv1 obohacený o položku RIPv2, která určuje délku prefixu dané sítě. Oproti plné RIPv2 hlavičce chybí položka TAG, z důvodu, že v současné době není redistribuce z jiných protokolů implementována, žádné další nejsou zatím k dispozici. Také chybí pole dalšího skoku, implicitně se vždy využívá zdrojové adresy paketu. Změnou oproti tradiční struktuře RIP paketů, kde je první záznam využit pro autentizační údaje, je vytvoření samostatné proměnné, která tyto údaje obsahuje.

Struktura paketu - rtProtoRIP_packets.h:

```
int             command_;
int             version_;
const char *    authentication_;
RIP_route_list  routes_;
```

Proměná `routes_` zastupuje záznamy jednotlivých sítí v paketu. Každý záznam má následující strukturu, která odráží strukturu záznamu RIPu. Jediným rozdílem je ukazatel na další záznam.

```
struct RIP_route_list_entry {
        int                     afi;
        u_int32_t               ip;
        int                     prefix;
        int                     metric;
        RIP_route_list_entry*   next;
};
```

Je důležité si povšimnout níže uvedených řádků uvnitř struktury hdr_rtProtoRIP_pkt. K informacím v RIP hlavičce paketu je potřeba přistupovat a protože paket sebou nese všechny hlavičky, které jsou poskládány jedna za druhou v poli znaků, je nutné vědět kde přesně se RIP hlavička nachází. Právě přístup k ní je zajištěn níže uvedenou funkcí v rtProtoRIP_packets.h

```
static int      offset_;

inline static hdr_rtProtoRIP_pkt* access(const Packet* p) {
        return (hdr_rtProtoRIP_pkt*)p->access(offset_);
}
```

Pro přístup se také často definuje makro s parametrem:

```
#define HDR_RTPROTORIP_PKT(p) hdr_rtProtoRIP_pkt::access(p)
```

Hodnotu `offset_` je nutné mít zpřístupněnou skrz Tcl, aby byla inicializována simulátorem na hodnotu odpovídající posunutí RIP hlavičky v poli znaků.

Zpřístupnění RIP hlavičky - rtProtoRIP.cc:

```
static class rtProtoRIPHeaderClass : public PacketHeaderClass
{
public:
        rtProtoRIPHeaderClass() : PacketHeaderClass("PacketHeader/rtProtoRIP",
        sizeof(hdr_rtProtoRIP_pkt)) {
                bind_offset(&hdr_rtProtoRIP_pkt::offset_);
        }
} class_rtProtoRIP_hdr;
```

Nově definovaná hlavička se musí přidat do simulátoru. Všechny hlavičky, které simulátor zná jsou definované v souboru packet.h. Nejprve je nutné zadat novou hodnotu do výčtu `packet_t`. Dále pak je nutné přidat informaci o názvu paketu do třídy `p_info`. Tento název bude použit při vytváření souboru pro NAM, kde jsou záznamy o jednotlivých paketech, s definovaným jménem.

Přidání nové hlavičky - packet.h:

```
enum packet_t {
.
PT_RTPROTO_RIP,
.
};

class p_info {
public:
```

```
.
name_[PT_RTPROTO_RIP]="rtProtoRIP";
.
};
```

4.3.2 Hlavní třída

Implementace

Hlavní třída směrovacího protokolu RIP definuje proměnné a metody pro běh samotného protokolu, přijímání a odesílání paketů, vytváření databáze a reakce na změnu sítě. Její metody pracují nad 2 základními strukturami, databází a seznamem rozhraní.

Struktura třídy směrovací protokolu - rtProtoRIP.h

```
class rtProtoRIP : public Agent {
        u_int32_t                       node_id_;       //id uzlu
        u_int32_t                       node_addr_;     //adresa uzlu
        u_int32_t                       agent_port_;    //cislo portu

        int                             infinity_;      //max. metrika

        int                             version_;       //verze protokolu

        bool                            auto_summary_;  //nastaveni autosumarizace

        u_int32_t                       rip1address_;   //adresy pro pakety
        u_int32_t                       rip2address_;   //jednotlivych verzi

        list<interface_entry>           interfaces_;    //seznam rozhrani

        list<RIP_database_entry *>      RIP_database_;  //databaze siti

        int                             update_time_;   //implicitni
        int                             invalid_time_;  //hodnoty casovacu
        int                             flush_time_;

        u_int32_t                       debug_;         //nastaveni debuggingu

        RIP_UpdTimer                    upd_timer_;     //casovac aktualizaci
};
```

Databáze

Návrh

V databázi se uchovávají informace o sítích a cestách k nim. Databáze musí obsahovat všechny potřebné informace pro směrování, ale také pro vytváření zpráv pro ostatní instance. Každá záznam v databázi musí mít tyto informace:

- Adresa sítě
- Prefix sítě
- Metrika

- Adresa dalšího skoku
- Rozhraní na které tato informace přišla

Dále je s každým záznamem spojeno několik časovačů a je vhodné mít tyto časovače k dispozici skrz tento záznam o sítí, abychom nemuseli v některé další struktuře vyhledávat tyto časovače v případě, že je budeme chtít resetovat, měnit.

Implementace

Databáze je koncipována jako standardní kontejner list. Není na ní aplikováno žádné řazení, protože neslouží k vyhledávání cílů jednotlivých paketů. Slouží pouze k uložení informací, které se předají dál rtObjectu, který tyto informace propaguje dále do klasifikátoru.

Struktura záznamu databáze - rtProtoRIP.h

```
struct RIP_database_entry {
        u_int32_t               ip;             //identifikace site
        int                     prefix;
        int                     metric;
        u_int32_t               next_hop_addr;

        u_int32_t               intf_id;        //identifikace rozhrani

        bool                    notify;         //zda se muze odpovidat
                                                //na dotazy na tuto sit

        RIP_InvalidTimer*       invalid;        //casovace
        RIP_FlushTimer*         flush;
};
```

4.3.3 Seznam rozhraní

Návrh

Jednotlivé směrovací protokoly si musí sami udržovat stav jednotlivých rozhraní kvůli způsobu reakce ns-2 na tuto událost. Pokud se změní stav linky, jsou na to připojené uzly upozorněny. Uzly poté informují rtObject o změne rozhraní, ale bohužel neudavají které, a ten poté všechny směrovací protokoly. Z důvodu, že není uvedené které rozhraní se změnilo, musí si jednotlivé směrovací protokoly zjisit změny oproti jejich uloženému stavu a stavu skutečnému.

Nutnost mít vlastní seznam rozhraní, je však také vyžadována z důvodu nejrůznějších nastavení, která se provádějí na jednotlivých rozhraních, např. zda je rozhraní pasivní. Záznam o rozhraní by měl obsahovat alespoň tyto informace:

- Adresa rozhraní
- Prefix rozhraní
- Stav rozhraní
- Připojená linka

A poté záznamy o nastavení RIPu

- Zasílaná verze
- Přijímaná verze
- Rozhraní je zahrnuto v RIPu
- Pasivní rozhraní
- Stav rozděleného horizontu
- Způsob autentizace
- Seznam hesel

Ukazatel na linku mezi sousedem, je nutný z důvodu, že komunikace RIP probíhá broadcastových a multicastových adresách. Běžný agent pošle vytvořený paket klasifikátoru a ten již sám rozhodne kam, daná zpráva patří. RIP ale potřebuje poslat pakety se stejnou cílovou adresou, 255.255.255.255 nebo 224.0.0.9, ale různým obsahem na různé rozhraní. Z tohoto důvodu je nutné mít možnost poslat paket na konkrétní linku a k tomu slouží tento ukazatel.

Implementace

Seznam rozhraní je podobně jako databáze implementován jako standardní kontejner list. Jeho funkcí je uchovávat stav jednotlivých rozhraní a s ním spojená nastavení směrovacího protokolu RIP. Důležitou proměnou je ukazatel na linku mezi tímto a sousedským uzlem, kvůli možnosti odesílat zprávy na jednotlivá rozhraní. Jednotlivé zprávy jsou totiž různé v závislosti na rozhraní z důvodu aktivace různých metod zabraňujících počítání do nekonečna.

Z nastavení RIPu je podstatná proměnná `RIP_peer`, která označuje zda rozhraní bylo zahrnuto do RIP procesu pomocí příkazu `network`.

Struktura záznamu rozhraní - rtProtoRIP.h

```
struct interface_entry
{
        u_int32_t               nei_id;         //identifikator rozhrani

        u_int32_t               intf_addr;      //adresa rozhrani
        int                     intf_prefix;

        NsObject*               link;           //objekt pripojene linky

        bool                    intf_up;        //stav linky

        int                     send_version;   //odesilana verze
        int                     receive_version;//prijimana verze
        bool                    RIP_peer;       //je rozhrani zahrnuto v RIP?
        bool                    passive;        //pasivni rozhrani
        bool                    split_horizont; //rozdeleny horizont

        int                     auth_mode;      //mod autentizace
```

```
        char *                  key_chain;      //jmeno klicenky
        password_list           passwords;      //seznam hesel
};
```

4.3.4 Propojení C++ a OTcl tříd

Podobně jako u směrovacího modulu je nutné, propojit vytváření tříd v OTcl a C++. Při vytvoření třídy v OTcl se volá na pozadí funkce `create`, která se postará o vytvoření třídy v C++

Propojení OTcl/C++ - rtProtoRIP.cc:

```
static class rtProtoRIPClass : public TclClass
{
public:
        rtProtoRIPClass() : TclClass ("Agent/rtProto/RIP") {}
        TclObject* create (int argc, const char*const* argv) {
                return (new rtProtoRIP);
        }
} class_rtProtoRIP;
```

Nezbytné funkce

Pokud agent chce přijímat pakety musí mít definovanou metodu `recv(Packet *p, Handler *)`, která je volána ostatními částmi simulátoru. Prvek, který chce odeslat paket agentovi, zavolá metodu agenta `recv` a předá ji jako parametr zasílaný paket.

Dále musí být definována metoda `command`, která vytváří rozhraní pro OTcl, aby se z něj dali volat metody C++. OTcl při vyhodnocování funkcí interpretr nejdříve hledá metodu OTcl třídy, pokud ji nenalezne, zavolá metodu `unknown`, která volá metodu `cmd`. Ta volá metodu `command` objektu v C++.

Metoda ma poměrně jednoduchou strukturu několika `if`ů, rozdělující příkazy dle počtu parametrů. Prvni argument, `argv[0]` obsahuje vždy název metody, v tom to případe je to vždy 'cmd'. V `argv[1]` je poté uložena požadovaná operace. Na příkladu vidíme, že se jedná o operaci `version-c`. Další argumenty jsou poté jen argumenty dané operace. Návratová hodnota funkce musí být `TCL_OK` pokud vše proběhlo v pořádku nebo `TCL_ERROR` pokud se někde vyskytla chyba.

Příklad struktury metody `command` - rtProtoRIP.cc

```
if (argc == 3) {
        if (strcmp(argv[1], "version-c") == 0) {
                set_version(NULL, atoi(argv[2]), 0);
                return TCL_OK;
        }
}
```

4.3.5 Chování procesu

Po vytvoření objektu v Tcl a přidání adres, na které jsou odesílání směrovací aktualizace, do klasifikátoru je volána skrz příkaz `command` metoda `init` v C++ prostředí, která nastaví parametry procesu na implicitní hodnoty, které jsou uloženy v souboru ns-rtProtoRIP.tcl,

popřípadě tak aby odpovídalo chování Cisco prvků. Aktivuje také časovač aktualizací. Dále inicializuje seznam rozhraní, pro které také nastaví implicitní hodnoty.

V tomto stavu proces neodesílá ani nepřijímá žádné pakety. Pro aktivaci je nutné přidat alespoň jedno rozhraní do procesu pomocí příkazu `network`. Na linky, které jsou připojeny na přidaná rozhraní, jsou odeslány pomocí funkce `send_request` RIP pakety typu dotaz. Proces vytváření dotazu řídí metoda `make_request`, která se stará o naplnění hlavičky správnými informacemi, dle nastavení rozhraní. Samotné odeslání zprávy se provádí voláním metody `recv` linky na kterou chceme paket odeslat.

Pokud procesu dojde paket, někdo zavolal jeho metodu `recv`, je nejprve zkontrolováno zda paket byl přijat na rozhraní, které je zahrnuto v RIP procesu. Vzhledem k tomu, že uzel má pouze jedno vstupní rozhraní, je nutné projít celý seznam rozhraní a spolehnout se na kontrolu zda zdrojová adresa patří do stejné sítě jako adresa rozhraní. Je-li rozhraní zahrnuto v RIP procesu, je paket předán dál metodám, které řídí zpracování paketu. Metody jsou 2 v závislosti na typu paketu, `handle_response` a `handle_request`.

Funkce pro zpracování paketů nejprve kontrolují zda verze paketu odpovídá přijímané verzi na daném rozhraní a autentizaci paketu. Pokud paket projde těmito testy je dále zpracováván. V případě dotazu je zpět odeslaná odpověď. Odpověď je zpracovávána záznam po záznamu a jednotlivé záznamy jsou kontrolovány s databází. Pokud dojde ke změně v databázi je o této skutečnosti informován rtObject pomocí funkcí `add_route_entry` nebo `delete_route_entry`.

Po vypršení aktualizačního časovače jsou pomocí metody `send_updates`, která volá metodu `send_response` pro všechnny rozhraní. Časovač se poté nastaví na novou hodnotu. Podobně jako `make_request` i `send_response` volá funkci pro vytvoření paketu `make_response`. O záznamech, které budou či nebudou přidány do odpovědi se stará samostatná metoda `make_update`. Tato metoda zohledňuje všechna pravidla, která se používají při rozhodování, viz. sekce 2.2.1.

Reakce na změnu je definována v metodě `intf_changed_notify`.

4.4 Implementace uživatelského rozhraní

Návrh

Směrovací protokol RIP by měl být konfigurovatelný příkazy operačního systému, který běží na prvcích Cisco. Příkazy by ale měli byt co nejpodobnější.

Uživateli by měla stačit znalost uzlu, na kterém proces běží, aby mohl provést konfiguraci RIPu a nemusel adresovat samotný proces, agenta, ke kterému by se jinak musel dostávat přes rtObject uzlu, který obsahuje odkazy na směrovací protokoly.

Implementace

soubory: ns-nodeIPv4.tcl

Uživatelské rozhraní je reprezentováno metodou uzlu `#`, podobně jako symbol pro indikaci privilegovaného módu na Cisco zařízeních. Metoda přijímá jako argumenty CISCO příkazy. Při přecházení mezi módy privilegovaného režimu, tedy i konfiguračními, se postupně mění proměnná uzlu `ios_mode_`. Ta obsahuje aktuální pozici v privilegovaném režimu.

Příklad:

```
ios_mode_ = {config interface}
```

Pozice v režimech je interpretována seznamem řetězců, kde jednotlivé prvky jsou zároveň identifikátory módů.

Když uživatel volá nějaký IOS příkaz, pomocí metody `#`, tak je před tento příkaz přidán obsah proměnné `ios_mode_`. Takto vytvořený řetězec je předán interpretu OTcl jako funkce.

Příklad:

```
ios_mode_ = {config interface}

$n(1) # ip address 10.4.0.3 255.255.255.0
```

Vytvořený řetězec:

```
config interface ip address 10.4.0.3 255.255.255.0
```

Každý název módu, který je přidáván do proměnné `ios_mode_`, je zároveň definován i jako metoda. Řetězec v příkazu je tedy v podstatě metodou `config` s parametry `interface ip address 10.4.0.3 255.255.255.0`. Tyto parametry jsou poté dále interpretovány jako jednotlivé metody, dokud se nenarazí na mód, ve kterém je provedena samotná metoda uzlu, popřípadě se volá metoda určitého směrovacího protokolu.

Konfigurace uzlu poté může vypadat následujícím způsobem:

```
$n(1) # configure terminal
$n(1) # interface $n(2)
$n(1) # ip address 192.168.2.1 255.255.255.0
$n(1) # router rip
$n(1) # version 2
$n(1) # network 192.168.2.0
$n(1) # end
$n(1) # show ip route
```

4.4.1 Přehled implementovaných příkazů

V této sekci jsou uvedeny všechny implementované příkazy s jejich případnými omezeními.

- Všechny mody
  - `exit` - přejde v hierarchii módů o úroveň výš
  - `end` - přejde zpět do privilegovaného režimu
- Privilegovaný mód
  - `debug ip rip {packets|evetns|commands|all}` - debugovaní RIPu. Výpis se provádí do samostatného souboru. Defaultní název je Node_id.out, kde id je id daného uzlu
  - `show ip route` - výpis směrovací tabulky
  - `clear ip route {síť síťová_maska | *}` - smazání sítí z dynamických směrovacích protokolů
- Konfigurační mód - přístup z privilegovaného módu pomoci příkazu `configure [terminal]`
  - `ip route síť síťová_maska adresa_dalšího_skoku [preference]` - definování statické cesty, oproti Cisco zařízením je možný pouze 1 volitelný parametr a to preference. Dále chybí možnost definovat další skok pomocí rozhraní, lze pouze pomocí adresy.

  - `ip default-network síť` - definice defaultní síťě, funguje pouze ve smyslu propagace defaultní sítě v RIPu, na uzlu kde je tento příkaz zadán není žádná síť označna jako defaultní
  - `keychain název_klíčenky` - definice nové klíčenky

- Konfigurace klíčenky
  - `key číslo` - definice nového klíče v klíčence

- Konfigurace klíčů
  - `key-string heslo` - přiřazení hodnoty klíči

- Konfigurace RIP - přístup z konfiguračního módu pomocí příkazu `router rip`
  - `version {1|2}` - nastavení verze
  - `network síť` - přidaní sítí do procesu ripu
  - `auto-summary` - nastaveni auto sumarizace
  - `passive-interface {Sousedský_uzel| default}` - nastavení pasivního rozhraní
  - `timers basic update invalid holddown flush [sleep]` - nastavení časovačů, na proces ale mají vliv pouze update, invalid, flush

- Konfigurace rozhraní
  - `ip split-horizont` - nastavení rozděleného horizontu
  - `ip rip version {send|receive} {1|2|1 2}` - nastavení verze ripu na rozhraní
  - `ip rip authentication key-chain název_klíčenky` - přirazení klíčenky k rozhraní
  - `ip rip authentication mode název_módu` - nastavení autentizačního módu

4.5 Debuggování procesů

Návrh

Na Cisco zařízeních lze sledovat stav jednotlivých procesů pomocí nejrůznějších výpisů, například výpis směrovací tabulky, výpisy databází. Lze také aktivovat debugging, kdy se jednotlivé události procesů vypisují do konzole administrátora, který je připojený na dané zařízení.

Podobné funkce by měli být implementovány také v této implementaci. Bylo by vhodné zapisovat záznamy z jednotlivých uzlů do samostatných souborů, pokud by všechny záznamy byly vypisovány na standardní výstup bylo by to velice nepřehledné.

Implementace

Procesy mají k dispozici metodu uzlu `debug-msg`, které jako parametr předají debugovací výpis. Ten je poté odeslán na standardní výstup.

Pokud není žádoucí aby všechny uzly vypisovali svoje zprávy na standardní výstup, je možné definovat jednotlivým uzlům soubor, do kterého bude zapisován jejich výstup. K definici slouží metoda uzlu `enable-debug`, jako parametr se předává logovací soubor.

Logovací soubor je poté nutné uzavřít metodou uzlu `disable-debug`.

4.6 Konečné úpravy

Soubory: ns-lib.tcl, Makefile.in

Poslední úpravy se týkali přidání nově vytvořených Tcl souborů do knihovny, soubor ns-lib.tcl, jako zdrojů. A nakonec přidaní všech nových souboru do Makefile.in

Přidání souborů do knihovny - ns-lib.tcl

```
source ../rip/ns-rtProtoRIP.tcl

# Static
source ../routing_ipv4/ns-rtProtoStaticIPv4.tcl

# IPv4
source ../routing_ipv4/ns-rtObjectIPv4.tcl
source ../routing_ipv4/ns-rtProtoDirectIPv4.tcl
source ../routing_ipv4/ns-rtModuleIPv4.tcl
source ../routing_ipv4/ns-nodeIPv4.tcl
```

Přidání do makefilu - Makefile.in

```
OBJ_CC = \
.
rip/rtProtoRIP.o \
routing_ipv4/classifier-ipv4.o \
.

NS_TCL_LIB = \
.
tcl/rip/ns-rtProtoRIP.tcl \
tcl/routing_ipv4/ns-rtProtoDirectIPv4.tcl \
tcl/routing_ipv4/ns-rtObjectIPv4.tcl \
tcl/routing_ipv4/ns-rtProtoStaticIPv4.tcl \
tcl/routing_ipv4/ns-rtModuleIPv4.tcl \
tcl/routing_ipv4/ns-nodeIPv4.tcl \
.
```

Shrnutí

V této kapitole byl popsán návrh a implementace směrovacího protokolu RIP a několika dalších podpůrných struktur. Dále byl uveden i způsob integrace této nové implementace s původním kódem ns-2.

Kapitola 5

Testy

Úvod

Implementaci je nutné otestovat zda odpovídá směrovacímu protokolu RIP, který je k dispozici na směrovačích společnosti Cisco.

5.1 Návrh testů

Jako referenční program byl použit program společnosti Cisco Packet Tracer. Testy byli navrženy tak aby pokryli všechny implementované vlastnosti protokolu RIP. Byli vytvořeny scénáře v Packet Traceru i v ns-2, na nově implementované architektuře. Výstupy z obou programů byly poté porovnány, zda jsou shodné.

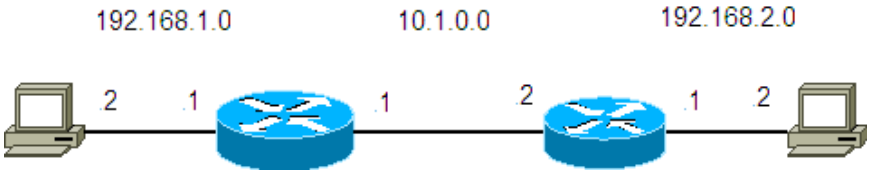
5.2 Test č. 1

Účel a popis testu

2 směrovače si mezi sebou vyměňují informace o svých připojených uživatelských sítí. Jeden z nich má deaktivovaný rozdělený horizont a rozhraní do uživatelské sítě nastavené jako pasivní. Očekávaným chováním by mělo být, že do této sítě nebudou propagovány aktualizace. Dále by měl být viditelný rozdíl v obsahu aktualizací z důvodu deaktivovaného rozděleného horizontu.

Účelem testu je testování rozdílu mezi verzemi, rozděleného horizontu a pasivního rozhraní.

Scénář je testován pro obě verze RIPu.



Obrázek 5.1: Topologie 1. testu

Verze 1

Packet Tracer

```
     10.0.0.0/24 is subnetted, 1 subnets
C       10.1.0.0 is directly connected, FastEthernet0/0
     192.168.1.0/27 is subnetted, 1 subnets
C       192.168.1.0 is directly connected, Ethernet1/0
R    192.168.2.0/24 [120/1] via 10.1.0.2, FastEthernet0/0

RIP: received v1 update from 10.1.0.2 on FastEthernet0/0
      10.1.0.0 in 1 hops
      192.168.1.0 in 2 hops
      192.168.2.0 in 1 hops

RIP: sending  v1 update via FastEthernet0/0 (10.1.0.1)
RIP: build update entries
      network 192.168.1.0 metric 1
```

ns-2

```
rtObject(1) routes:
C    10.1.0.0/24     is directly connected, 1->2
C    192.168.1.0/27  is directly connected, 1->0
R    192.168.2.0/24  [120/1] via 10.1.0.2, 1->2

RIP(1) received RESPONSEv1 from 10.1.0.2 at 1->2
        192.168.1.0/24 metric 2
        192.168.2.0/24 metric 1
        10.1.0.0/24 metric 1

RIP(1) sending RESPONSEv1 via 1->2
        192.168.1.0/24 metric 1
```

Verze 2 s automatickou sumarizací

Packet Tracer

```
     10.0.0.0/24 is subnetted, 1 subnets
C       10.1.0.0 is directly connected, FastEthernet0/0
     192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
R       192.168.1.0/24 [120/2] via 10.1.0.2, FastEthernet0/0
C       192.168.1.0/27 is directly connected, Ethernet1/0
R    192.168.2.0/24 [120/1] via 10.1.0.2, FastEthernet0/0

RIP: sending  v2 update via FastEthernet0/0 (10.1.0.1)
RIP: build update entries
      192.168.1.0/24 via 0.0.0.0, metric 1, tag 0

RIP: received v2 update from 10.1.0.2 on FastEthernet0/0
      10.1.0.0/24 via 0.0.0.0 in 1 hops
      192.168.1.0/24 via 0.0.0.0 in 2 hops
      192.168.2.0/24 via 0.0.0.0 in 1 hops
```

ns-2

```
rtObject(1) routes:
C    10.1.0.0/24     is directly connected, 1->2
R    192.168.1.0/24  [120/2] via 10.1.0.2, 1->2
C    192.168.1.0/27  is directly connected, 1->0
R    192.168.2.0/24  [120/1] via 10.1.0.2, 1->2

RIP(1) sending RESPONSEv2 via 1->2
        192.168.1.0/24 metric 1

RIP(1) received RESPONSEv2 from 10.1.0.2 at 1->2
        192.168.1.0/24 metric 2
        192.168.2.0/24 metric 1
        10.1.0.0/24 metric 1
```

Porovnání výsledků

Výstupy z IOS byli zkráceny o některé méně podstatné informace, časy od poslední aktualizace, adresa na kterou byli odeslány aktualizce. Z výstupu ns-2 byly odstraněny časová razítka. Záznamy paketu byli seřazeny do odpovídajícího pořadí.

Z výše uvedených výsledků si můžeme povšimnout, že si výstupy odpovídají.

Rozdíl verzí je evidentní z obsahů směrovacích tabulek, každá verze zpracovává informace jiným způsobem. Funkce rozděleného horizontu je možné si ověřit, tím že uzel odesílá pouze svoji uživatelskou síť. Naopak druhý uzel, kde je funkce rozděleného horizontu vypnuta, posílá zpět i sítě, které získal od tohoto směrovače.

Skrz pasivní rozhraní, směrem k uzlu číslo 0, popřípadě Ethernet1/0, nejsou odesílány aktualizace.

5.3 Test č. 2

Účel a popis testu

4 směrovače jsou zapojeny podle níže uvedené topologie. Mezi horními 2 je nastavena textová autentizace, mezi spodními md5 verze. Na spojnicích mezi horními a dolními byli nastaveny neplatné kombinace autentizace, špatné heslo a rozdílné módy. V topologii je nespojitá síť 10.0.0.0, jejíž podsítě slouží jako uživatelské sítě. V případě, že by byla aktivní automatická sumarizace, propagovala by se síť ve formě, 10.0.0.0 a ostatní uzly by si ji nepřidali nebo ve tvaru 10.0.0.0/8, což je nežádoucí chování. Automatická sumarizace je na všech směrovacích vypnuta.

Legenda k tabulce 5.3, jednotlivá písmena určují umístění směrovače: T - horní, B - dolní, L - levý, R - pravý

Účelem testu bylo otestovat autentizaci aktualizací a způsob chování při neaktivní autosumarizaci. Další testovanou skutečností bylo přiřazení přijímaných a odesílaných verzí na

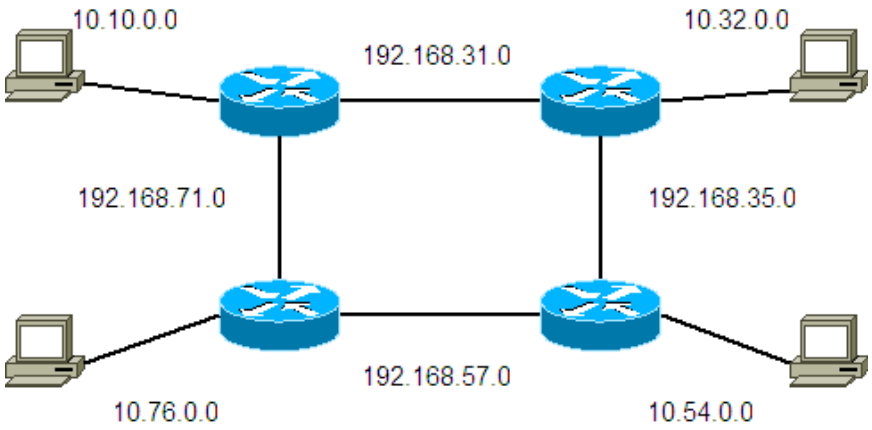
Tabulka 5.1: Znalosti hesel jednotlivých směrovačů

Směrovač	Rozhraní k	Typ	Hesla na rozhraní
TL	TR	text	1key3, 3key1
	BL	text	1key7, 7key1
TR	TL	text	1key3, 3key1
	BR	text	3key5
BR	TR	text	5key3
	BL	md5	5key7, 7key5
BL	BR	md5	5key7, 7key5
	TL	md5	1key7, 7key1

jednotlivá rozhraní.

V současné verzi Paket Traceru chybí podpora pro autentizaci paketu i přiřazení verzí na rozhraní. Jako validní chování bylo zvoleno chování podle RFC, tzn. pokud je má byt rozhraní autentizováno a příchozí paket nemá autentizaci není přijat. Není přijat také v případě, že hodnota autentizace je různá od nestaveného hesla, popřípadě od vypočítaného hashe.

Dále pokud je na rozhraní nastaveno aby odesílalo verzi 1 i 2, mají být podle informací z [5] odeslány oba pakety.



Obrázek 5.2: Topologie 2. testu

Porovnání výsledků

V Paket Traceru byla použita menší topologie, pouze horní 2 směrovače, pro otestování automatické sumarizace. Větší nebyla použita z důvodu nemožnosti testovat autentizaci. Výstupy si odpovídají, pouze ve výstupu z ns-2 jsou navíc sítě, konkrétně 192.168.71.0 a 192.168.35.0. Správná funkce deaktivované autentizace je indikována tím, že podsítě sítě 10.0.0.0 jsou propagovány se správným prefixem.

Výstup z uzlu 3 ukazuje, že pokud dojde paket bez autentizace, např. paket verze 1, na autentizované rozhraní, je zahozen. V případě, že dojde paket s autentizací, je prohledána databáze hesel přiřazená k danému rozhraní.

Chování autosumarizace

Packet Tracer

```
     10.0.0.0/24 is subnetted, 2 subnets
C       10.10.0.0 is directly connected, Ethernet1/0
R       10.32.0.0 [120/1] via 192.168.13.3, FastEthernet0/0
C    192.168.13.0/24 is directly connected, FastEthernet0/0

RIP: received v2 update from 192.168.13.3 on FastEthernet0/0
      10.32.0.0/24 via 0.0.0.0 in 1 hops

RIP: sending  v2 update via FastEthernet0/0 (192.168.13.1)
RIP: build update entries
      10.10.0.0/24 via 0.0.0.0, metric 1, tag 0
```

ns-2

```
rtObject(1) routes:
C    10.10.0.0/24    is directly connected, 1->0
R    10.32.0.0/24    [120/1] via 192.168.13.3, 1->3
C    192.168.13.0/24 is directly connected, 1->3
R    192.168.35.0/24 [120/1] via 192.168.13.3, 1->3
C    192.168.71.0/24 is directly connected, 1->7

RIP(1) received RESPONSEv2 from 192.168.13.3 at 1->3
        authentication: 1key3
        192.168.35.0/24 metric 1
        10.32.0.0/24 metric 1

RIP(1) sending RESPONSEv2 via 1->3
        authentication: 3key1
        192.168.71.0/24 metric 1
        10.10.0.0/24 metric 1
```

Chování autentizace

Uzel 3

```
RIP(3) received RESPONSEv1 from 192.168.13.1 at 3->1
        authentication failed

RIP(3) received RESPONSEv2 from 192.168.13.1 at 3->1
        authentication: 3key1
        192.168.71.0/24 metric 1
        10.10.0.0/24 metric 1

RIP(3) received RESPONSEv2 from 192.168.35.5 at 3->5
        authentication: 5key3
        authentication failed

RIP(3) sending RESPONSEv2 via 3->1
        authentication: 1key3
        192.168.35.0/24 metric 1
        10.32.0.0/24 metric 1

RIP(3) sending RESPONSEv2 via 3->5
        authentication: 3key5
        10.10.0.0/24 metric 2
        192.168.71.0/24 metric 2
        192.168.13.0/24 metric 1
        10.32.0.0/24 metric 1
```

Uzel 7

```
RIP(7) received RESPONSEv2 from 192.168.71.1 at 7->1
        authentication: 7key1
        authentication failed

RIP(7) sending RESPONSEv2 via 7->5
        authentication: Hash_5key7
        192.168.71.0/24 metric 1
        10.0.0.0/8 metric 1

RIP(7) sending RESPONSEv2 via 7->1
        authentication: Hash_1key7
        10.0.0.0/8 metric 1
        192.168.35.0/24 metric 2
        192.168.57.0/24 metric 1

RIP(7) received RESPONSEv2 from 192.168.57.5 at 7->5
        authentication: Hash_7key5
        192.168.35.0/24 metric 1
```

Při komunikace mezi uzly 3 a 1 jsou použitá hesla v databázích obou směrovačů. Mezi 3 a 5 chybí stejná hesla, z toho důvodu komunikace mezi nimi neprobíhá, jak informuje hláška `autentication failed` po přijetí paketu. Mezi uzly 5 a 7 je komunikace v pořádku, ale je použita md5 autentizace, tento typ se odlišuje prefixem `Hash_` před heslem. Uzly 1 a 7 by spolu mohli komunikovat, mají stejná hesla, ale bohužel mají každý nastavený jiný mód autentizace.

5.4 Test č. 3

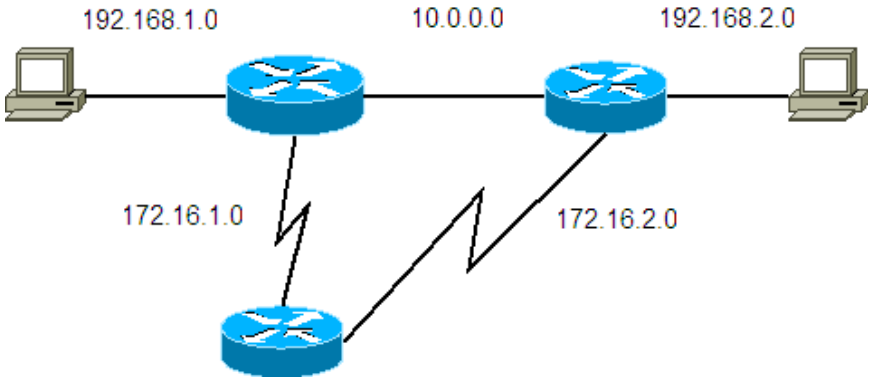
Účel a popis testu

Test má prokázat správnou funkci rtObjectu při kombinaci více protokolů, statického směrování a částečnou funkci příkazu `ip defaul-network`. Dále také chování, způsob propagace sítí, při spadnutí linky.

Pravý směrovač propaguje levému implicitní cestu, 0.0.0.0, skrz sebe. Pokud je linka mezi pravým a levým horním směrovačem nefunkční, použije pravý směrovač pomalejší záložní cestu k dolnímu směrovači.

Porovnání výsledků

Výstupy si odpovídají, až na označování implicitní cesty ve směrovací tabulce. V současné době není tato vlastnost implementována.



Obrázek 5.3: Topologie 3. testu

Cisco IOS

```
     10.0.0.0/24 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, FastEthernet0/0
     172.16.0.0/24 is subnetted, 1 subnets
C       172.16.2.0 is directly connected, FastEthernet0/1
R    192.168.0.0/24 [120/1] via 10.0.0.1, FastEthernet0/0
C    192.168.2.0/24 is directly connected, FastEthernet1/0
R*   0.0.0.0/0 [120/1] via 10.0.0.1, FastEthernet0/0

Line protocol on Interface FastEthernet0/0, changed state to down

     172.16.0.0/24 is subnetted, 1 subnets
C       172.16.2.0 is directly connected, FastEthernet0/1
C    192.168.2.0/24 is directly connected, FastEthernet1/0
S*   0.0.0.0/0 [200/0] via 172.16.2.4

RIP: sending  v1 update via FastEthernet1/0 (192.168.2.2)
RIP: build update entries
      network 0.0.0.0 metric 16
      network 192.168.0.0 metric 16

Line protocol on Interface FastEthernet0/0, changed state to up

RIP: sending  v1 request via FastEthernet0/0 (10.0.0.2)

     10.0.0.0/24 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, FastEthernet0/0
     172.16.0.0/24 is subnetted, 1 subnets
C       172.16.2.0 is directly connected, FastEthernet0/1
R    192.168.0.0/24 [120/1] via 10.0.0.1, FastEthernet0/0
C    192.168.2.0/24 is directly connected, FastEthernet1/0
R*   0.0.0.0/0 [120/1] via 10.0.0.1, FastEthernet0/0
```

ns-2

```
rtObject(2) routes:
R    0.0.0.0/0       [120/1] via 10.0.0.1, 2->1
C    10.0.0.0/24     is directly connected, 2->1
C    172.16.2.0/24   is directly connected, 2->4
R    192.168.0.0/24  [120/1] via 10.0.0.1, 2->1
C    192.168.2.0/24  is directly connected, 2->3

RIP(2->1) interface changed state to down

rtObject(2) routes:
S    0.0.0.0/0       [200/0] via 172.16.2.4, 2->4
C    172.16.2.0/24   is directly connected, 2->4
C    192.168.2.0/24  is directly connected, 2->3

RIP(2) sending RESPONSEv1 via 2->3
        0.0.0.0/0 metric 16
        192.168.0.0/24 metric 16

RIP(2->1) interface changed state to up

RIP(2) sending REQUESTv1 via 2->1

rtObject(2) routes:
R    0.0.0.0/0       [120/1] via 10.0.0.1, 2->1
C    10.0.0.0/24     is directly connected, 2->1
C    172.16.2.0/24   is directly connected, 2->4
R    192.168.0.0/24  [120/1] via 10.0.0.1, 2->1
C    192.168.2.0/24  is directly connected, 2->3
```

Souhrn

Byli otestovány všechny funkce, které byli implementovány a odpovídají protokolu RIP, tak jak je implementován v programu Packet Tracer, dá se tedy předpokládat, že bude odpovídat i implementaci na skutečném zařízení.

Kapitola 6

Závěr

Implementace RIPu rozšířila možnosti směrování síťového simulátoru ns-2, které byly poměrně omezené ve smyslu podpory IPv4. Dále bylo vytvořeno uživatelské rozhraní, které dovoluje uživateli konfigurovat jednotlivé agenty a uzly podobně jako Cisco prvky. Implementace odpovídá chováním implementaci na Cisco zařízeních.

Implementace není konečná a poskytuje prostor pro další rozšíření. Konkrétně podpůrná struktura rtObject je možné rozšířit o podporu rozdělování zátěže či výběr implicitní sítě. Další vhodnou úpravou z dlouhodobého hlediska, je implementovat rozhraní jako samostatné objekty, místo jejich současné reprezentace v podobě pouhé adresy. Na těchto objektech by bylo možné lépe implementovat podporu pro nižší vrstvy popřípadě s nimi svázat přístupové seznamy.

Literatura

[1] P. Ruiz F. Ros. Implementing a new manet unicast routing protocol in ns2. Technical report, University of Mauricia, 2004.

[2] C. Hedrick. Rfc 1058, routing information protocol. Technical report, Rutgers University, 1988.

[3] J. Lindqvist. Counting to infinity. Technical report, Helsinki University of Technology, 2004.

[4] G. Malkin. Rfc 2453, rip version 2. Technical report, Bay Networks, 1998.

[5] WWW stránky. Cisco ios release 12.0 network protocols configuration guide. `http://www.cisco.com/en/US/docs/ios/12_0/np1/configuration/guide/np1_c.html`.

[6] WWW stránky. The network simulator ns-2. `http://www.isi.edu/nsnam/ns/`.

[7] WWW stránky. Wikipedia. `http://www.wikipedia.org`.