VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ PROTOKOLU PIM-SM V PROSTŘEDÍ OMNET++

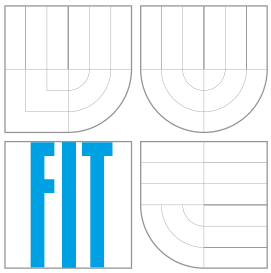
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ PROCHÁZKA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ PROTOKOLU PIM-SM V PROSTŘEDÍ OMNET++

MODELLING PIM-SM IN OMNET++

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ PROCHÁZKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR VESELÝ

BRNO 2013

Abstrakt

Ve své diplomové práci se zabývám modelováním a simulací multicastového směrovacího protokolu PIM Sparse Mode v nástroji OMNeT++. Čtenář se seznámí se základními informacemi o multicastu, s protokolem PIM-SM, jeho konfigurací na zařízeních Cisco. Zároveň bude uveden do problematiky vizualizace multicastových toků v síti. Práce je zaměřena zejména na návrh a implementaci protokolu PIM-SM v OMNeT++ a rozšíření knihovny ANSAINET o další multicastový směrovací protokol.

Abstract

In this master's thesis I deal with modelling and simulating of multicast routing protocol PIM Sparse Mode in OMNeT++. I also describe basic information about multicast, protocol PIM-SM, its configuration and multicast data streams visualization in computer networks. The thesis is especially focused on design and implementation of PIM-SM in OMNeT++ and extension of ANSAINET library.

Klíčová slova

PIM Sparse Mode, PIM, OMNeT++, INET, multicast, simulace, multicastové směrování, vizualizace multicastu v síti

Keywords

PIM Sparse Mode, PIM, OMNeT++, INET, multicast, simulation, multicast routing, multicast visualization in network

Citace

Tomáš Procházka: Modelování protokolu PIM-SM v prostředí OMNeT++, diplomová práce, Brno, FIT VUT v Brně, 2013

Modelování protokolu PIM-SM v prostředí OMNeT++

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Veselého

.....
Tomáš Procházka
16. května 2013

Poděkování

Touto cestou bych rád vyjádřil své poděkování všem, kteří mě podpořili při tvorbě této práce. Nejdříve bych chtěl poděkovat Vladimírovi. Nejen za odbornou pomoc, ale i za jeho nadhled, klid a trpělivost, díky kterým jsem z konzultací odcházel povzbuzen do další práce a těšil se na následující schůzky. Dále bych rád poděkoval svým kolegům a kamarádům z výzkumné skupiny ANSA, kteří mi pomohli s nástrahami OMNeTu a INETu.

Velký dík patří zejména mým rodičům, kteří mě podporovali po celou dobu studia, a bez kterých bych jen těžko dosáhl až na práh inženýrského studia. V neposlední řadě bych chtěl poděkovat své přítelkyni za její podporu a pomoc při řešení záludností s gramatikou.

Obsah

1 Úvod **4**

2 Multicast **5**
2.1 Multicast obecně . . . 5
2.1.1 L3 a L2 adresace . . . 6
2.1.2 IGMP protokol . . . 6
2.1.3 IGMP snooping . . . 7
2.1.4 Multicastové směrování . . . 7

3 PIM Sparse Mode **8**
3.1 Základní pojmy . . . 8
3.2 Činnost protokolu PIM Sparse Mode . . . 9
3.3 Zprávy protokolu PIM Sparse Mode . . . 11
3.3.1 Hello zprávy . . . 13
3.3.2 PIM Register zpráva . . . 14
3.3.3 PIM Join/Prune zpráva . . . 15
3.3.4 PIM Assert zpráva . . . 16
3.4 Stavy a konečné automaty protokolu PIM . . . 17
3.4.1 Register stavový automat . . . 17
3.4.2 Downstream (*,*,RP) stavový automat . . . 19
3.4.3 Downstream (*,G) stavový automat . . . 20
3.4.4 Downstream (S,G) stavový automat . . . 21
3.4.5 Downstream (S,G,rpt) stavový automat . . . 21
3.4.6 Upstream (*,*,RP) stavový automat . . . 23
3.4.7 Upstream (*,G) stavový automat . . . 24
3.4.8 Upstream (S,G) stavový automat . . . 25
3.4.9 Upstream (S,G,rpt) stavový automat . . . 25
3.4.10 Assert (S,G) stavový automat . . . 26
3.4.11 Assert (*,G) stavový automat . . . 30
3.5 Volba RP směrovače . . . 33
3.5.1 Bootstrap mechanismus . . . 33

4 Simulační nástroj OMNeT++ a knihovna INET **34**
4.1 OMNeT++ . . . 34
4.1.1 Modelování v OMNeT++ . . . 34
4.2 INET . . . 34
4.2.1 Současné možnosti multicastu v prostředí OMNeT++ a INET . . . 35

5 Podpora multicastu na Cisco zařízeních **36**
5.1 Konfigurace PIM-SM . . . 36
5.1.1 Povolení multicastového směrování . . . 36
5.1.2 Nastavení módu směrování . . . 36
5.1.3 Nastavení spt-threshold . . . 36
5.1.4 Konfigurace RP směrovače . . . 37
5.2 Ověření konfigurace . . . 38
5.3 Multicastové zprávy . . . 38

6 Nástroje pro vizualizaci multicastových toků **40**
6.1 Získávání informací o síťových topologiích . . . 40
6.1.1 SNMP . . . 40
6.1.2 CDP a LLDP . . . 41
6.1.3 Ping a traceroute . . . 41
6.1.4 DNS dotazy . . . 41
6.2 Mapování multicastu . . . 42
6.2.1 SNMP a multicast . . . 42
6.2.2 Vybrané nástroje . . . 42
6.2.3 HP Network Node Manager i . . . 44
6.2.4 EMC Ionix Multicast Manager . . . 45

7 Návrh rozšíření simulátoru OMNeT++ o podporu PIM-SM **46**
7.1 PIM-SM . . . 46
7.1.1 Diagram aktivit pro interní zprávy a události . . . 47
7.1.2 Diagram aktivit pro externí zprávy . . . 47
7.2 Konfigurace PIM-SM na síťových prvcích v OMNeT++ . . . 49

8 Implementace multicastového směrování v prostředí OMNeT++ **51**
8.1 Úpravy a portování modulu PIM pro novou verzi INET . . . 51
8.1.1 Multicastové směrování . . . 52
8.1.2 Síťová vrstva . . . 52
8.2 Implementace podpůrných struktur . . . 53
8.2.1 PIM zprávy . . . 53
8.2.2 PIM časovače . . . 53
8.3 Implementace protokolu PIM-SM . . . 54
8.3.1 Modul PIM Splitter . . . 54
8.3.2 Implementace modulu pimSM . . . 55
8.3.3 Shrnutí . . . 59

9 Porovnání simulace s chováním reálné sítě v prostředí Cisco **61**
9.1 Metodika porovnávání . . . 61
9.2 Testovací scénáře . . . 61
9.2.1 Scénář 1 - „registrace, nejdříve příjemci poté zdroj“ . . . 62
9.2.2 Scénář 2 - „registrace, nejdříve zdroj poté příjemci“ . . . 66
9.2.3 Scénář 3 - ověření činnosti časovačů PIM-SM . . . 68
9.3 Shrnutí . . . 73

10 Závěr **74**

A Seznam zkratek **81**

B Obsah CD **82**

C Makra **83**
C.1 Makra pro výběr DR směrovače 83
C.2 Makro CouldRegister(S,G) Register stavového automatu 83
C.3 Makro CouldAssert(S,G,I) pro Assert (S,G) stavový automat 84
C.4 Makra JoinDesired pro příslušné stavové automaty 84
C.5 Makra pro Upstream (S,G,rpt) stavový automat 85

D Implementace protokolu PIM-SM **86**
D.1 Implementace hlavičky třídy pimSM 86

Kapitola 1

Úvod

S odstupem času a s narůstajícím počtem uživatelů internetu vznikly nové požadavky k využívání vysokorychlostního připojení. Stále více mezi uživateli roste zájem o multimediální aplikace a komunikaci jako například: videokonference, živé rádiové/televizní vysílání, atd. Tyto požadavky byly impulsem pro efektivnější využití přenosového pásma, kdy typicky jeden zdroj vysílá data mezi více příjemců. Pro splnění těchto požadavků byl vyvinut multicast, který je dnes využíván pro streamování vysílání rádiových stanic, televizních pořadů, videostreamingu a pro další účely, kde se nám vyplatí efektivní komunikace mezi jedním odesílatelem a více příjemci.

Tato práce se zabývá metodou přeposílání IP datagramů více koncovým subjektům v počítačových sítích, tedy multicastem. Zaměřuje se zejména na multicastový směrovací protokol PIM Sparse Mode. Dále jsou v práci zkoumány možnosti mapování síťových topologií a multicastových toků.

Práce je členěna do několika kapitol. V první se zaměříme na shrnutí základních poznatků o multicastu. Ve druhé si podrobně popíšeme směrovací protokol PIM v Sparse módu. Dále je v práci popsán nástroj OMNeT++ a knihovna INET. Čtenář je také seznámen s konfigurací a podporou multicastu na Cisco zařízeních. Teoretickou část uzavírá kapitola věnovaná vizualizaci multicastových toků v síti. Praktická část práce zahrnuje návrh na rozšíření simulátoru OMNeT++ a návrh konfiguračního souboru. V dalších kapitolách je popsána implementace protokolu PIM-SM a porovnání výsledků simulace s chováním reálné sítě.

Kapitola 2

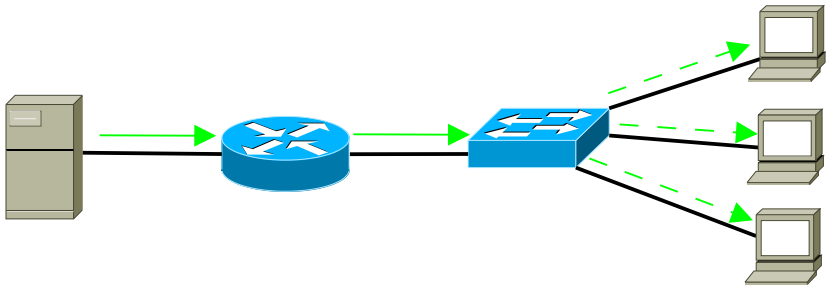
Multicast

Historie multicastu spadá do 80. let, kdy student doktorského studia Steve Deering pracoval na prvním distribuovaném systému zvaném Vsystem.

Steve Deering se snažil vymyslet protokol, který by byl schopný podporovat multicastové toky v IP sítích. Jeho prvním protokolem byl IGMP (Internet Group Message Protokol), který umožňoval koncovým stanicím připojení (join) nebo odpojení (leave) do multicastových skupin. Druhý Deeringův protokol DVMRP (Distance Vector Multicast Routing Protocol) byl navržen pro spolupráci směrovačů s podporou multicastu pro sdílení informací o koncových stanicích požadujících multicast. První multicastový systém MBone (Multicast backbone), který využíval právě protokolů IGMP a DVMRP, vznikl roku 1992. [17]

2.1 Multicast obecně

Multicast v této práci budeme chápat na druhé (linkové) a třetí (síťové) vrstvě ISO/OSI modelu. Jak již bylo naznačeno, multicast používáme tehdy, jestliže požadujeme rozeslání stejných dat více příjemcům, typicky videostreaming a další podobné služby. V praxi je tato distribuce dat vykonávána pomocí směrovačů, přepínačů a protokolů podporujících multicast. Data jsou ze zdroje rozesílána v jednom toku, který je směrován pomocí multicastových protokolů až k přepínači, ke kterému jsou připojeny koncové stanice odebírající multicastová data. Z tohoto přepínače jsou data následně replikována všem stanicím, které o daný multicastový tok zažádaly. Princip je znázorněn na obrázku 2.1.



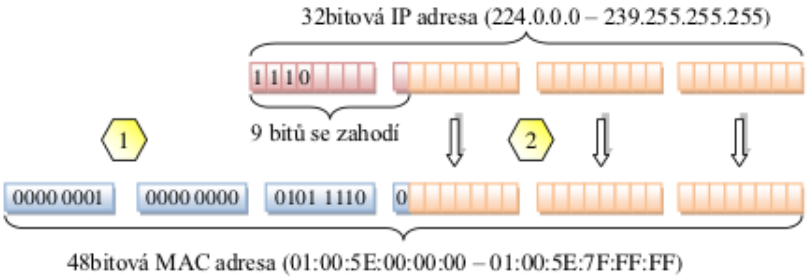
Obrázek 2.1: Základní myšlenka multicastu

2.1.1 L3 a L2 adresace

Multicastová komunikace používá proti unicastu rozdílné adresování jak na L3, tak na L2 vrstvě. Pro označení multicastové skupiny je používána multicastová IP adresa.

Na L3 vrstvě ISO/OSI modelu je používána IPv4 adresa třídy D v rozsahu 224.0.0.0 – 239.255.255.255. Tento rozsah je dále rozdělen do tří skupin: Lokální, Globální a Administrativní. U protokolu IPv6 se omezíme pouze na informaci, že multicastová adresa začíná prefixem FF**, kde ** rozlišují dosah, pro který jsou adresy využívány.

Adresace na linkové, tedy L2 vrstvě, používá mapování multicastové IP adresy na MAC adresu. 25 „horních“ bitů MAC adresy zůstává zachováno a následných 23 bitů je vyplněno multicastovou IP adresou, ze které je zahozeno prvních 9 bitů. Tento postup zachycuje obrázek 2.2.



Obrázek 2.2: Mapování multicastové IP adresy na adresu MAC [13]

2.1.2 IGMP protokol

Jedná se o protokol pracující na 3. vrstvě ISO/OSI modelu mezi stanicí a směrovačem popřípadě přepínačem. IGMP umožňuje:

- stanicím přihlašovat nebo odhlašovat od multicastových skupin
- směrovačům/přepínačům zjišťovat, zda má některá ze stanic zájem o multicast

Od vzniku tohoto protokolu došlo k úpravám, které jej dovedly do aktuální verze IGMPv3, definovanou v RFC 3376.

- IGMPv1 (General Query) — Směrovač nabízí jakýkoliv multicast. Klienti si určí, které multicastové skupiny budou odebírat.
- IGMPv2 (Group Specific Query) — Směrovač nabízí multicast konkrétní multicastové skupiny.
- IGMPv3 (Group and Source Query) — Směrovač specifikuje jak skupinu, tak zdroj multicastu, který nabízí.

Alternativou IGMP pro IPv6 je protokol MLD (Multicast Listener Discovery). Aktuální verze MLDv2 definovaná RFC 3810 je podobná protokolu IGMPv2. [14][9]

2.1.3 IGMP snooping

IGMP snooping je technika, která řeší obvyklé chování přepínačů při příjmu multicastových rámců. V tomto případě se k multicastovým datům zachovají jako k broadcastu a rozešlou je všemi porty přepínače, což může vést k zahlcení sítě.

Při IGMP snoopingu prozkoumáváme pakety a zjišťujeme, zda obsahují nějaké informace o multicastu, při kterém si přepínače vytvoří tabulku mapování multicastových skupin na svá rozhraní.

Proces zkoumání paketů je však náročný na výpočetní prostředky přepínačů, které byly primárně navrženy pro přepínání rámců. Z tohoto důvodu vznikaly další protokoly, které problém řešily. Jedním z nich je například protokol CGMP (Cisco Group Management Protocol).

2.1.4 Multicastové směrování

Unicastové směrování probíhá za pomoci směrovacích protokolů jako například RIP, EIGRP, OSPF na základě toho, kam mají být data doručena. Multicastové směrování je založeno na zdroji dat a distribučních stromech. Mezi multicastové protokoly řadíme [14]:

- Směrovací protokoly:
  - MOSPF — rozšíření OSPF, není používáno
  - DVMRP — základ MBONE, inspirace protokolem RIP
  - PIM — nejpoužívanější multicastový protokol
  - M-BGP — nástavba nad BGP
- Podpůrné protokoly:
  - MSDP — šíření zdrojů multicastu mezi autonomními systémy
  - RGMP — CGMP pro routery v páteřní vrstvě

Kapitola 3

PIM Sparse Mode

PIM neboli Protocol Independent Multicast je protokol sloužící pro směrování multicastových dat. Jeho název je odvozen z faktu, že při své činnosti nevyužívá žádný jiný unicastový směrovací protokol (například RIP, EIGRP, OSPF), a to při technologii RPF (Reverse Path Forwarding), která zabraňuje vzniku smyček při směrování. Tato technologie je svázána s myšlenkou, která říká: „Přepošli multicastová data na všechny své porty tehdy a jedině tehdy, když přišla portem, který je ve směru ke kořeni distribučního stromu [13].“ K tomuto účelu RPF používá porovnání zdrojové IP adresy paketu a unicastové IP adresy ze směrovací tabulky routeru. Protokol PIM může pracovat v několika módech:

- PIM Sparse Mode
- PIM Dense Mode
- Bidirectional PIM
- PIM Source-Specific Multicast

Tato kapitola popisuje PIM Sparse Mode (PIM-SM). Sparse, neboli řídký, značí chování módu, kdy jsou multicastová data zasílána pouze těm odběratelům, kteří si o daný zdroj zažádají. Toto chování je opačné PIM Dense módu, kdy je multicast zasílán všem uzlům v síti, a pouze ti, kteří nechtějí odebírat multicastová data, upozorní odesílatele, aby jim nebyla zasílána. Následující informace byly čerpány z RFC 4601 [7] a [1].

3.1 Základní pojmy

Při popisu protokolu PIM se setkáme s několika důležitými pojmy, které si nyní objasníme.

- **Rendezvous point (RP)** — směrovač, který byl vybrán a nakonfigurován jako kořen sdíleného stromu vybrané multicastové skupiny. RP přijímá zrávy *Join* (viz dále) od odběratelů multicastu a data od zdrojů multicastu. Pro každou multicastovou skupinu musí být nakonfigurován jeden RP, jinak by nebyl PIM-SM schopný fungovat, a to z toho důvodu, že by v síti nemohl být vytvořen zdrojový strom mezi RP směrovačem a zdrojem multicastových dat a sdílený strom mezi RP směrovačem a příjemcem multicastu. V takové počítačové síti by nebylo možné provádět distribuci multicastových dat pomocí PIM-SM.

- **Designed router (DR)** — jeden ze směrovačů PIM-SM, který byl vybrán v daném LAN segmentu nebo na *point-to-point* rozhraních jako prostředník mezi ostatními PIM-SM směrovači a uzly, které odebírají multicast.
- **Multicast Routing Information Base (MRIB)** — je to multicastová směrovací tabulka, která je většinou vytvořena z unicastové směrovací tabulky nebo ze směrovacích protokolů, jako například MBGP. MRIB slouží k rozhodování, kam odesílat *Join/Prune* zprávy, a k určení směrovacích metrik pro cílové IP adresy při zasílání *Assert* zpráv na základě adresy *next hop* směrovače. Data jsou pak zasílána opačným směrem, než *Join/Prune* zprávy.
- **Reverse Path Forwarding Neighbor** — *RPF Neighbor* je směrovač, který na základě IP adresy a MRIB může být použit pro odesílání paketů na tuto IP adresu. V případě PIM-SM multicastové skupiny to je směrovač, kam budou směrovány *Join* zprávy pro danou multicastovou skupinu.
- **Tree Information Base (TIB)** — databáze stavů na PIM-SM směrovači, která je měněna na základě PIM, IGMP nebo MLD zpráv. Tato databáze vpodstatě ukládá stav všech distribučních stromů v směrovači. Informace z tabulky TIB jsou důležité pro zasílání multicastových datagramů.
- **Multicast Forwarding Information Base (MFIB)** — zasílání multicastových datagramů je však pomocí tabulky TIB neefektivní, proto je zavedena tabulka MFIB. Tabulka MFIB je vytvořena na základě tabulky TIB, avšak tak, aby bylo zasílání multicastových datagramů efektivní. Provedení tohoto mechanismu záleží na implementaci, v našem případě pomocí diagramu aktivit uvedeného v kapitole 7.
- **Upstream rozhraní** — rozhraní směrem ke kořeni stromu. Za kořen může být považován zdroj nebo RP směrovač.
- **Downstream rozhraní** — rozhraní směrem od kořene stromu.

3.2 Činnost protokolu PIM Sparse Mode

Protokol PIM Sparse Mode pracuje ve třech režimech. Jelikož se příjemci a zdroje multicastu mohou připojovat a odpojovat „dynamicky“, mohou tyto fáze nastat souběžně.

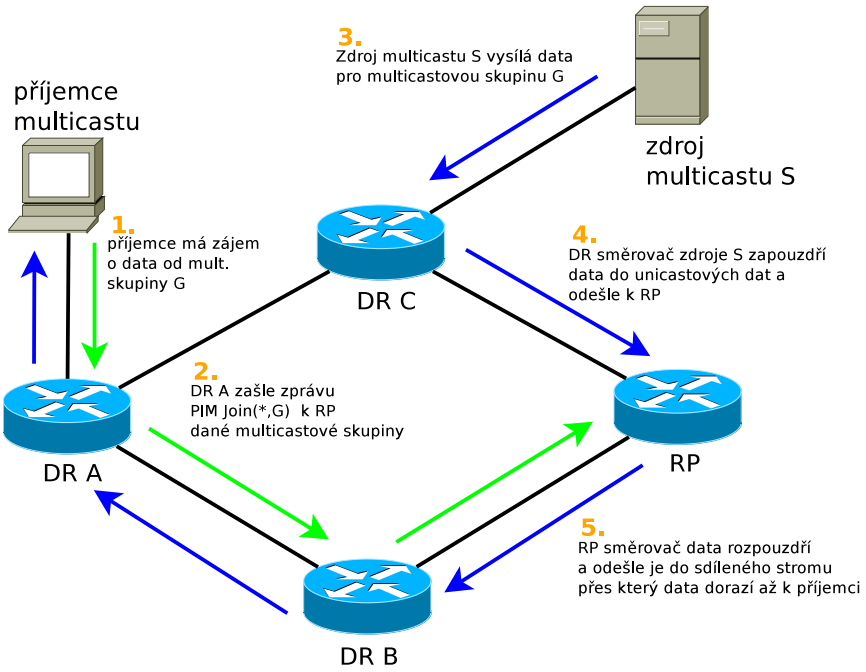
1. **První fáze: RP tree**
   V první fázi příjemci multicastových dat dávají najevo (posílají IGMP či MLD zprávy), že chtějí odebírat nějaká multicastová data. Svoji žádost zasílají vybranému DR směrovači ze svého subnetu, který zašle zprávu *Join(*,G)*, která putuje k RP směrovači pro danou multicastovou skupinu, nebo případně prvnímu směrovači, který je ve stavu *Join(*,G)* pro tuto skupinu. Po připojení několika zájemců o multicast vzniká sdílený strom pro skupinu G s kořenem v RP směrovači.

   Zprávy *Join* jsou zasílány periodicky, dokud má vybraný uzel zájem o multicast. Jakmile všichni příjemci multicastu opustí skupinu (nemají již dále zájem o multicastová data), DR router zašle zprávu *Prune(*,G)* pro zrušení stromu pro multicastovou skupinu G.

   Zdroj multicastových dat je připojen k DR směrovači, který vysílaná data „zabalí“ a podle unicastové IP adresy je přímo zašle RP směrovači, který data rozbalí a provádí

distribuci a replikaci dat ve sdíleném stromě. Proces zapouzdřování paketů je známý jako *Register packets*.

Na konci této fáze proudí data od zdroje zapouzdřená do RP směrovače. Od RP směrovače jsou zasílána klasická nezapouzdřená data sdíleným stromem směrem k příjemcům multicastu. Obrázek 3.1 ilustruje první fázi — *RP tree*.



Obrázek 3.1: Znázornění fáze RP tree

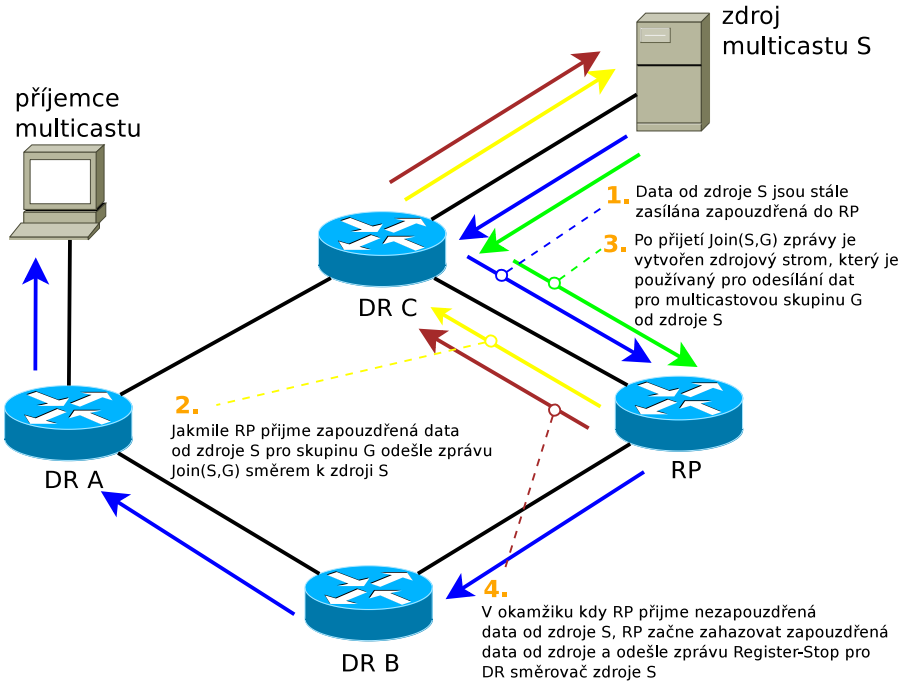
2. **Druhá fáze: Register-Stop**
   Proces registrace je neefektivní jednak kvůli zapouzdřování a odpouzdřování dat, které zbytečně zatěžuje směrovač, a jednak v situaci, kdy je zdroj multicastu blízko příjemcům a data musí putovat k RP směrovači a zpět k příjemcům.

   Jelikož „registrace“ může probíhat donekonečna, RP směrovač je „přepnut“ na standardní přeposílání dat. Jakmile RP směrovač obdrží paket se zapouzdřenými daty ze zdroje S pro skupinu G, vyšle zprávu *Join(S,G)*, která postupně prochází všemi směrovači po cestě ke zdroji dat S a zároveň vytváří zdrojový strom (S,G).

   V tomto okamžiku RP přijímá dvě kopie dat. Jedny, které jsou zapouzdřeny, a druhé, nativně vysílané ze zdroje multicastu S. V tomto bodě RP začne zahazovat kopie se zapouzdřenými daty a zašle zprávu *Register-Stop* DR směrovači, který má na starosti zdroj S. DR směrovač zdroje poté přestane zasílat zapouzdřená data.

   Na konci druhé fáze jsou data zasílána zdrojovým stromem od S k RP, který tato data přijímá a dále je přeposílá sdíleným stromem k odběratelům multicastu. Obrázek 3.2 ilustruje druhou fázi protokolu PIM-SM — *Register-Stop*.

3. **Třetí fáze: Shortest-Path Tree**
   Ve druhé fázi se nám podařilo odstranit zatěžování směrovačů kvůli zapouzdřování a rozpouzdřování dat, ale stále není činnost PIM-SM optimální. Pro mnoho příjemců



Obrázek 3.2: Znázornění fáze Register-Stop

multicastu nejsou totiž jejich cesty ke zdrojům nejkratší při porovnání cest od zdroje k příjemci a cesty od zdroje přes RP a k příjemci.

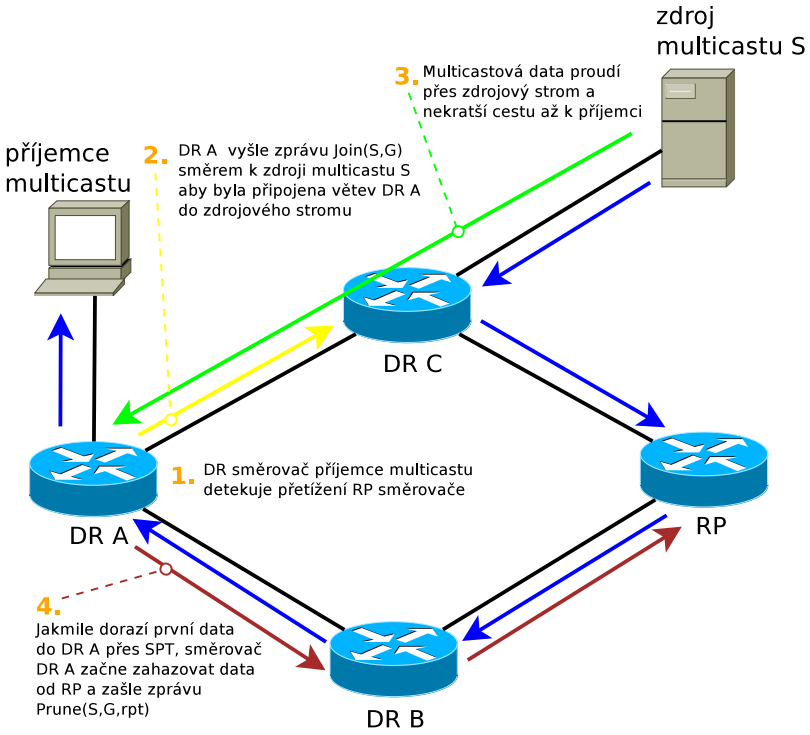
Tento problém řeší směrovač na straně příjemce multicastu. DR směrovač příjemce může iniciovat přenos ze sdíleného stromu, například na základě vytížení RP směrovače, do zdrojového stromu s nejkratší cestou — *Shortest Path Tree (SPT)*. To je zajištěno zprávou *Join(S,G)* směrem ke zdroji multicastových dat. Jakmile zdroj S zprávu přijme, začne vysílat data nejkratší cestou skrz (S,G), dokud nedorazí k příjemcům multicastu. V tomto okamžiku však příjemce multicastu dostává dvě kopie dat, jednu od RPT a drouhou skrz SPT. Jakmile příjemce získá první data od SPT, DR směrovač příjemce zahodí data získaná přes RP strom a zašle zprávu *Prune(S,G,rpt)* směrem k RP. Po tomto kroku příjemce dostává pouze jednu kopii dat přes nejkratší cestu od zdroje S. Obrázek 3.3 ilustruje poslední třetí fázi — *Shortest-Path Tree*.

3.3 Zprávy protokolu PIM Sparse Mode

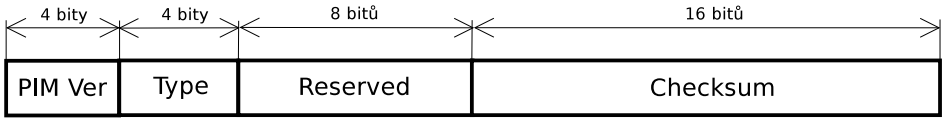
Protokol PIM-SM používá následující zprávy:

- *Hello*
- *PIM Register* a *Register-Stop*
- *PIM Join/Prune*
- *PIM Assert*

Všechny PIM zprávy mají nastaven IP protokol na hodnotu 103 a TTL je nastaveno na hodnotu 1. Hlavička protokolu PIM, která má stejný formát pro všechny typy zpráv, je uvedena na obrázku 3.4.



Obrázek 3.3: Znázornění fáze Shortest-Path Tree



Obrázek 3.4: Formát hlavičky zpráv PIM [7]

Pole **PIM Version** je nastaveno na hodnotu 2. **Type** udává typ pro každou PIM zprávu. Hodnoty Type jsou uvedeny v tabulce 3.1. Hodnota **Reserved** je nastavena na 0 při přenosu nebo je ignorována po přijetí. **Checksum** je standardní IP checksum.

Jestliže je zpráva přijata s nerozeznatelným **PIM Version**, **Type** polem nebo cíl zprávy neodpovídá tabulce 3.1, zpráva musí být zahozena.

Typ zprávy	**Cílová adresa**
0 = Hello	Multicast pro ALL-PIM-ROUTERS
1 = Register	Unicast pro RP
2 = Register-Stop	Unicast pro DR směrovač, který vyslal Register
3 = Join/Prune	Multicast pro ALL-PIM-ROUTERS
4 = Bootstrap	Multicast pro ALL-PIM-ROUTERS
5 = Assert	Multicast pro ALL-PIM-ROUTERS
6 = Graft (pouze PIM-DM)	Unicast pro RPF(S) (uzel rodiče ve stromu)
7 = Graft-Ack (pouze PIM-DM)	Unicast pro uzel potomka ve stromu
8 = Candidate-RP-Advertisment	Unicast pro doménový BSR směrovač

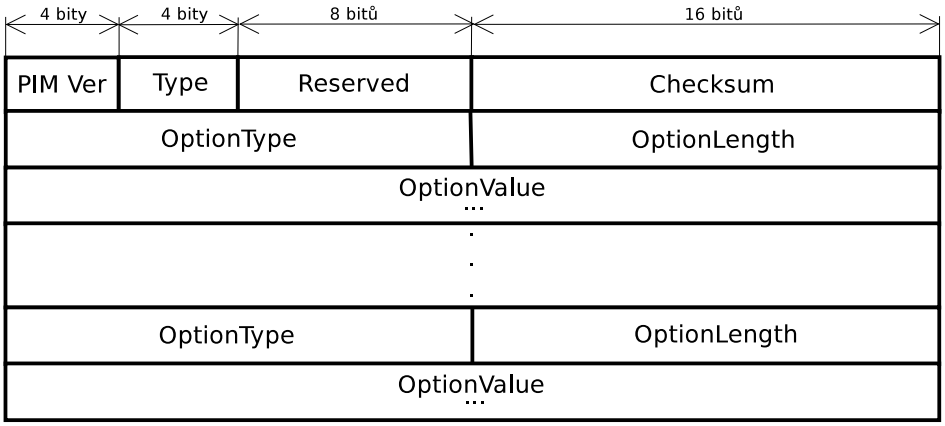
Tabulka 3.1: Přehled hodnot pole Type [7]

3.3.1 Hello zprávy

Jak již bylo výše uvedeno, *Designed router* (DR) je jeden ze směrovačů v daném síťovém segmentu, který slouží jako prostředník mezi zájemci o multicast a ostatními PIM-SM směrovači. Výběr DR je prováděn na všech jeho rozhraních bez ohledu na to, jesti se jedná o LAN, nebo *point-to-point* rozhraní.

Volba DR je určena na základě *Hello* zpráv, které jsou zasílány každým rozhraním směrovače, jenž má nakonfigurováno povolený PIM protokol. PIM Hello zprávy umožňují identifikaci sousedních PIM směrovačů a slouží také k výběru DR směrovače a k nastavení dalších možností. *Hello* zpávy jsou odeslány na všechna PIM aktivní rozhraní směrovače do multicastové skupiny *ALL-PIM-ROUTERS*, která má IP adresu 224.0.0.13 pro IPv4 a ff02::d pro IPv6.

Každé rozhraní má svůj *Hello* časovač, který spouští odesílání *Hello* zpráv. Jestliže je na daném rozhraní povolen PIM a směrovač byl poprvé zapnut, je pro dané rozhraní časovač nastaven na náhodnou hodnotu mezi nulou a `Triggered_Hello_Delay`. Nastavení náhodné hodnoty časovače slouží k zamezení rozeslání velkého počtu *Hello* zpráv v jednom okamžiku v případě, že je zároveň zapnuto více směrovačů. Následně jsou *Hello* zprávy zasílány každý časový úsek v sekundách, který je určen proměnnou `Hello_Period`. Žádný směrovač, který od svého souseda neobdržel *Hello* zprávu, nesmí přijmout žádnou *Join/Prune* zprávu. Jestliže daný směrovač potřebuje zaslat *Join/Prune* nebo *Assert* zprávu, odešle *Hello* zprávu sousedovi bez čekání na vypršení časovače.



Obrázek 3.5: Formát PIM Hello zprávy [7]

Formát *Hello* zprávy je uveden na obrázku 3.5. Skládá se z polí **PIM Version**, **Type**, **Reserved** a **Checksum**, jejichž význam byl uveden výše. *Hello* zpráva dále obsahuje:

- **OptionType** — udává typ pole **OptionValue**
- **OptionLength** — je velikost pole **OptionValue** v bytech
- **OptionValue** — pole o proměnné délce udávající hodnotu parametru

Těchto položek může zpráva obsahovat několik zasebou. Přehled parametrů **OptionType** je uveden v tabulce 3.2.

Parametr **DR_Priority** je používán pro upřednostnění vybraného směrovače administrátorem při volbě DR směrovače. **GEN_ID** je 32-bitová proměnná, která je nastavena

OptionType	Délka	Parametry
1 — Holdtime	2	Holdtime
2 — LAN Prune Delay	4	Propagation_Delay, Override_Interval
3 – 16	–	rezervováno
18	–	zastaralé, nepoužíváno
19 — DR priority	4	DR priority
20 — Generation ID	4	Generation ID
24 — Address list	proměnná	Secondary Address 1 – N

Tabulka 3.2: Přehled hodnot a parametrů OptionType [7]

při začátku přenosu PIM nebo restartu směrovače. Informace z *Hello* zprávy se starým **GEN_ID** jsou nahrazeny novými. Parametr **Lan Prune Delay** je jednoduše řečeno využíván k umožnění přepsání proměnných `Propagation_Delay`[1] a `Override_Interval`[2], které ovlivňují nastavení **Prune-Pending**, **Upstream Join** a **Override Timers**. Poslední parametr **Adress List Option** obsahuje všechny další adresy, které jsou spojeny se zdrojovým portem směrovače, odkud byla odeslána *Hello* zpráva. Jestliže má dané rozhraní více adres (v případě, že bylo nakonfigurováno příkazem: `Router(config-if)# ip address ip-address mask secondary`), musí být tento parametr v *Hello* zprávě uveden. V opačném případě může být vynechán.

Po přijetí *Hello* zprávy na příslušné rozhraní jsou zaznamenány následující informace:

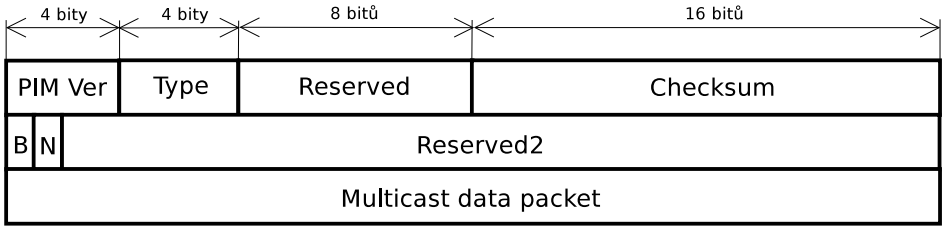
- **neighbor.interface** — rozhraní, ze kterého byla přijata *Hello* zpráva
- **neighbor.primary_ip_address** — IP adresa souseda, z níž byla odeslána *Hello* zpráva
- **neighbor.genid** — **Generation ID** PIM souseda
- **neighbor.dr_priority** — DR prioritní číslo PIM souseda, pokud je v *Hello* zprávě uvedeno
- **neighbor.dr_priority_present** — indikuje zda, je v *Hello* zprávě uveden **DR priority** parametr
- **neighbor.timeout** — doba udávající platnost informací sousedního směrovače

Makra, která jsou použita pro výběr DR směrovače, jsou uvedena v příloze C.1.

3.3.2 PIM Register zpráva

Jak bylo výše uvedeno, DR směrovače starající se o zdroje multicastových dat zapouzdřují multicastové pakety pro příslušné multicastové skupiny a zasílají je směrem k RP směrovači na základě tzv. PIM *Register* zprávy, která vyvolá zasílání zapouzdřených multicastových dat směrem od DR multicastového zdroje k RP směrovači. Tento proces probíhá, dokud DR směrovač neobdrží (S,G) nebo (*,G) *Register-Stop* zprávu od RP. Zapouzdřování paketů budeme dále chápat jako *Register-Tunnel* rozhraní, které je přidáno nebo odebráno z (S,G) `olistu`[3].

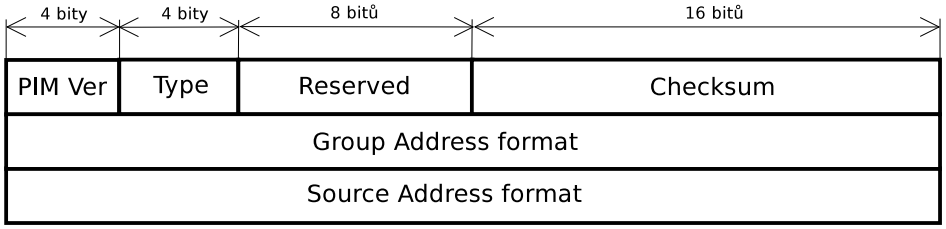
[1] Proměnná udávající zpoždění na lokální lince
[2] Proměnná udávající zpoždění při plánování zpoždění *Join* zprávy
[3] Seznam výstupních rozhraní směrovače



Obrázek 3.6: Formát Register zprávy [7]

Formát *Register* zprávy je zobrazen na obrázku 3.6. Pole **PIM Version**, **Type**, **Reserved**, **Checksum** byly již popsány. Aby se zabránilo zátěži při zapouzdřování je checksum prováděn pouze z prvních 8 bytů včetně PIM hlavičky a datových bloků paketu. Pole **B** značí **Border bit**, který je nastaven na hodnotu 0, jestliže je směrovač DR pro zdroj, který je k němu přímo připojen. Hodnotu 1 nabývá, jestliže je router PMBR[4] pro zdroj v přímo připojeném cloudu. Pole **N** označuje **Null-Register bit**. Tento bit je nastavován směrovačem DR na hodnotu 1 při pokusu o kontakt s RP dříve, než vyprší **Register-Suppresion Timer**[5]. Jinak je nastaven do 0. Pole **Reserved2** je nastaveno při přenosu na hodnotu 0, po přijetí je ignorováno. **Multicast Data Packet** obsahuje originální paket odeslaný zdrojem. Tento paket musí být stejné rodiny IP protokolu jako zapouzdřující paket.

Obrázek 3.7 představuje formát *Register-Stop* zprávy. Unicastová zpráva *Register-Stop* je zasílána z RP směrovače na DR směrovač za účelem přerušení zasílání zpráv typu *Register*.



Obrázek 3.7: Formát Register-Stop zprávy [7]

Mimo jiné *Register-Stop* obsahuje pole **Group Address** udávající adresu skupiny v multicastovém data paketu ve zprávě *Register*. **Source Address** obsahuje adresu zdroje, odkud pochází multicastový paket.

3.3.3 PIM Join/Prune zpráva

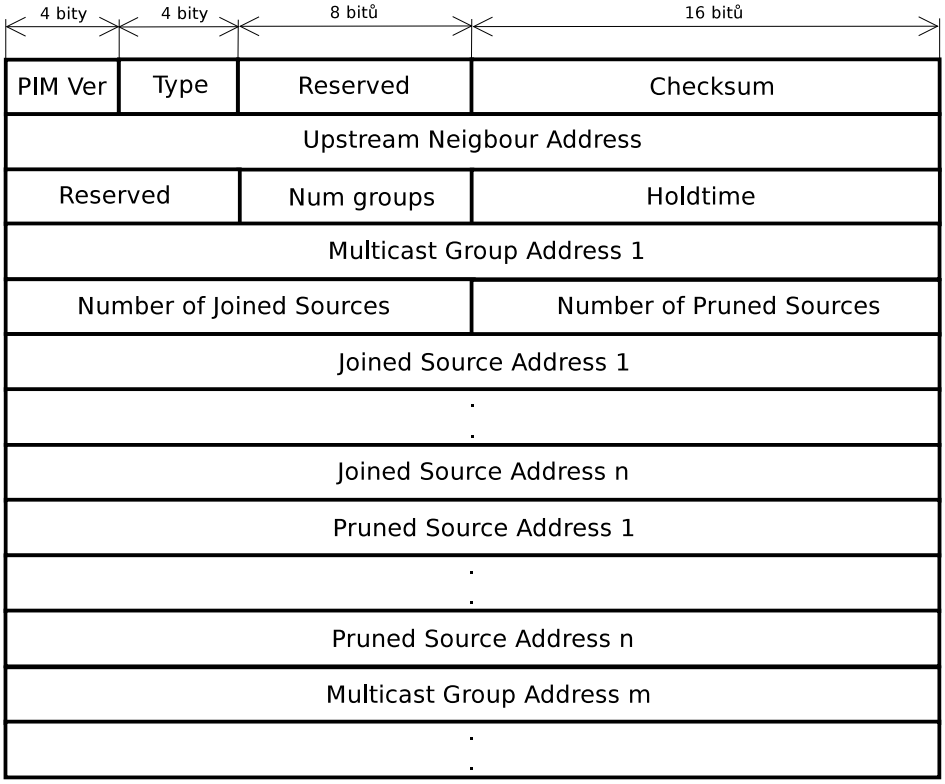
Zprávy *Join/Prune* jsou zasílány směrek k upstream zdrojům a RP směrovačům. *Join* zprávy slouží pro budování sdílených nebo zdrojových stromů. *Prune* zprávy jsou použity pro odříznutí zdrojových stromů, jestliže odběratel multicastu opustí příslušnou skupinu nebo všechny zdroje multicastu již nepoužívají příslušný sdílený strom.

Při zpracování *Join/Prune* zprávy je každý zdroj multicastových dat, který má být připojen či odpojen, pečlivě a efektivně zpracován a výsledek je aplikován na jeden nebo více stavových automatů.

[4] *PIM Multicast Border Router* - směrovač připojující PIM doménu k další multicastové doméně

[5] Čas udávající, za jak dlouho DR přestane zasílat zapouzdřená data do RP po přijetí *Register-Stop* zprávy

Formát zprávy *Join/Prune* (obrázek 3.8) obsahuje, mimo již uvedených, následující parametry: **Upstream Neighbor Address** — IP adresu *upstream směrovače*, **Num Groups** — počet multicastových skupin obsažených ve zprávě, **Holdtime** — počet sekund setrvání příjemce zprávy v *Prune* stavu. *Join/Prune* zpráva dále obsahuje skupiny polí pro každou multicastovou skupinu: **Multicast Group Address** — adresa multicastové skupiny, **Number of Joined Sources** — počet zdrojů, ke kterým se chce odesílatel připojit, obdobně **Number of Pruned Sources**. Následují adresy těchto zdrojů — **Joined Source Address**, **Pruned Source Address**.



Obrázek 3.8: Formát Join/Prune zprávy [7]

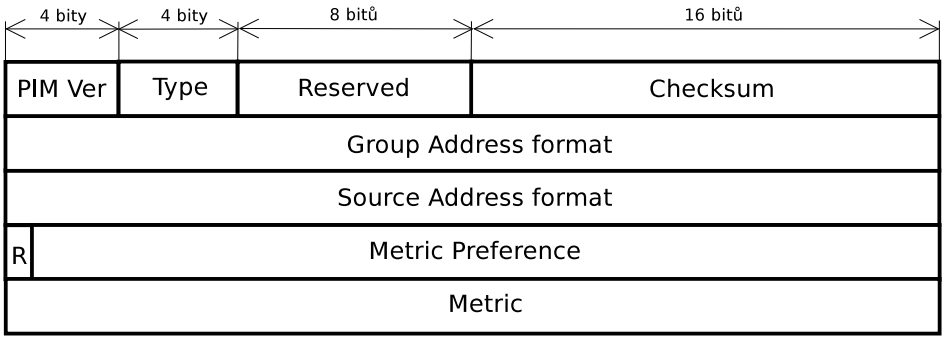
3.3.4 PIM Assert zpráva

Assert zprávy (obrázek 3.9) jsou zasílány v případě, že v LAN má více *upstream* směrovačů platný stav pro odeslání paketu, což by mohlo vést k duplikaci. PIM se tomuto jevu nesnaží explicitně bránit. K zabránění duplikace používá právě *Assert* zprávy, na jejichž základě je proveden výběr *upstream* směrovače pro odeslání paketu. *Assert* zprávy mohou být přijímány také *downstream* směrovači, jejichž příjem vyvolá odeslání *Join/Prune* zprávy směrem k *upstream* směrovači, který byl vybrán jako odesílatel.

PIM *Assert* zprávy by měly být akceptovány pouze od ověřených sousedních směrovačů (proběhla výměna *Hello* zpráv). Navíc mohou být zneužity útočníkem k zamezení šíření multicastu do celého segmentu sítě, a to takovým způsobem, že útočník zašle falešnou *Assert* zprávu (např. s vyšší metrikou) DR směrovači, která donutí DR směrovač odpojit svou větev od sdíleného stromu a zároveň tak přeruší distribuci multicastových dat k příjemci.

Pole PIM **Version**, **Type**, **Reserved**, **Checksum** byly již popsány výše. Zpráva *Assert*

dále obsahuje: **Group Address** — adresa skupiny, pro kterou je řešen konflikt. **Source Address** — zdrojová adresa směrovače, který žádá o vyřešení konfliktu. **R** — **RPT** bit je nastaven do 1 pro *Asert(*,G)* zprávu a do 0, jestliže se jedná o *Assert(S,G)* zprávu. **Metric Preference** — preferenční hodnota nastavená unicastovému směrovacímu protokolu, který poskytuje informaci o cestě do RP směrovače nebo ke zdroji. **Metric** — metrika použitá u unicastového směrovacího protokolu.



Obrázek 3.9: Formát Assert zprávy [7]

3.4 Stavy a konečné automaty protokolu PIM

V této kapitole budou popsány všechny dostupné stavy a konečné automaty protokolu PIM-SM, které jsou důležité pro jeho správnou činnost.

V podkapitole se základními pojmy bylo zmiňováno, že pro ukládání stavů protokolu PIM na jednotlivých směrovačích jsou používány tabulky TIB *(Tree Information Base)*. Tabulky TIB sdružují všechny stavy multicastových distribučních stromů na směrovači a nejčastěji jsou využívány pro vytváření multicastových tabulek. Tabulku TIB můžeme rozdělit do následujících částí:

- **(*,*,RP) stav** — uchovává všechny stromy pro daný RP směrovač
- **(*,G) stav** — udržuje RP strom pro multicastovou skupinu G
- **(S,G) stav** — udržuje zdrojové stromy pro zdroj S a skupinu G
- **(S,G,rpt) stav** — stav, pro který existuje jak zdrojový (S,G), tak sdílený (*,G) strom

Pro každý výše uvedený stav existuje *upstream* a *downstream* konečný automat, navíc pro stavy (S,G) a (*,G) existují *Assert* konečné automaty. DR směrovače mají *Register* konečný automat. PIM-SM tedy pracuje celkem s 11 stavovými automaty. [10]

3.4.1 Register stavový automat

Register stavový automat (obrázek 3.10) slouží ke zpracování *Register* zpráv na DR směrovači. Automat je složen ze 4 stavů:

- **Join (J)** — *register* tunel je připojený.

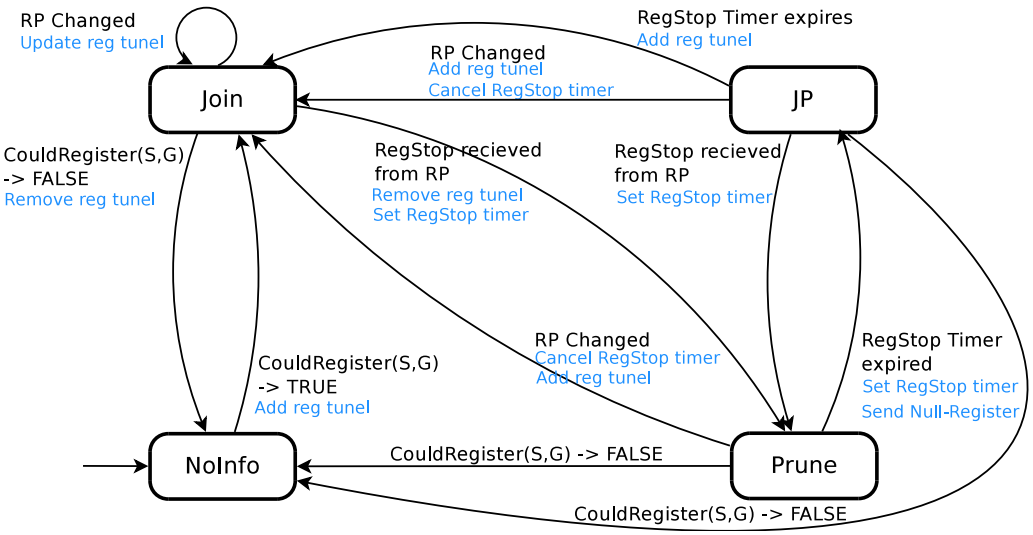
- **Prune (P)** — *register* tunel je odpojený (nastane po *Register-Stop* zprávě).
- **Join-Pending (JP)** — *register* tunel je odpojený, ale DR směrovač se jej snaží připojit.
- **NoInfo (NI)** — žádná informace (při počátečním stavu nebo pokud směrovač není DR).

Pro popis níže uvedených stavových automatů bude udržována následující konvence. Černý popis u přechodu mezi stavy značí nastalou událost, modrý popisuje akci, která je provedena při přechodu mezi stavy. Šipky mezi stavy představují přechody mezi stavy.

Akce, které mohou být vykonány na základě vzniklých událostí v *Register* stavovém automatu:

- **Add Register Tunnel** — vytvoření virtuálního rozhraní *Register* tunel od DR směrovače zdroje do RP(G). *DownstreamJPState (S,G,VI*[6]*)* je nastaven do *Join* stavu přidáním tunnel rozhraní do `immediate_olist(S,G)` a `inherited_olist(S,G)`.
- **Remove Register Tunnel** — *DownstreamJPState (S,G,VI)* je nastaven do stavu *NoInfo* odebráním tunel rozhraní z `immediate_olist(S,G)`[7] a `inherited_olist(S,G)`[8]. Jestliže je stav *DownstreamJPState (S,G,VI)* v *NoInfo* stavu pro všechny (S,G), potom může být virtuální rozhraní smazáno.
- **Update Register Tunnel** — tato akce nastane při změně RP(G), kdy je změněn stav *DownstreamJPState (S,G,VI_old)* do *NoInfo* a *DownstreamJPState (S,G,VI_new)* do *Join* stavu. Virtuální rozhraní *old* je poté smazáno.

Makro `CouldRegister` ze stavového automatu je uvedeno v příloze C.2.



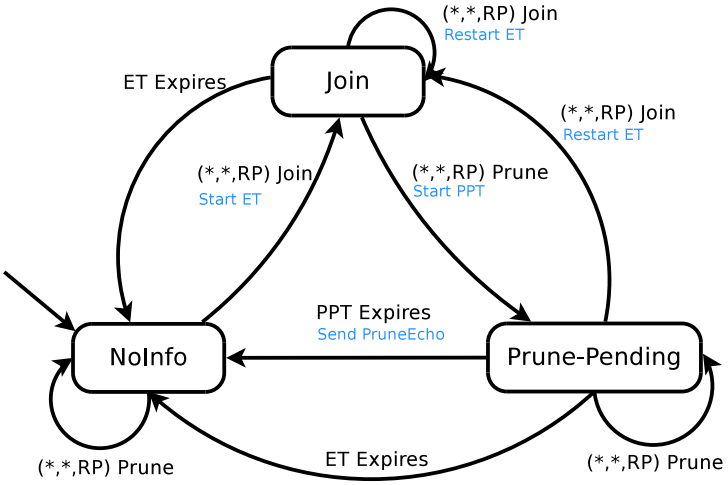
Obrázek 3.10: Stavový automat Register pro každý (S,G) [7]

[6] *Virtual Interface*
[7] *olist* vybudovaný na směrovači, který má právě (S,G) stav
[8] *olist* vybudovaný z různých stavů směrovače

3.4.2 Downstream (*,*,RP) stavový automat

Tento stavový automat slouží pro příjem (*,*,RP) *Join/Prune* zpráv. Jeho grafická podoba je na obrázku 3.11. Automat obsahuje celkem tři stavy a dva časovače:

- **Join (J)** — rozhraní je ve stavu (*,*,RP). Směrovač s tímto stavem odesílá pakety pro jakoukoliv multicastovou skupinu přes daný RP směrovač.
- **Prune-Pending (PP)** — směrovač obdržel zprávu *Prune(*,*,RP)* od svého *downstream* souseda a vyčkává, zda nebude *Prune* zpráva přepsána od jiného downstream směrovače.
- **NoInfo (NI)** — počáteční stav automatu. Rozhraní není ve stavu *Join(*,*,RP)* a není spuštěn žádný časovač.
- **ExpiryTimer (ET)** — časovač je resetován po každém přijetí *Join(*,*,RP)* zprávy. Po vypršení časovače je stavový automat uveden do stavu *NoInfo*.
- **Prune-Pending Timer (PPT)** — časovač je nastaven při přijetí *Prune(*,*,RP)*. Po vypršení časovače je rozhraní uvedeno do stavu *NoInfo*.



Obrázek 3.11: Downstream (*,*,RP) stavový automat [7]

Ze stavu *NoInfo* **Downstream (*,*,RP)** stavového automatu lze přejít do stavu *Join* přijetím zprávay *Join(*,*,RP)*, přičemž je spuštěn ET časovač. Přijetím zprávy *Prune(*,*,RP)* v *NoInfo* stavu je zachován tento stav.

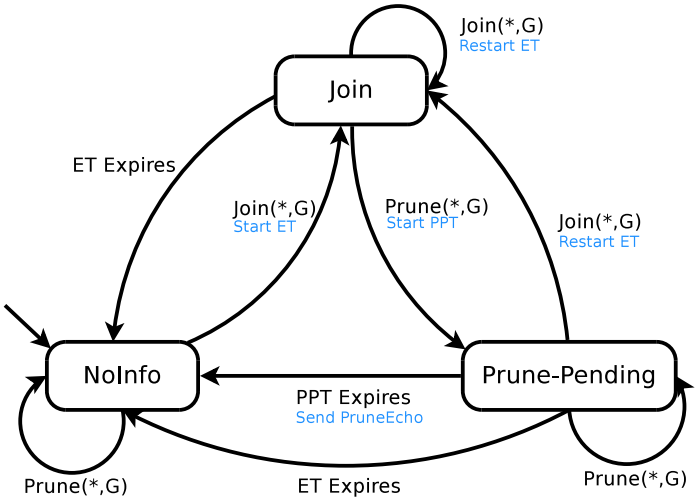
Jestliže se stavový automat nachází v *Join* stavu a vyprší ET časovač, je proveden přechod z *Join* stavu do *NoInfo* stavu. Při přijetí zprávy *Join(*,*,RP)* je resetován ET. Přijetí zprávy *Prune(*,*,RP)* v *Join* stavu vyvolá přechod do stavu *Prune-Pending*.

V *Prune-Pending* stavu se mohou provést následující přechody. Při obdržení zprávy *Join(*,*,RP)* je proveden přechod do *Join* stavu a následně je resetován ET časovač. Přijetím zprávy *Prune(*,*,RP)* zůstává stavový automat v témže stavu. Jestliže ve stavu *Prune-Pending* vyprší časovač PPT, je proveden přechod do *NoInfo* stavu a zaslána zpráva *Pruned-Echo(*,*,RP)*. Vyprší-li časovač ET, je proveden pouze přechod do *NoInfo* stavu.

3.4.3 Downstream (*,G) stavový automat

Následující automat (obrázek 3.12) slouží pro příjem *Join(*,G)* zpráv. Po přijetí této zprávy musí být nejdříve zkontrolováno, zda RP směrovač ve zprávě odpovídá RP(G) směrovači. Jestliže neodpovídá, zpráva *Join(*,G)* by měla být zahozena. Pokud směrovač nemá žádnou informaci o RP, může *Join(*,G)* přijmout.

- **Join (J)** — rozhraní je ve stavu *Join(*,G)*. Směrovač s tímto stavem odesílá z daného rozhraní pakety pro multicastovou skupinu G.
- **Prune-Pending (PP)** — směrovač obdržel zprávu *Prune(*,G)* od svého *downstream* souseda a vyčkává, zda nebude *Prune* zpráva přepsána od jiného *downstream* směrovače.
- **NoInfo (NI)** — počáteční stav automatu. Rozhraní není ve stavu *Join(*,G)* a zároveň není spuštěn žádný časovač.
- **ExpiryTimer (ET)** — časovač je resetován po každém přijetí *Join(*,G)* zprávy. Po vypršení časovače je rozhraní uvedeno do stavu *NoInfo*.
- **Prune-Pending Timer (PPT)** — nastaví časovač při přijetí *Prune(*,G)*. Po vypršení časovače je rozhraní uvedeno do stavu *NoInfo*.



Obrázek 3.12: Downstream (*,G) stavový automat [7]

Z *NoInfo* stavu **Donwstream (*,G)** konečného automatu lze přejít přijetím *Join(*,G)* zprávy do stavu *Join*. Následně je spuštěn ET časovač. Přijetím zprávy *Prune(*,G)* v *NoInfo* stavu je tento stav zachován.

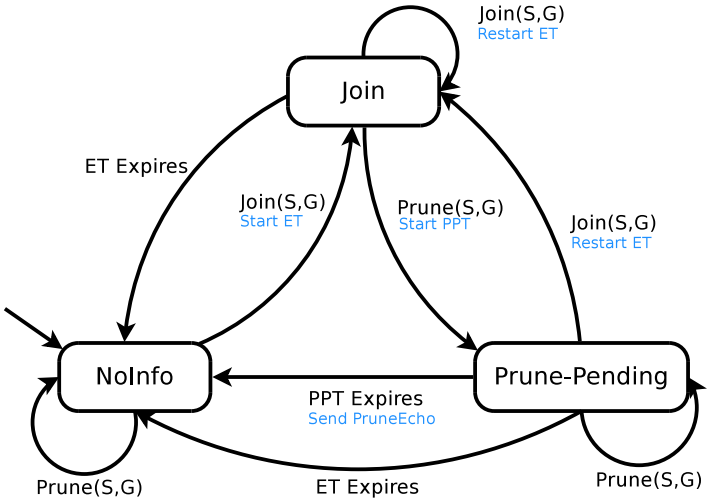
Ze stavu *Join* mohou být provedeny tři přechody. Při přijetí zprávy *Join(*,G)* je ponechán stejný stav a časovač ET je restartován. Po přijetí zprávy *Prune(*,G)* je proveden přechod do stavu *Prune-Pending* a spuštění PPT časovače. Jestliže stavový automat zůstane v *Join* stavu a vyprší ET časovač, je proveden přechod do stavu *NoInfo*.

Prune-Pending stav má celkem čtyři proveditelné přechody. Po přijetí *Join(*,G)* zprávy je proveden přechod do *Join* stavu a zároveň je restartován ET časovač. Přijetím zprávy

Prune(,G)* se stav automatu nezmění a zůstává v *Prune-Pending* stavu. Vypršením časovače PPT ve stavu *Prune-Pending* dojde k odeslání *Prune-Echo(*,G)*. Po vypršení časovače ET je proveden přechod do stavu *NoInfo*.

3.4.4 Downstream (S,G) stavový automat

Stavový automat **Downstream (S,G)** slouží pro příjem *Join/Prune(S,G)* zpráv a je obdobou automatu **Downstream (*,G)**. **Downstream (S,G)** má identické stavy: *Join*, *Prune-Pending*, *NoInfo*. Jediným rozdílem je typ zpráv – *Join/Prune(S,G)*, na jejichž základě je prováděn přechod do jiného stavu. Grafická podoba automatu je na obrázku 3.13.



Obrázek 3.13: Downstream (S,G) stavový automat [7]

3.4.5 Downstream (S,G,rpt) stavový automat

Stavový automat **Downstream (S,G,rpt)** (obrázek 3.14) pro příjem *Join/Prune* zpráv má celkem pět stavů a dva, nám již známé, časovače:

- **Prune (P)** — rozhraní je ve stavu *Prune(S,G,rpt)*, ve kterém směrovač neposílá pakety ze zdroje S pro skupinu G.
- **Prune-Pending (PP)** — směrovač obdržel zprávu *Prune(S,G,rpt)* od svého *downstream* souseda a vyčkává, zda nebude *Prune* zpráva přepsána od jiného *downstream* směrovače.
- **PruneTmp (P')** — přechodný stav se stejným chováním při přeposílání zpráv jako *Prune* stav. Jestliže zpráva *Join/Prune* obsahuje *Join(*,G)*, je proveden přechod právě do stavu *PruneTmp (P')*. Jestliže ve zprávě *Join* narazíme na *Prune(S,G,rpt)*, je obnoven *Prune* stav. Pokud *Join* neobsahuje *Prune(S,G,rpt)*, je proveden přechod do *NoInfo* stavu.
- **Prune-Pending-Tmp (PP')** — přechodný stav, který je podobný stavu *PruneTmp*. Liší se v tom, že je spojován se stavem *Prune-Pending* a ne se stavem *Prune*.

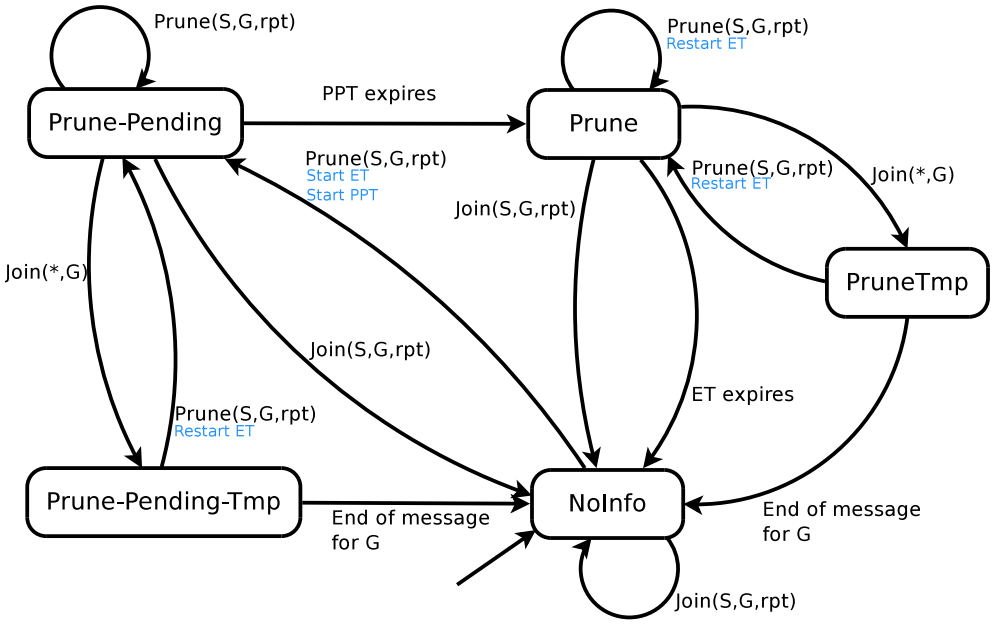
- **NoInfo (NI)** — počáteční stav automatu. Rozhraní není ve stavu *Prune(S,G,rpt)* a není spuštěn žádný časovač.
- **ExpiryTimer (ET)** — časovač je resetován po každém přijetí *Join(*,G)* zprávy. Po vypršení časovače je rozhraní uvedeno do stavu *NoInfo*.
- **Prune-Pending Timer (PPT)** — nastaví časovač při přijetí *Prune(S,G,rpt)*. Po vypršení časovače je rozhraní uvedeno do stavu *NoInfo*.

Popis stavového automatu začneme ve stavu *NoInfo*. Přijetím zprávy *Prune(S,G,rpt)* je proveden přechod do *Prune-Pending* stavu. ET časovač je nastaven na hodnotu `HoldTime` získanou z *Join/Prune* zprávy.

Jestliže má směrovač na rozhraní více sousedních směrovačů, je nastaven PPT časovač na hodnotu `J/P_Override_Interval(I)`. V opačném případě je proměnná `J/P_Override_Interval(I)` nastavena do 0, tzn. že nastane okamžitá expirace časovače.

Z *Prune-Pending* stavu přijetím zprávy *Join(*,G)* přejde automat do stavu *Prune-Pending-Tmp* po dobu zpracování zbytku přijaté zprávy. Zpráva *Join(S,G,rpt)* vyvolá ve stavu *Prune-Pending* přechod do *NoInfo* stavu a zastaví běžící časovače. Vypršením PPT časovače je stavový automat na rozhraní `I` nastaven do *Prune* stavu.

Přijetím zprávy *Join(*,G)* ve stavu *Prune* je proveden přechod do stavu *PruneTmp* po dobu zpracování zbytku přijaté zprávy. Jestliže je v *Prune* stavu přijata zpráva *Join(S,G,rpt)*, stavový automat provede přechod do *NoInfo* stavu a jsou zastaveny běžící časovače ET a PPT. Přijetím zprávy *Prune(S,G,rpt)* je spuštěn ET časovač a nastaven na větší z hodnot: `Holdtime` nebo jeho aktuální hodnoty.



Obrázek 3.14: Downstream (S,G,rpt) stavový automat [7]

Z dočasného stavu *Prune-Pending-Tmp* lze provést dva přechody. První přechod je proveden při přijetí zprávy *Prune(S,G,rpt)*, a to do stavu *Prune-Pending*. Zároveň je restartován časovač ET a nastaven na větší z hodnot `Holdtime` nebo jeho aktuální hodnoty. Jestliže při procházení složené zprávy *Join/Prune* bylo dosaženo jejího konce, je proveden přechod do *NoInfo* stavu a běh ET a PPT časovačů je zrušen.

Obdobné akce a přechody jsou prováděny ze stavu *PruneTmp*. Po přijetí zprávy *Prune(S,G,rpt)* je místo přechodu do *Prune-Pending* stavu proveden přechod do *Prune* stavu. Při dosažení konce složené *Join/Prune* zprávy je proveden přechod do *NoInfo* stavu a je zrušen běh časovače ET.

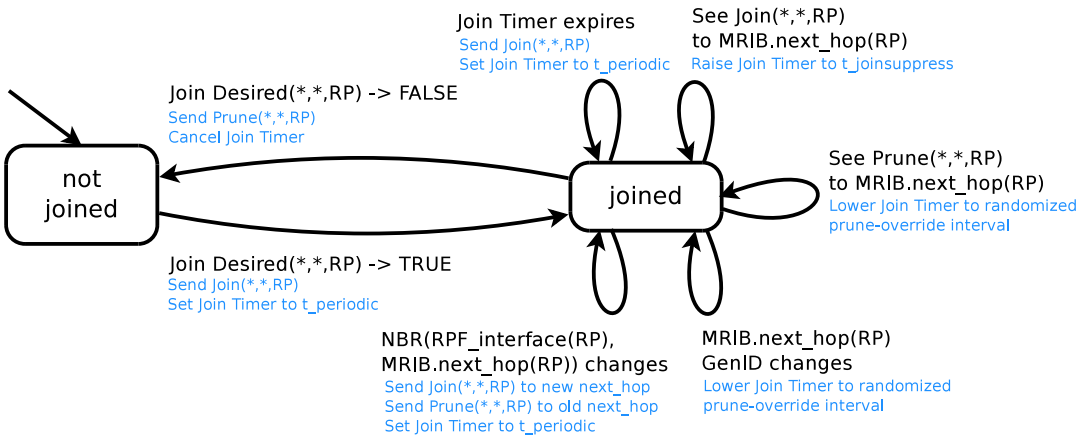
3.4.6 Upstream (*,*,RP) stavový automat

Upstream (*,*,RP) stavový automat (obrázek 3.15) udržuje *Join* stav z *downstream* PIM směrovače a na základě dalších informací od sousedů rozhoduje, zda směrovač zašle zprávu *Join(*,*,RP)* směrem k RP směrovači nebo zprávu *Prune(*,*,RP)* pro *downstream* směrovač.

V případě, že má být zpráva *Join(*,*,RP)* poslána směrovačem dále, musí směrovač zkontrolovat zprávy na jeho *upstream* rozhraní od ostatních směrovačů. Stavový automat **Upstream (*,*,RP)** má dva stavy:

- **Not Joined** — *downstream* stavový automat a lokální informace od sousedních směrovačů neindikují, že by měla být zaslána *Join(*,*,RP)* zpráva směrem k danému RP směrovači. Počáteční stav konečného automatu.
- **Joined** — *downstream* stavový automat a lokální informace od sousedních směrovačů indikují, že by měla být zaslána *Join(*,*,RP)* zpráva směrem k danému RP směrovači.

Časovač `Join Timer` (JT) je používán pro načasování odesílání *Join(*,*,RP)* zpráv směrem k RP směrovači.



Obrázek 3.15: Upstream (*,*,RP) stavový automat [7]

Funkce `MRIB.next_hop(S)` z výše uvedeného stavového automatu vrací adresu *next-hop* sousedícího směrovače směrem ke zdroji S. Funkce `NBR(I,A)` mapuje IP adresu A přímo připojeného sousedního směrovače na rozhraní I. Makro `JoinDesired` použité ve stavovém automatu je uvedeno v příloze C.4.

Nachází-li se stavový automat ve stavu *NotJoined*, lze provést jediný přechod, a to v případě, že makro `JoinDesired` vrátí hodnotu True. Znamená to, že byl změněn stav (*,*,RP) *downstream* směrovače přidáním nejméně jedno rozhraní do `immediate_olist(*,*,RP)`. Při přechodu je odeslána *Join(*,*,RP)* zpráva směrem k příslušnému *upstream* směrovači a je nastaven JT časovač na hodnotu `t_periodic`.

Ve stavu *Joined* může makro `JoinDesired` vrátit hodnotu false a přejít tak do stavu *NotJoined* v případě, že se změnil stav (*,*,RP) tak, že v `immediate_olist(*,*,RP)` není žádné rozhraní. Při přechodu je také zaslána zpráva *Prune(*,*,RP)* k příslušnému *upstream* směrovači a je zrušen běh JT časovače. Jestliže ve stavu *Joined* vyprší JT časovač, je odeslána zpráva *Join(*,*,RP)* k příslušnému *upstream* směrovači a je restartován časovač JT.

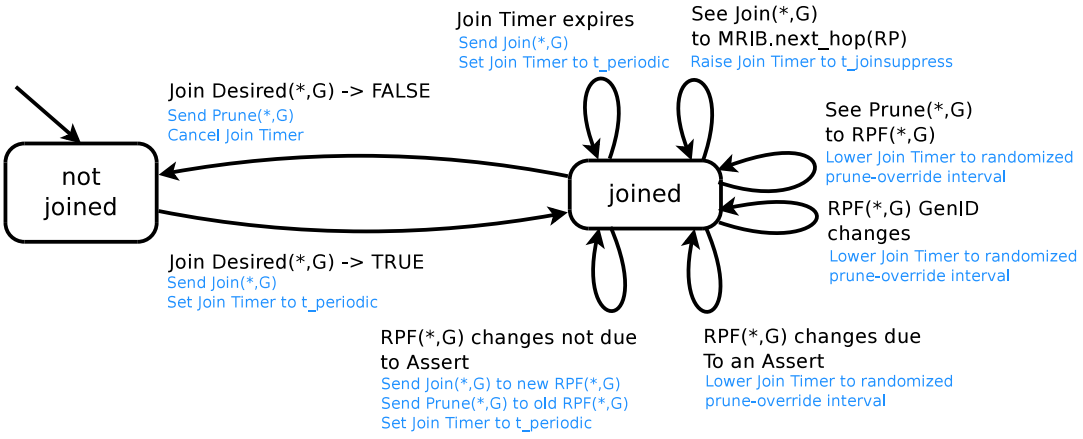
Další akce a přechody vedou zpět do stavu *Joined*. Jejich podrobný popis je uveden v [7].

3.4.7 Upstream (*,G) stavový automat

Tento stavový automat udržuje *Join* stav *downstream* směrovače, ze kterého je dále určeno, zda bude propagována zpráva *Join(*,G)* směrem k RP směrovači. Jestliže se směrovač rozhodl propagovat *Join(*,G)* zprávu, musí kontrolovat *uptstream* rozhraní od jiných směrovačů, jelikož mohou změnit jeho další činnost. Stavový automat **Upstream (*,G)** (obrázek 3.16) má dva stavy:

- **Not Joined** — stav indikující, že směrovač nepotřebuje připojit RP strom pro skupinu G. Počáteční stav konečného automatu.
- **Joined** — stav indikuje, že směrovač požaduje připojení RP stromu pro skupinu G.

Časovač `Join Timer` (JT) je používán pro načasování odesílání *Join(*,G)* zpráv směrem k RP, RPF(*,G) směrovači.



Obrázek 3.16: Upstream (*,G) stavový automat [7]

Makro `JoinDesired`, které využívá stavový automat je uvedeno v příloze C.4.

Základní popis automatu začneme ve stavu *NotJoined*. Jestliže se změnil *downstream* stav pro (*,G), tzn. nejméně jedno rozhraní je v `immediate_olist(*,G)`, `JoinDesired(*,G)` makro vrátí hodnotu true a bude proveden přechod do stavu *Joined*. Dále se odešle *Join(*,G)* zpráva k příslušnému *upstream* sousedovi a nastaví se JT časovač na hodnotu `t_periodic`.

Ze stavu *Joined* lze provést přechod do stavu *NotJoined* v případě, že makro *JoinDesired* vrátí hodnotu false. A to tehdy, když byl změněn *downstream* stav pro (*,G), což znamená, že žádné rozhraní není v `immediate_olist(*,G)`. Při přechodu je zaslána zpráva *Prune(*,G)* a zastaven časovač JT.

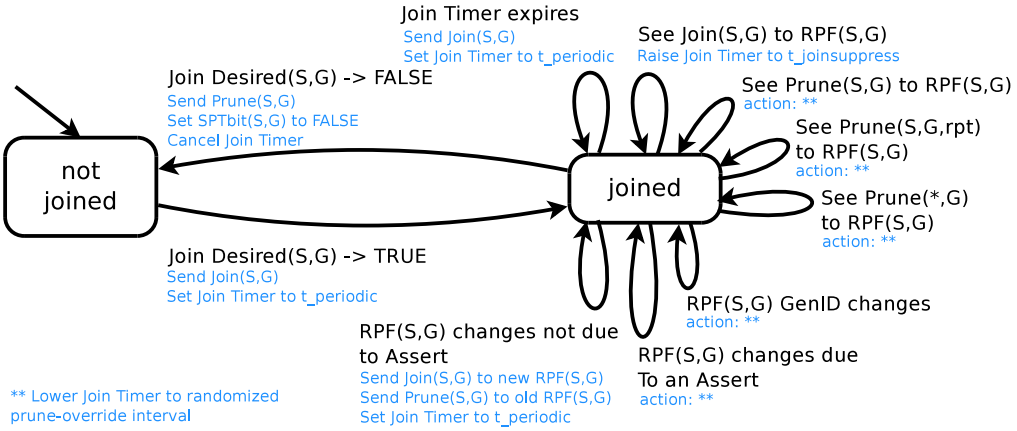
Obdobně jako u předchozího automatu jsou další akce a přechody podrobně popsány v [7].

3.4.8 Upstream (S,G) stavový automat

Stavový automat **Upstream (S,G)** (obrázek 3.17) pracuje obdobně jako předchozí automaty s tím rozdílem, že své přechody a akce mezi stavy provádí na základě zpráv, které mají určený zdroj a cílovou multicastovou skupinu, tedy (S,G). Stavový automat opět udržuje stav *Join* z *downstream* směrovače, na jehož základě se rozhoduje, zda směrovač zašle *Join(S,G)* zprávu směrem ke zdroji. **Upstream (S,G)** má také dva stavy:

- **Not Joined** - stav indikující, že se směrovač nepotřebuje připojit k *Shortest-Path Tree* pro (S,G). Počáteční stav konečného automatu.
- **Joined** - stav indikuje, že router požaduje připojení k *Shortest-Path Tree* pro (S,G).

Časovač `Join Timer` (JT) je používán pro časování odesílání *Join(S,G)* zpráv směrem ke zdroji S.



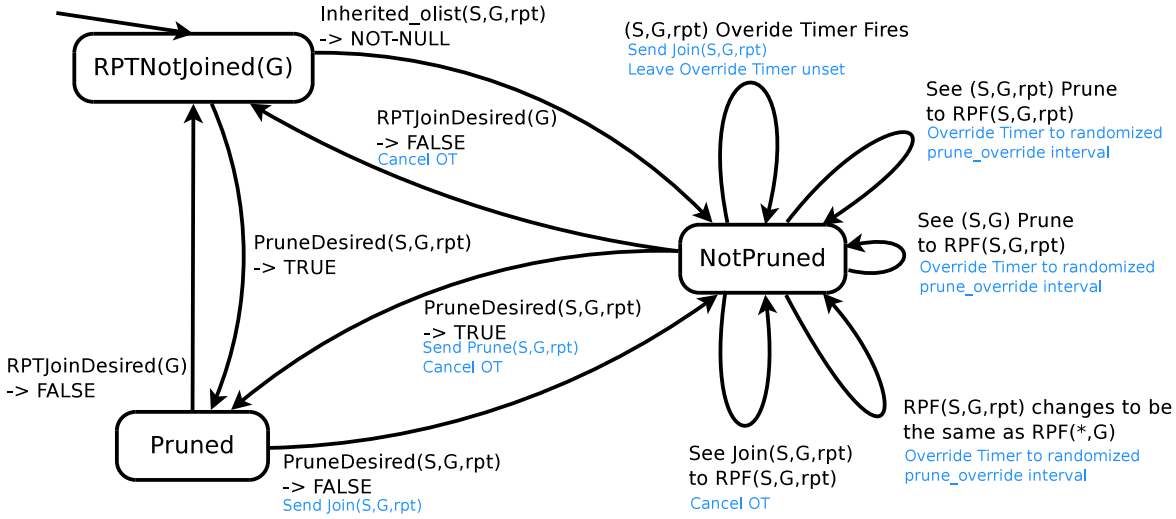
Obrázek 3.17: Upstream (S,G) stavový automat [7]

Pseudokód makra `JoinDesired` je uveden v příloze C.4.

3.4.9 Upstream (S,G,rpt) stavový automat

Stavový automat **Upstream (S,G,rpt)** (obrázek 3.18) je používán u směrovačů, které mají na svém rozhraní vytvořen zdrojový strom (S,G). Tento směrovač si přeje odpojení zdroje S od RP. Automat obsahuje celkem tři stavy a jeden časovač:

- **Pruned(S,G,rpt)** — směrovač je v jednom z následujících *Join* stavů (*,G) nebo (*,*,RP(G)) a zároveň ve stavu *Pruned(S,G,rpt)*.
- **NotPruned(S,G,rpt)** — směrovač je v jednom z následujících *Join* stavů (*,G) nebo (*,*,RP(G)) a současně není ve stavu *Pruned(S,G,rpt)*.
- **RPTNotJoined(G)** — směrovač nemá ani jeden z (*,G) a (*,*,RP(G)) *Join* stavů. Počáteční stav konečného automatu.



Obrázek 3.18: Upstream (S,G,rpt) stavový automat [7]

Časovač `Override Timer` (OT) je používán pro zpoždění odeslání *Join(S,G,rpt)* zprávy, aby se zabránilo kolizím mezi těmito zprávami.

Stavový automat obsahuje makra `RPTJoinDesired(G)` a `PruneDesired(S,G,rpt)` s pseudokódy uvedenými v příloze C.5.

`PruneDesired(S,G,rpt)` vrací hodnotu true, právě když ji vrátí makro `RPTJoinDesired(G)`. To nastane v případě, že směrovač nemá žádné odchozí rozhraní pro zdroj S nebo má směrovač aktivní (S,G) stav, ale platí `RPF(*,G) != RPF(S,G)`.

Ze stavu *Not Pruned* lze provést přechody při událostech: *See Join(S,G,rpt) to RPF(S,G,rpt)*, při které směrovač detekoval cizí zprávu *Join(S,G,rpt)*. Reakcí na událost je zrušení běhu OT časovače. Při události *See Prune(S,G,rpt) to RPF(S,G,rpt)* je proveden přechod zpět do *Not Pruned* stavu a nastavení OT časovače na náhodnou hodnotu *t_override*. Stejná akce a přechod je proveden při události *See Prune(S,G) to RPF(S,G,rpt)*. Událost *(S,G,rpt) prune Override Timer expires* může nastat také v *NotJoined* stavu. Reakcí na událost je zaslání zprávy *Join(S,G,rpt)* do *RPF(S,G,rpt)*. Další událostí v *NotPruned* stavu je *RPF(S,G,rpt) changes to become equal to RPF(*,G)*, při které je provedeno nastavení OT časovače.

Jednou z událostí, které mohou nastat v *RPTNotJoined (G)* stavu, je *inheritedy_olist(S,G,rpt) becomes non-NULL*. Tato událost nastane v případě, že směrovač má připojen RP strom a chce přijímat data ze zdroje S, což způsobí přechod do *NotPruned (S,G,rpt)* stavu.

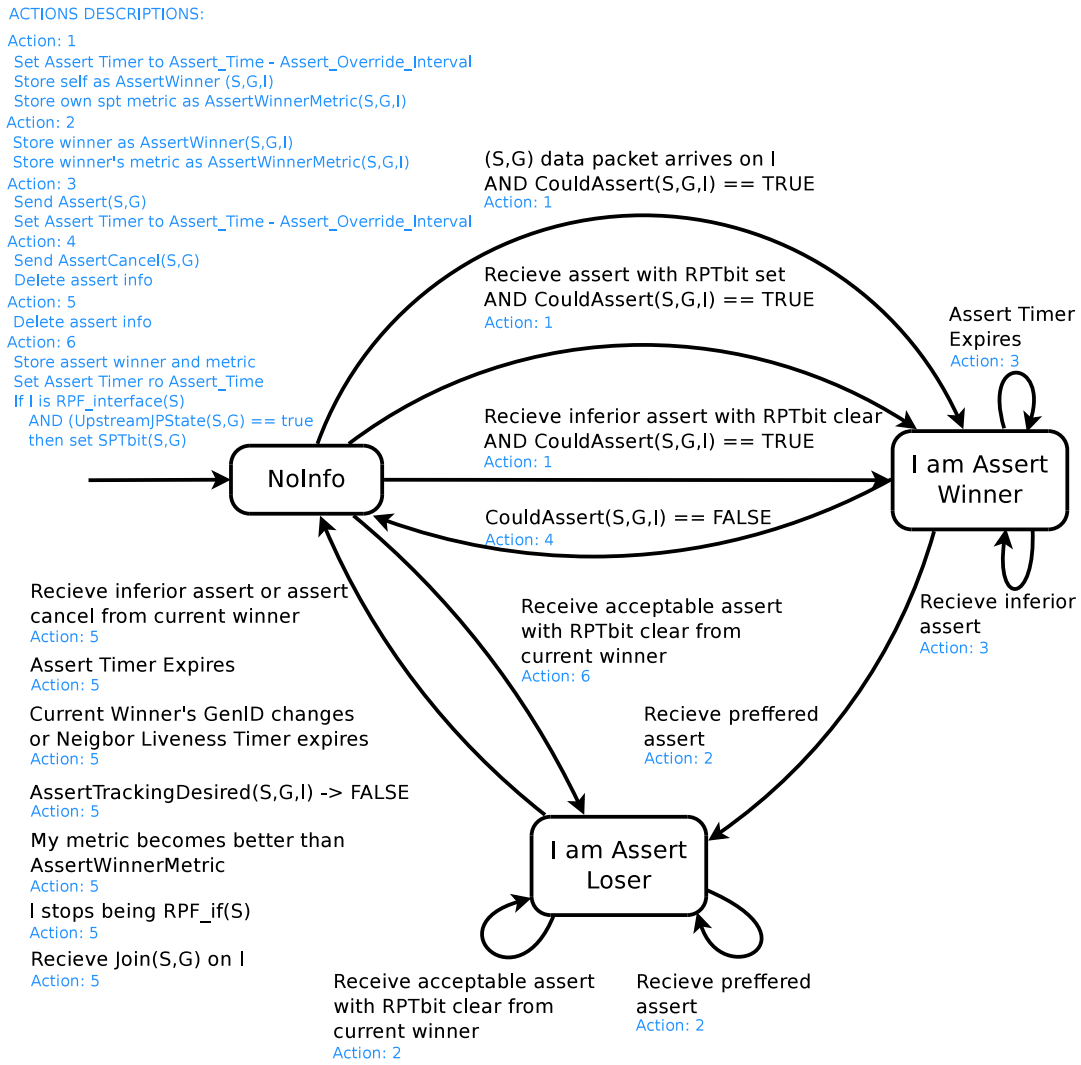
3.4.10 Assert (S,G) stavový automat

Stavový automat **Assert (S,G)** (obrázek 3.19) pro rozhraní I má celkem tři stavy:

- **NoInfo(NI)** — vybraný směrovač nemá *Assert(S,G)* stav na rozhraní I.
- **I am Assert Winner (W)** — vybraný směrovač byl vybrán *Assert* zprávami pro odesílání multicastových dat ze zdroje S pro skupinu G přes rozhraní I (vyhrál Assert volbu).

- **I am Assert Loser (L)** — vybraný směrovač prohrál volbu směrovače pro odesílání dat ze zdroje S pro skupinu G. Tento směrovač nesmí zasílat multicastová data ze zdroje S pro skupinu G.

V **Assert (S,G)** stavovém automatu je používán `Assert Timer` (AT), který slouží při jeho vypršení k opětovnému zaslání *Assert* zprávy od směrovače, který prohrál assert volbu, k vítězi volby.



Obrázek 3.19: Assert(S,G) stavový automat [7]

U následujících událostí je používáno porovnávání metriky. Pro její ohodnocení je používáno makro `my_assert_metric(S,G,I)`. Pseudokódy související s tímto makrem jsou uvedeny v [7].

Jelikož je popis následujících dvou automatů obsáhlý, zavedeme pro přehlednost následující konvenci. Nejdříve bude uveden název popisovaného stavu automatu následovaný událostí. Každá událost je doplněna o přesnější popis a akci, která je provedena jako reakce na tuto událost.

Stav automatu: *I am Assert Winner*

- **Událost:** Vypršení `Assert` časovače
  - **Popis:** vypršel `Assert(S,G)` časovač
  - **Akce:** zaslání zprávy *Assert(S,G)* a restart časovače AT
- **Událost:** Přijetí *Assert* zprávy s horší metrikou
  - **Popis:** přijetí zprávy *Assert(S,G)* nebo *(*,G)* s metrikou pro zdroj S, která je horší než vlastní nastavená
  - **Akce:** zaslání zprávy *Assert(S,G)* a restart časovače AT
- **Událost:** Přijetí *Assert* zprávy s lepší metrikou
  - **Popis:** přijata zpráva *Assert(S,G)* s metrikou lepší než vlastní nastavená
  - **Akce:** provedení akce 2 (viz. popis akcí u obrázku 3.19)
- **Událost:** Makro `Could Assert(S,G,I)` vrátí logickou hodnotu false
  - **Popis:** hodnota makra je změněna na hodnotu false
  - **Akce:** při přechodu do stavu *NoInfo* je odeslána zpráva *AssertCancell(S,G)*

  Pseudokód makra `Could Assert(S,G,I)` je uveden v příloze C.3.

Stav automatu: *I am Assert Loser*

- **Událost:** Přijetí *Assert* zprávy s lepší metrikou
  - **Popis:** byla přijata zpráva *Assert* s lepší metrikou než má daný směrovač nastaven
  - **Akce:** je proveden přechod do stejného stavu a vykonána akce 2
- **Událost:** Přijetí *Assert* zprávy s lepší metrikou a vynulovaným RPT bitem od aktuálního vítěze Assert volby
  - **Popis:** byla přijata *Assert* zpráva od vítěze Assert volby s lepší metrikou než je aktuální na směrovači
  - **Akce:** je proveden přechod do stejného stavu a vykonána akce 2
- **Událost:** Vypršení `Assert` časovače
  - **Popis:** časovač AT (S,G) vypršel
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5 (smazání Assert informace)
- **Událost:** Rozhraní I přestalo být používáno jako `RPF_interface(S)`
  - **Popis:** rozhraní I přestalo být pužíváno jako RPF rozhraní pro zdroj S
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

- **Událost:** Makro `AssertTrackingDesired(S,G,I)` vrátí logickou hodnotu false
  - **Popis:** uvedené makro změnilo svou hodnotu na false
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5
- **Událost:** Přijetí *Assert* zprávy s lepší metrikou nebo *Assert Cancel* od aktuálního vítěze Assert volby
  - **Popis:** byla přijata *Assert* zpráva od vítěze Assert volby s horší metrikou než je aktuální na směrovači
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5
- **Událost:** Došlo k změně **GenID** nebo vypršel NLT časovač
  - **Popis:** vypršel časovač vítěze Assert volby NLT (`Neigbor Liveness Timer`) nebo byla přijata *Hello* zpráva od vítěze s jiným **GenID**
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5
- **Událost:** *Assert* metrika směrovče se stala lepší než metrika vítěze Assert volby
  - **Popis:** metrika `my_assert_metric(S,G,I)` byla změněna tak, že je nyní lepší než metrika Assert vítěze volby
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5
- **Událost:** Přijetí zprávy *Join(S,G)* na rozhraní I
  - **Popis:** byla přijata zpráva *Join(S,G)* s nastaveným parametrem **Upstream Neigbour Assres field** na IP adresu rozhraní I
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

Stav automatu: *NoInfo*

- **Událost:** Přijetí zprávy *Assert* s horší metrikou a nenastaveným RPT bitem, zároveň makro `CouldAssert(S,G,I)` vrací logickou hodnotu true
  - **Popis:** přijata zpráva *Assert* s nenastaveným RPT bitem a nižší metrikou než vlastní nastavená
  - **Akce:** je proveden přechod do stavu *I am Assert Winner* a vykonána akce 1
- **Událost:** Přijetí zprávy *Assert* s nenastaveným RPT bitem, zároveň makro `CouldAssert(S,G,I)` vrací logickou hodnotu true
  - **Popis:** přijata zpráva *Assert(S,G)* na rozhraní I s nenastaveným RPT bitem, makro `CouldAssert(S,G,I)` vrací hodnotu true pouze pokud jsme v (S,G) stavu
  - **Akce:** je proveden přechod do stavu *I am Assert Winner* a je provedena akce 1
- **Událost:** Přijetí (S,G) datového paketu na rozhraní I, zároveň makro `CouldAssert(S,G,I)` vrací hodnotu true
  - **Popis:** datový paket (S,G) byl přijat na *downstream* rozhraní, které je uvedeno v *outgoing interface list*

  - **Akce:** je proveden přechod do stavu *I am Assert Winner* a je provedena akce 1

- **Událost:** Přijetí zprávy Assert s lepší metrikou a nenastaveným RPT bitem, zároveň makro `AssertTrackingDesired(S,G,I)` vrací logickou hodnotu true

  - **Popis:** pseudokód makra `AssertTrackingDesired(S,G,I)` je uveden v [7]
  - **Akce:** na základě makra `AssertTrackingDesired(S,G,I)` a *Assert* zprávy s nenastaveným RPT bitem je nebo není proveden přechod do stavu *I am Assert Looser* a vykonána akce 6

3.4.11 Assert (*,G) stavový automat

Stavový automat **Assert(*,G)** má stejné tři stavy jako předchozí automat. Grafická podoba automatu je na obrázku 3.20. Assert(*,G) stavový automat používá zároveň `Assert Timer`.

- **NoInfo (NI)** — rozhraní I není nastaveno do *Assert(*,G)* stavu.
- **I am Assert Winner (W)** — daný směrovač vyhrál Assert volbu na rozhraní I a jeho úkolem je přeposílat datový provoz na rozhraní I s výjimkou dat, pro které je nastaven stav (S,G) *I am Assert Looser*.
- **I am Assert Loser (L)** — směrovač prohrál Assert volbu a ztratil stav *Assert(*,G)*. Směrovač přeposílá pouze datový provoz od zdrojů, které mají stav *I am Assert Winner(S,G)*.

Stav automatu: *NoInfo*
Ve stavu *NoInfo* mohou nastat následující události, avšak pouze v případě, že automat **Assert(S,G)** zůstává ve stavu *NoInfo*.

- **Událost:** Přijetí zprávy *Assert* s horší metrikou a nastaveným RPT bitem, zároveň makro `CouldAssert(*,G,I)` vrací logickou hodnotu true

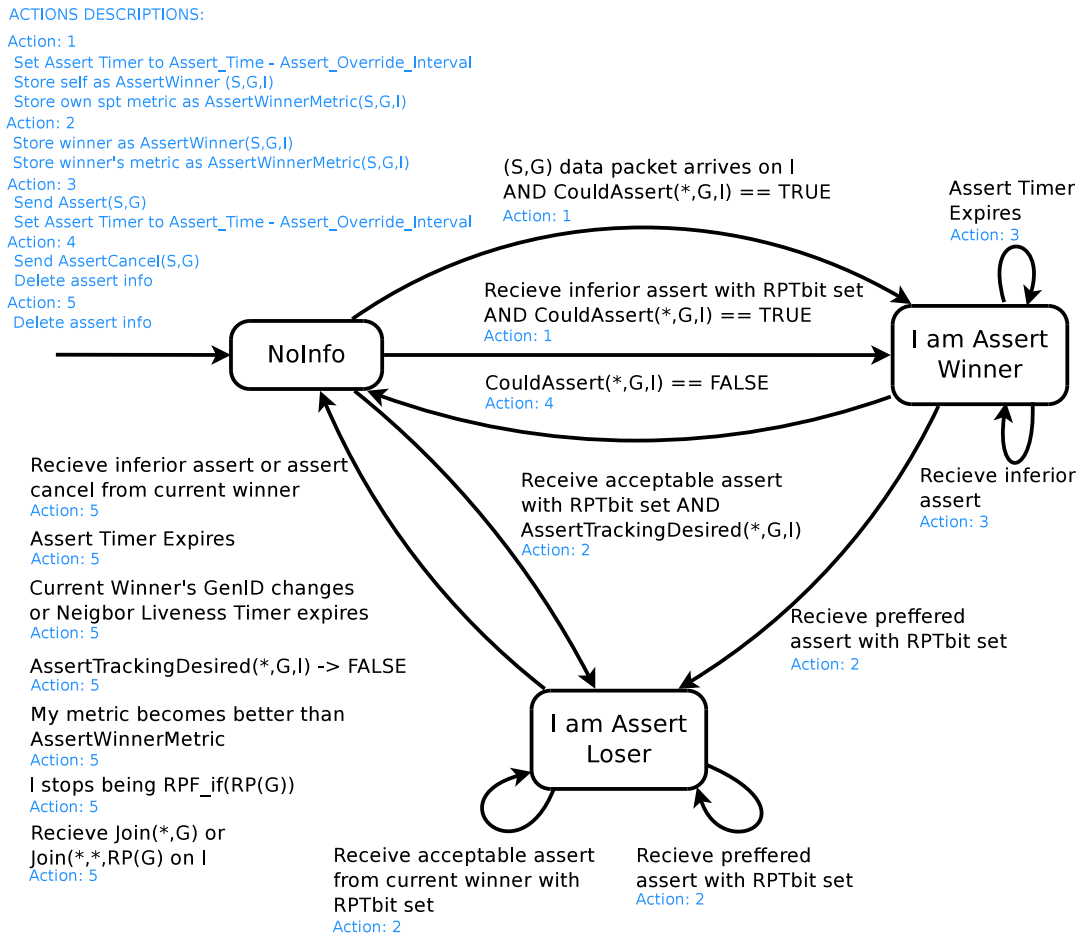
  - **Popis:** byla přijata *Assert(*,G)* zpráva a makro `CouldAssert(*,G,I)` vrací hodnotu true
  - **Akce:** reakcí na tuto událost je přechod do stavu *I am Assert Winner* a provedení akce 1

- **Událost:** Datový paket adresovaný pro skupinu G byl přijat na rozhraní I a zároveň makro `CouldAssert(*,G,I)` vrací logickou hodnotu true

  - **Popis:** přijetí paketu s cílovou adresou G na *downstream* rozhraní směrovače, které je v *(*,G) outgoing interface* seznamu
  - **Akce:** reakcí na tuto událost je přechod do stavu *I am Assert Winner* a provedení akce 1

- **Událost:** Přijetí zprávy *Assert* s lepší metrikou a nastaveným RPT bitem, zároveň makro `AssertTrackingDesired(*,G,I)` vrací logickou hodnotu true

  - **Popis:** byla přijata *Assert* zpráva s lepší metrikou než vlastní nastavená
  - **Akce:** je proveden přechod do stavu *I am Assert Loser* a provedena akce 2



Obrázek 3.20: Assert(*,G) stavový automat [7]

Stav automatu: *I am Assert Winner*
Ve stavu *I am Assert Winner* mohou nastat následující události, avšak pouze v případě, že automat **Assert(S,G)** zůstává ve stavu *NoInfo*.

- **Událost:** Přijetí *Assert* zprávy s horší metrikou
  - **Popis:** byla přijata *Assert(*,G)* zpráva s metrikou horší, než vlastní nastavená
  - **Akce:** automat zůstává ve stejném stavu, při přechodu je znovu zaslána *Assert(*,G)* zpráva a restartován AT (akce 3)
- **Událost:** Přijetí *Assert* zprávy s lepší metrikou
  - **Popis:** byla přijata *Assert(*,G)* zpráva s metrikou lepší, než vlastní nastavená
  - **Akce:** při této události je proveden přechod do stavu *I am Assert Loser* a provedena akce 2

Ve stavu *I am Assert Winner* mohou nastat další události, které nezávisí na výše uvedené podmínce:

- **Událost:** Vypršení časovače `Assert Timer`

  - **Popis:** vypršel časovač AT (*,G)
  - **Akce:** je provedena akce 3, stav automatu zůstává stejný

- **Událost:** Makro `CouldAssert(*,G,I)` vrátí logickou hodnotu false
  - **Popis:** makro `CouldAssert(*,G,I)` změnilo svou hodnotu na false
  - **Akce:** v automatu je proveden přechod do stavu *NoInfo* a je vykonána akce 4

Stav automatu: *I am Assert Loser*
Následující přechody jsou provedeny pouze tehdy, když byl předchozí stav **Assert(S,G)** automatu v *NoInfo*.

- **Událost:** Přijetí *Assert* zprávy s lepší metrikou a nastaveným RPT bitem
  - **Popis:** byla přijata zpráva *Assert(*,G)* s lepší metrikou než má vítěz volby
  - **Akce:** s přechodem je provedena akce 2

- **Událost:** Přijetí *Assert* zprávy s lepší metrikou od aktuálního vítěze Assert volby s nastaveným RPT
  - **Popis:** přijata *Assert(*,G)* zpráva od Assert vítěze s lepší metrikou než je naše aktuální pro skupinu G
  - **Akce:** opět je provedena akce 2

- **Událost:** Přijetí *Assert* zprávy s horší metrikou nebo *Assert Cancel* od aktuálního vítěze Assert volby
  - **Popis:** přijata zpráva *Assert* od vítěze assert volby, avšak s horší metrikou
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5 (smazání přijaté Assert zprávy)

Dále mohou být provedeny, bez předchozí podmínky, následující přechody:

- **Událost:** Vypršení časovače `Assert Timer`
  - **Popis:** časovač AT (*,G) vypršel
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

- **Událost:** Došlo k změně **GenID** nebo vypršel NLT časovač
  - **Popis:** vypršel časovač NLT nebo byla přijata *Hello* zpráva od vítěze Assert volby s jiným **GenID**
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

- **Událost:** Makro `AssertTrackingDesired(*,G,I)` vrací logickou hodnotu false
  - **Popis:** výše uvedené makro vrátí hodnotu false
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

- **Událost:** *Assert* metrika směrovče se stala lepší než metrika vítěze Assert volby

  - **Popis:** byla změněna směrovací metrika `rpt_assert_metric(G,I)` tak, že metrika pro (*,G) je lepší než aktuální metrika vítěze Assert volby
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

- **Událost:** Rozhraní I přestalo být používáno jako `RPF_interface(S)`
  - **Popis:** rozhraní I přestalo být používáno jako RPF rozhraní pro RP(G), metrika (*,G) je lepší než aktuální metrika vítěze Assert volby
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

- **Událost:** Přijetí zprávy *Join(*,G)* nebo *Join(*,*,RP(G))* na rozhraní I
  - **Popis:** byla přijata zpráva *Join(*,G)* nebo *Join(*,*,RP(G))*, která má nastavenou **Upstream Neighbor Address** na primární IP adresu rozhraní I
  - **Akce:** je proveden přechod do stavu *NoInfo* a vykonána akce 5

3.5 Volba RP směrovače

Jak již bylo výše napsáno, PIM *Sparse Mode* by nebyl schopný své správné činnosti bez zvoleného RP směrovače pro každou multicastovou skupinu. Ostatní směrovače musí znát IP adresu RP. Tato adresa může být získána statickou konfigurací, nebo automaticky pomocí *bootstrap* mechanismu.

Jak je zřejmé, statické konfigurace nejsou vhodné, a to zejména v rozsáhlých sítích. Proto je často využíváno *Bootstrap Router* mechanismu (BSR) pro dynamický výběr RP směrovače. Celý mechanismus popisuje RFC 5059 [5].

3.5.1 Bootstrap mechanismus

Před samotnou konfigurací PIM-SM jsou v síti vybrány a nakonfigurovány některé ze směrovačů jako *Candidate-RP* a *Candidate-BSR*. V první řadě je zvolen BSR směrovač, který následně ze všech RP kandidátů vybere jeden. Vybraný RP směrovač poté rozesílá informace o mapování multicastových skupin na RP směrovač. Proces volby RP směrovače [10]:

1. **Výběr BSR směrovače** — BSR kandidáti si navzájem v PIM doméně rozesílají své *Bootstrap* zprávy s prioritou. Směrovač, který přijal zprávu s vyšší prioritou, než je jeho vlastní, se dál nebude účastnit BSR volby. Na konci první fáze zůstane pouze jeden kandidát, který je prohlášen za BSR směrovač a tuto informaci vyšle do sítě *Bootstrap* zprávou.

2. **Ohlášení Candidate-RP** — RP kandidáti zasílají zprávu *Candidate RP Advertisment* zvolenému BSR. Ve zprávě je uvedena priorita RP a seznam multicastových skupin, pro které daný směrovač kandiduje.

3. **Vytvoření RP množiny** — BSR směrovač vybere množinu RP kandidátů. Tato množina by neměla být příliš velká, aby ji nebylo náročné rozdistribuovat. Zároveň by množina neměla být příliš malá, aby nebyla zátěž některých RP příliš vysoká.

4. **Šíření RP množiny** — BSR periodicky zasílá *Bootstrap* zprávy do PIM domény s vybranou RP množinou.

Kapitola 4

Simulační nástroj OMNeT++ a knihovna INET

V následujících kapitolách si stručně popíšeme simulační nástroj OMNet++ a knihovnu INET. Detailnější informace lze získat z [6], [11] a [12].

4.1 OMNeT++

OMNeT++ je rozšiřovatelný, modulární a objektový simulační framework, vhodný zejména pro vytváření simulací v počítačových sítích. V OMNeT++ můžeme modelovat drátové a bezdrátové sítě, síťové protokoly, multiprocesory a distribuované systémy. Dále také umožňuje validaci hardwarových architektur a vyhodnocování komplexních softwarových systémů. Obecně zvládá modelování a simulaci diskrétních systémů, které mohou komunikovat zasíláním zpráv.

OMNeT++ je založen na komponentách naprogramovaných v objektovém jazyce C++ a na vlastním jazyce NED (*Network Description*) pro popis struktury simulačních modelů. Simulátor poskytuje grafická rozhraní a byl otestován na operačních systémech Linux, Mac a Windows. Pro akademické účely a pro nekomerční použití lze OMNeT++ využívat bezplatně.

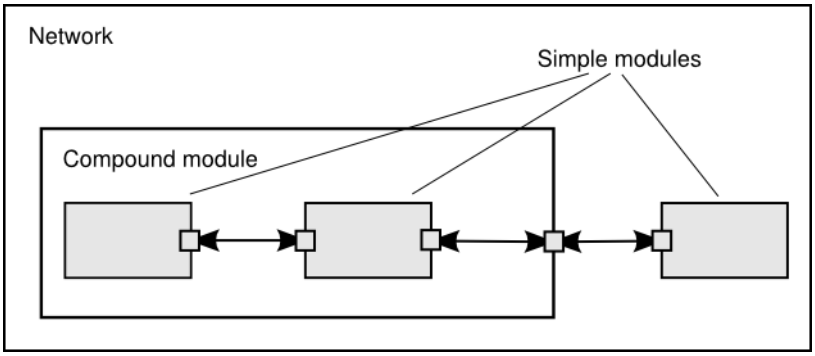
4.1.1 Modelování v OMNeT++

Simulační model se skládá z modulů, které spolu vzájemně komunikují zasíláním zpráv. Moduly mohou být jednoduché s funkcionalitou napsanou v jazyku C++ a složené, jejichž součástí jsou jiné složené nebo jednoduché moduly (viz. obrázek 4.1).

Moduly komunikují zprávami, které jsou většinou zasílány přes brány (vstupní/výstupní) reprezentující rozhraní modulů. Tzv. linky spojují moduly mezi sebou. V rámci jednoho modulu lze spojit jeho podmoduly nebo spojit podmodul s rodičovským modulem.

4.2 INET

INET framework je opensource simulační knihovna pro prostředí OMNeT++. Obsahuje modely pro rodinu protokolů TCP/IP (IPv4, IPv6, TCP, UDP, SCTP), dále pak pro MPLS, LDP, PPP, Ethernet a 802.11. INET framework také podporuje simulace mobilních a bezdrátových sítí. [12]



Obrázek 4.1: Složené a jednoduché moduly [11]

INET využívá stejného konceptu jako OMNeT++, a to modulů komunikujících na základě zpráv. Každý protokol je reprezentován svým modulem, jejichž rozhraní je popsáno NED jazykem. Implementace modulů je opět v C++. Složené moduly lze skládat z jednoduchých pomocí NED.

4.2.1 Současné možnosti multicastu v prostředí OMNeT++ a INET

OMNeT++ nejnovější verze 4.2.2 neposkytuje žádnou podporu pro multicast. Knihovna INET od nové verze 2.0 obsahuje směrovací tabulku, do které je možné vkládat multicastové směrovací záznamy.

Dále byla do knihovny INET přidána podpora IGMPv2. Co se týče podpory multicastových směrovacích protokolů, INET ani OMNeT++ žádnou nemají.

V rámci diplomové práce [10] vznikla implementace multicastového směrovacího protokolu PIM-DM a dále byly vytvořeny potřebné moduly a struktury pro multicastové směrování, avšak nyní již pro staré verze OMNeT++ a INET.

Kapitola 5

Podpora multicastu na Cisco zařízeních

Následující kapitola popisuje konfiguraci multicastového směrování na Cisco zařízeních. Zaměříme se především na protokol PIM-SM a s ním spojené konfigurace. V podkapitole 5.3 budou popsány typy zpráv, které můžeme odchytit při multicastovém provozu.

5.1 Konfigurace PIM-SM

Aby bylo multicastové směrování funkční, musí být kromě konfigurace multicastového směrovacího protokolu nakonfigurována také základní konektivita v síti, tzn. IP adresace a unicastové směrování.

5.1.1 Povolení multicastového směrování

Jestliže máme nastavenu základní konektivitu, povolíme na Cisco směrovači multicastové směrování, které je defaultně vypnuté:

```
Router(config)# ip multicast routing
```

5.1.2 Nastavení módu směrování

Po povolení multicastového směrování, nastavíme jeho mód. V našem případě nás bude zajímat *Sparse Mode*. Cisco podporuje i další módy: *Dense Mode*, *Source Specific Mode* a *Bidirectional Mode*.

```
Router(config)# interface interface-type/interface-number
Router(config-if)# ip pim { sparse-mode | dense-mode | sparse-dense}
```

5.1.3 Nastavení spt-threshold

Pokud bychom nechtěli, aby PIM-SM prováděl optimalizaci v podobě hledání nejkratší možné cesty pro příjem multicastových dat od zdroje, je nutné provést níže uvedené nastavení na všech směrovačích. Defaultně je optimalizace prováděna.

```
Router(config)# ip pim spt-threshold {kbps | infinity}
                  [group-list access-list]
```

Volbou `kbps` v příkazu `ip pim spt-threshold` specifikujeme práh, po jehož překročení se příjemce multicastu napojí přímo nejkratší cestou do zdrojového stromu. Jestliže chceme zabránit této optimalizaci, použijeme přepínač `infinity`. `Group-list` určí, pro které multicastové skupiny je tento práh nastaven.

5.1.4 Konfigurace RP směrovače

Jak již bylo uvedeno v kapitole věnující se PIM-SM, tento multicastový směrovací protokol vyžaduje nakonfigurovaný RP směrovač. Bez něj by nebyl PIM-SM schopný korektně fungovat. Konfiguraci RP směrovače můžeme zvolit statickou nebo dynamickou.

Statická konfigurace

Statická konfigurace je jednodušší než dynamická. Nevýhodou je, že musíme provést nastavení RP směrovače v celé PIM doméně. Pokud PIM doména obsahuje mnoho směrovačů, může být taková konfigurace zdlouhavá a navíc se nám může stát, že některý ze směrovačů nakonfigurujeme jinak než ostatní.

```
Router(config)# ip pim rp-address ip-address [access-list] [override]
```

Dynamická konfigurace

Po spuštění multicastového směrování Cisco směrovače implicitně používají PIM verze 2. Pro přepnutí verze můžeme použít příkaz:

```
Router(config-if)# ip pim version { 1 | 2 }
```

Dynamická konfigurace směrovače u protokolu PIM verze 1 se liší od verze 2. My si ukážeme dynamickou konfiguraci protokolu PIM verze 2, kterou Cisco směrovače používají ve výchozím nastavení.

Dynamická konfigurace využívá tzv. *Bootstrap* mechanismu, kde jsou některé směrovače nakonfigurovány jako *Candidate-RP* a *Candidate-BSR*. Z BSR kandidátů je vybrán jeden směrovač, který následně ze všech RP kandidátů určí RP směrovač.
Konfigurace BSR kandidáta:

```
Router(config)# ip pim bsr-candidate interface-type/interface-number
                  hash-mask-length [priority]
```

Parametr `hash-mask-length` slouží k přiřazení RP směrovače pro multicastovou skupinu. Parametrem `priority` je číslo, které se používá k upřednostnění daného směrovače. Směrovač s nejvyšším číslem je zvolen jako BSR. Jakmile máme nakonfigurovaného BSR kandidáta, přejdeme ke konfiguraci RP kandidáta:

```
Router(config)# ip pim rp-candidate interface-type/interface-number ttl
                  [group-list access-list-number]
```

5.2 Ověření konfigurace

Pro kontrolu konfigurace můžeme použít několik `show` příkazů. Následující příkaz zobrazí multicastovou směrovací tabulku.

```
Router# show ip mroute [group-address] [source] [summary] [count]
          [active kbps]
```

Dalšími volbami můžeme specifikovat záznamy pro kontkrétní multicastovou skupinu či pro konkrétní multicastový zdroj. Přepínačem `summary` získáme zkrácený výpis, `count` vypíše statistiky a `active` zobrazí aktivní multicastové zdroje.

K zobrazení informací o PIM sousedech daného směrovače použijeme následující příkazy:

```
Router# show ip pim neighbor [interface-type interface-number]
Router# show ip pim interface [type number] [count]
```

Oba příkazy jsou si podobné. Můžeme specifikovat, z jakého rozhraní má být informace o PIM sousedovi zobrazena, a navíc lze u druhého příkazu `ip pim interface` přepínačem zobrazit statistiku odeslaných a přijatých paketů přes dané rozhraní.

Informace o aktivních RP směrovačích zobrazíme příkazem:

```
Router# show ip pim rp [mapping] [rp-address]
```

Získání informací o daném RP směrovači lze dosáhnout zadáním RP adresy směrovače. Přepínačem `mapping` vypíšeme pro zadanou IP adresu multicastové skupiny daný RP směrovač. Podobně se chová příkaz:

```
Router# show ip pim rp-hash {group-address | group-name}
```

Jestliže specifikujeme IP adresu multicastové skupiny nebo její název, je výše uvedeným příkazem určen RP směrovač, který byl vybrán např. při volbě RP.

Příkazem `mrinfo` vypíšeme multicastové sousedy směrovače, na kterém byl příkaz zadán.

```
Router# mrinfo [host-address]
```

Volbou `host-address` určíme, pro který směrovač v síti má být příkaz vykonán. Pokud ji neuvedeme, vykoná příkaz „lokálně“.

Informace a statistiky o multicastovém provozu na daném rozhraní směrovače získáme příkazem:

```
Router# show ip multicast [type number]
```

5.3 Multicastové zprávy

Následující podkapitola je zaměřena na multicastové zprávy zasílané směrovači při činnosti protokolu PIM-SM. Nejdříve bude popsána komunikace mezi multicastovým zdrojem dat a RP směrovačem a následně mezi příjemcem multicastu a RP směrovačem. V tabulce 5.1 je uveden krátký přehled vybraných PIM zpráv.

Jakmile začne zdroj multicastu vysílat data, DR směrovač, ke kterému je tento zdroj připojen, tato data zabalí do multicastové zprávy typu *Register* a unicastově je zašle RP směrovači. Odpovědí RP směrovače na zprávu *Register* je *Register-Stop*, která je zaslána DR

Typ	Cílová IP adresa	Popis
Hello	224.0.0.13	Může určovat priority DR směrovačů a obsahuje další parametry multicastového směrování.
Register	RP směrovače	Unicastově vysílaná DR směrovačem zdroje, obsahuje zapouzdřený multicastový paket.
Register-Stop	registrující	Unicastově vysílaná RP směrovačem směrem k směrovači, který poslal Register zprávu.
Join/Prune	224.0.0.13	Obsahuje informace o připojované/odpojované multicastové skupině G, popřípadě i zdroji S.

Tabulka 5.1: Přehled vybraných PIM-SM multicastových zpráv [13]

směrovači, který zaslal *Register* zprávu. Následně RP zasílá zprávu *Join(S,G)* pro všechny PIM směrovače na adresu 224.0.0.13. Tato zpráva oznamuje vytvoření zdrojového stromu (S,G).

Jestliže má nějaký počítač zájem o multicastová data, přihlásí se o odběr dat zprávou *IGMP Membership Report* svému DR směrovači. Tento směrovač následně zašle zprávu *Join(*,G)* všem PIM směrovačům. V okamžiku, kdy příjemce multicastu ukončí odběr dat, vyšle zprávu *IGMP Leave group*. DR směrovač příjemce následně odešle zprávu *Prune(*,G)* k RP směrovači.

V předchozím textu jsme neuvažovali třetí fázi činnosti protokolu PIM-SM — *Shortest-Path Tree*. Data tedy putují od zdroje multicastu k RP a od něj k příjemci, což nemusí být nejoptimálnější cesta. V případě, že DR směrovač příjemce překročí práh datového provozu (defaultně je práh nastaven na 0), dojde k přímému připojení multicastového zdroje pomocí zprávy *Join(S,G)*. Data od RP již není třeba přijímat, a proto DR směrovač příjemce odřízne svou větev od RP zprávou *Prune(S,G)* s nastaveným RP bitem.

Dále jsou mezi směrovači na rozhraních, která mají povolena multicastové směrování, zasílány periodické PIM *Hello* zprávy.

Kapitola 6

Nástroje pro vizualizaci multicastových toků

Při návrhu a vytváření nových sítí by měly být všechny její části a konfigurace náležitě zdokumentovány. S postupem času a přidáváním nových zařízení do sítě se však málokdy podaří udržet její počáteční přehlednost. Dalším aspektem, proč začaly vznikat nástroje pro vizualizaci a mapování síťových topologií, je možnost pohodlného udržování aktuálního pohledu na stav sítě. V následující kapitole se zaměříme na průzkum nástrojů pro mapování síťových topologií a multicastových toků.

6.1 Získávání informací o síťových topologiích

Nástrojů pro mapování síťových topologií nalezneme mnoho. Liší se však svojí robustností, funkčností a tím, jak prozkoumávají a získávají informaci o síti, do které jsou nasazeny. Některé jsou freewarové, další komerční a některé jsou omezeny počtem detekovatelných zařízení.

Při průzkumu nástrojů se zaměříme zejména na technologie, které využívají tyto nástroje pro mapování síťového okolí. Nejčastěji používanými technologiemi jsou:

- SNMP (Simple Network Management Protocol)
- CDP (Cisco Discovery Protocol)
- LLDP (Link Layer Discovery Protocol)
- ping a traceroute
- DNS (Domain Name System) dotazy

6.1.1 SNMP

Většina nástrojů je založena právě na SNMP, které poskytuje velice širokou škálu informací ze zařízení. Mezi nimi nalezneme ty, které jsou užitečné při mapování síťových topologií jako například: informace o typu a názvu zařízení, připojených zařízeních k síťovým rozhraním, multicastových tocích, o protokolech IGMP, PIM a další.

SNMP se tedy jeví jako ideální při prozkoumávání zapojení sítě. Nástroje, které používají pouze protokol SNMP jsou však „krátké“ na sítě, kde není nakonfigurován tento

protokol. Záleží na síťové topologii, jak moc úspěšný bude nástroj využívající pouze SNMP při prohledávání sítě. Proto většina nástrojů používá i jiné prostředky pro detekci a získávání informací o síťových zařízeních.

6.1.2 CDP a LLDP

Jak již název napovídá, Cisco Discovery Protocol je proprietární protokol od firmy Cisco, který je využíván mezi Cisco zařízeními pro výměnu informací o svých sousedech. Tento protokol používá například program Jdisc Network Topology[1], který prozkoumává topologii na linkové a síťové vrstvě.

CDP se především používá pro získání informací o sousedních zařízeních a to takových jako:

- IP adresy síťových rozhraní
- nakonfigurované protokoly
- verze operačního systému
- hostname
- nastavení duplexu
- a další

CDP je nezávislé na typu používaného média (Ethernet, Frame Relay, ATM) a protokolu mezi dvěma zařízeními. Použití SNMP a CDP Management Information Base umožňuje aplikacím, které mapují počítačové sítě, získat informace o typu zařízení a SNMP adresy agentů sousedních zařízení. CDP je ve výchozím nastavení povoleno na všech aktivních Cisco zařízeních.[16]

LLDP (Link Layer Discovery Protocol) je neproprietární variantou protokolu CDP, který může být použit pro získávání informací o svých sousedních zařízeních.

6.1.3 Ping a traceroute

Utility jako ping a traceroute jsou mapovacími nástroji používány k detekci zařízení v síti. Ze stanice, na které je spuštěn nástroj pro mapování síťové topologie je odeslán ICMP paket typu request na vybranou IP adresu. Jestliže adresované zařízení odpoví, program si zapamatuje IP adresu, kterou dále použije pro získávání podrobnějších informací, například pomocí SNMP či DNS. IP adresy jsou typicky testovány podle zadaného rozsahu v programu. Program tedy oskenuje všechny IP adresy ze zadaného rozsahu.

Nevýhodou tohoto přístupu je například mapování firewalů v síti, které mohou mít zakázáno odpovídat na ICMP pakety, a případně také koncových uživatelských stanic, které mohou být chráněny firewalem.

6.1.4 DNS dotazy

Dalším užitečným nástrojem pro získávání informací jsou DNS dotazy. DNS obsahuje mnoho užitečných informací o zařízeních v síti, zejména pak počítačů. Nejčastěji je používán DNS dotaz typu PTR. [8]

[1] http://www.jdisc.com/en/products/network-topology

Protokol	RFC MIB Name	RFC
IP multicast routing	IPMROUTE-STD-MIB.my	RFC 2932
IGMP	IGMP-STD-MIB.my	RFC 2933
PIM	PIM-MIB.my	RFC 2934
MSDP[2]	MSDP-MIB.my	RFC 4624

Tabulka 6.1: MIB definované IETF [18]

6.2 Mapování multicastu

Mapování multicastu je užitečné v počítačových sítích s vysokým počtem multicastových toků pro pohodlný management, kontrolu a řešení problémů v síti spojených s multicastem. Tyto nástroje nám umožňují:

- detekci multicastového směrování
- grafickou vizualizaci multicastových distribučních stromů v reálném čase
- detekci zahozených multicastových paketů

Velká většina programů požaduje pro zahájení prohledávání počítačové sítě nadefinovat nějaké „počáteční“ zařízení v síti, ze kterého bude dále zkoušet dostupnost sousedních zařízení. Jestliže je sousední zařízení aktivní — většinou je ověřováno pomocí utility ping, snaží se dalšími způsoby, které byly uvedeny v kapitole 3.1, získat informace o propojení síťových prvků a o multicastu. Tento postup je aplikován na všechna dostupná zařízení. Jakmile program získá dostatek informací, může začít vytvářet topologii sítě, zobrazit její mapu a informace o multicastu.

6.2.1 SNMP a multicast

Získávání informací o multicastových tocích je nejčastěji zjišťována pomocí protokolu SNMP. SNMP pracuje s Management Information Base (MIB) moduly, obsahující objekty, které sdružují podobné „managovací“ informace. MIB je hierarchicky členěno do formy stromu, kde každý spravovaný objekt mé své ID. Stejně tak tomu je se zdroji informací o multicastu. V tabulce 6.1 jsou uvedena MIB pro IP multicast od Internet Engineering Task Force (IETF).

Tabulka 6.2 obsahuje vybrané MIB od různých výrobců zařízení zaměřené na multicast. Díky tomu, že každý výrobce používá své MIB, lze mimo standardních informací o zařízeních v síti získat specifičtější informace nejen o multicastu.

Informace o multicastu lze získat zasláním požadavku z NMS (Network Management System) SNMP agentovi, který je nakonfigurován na zařízení nebo také pomocí SNMP trap, kdy agent zašle na NMS informace při splnění definované podmínky. V tabulce 6.3 jsou definovány SNMP události pro zařízení Cisco.

6.2.2 Vybrané nástroje

V této kapitole se budeme zabývat vybranými nástroji mapujícími síťovou topologii a multicastové toky. Nejdříve bude podrobněji popsán *HP Network Node Manager i* a následně bude přiblížen *EMC Ionix Multicast Manager* jako alternativa k prvnímu jmenovanému programu.

Výrobce zařízení	**název MIB**
Alcatel	ALCATEL-IND1-PIMSM-MIB
Alcatel	PIM-MIB
Alcatel	IGMP-STD-MIB
Cisco	CISCO-FC-MULTICAST-MIB
Cisco	CISCO-SWITCH-MULTICAST-MIB
Cisco	CISCO-IGMP-SNOOPING-MIB
Cisco	CISCO-IGMP-FILTER-MIB
Cisco	IGMP-MIB
Cisco	CISCO-PIM-MIB
HP	HP-SN-IGMP-MIB
HP	HP-ICF-PIM
Juniper	Juniper-PIM-MIB
Juniper	Juniper-IGMP-MIB
Nortel	Wellfleet-IGMP-MIB
Nortel	Wellfleet-PIM-MIB
3Com	A3COM0073IGMP-SNOOP

Tabulka 6.2: MIB zaměřené na multicast od různých výrobců [18]

Název události	**ID objektu**
pimNeighborLoss	1.3.6.1.3.61.1.0.1
ciscoPimInterfaceUp	1.3.6.1.4.1.9.9.184.2.0.1
ciscoPimInvalidJoinPrune	1.3.6.1.4.1.9.9.184.2.0.5
ciscoIpMRouteMissingHeartBeats	1.3.6.1.4.1.9.10.2.3.1.0.1
pimNeighborLoss	1.3.6.1.3.61.1.0.1
pimNeighborLoss	1.3.6.1.3.61.1.0.1

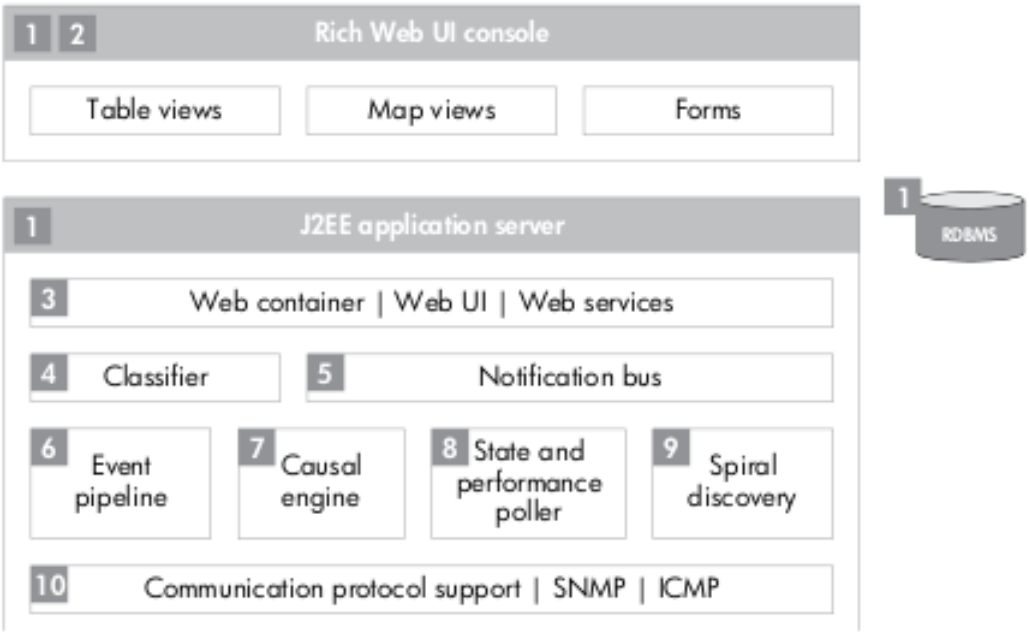
Tabulka 6.3: SNMP traps pro zařízení Cisco [18]

6.2.3 HP Network Node Manager i

Tento velice robustní, avšak škálovatelný program od Hewlett Packard, hledá v reálném čase síťové uzly a přináší tak aktuální pohled na síťovou topologii. Dále pak provádí analýzu sítě pro dosažení minimálních výpadků v síti a zvýšení výkonu sítě. Minimalizaci času při řešení problémů se sítí nám umožní univerzální logovací management, kterého je možno dosáhnout při integraci mezi Network Node Manager i (NNMi) a ArcSight Logger.

Jak by se mohlo zdát, NNMi není určen pouze pro zařízení HP. Škálovatelná architektura umožňuje podporu zařízení od různých firem jako: Cisco, Juniper nebo Alcatel. Obrázek 6.1 znázorňuje architekturu programu NNMi.

Získávání informací pro mapování síťové topologie a multicastových toků jsou získávány zejména pomocí SNMP s podporou využití protokolu ICMP a DNS.



Obrázek 6.1: Architektura progamu HP NNMi [4]

1. N-tier architecture provides deployment flexibility
2. Rich Web client provides greater accessibility
3. Web services and future-proof SOA-based integration interface
4. Classifier provides automatic containerization and grouping
5. Notification bus enables accurate and real-time updates
6. Event pipeline provides high-performance event correlation
7. Causal engine provides advanced deterministic RCA
8. Network state poller with aggressive polling intervals

9. Nearly continuous spiral discovery delivers hyper accurate topology in dynamic environment

10. Diverse communication protocol support

Obrázek s architekturou programu a popis byl převzat z [4].

HP NNMi dále používá tzv. iSPI moduly (Smart Plug-Ins), které rozšiřují základní funkcionalitu celého programu. Narozdíl od programu HP NNMi, není pro tyto moduly uvolněna zkušební licence. Přehled vybraných iSPI modulů:

- HP NNM iSPI pro metriky
- HP NNM iSPI pro IP telefonii
- HP NNM iSPI pro IP multicast
- HP NNM iSPI pro IP multiprotocol label switching (MPLS)

iSPI pro multicast Umožňuje monitorovat multicasové prostředí, prohledává a zobrazuje multicastové uzly a jejich rozhraní a monitoruje problémy spojené s multicastem. Vybrané klíčové vlastnosti modulu iSPI pro multicast [3] :

- rychlá konvergence při změnách v síti
- grafické znázornění síťové topologie
- hledání a izolace chyb při multicastovém směrování
- vyhledávání multicastových toků
- vysoká škálovatelnost — správa až 1000 multicastových uzlů, 15000 multicastových rozhraní a 500 multicastových toků jedinou serverovou instalací HP NNMi.

6.2.4 EMC Ionix Multicast Manager

Dalším programem, který umí managovat multicastové prostředí v síti, je *Ionix Multicast Manager* (IMC) od firmy EMC. Stejně jako HP NNMi, umí IMC vyhledávat a následně vizualizovat multicastové „entity“ a monitorovat chování sítě. Taktéž lze zobrazit členství multicastových skupin, multicastové stromy, aktivní relace a multicastové protokoly. Klíčové vlastnosti IMC [2]:

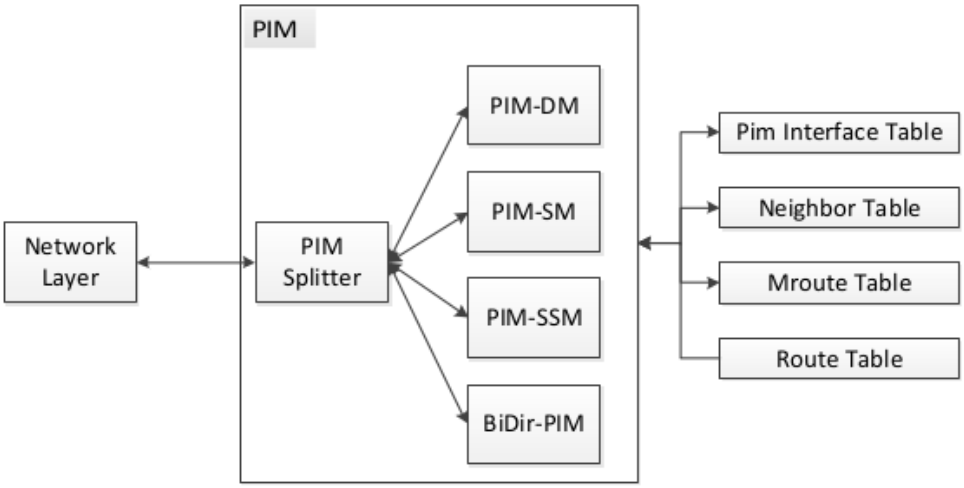
- podpora zařízení s různými multicastovými protokoly (PIM-SM, PIM-SSM, PIM-DM, PIM-SD, IGMPv1/v2/v3, MSDP, a další)
- vyhledávání multicastových zařízení včetně Designated Routers, Rendezvous Points, Bootstrap Routers, mapping agents a další
- podpora L3 multicastu u Cisco a Juniper zařízení
- monitorování probíhá na základě protokolu SNMP využívajícího standardní MIB

Kapitola 7

Návrh rozšíření simulátoru OMNeT++ o podporu PIM-SM

Pro vytvoření návrhu na rozšíření simulátoru bylo získáno dostatek teoretických informací v předešlých kapitolách. Při návrhu jsme vycházeli z diplomové práce Ing. Veroniky Rybové [10], která zahrnuje implementaci PIM-DM a potřebné moduly pro PIM-SM.

Obrázek 7.1 zachycuje návrh složeného PIM modelu. Jednoduché moduly *PIM Splitter* a *PIM-DM* a abstraktní datové typy jako *PIM Interface Table*, *Neighbour Table*, *Mroute Table* a *Route Table* byly již vytvořeny a publikovány v [10]. Tato kapitola se bude věnovat návrhu jednoduchého modulu pro PIM-SM. Dále bude z původního návrhu a implementace odstraněna multicastová směrovací tabulka, protože nová verze knihovny INET 2.0 obsahuje potřebné metody a struktury pro práci s multicastovými směrovacími informacemi přímo v unicastové směrovací tabulce.



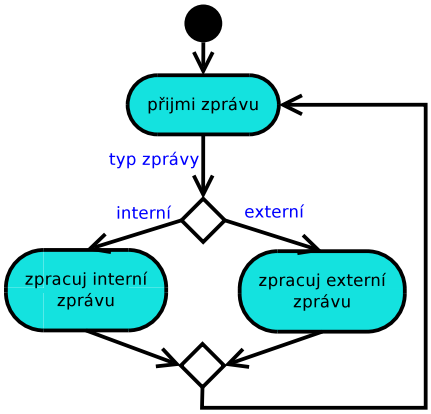
Obrázek 7.1: Architektura PIM modelu [10]

7.1 PIM-SM

V této podkapitole se budeme zabývat návrhem jednoduchého modulu pro PIM-SM. Činnost protokolu PIM-SM je v RFC4601 [7] popsána jedenácti stavovými automaty. Pro lepší

přehlednost a srozumitelnost vytvoříme návrh pomocí diagramu aktivit.

Celý diagram by byl velmi rozsáhlý, z tohoto důvodu byl rozdělen na dva menší, jak je znázorněno na obrázku 7.2. První diagram aktivit zachycuje chování protokolu při příjmu interních zpráv a událostí (obrázek 7.3), kterými jsou například informace o IGMP změnách, novém multicastovém toku, expirace časovačů, atd. Druhý diagram znázorňuje činnost protokolu při příjmu externích zpráv a událostí (obrázek 7.4) jako například: *Join*, *Assert*, *Prune* a *Register*.



Obrázek 7.2: Zjednodušený diagram aktivit pro PIM-SM [10]

7.1.1 Diagram aktivit pro interní zprávy a události

Jestliže směrovač obdrží nový multicastový tok, `PIM Splitter` vytvoří novou multicastovou cestu a doplní informaci o vstupním rozhraní. Další zpracování bude probíhat v modulu PIM-SM, který mimo jiné dokončí nastavení multicastové cesty a provede registraci zdroje. DR směrovač odešle *Register* zprávu adresovanou RP. Zprávy jsou zasílány do doby, než DR směrovač obdrží od RP *Register-Stop* zprávu. Po přijetí musí DR směrovač zdroje nastavit RST časovač.

Odběratelé multicastu informují o svém zájmu či nezájmu o data DR směrovače pomocí IGMP zpráv. Na jejich základě může dojít k přidání či odebrání IP adresy z `oilistu`[1].

V případě, že se `oilist` stane neprázdným, je zaslána *Join* zpráva směrem k *upstream* směrovačům. Následně je nastaven JT časovač, při jehož vypršení je opětovně zaslána *Join* zpráva. Opačným případem je vyprázdnění `oilistu`, kdy je směrem k *upstream* směrovačům zaslána *Prune* zpráva a běh časovače JT je zrušen.

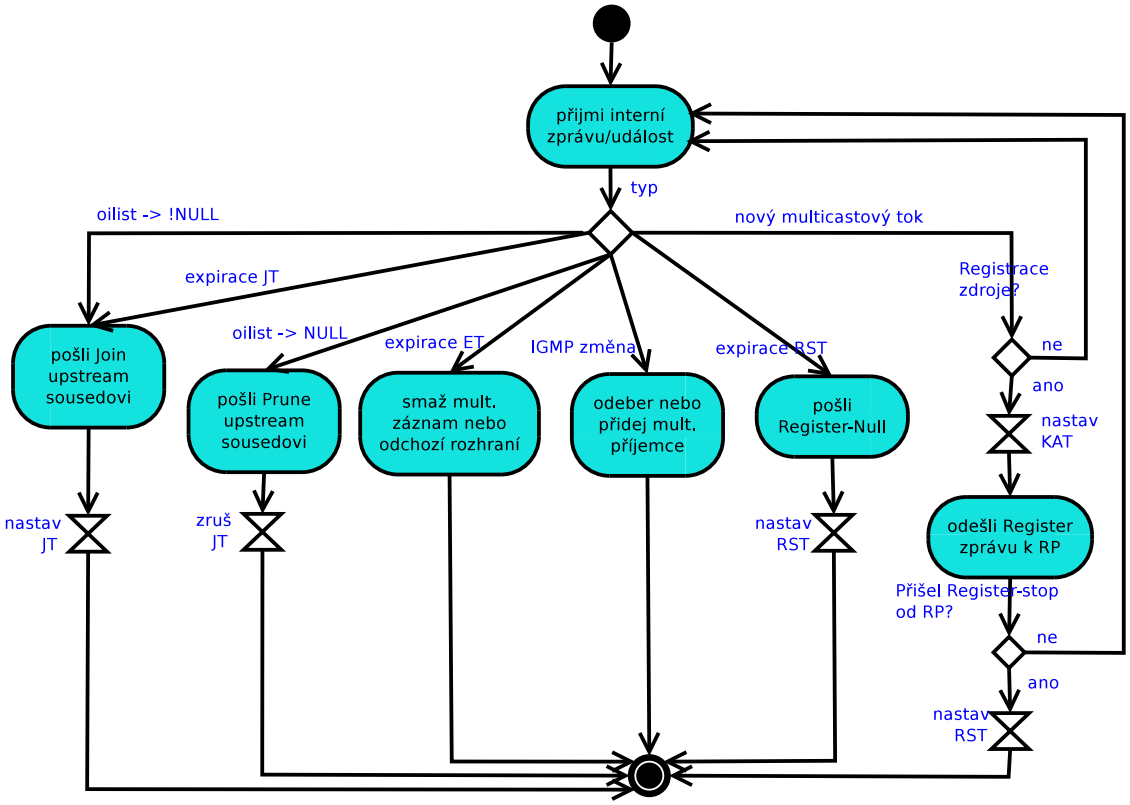
Expirace ET časovače může nastat jak pro odchozí rozhraní, tak pro multicastový záznam. Vypršení ET pro rozhraní iniciuje smazání daného rozhraní. Při expiraci ET pro multicastovou cestu, musí být záznam smazán ze směrovací tabulky.

Jestliže dojde k expiraci RST časovače na DR směrovači zdroje, musí být zaslána *Register-Null* zpráva RP směrovači a znovu nastaven RST časovač.

7.1.2 Diagram aktivit pro externí zprávy

Po přijetí *Join* zprávy na *downstream* rozhraní je v případě existence multicastové směrovací cesty restartován ET časovač. Jestliže záznam v tabulce neexistuje, je vytvořen společně

[1] Seznam odchozích rozhraní



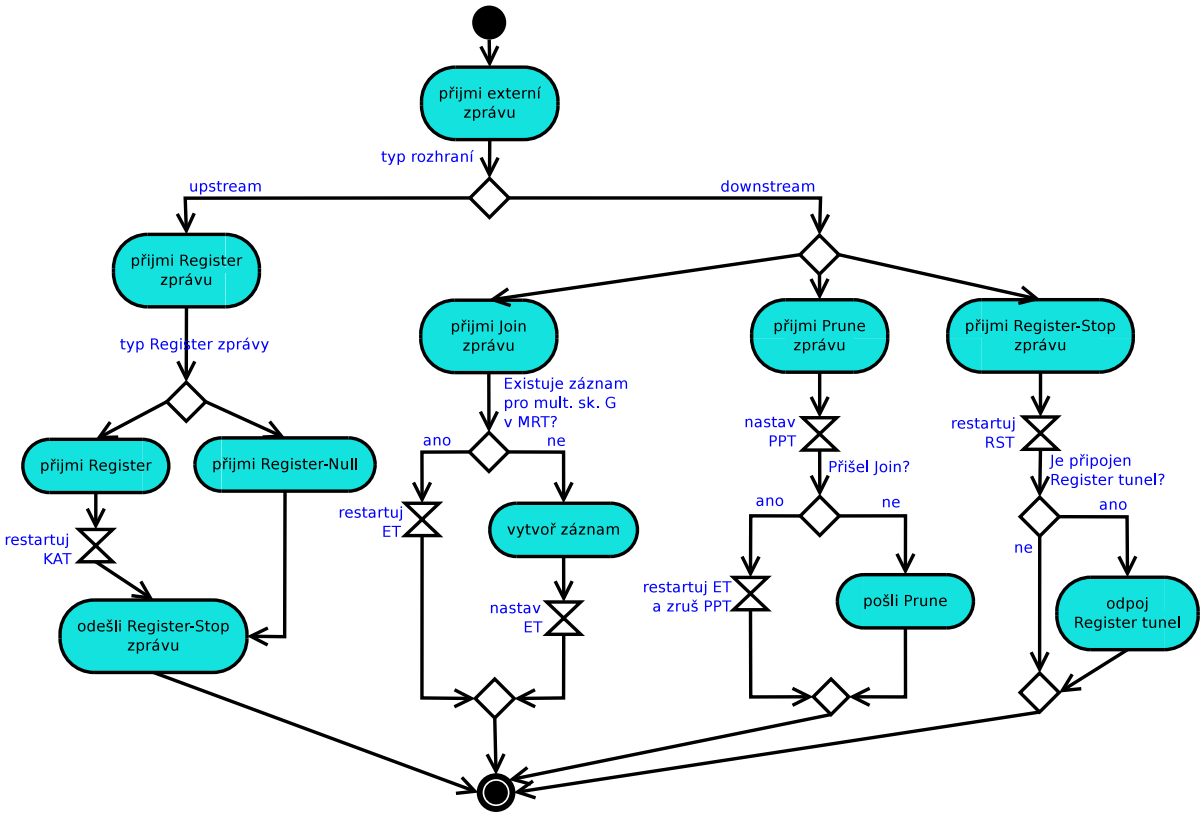
Obrázek 7.3: Diagram aktivit pro příjem interních zpráv a událostí

s ET časovačem.

V případě, že na *downstream* rozhraní dorazí zpráva *Prune*, je toto rozhraní smazáno z `oilistu`. Jestliže se `oilist` vyprázdní, nastaví se PPT a po jeho expiraci je odeslána *Prune* zpráva *upstream* směrovači. Přijde-li *Join* zpráva před vypršením časovače, zruší se PPT a je restartován ET časovač.

Při registraci nového multicastového toku přijde na *upstream* rozhraní RP směrovače *Register* zpráva. Jedná-li se o typ *Register-Null*, je nazpět odeslána *Register-Stop* zpráva. Přijetím klasické *Register* zprávy je restartován KAT časovač a následně odeslána *Register-Stop* zpráva.

Přijetím *Register-Stop* zprávy na *downstream* rozhraní DR směrovače multicastového zdroje je restartován RST časovač. Jestliže je připojen *Register* tunel, bude odpojen a tím bude zaručeno, že další multicastová data nebudou dále zapouzdřována do *Register* zpráv.



Obrázek 7.4: Diagram aktivit pro příjem externích zpráv

7.2 Konfigurace PIM-SM na síťových prvcích v OMNeT++

Aby bylo možné provádět simulaci, je nutné směrovače příslušně nakonfigurovat. K tomuto účelu slouží modul `deviceConfigurator`, který vznikl v rámci diplomové práci Ing. Marka Černého [19]. Tento modul při inicializaci simulace načte konfigurační soubory pro směrovače.

PIM-DM a PIM-SM mají, co se týče konfigurace na směrovačích, mnoho společného. S výhodou proto můžeme využít upravený `deviceConfigurator` pro PIM a konfigurační soubor z výše zmiňované práce[10]:

```
<Routing>
    <Multicast enable="1">
        <Pim>
            <RPAddress>
                <IPAddress>192.168.1.2</IPAddress>
                <Acl>1</Acl>
            </RPAddress>
            <BSRCandidate>
                <Interface>eth0</Interface>
                <Mask>30</Mask>
            </BSRCandidate>
            <RPCandidate>
                <Interface>eth1</Interface>
```

```
                <Ttl>2</Ttl>
                <GroupList>1</GroupList>
            </RPCandidate>
            <SPTthreshold>infinity</SPTthreshold>
        </Pim>
    </Multicast>
</Routing>
<Interfaces>
    <Interface name="eth0">
        <Pim>
            <Mode>sparse-mode</Mode>
            <Border />
            <StateRefresh>
                 <OriginationInterval></OriginationInterval>
            </StateRefresh>
        </Pim>
    </Interface>
</Interfaces>
```

V navrženém konfiguračním souboru provedeme pouze dvě úpravy. První se týká rozhraní označeného elementem `Interfaces` v zanořeném elementu `Mode`, kde je nutné specifikovat mód PIM na `sparse-mode`. Dále je potřeba přidat element `SPTthreshold` mezi element `Pim` v hlavním elementu `Routing`, který umožňuje nastavení optimalizační fáze činnosti PIM-SM.

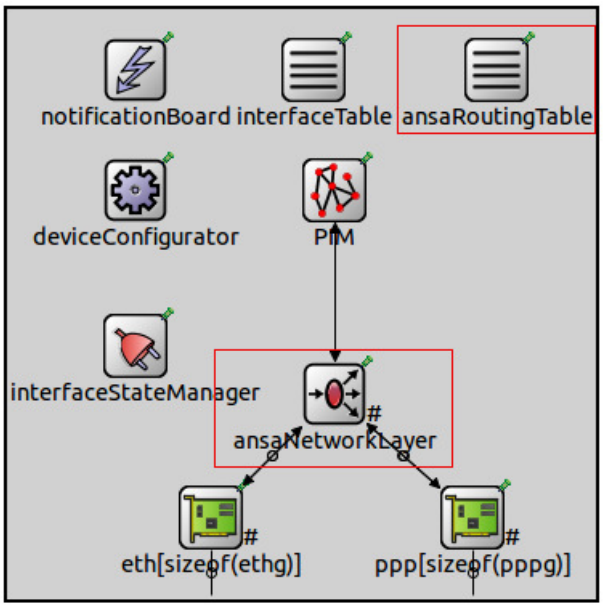
Kapitola 8

Implementace multicastového směrování v prostředí OMNeT++

V následující kapitole si popíšeme důležité části kódu, nově naprogramované třídy a úpravy, které bylo potřeba realizovat pro vytvoření funkčního modelu PIM-SM. Implementace probíhala na základě návrhu z kapitoly 7.

8.1 Úpravy a portování modulu PIM pro novou verzi INET

Před samotnou implementací protokolu PIM-SM bylo nutné upravit tu stávající pro PIM-DM a jeho podpůrné částí, neboť byla uvolněna nová verze frameworku INET 2.0, se kterou nebyly původní zdrojové kódy kompatibilní. Na obrázku 8.1 je zobrazen model PIM routeru s vyznačenými moduly, které bylo třeba upravit.



Obrázek 8.1: Modul ansaPIMRouteru s upravenými moduly

8.1.1 Multicastové směrování

K velkým změnám došlo u tříd `multicastRoutingTable` a `multicastIPRoute`, vytvořených v rámci [10]. Většina metod a proměnných ze třídy `multicastRoutingTable` je v nové verzi INETu 2.0 dostupná pod jinými názvy v třídě `RoutingTable`. Z tohoto důvodu bylo potřeba vyhledat správnou verzi nové metody a nahradit ji v původním kódu. Pro lepší kompatibilitu a vzhledem k tomu, že některé metody z `multicastRoutingTable` nebyly dostupné v `RoutingTable`, byla tato třída poděděna jako `AnsaRoutingTable`, kde byly doplněny chybějící metody. Obdobné úpravy byly provedeny s původní třídou `MulticastIPRoute`, nahrazenou v novém INETu třídou `IPv4MulticastRoute`. Metody a struktury specifické pro PIM-SM a PIM-DM byly doimplementovány do poděděné třídy `AnsaIPv4MulticastRoute`.

Pro PIM-SM byly do třídy `AnsaIPv4MulticastRoute` doplněny metody pro práci s časovači (KAT (*Keep Alive Timer*), RST (*Register Stop Timer*), JT (*Join Timer*), ET (*Expiry Timer*), PPT (*Prune Pending Timer*)). Dále byly vytvořeny pomocné funkce, např. pro nastavení odchozího rozhraní, přičemž příslušná struktura byla rozšířena o proměnné související s PIM-SM.

```
struct outInterface
{
    ...
    PIMet           *expiryTimer;   /**< Pointer to PIM Expiry Timer*/
    RegisterState   regState;       /**< Register state. */
    bool            shRegTun;       /**< Show interface which is
                                    also register tunnel interface*/
};
```

8.1.2 Síťová vrstva

V modulu síťové vrstvy `ansaNetworkLayer` bylo nutné provést několik úprav, které jsou spojeny zejména s metodou `routeMulticastPacket()`. Do metody bylo přidáno rozdělení řízení programu na základě módu, ve kterém se nachází příchozí rozhraní paketu. Jestliže má rozhraní nakonfigurován *Dense Mode*, je zavolána metoda `routePimDM()`. V případě, že je mód rozhraní *Sparse Mode*, datagram je zpracován funkcí `routePimSM()`.

Metoda `routePimSM()` nejdříve předá řízení pomocí `Notification Boardu`[1] do modulu `pimSM`, kde je restartován časovač KAT, pokud pro danou multicastovou cestu existuje. Následně je získán seznam odchozích rozhraní pro danou multicastovou skupinu. Jestliže je rozhraní ve stavu *Forward*, následuje před samotným odesláním multicastového paketu RPF kontrola.

```
for (unsigned int i=0; i<outInt.size(); i++) {
    // do not send to pruned interface
    if (outInt[i].forwarding == Pruned)
        continue;
    // RPF check before datagram sending
    if (outInt[i].intId == rpfInt->getInterfaceId())
        continue;
    fragmentAndSend(datagramCopy, destIE, destAddr);
}
```

[1] Podrobnější popis u modulu `pimSplitter`, kaptiola 8.3.1

Speciální případ směrování multicastových dat může nastat na DR směrovači, který se stará o registraci zdroje dat. Tato situace je identifikována pomocí příznaků nastavených pro multicastovou cestu a stavem rozhraní, které je použito jako *Register tunnel*. V případě, že má dojít k registraci multicastového zdroje, je předáno řízení do modulu `pimSM`, který se postará o správné zpracování multicastového paketu.

Stávající implementace IGMP nebyla použitelná pro kooperaci s PIM-SM. Z tohoto důvodu bylo ve funkci `handlePacketFromNetwork()` speciálně ošetřeno zpracování IGMP zprávy, která je generována modulem `IPgen`. Tato zpráva má nastaveno stejné protokolové číslo jako standardní IGMP paket, ovšem formát zprávy je rozdílný. Pro účely připojení a odpojení multicastových příjemců bude toto řešení dostačující.

8.2 Implementace podpůrných struktur

Mnoho podpurných a použitelných struktur pro PIM-SM bylo již vytvořeno v [10], avšak bylo potřeba vytvořit nové časovače pro PIM-SM a provést několik úprav v PIM zprávách.

8.2.1 PIM zprávy

Struktura PIM zpráv je definována podle RFC. Byly vynechány pouze ty části paketu, které jsou nedefinované, nulové, či nejsou použitelné při simulaci [10]. Veškeré PIM zprávy jsou definovány v souboru `PIMPacket.msg`, ze kterého je při překladu vytvořen C++ kód.

U PIM zpráv bylo potřeba upravit `PIMRegister` paket, který žádným způsobem nebyl přizpůsoben pro zapouzdření a přenos multicastových dat. Tento problém byl vyřešen zjednodušenou strukturou `MultData`, která obsahuje pouze proměnné s IP adresou zdroje multicastu a IP adresou multicastové skupiny.

V souboru `PIMPacket.msg` bylo nutné vytvořit strukturu `EncodedAddress`, která představuje IP adresu s třemi bity, které ve zprávě *Join/Prune* určují, o jaký typ zprávy *Join* nebo *Prune* se jedná.

```
struct EncodedAddress
{
    IPv4Address IPaddress;
    bool        S;          // Sparse bit
    bool        W;          // WildCard bit
    bool        R;          // RPT bit
}
```

8.2.2 PIM časovače

Časovače jsou specifikovány v souboru `PIMTimer.msg`. Pro PIM-SM bylo třeba vytvořit časovače KAT, ET, JT a RST. Při simulaci je časovač modelován jako zpráva, která je doručena objektu, který ji vytvořil, s nastaveným zpožděním. Struktura zpráv pro časovače *Join* a *Register-Stop* vypadá následovně:

```
// represents PIM Join timer
packet PIMjt extends PIMTimer
{
    timerKind = JoinTimer;
    IPv4Address      group;             // Multicast group address
```

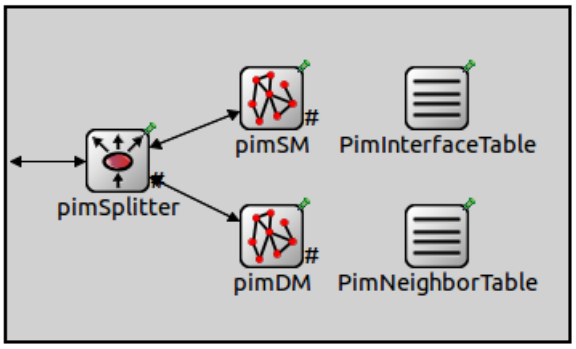
```
    IPv4Address      JoinPruneAddr;     // Joined or Pruned IP Address
    IPv4Address      upstreamNbr;       // Address of upstream neigbor
    int              JoinType;          // Type of Join message
}

// represents PIM Register-Stop timer
packet PIMrst extends PIMTimer
{
    timerKind = RegisterStopTimer;
    IPv4Address      source;            // Source of multicast
    IPv4Address      group;             // Multicast group address
}
```

U každého časovače je uveden jeho typ. Další proměnné se odvíjí od toho, pro jaký účel je časovač použit. Například u časovače *Join* potřebujeme znát IP adresy multicastové skupiny, „*Join*“ adresu, která určuje kam zpráva směřuje, IP adresu *upstream* směrovače a typ *Join* zprávy. Tyto informace jsou poté použity při expiraci časovače pro odeslání *Join* zprávy *upstream* sousedovi.

8.3 Implementace protokolu PIM-SM

Protokol PIM-SM je implementován podle návrhu z kapitoly 7 jako jednoduchý modul, který je jednou z součástí složeného modulu `pim` z obrázku 8.2. Dalšími částmi je tabulka sousednosti (`PimNeighborTable`) a tabulka PIM rozhraní (`PimInterfaceTable`), které jsou podrobněji popsány v [10]. Oba jednoduché moduly `pimSM` a `pimDM` jsou připojeny k `pimSplitteru`, jehož implementaci bylo potřeba upravit pro správné fungování s modulem `pimSM`.



Obrázek 8.2: Složený model protokolu PIM

8.3.1 Modul PIM Splitter

Tento modul je přímo připojen k síťové vrtsvě `ansaNetworkLayer`, ze které přijímá PIM zprávy od jiných směrovačů, nebo je naopak zasílá. PIM zprávy jsou dále zasílány do příslušných modulů (`pimSM`, `pimDM`) podle toho, jaký mód má nastaveno příchozí rozhraní. `pimSplitter` se také stará o zpracování *Hello* zpráv a udržování sousedství.

Důležitou částí pro PIM-SM v modulu `pimSplitter` je metoda `newMulticast`, která zpracovává nový multicastový tok. Tuto událost potřebujeme zachytit při registraci nového multicastového zdroje na DR směrovači. Metoda vytvoří novou multicastovou cestu, nastaví některé její parametry (multicastovou skupinu, zdroj multicastu a vstupní rozhraní) a podle nakonfigurovaného PIM módu předá řízení do modulu `pimDM` nebo `pimSM` pomocí `NotificationBoardu`.

```
// notification for PIM module about new multicast route
if (pimInt->getMode() == Dense)
    nb->fireChangeNotification(NF_IPv4_NEW_MULTICAST_DENSE, newRoute);
if (pimInt->getMode() == Sparse)
    nb->fireChangeNotification(NF_IPv4_NEW_MULTICAST_SPARSE, newRoute);
```

Modul `NotificationBoard` z knihovny INET lze použít pro zasílání asynchronních upozornění mezi moduly. Pro vyvolání údalosti se používá `fireChangeNotification()` a pro její příjem musí cílová metoda, kde má být událost zpracována, reimplementovat metodu `receiveChangeNotification()`. Obě metody mají dva parametry. Prvním je číslo události, které je specifikováno v souboru `NotifiersConst.h`. Druhým je libovolný objekt, který je předán do modulu příjemce notifikace. S výše uvedenými informacemi nám ještě notifikace nebude fungovat. Pro její zpracování je potřeba se příhlásit k odběru pomocí metody `subscribe()` s parametrem udávajícím číslo události.

Pro upozornění modulu `pimSM` na IGMP změny byly v metodě `igmpChange()` přidány notifikace pro přihlášení (`NF_IPv4_NEW_IGMP_ADDED_PISM`) a odhlášení (`NF_IPv4_NEW_IGMP_REMOVED_PIMSM`) multicastových příjemců. Modul `pimSM` poté zpracuje notifikace podle příslušné akce. Toto zpracování bude blíže popsáno v následující kapitole.

8.3.2 Implementace modulu pimSM

Modul `pimSM` je přes `pimSplitter` připojen k síťové vrstvě `ansaNetworkLayer`. Jeho chování implementuje třída `pimSM`, jejíž hlavičkový soubor je uveden v příloze D.1. Implementace probíhala dle návrhu v kapitole 7.1 na základě diagramu aktivit.

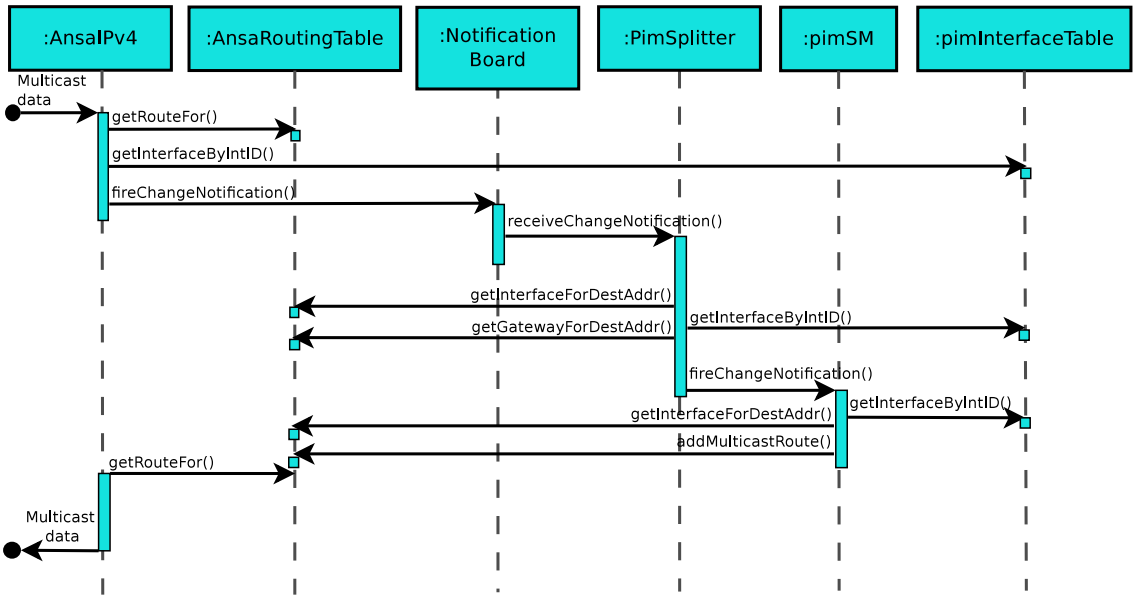
Asynchronní události

Nejdříve začneme popisem zpracování asynchronních událostí, které, jak jsme si již dříve uvedli, obsluhuje modul `NotificationBoard`. Mezi implementované asynchronní události patří:

- nový multicastový tok (`NF_IPv4_NEW_MULTICAST_SPARSE`)
- multicastová data na DR směrovači zdroje multicastu (`NF_IPv4_MDATA_REGISTER`)
- nový zájemce o multicastová data (`NF_IPv4_NEW_IGMP_ADDED_PISM`)
- příjemce již nemá o multicast zájem (`NF_IPv4_NEW_IGMP_REMOVED_PIMSM`)
- data na RPF rozhraní (`NF_IPv4_DATA_ON_RPF_PIMSM`)

V podkapitole 8.3.1 jsme začli s popisem zpracování události při detekci nového multicastového toku. Řízení je poté předáno z modulu `pimSplitter` do `pimSM`, kde je notifikace zpracována metodou `receiveChangeNotification()`. Ta zajistí předání řízení do

`newMulticastRegisterDR()`. Metoda doplní odchozí rozhraní do multicastové cesty pro daný zdroj a multicastovou skupinu a nastaví příznaky: P (odříznutá cesta), F (směrovač provádí „registraci“) — je připojen přímo ke zdroji multicastu, T (indikuje, že nejméně jeden paket byl přijat přes SPT strom). Pro multicastovou cestu je vytvořen KAT časovač (`createKeepAliveTimer()`), který po expiraci zavolá metodu `processKeepAliveTimer()`, která odstraní multicastový záznam ze směrovací tabulky, se kterou je časovač svázán. Při registraci musí být také vytvořen záznam typu (*,G) s KAT časovačem, který má nastaveno příchozí rozhraní (směrem k RP směrovači) a prázdné (Null) odchozí rozhraní. Po jejich nastavení jsou cesty přidány do směrovací tabulky metodou z třídy `ansaRoutingTable`: `addMulticastRoute()`. Sekvenční diagram popisující registraci nového multicastového zdroje je uveden na obrázku 8.3.

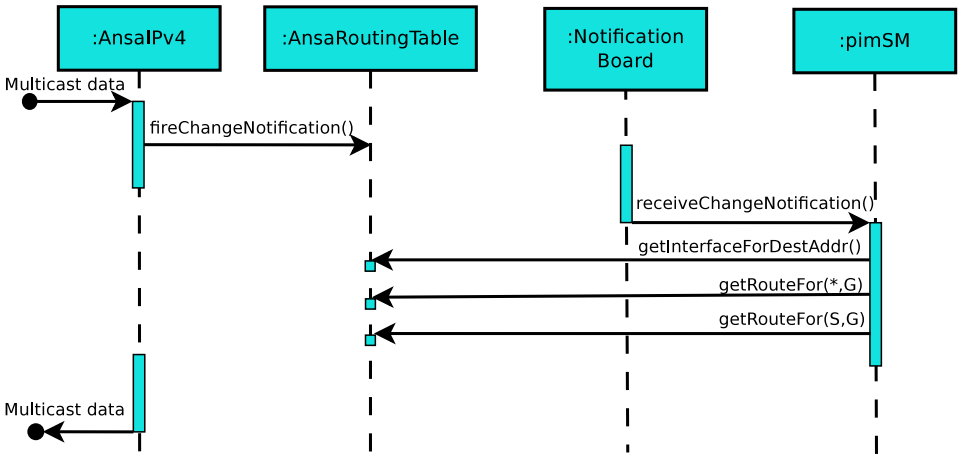


Obrázek 8.3: Sekvenční diagram popisující zaregistrování nového multicastového zdroje

Asynchronní událost starající se o zpracování multicastových dat, která přišla na DR směrovač zdroje multicastu, je vyvolána ze síťové vrstvy. V `pimSM` modulu je zpracována metodou `sendPIMRegister()`, která se postará o restartování KAT časovačů. Dále je zkontrolováno, v jakém stavu se nachází *Register* tunel mezi DR a RP směrovačem. Jedná se o virtuální rozhraní, které má sloužit pro zasílání *Register* zpráv a určovat, kdy mají (rozhraní je připojeno) a nemají (rozhraní je odpojeno) být *Register* zprávy dále zasílány. V našem případě se jedná o klasické rozhraní, které má navíc specifikováno *Register* stav. Jestliže je stav rozhraní *Join*, odešle se *Register* zpráva. *Prune* stav rozhraní indikuje, že se zpráva zasílat nemá. Sekvenční diagram popisující zpracování multicastových dat na DR směrovači zdroje je uveden na obrázku 8.4.

```
if (routeSG->getRegStatus(intToRP->getInterfaceId()) == Join)
    ...    // create and send Register message
else if (routeSG->getRegStatus(intToRP->getInterfaceId()) == Prune)
    ...    // Register tunnel disconnected - don't sentRegister message
```

Přidání nového příjemce multicastu zajišťuje metoda `newMulticastReciever()`. Z parametru `addRemoveAddr` je zjištěno, jaké adresy byly zaregistrovány na rozhraní směrovače



Obrázek 8.4: Sekvenční diagram znázorňující zpracování multicastových dat na DR směrovači zdroje

pomocí IGMP. Pro každou IP adresu je ověřováno, zda pro danou multicastovou skupinu, ke které se příjemce přihlašuje, existuje cesta typu (*,G). V případě, že ne, je záznam vytvořen, nastaví se adresy pro mult. skupinu a RP směrovač, dále jsou přidány vstupní a výstupní rozhraní a nastaveny příznaky: S (značící Sparse Mode), C (router je přímo přpojen k příjemci). Protože již není seznam odchozích rozhraní nulový (oilist != Null) musí být odeslána *Join* zpráva směrem k *upstream* směrovači. U (*,G) cesty vytvoříme a nastavíme JT časovač (`createJoinTimer()`), který periodicky zasílá *Join* zprávy směrem k RP, dále ET časovač (`createExpiryTimer()`) pro odchozí rozhraní, po jehož vypršení je metodou `processExpiryTimer()` dané rozhraní smazáno. ET časovač je také nastaven pro celou multicastovou cestu. Jeho restartování je prováděno na základě příjmu `Join` zprávy. Jakmile časovač pro záznam vyprší, provede se jeho smazání. Při přidávání nového příjemce však může nastat situace, že cesta pro danou multicastovou skupinu existuje (již je vybudován RPT strom od RP k příjemcům). V této situaci se nový příjemce pouze napojí do stávajícího RPT stromu (přidá se odchozí rozhraní k existující cestě) a nastaví se příznak C.

Odhlášení příjemce multicastu je velmi podobné. O tuto činnost se stará metoda `removeMulticastReciever()`, která smaže rozhraní u příslušné multicastové cesty a zruší nastavený ET.

```
for (k = 0; k < outInt.size(); k++)
{ // delete interface given by pimInt
    if (outInt[k].intId == pimInt->getInterfaceID())
    {
         if (outInt[k].expiryTimer)          // delete ET timer
         {
              cancelEvent(outInt[k].expiryTimer);
              delete (outInt[k].expiryTimer);
         }
         outInt.erase(outInt.begin() + k);   // delete interface
    }
}
```

Jestliže multicastová cesta již nemá žádné odchozí rozhraní, musí být zaslána *Prune* zpráva směrem k RP směrovači pomocí metody `sendPIMJoinPrune()` a zrušen JT časovač.

Poslední implementovanou asynchronní událostí je příjem multicastových dat na RPF rozhraní, což znamená, že je vybudován jak zdrojový (SPT) strom, tak sdílený (RPT), kde příjemci odebírají multicastová data. Notifikace o datech na RPF rozhraní vysílá síťová vrstva přímo do `pimSM` modulu. Zde se o zpracování této události postará metoda `dataOnRpf()`, která pro danou multicastovou cestu restartuje její KAT časovače. Odlišností u implementace Cisco a RFC 4601 pro PIM-SM jsou nastavované časové intervaly. V RFC 4601 je pro KAT určeno 210 s, avšak zařízení od firmy Cisco nastaví KAT na 180 s. Hodnota 210 s je použita v okamžiku, kdy některý z příjemců multicastu začne odebírat multicastová data.

Externí zprávy

O zpracování externích zpráv se stará metoda `handleMessage()`, která určí, zda se jedná o vlastní zprávu (zpráva byla zaslána ze stejného modulu). Tento případ nastane u časovačů při jejich vypršení. Jestliže se jedná o zprávu typu `PIMPacket`, je zpráva dále zpracována metodou `processPIMPkt()`, která podle typu paketu (`JoinPrune`, `Register`, `RegisterStop`) zavolá příslušnou metodu.

Jestliže zdroj multicastu začne nabízet multicastová data, provede se registrace zdroje na DR směrovači a je zaslán *Register* paket adresovaný RP směrovači. Ten je potom na RP zpracován metodou `processRegisterPacket()`. V případě, že router nemá ve své směrovací tabulce příslušné multicastové cesty, vytvoří je. Má-li multicastová cesta (*,G) nastaveno nějaké odchozí rozhraní (existují příjemci multicastu) a je ve stavu *Forward*, provede se „rozbalení“ dat z *Register* paketu. V našem případě je získána struktura `MultData`, která obsahuje IP adresu multicastového zdroje a multicastové skupiny. Datový paket je potom zaslán metodou `forwardMulticastData()` do RPT stromu k příjemcům multicastových dat. Jelikož data přišla zapouzdřena v *Register* paketu, musí být odeslána *Join* a *Register-Stop* zpráva k DR směrovači zdroje. V případě, že se jedná o tzv. *Register-Null* paket, který je periodicky zasílán z RP směrovače (pro udržování *Register* tunelu v odpojeném stavu), provede se pouze odeslání *Register-Stop* zprávy.

Register-Stop zpráva z RP je odeslána metodou `sendPIMRegisterStop()`, která vytvoří `PIMRegisterStop` zprávu, nastaví řídící parametry pro zprávu a odešle ji. Její zpracování provádí metoda `processRegisterStopPacket()`. Nejdříve je vytvořen RST časovač (`createRegisterStopTimer()`), který zajistí pravidelné odesílání *Register-Null* zpráv. Jestliže je tunel připojený, je nastaven do odpojeného stavu. Tím se zabrání při dalším příjmu multicastových dat opětovnému zasílání *Register* zpráv.

K nejrozsáhlejší implementaci patří zpracování *Join/Prune* zprávy, které bylo rozděleno do několika dílčích metod. Metoda `processJoinPrunePacket()` určí, zda se jedná o *Join* nebo *Prune* zprávu. V případě *Join* zprávy metoda `processJoinPacket()` určí, o jaký typ se jedná, zda-li o (*,G) nebo o (S,G). Podle této informace začne zprávu příslušně zpracovávat.

Při zpracování *Join(*,G)* zprávy je vytvořena multicastová cesta s nastaveným příchozím rozhraním, které je směrem k RP směrovači, odchozími rozhraními směrem k příjemcům multicastu a příznaky. Pro multicastový záznam musí být vytvořeny časovače: JT, ET a také ET pro každé odchozí rozhraní. V případě, že v seznamu odchozích rozhraní jedno přibylo, musí být vyslána *Join(*,G)* zpráva k *upstream* sousedovi. Ovšem pouze tehdy, když se nejedná o RP směrovač, který musí zaslat *Join(S,G)*.

Jestliže již multicastový záznam ve směrovači existuje a směrovač, který přijal zprávu *Join*, je RP, bude zpráva zpracována metodou `processJoinRouteGexistOnRP()`. Ta postupně prochází veškeré multicastové cesty, které odpovídají multicastové skupině v *Join* zprávě. Pro (*,G) záznamy jsou odebrány příznaky P a F, restartován KAT časovač a v případě, že rozhraní, na které přišla *Join* zpráva, není v seznamu odchozích rozhraní, je přidáno.

Zprávy *Join* jsou zasílány také periodicky po vypršení JT časovače. Cílem těchto periodických zpráv je udržovat multicastové směrovací informace aktivní. Pokud by nebyly zasílány, ET časovače pro rozhraní a cestu by vypršely a záznam o odchozím rozhraní, resp. celý multicastový záznam, by byl smazán. Restartování ET časovačů zajišťuje metoda `restartExpiryTimer()`, která provede obnovení času jak pro ET časovač odchozího rozhraní, tak pro celou cestu.

Struktura zpracování *Join(S,G)* zprávy (`processSGJoin()`) je obdobná s tím rozdílem, že při budování SPT stromu musí být nejdříve vytvořena (*,G) cesta směrem od RP a následně (S,G) multicastová cesta. Sousedním *upstream* směrovačům jsou zasílány *Join(S,G)* zprávy.

Prune zprávy jsou zpracovávány metodou `processPrunePacket()`. Zpracování *Prune(S,G)* a *Prune(*,G)* probíhá velice podobně. Nejdříve je získán seznam výstupních rozhraní dané cesty pro multicastovou skupinu uvedenou v *Prune* zprávě. Následně je smazáno rozhraní, na které dorazila zpráva *Prune*. Před samotným smazáním rozhraní je nutné zrušit a odstranit ET časovač. Jestliže se seznam odchozích rozhraní stal prázdným, musí být nastaveny příslušné příznaky pro multicastový záznam a odeslána zpráva *Prune* k *upstream* směrovači.

Protože implementace probíhala zpočátku dle RFC 4601 byl naprogramován PPT časovač, který má zajistit zpoždění při odesílání *Prune* zprávy. Po detailnějším prozkoumání na reálném Cisco zařízení však bylo zjištěno, že při přijetí *Prune* zprávy je tato ihned přeposílána *upstream* sousedovi (pouze v případě, že se seznam odchozích rozhraní stane prázdným). Z tohoto důvodu byl definován parametr `CISCO_SPEC_SIM`, který uvádí, zda se má simulace chovat dle Cisco specifikace, nebo dle RFC. Na základě tohoto parametru je poté proveden podmíněný překlad programu.

8.3.3 Shrnutí

Mnoho podpůrných struktur a funkční multicastové prostředí bylo vytvořeno v rámci [10]. S příchodem nové verze INETu a OMNeTu++ se předchozí implementace stala nepoužitelnou. Proto bylo nejdříve nutné upravit stávající verzi kódů, které se týkaly multicastu, a poté začít s implementací módu PIM-SM. Největších změn doznala původní multicastová směrovací tabulka `MulticastRoutingTable`, která byla zrušena a nahrazena tabulkou `AnsaRoutingTable`. Nová tabulka dědí implementaci originální směrovací tabulky z knihovny INET `RoutingTable` rozšířené o metody z původní `MulticastRoutingTable`.

Některé stávající struktury byly rozšířeny pro potřeby *Sparse* módu, např. struktura odchozího rozhraní `outInterface`. Byly také vytvořeny nové pomocné metody, které ušetřily mnoho řádků kódu a ulehčily práci při samotném programování PIM-SM (`setAddresses()`, `addOutIntFull()`). V některých částech implementace modulů `PimSplitteru` a původní `MulticastRoutingTable` nebyla zohledněna případná implementace PIM-SM, a proto byly příslušné metody upraveny.

Nejobsáhlejší a nejobtížnější byla implementace modulu `pimSM`. Pro PIM-SM byly vytvořeny nové časovače a byly upraveny některé zadefinované PIM zprávy. Samotná im-

plementace PIM-SM obsahuje velké množství metod, které jsou uvedeny v hlavičkovém souboru třídy `pimSM` v příloze D.1.

Implementace byla ověřena na několika testovacích scénářích, které jsou nyní součástí ANSA knihovny. Detailnější popis a porovnání výsledků simulace a reálné sítě bude uveden v následující kapitole 9.

Kapitola 9

Porovnání simulace s chováním reálné sítě v prostředí Cisco

V poslední kapitole provedeme srovnání chování simulace s reálnou sítí na základě obsahu směrovacích tabulek a zpráv, které jsou zasílány mezi směrovači. Pro testování na reálné síti byly použity směrovače Cisco C2691 s verzí IOS 12.4(23).

9.1 Metodika porovnávání

K porovnávání budeme používat obsah směrovacích tabulek pouze na směrovačích, kde došlo ke změně, aby byla zachována přehlednost a práce nebyla plná obrázků.

Pro zachytávání síťového provozu byl použit program Wireshark, pomocí kterého byly vyfiltrovány pouze PIM pakety. Výstup z Wiresharku je ve formátu: číslo paketu, čas zachycení paketu, IP adresa zdroje, IP adresa cíle, protokol (PIMv2) a typ zprávy.

Pro lepší přehlednost při simulaci byly k originálním názvům zpráv do závorky přidány upřesňující popisky. Například u zprávy *Join/Prune* nevíme, zda se jedná o *Join*, nebo *Prune*. Museli bychom nahlédnout do podrobného výpisu při simulaci. Proto je výpis v simulaci upřesněn: *Join/Prune(Join)* — zde zpráva nese informaci typu *Join*. Obdobně je přidán popisek ke zprávě *Register*, když se jedná o tzv. *Register-Null* zprávu.

Formát výpisu zachycených PIM zpráv při simulaci z konzole je následující: simulační čas, název směrovače s rozhraním, kde byla zpráva přijata, následuje typ zprávy a v závorce IP adresa odesílatele a příjemce.

Zařízení, která jsou použita v testovací topologii budou v textu pro lepší přehlednost označena kurzívou.

9.2 Testovací scénáře

Pro testování bylo vytvořeno několik scénářů, které ověřují různé situace v síti, které mohou nastat:

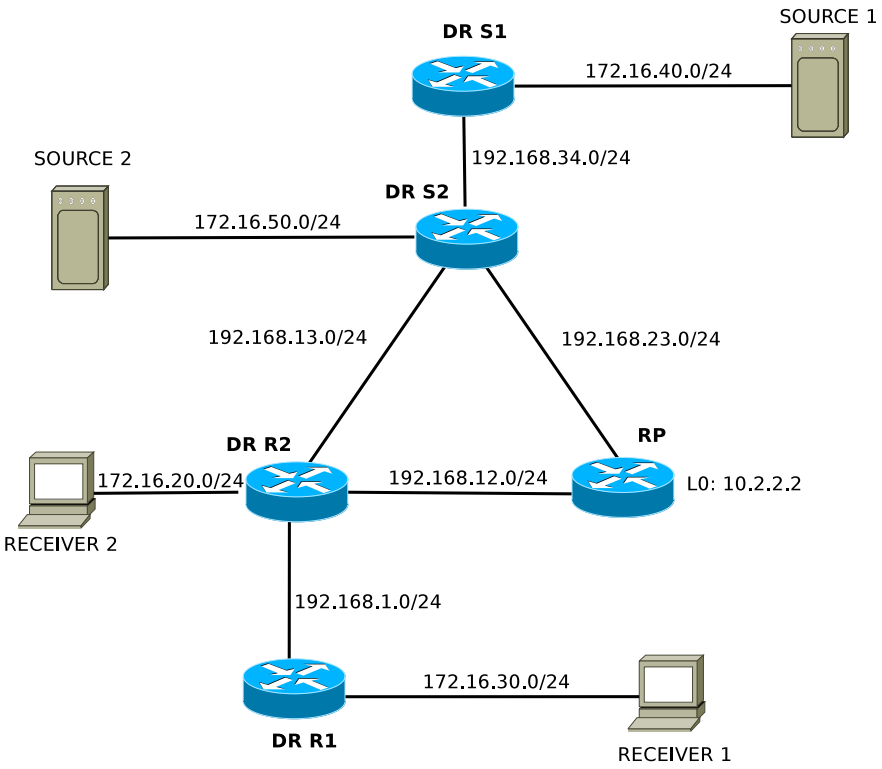
1. Ověření funkčnosti registrace zdroje. Nejdříve mají o multicast zájem příjemci, poté začne multicastový zdroj vysílat data.

2. Ověření funkčnosti registrace zdroje. Zdroj multicastových dat začne vysílat data, následně se o ně přihlásí zájemci.

3. Ověření funkčnosti časovačů.

4. Dvě multicastové skupiny od dvou multicastových zdrojů v síti. Každá skupina má svého příjemce.

Testování probíhalo na topologii z obrázku 9.1. Zdroj multicastu *Source 2* je aktivní a vysílá multicastová data pouze ve čtvrtém scénáři. Tento scénář je zahrnut v příkladech, které jsou součástí ANSA knihovny (příloha B).



Obrázek 9.1: Testovací topologie společná pro všechny scénáře

9.2.1 Scénář 1 - „registrace, nejdříve příjemci poté zdroj“

Následující scénář má za úkol ověřit funkčnost implementace při registraci multicastového zdroje v případě, že v síti „čekají“ příjemci na multicastová data. Podrobný rozpis událostí iniciovaný zdrojem multicastu a příjemci je uveden v tabulce 9.1. Seznam akcí, které mohou nastat:

- SS — zdroj multicastu vysílá data
- SE — zdroj multicastu přestal vysílat data
- RS — příjemce multicastu se hlásí o příjem multicastových dat
- RE — příjemce multicastu již nemá zájem o multicastová data

Č.	Čas [s]	Síťové zařízení	Multicastová skupina	Akce
1	8.0	Receiver 2	239.0.0.11	RS
2	10.0	Receiver 1	239.0.0.11	RS
3	12.0	Source 1	239.0.0.11	SS

Tabulka 9.1: Rozpis událostí pro 1. scénář

Krok 1

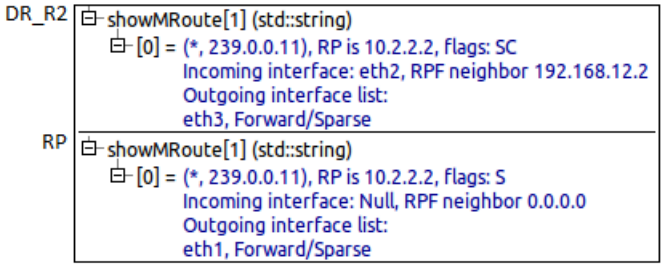
Receiver 2 se přihlásí k multicastové skupině 239.0.0.11, směrovač *DR_R2* si vytvoří pro multicastovou skupinu záznam do směrovací tabulky a vyšle *Join* zprávu směrem k *RP* směrovači. Sítí se již od začátku šíří *Hello* zprávy. Obsah směrovacích tabulek je na obrázku 9.2 a 9.3.

Zprávy přijaté na směrovačích při simulaci:

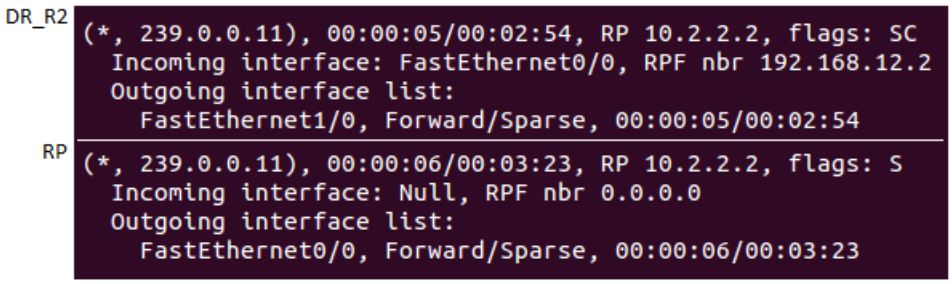
```
Send: IGMP-239.0.0.11 at time = 8
8.000019459998 RP(102): PIMJoin/Prune(Join) (192.168.12.1, 224.0.0.13)
```

Pakety zachycené na lince mezi *DR_R2* – *RP* v reálné síti:

```
84    21:50:55.003811    192.168.12.1    224.0.0.13    PIMv2    Join/Prune
```



Obrázek 9.2: Obsah multicastových směrovacích tabulek během simulace – scénář 1, 1. krok



Obrázek 9.3: Obsah multicastových směrovacích tabulek v reálné síti – scénář 1, 1. krok

Z výčtu zpráv zachycených při simulaci si můžeme všimnout, že bylo zaznamenáno: `Send: IGMP-239.0.0.11 at time = 8`. Předchozí zpráva znamená, že *Receiver 2* zaslal IGMP zprávu. Ta zaručí, že se na *DR_R2* zaregistruje multicastová skupina na rozhraní a tato událost iniciuje vyslání *Join(*,G)* zprávy směrem k *RP*.

Krok 2

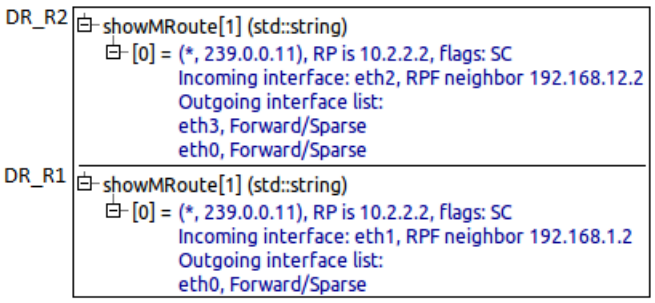
Receiver 1 se také přihlásí k multicastové skupině 239.0.0.11, směrovač *DR_R1* si vytvoří pro multicastovou skupinu záznam do směrovací tabulky a vyšle *Join* zprávu adresovanou *DR_R2* směrovači. Na *DR_R2* již existuje RPT strom pro multicastovou skupinu 239.0.0.11. Z tohoto důvodu je pouze přidáno odchozí rozhraní, na které přišla *Join* zpráva, do seznamu odchozích rozhraní multicastové cesty na *DR_R2*. Obsah směrovacích tabulek je na obrázku 9.4 a 9.5.

Zprávy přijaté na směrovačích při simulaci:

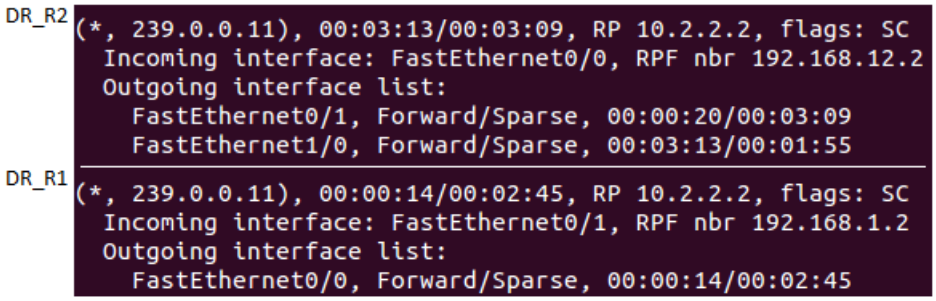
```
Send: IGMP-239.0.0.11 at time = 10
10.000019459998 DR_R2(101): PIMJoin/Prune(Join) (192.168.1.1, 224.0.0.13)
```

Pakety zachycené na lince mezi *DR_R1* – *DR_R2* v reálné síti:

```
125     22:00:45.942671     192.168.1.1     224.0.0.13     PIMv2     Join/Prune
```



Obrázek 9.4: Obsah multicastových směrovacích tabulek během simulace – scénář 1, 2. krok



Obrázek 9.5: Obsah multicastových směrovacích tabulek v reálné síti – scénář 1, 2. krok

Krok 3

Source 1 začne vysílat multicastová data. *DR_S1* data přijme, vytvoří záznamy do multicastové směrovací tabulky, zabalí data do PIM *Register* zprávy a zasílá ji *RP* směrovači. *RP* přijme data, také vytvoří multicastové cesty do směrovací tabulky a vyšle data do sdíleného stromu směrem k příjemcům. *RP* pošle *Join(S,G)* pro vytvoření zdrojového stromu mezi *RP* a *DR_S1* a následně odešle *Register-Stop* zprávu, aby se zabránilo dvojímu odesílání dat z *DR_S1*. Další multicastová data procházejí nativně až k příjemcům. Obsah směrovacích

tabulek je na obrázku 9.6 a 9.7.

Zprávy přijaté na směrovačích při simulaci:

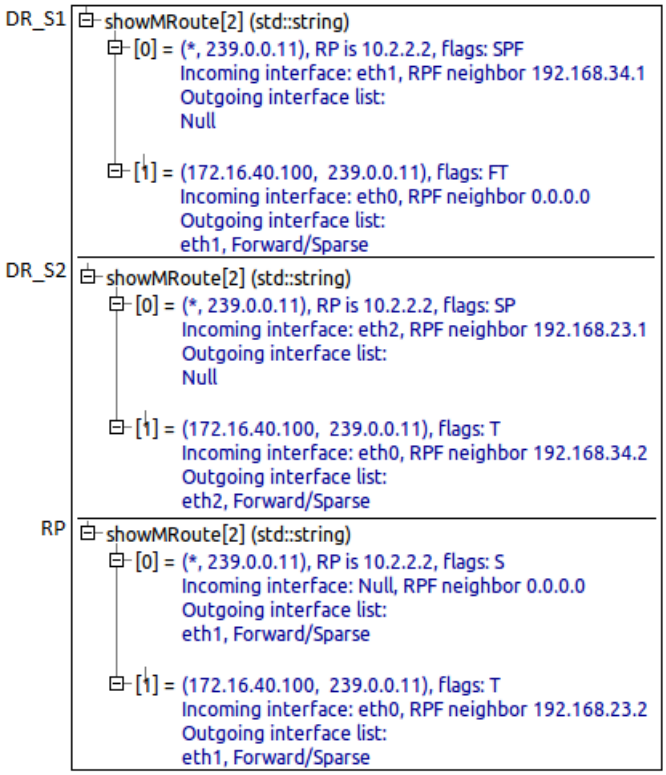
```
Send: MultData 239.0.0.11 at time = 12
12.000053309993 RP(101): PIMRegister (192.168.34.2, 10.2.2.2)
12.000060079992 DR_S2(103): PIMJoin/Prune(Join) (192.168.23.1, 224.0.0.13)
12.000066849991 DR_S1(102): PIMJoin/Prune(Join) (192.168.34.1, 224.0.0.13)
12.00007356999 DR_S1(102): PIMRegisterStop (10.2.2.2, 192.168.34.2)
Send: MultData 239.0.0.11 at time = 14
Send: MultData 239.0.0.11 at time = 16
```

Pakety zachycené na lince mezi *DR_S1* – *DR_S2* v reálné síti:

```
37 23:43:15.043511 192.168.34.2 10.2.2.2 PIMv2 Register
38 23:43:15.060154 192.168.34.1 224.0.0.13 PIMv2 Join/Prune
48 23:43:19.066139 10.2.2.2 192.168.34.2 PIMv2 Register-stop
```

Pakety zachycené na linkách mezi *DR_S2* – *DR_RP* v reálné síti:

```
64 23:43:15.045631 192.168.34.2 239.0.0.11 PIMv2 Register
65 23:43:15.055955 192.168.23.1 224.0.0.13 PIMv2 Join/Prune
74 23:43:19.064024 10.2.2.2 192.168.34.2 PIMv2 Register-stop
```



Obrázek 9.6: Obsah multicastových směrovacích tabulek během simulace – scénář 1, 3. krok

DR_S1

```
(*, 239.0.0.11), 00:04:43/stopped, RP 10.2.2.2, flags: SPF
  Incoming interface: FastEthernet0/0, RPF nbr 192.168.34.1
  Outgoing interface list: Null

(172.16.40.100, 239.0.0.11), 00:04:43/00:03:25, flags: FT
  Incoming interface: FastEthernet0/1, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse, 00:00:35/00:02:54
```

DR_S2

```
(*, 239.0.0.11), 00:00:30/stopped, RP 10.2.2.2, flags: SP
  Incoming interface: FastEthernet0/1, RPF nbr 192.168.23.1
  Outgoing interface list: Null

(172.16.40.100, 239.0.0.11), 00:00:30/00:03:16, flags: T
  Incoming interface: FastEthernet0/0, RPF nbr 192.168.34.2
  Outgoing interface list:
    FastEthernet0/1, Forward/Sparse, 00:00:30/00:03:10
```

RP

```
(*, 239.0.0.11), 00:34:42/stopped, RP 10.2.2.2, flags: S
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse, 00:34:42/00:03:09

(172.16.40.100, 239.0.0.11), 00:00:16/00:02:58, flags: T
  Incoming interface: FastEthernet0/1, RPF nbr 192.168.23.2
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse, 00:00:16/00:02:43
```

Obrázek 9.7: Obsah multicastových směrovacích tabulek v reálné síti – scénář 1, 3. krok

9.2.2 Scénář 2 - „registrace, nejdříve zdroj poté příjemci“

Následující scénář má za úkol ověřit funkčnost a správnost implementace v situaci, kdy nejdříve začne vysílat data multicastový zdroj, a až poté se o data přihlásí příjemci.

V tomto scénáři se zaměříme zejména na posloupnost zachycených zpráv, než na obsah směrovacích tabulek. Ten zůstává stejný jako u předchozího scénáře, a proto budou vynechány, abychom práci nezahltili opakujícími se obrázky.

Rozpis akcí pro 2. scénář je uveden v tabulce 9.2. Značení akcí zůstává stejné jako v předchozím příkladě.

Č.	Čas [s]	Síťové zařízení	Multicastová skupina	Akce
1	9.0	Source 1	239.0.0.11	SS
2	15.0	Receiver 1	239.0.0.11	RS
3	20.0	Receiver 2	239.0.0.11	RS

Tabulka 9.2: Rozpis událostí pro 2. scénář

Krok 1

Source 1 začne vysílat multicastová data. Na *DR_S1* směrovači proběhne registrace nového zdroje multicastu a pro *RP* je zaslána *Register* zpráva. Jelikož na *RP* nejsou zatím žádní příjemci, je nazpět směrovači *DR_S1* odeslána *Register-Stop* zpráva. Oba směrovače si vytvoří příslušné multicastové záznamy. Na *DR_S2* není žádná multicastová cesta, protože *Register* zpráva je posílána unicastově mezi *RP* a „registrujícím“ směrovačem. Do doby,

než se o multicastová data přihlásí příjemci, nejsou přes směrovač *DR_S1* zasílána. Obsah směrovacích tabulek je na obrázku 9.8 a 9.9.

Zprávy přijaté na směrovačích při simulaci:

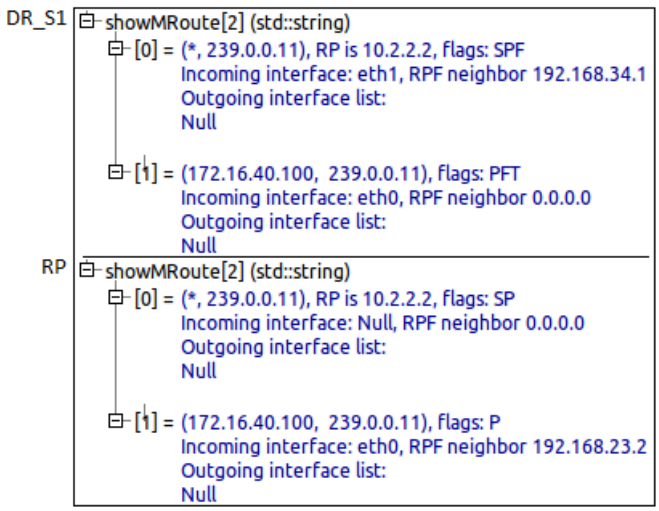
```
Send: MultData 239.0.0.11 at time = 9
9.000053309993 RP(101): PIMRegister (192.168.34.2, 10.2.2.2)
9.000066849991 DR_S1(102): PIMRegisterStop (10.2.2.2, 192.168.34.2)
Send: MultData 239.0.0.11 at time = 14
```

Pakety zachycené na lince mezi *DR_S1* – *DR_S2* v reálné síti:

```
90 21:02:29.852902 192.168.34.2 10.2.2.2 PIMv2 Register
91 21:02:29.862045 10.2.2.2 192.168.34.2 PIMv2 Register-stop
```

Pakety zachycené na lince mezi *DR_S2* – *RP* v reálné síti:

```
90 21:02:29.855025 192.168.34.2 10.2.2.2 PIMv2 Register
91 21:02:29.859179 10.2.2.2 192.168.34.2 PIMv2 Register-stop
```

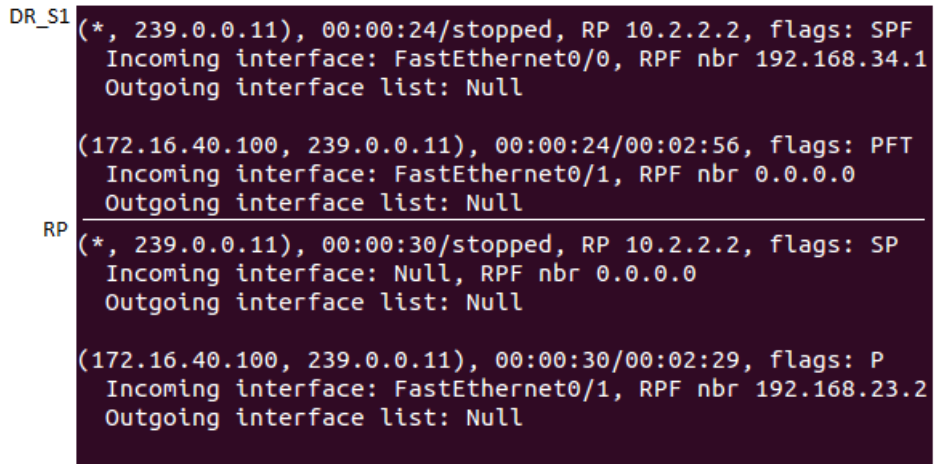


Obrázek 9.8: Obsah multicastových směrovacích tabulek během simulace – scénář 2, 1. krok

Krok 2

Receiver 1 se na základě IGMP zprávy přihlásí o multicastová data. Tato událost iniciuje vyslání *Join(*,G)* zprávy směrem k *RP* směrovači. Jelikož je v síti aktivní zdroj dat a na *RP* jsou vytvořeny příslušné multicastové cesty, je z *RP* směrovače odeslána *Join(S,G)* zpráva. Ta je šířena až k *DR_S1* směrovači zdroje, čímž je vytvořen zdrojový strom a další data mohou nativně procházet až k přihlášeným příjemcům (*Receiver 1*).

Obsah směrovacích tabulek na směrovačích *RP*, *DR_S2* a *DR_S1* je totožný jako v předchozím příkladě, v kroku 3 (9.2.1).



Obrázek 9.9: Obsah multicastových směrovacích tabulek v reálné síti – scénář 2, 1. krok

Zprávy přijaté na směrovačích při simulaci:

```
Send: IGMP-239.0.0.11 at time = 15
15.000019459998 DR_R2(101): PIMJoin/Prune(Join) (192.168.1.1, 224.0.0.13)
15.000026229997 RP(102): PIMJoin/Prune(Join) (192.168.12.1, 224.0.0.13)
15.000032999996 DR_S2(103): PIMJoin/Prune(Join) (192.168.23.1, 224.0.0.13)
15.000039769995 DR_S1(102): PIMJoin/Prune(Join) (192.168.34.1, 224.0.0.13)
Send: MultData 239.0.0.11 at time = 19
```

Pakety zachycené na lince mezi (postupně) *DR_R1 – DR_R2*, *DR_R2 – RP*, *RP – DR_S2*, *DR_S2 – DR_S1* v reálné síti:

```
49 21:21:33.092732 192.168.1.1 224.0.0.13 PIMv2 Join/Prune
60 21:21:33.094813 192.168.12.1 224.0.0.13 PIMv2 Join/Prune
181 21:21:33.099003 192.168.23.1 224.0.0.13 PIMv2 Join/Prune
185 21:21:33.103228 192.168.34.1 224.0.0.13 PIMv2 Join/Prune
```

Krok 3

Receiver 2 se přihlásí o multicastová data. Protože je již vybudován sdílený strom od *RP* k příjemci 1, je pouze přidáno odchozí rozhraní do multicastové cesty na směrovači *DR_R2*. Další multicastová data budou procházet k oběma příjemcům multicastu.

9.2.3 Scénář 3 - ověření činnosti časovačů PIM-SM

V posledním scénáři ověříme funkčnost časovačů, které jsou v PIM-SM používány. Jedná se o následující časovače: *Keep Alive Timer*, *Expiry Timer*, *Join Timer* a *Register Timer*. Při registraci budeme uvažovat variantu z druhého scénáře, tedy nejdříve začne vysílat multicastový zdroj, poté se o multicast přihlásí příjemci.

Implementace PIM-SM používá velice zjednodušenou podporu IGMP, která pouze signalizuje zájem nebo nezájem o multicastová data. Z tohoto důvodu se při ověřování funkčnosti časovačů zaměříme zejména na pořadí expirace jednotlivých časovačů.

Porovnávání provedeme od simulačního času t=38 s, kdy jsou data zaslána multicastovým zdrojem nativně až k příjemcům. Před uvedeným časem proběhne registrace zdroje a

přihlášení příjemců, které jsme si ukázali v druhém scénáři. Multicastová data jsou zasílána každých 25 s až do času t=238 s, kdy je zaslán poslední datový paket. Seznam akcí, které nastanou při simulaci, je uveden v tabulce 9.4.

Č.	Čas [s]	Síťové zařízení	Multicastová skupina	Akce
1	13.0	Source 1	239.0.0.11	SS
2	20.0	Receiver 2	239.0.0.11	RS
3	30.0	Receiver 1	239.0.0.11	RS
5	100.0	Receiver 1	239.0.0.11	RE
6	170.0	Receiver 2	239.0.0.11	RE
7	238.0	Source 1	239.0.0.11	SE

Tabulka 9.3: Rozpis událostí pro 3. scénář

Krok 1

Na všech směrovačích jsou již vytvořeny příslušné multicastové záznamy (viz. scénář 2). V čase t=38 s jsou ze zdroje *Source 1* zaslána multicastová data celým stromem až k příjemcům. Při průchodu těchto dat jednotlivými směrovači musí být restartovány KAT časovače pro (*,G) a (S,G) multicastové cesty na *DR_S1*, *DR_S2* a *RP*. Tato akce je opakována při každém přijetí dat od zdroje.

Krok 2

V čase t=73 s vyprší RST časovač (uplynulo 60 s od registrace zdroje). Na základě této události je odeslána *Register-Null* zpráva pro RP směrovač. Odpovědí *RP* směrovače je zaslání *Register-Stop* zprávy adresované *DR_S1*.

Zprávy přijaté na směrovačích při simulaci:

```
73.000080389989 R2(101): PIMRegister(Null) (192.168.34.2, 10.2.2.2)
73.000093929987 DR_S1(102): PIMRegisterStop (10.2.2.2, 192.168.34.2)
```

Pakety zachycené na lince mezi *DR_S1* – *DR_S2* v reálné síti:

```
132 16:41:17.639259 192.168.34.2 10.2.2.2 PIMv2 Register
133 16:41:17.647829 10.2.2.2 192.168.34.2 PIMv2 Register-stop
```

Pakety zachycené na lince mezi *DR_S2* – *RP* v reálné síti:

```
126 16:41:17.643657 192.168.34.2 10.2.2.2 PIMv2 Register
127 16:41:17.645692 10.2.2.2 192.168.34.2 PIMv2 Register-stop
```

Krok 3

Expirace JT časovače v čase t=80 s (uplynulo 60 s od přihlášení přijímače 2 k multicastové skupině 239.0.0.11) na směrovači *DR_R2* vyvolá odeslání periodické *Join/Prune* zprávy pro *RP* směrovač (v našem případě se jedná o typ *Join*). Jakmile *RP* přijme tuto zprávu, restartuje ET časovač pro rozhraní, na kterém ji přijal, a také pro celý multicastový záznam. JT časovače postupně vyprší také na směrovačích *RP*, *DR_S2*, *DR_S1* a v čase t=90 s také na *DR_R1*.

Zprávy přijaté na směrovačích při simulaci:

```
80.000019459998 RP(102): PIMJoin/Prune(Join) (192.168.12.1, 224.0.0.13)
80.000026229997 DR_S2(103): PIMJoin/Prune(Join) (192.168.23.1, 224.0.0.13)
80.000032999996 DR_S1(102): PIMJoin/Prune(Join) (192.168.34.1, 224.0.0.13)
90.000019459998 DR_R2(101): PIMJoin/Prune(Join) (192.168.1.1, 224.0.0.13)
```

Pakety zachycené na lince mezi (postupně) *DR_R2 – RP*, *RP2 – DR_S2*, *DR_S2 – DR_S1*, *DR_R1 – DR_R2* v reálné síti:

```
120 16:41:19.708040 192.168.12.1 224.0.0.13 PIMv2 Join/Prune
120 16:41:12.764571 192.168.23.1 224.0.0.13 PIMv2 Join/Prune
136 16:41:21.040498 192.168.34.1 224.0.0.13 PIMv2 Join/Prune
110 16:41:29.583302 192.168.1.1 224.0.0.13 PIMv2 Join/Prune
```

Krok 4

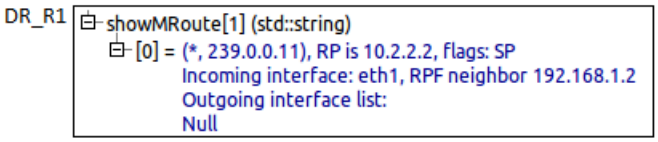
Příjemce multicastu *Receiver 1* se odhlásí v čase t=100 s od multicastové skupiny 239.0.0.11. Směrovač *DR_R1* po obdržení IGMP zprávy smaže odchozí rozhraní k příjemci z multicastové cesty a JT časovač (nebude zasílána periodická *Join/Prune* zpráva). Multicastová cesta již neobsahuje žádné odchozí rozhraní, a proto se směrovač odřeže pomocí *Join/Prune* zprávy zaslané *upstream* sousedovi. Obsah směrovacích tabulek je na obrázku 9.10 a 9.11.

Zprávy přijaté na směrovačích při simulaci:

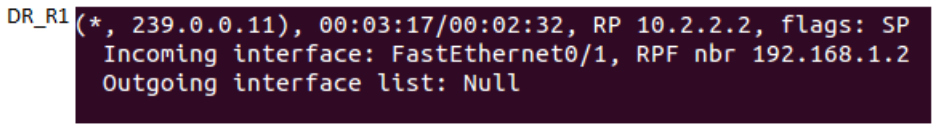
```
Send: IGMP-239.0.0.11 at time = 100
100.000019459998 DR_R2(101): PIMJoin/Prune(Prune) (192.168.1.1, 224.0.0.13)
```

Pakety zachycené na lince mezi *DR_R1 – DR_R2* v reálné síti:

```
122 16:41:42.798608 192.168.1.1 224.0.0.13 PIMv2 Join/Prune
```



Obrázek 9.10: Obsah multicastové směrovací tabulky během simulace – scénář 3, 4. krok

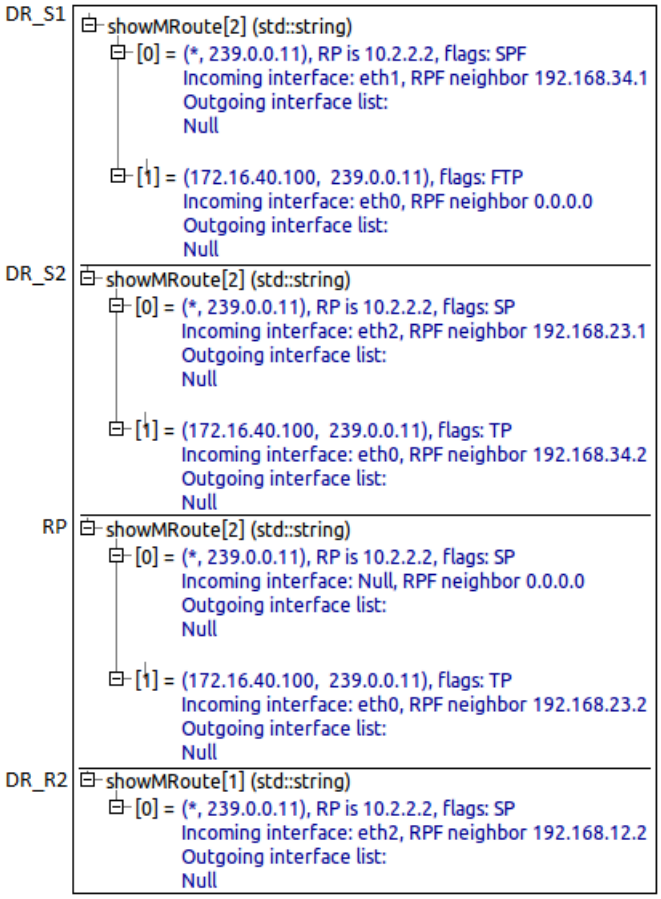


Obrázek 9.11: Obsah multicastové směrovací tabulky v reálné síti – scénář 3, 4. krok

Krok 5

Směrovač *DR_S1* zasílá každých 60 s periodické *Register-Null* zprávy, na které RP směrovač odpovídá zprávou *Register-Stop*. Expirace JT časovačů na *DR_R2*, *RP*, *DR_S2* pravidelně iniciuje odeslání *Join/Prune* zprávy. Tyto události již nebudeme podrobně zaznamenávat a popisovat. Zaměříme se na události, které toto periodické chování přeruší. Např. vypršení ET časovače rozhraní na *DR_R2*.

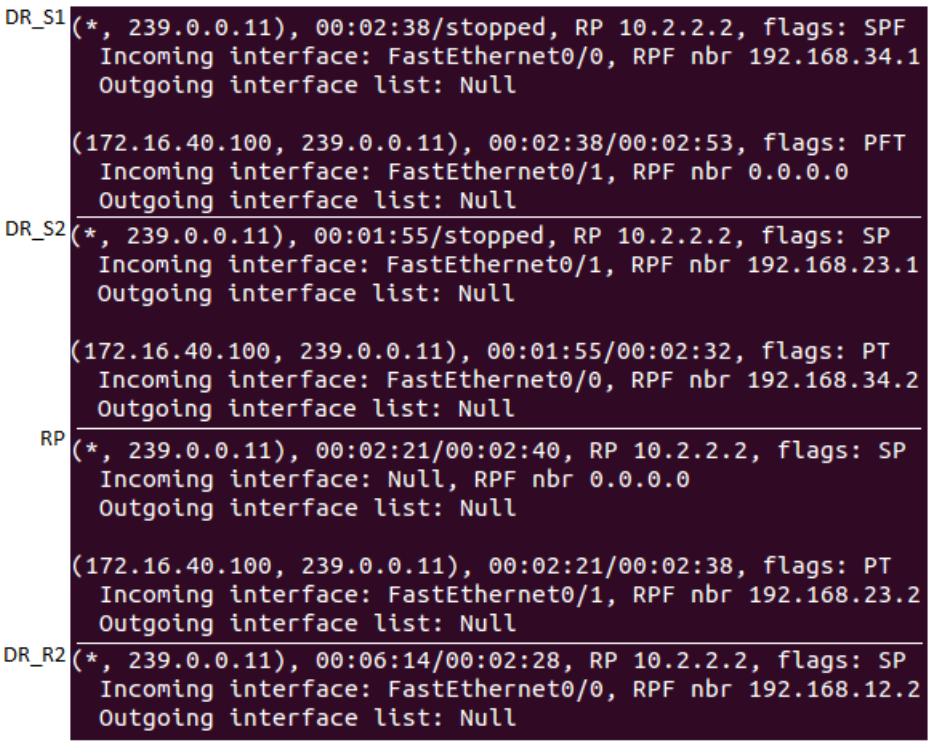
V čase t=170 s se odhlásí i druhý příjemce multicastu. Na směrovači *DR_S2* je tak smazáno odchozí rozhraní k tomuto příjemci. Jelikož se po odstranění rozhraní stal seznam odchozích rozhraní nulový, směrovač se odřeže zasláním *Join/Prune* zprávy. Od distribučního stromu se také odřežou směrovače *RP*, *DR_S2*. Obsah směrovacích tabulek je na obrázku 9.12 a 9.13.



Obrázek 9.12: Obsah multicastových směrovacích tabulek během simulace – scénář 3, 5. krok

Zprávy přijaté na směrovačích při simulaci:

```
Send: IGMP-239.0.0.11 at time = 170
170.000019459998 RP(102): PIMJoin/Prune(Prune) (192.168.12.1, 224.0.0.13)
170.000026229997 DR_S2(103): PIMJoin/Prune(Prune) (192.168.23.1, 224.0.0.13)
170.000032999996 DR_S1(102): PIMJoin/Prune(Prune) (192.168.34.1, 224.0.0.13)
```



Obrázek 9.13: Obsah multicastových směrovacích tabulek v reálné síti – scénář 3, 5. krok

Pakety zachycené na lince mezi (postupně) *DR_R2 – RP*, *RP2 – DR_S2*, *DR_S2 – DR_S1*, v reálné síti:

```
188 16:42:53.477760 192.168.12.1 224.0.0.13 PIMv2 Join/Prune
197 16:42:53.479929 192.168.23.1 224.0.0.13 PIMv2 Join/Prune
205 16:42:53.482070 192.168.34.1 224.0.0.13 PIMv2 Join/Prune
```

Krok 6

Od času t=240 s bude docházet k postupnému mazání multicastových cest ve směrovacích tabulkách z důvodu expirace KAT nebo ET časovačů. Mezi *RP* a *DR_S1* směrovačem jsou stále periodicky zasílány *Register-Stop* a *Register-Null* zprávy.

Při měření expirací časovačů v reálné síti byly přepočítány relativní časy ze simulace do denního času a z multicastových směrovacích tabulek byl zjištěn okamžik, kdy daný časovač vypršel.

Z tabulky 9.4 je zřejmé, že pořadí vypršení časovačů zůstalo zachováno. Časy pro expiraci (*,G) ET časovače na *DR_R1* a *DR_R2* se pro simulaci a reálnou síť liší nejvíce. To je dáno zjednodušenou implementací IGMP, která mezi směrovačem a příjemcem multicastu pouze zajišťuje přihlášení a odhlášení zájemců o multicast. V reálné síti se ovšem router dotazuje hostů, zda mají zájem o multicast. Příjemci multicastu na tento dotaz zasílají odpověď, na základě které je na směrovači proveden restart ET časovače.

Čas [s] (sim.)	**Čas [s]** (real.)	**Zařízení**	**Událost**
240	280	DR_R1	Expirace ET pro (*,G) mult. cestu
300	350	DR_R2	Expirace ET pro (*,G) mult. cestu
373	382	DR_S2, RP	Expirace KAT pro (S,G) mult. cestu
448	417	DR_S1	Expirace KAT pro (S,G) mult. cestu
553	559	DR_S2, RP	Expirace KAT pro (*,G) mult. cestu
628	597	DR_S1	Expirace KAT pro (*,G) mult. cestu

Tabulka 9.4: Srovnání Expirace časovačů pro 3. scénář

9.3 Shrnutí

Předchozími scénáři jsme otestovali chování implementace PIM-SM v různých situacích. Ze získaných dat je patrné, že PIM zprávy, které jsou v síti zasílány, jsou shodné a jsou doručovány ve stejném pořadí. Informace zaznamenané v simulačních multicastových tabulkách jsou téměř stejné jako ty z reálných zařízení. Liší se pouze názvy příchozích a odchozích rozhraní. V testovaných topologiích (reálné a simulační) si však tato rozhraní odpovídají.

U PIM časovačů jsme si ověřili, že vyprší ve stejném pořadí a přibližně ve stejných časech. Nepřesnosti, které vznikly, byly způsobeny zjednodušenou implementací IGMP. Jistou roli sehrál i fakt, že jednotlivé události v reálné síti bylo obtížné načasovat na přesně daný čas tak, jako v simulaci.

Těmito testy jsme ověřili, že implementace PIM-SM je funkční, správná a odpovídá chování Cisco implementace. Výše uvedené simulační příklady a příklad simulující dva zdroje multicastu a jejich příjemce jsou dostupné na přiloženém CD (příloha B).

Kapitola 10

Závěr

Práce se věnovala problematice multicastového směrování pomocí protokolu PIM-SM. Seznámila nás se samotným protokolem a s nástroji pro vizualizaci multicastu. Dále se práce zabývala simulačním nástrojem OMNeT++, knihovnami INET a ANSAINET a jejich využitím pro modelování a simulaci.

Cílem práce bylo rozšíření simulátoru o podporu multicastového směrování pomocí protokolu PIM-SM. Navržené rozšíření bylo implementováno do knihovny ANSAINET a bylo ověřeno na testovacích scénářích.

Vlastní přínos

Nejdříve jsme se obecně seznámili s multicastingem a poté jsme se zaměřili na samotný multicastový směrovací protokol PIM Sparse Mode. Díky podrobnému nastudování RFC jsme v práci objasnili složitou a rozsáhlou problematiku stavových automatů v PIM-SM. V části zabývající se protokolem PIM-SM jsme si navíc popsali používané zprávy a časovače v tomto protokolu.

Následně jsme se seznámili se simulačním nástrojem OMNeT++ a knihovnou INET a zjistili jsme jejich aktuální možnosti pro využití multicastového směrování. Původní knihovna ANSAINET již obsahovala podporu pro multicastové směrování, která se s příchodem nové verze INETu stala nefunkční a nepoužitelnou.

Dále jsme se seznámili s konfigurací multicastového směrování na Cisco zařízeních. Zaměřili jsme se zejména na konfiguraci protokolu PIM-SM a RP směrovačů. Ve školní laboratoři jsme si vyzkoušeli PIM-SM v reálném provozu a programem Wireshark jsme získali referenční data.

Teoretickou část práce jsme uzavřeli prozkoumáním problematiky detekce a vizualizace mutlicastových toků v síti.

Po získání potřebných informací jsme začali s návrhem na rozšíření simulátoru o podporu multicastového směrování v podobě protokolu PIM-SM. Návrh zahrnoval rozšíření již stávající struktury pro protokol PIM. Vytvořili jsme zjednodušené diagramy aktivit pro interní a externí události a zprávy, které popisovaly činnost protokolu PIM-SM.

Pro správné nakonfigurování směrovačů v simulační topologii bylo potřeba navrhnout rozšíření stávajícího konfiguračního souboru a provést s tím související úpravy v modulu Device Configurator.

Návrhem jsme se řídili při implementaci, které předcházela úprava a integrace původních zdrojových kódů do nové verze knihovny INET. Ta zahrnovala úpravy ve strukturách

podporujících multicastové směrování a v síťové vrstvě. V jazyce NED jsme vytvořili nový model PIM routeru doplněný o původní složený PIM model. Dále jsme vytvořili velké množství metod pro modul pimSM, které implementují chování samotného protokolu a zjednodušenou podporu pro IGMP. Částečně bylo potřeba upravit stávající implementaci modulu PIM Splitter, PIM zprávy a vytvořit nové PIM časovače pro PIM-SM.

Implementace byla ověřena na testovacích scénářích, které prověřili činnost PIM-SM v různých situacích. Výsledky získané ze simulace byly porovnány s daty získanými z reálné sítě sestavené z Cisco zařízení. Srovnání zachycených dat dokazuje, že je implementace funkční a správná.

Část práce, zabývající se integrací původní podpory pro multicastové směrování, byla publikována v [15] a představena na konferenci SIMUTools 2013.

Další vývoj

Díky nově vytvořeným třídám, které dědí implementaci z knihovny INET, bude nyní rozšiřování frameworku podstatně jednodušší. Zároveň by měly být přechody na nové verze INETu bezproblémové. To se ostatně ukázalo s nedávnou migrací na INET 2.1.0, který proběhl hladce, bez potřeby měnit názvy metod, či jinak zasahovat do kódu.

Velice užitečné by bylo rozšíření podpory pro protokol IGMP, který úzce spolupracuje s protokolem PIM při přihlašování příjemců k multicastu. Možností rozšiřování v rámci protokolu PIM je celá řada. Může se například jednat o implementaci nových módů (PIM-SSM, BiDir-PIM).

Možnosti pro rozšíření se nabízí i v samotném PIM-SM. Jednou z nich je implementace optimalizační fáze Shortest-Path Tree, při které směrovač připojený nejblíže k příjemci multicastu vytvoří nejkratší cestu přímo ke zdroji multicastu. Dále je možné implementovat zpracování PIM Assert zpráv. V tomto případě by také bylo nutné doimplementovat podporu multicastu na L2 vrstvě. Dalším možným rozšířením je dynamická volba RP směrovače a DR směrovačů.

Literatura

[1] PIM Sparse Mode [online]. 2001 [cit. 2013-02-18].
URL ftp://ftp-eng.cisco.com/ipmulticast/training/Module5.pdf

[2] EMC Ionix Multicast Manager [online]. 2009 [cit. 2012-06-23].
URL http://www.emc.com/collateral/software/data-sheet/s0004-ionix-multicast-ds.pdf

[3] NNM iSPI for IP Multicast 9.10 [online]. 2011-06-11 [cit. 2012-06-23].
URL http://www8.hp.com/us/en/software/enterprise-software.html

[4] HP Network Node Manager i software [online]. 2011 [cit. 2012-06-23].
URL http://www8.hp.com/us/en/software-solutions/\software.html?compURI=1170657

[5] Bhaskar, N.; Gall, A.; Lingard, J.; aj.: RFC 5059 - Bootstrap Router (BSR) Mechanism for Protocol Independent Multicast (PIM) [online]. [cit. 2012-10-07].
URL http://tools.ietf.org/html/rfc5059

[6] Danko, M.: Modelování směrovacích protokolů OSPF v simulátoru OMNeT++. Bakalářská práce, 2009.

[7] Fenner, B.; Handley, M.; Holbrook, H.; aj.: RFC 4601 - Protocol Independent Multicast - Sparse Mode (PIM-SM) [online]. [cit. 2012-05-18].
URL http://tools.ietf.org/html/rfc4601

[8] Keshav, S.: Project Octopus - Domain Topology Generation. 2010-10-05 [cit. 2012-06-23].
URL
http://www.cs.cornell.edu/cnrg/topology_aware/topology/topology.html

[9] Mateleško, P.: Simulování multicastových přenosů v simulátoru OMNeT++. Bakalářská práce, 2012.

[10] Rybová, V.: *Modelování multicastového směrování v prostředí Omnet ++*. Diplomová práce, FIT VUT v Brně, 2012.

[11] Varga, A.: *OMNeT++ User Manual*. 2011 [cit. 2012-011-01].
URL http://www.omnetpp.org/doc/omnetpp/Manual.pdf

[12] Varga, A.: *INET Framework for OMNeT++*. 2012 [cit. 2012-011-01].
URL http://inet.omnetpp.org/doc/INET/inet-manual-draft.pdf

[13] Veselý, V.: *Architektury systémů na Internetu se skupinovým adresováním.* Diplomová práce, FIT VUT v Brně, 2009.

[14] Veselý, V.: Multicast. Materiály do předmětu PDS, 2010 [cit. 2012-06-23].

[15] Veselý, V.; Ryšavý, O.; Švéda, M.: IPv6 Unicast and IPv4 Multicast Routing in OMNeT++. In *Proceedings of the IEEE 6th International ICST Conference on Simulation Tools and Techniques*, International Communication Sciences and Technology Association, 2013, ISBN 978-1-936968-47-3, s. 1–4.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10290

[16] WWW stránky: Configuring Cisco Discovery Protocol (CDP).
URL http://www.cisco.com/en/US/docs/ios/12_1/configfun/configuration/guide/fcd301c.html

[17] WWW stránky: IP-Multicasting Technology Part I: History and Overview.
URL http://www.intelligraphics.com/ip-multicasting-technology-part-i-history-and-overview

[18] WWW stránky: MibDepot.
URL http://www.mibdepot.com

[19] Černý, M.: *Modelování IPv6 v prostředí OMNeT++*. Diplomová práce, FIT VUT v Brně, 2011.

Seznam obrázků

2.1 Základní myšlenka multicastu 5
2.2 Mapování multicastové IP adresy na adresu MAC [13] 6

3.1 Znázornění fáze RP tree 10
3.2 Znázornění fáze Register-Stop 11
3.3 Znázornění fáze Shortest-Path Tree 12
3.4 Formát hlavičky zpráv PIM [7] 12
3.5 Formát PIM Hello zprávy [7] 13
3.6 Formát Register zprávy [7] 15
3.7 Formát Register-Stop zprávy [7] 15
3.8 Formát Join/Prune zprávy [7] 16
3.9 Formát Assert zprávy [7] 17
3.10 Stavový automat Register pro každý (S,G) [7] 18
3.11 Downstream (*,*,RP) stavový automat [7] 19
3.12 Downstream (*,G) stavový automat [7] 20
3.13 Downstream (S,G) stavový automat [7] 21
3.14 Downstream (S,G,rpt) stavový automat [7] 22
3.15 Upstream (*,*,RP) stavový automat [7] 23
3.16 Upstream (*,G) stavový automat [7] 24
3.17 Upstream (S,G) stavový automat [7] 25
3.18 Upstream (S,G,rpt) stavový automat [7] 26
3.19 Assert(S,G) stavový automat [7] 27
3.20 Assert(*,G) stavový automat [7] 31

4.1 Složené a jednoduché moduly [11] 35

6.1 Architektura progamu HP NNMi [4] 44

7.1 Architektura PIM modelu [10] 46
7.2 Zjednodušený diagram aktivit pro PIM-SM [10] 47
7.3 Diagram aktivit pro příjem interních zpráv a událostí 48
7.4 Diagram aktivit pro příjem externích zpráv 49

8.1 Modul ansaPIMRouteru s upravenými moduly 51
8.2 Složený model protokolu PIM 54
8.3 Sekvenční diagram popisující zaregistrování nového multicastového zdroje . 56
8.4 Sekvenční diagram znázorňující zpracování multicastových dat na DR směrovači zdroje 57

9.1 Testovací topologie společná pro všechny scénáře 62

9.2 Obsah multicastových směrovacích tabulek během simulace – scénář 1, 1. krok 63
9.3 Obsah multicastových směrovacích tabulek v reálné síti – scénář 1, 1. krok . 63
9.4 Obsah multicastových směrovacích tabulek během simulace – scénář 1, 2. krok 64
9.5 Obsah multicastových směrovacích tabulek v reálné síti – scénář 1, 2. krok . 64
9.6 Obsah multicastových směrovacích tabulek během simulace – scénář 1, 3. krok 65
9.7 Obsah multicastových směrovacích tabulek v reálné síti – scénář 1, 3. krok . 66
9.8 Obsah multicastových směrovacích tabulek během simulace – scénář 2, 1. krok 67
9.9 Obsah multicastových směrovacích tabulek v reálné síti – scénář 2, 1. krok . 68
9.10 Obsah multicastové směrovací tabulky během simulace – scénář 3, 4. krok . 70
9.11 Obsah multicastové směrovací tabulky v reálné síti – scénář 3, 4. krok . . . 70
9.12 Obsah multicastových směrovacích tabulek během simulace – scénář 3, 5. krok 71
9.13 Obsah multicastových směrovacích tabulek v reálné síti – scénář 3, 5. krok . 72

Seznam tabulek

3.1 Přehled hodnot pole Type [7] 12
3.2 Přehled hodnot a parametrů OptionType [7] 14

5.1 Přehled vybraných PIM-SM multicastových zpráv [13] 39

6.1 MIB definované IETF [18] 42
6.2 MIB zaměřené na multicast od různých výrobců [18] 43
6.3 SNMP traps pro zařízení Cisco [18] 43

9.1 Rozpis událostí pro 1. scénář 63
9.2 Rozpis událostí pro 2. scénář 66
9.3 Rozpis událostí pro 3. scénář 69
9.4 Srovnání Expirace časovačů pro 3. scénář 73

B.1 Obsah přiloženého CD 82

Příloha A

Seznam zkratek

ANSA = Automated Network Simulation and Analysis
BGP = Border Gateway Protocol
BSR = Bootstrap Router
CDP = Cisco Discovery Protocol
CGMP = Cisco Group Management Protocol
DNS = Domain Name System
DR = Designated Router
DVMRP = Distance Vector Multicast Routing Protocol
IGMP = Internet Group Message Protokol
LDP = Label Distribution Protocol
LLDP = Link Layer Discovery Protocol
M-BGP = Multiprotocol BGP
MFIB = Multicast Forwarding Information Base
MLD = Multicast Listener Discovery
MOSPF = Multicast Open Shortest Path First
MPLS = Multiprotocol Label Switching
MRIB = Multicast Routing Information Base
MSDP = Multicast Source Discovery Protocol
NED = Network Description
PIM = Protocol Independent Multicast
PIM-DM = PIM — Dense Mode
PIM-SM = PIM — Sparse Mode
PPP = Point-to-Point Protocol
RGMP = Router Gateway Management Protocol
RP = Rendezvous Point
RPF = Reverse Path Forwarding
RPT = Rendezvous Point Tree
SCTP = Stream Control Transmission Protocol
SNMP = Simple Network Management Protocol
SPT = Shortest-Path Tree
TIB = Tree Information Base

Příloha B

Obsah CD

V následující tabulce B.1 je uveden obsah přiloženého CD.

dip-xproch21.pdf	Elektronická verze práce ve formátu PDF
latex/	Zdrojový text technické zprávy
ANSAINET/	Zdrojové kódy knihovny ANSAINET, které byly použity pro tuto práci.
src/	Zdrojové kódy implementující PIM-SM (taktéž dostupné ve složce ANSAINET)
examples/	Testovací scénáře (taktéž dostupné ve složce ANSAINET)
doc/	Programová dokumentace vygerovaná proramem Doxygen
install/	Soubory pro instalaci OMNeT++ a INET

Tabulka B.1: Obsah přiloženého CD

Příloha C

Makra

Následující příloha obsahuje makra uváděná v RFC 4601 [7].

C.1 Makra pro výběr DR směrovače

```
host DR(I) {
    dr = me
    for each neighbor on interface I {
        if ( dr_is_better( neighbor, dr, I ) == TRUE ) {
            dr = neighbor
        }
    }
    return dr
}

bool dr_is_better(a,b,I) {
     if( there is a neighbor n on I for which n.dr_priority_present
             is false ) {
         return a.primary_ip_address > b.primary_ip_address
     } else {
         return ( a.dr_priority > b.dr_priority ) OR
                ( a.dr_priority == b.dr_priority AND
                  a.primary_ip_address > b.primary_ip_address )
     }
}
```

C.2 Makro CouldRegister(S,G) Register stavového automatu

```
bool CouldRegister(S,G)
{
    return ( I_am_DR( RPF_interface(S) ) AND
            KeepaliveTimer(S,G) is running AND
            DirectlyConnected(S) == TRUE )
}
```

C.3 Makro CouldAssert(S,G,I) pro Assert (S,G) stavový automat

```
CouldAssert(S,G,I) = SPTbit(S,G)==TRUE AND (RPF_interface(S) != I)
    AND (I in ( ( joins(*,*,RP(G)) + joins(*,G) - prunes(S,G,rpt) )
              + ( pim_include(*,G) - pim_exclude(S,G) ) - lost_assert(*,G)
              + joins(S,G) + pim_include(S,G) ) )
```

C.4 Makra JoinDesired pro příslušné stavové automaty

```
bool JoinDesired(*,*,RP)
{
   if immediate_olist(*,*,RP) != NULL
       return TRUE
   else
       return FALSE
}

bool JoinDesired(*,G)
{
   if (immediate_olist(*,G) != NULL OR
      (JoinDesired(*,*,RP(G)) AND
       AssertWinner(*, G, RPF_interface(RP(G))) != NULL))
         return TRUE
   else
         return FALSE
}

bool JoinDesired(S,G)
{
    return( immediate_olist(S,G) != NULL
        OR ( KeepaliveTimer(S,G) is running
             AND inherited_olist(S,G) != NULL ) )
}
```

C.5 Makra pro Upstream (S,G,rpt) stavový automat

```
bool PruneDesired(S,G,rpt)
{
    return ( RPTJoinDesired(G) AND
         ( inherited_olist(S,G,rpt) == NULL
           OR (SPTbit(S,G)==TRUE
           AND (RPF(*,G) != RPF(S,G)) )))
}

bool RPTJoinDesired(G)
{
    return (JoinDesired(*,G) OR JoinDesired(*,*,RP(G)))
}
```

Příloha D

Implementace protokolu PIM-SM

D.1 Implementace hlavičky třídy pimSM

```
class pimSM : public cSimpleModule, protected INotifiable
{
private:
  AnsaRoutingTable  *rt;     /**< Pointer to routing table. */
  IInterfaceTable   *ift;    /**< Pointer to interface table. */
  NotificationBoard *nb;     /**< Pointer to notification table. */
  PimInterfaceTable *pimIft; /**< Pointer to table of PIM interfaces. */
  PimNeighborTable  *pimNbt; /**< Pointer to table of PIM neighbors. */

  IPv4Address RPAddress;
  std::string SPTthreshold;

private:
  void receiveChangeNotification(int category, const cPolymorphic *details);
  void newMulticastRegisterDR(AnsaIPv4MulticastRoute *newRoute);
  void newMulticastReciever(addRemoveAddr *members);
  void removeMulticastReciever(addRemoveAddr *members);


  // process timers
  void processPIMTimer(PIMTimer *timer);
  void processKeepAliveTimer(PIMkat *timer);
  void processRegisterStopTimer(PIMrst *timer);
  void processExpiryTimer(PIMet *timer);
  void processJoinTimer(PIMjt *timer);
  void processPrunePendingTimer(PIMppt *timer);


  void restartExpiryTimer(AnsaIPv4MulticastRoute *route,
                          InterfaceEntry *originIntf, int holdTime);
  void dataOnRpf(AnsaIPv4MulticastRoute *route);
```

```
// set timers
PIMkat* createKeepAliveTimer(IPv4Address source, IPv4Address group);
PIMrst* createRegisterStopTimer(IPv4Address source, IPv4Address group);
PIMet*  createExpiryTimer(int intID, int holdtime, IPv4Address group,
                          IPv4Address source, int StateType);
PIMjt*  createJoinTimer(IPv4Address group, IPv4Address JPaddr,
                        IPv4Address upstreamNbr, int JoinType);
PIMppt* createPrunePendingTimer(IPv4Address group, IPv4Address JPaddr,
                                IPv4Address upstreamNbr, JPMsgType JPtype);

// pim messages
void sendPIMRegister(IPv4ControlInfo *ctrl);
void sendPIMRegisterStop(IPv4Address source, IPv4Address dest,
                         IPv4Address multGroup, IPv4Address multSource);
void sendPIMRegisterNull(IPv4Address multSource, IPv4Address multDest);
void sendPIMJoinPrune(IPv4Address multGroup, IPv4Address joinPruneIPaddr,
                      IPv4Address upstreamNbr, joinPruneMsg JoinPrune,
                      JPMsgType JPtype);
void sendPIMJoinTowardSource(multDataInfo *info);
void forwardMulticastData(multDataInfo *info);

// process PIM messages
void processPIMPkt(PIMPacket *pkt);
void processRegisterPacket(PIMRegister *pkt);
void processRegisterStopPacket(PIMRegisterStop *pkt);
void processJoinPacket(PIMJoinPrune *pkt, IPv4Address multGroup,
                       EncodedAddress encodedAddr);
void processPrunePacket(PIMJoinPrune *pkt, IPv4Address multGroup,
                        EncodedAddress encodedAddr);
void processJoinPrunePacket(PIMJoinPrune *pkt);
void processSGJoin(PIMJoinPrune *pkt,IPv4Address multOrigin,
                   IPv4Address multGroup);
void processJoinRouteGexistOnRP(IPv4Address multGroup,
                                IPv4Address packetOrigin, int msgHoldtime);
```

```
public:
  //methods for PIM-SM
  void setRPAddress(std::string address);
  void setSPTthreshold(std::string address);
  IPv4Address getRPAddress () {return RPAddress;}
  std::string getSPTthreshold () {return SPTthreshold;}
  virtual bool IamRP (IPv4Address RPaddress);
  bool IamDR (IPv4Address sourceAddr);
  IPv4ControlInfo *setCtrlForMessage (IPv4Address destAddr,
                                      IPv4Address srcAddr,
                                      int protocol, int interfaceId,
                                      int TTL);

protected:
  virtual int numInitStages() const  {return 5;}
  virtual void handleMessage(cMessage *msg);
  virtual void initialize(int stage);
};
```