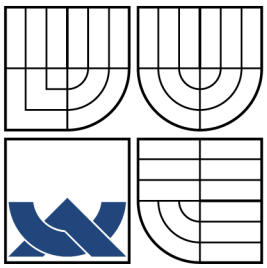
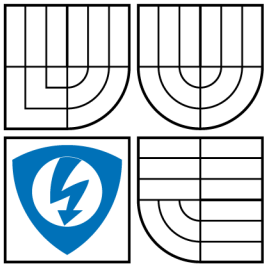


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SPRÁVA SÍTÍ S MOBILE IP POMOCÍ SNMP

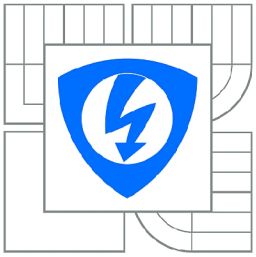
NETWORK MANAGEMENT OF MOBILE IP USING SNMP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. ONDŘEJ ZÁVODNÝ

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. MICHAL SKOŘEPA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Ondřej Závodný	***ID:***	72763
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Správa sítí s Mobile IP pomocí SNMP

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte problematiku mobility uzlů v protokolu Mobile IPv4. Sestavte testovací síť tvořenou směrovači firmy Cisco a implmentujte v ní podporu Mobile IPv4. Směrovače nastavte na podporu SNMP pro protokol Mobile IPv4. V jazyce JAVA naprogramujte aplikaci, která bude prostředníctvím SNMP spravovat jednotlivé agenty v MIPv4. Aplikace by měla sledovat stav agentů, umožňovat nastavování a také monitorovat aktivitu mobilních uzlů, které se v rámci sítě pohybují.

DOPORUČENÁ LITERATURA:

[1] RAAB, Stefan. Cisco: Mobilní IP technologie a aplikace, Grada, 2007, 299 s., ISBN: 978-80-247-1611-4.
[2] Perknis D. T. Understanding SNMP MIBs, Prentice Hall, 1996, 528 s., ISBN: 978-0134377087.

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Michal Skořepa

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

Obsahem této diplomové práce je prozkoumání možností využití protokolu SNMP pro správu aktivních prvků, které poskytují služby protokolu Mobile IP.

První dvě kapitoly se zabývají teoretickým rozborem protokolu Mobile IP a Simple Network Management Protokolu. V protokolu Mobile IP se rozbor zaměřuje především na prvky domovský agent(Home agent) a cizí agent(Foreign agent). U SNMP protokolu se práce zaměřuje nejvíce na jednotlivé způsoby výměny zpráv.

Další část dokumentu je věnována popisu sestavené laboratorní sítě ze směrovačů Cisco 1841 a jejich konfiguraci. Nakonec je popsána naprogramovaná aplikace pro správu těchto zařízení v jazyce JAVA, která je součástí přílohy. Ta pomocí periodicky zasílaných SNMP dotazů zjišťuje stavy vazeb mezi domovským agentem a mobilními uzly, dále pak seznam návštěvníků cizího agenta. Aplikace má uživatelsky přívětivé rozhraní, které je znázorněno na přiložené flashové animaci. Závěrem práce jsou shrnuty popsané koncepty z předchozích kapitol a cíle, kterých se dosáhlo.

KLÍČOVÁ SLOVA

SNMP, JAVA, Mobile IP, správa, Cisco

ABSTRACT

The content of this thesis is to research possibilities of using SNMP for management of Mobile IP activated entities.

The first two chapters deal with theoretical analysis of Mobile IP protocol, and Simple Network Management Protocol. The Mobile IP protocol analysis focuses especially on the elements Home agent and Foreign agent. In the SNMP chapter the analysis focuses mainly on ways to exchange messages.

Another part of the document is devoted to a description of laboratory networks composed of routers Cisco 1841 and the configuration of them. Finally is described the programmed application to manager these devices in JAVA, which is included in Annex. The program periodically sends SNMP queries and finds relationships between the Home agent and the mobile nodes, and between the foreign agent and the mobile nodes. The application has a user-friendly interface that is shown on the attached flash animation. Finally, the thesis summarizes the concepts described in previous chapters and the goals to achieve.

KEYWORDS

SNMP, JAVA, Mobile IP, management, Cisco

ZÁVODNÝ, Ondřej *Správa sítí s Mobile IP pomocí SNMP*: diplomová práce. BRNO: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 60 s. Vedoucí práce byl Ing. Michal Skořepa

Poděkování

Děkuji vedoucímu diplomové práce Ing. Michalovi Skořepovi za pomoc a cenné rady při zpracování mé diplomové práce.

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

V dnešní době se již bere jako samozřejmost, že osobní počítače a spousta dalších zařízení mezi sebou v reálném čase komunikují. V současnosti jsou, ale osobní počítače kvůli pevné vazbě s kabeláží vytlačovány menšími a přenosnými zařízeními – notebooky, netbooky ba dokonce i menšími zařízeními jako kapesní počítače, nebo chytré telefony. Ovšem čím menší zařízení vlastníme, tím větší cítíme potřebu mobility.

Jenže co je to mobilita? První slovo co mě napadne v souvislosti s mobilitou je Internet. V dnešní době opravdu velký fenomén, každý jej zná a využívá. Při pohledu z vesmíru, se sice tak velký a mocný nezdá, ale pokud zůstaneme na naší rodné Zemi, chceme být v rámci Internetu mobilní.

Tuto mobilitu si můžeme vyložit dvěma způsoby. Na první způsob jsme asi všichni zvyklí. Dojdeme do veřejné knihovny, zapneme svůj přenosný počítač, vyhledáme dostupnou bezdrátovou síť, dostaneme IP adresu a prostřednictvím místního směrovače jsme připojeni k Internetu. Autor knihy Cisco: Mobilní IP a technologie Madhaví W. Chandra člověka, který se takto připojuje, nazývá „kočovník“. Tento způsob má své nevýhody v tom, že v každé jiné lokalitě se nám mění IP adresa, pod kterou vystupujeme v síti Internet. Navíc je tato IP adresa velmi často sdílená více uživateli. Právě tady vzniká prostor pro protokol Mobile IP. Pokud potřebuji, abych byl dostupný pod jednou konkrétní IP adresou, použiji protokol Mobile IP. Tuto potřebu mají často protokoly pro videokonference nebo VoIP jako jsou SIP nebo H.323, které se potýkají s problémy pokud se nacházíme za firewallem, nebo naše vnitřní privátní adresa je překládána na společnou veřejnou adresu prostřednictvím metody NAT overloading. Protokol Mobile IP je tedy, jak říká definice protokolem dynamického směrování, jehož koncová zařízení signalizují vlastní aktualizace trasy směrování a jehož dynamické tunely odstraňují potřebu šíření trasy k hostiteli.[1]. Vlastními slovy bych protokol Mobile IP charakterizoval jako možnost pohybovat se v rámci sítě a být neustále dostupný pod stejnou IP adresou. Protokol Mobile IP, vyžaduje určitou připravenost sítě poskytovat tyto služby. Tato problematika je nastíněna v kapitole 1.

Protože, mě tato tématika úzce zajímá, zvolil jsem si téma diplomové práce

„Správa sítí s Mobile IP pomocí SNMP“ pod vedením Ing. Skořepy. Pro potřeby práce bude sestavena laboratorní počítačová síť tvořená směrovači renomované firmy Cisco a běžnými počítači. Zapojení směrovačů je důkladně popsáno v kapitole 3.

Jelikož v současnosti, není na trhu dostupný software, který by spravoval prvky Mobile IP, bude pro tento účel vytvořena aplikace v jazyce JAVA. Ta bude pomocí SNMP dotazů spravovat dané prvky. Taková aplikace se dá obecně nazvat jako SNMP manager. Aby mohla výměna zpráv probíhat, musí se na druhé straně, na směrovačích Cisco, zprovoznit funkce SNMP agenta, který bude na SNMP dotazy od managera odpovídat. Aplikace má za úkol takto získaná data vhodně zobrazit a dynamicky reagovat na změny v síti.

1 MOBILE IP

Protokol Mobile IP je protokolem mobilní technologie. Tato technologie bývá často spojována s bezdrátovou technologií, ale nejsou to synonyma. Bezdrátové sítě pouze umožňují mobilním uživatelům(uzlům sítě) pohybovat se napříč sítí bez potřeby připojení prostřednictvím kabelů. To ale neznamená, že by se pevného připojení nedalo vůbec využít. Naopak této možnosti bylo využito v laboratorní síti, protože tak lze přesně určit mobilnímu uzlu, kde se má nacházet. S bezdrátovou technologií by tato úloha bylo mnohem složitější, zároveň by neměla žádný vliv na logickou strukturu sítě[1].

Protokol Mobile IP (MIP) byl stejně jako ostatní internetové protokoly vyvíjen a schvalován sdružením Internet Engineering Task Force (IETF) od roku 1996 a v současnosti je popsán v dokumentu RFC 3344. Doplňující funkce pro zabezpečení MIP jsou popsány v dokumentu RFC 4721. V následujících kapitolách bude tento protokol podle uvedených dokumentů stručně popsán.

1.1 Mobile IP v modelu OSI

Referenční mode Open Systems Interconnection (OSI) je obecně známý model, při popisování komunikace v dnešních síťích. Skládá ze sedmi vrstev a komunikace pak probíhá na odesílající straně od nejvyšší vrstvy směrem ke nejnižší vrstvě následně prostřednictvím komunikačního kanálu se přenese k přijímací straně a přechodem od nejnižší vrstvy k nejvyšší přenos zprávy končí.

Ze znalosti tohoto modelu vyplývá, že nejlepším umístněním pro protokol Mobile IP je třetí vrstva, nazývaná síťová. Na této vrstvě je v současnosti dominantní protokol IPv4, přestože společnost IANA, která spravuje rozdělování IP adres, všechny veřejné IP adresy rozdala.

Díky tomu, že protokol Mobile IP pracuje na této vrstvě, mohou aplikace fungující na vyšších vrstvách pracovat, bez toho aniž by poznaly jak se logická struktura na síťové vrstvě mění. Analogicky toto funguje směrem k nižším vrstvám, je úplně jedno zda použijeme pevné sítě Ethernet nebo některou z dostupných bezdrátových

sítí. Pokud ale použijeme bezdrátové technologie, dokáže protokol Mobile IP s linkovou vrstvou úzce spolupracovat a předávání mezi jednotlivými přístupovými body se stává mnohem rychlejší a může vézt až k naprosto plynulému přechodu mezi sítěmi[1].

1.2 Základní prvky MIP

Mobile IPv4 obecně pracuje nad směrovacím protokolem, nejlépe ze skupiny Link-state např. Open shortest path first (OSPF) do kterého se redistribuuje. Mezi základní prvky patří[1]:

- Domovský agent (Home agent)
- Cizí agent (Foreign agent)
- Mobilní uzel (Mobile node)

V následujících kapitolách budou tyto prvky stručně popsány.

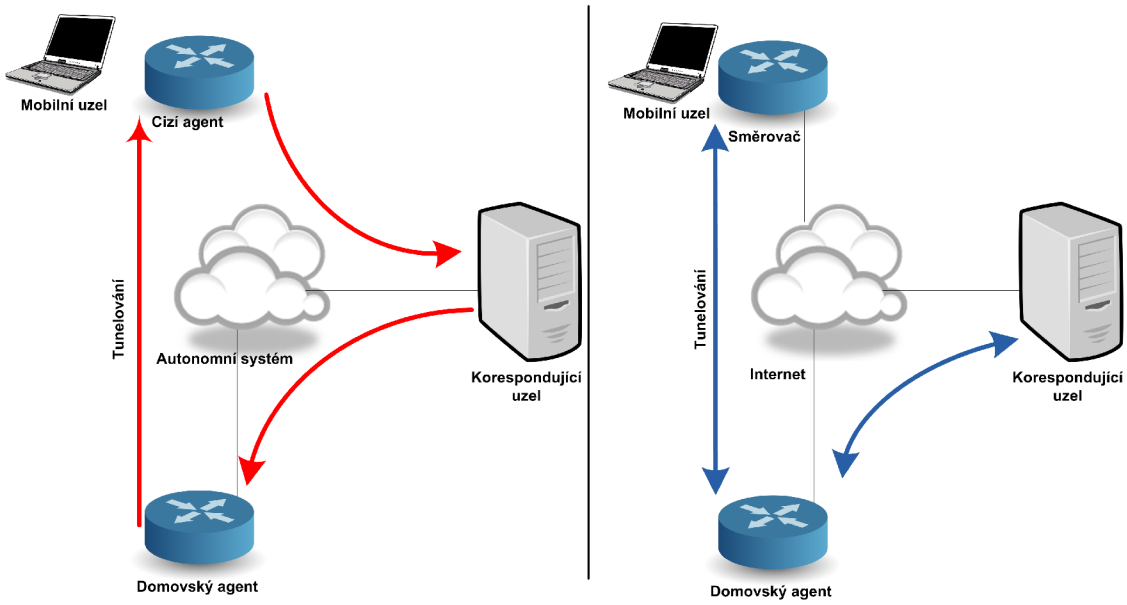
1.2.1 Domovský agent – Home agent

Domovský agent neboli Home agent (HA) je stěžejním prvkem protokolu MIP. Tento prvek definuje domovskou síť, kde jsou mobilní uzly doma, udržuje jejich seznam. Dále domovský agent musí definovat jaká autentizace se bude používat s daným mobilním hostem. Na výběr je hexadecimální klíč o délce 32 znaků pro menší počet uživatelů. Pro velké počty uživatelů je vhodnější využít AAA serveru(autentizace, autorizace, accounting)serveru nazývaného Remote Authentication Dial In User Service (RADIUS) [3].

Mobile IP funguje tak, že všechen provoz pro mobilní uzel je nejprve směrován do domovské sítě, kterou spravuje domovský agent. Ten podle toho kde se mobilní uzel nachází rozhodne, kam se provoz přesměruje. Pokud se mobilní uzel nenachází v domácí síti vytvoří tunel směrem k cizímu agentovi u kterého je mobilní uzel připojen. Tímto tunelem se pak přenáší veškerý provoz určený pro mobilní uzel. Provoz ve směru od mobilního uzlu může putovat dvěma způsoby. Buď tunelováním zpět k domovskému agentovi, nebo přímo ke korespondujícímu uzlu. V druhém

[1]Prvek Correspondent node není uveden, protože Mobile IP vůbec nemusí podporovat

případě vzniká tzv. trojúhelníkové směrování, které lze realizovat pouze v rámci autonomního systému[2]. Situace je znázorněna na obr. 1.1.



Obr. 1.1: Vlevo trojúhelníkové směrování, vpravo zpětné tunelování

Hlavní důvod, proč mobilní uzel jednoduše neuvědomí korespondující uzel o svém novém umístnění a nevytvoří symetrickou směrovací trasu, lze popsat jediným slovem – zabezpečení. Způsoby zabezpečení jsou popsány v kapitole 1.3[1].

1.2.2 Cizí agent – Foreign agent

Cizí agent FA je směrovač, který zprostředkovává mobilní služby pro mobilní uzel, které se nachází v jeho dosahu. Základní funkcí je předávání registračních žádostí mezi mobilním uzlem a domovským agentem. Dále poskytuje tzv.Care of Address (CoA), která představuje logické umístnění mobilního uzlu v síti. Po úspěšné registraci sestavuje mobilním uzlům tunel, kterým se přenáší provoz mezi daným mobilním uzlem a domovským agentemMN[2].

1.2.3 Mobile node – mobilní uzel

Tento prvek představuje účastnické zařízení, které může přecházet mezi různými sítěmi a je přitom dostupné přes svou domovskou adresu. Pokud je mobilní uzel

připojen u domovského agenta protokol MIP se při směrování paketů neprojeví. Teprve až se připojí k jiné síti zjišťuje zda se v ní nachází cizí agent, který mu zprostředkuje spojení s domovským agentem prostřednictvím CoA. Tu zjistí z pravidelně vysílaných `agent advertisement` na rozhraní kde cizí agent poskytuje MIP služby. Tyto oznámení jsou šířeny ICMP Router Discovery Messages (IRDP) protokolem všesměrově. Pokud se v síti cizího agenta nenachází, tak se mobilní uzel pokusí sám navázat tunel s domovským agentem prostřednictvím Co-located Care of Address (CCoA)[2][7].

1.3 Zabezpečení v protokolu Mobile IP

Aby si mobilní uzel a korespondující uzel mohly během procesu registrace protokolu Mobile IP vyměňovat informace o tom, kde se mobilní uzel nachází, musí mezi nimi existovat vztah důvěry. V opačném případě by byla komunikace mezi uzly sítě jednoduše napadnutelná útoky typu DoS, dle mého názoru i MitM. Uzel útočníka by mohl posla korespondujícímu uzlu falešnou registrační zprávu s falešnou adresou CoA a snadno by unesl navázané spojení. Musel by se použít nějaký systém např. certifikační autority, která by poskytovala důvěrnost mezi uzly. To by vyžadovalo i implementaci na straně korespondujícího uzlu. Proto protokol Mobile IP tento problém řeší tak, že mobilní uzel se registruje k domovskému agentu. Vztah důvěrnosti je tedy mobilní uzel – domovský agent. Provoz od korespondujícího uzlu je standardně směrován nejprve do domovské sítě a teprve domovský agent rozhodne kam dál bude směrován. Provoz od mobilního uzlu směrem ke korespondujícímu uzlu jde již přímo. Díky tomu vzniká typické, pro protokol Mobile IP, trojúhelníkové směrování.

Směrování protokolu IP z jeho definice používá cílovou adresu a nezohledňuje zdrojovou. Protokol Mobile IP tohoto faktu využívá pro optimalizaci doručování provozu. Mobilní uzel nacházející se v cizí síti tak může jako zdrojovou adresu použít svou domovskou adresu aniž by se v domovské síti nacházel.

Metody zabezpečení jsou v tomto protokolu určeny pouze k ochraně řídícího provozu, jmenovitě žádosti o registraci a potvrzení registrace. Nejdůležitějším úko-

lem je správně ověřit totožnost a bránit se proti útoku přehrání již proběhlé relace. Pro ověření totožnosti se využívá algoritmu HMAC definovaného v dokumentu RFC 2014. Pro nejjednodušší konfiguraci přidružení zabezpečení je možné využít kontextu identifikovaného hodnotou Security Parametr Index (SPI) s bezpečnostním klíčem. Toto přidružování je ale pro konfiguraci celkem složité a náročné na údržbu(klíče jsou v konfiguračních souborech směrovačů uchovávaný nešifrovaně), proto je v případě nasazení v praxi vhodnější využít serveru AAA. Jako metoda pro zabezpečení před přehráním proběhlé relace se používá metoda časového razítka `timestamp`[1].

1.4 Správa domovské adresy mobilních uzlů

V systému IOS je podpora možná správa přidělován domovské adresy třemi způsoby:

- Statické přidělování adresy bez identifikátoru Network Access Identifier (NAI)
- Statické přidělování adresy s identifikátorem NAI
- Dynamické přidělování adres s identifikátorem NAI

V práci bude využito první metody, a proto bych její podstatu ve zkratce nastínil. Statické adresy se předem v mobilních uzlech a jsou jejich jednoznačnými identifikátory. Tím pádem jsou vlastníky pevně dané dosažitelné IP adresy, což je velkou výhodou. Každá výhoda má i svou nevýhodu a tady se skrývá v malé škálovatelnosti. Je zapotřebí ve všech mobilních uzlech předem nakonfigurovat domovské adresy[1]. Tuto adresu pak může používat pouze jeden mobilní uzel – plýtvání adresním prostorem. Pro malou síť jako je laboratorní, je však tato metoda plně vyhovující.

1.4.1 Domovská síť

Všechny domovské adresy bez ohledu na způsob přidělování adresy musí být přidruženy k domovské síti, která je přímo připojena k domovskému agentovi. Domovskou síť tedy tvoří prefix, společný pro mobilní uzly. Existují dva způsoby jak v systému IOS domovskou síť definovat:

- Fyzická síť přidružená k jednomu z rozhraní
- Virtuální síť

Obecně se předpokládá, že mobilní uzel bude připojen k fyzické síti. Tato možnost se nejvíce využívá, když mobilní uzel většinu času tráví v domovské síti. Tato metoda má ale jednu nevýhodu, pokud se mobilní uzel nachází mimo domácí síť a rozhraní domovského agenta z nějakého důvodu selže, tak se mobilní uzel nemůže k domovské síti registrovat. Pokud ale předpokládáme, že většinu času bude mobilní uzel mimo domovskou síť, je vhodné jej přiřadit do tzv. virtuální sítě. Tato síť má výhodu v tom, že je vždy dostupná, nezávisí na žádném rozhraní. O uzlu, který se nikdy nenachází v domovské síti říkáme, že je uzel s trvalým roamingem. Ve skutečném nasazení mohou kvůli tomu vzniknout určité problémy se sémantikou. Mobilní uzel může být připojen u domovského agenta, ale bude jej využívat, jako by byl v cizí síti. Virtuální síť lze přirovnat k rozhraní zpětné smyčky a v systému IOS se konfiguruje tímto způsobem:

```
ip mobile virtual-network prefix maska
```

Po nadefinování této virtuální sítě, lze přiřadit jednotlivé mobilní uzly do této sítě následujícím příkazem:

```
ip mobile host address virtual-network prefix maska
```

V případě, že použijeme přiřazení k fyzické síti je třeba použít tohoto příkazu:

```
ip mobile host address interface název
```

Nakonec je potřeba vytvořit povinné přidružení zabezpečení mezi domovským agentem a mobilnímy uzly:

```
ip mobile secure host address spi index_spi key hex řetězec klíče
algorithm název_algoritmu
```

Tím je konfigurace přidružení ze strany domovského agenta dokončena[1].

2 SIMPLE NETWORK MANAGEMENT PROTOCOL

V dnešní době je často náš životní styl ovlivněn Internetem. Stáváme se závislými na připojení k internetu, firmy při výpadku připojení přichází o nemalé peníze. V těchto případech potřebujeme nějakým způsobem dohlížet na chod prvků, které nás spojují - směrovače, přepínače atd. Takové sítě jsou již v praxi velmi rozsáhlé, což vede k jejich nepřehlednost. Porucha, nebo přetížení jednoho prvku může často znamenat kolaps. Právě proto již v roce 1988 vznikl protokol Simple Network Management Protocol (SNMP) definovaný v RFC 1067, který měl za cíl vzdáleně spravovat prvky sítě a neustále nad nimi dohlížet. O 2 roky později byl dokument RFC 1067 aktualizován dokumentem RFC 1157.
Díky škálovatelnosti tohoto protokolu lze s jeho pomocí spravovat vše od výkonných Cisco přepínačů a směrovačů až po mikrovlnou troubu vybavenou Wi-Fi adaptérem. Jedinou podmínkou je, aby zařízení pracovalo nad IP protokolem. Hranice využitelnosti protokolu, jsou téměř nekonečné, především ale jde o sbírání statistik nebo stavových informací ze síťových zařízení. Dále umožňuje i vzdáleně daná zařízení konfigurovat. Tato výměna zpráv probíhá následovně – SNMP manager se dotazuje na konkrétní hodnoty v databázi a agent pokud zná odpověď odpovídá. Další možností výměny informací, jsou varovné zprávy trap. Tyto zprávy zasílá agent bez výzvy managera, pokud zaznamená nastavenou kritickou situaci.

Ve zkratce se tedy jedná o aplikační protokol, pomocí kterého se zjišťují nebo nastavují hodnoty agentovy databáze MIB.

2.1 Architektura SNMP

SNMP pracuje na architektuře agent/manager. Síťová zařízení, která sledujeme, představují agentskou část architektury. Jejich počet může být teoreticky neomezen. Aplikace, která běží na takovém zařízení se nazývá SNMP agent. Sledující zařízení je v síti typicky jen jedno, ale může jich být i více na sobě nezávislých. Program běžící na takovém zařízení se nazývá SNMP manager[4][5]. Tento program může

být zabudován v operačním systému aktivního prvku, nebo běžet jako samostatná aplikace na osobním počítači. Naprogramování takové aplikace v jazyce JAVA bude cílem této práce.

2.2 Verze SNMP

Protokol SNMP v současnosti existuje ve 3 verzích, které se liší hlavně podporovanými funkcemi, ale cíl mají stejný. Nejvíce se využívá verze 2c, která oproti verzi 1 podporuje zprávu typu GetBulk. Tento typ zprávy umožňuje v jednom paketu přenést více než jednu dotazovanou hodnotu. SNMP verze 3 zavádí podporu šifrování. Srovnání rozdílů je na tabulce 2.1.

Tab. 2.1: Hlavní rozdíly ve verzi SNMP

	SNMPv1	SNMPv2c	SNMPv3
Get	X	X	X
Set	X	X	X
GetNext	X	X	X
Trap	X	X	X
GetBulk		X	X
Šifrování			X

2.3 Management Information Base

Spravované objekty(hodnoty) jsou dostupné přes virtuální úložiště, známé jako Management Information Base (MIB). Objekty v MIB jsou definovány jazykem Structure of Management Information (SMI), který využívá základu z jazyka Abstract Syntax Notation One (ASN.1). Z takto definovaných objektů se vytvářejí MIB ve stromové struktuře. Databáze je tak logicky rozdělena do přehledných částí.

Takto definované databáze jsou normalizovány, schvalovány a dokumentovány v jednotlivých RFC např RFC 2233, v případě potřeby i společnosti jako je Cisco

mohou vytvářet své vlastní MIB. V současnosti jich existuje nespočetné množství. Protože cílem práce bude vytvoření aplikace pro správu Mobile IP budu se zabývat pouze MIP-MIB definované v RFC 2006. Dále budu využívat MIB definované společností Cisco CISCO-MOBILE-IP-MIB. Definice je dostupná na této adrese „ftp://ftp.cisco.com/pub/mibs/v2/CISCO-MOBILE-IP-MIB.my“ a je i na přiloženém CD.

2.4 Object Identifier

Každá instance libovolného objektu definovaného v MIB má 2 identifikátory: číselný a slovní. Číselný identifikátor umožňuje hierarchicky uspořádávat objekty v MIB. Je tvořen číslicemi oddělenými tečkou. Protože je takto zapsaný identifikátor pro člověka špatně čitelný, ke každému číselnému identifikátoru přiřazujeme slovní identifikátor. Díky němu jsme si schopni jednoduše odvodit jakou hodnotu daný objekt představuje. Např. OID 1.3.6.1.2.1.1.5 představuje `sysName` známé jako hostname. Pokud hodnotu potřebujeme přímo vyčíst musíme na konec OID přidat .0 jako číslo instance[1]. Dále je patrné, že i textové výrazy vykazují určitou hierarchii. Např. objekt `ifNumber` patří do databáze `IF-MIB` a označuje počet síťových rozhraní na daném zařízení. Příklad zápisu MIB objektu v MIB databázi:

```
ifNumber  OBJECT-TYPE
 SYNTAX      Integer32
 MAX-ACCESS  read-only
 STATUS      current
 DESCRIPTION
  "The number of network interfaces (regardless of their
   current state) present on this system."
 ::= { interfaces 1 }
```

[1] sysName má pouze 1 instanci

2.5 Výměna zpráv

Výměna zpráv probíhá asynchronně, tzn. bez ohledu na stav prvků v síti. SNMP manager se dotazuje SNMP agenta a ten odpovídá. Každý dotaz může obsahovat žádost i na více než jeden objekt, velikost dotazu je omezena pouze maximální podporovanou velikostí zprávy. Jednotlivé údaje dotazu nazýváme variable binding, což je spojení OID a návratové hodnoty daného objektu. Dotazy mohou být následujícího typu[6]:

GetRequest

Zpráva typu GetRequest je generována aplikací SNMP manager a následně odeslána na požadovanou cílovou adresu SNMP agenta. Jedná se o nejjednodušší možný dotaz, využívá se hlavně pro čtení objektů, které mají pouze jednu instanci.

GetNextRequest

Tato zpráva je stejně jako předešlá generována SNMP managerem. Nevrací však hodnotu objektu, ale OID následujícího objektu. S kombinací zprávy GetRequest je možné tedy v kombinaci s dotazy typu GetRequest procházet MIB databázi. Tento způsob se však jeví jako nevhodný pro velké množství dotazů a odpovědí. Z tohoto důvodu SNMPv2 zavádí dotaz typu GetBulk.

GetBulk

Díky této zprávě lze přenášet velké množství informací. Při použití této zprávy je nutné nastavit hodnoty non-repeaters a max-repetitions. První jmenovaná hodnota udává kolik objektů se má považovat za skalární, tedy vyčítat pouze metodou GetRequest. Druhá hodnota říká, kolikrát se má maximálně použít metoda GetNextRequest. Celkový počet odpovědí lze spočítat ze vzorce N + M x R, kde N je non-repeaters, M je max-repetitions a R = Celkový počet proměnných OID v dotazu mínus N. Mějme tedy dotaz o deseti proměnných OID, z toho tři jsou typu non-repeaters a z ostatních chceme vyčíst maximálně 4 řádky. Po dosazení do vzorce 3 + 4 x (10-3) dostáváme 31 údajů.

SetRequest

SNMP kromě získávání informací z agentů, umožňuje vybrané objekty vytvářet nebo modifikovat. K tomuto účelu slouží SetRequest. Pokud by se v praxi této možnosti využívalo, silně doporučuji z bezpečnostního hlediska využít SNMPv3, které umožňuje šifrování a autentizaci.

Response

Jak název napovídá, jedná se o odpověď agenta managerovi. Odpovědi se zasílají ve formě variable binding, tzn. vrací se OID společně s hodnotou.

Trap

Tuto zprávu vytváří agenti pokud zaznamenají nastavenou událost. Pokud se využívá, správa sítě se stává mnohem dynamičtější, protože manager na tuto zprávu může okamžitě reagovat.

2.5.1 Formát zprávy

Zprávy se zapisují a přenášejí strukturované podle anotace ASN.1. Ta využívá kódování Basic Encoding Rules (BER). Obecný formát zprávy pak vypadá takto[6]:

```
PDU ::= SEQUENCE {
        request-id INTEGER (-214783648..214783647),
        error-status    -- v některých případech se neuvádí
                INTEGER {
                        noError(0),
                        ...
                },
        error-index     -- v některých případech se neuvádí
                INTEGER (0..max-bindings),
        variable-bindings
                VarBindList
}
```

2.5.2 Přenosový protokol

Pro přenos dat mezi entitami se využívá UDP protokol. Pro SNMP jsou vyhrazeny porty 161 a 162. Port 161 se využívá jako zdrojový při odeslání dotazu, nebo jako cílový při odesílání odpovědí. Port 162 je vyhrazen pro SNMO managera pro příjem trapů.[6]

2.5.3 Chybové zprávy SNMP protokolu – výjimky

V určitých situacích může dojít k tomu, že SNMP agent nezná odpověď a proto je potřeba vědět jak takové *výjimky* ošetřit. Pokud by agent vrátil pouze prázdnou hodnotu, nevěděli bychom, zda je prázdná, nebo nastala *výjimka*. Díky implementaci *výjimek* se také nemůže stát, že by agent zkolaboval při dotazu na neznámý objekt. Seznam nejčastějších výjimek podle RFC 3416 je uveden v tabulce 2.2.[6] Daná výjimka je oznamována v poli `error-status` pouze svým indexem z tabulky(2.2) a pole `error-index` udává, pro kterou hodnotu z `VarBindList` je výjimka oznamována.

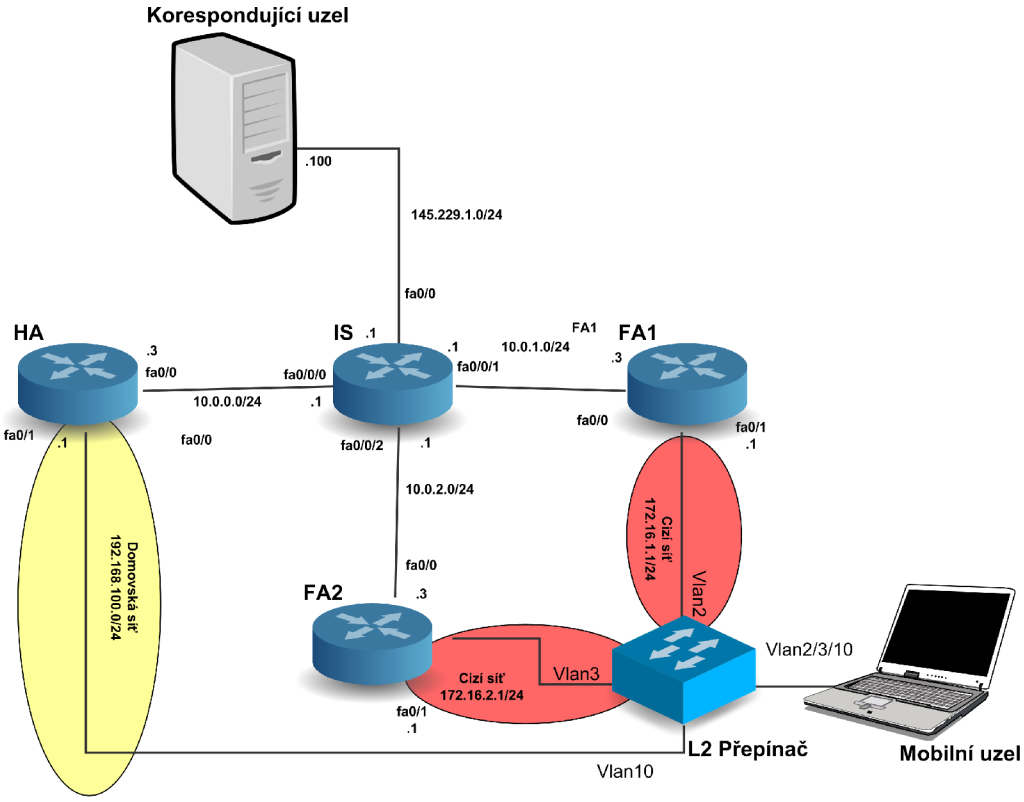
Tab. 2.2: Nejčastější výjimky

Název	Index	Význam
noError	0	žádná výjimka nenastala
tooBig	1	velikost PDU odpovědi je příliš velká pro přenos
noSuchName	2	název dotazovaného objektu nebyl nalezen
readOnly	4	byl zaslán příkaz Set na hodnotu „read only“
authorizationError	16	špatná autorizace

3 VYPRACOVÁNÍ

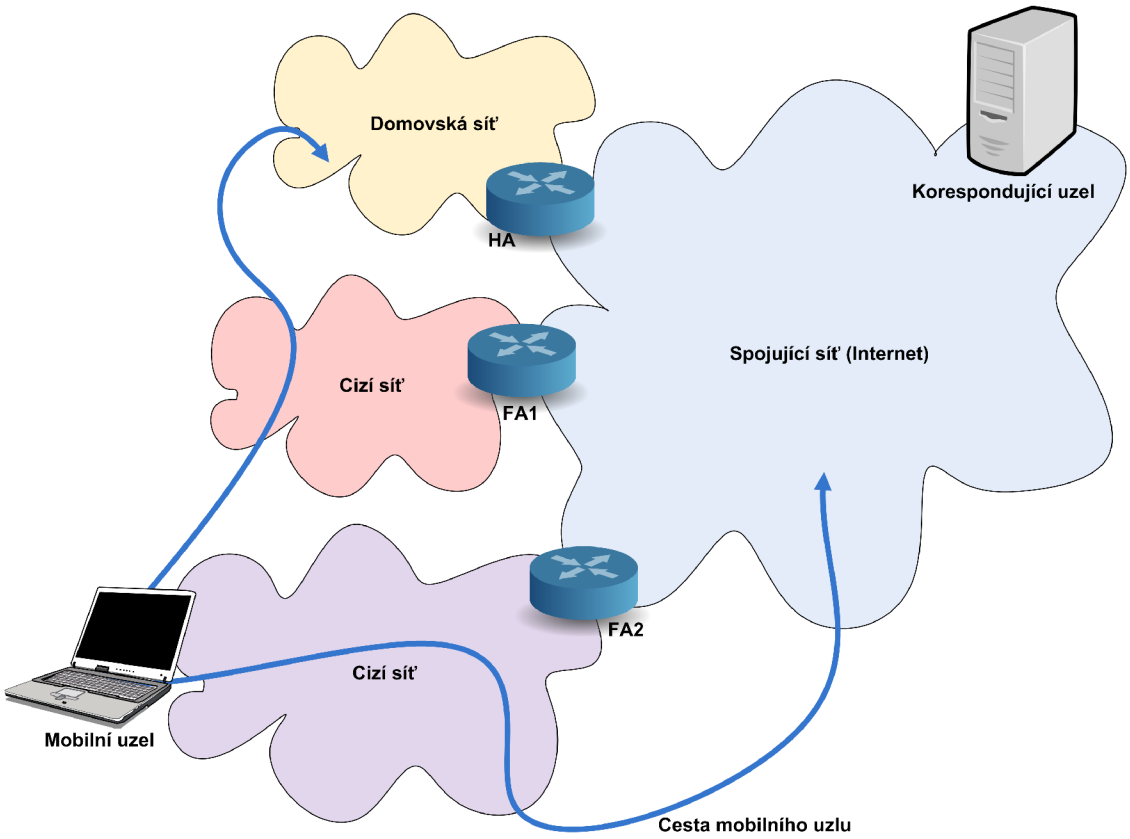
3.1 Testovací síť

Testovací síť pro účel mé práce bude tvořena směrovači od renomované firmy Cisco modelové řady 1841 s verzí IOS 12.4. Navržená fyzická topologie je znázorněna na obrázku 3.1. V praxi je ovšem toto zapojení samozřejmě nesmyslné. V laboratoři má pouze za úkol simulovat logické zapojení, znázorněné na obrázku 3.2.



Obr. 3.1: Fyzická topologie laboratorní sítě

Jsou zde celkem 4 směrovače, 1 přepínač a 2 běžné počítače. Sítě začínající prefixem 10.0.x.x představují internetové sítě, 192.168.x.x domovskou síť a 172.16.x.x jsou cizí sítě. Kompletní nastavení jednotlivých směrovačů je uvedeno v příloze. V následujících kapitolách budou uvedeny nejdůležitější kroky pro nastavení.



Obr. 3.2: Logická topologie laboratorní sítě

3.1.1 Internetové řešení – Internet solution

Prostřední směrovač s názvem Internet solution (IS) velice zjednodušeně simuluje internetové prostředí. Je na něm spuštěn zvolený směrovací protokol OSPF, který zaručuje velice rychlou konvergenci a adaptabilitu sítě. Dále poskytuje připojení pro CN. U tohoto směrovače je nejdůležitější správné nastavení. To se provede zadáním následujících příkazů do konzole IOS v konfiguračním režimu:

```
router ospf 1
 network 10.0.0.0 0.0.255.255 area 0
 network 145.229.1.0 0.0.0.255 area 0
```

Směrovače firmy CISCO mají standardně pouze dva Fast Ethernetové porty v úloze. U jednoho směrovače, ale potřebujeme alespoň čtyři. Jako nejjednodušší řešení v laboratorních podmínkách se jeví použití přídavného switch modulu. Jednotlivé porty „switch“ modulu jsou zařazeny do různých VLAN. Každá VLAN má pak vytvořeno své vlastní rozhraní na síťové vrstvě, čímž se z modulu pracujícího

na linkové vrstvě stane zařízení pracující na vrstvě síťové[1].

```
interface FastEthernet0/0/0
 switchport access vlan 10
interface FastEthernet0/0/1
 switchport access vlan 11
!
interface Vlan10
 ip address 10.0.0.1 255.255.255.0
interface Vlan11
 ip address 10.0.1.1 255.255.255.0
```

Poznámka

Během studia Cisco akademie CCNP – Switching, jsem se ale dozvěděl o mnohem elegantnějším způsobu. Na jednotlivých rozhraních stačí použít příkaz `no switchport` a rozhraní se začne chovat jako běžné rozhraní na síťové vrstvě, kterému lze standardním příkazem `ip address` přiřadit IP adresu. Síť ale fungovala i podle předešlého nastavení, tak jsem se řídil heslem „If it works, don't fix it!". Česky řečeno pokud to funguje, neopravuj to!

3.1.2 Domovský agent – Home agent

Služby domovského agenta jsou zprovozněny na levém směrovači. Funkce Mobile IP se na směrovači aktivuje zadáním příkazu `router mobile`. Dále je potřeba protokol MIP redistribuovat do směrovacího protokolu:

```
router ospf 1
 redistribute mobile subnets
```

Výraz subnets na konci příkazu znamená, že se budou přenášet i masky podsítí. Pokud by se tato informace nepřenášela, směrovač by masku odvodil podle třídy IP adresy. Příkazem `ip mobile home-agent` se určí, že směrovač má tuto funkci. Přiřazení mobilních uzlů lze provézt dvěma způsoby. V prvním případě musíme mít vytvořenou domovskou síť na jednom z rozhraní a do této sítě mobilní uzly

[1] dle ISO/OSI modelu

přiřazujeme. Toto řešení má jistou nevýhodu v tom, že když je dané rozhraní ve stavu „DOWN“, mobilní uzly nejsou schopny se ani z cizí sítě přiřadit k domovskému agentovi, protože jejich domovská síť je nedostupná.

```
interface FastEthernet0/0/0
 ip address 192.168.100.1 255.255.255.0
!
ip mobile home-agent address 192.168.100.1
ip mobile host 192.168.100.10 192.168.100.20 interface fa0/0/0 lifetime 100
```

Druhá možnost tento problém řeší přiřazením mobilních uzlů do virtuální sítě, což se jeví jako vhodnější řešení[1]:

```
ip mobile home-agent address 192.168.100.1
ip mobile virtual-network 192.168.100.0 255.255.255.0
ip mobile host 192.168.100.10 192.168.100.20 virtual-network 192.168.100.0
 255.255.255.0 lifetime 100
```

3.1.3 Cizí agenti – Foreign agenti

Konfigurace dvou znázorněných cizích agentů se liší pouze v IP adresách, jinak je konfigurace identická. Stejně jako u domovského agenta je potřeba aktivovat funkci MIP a nastavit redistribuci do OSPF. Služby FA, nastavení CoA a IRDP protokolu jsou aktivovány následujícími příkazy:

```
interface FastEthernet0/1
 ip address 172.16.1.1 255.255.255.0
 ip mobile foreign-service
 ip irdp
 ip irdp maxadvertinterval 4
 ip irdp minadvertinterval 3
 ip irdp holdtime 9
ip mobile foreign-agent care-of FastEthernet0/1
ip mobile foreign-service
```

3.1.4 Přepínač

Znázorněný přepínač připojené sítě nespojuje do jedné všesměrové(broadcastové) domény, ale pomocí Virtual Local Area Network (VLAN) je všechny odděluje. Přepnutím VLAN na portu mobilního uzlu je docíleno simulování cestování mobilního uzlu. Pro rychlejší navázání spojení je vhodné u daných rozhraních nastavit `switchport host`, což je standardně vytvořené makro v IOS pro příkazy:

```
Switch(config)#interface range fastEthernet 0/1 - 48
 switchport mode access
 spanning-tree portfast
 no channel-group
```

Pomocí prvního příkazu se dostaneme z globálního konfiguračního režimu do režimu konfigurace rozhraní portů „Fast Ethernet“ 1 až 48. Další příkazy jsou obsaženy v uvedeném makru. První příkaz nastaví pevně port jako přístupový. Často bývá u zařízení Cisco nastaven mód jako dynamický, což vede ke zdržení, protože přepínač zjišťuje, zda se na druhém konci nenachází také přepínač a nebylo by tedy vhodnější nastavit port jako trunk[2]. Druhý příkaz řekne protokolu „Spanning tree“ aby rozhraní přivedl okamžitě do aktivovaného stavu a neočekával, že by mohla nastat smyčka. Toto nastavení se silně nedoporučuje při propojování více přepínačů. Poslední příkaz vyřadí port ze skupiny „EtherChannel“ pokud v nějaké byl.

3.1.5 Konfigurace SNMP

V diplomové práci budu pracovat s SNMPv2c, díky tomu stačí nastavit pouze komunitu pro čtení SNMP údajů:

```
snmp-server community public RO
```

Název komunity je `public` a parametr `RO` znamená `ReadOnly`.

3.1.6 Mobile node – Mobilní uzel

Funkce mobilního uzlu jsou zprovozněny na laboratorním PC s operačním systémem Windows XP a nainstalovaným softwarem Cisco mobile client verze 2.0.14[3]. Jelikož

[2] trunk je schopen na jednom rozhraní přenášet data více než jedné VLAN

[3] software je podporován pouze pro Windows XP

tento software ne vždy pracuje správně, je někdy potřeba jej opakovaně restartovat, a dokonce je někdy nutné restartovat celý operační systém . Hlavním problémem je, že klient při změně nastavení neaplikuje správně nové nastavení. Přijímající domovský agent nebo cizí agent, pak takovou zprávu se špatným nastavením může zamítnout. Pro odstranění tohoto problému je vhodné zapnout na směrovači debug: `debug ip mobile`. Konfigurace mobilního klienta je uložena v profilovém souboru s koncovkou `.mcprofile`. Obsah tohoto souboru je uveden v příloze B.5. Výpis debugu domovského agenta s navázáním mobilní vazby[4]:

```
MobileIP: HA 262 rcv registration for MN 192.168.100.10
 on FastEthernet0/0 using HomeAddr 192.168.100.10 COA 172.16.1.1 HA 10.0.0.3
 lifetime 36000 options sbdmg-t- identification D0AB8E5C2900237B
MobileIP: UDP Tunnel Request rejected
MobileIP: Authenticating MN 192.168.100.10 using SPI 100
MobileIP: Authentication algorithm HMAC-MD5 and 16 byte key
MobileIP: Authenticated MN 192.168.100.10 using SPI 100 and 16 byte key
MobileIP: Mobility binding for MN 192.168.100.10 updated - tunnel changed
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel1, changed state to up
MobileIP: Tunnel1 (IP/IP) created with src 10.0.0.3 dst 172.16.1.1
MobileIP: MN 192.168.100.10 Tunnel route deleted for 192.168.100.10
 via gateway 192.168.100.100
MobileIP: Deleted Tunnel0 src 10.0.0.3 dest 192.168.100.100
MobileIP: MN 192.168.100.10 Insert route for 192.168.100.10/255.255.255.255
 via gateway 172.16.1.1 (metric 1) on Tunnel1
Null mn or mn->aaa_id in ipmobile_aaa_acct_net_update
MobileIP: Roam timer started for MN 192.168.100.10 using 192.168.100.10,
MobileIP: HA accepts registration from MN 192.168.100.10
MobileIP: Dynamic and Static Network Extension Length 0 - 0
MobileIP: Sending Registration Reply message for MN 192.168.100.10 with
 Lifetime 36000, Home Address 192.168.100.10, Home Agent 10.0.0.3 Code 0
MobileIP: Composed mobile network extension length:0
MobileIP: Authentication algorithm HMAC-MD5 and 16 byte key
MobileIP: MN 192.168.100.10 MHAE added to MN 192.168.100.10 using SPI 100
MobileIP: MN 192.168.100.10 - HA sent reply to 172.16.1.1
```

[4]angl. Mobile binding

4 SPRÁVA MOBILE IP POMOCÍ SNMP

Téma práce vzešlo z neexistence programu, který by spravoval prvky Mobile IP, a proto jsem si kladl za cíl vytvořit program, který bude lehké používat nezávisle na operačním systému a zároveň pokryje požadavky na něj kladené. Program jsem se rozhodl napsat v jazyce JAVA. Bude představovat SNMP manager spravující aktivní prvky testovací sítě (3.1). Funkční základ programu pro komunikaci prostřednictvím SNMP protokolu tvoří Application Programming Interface (API) s názvem SNMP4J z balíčku `org.snmp4j.x`. Uživatelské rozhraní je vytvořeno ze standardních prvků balíčku `javax.swing.*`. Pro ukládání nastavení do XML souborů bylo využito balíčků `javax.xml.*` a `org.w3c.dom.*`.

4.1 Požadavky na aplikaci

Před vývojem aplikace je nutné stanovit, jak se má aplikace v daných situacích chovat a jaké schopnosti má mít. Podle zadání má aplikace sledovat stav agentů protokolu MIPv4. To znamená zjišťovat především navázané vazby mezi domovským agentem a mobilním uzlem. Z této vazby lze následně určit, zda se mobilní uzel nachází v domácí síti domovského agenta, zda využívá některé sítě a CoA cizích agentů nebo se nachází někde v internetu, tunel si sestavuje sám použitím CCoA. Tím se pokryla i požadovaná funkce monitorování aktivity mobilních uzlů. Tyto funkce jsem považoval za hlavní a jsem rád, že se mi je podařilo velmi dobře implementovat. Na následujících stránkách, se pokusím přiblížit tuto implementaci.

Dalším úkolem bylo umožnit nastavování stavu agentů. Při bližším zkoumání se ale ukázalo, že je jen málo objektů v SNMP databázi lze modifikovat, a proto jsem tuto funkci neimplementoval. Jedním z takových objektů je `mipEnable`, který indikuje zda je protokol Mobile IP na dané entitě aktivovaný. Při pokusu o deaktivaci a následnou aktivaci prostřednictvím SNMP došlo ke kompletnímu smazání nastavení MIP, a proto tato funkce není implementována.

4.2 API SNMP4J

Před vývojem aplikace je nutné se rozhodnout, zda si programátor všechny funkce potřebné pro běh programu napíše sám nebo se rozhodne využít některého z dostupných API. První případ představuje zbytečně složitou a trnitou cestu, protože v dnešní době je dostupné velké množství i opensource API. Díky těmto API se programátor nemusí dopodrobna starat o základní funkce, ale může využít funkcí vyšších dostupných z API.
Jedním takovým API je SNMP4J. Jedná se, dle mého názoru, o velice detailně a kvalitně zpracované API pro jazyk JAVA. Je poskytováno zdarma z webu projektu: `www.snmp4j.org`. Má plně otevřený kód pod licencí Apache 2.0. S jeho využitím lze sestavit jak SNMP managera tak i SNMP agenta. API obsahuje i implementaci SNMPv3, tudíž prostředky všechny prostředky pro zabezpečení komunikace jsou dostupné.
V mé práci jsem využil pouze zlomek tříd a funkcí z SNMP4J, což dokazuje jeho komplexnost a potenciál pro dobrý vývoj.

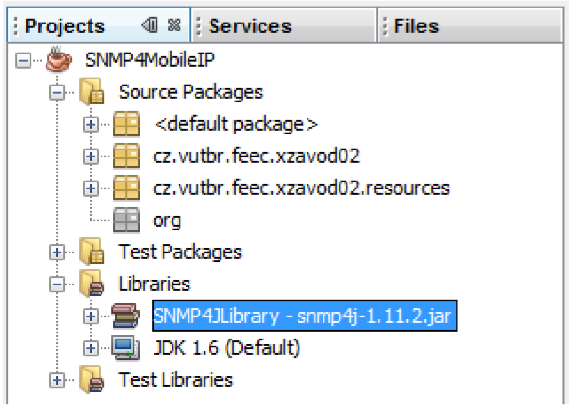
4.3 Vývoj vlastní aplikace

Pro vývoj vlastní aplikace bylo využito vývojového prostředí NetBeans verze 6.9.1. V prostředí je nutno nejdříve vytvořit projekt a do něj naimportovat požadované API SNMP4J. V době psaní práce byla dostupná verze zdrojových kódů 1.11.2. Stažený soubor `snmp4j-1.11.2-distribution.zip` obsahuje složky:

```
snmp4j-1.11.2/          //kořenová složka
 dist/                  //obsahuje zkompilované soubory *.jar
 lib/                   //knihovny potřebné při vývoji balíčku SNMP4J
 mibs/                  //databázové MIB soubory
 src/                   //zdrojové kódy projektu
```

Aby bylo možné ve vlastním projektu využít balíčku, je potřeba v NetBeans vytvořit vlastní knihovnu, do které se následně naimportuje soubor `snmp4j-1.11.2.jar`. Dále je vhodné si přidat i soubor `snmp4j-1.11.2-javadoc.jar` do záložky javadoc,

diky kterému se práce v NetBeans stává mnohem pohodlnější díky nápovědě během programování. Správný import knihovny je znázorněn na obrázku 4.1. V následujících sekcích budou popsány jednotlivé třídy a funkce programu, které vytvářejí balíček `cz.vutbr.feec.xzavod02`.

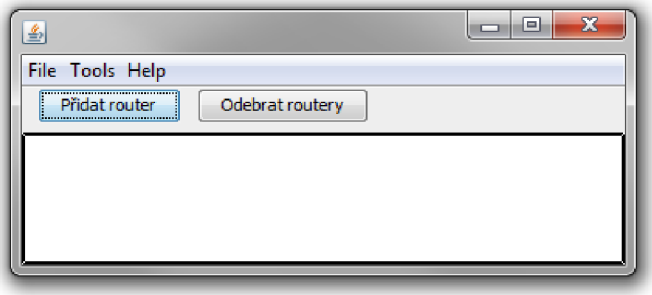


Obr. 4.1: Import knihovny do prostředí NetBeans

4.3.1 SNMP4MobileIP.java

Pro běh programu v jazyce JAVA, je potřeba aby některá ze tříd obsahovala spustitelnou metodu main. To je právě případ třídy `SNMP4MobileIP.java`. Ta navíc dědí ze třídy `JFrame`, čímž vytváří základní viditelnou pracovní plochu programu – okno, znázorněné na obrázku 4.2.

V horní části okna se nachází menu vytvořené z `JMenuBar` obsahující položky `File`, `Tools` a `Help`. Pod menu položkou `File` se skrývají standardně známé funkce jako



Obr. 4.2: Základní okno programu

je `Open`, `Save` a `Exit`. Funkce `Save` slouží k uložení stavu a nastavení programu do XML souboru tak jak je popsáno v kapitole 4.3.5. Položka `Open` pak slouží k otevření takového souboru a načtení parametrů z něj. Poslední položkou v menu `File` je `Exit`, která ukončí běh programu.

Druhá menu položka `Tools` obsahuje funkce `Mobile node list`(viz 4.3.4) a `SNMP probe`(viz 4.3.6), blíže popsané v příslušných kapitolách.

Pod panelem menu se nacházejí 2 tlačítka. První z nich přidává do bílé pracovní plochy `JDesktopPane` interní rámec `JInternalFrame`:

```
JInternalFrame frame = createIFrame(fr);
//vytvoření instance rámce s~indexem fr
jDesktopPane1.add(frame);
//přidání interního rámce do pracovní plochy
jDesktopPane1.getDesktopManager().activateFrame(frame);
//nastavení interního rámce jako aktivní
```

Důležitá je zde funkce `createIFrame`, která vrací novou instanci typu `JInternalFrame`, s názvem „Router + index“. Dále nastaví vlastnosti rámce, aby jej bylo možné minimalizovat, maximalizovat, měnit jeho rozměry a zavírat. Následně je do rámce přidán panel `RouterJPanel`(viz 4.3.3)

```
private javax.swing.JInternalFrame createIFrame (int index){
 JInternalFrame iframe;
 iframe = new JInternalFrame("Router "+index, true,true,true,true);
 RouterJPanel panel = new RouterJPanel(index);
 r.setPanel(panel);
 r.setInternalFrame(iframe);
 iframe.add(panel);
 iframe.grabFocus();
 fr++;
 return iframe;
 }
```

Druhé tlačítko slouží k odebrání všech `JInternalFrame` z pracovní plochy:

```
jDesktopPane1.removeAll();          //odstranění všech rámců
jDesktopPane1.repaint();            //překreslení prvku
```

4.3.2 Request.java

Třída `Request.java` představuje část programu, která se stará o odesílání a následné přijímání SNMP zpráv. Při vytvoření instance této třídy jsou jako výchozí hodnoty nastaveny tyto parametry:

```
commtarget.setCommunity(new OctetString("public"));
//komunita pro odesílání SNMP zpráv
commtarget.setVersion(SnmpConstants.version2c);//verze SNMP
commtarget.setRetries(0);//počet opakování
commtarget.setTimeout(5000);//timeout zprávy
```

Pokud by bylo potřeba tyto hodnoty změnit, tak lze k objektu `commtarget` přistupovat i z venku třídy `Request.java` a nastavit tak libovolné parametry dodatečně. O odeslání SNMP zprávy se stará veřejná funkce `send`, která má tři vstupní proměnné:

- cílová IP adresa typu `String`
- dotazované PDU typu `PDU`
- `boolean` hodnota určující, zda odpověď bude ve formátu „OID = odpověď“ nebo pouze „odpověď“.

Deklarace metody `send`:

```
public void send (String ipAddress, PDU pdu, boolean only_data) throws
 Exception
```

Po zavolání této metody se nastaví cílová adresa v objektu `commtarget` a začne se naslouchat na výchozím portu pro příchozí UDP datagramy:

```
commtarget.setAddress(new UdpAddress(ipAddress + "/" + port));
TransportMapping transport = new DefaultUdpTransportMapping();
transport.listen();
```

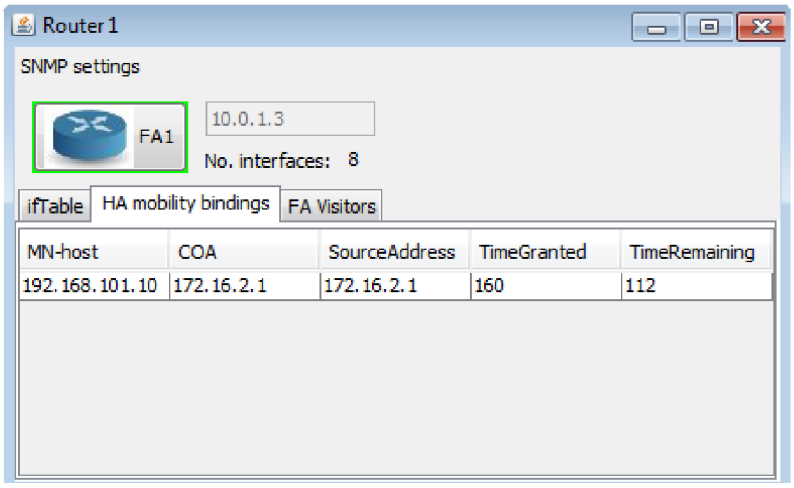
Dále se vytvoří instance objektu SNMP, které se předá mapování portů, podle typu PDU[1] se použije metoda get pro získání požadované hodnoty:

```
Snmp snmp = new Snmp(transport);
ResponseEvent response;
    if (pdu.getType() == pdu.GETBULK) {
          response = snmp.getBulk(pdu, commtarget);
          }else if... //více v příloze
         ... else response=null;
```

4.3.3 RouterJPanel.java

Tato funkce, která dědí ze standartní třídy `Jpanel`, tvoří srdce celé aplikace. Pro každý směrovač, který chceme sledovat si vytvoříme jednu instanci této třídy a ta nám bude daný směrovač sledovat a výsledky z něj získané zobrazovat na svých jednotlivých prvcích. Vzhled a rozmístění jednotlivých prvků je znázorněno na obr. 4.3.

Hlavním ovládacím prvkem je zde `JButton` s ikonou směrovače. Toto tlačítko stří-



Obr. 4.3: Rozložení jednotlivých prvků na RouterJPanelu

davě zapíná a vypíná odesílání SNMP dotazů na IP adresu uvedenou v `JTextField`. Pokud je odesílání aktivní, tak se textové pole deaktivuje, aby nebylo možné IP

[1] z argumentu funkce send

adresu nedopatřením změnit. Barva rámečku okolo tlačítka indikuje, zda je daný IP uzel dosažitelný(zelená) či nikoliv(červená). To zda je daný uzel „Online“ zjistí funkce `isOnline` pomocí třídy `InetAddress` z balíčku `java.net.*`:

```
private boolean isOnline() {
    try {
        adresa = InetAddress.getByName(this.getIP());
        if (adresa.isReachable(2000)) {
            return true; //pokud je dostupný do 2000ms vrací true
        } else return false; //jinak vrací false
        }//...následuje ošetření výjimky
```

Pokud je uzel dostupný následují tyto kroky:

1. Barva ikony směrovače se nastaví na zelenou

```
RouterBtn.setBackground(Color.green);
```

2. Pomocí funkce getHostname() se zjistí název uzlu, který se napíše vedle ikony

```
public Request req = new Request();
PDU hostnamePDU = new PDU();
hostnamePDU.setType(PDU.GET);
hostnamePDU.add(new VariableBinding(new OID("1.3.6.1.2.1.1.5.0")))
req.clearResponse_text(); //vyčištění odpovědi od minulého dotazu
req.send(getIP(), hostnamePDU, true);//odeslání dotazu
hostname =  req.response_text.getFirst();//zjištění odpovědi
RouterBtn.setText(hostname);//zobrazení odpovědi vedle tlačítka
```

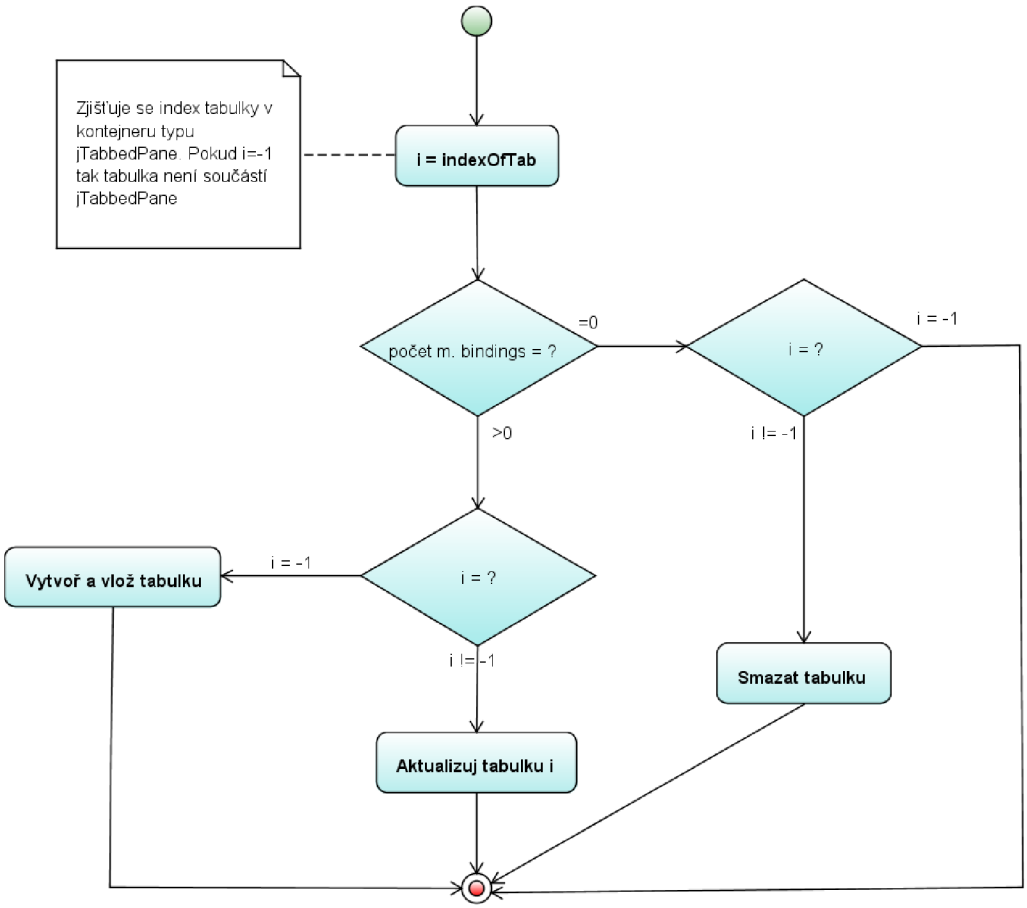
3. Než se začnou ze síťového prvku získávat informace pro Mobile IP, je zapotřebí pomocí funkce checkMIPEntities() zjistit, které entity protokolu Mobile IP jsou na daném uzlu v provozu. Program se dotazuje na objekt s názvem mipEntities. Vrácená hodnota udává bitově, které entity jsou aktivovány: bit 0 pro mobilní uzel, bit 1 pro cizího agenta a bit 2 pro domovského agenta. Např. tedy pokud je aktivován domovský i cizí agent tak je vrácená hodnota 0x60, binárně 01100000_2.
4. Funkce getIfTable() nejdříve zjistí celkový počet rozhraní na zařízení a následně získá tabulku parametrů těchto rozhraní. Zjišťují se hodnoty indexu

rozhraní(ifIndex), popis rozhraní(ifDescr), přijaté(ifIn) a odeslané(ifOut) bajty na rozhraní. Zdrojový kód této funkce je již poměrně dlouhý, a proto je uveden v příloze.

5. Pokud se v kroku 3 zjistilo, že zařízení poskytuje služby domovského agenta, program zjistí pomocí funkce getHATable() počet vazeb s mobilními uzly a následně zjistí jednotlivé hodnoty jako jsou domácí IP adresa mobilního uzlu, CoA, zdrojovou adrese, čas v sekundách který přiděluje při každé registraci mobilního uzlu a zbývající čas v sekundách. Pokud mobilní uzel neprovede reregistraci do vypršení času, je mobilní vazba zrušena.
6. Podobně jako v předchozím kroku se zjišťují služby cizího agenta(pokud se v bodě 3 zjistilo, že tyto služby jsou poskytovány). Nejdříve se zjistí počet návštěvníků a následně se pro všechny zjistí parametry: návštěvníkova IP adresa, jeho domácí adresa, adresa domácího agenta, u kterého je registrovaný, a časy jako v minulém případě.
7. Tento postup se opakuje každých 5 sekund, dokud není uživatelem zastaven, nebo uzel nepřestane odpovídat(signalizováno červeně).

Ve shrnutí se program chová tak(pokud je cílené zařízení dostupné), že tabulka rozhraní se zobrazí vždy i na zařízeních, které Mobile IP vůbec nepodporují. Pokud Mobile IP podporují, tak se dynamicky přidávají a odebírají tabulky mobilních vazeb návštěvníků. Přidávání a odebírání těchto tabulek pracuje podle algoritmu znázorněného na obrázku 4.4:

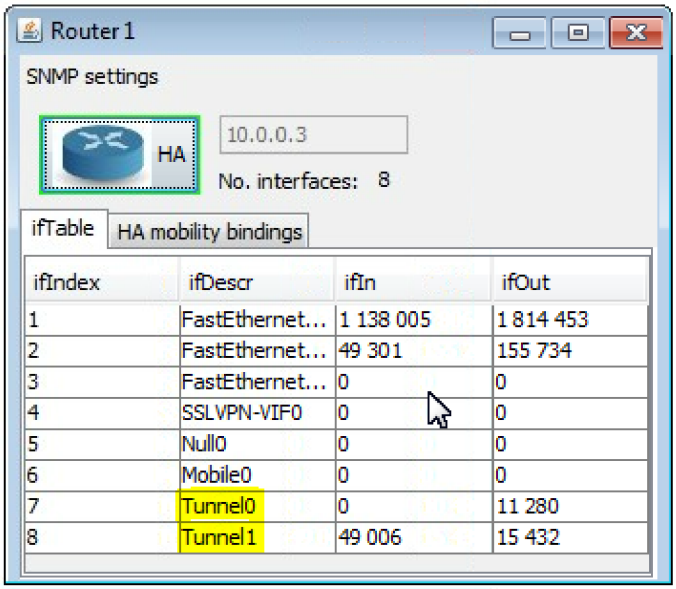
Navíc je vidět, že při vytvoření mobilní vazby, se v tabulce rozhraní vytváří nové virtuální rozhraní – tunel. Ten se pro každého účastníka vytváří zvlášť tak jak se teoreticky předpokládá. Rozhraní tohoto tunelu se vytváří i v případě cizího agenta. Z pozorování počtu odeslaných bytů, bylo potvrzeno, že v případě kdy je MN zaregistrovaný u některého z FA dochází k trojúhelníkovému směrování, protože počet bytů narůstal pouze v jednom směru. U HA v odchozím směru a u FA v příchozím směru. V opačných směrech byly počty nulové. Pokud se ale MN registroval prostřednictvím CCoA, tak tunelování provozu probíhalo oběma směry. Tento stav je zobrazen na obrázku 4.5



Obr. 4.4: Algoritmus udržování tabulky mobilních vazeb

4.3.4 Zobrazení seznamu mobilních hostů

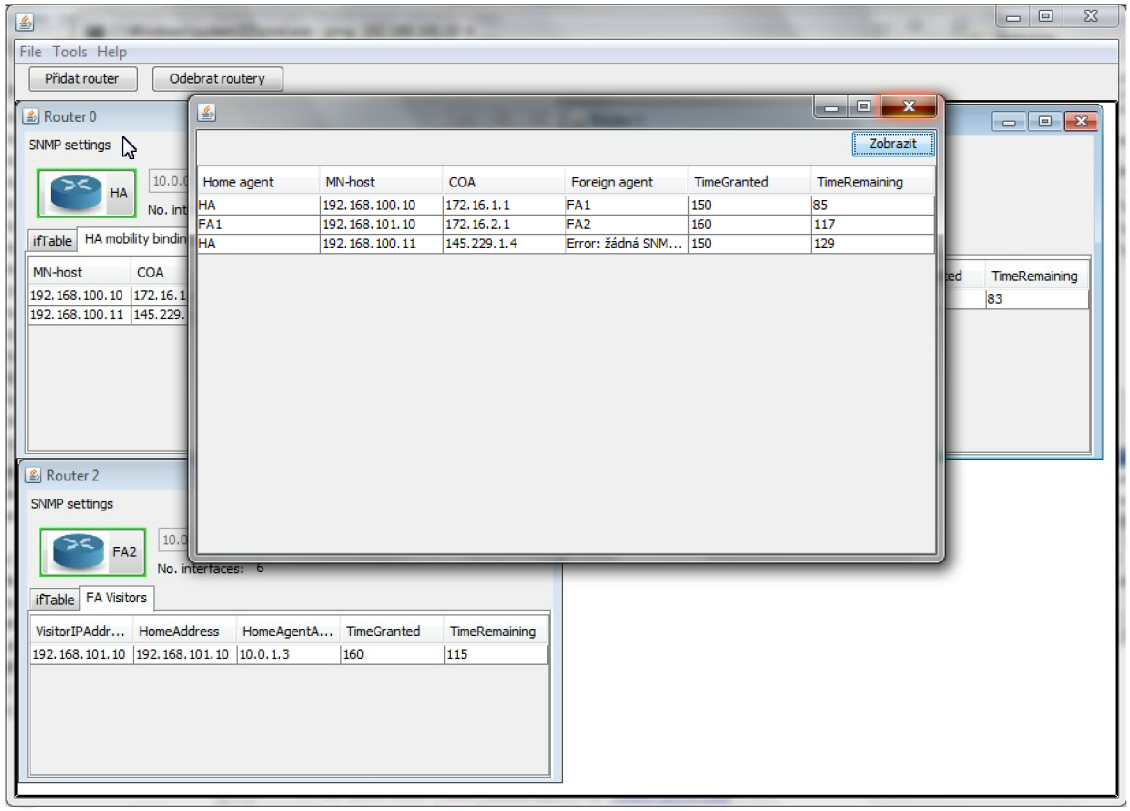
Pokud sledujeme větší síť, kde je více než jeden domovský agent, a potřebujeme zjistit, kde se daný mobilní uzel nachází. Je velice nepohodlné, hledat ke kterému domovskému agentovi je přiřazen a pak jej hledat v seznamu vazeb. Mnohem lepší se jeví způsob, kdy do společného seznamu mobilních uživatelů přispívají jednotliví domovští agenti. Proto se během funkce getHAtable() v třídě `RouterJPanel.java`(4.3.3) kontroluje, zda již daná mobilní vazba v globálním seznamu existuje, je potřeba ji modifikovat, nebo smazat. Globální seznam má strukturu `LinkedList`, a je součástí třídy `SNMP4MobileIP.HaMobilityBindings`. Způsob udržování seznamu je naznačen v tomto úryvku kódu:



Obr. 4.5: Počet přijatých bajtů v jednotlivých tunelech

```
HaMobilityBinding h = new HaMobilityBinding();
h.setHaHostname(hostname);
... //nastavení hodnot z˜přijatých dat do objektu h
h.setHaMobilityBindingTimeRemaining(data[j][4]);
LinkedList<HaMobilityBinding> haBindings;\\ukazatel na glob. seznam
haBindings = SNMP4MobileIP.HaMobilityBindings;
boolean containsMN = false;//předpokládá se, že MN není v˜seznamu
int l = 0;//pokud ano tak do proměnné l se uloží jeho index
 for (int k˜= 0; k˜< haBindings.size(); k++) {
  if (haBindings.get(k).getHaMobilityBindingMN().hashCode() ==
   h.getHaMobilityBindingMN().hashCode()) {//porovnání hashe
   containsMN = true;//pokud jsou stejné tak MN je v˜seznamu
   l = k;}}//nastavení hodnoty l na index v˜seznamu
 if (!containsMN) {//pokud MN není v˜seznamu
  haBindings.add(h);//přidá se do seznamu
 }else if( containsMN){//jinak pokud je v˜seznamu
 haBindings.get(l).set...}//všechny hodnoty se aktualizují
```

Zobrazení seznamu mobilních uzlů proběhne na samostatném okně vyvolaného z menu Tools > Mobile node list. Po stisku tlačítka „Zobrazit“ se naplní tabulka hodnotami z globálního seznamu `SNMP4MobileIP.HaMobilityBindings` Systém pl-



Obr. 4.6: Okno seznamu mobilních uzlů

nění je obdobný jako ve třídě `RouterJPanel.java`, pouze s tím rozdílem, že na adresu CoA se zašle SNMP dotaz na parametr sysName, což odpovídá hostname uzlu cizího agenta. Z obrázku 4.6 je vidět, že v prvních dvou řádcích došlo k úspěšnému zjištění názvu cizího agenta, ale v posledním se nepodařilo hostname zjistit, protože mobilní uzel `192.168.100.11` je registrován prostřednictvím CCoA, tudíž by musel na SNMP dotaz odpovědět sám.

4.3.5 Ukládání nastavení do XML souborů

Uložení nastavení do souboru se mohlo provézt i prostým uložením do obyčejného textového souboru. To ale neposkytuje takovou přehlednost zápisu jako je to v případě XML souboru. Extensible Markup Language (XML) je obecně známý značkovací jazyk. Byl vyvinut konsorciem W3C[8] pro univerzální výměnu dat mezi aplikacemi. Nestará se o vzhled(styl) dat, ale pouze o jejich datovou strukturu. Pro zpracování XML souborů, lze obecně i v jazyce JAVA využít dvou přístupů:

- Document Object Model (DOM) parser – data se načtou a z nich se vytvoří stromová struktura představující původní dokument. Je jednodušší na implementaci, ale při velkých souborech spotřebovává velké systémové požadavky.
- The Simple API for XML (SAX) parser – tak jak jsou data postupně načítána, parser generuje postupně sérii událostí. Je složitější na implementaci. Své výhody využívá při zpracování velkých souborů, protože nemusí být celé načtené do paměti.

Protože budu zpracovávat pouze malé soubory, rozhodl jsem se v práci využít DOM parseru. Základem každého XML dokumentu je kořenová entita(root) obsahující další otagované elementy. Příklad vytvořeného XML souboru:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Routers>//kořen dokumentu
 <Router id="0">// první RouterJPanel s˜id 0
  <IP>10.0.0.3</IP> //IP adresa
 <frameBounds>//vlastnosti okna
  <x>15</x> //pozice ve směru x
  <y>20</y> //pozice ve smeru y
  <width>453</width> //šířka okna
  <height>298</height> //výška okna
  </frameBounds>
  </Router>
</Routers>//ukončení dokumentu
```

Pro vytváření těchto souborů byla vytvořena třída `XMLCreator.java`. Pro uložení do XML souboru slouží funkce `saveAs`, které předáme pomocí argumentů seznam RouterJPanelů s parametry a odkaz na soubor do kterého budeme hodnoty zapisovat.

```
public void saveAs(LinkedList<RouterData> listOfRouters, File file)
```

Funkce vytvoří novou instanci dokumentu, kořen dokumentu nazve „Routers“. Se seznamu `listOfRouters` postupně získává data a vytváří z nich elementy. Příklad vytvoření zjednodušené struktury dokumentu je zobrazen na následujícím kódu:

```
DocumentBuilderFactory docFactory=DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

Document doc = docBuilder.newDocument();
Element rootElement = doc.createElement("Routers");
doc.appendChild(rootElement);
Element router = doc.createElement("Router");
rootElement.appendChild(router);
router.setAttribute("id",Integer.toString(j));
```

Opačný účel jako funkce `saveAs` má metoda `Open`.

```
public LinkedList Open(File file)
```

Ze zvoleného souboru file, za pomocí DOM parseru, sestaví seznam oken `RouterJPanel`-ů a nadřazená třída `SNMP4MobileIP`(4.3.1) si z něj sestaví okna s rozmístěním tak jak tomu bylo při jeho uložení. Podrobný zdrojový kód je uveden v příloze A.

4.3.6 SNMP probe – testování funkčnosti SNMP

Protože je v určitých případech nutno ověřit správnou funkci SNMP protokolu, byla sestavena funkce pro ověření správnosti. Tato funkce se skrývá ve třídě `SNMPProbeJDialog.java`. Jak název napovídá, jedná se o dialogové okno `JDialog`, které lze zobrazit v menu Tools > SNMP probe. Ve výchozím nastavení je nastavení jsou nastaveny hodnoty podle tabulky 4.1. Pokud je SNMP agent správně nastaven a je dostupný, tak se po stisknutí tlačítka `Send` zjistí objekt s OID 1.3.6.1.2.1.1.5.0 neboli sysName ve tvaru `OID = hodnota`. Pokud nastane výjimka, agent vrátí index této výjimky viz tabulka 2.2.

4.3.7 Analýza komunikace aplikace

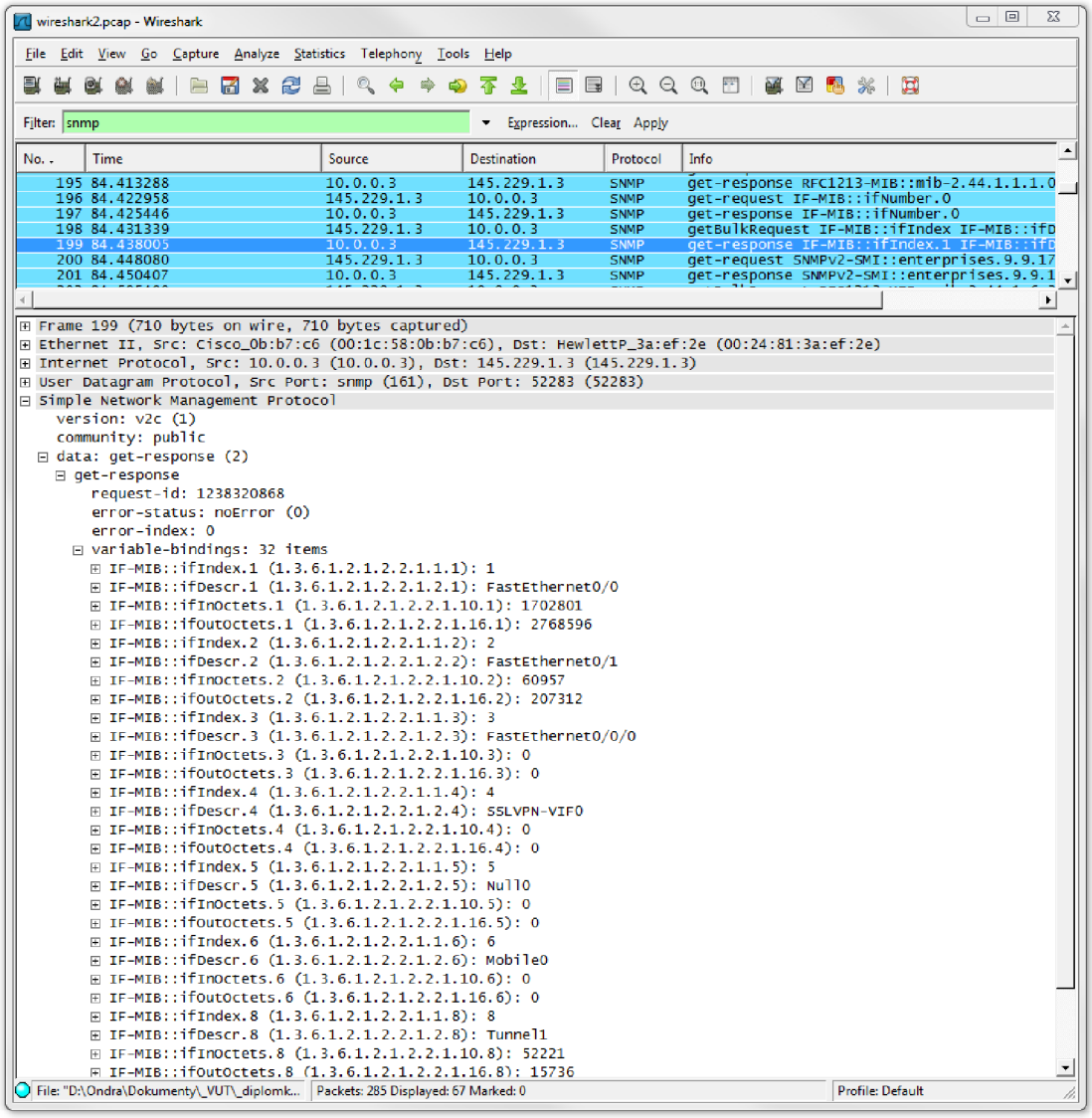
V předešlých kapitolách bylo nastíněno funkční jádro celé aplikace. Za velmi důležité považuji i znázornit jak se aplikace chová „navenek“.

Tab. 4.1: Výchozí hodnoty SNMP probe

Název hodnoty	Hodnota
Community	public
SNMP verze	2c
Počet opakovaných pokusů	5
Timeout zprávy	2000ms
UDP Port	161
Typ SNMP zprávy	GET
IP adresa	10.0.0.3
OID	1.3.6.1.2.1.1.5.0

Pro zachycení komunikace, lze využít libovolného síťového analyzátoru. K těm nejznámějším a nejdostupnějším patří analyzátor `Wireshark`. Z komunikace je velice zřejmé jak SNMP protokol funguje. SNMP manager nejdřív vyšle SNMP dotaz např. `getBulkRequest` na cílový UDP port 161 s náhodně zvolenou hodnotou zdrojového portu např. 52280. SNMP agent po obdržení dotazu porovná, zda se řetězec `community` shoduje s jeho nastaveným řetězcem. Pokud ano odpovídá z portu 161 na port, ze kterého došel dotaz. Při dalším dotazu SNMP manager hodnotu zdrojového portu inkrementuje o 1. Obrázek 4.7 zobrazuje zachycenou komunikaci, konkrétně odpověď na dotaz `getBulkRequest`. Z analýzy vyplývá, že je mnohem ekonomičtější používat zprávy typu `Bulk`, protože všech 32 hodnot, se přeneslo v jediném paketu. Bohužel, jak jsem se přesvědčil, tyto zprávy nemají zařízení všech firem správně implementovány. Např. ADSL směrovač firmy D-Link s označením DSL-584T místo aby vrátil pouze dotazované objekty z MIB tabulky `ifTable`, vrátil první 4 objekty této tabulky.

Dále je zřejmé, že komunikace probíhá nešifrovaně. To je způsobenou použitím SNMPv2c. V praxi by bylo nutné přejít na SNMPv3, aby komunikace nemohla být při analýze takto snadno čitelná. Na druhou stranu, tato jednoduchá čitelnost napomáhala při vývoji aplikace, pro kontrolu správnosti odesílaných a přijímaných dat.



Obr. 4.7: Analýza komunikace programu

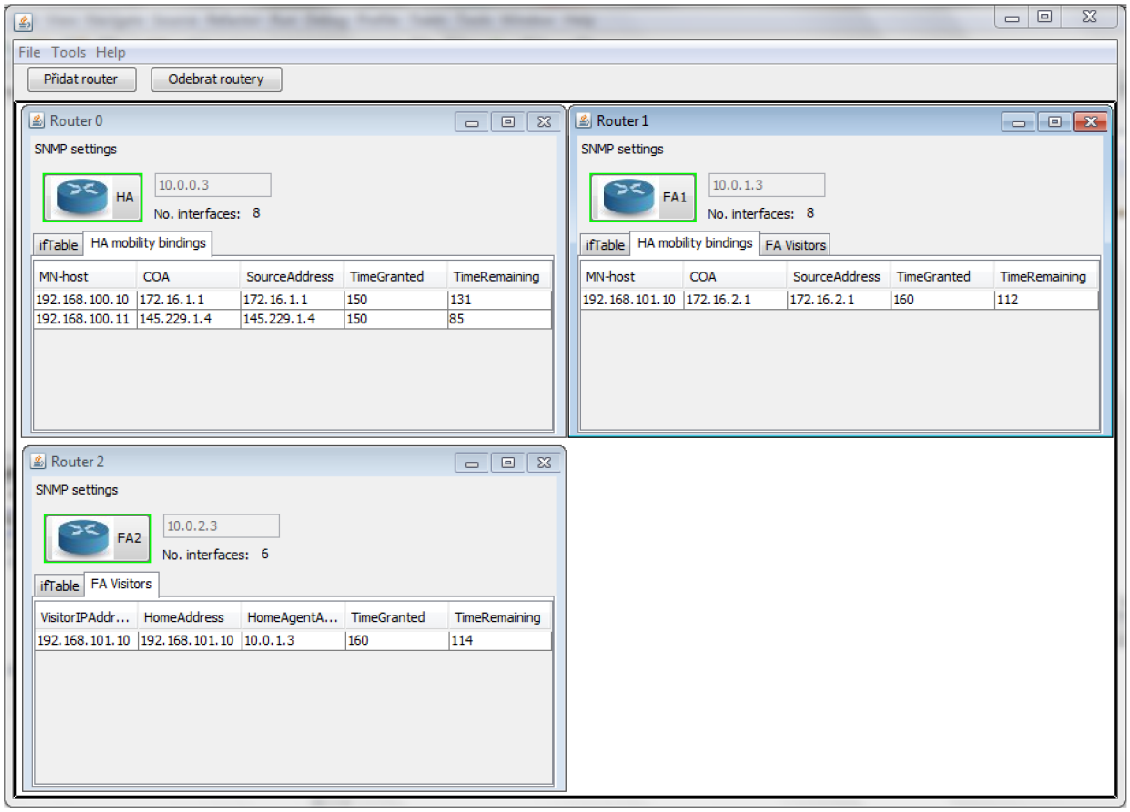
4.4 Návod k použití aplikace

Aplikace je zkompilována do jediného souboru s příponou `.jar` a je obsažena na přiloženém disku viz příloha A. Spustit ji lze v prostředí Windows příkazem

```
java -jar SNMP4MobileIP.jar
```

Po spuštění se zobrazí okno zobrazené na obrázku 4.2. Pokud začínáme sledovat novou síť, přidáváme pomocí tlačítka „Přidat router“ zařízení, která budeme sledovat. V pracovním prostoru se zobrazí „podokno“ zobrazené na obrázku 4.3. Do textového panelu vložíme IP adresu uzlu a pomocí tlačítka se aktivuje sledování tohoto uzlu.

Ve výchozím nastavení se každých pět sekund zasílají SNMP dotazy na daný uzel. Nejprve se zjišťuje hostname uzlu, který se zapíše vedle ikony směrovače, dále se zjistí tabulka rozhraní, která obsahuje jejich indexy, názvy a počty přijatých a odeslaných bajtů na jednotlivých rozhraních. Dále se zjistí, které entity protokolu Mobile IP jsou na uzlu aktivní. Pokud jsou aktivní služby domovského agenta, přidá se vedle tabulky s rozhraními tabulka s navázanými vazbami s mobilními uzly. Obdobně se zobrazí tabulka v případě cizího agenta, s tím, že se zobrazuje seznam návštěvníků. Celá aplikace vypadá tak jak je ukázáno na obrázku 4.8. Na obrázku jsou vidět zare-



Obr. 4.8: Zobrazení celé aplikace

gistrované mobilní uzly – 2 u směrovače s názvem HA a 1 dodatečně u FA1. Seznam všech mobilních uzlů je možné zobrazit z menu `Tools` > `Mobile node list` tak jak je vidět na obrázku 4.6. SNMP nastavení je možné upravit pro jednotlivé uzly v menu `SNMP settings` Rozmístění oken, včetně se zadanými adresami, lze uložit pomocí menu `File` > `Save` do XML souboru. To umožní, při dalším spuštění aplikace pomocí menu `File` > `Open` tento soubor otevřít a nastavení aplikovat do aplikace.

5 ZÁVĚR

Během této práce jsem se podrobně seznámil s protokoly Mobile IP a SNMP. Problematiku protokolu MIP jsem nastínil v kapitole 1. V průběhu práce jsem si uvědomil potenciál tohoto protokolu a zároveň mě mrzí, že tento protokol se příliš neuchytil co se týče IPv4. Vypadá to, že pro tuto verzi již protokol velké uplatnění nenajde. Do budoucna se má ale šanci uchytit se zaváděním IPv6, což bude ale ještě předmětem mnohých výzkumů a vývojů technologií.

V další části práce jsem teoreticky rozebral podstatu protokolu SNMP. Ten velmi dobře plní funkci prostředníka při vzdáleném spravování síťových prvků. Třetí kapitola popisuje testovací síť tvořenou výkonnými aktivními prvky od firmy Cisco modelové série 1841. Jako mobilní uzel byl použit stolní počítač s operačním systémem Windows XP a programem Cisco Mobile Client verze 2.0.14. Funkčnost celé sítě a protokolu Mobile IP byla otestována a případné problémy pomocí debugu na konzoli IOS odstraněny. V poslední kapitole popsán vývoj aplikace v jazyce JAVA za pomocí API `SNMP4J`. Ta je schopná velice dobře a názorně zobrazit co se s prvky sítě děje. Nejdříve zjistí, zda je daný uzel sítě dostupný a následně na něj zasílá SNMP dotazy, ze kterých dostává potřebná data. Z dat pak zjistí např. hostname daného síťového uzlu nebo podporované služby protokolu Mobile IP. Tyto data pak podle typu vhodně zobrazí buď jen na panelu jedná-li se o jednoduché hodnoty. Jestliže jsou data tabulkového typu, je z nich potřebná tabulka sestavena a zobrazena. Program dokáže zobrazit celkem 3 různé tabulky: seznam síťových rozhraní s počty odeslaných a přijatých bajtů, seznam vazeb domovského agenta a seznam návštěvníků cizího agenta. Programu od počátku práce prodělal velké změny. V semestrální práci byla prezentována aplikace, která zjišťovala jen zlomek toho co dělá nyní. Počáteční problém bylo pouze zaslat dotaz a přijmout odpověď. Jakmile bylo tohoto dosaženo, mohlo se pracovat na zasílání složitějších zpráv jako je `GetBulk` a sestavit z odpovědi tabulku. Bylo potřeba sestavit řadu pomocných funkcí, které umožnily přehlednější programování. Z doložených obrázků je vidět silný potenciál této aplikace při správě sítí s Mobile IP. Především přehlednost je znatelná, protože jediný jiný způsob jak zjistit zobrazená data(obr. 4.8) je se k jednotlivým zařízením

připojit pomocí konzole, telnetu nebo ssh a zadáváním příkazů požadovaná data zobrazit.

Naprogramování v jazyce JAVA umožnilo, že je program objektově orientovaný a tudíž dobře čitelný. Díky tomu by bylo možné program dále vyvíjet za hranice zadání a různě zlepšovat, např. ukládat do databáze, kde se jaký mobilní uzel v daném čase nacházel apod. To už by byl ale asi námět na jinou diplomovou práci.

LITERATURA

[1] RAAB, Stefan *Cisco: Mobilní IP technologie a aplikace, Grada,* 2007, 299 s., ISBN: 978-80-247-1611-4.

[2] Perkins, C., *"IP Mobility Support for IPv4"* 2002, RFC 3344. Dostupné z URL: <http://tools.ietf.org/html/rfc3344>.

[3] Perkins, C., *"Mobile IPv4 Challenge/Response Extensions (Revised)"* 2007, RFC 4721. Dostupné z URL: <http://tools.ietf.org/html/rfc4721>.

[4] Case, J., *"A Simple Network Management Protocol (SNMP)"* 1990, RFC 1157. Dostupné z URL: <http://tools.ietf.org/html/rfc1157>.

[5] Harrington, D., *"An Architecture for Describing SNMP Management Frameworks"* 2002, RFC 3411. Dostupné z URL: <http://tools.ietf.org/html/rfc3411>.

[6] Harrington, D., *"Version 2 of the Protocol Operations for the SNMP"* 2002, RFC 3416. Dostupné z URL: <http://tools.ietf.org/html/rfc3416>.

[7] Harrington, D., *"ICMP Router Discovery Messages"* 1991, RFC 1256. Dostupné z URL: <http://tools.ietf.org/html/rfc1256>.

[8] Konsorcium W3C *"Extensible Markup Language (XML) 1.0 (Fifth Edition)"* 2008. Dostupné z URL: <http://www.w3.org/TR/xml/>.

[9] Měcháček, J., ÚVT MU *"XML a Java"* 2001, Zpravodaj ÚVT MU. ISSN 1212-0901, roč. XII, č. 2, s. 9-12. Dostupné z URL: <http://www.ics.muni.cz/zpravodaj/articles/230.html>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API Application Programming Interface

HA Home agent

FA Foreign agent

MN Mobile node

CN Correspondent node

OSPF Open shortest path first

CoA Care of Address

CCoA Co-located Care of Address

IS Internet solution

IETF Internet Engineering Task Force

MIP Mobile IP

SNMP Simple Network Management Protocol

SMI Structure of Management Information

MIB Management Information Base

OID Object Identifier

IRDP ICMP Router Discovery Messages

XML Extensible Markup Language

DOM Document Object Model

SAX The Simple API for XML

ASN.1 Abstract Syntax Notation One

BER Basic Encoding Rules

VLAN Virtual Local Area Network

OSI Open Systems Interconnection

SPI Security Parametr Index

NAI Network Access Identifier

RADIUS Remote Authentication Dial In User Service

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO DISKU

A.1 Výpis obsahu disku

```
\root
 \animace        //flashová animace zobrazující
   prezentace.htm        //html stránka pro shlédnutí animace
   prezentace.swf        //macromedia flash
 \cisco                  //konfigurační soubory pro Cisco zařízení
 \dist           //složka se zkompilovaným programem
  \lib           //knihovny potřebné pro běh programu
   SNMP4MobileIP.jar     //samotný program
   spustme.bat  //spouštěcí soubor programu pro prostředí Windows
 \mibs  //využívané MIB
 \NetBeans projekt
   SNMP4MobileIP.zip     //zabalený projekt pro NetBeans
                         //se zdrojovými kódy
snmp4j-1.11.2-distribution.zip          //použité API SNMP4J
xzavod02_DP.pdf         //tento text v elektronické podobě
```

B KONFIGURAČNÍ SOUBORY SMĚROVAČŮ

B.1 Home agent

```
!
version 12.4
no service password-encryption
!
hostname HA
!
interface FastEthernet0/0
 ip address 10.0.0.3 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.0.1 255.255.255.0
 ip mobile foreign-service
 ip irdp
 ip irdp maxadvertinterval 4
 ip irdp minadvertinterval 4
 ip irdp holdtime 9
 duplex auto
 speed auto
!
interface FastEthernet0/0/0
 no ip address
 shutdown
 duplex auto
 speed auto
!
router mobile
!
router ospf 1
 redistribute mobile subnets
 network 10.0.0.0 0.0.0.255 area 0
 network 192.168.0.0 0.0.255.255 area 0
!
ip mobile home-agent
ip mobile virtual-network 192.168.100.0 255.255.255.0
ip mobile host 192.168.100.10 192.168.100.20 virtual-network 192.168.100.0 255.255.255.0
 lifetime 100
ip mobile foreign-agent care-of FastEthernet0/1
ip mobile secure host 192.168.100.10 spi 100 key hex 1234567890abcdef1234567890abcdef
 algorithm hmac-md5
ip mobile secure host 192.168.100.11 spi 100 key hex 1234567890abcdef1234567890abcdef
 algorithm hmac-md5
!
banner motd "Home Agent, xzavod02@stud.feec.vutbr.cz"
!
snmp-server community public RO
!
control-plane
!
line con 0
 logging synchronous
line aux 0
line vty 0 4
 privilege level 15
 no login
!
end
```

B.2 Foreign agenti

```
!
version 12.4
!
hostname FA1
!
interface FastEthernet0/0
 ip address 10.0.x.3 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 172.16.x.1 255.255.255.0
 ip mobile foreign-service
 ip irdp
 ip irdp maxadvertinterval 4
 ip irdp minadvertinterval 3
 ip irdp holdtime 9
 duplex auto
 speed auto
!
router mobile
!
router ospf 1
 redistribute mobile
 passive-interface FastEthernet0/1
 network 10.0.x.0 0.0.0.255 area 0
 network 172.16.x.0 0.0.0.255 area 0
!
ip mobile foreign-agent care-of FastEthernet0/1
ip mobile foreign-service
!
snmp-server community public RO
!
line con 0
 logging synchronous
line aux 0
line vty 0 4
 privilege level 15
 no login
!
end
```

B.3 Internet solution

```
!
version 12.4
hostname IS
!
no ip domain lookup
!
interface FastEthernet0/0
 ip address 145.229.1.1 255.255.255.0
!
interface FastEthernet0/1
 ip address 192.168.1.2 255.255.255.0
!
interface FastEthernet0/0/0
 switchport access vlan 10
!
interface FastEthernet0/0/1
 switchport access vlan 11
!
interface FastEthernet0/0/2
 switchport access vlan 12
!
interface FastEthernet0/0/3
!
```

```
interface Vlan1
 no ip address
!
interface Vlan10
 ip address 10.0.0.1 255.255.255.0
!
interface Vlan11
 ip address 10.0.1.1 255.255.255.0
!
interface Vlan12
 ip address 10.0.2.1 255.255.255.0
!
router ospf 1
 passive-interface FastEthernet0/0
 network 10.0.0.0 0.0.255.255 area 0
 network 145.229.1.0 0.0.0.255 area 0
!
banner motd "Internet Solution, xzavod02@stud.feec.vutbr.cz"
!
line con 0
 logging synchronous
line aux 0
line vty 0 4
 privilege level 15
 no login
!
scheduler allocate 20000 1000
end
```

B.4 Switch

```
interface FastEthernet0/1
 decription to HA
 switchport access vlan 1
 switchport mode access
 spanning-tree portfast
!
interface FastEthernet0/2
 decription to FA2
 switchport access vlan 2
 switchport mode access
 spanning-tree portfast
!
interface FastEthernet0/3
 decription to FA1
 switchport access vlan 3
 switchport mode access
 spanning-tree portfast
!
interface FastEthernet0/4
 decription to Mobile node
 switchport access vlan x
 switchport mode access
 spanning-tree portfast
!
```

B.5 Mobile node

výpis konfiguračního souboru xzavod02.mcprofile

```
#CiscoSystems
Version = 11.0
profile
  profile-header
    id = xzavod02
    key-version = 2.3
```

```
    ack-id = 0
    disable-protection = false
    device-mode = false
  end-profile-header

  mobile-ip
    local-ip-address = 192.168.100.10
    local-subnet = 255.255.255.0
    local-default-gateway = 192.168.100.1
    passthru = false
    broadcast-on = false
    broadcast-on-ppp = false
    enable-dynamic-ha = false
    initial-re-registration-time = 4
    re-registration-minimal-time = 30
    re-registration-threshold = 90
    colocated-registration-time = 600
    AAA-authenticate-always = false
    wait-colocated-time = 2
    force-reverse-tunneling = dont
    solicitation-address = 224.0.0.2
    registration-mode-classical
      home-agent-external-address = 10.0.0.3
      home-agent-internal-address = 10.0.0.3
    end-registration-mode-classical
    security-association
      remote-end = HA
      security-context
        spi = 0x100
        algorithm-identifier = HMAC-MD5
        key = 1234567890abcdef1234567890abcdef
        replay-protection-method = timestamp
      end-security-context
    end-security-association
    trusted-fa-list
      0.0.0.0
    end-trusted-fa-list
  end-mobile-ip

end-profile
```

C SNMP APLIKACE

Zdrojové kódy lze primárně nalézt na přiloženém disku, zde uvádím pouze jejich část.

C.1 Request.java

```
package cz.vutbr.feec.xzavod02;

import java.util.LinkedList;
import org.snmp4j.CommunityTarget;
import org.snmp4j.PDU;
import org.snmp4j.Snmp;
import org.snmp4j.TransportMapping;
import org.snmp4j.event.ResponseEvent;
import org.snmp4j.mp.SnmpConstants;
import org.snmp4j.smi.OctetString;
import org.snmp4j.smi.UdpAddress;
import org.snmp4j.transport.DefaultUdpTransportMapping;

public class Request{

  private static String  port    = "161";

    public static String getPort() {
        return port;
    }
    public static void setPort(String port) {
        Request.port = port;
    }
  public LinkedList<String> response_text = new LinkedList<String>();
  public LinkedList<String> oid = new LinkedList<String>();
  public CommunityTarget commtarget = new CommunityTarget();
  public void MyRequest(){
    commtarget.setCommunity(new OctetString("public"));
    commtarget.setVersion(SnmpConstants.version2c);
    commtarget.setRetries(0);
    commtarget.setTimeout(5000);
  }
    public void clearResponse_text(){
        if (!response_text.isEmpty()) {
        response_text.clear();
        oid.clear();
        }
    }
    public String getResponse_text(int arg) {
        return response_text.get(arg);
    }
    public int getResponse_textSize() {
        return response_text.size();
    }
   public void send (String ipAddress, PDU pdu, boolean only_data) throws Exception
  {
    commtarget.setAddress(new UdpAddress(ipAddress + "/" + port));
    TransportMapping transport = new DefaultUdpTransportMapping();
    transport.listen();

    Snmp snmp = new Snmp(transport);

    ResponseEvent response;
    if (pdu.getType() == pdu.GETBULK) {
          response = snmp.getBulk(pdu, commtarget);
      }else if (pdu.getType() == pdu.GET) {
        response = snmp.get(pdu, commtarget);
      } else if (pdu.getType() == pdu.GETNEXT){
          response = snmp.getNext(pdu, commtarget);
    }else if (pdu.getType() == pdu.SET){
```

```
            response = snmp.set(pdu, commtarget);
        } else response=null;

        if (response != null) {
            PDU responsePDU = response.getResponse();
          if (responsePDU != null)
          {
           int errorStatus = responsePDU.getErrorStatus();
           String errorStatusText = responsePDU.getErrorStatusText();

           if (errorStatus == PDU.noError) {
           String resp = null;
             for (int i = 0; i < responsePDU.size(); i++) {
             if (only_data)  resp = responsePDU.get(i).getVariable().toString();
             if (!only_data) resp = responsePDU.get(i).toString();
             this.response_text.add(resp);
             this.oid.add(responsePDU.get(i).getOid().toString());
             }

              } else
           {

           this.response_text.add("Error: Request Failed ");
           }
          }
          else
          {
              this.response_text.add("Error: žádná SNMP odpověď");

          }
        }
        else
        {
            this.response_text.add( "Error: Agent Timeout ... ");

        }
        snmp.close();
      }
}
```

C.2 RouterJPanel.java

Zdrojový kód tohoto souboru je zde ve zkrácené verzi(celý obsahuje cca 750 řádků). Celý zdrojový kód je možné nalézt na přiloženém disku.

```
package cz.vutbr.feec.xzavod02;

public class RouterJPanel extends javax.swing.JPanel {
    public int i;
    public Request req = new Request();
    PDU ifNumberPDU= new PDU();
    PDU cmiHaRegTotalMobilityBindings  = new PDU();
    PDU cmiFaRegTotalVisitors = new PDU();
    PDU mipEntities = new PDU();
    PDU hostnamePDU = new PDU();
    PDU ifTablePDU    = new PDU();
    PDU HATablePDU    = new PDU();
    PDU FATablePDU    = new PDU();
    String hostname;
    CommunityTarget commTarget = new CommunityTarget();
    JDialogSNMPSettings settings;
    String ifcolumnNames[] =     {"ifIndex", "ifDescr", "ifIn", "ifOut"};
    String HAcolumnNames[] =     {"MN-host", "COA", "TimeGranted", "TimeRemaining"};
    String FAcolumnNames[] =     {"VisitorIPAddress", "HomeAddress", "HAAddress",
                                  "TimeGranted", "TimeRemaining"};
    EmptyJPanel ifJPanel,haJPanel,faJPanel;
    JScrollPane ifJScrollPane, haJScrollPane, faJScrollPane;
    JTable ifJTable, haJTable, faJTable;
    static Timer timer = new Timer();
```

```
boolean homeAgent, foreignAgent;

VariableBinding ifTableVB[] = {
        //Interfaces
    new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.1")),//ifIndex
    new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.2")),//ifDescr
    new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.10")),//ifInBytes
    new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.16"))//ifOutBytes
    };
VariableBinding HATableVB[] = {
        //Interfaces
new VariableBinding(new OID("1.3.6.1.2.1.44.1.6.3.1.1.1")),//haMobilityBindingMN
new VariableBinding(new OID("1.3.6.1.2.1.44.1.6.3.1.1.2")),//haMobBindingCOA
new VariableBinding(new OID("1.3.6.1.2.1.44.1.6.3.1.1.3")),//SourceAddress
new VariableBinding(new OID("1.3.6.1.2.1.44.1.6.3.1.1.7")),//TimeGranted
new VariableBinding(new OID("1.3.6.1.2.1.44.1.6.3.1.1.8"))//TimeRemaining
     };
VariableBinding FATableVB[] = {
        //Interfaces
new VariableBinding(new OID("1.3.6.1.2.1.44.1.5.3.1.1.1")),//faVisitorIPAddress
new VariableBinding(new OID("1.3.6.1.2.1.44.1.5.3.1.1.2")),//faVisitorHomeAddres
new VariableBinding(new OID("1.3.6.1.2.1.44.1.5.3.1.1.3")),//HomeAgentAddress
new VariableBinding(new OID("1.3.6.1.2.1.44.1.5.3.1.1.4")),//TimeGranted
new VariableBinding(new OID("1.3.6.1.2.1.44.1.5.3.1.1.5")),//TimeRemaining
new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.16"))//ifOutBytes
     };

public boolean loop_enabled = false; //na začátku je opakování vypnuto
private InetAddress adresa = null;
//InterfaceBytes graf;
public int interval = 5000;

public RouterJPanel(int index) {
    initComponents();
    i = index;

    homeAgent = false;
    foreignAgent = false;
    ipAddrjTextField.setText(SNMP4MobileIP.Routers.get(i).getIP());

    ifNumberPDU.setType(PDU.GET);
    ifNumberPDU.add(new VariableBinding(new OID("1.3.6.1.2.1.2.1.0")));

    ifTablePDU.setType(PDU.GETBULK);
    ifTablePDU.addAll(ifTableVB);

    HATablePDU.setType(PDU.GETBULK);
    HATablePDU.addAll(HATableVB);

    FATablePDU.setType(PDU.GETBULK);
    FATablePDU.addAll(FATableVB);

    mipEntities.setType(PDU.GET);
    mipEntities.add(new VariableBinding(new OID("1.3.6.1.2.1.44.1.1.1.0")));

    cmiHaRegTotalMobilityBindings.setType(PDU.GET);
    cmiHaRegTotalMobilityBindings.add(new VariableBinding(
            new OID("1.3.6.1.4.1.9.9.174.1.2.1.1.0")));

    cmiFaRegTotalVisitors.setType(PDU.GET);
    cmiFaRegTotalVisitors.add(new VariableBinding(
            new OID("1.3.6.1.4.1.9.9.174.1.1.1.1.0")));

    hostnamePDU.setType(PDU.GET);
    hostnamePDU.add(new VariableBinding(new OID("1.3.6.1.2.1.1.5.0")));

    commTarget.setCommunity(new OctetString("public"));
    commTarget.setVersion(SnmpConstants.version2c);
    commTarget.setRetries(1);
    commTarget.setTimeout(1000);
```

```
        req.commtarget = commTarget;

        settings = new JDialogSNMPSettings(null, true, commTarget,req);

        timer.schedule(new TimerTask() {
            public void run() {
                if (loop_enabled) {
                    onLoop();
                }
            }
        }, 0, interval);
}
    static public String customFormat(long value) {
        DecimalFormat myFormatter = new DecimalFormat("######");
        String output = myFormatter.format(value);
        return output;
   }
    private void getHostname() throws Exception{
               req.clearResponse_text();
                //zjisteni hostname
                //nastaveni hostnamu na tlacitku
                req.send(getIP(), hostnamePDU, true);
                hostname =  req.response_text.getFirst();
                RouterBtn.setText(hostname);
    }
    private void checkMIPEntities() throws Exception{

            req.clearResponse_text();
            req.send(getIP(), mipEntities, true);
            if (!req.response_text.getFirst().contains("noSuchObject")  &&
                !req.response_text.getFirst().contains("Error")) {

                int bajt = (int)req.response_text.getFirst().charAt(0);

                if ((bajt & 0x40) == 64) {
                    foreignAgent = true;
                } else {
                    foreignAgent = false;
                }

                if ((bajt & 0x20) == 32) {
                    homeAgent = true;
                } else {
                    homeAgent = false;
                }


        }

    }

    public void onLoop() {
        if (isOnline()) {
            RouterBtn.setBackground(Color.green);
            try {
                getHostname();
                checkMIPEntities();
                getIfTable();
                if (homeAgent) {
                   getHATable();
                }
                if (foreignAgent) {
                   getFATable();
                }
        } else if (!isOnline()) {
            RouterBtn.setBackground(Color.red);
        }
    }

    private boolean isOnline() {
```

```
        try {
            adresa = InetAddress.getByName(this.getIP());
            if (adresa.isReachable(2000)) {
                return true;
            } else {
               return false;
            }



        } catch (IOException ex) {
            return false;
        }

    }


    private void RouterBtnActionPerformed(java.awt.event.ActionEvent evt) {
        if (ipAddrjTextField.isEnabled()) {
         try {
                adresa = InetAddress.getByName(ipAddrjTextField.getText());
                ipAddrjTextField.setText(adresa.getHostAddress());
                loop_enabled = true;
                onLoop(); // okamžitě vykonat smyčku, pak dle intervalů
                ipAddrjTextField.setEnabled(false);
            } catch (UnknownHostException ex) {
                loop_enabled = false;
                JOptionPane.showMessageDialog(SNMP4MobileIP.getFrames()[0],
                "Neplatná IP adresa nebo hostname",
                  "Špatná IP adresa", JOptionPane.WARNING_MESSAGE);
                ipAddrjTextField.setEnabled(true);
            }
        } else if (!ipAddrjTextField.isEnabled()) {
            loop_enabled = false;
            ipAddrjTextField.setEnabled(true);
            RouterBtn.setBackground(Color.black);
        }
 }
}
```