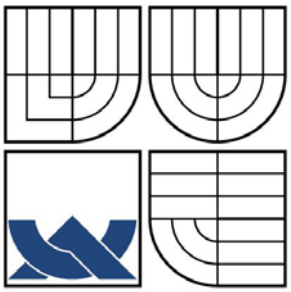
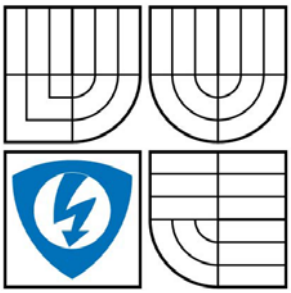


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

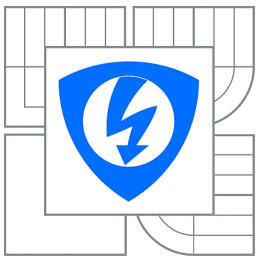
RIADENIE A KONFIGURÁCIA DATABÁZE QOS MIB
Managing and configurating of database QoS MIB

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. MIROSLAV MACURA
AUTHOR

VEDOUCÍ PRÁCE — ING. JIŘÍ HOŠEK
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Miroslav Macura	***ID:***	78546
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Správa a konfigurace zjednodušené databáze QoS MIB

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku protokolu SNMP a databází typu MIB (Management Information Base). Zaměřte se zejména na správu a konfiguraci (čtení, zápis) databáze MIB pomocí protokolu SNMP. Dále se seznamte se simulačním prostředím OPNET Modeler a navrhněte vhodný způsob implementace protokolu SNMP a zvolené databáze MIB do tohoto prostředí. Funkčnost implemetnace ověřte formou komunikačního modelu typu agent - manažer, v kterém bude manažer provádět dotazování agenta a vyčítat informace uložené v databázi MIB. Dosažené výsledky přehledně zdokumentujte.

DOPORUČENÁ LITERATURA:

[1] MAURO, D., SCHMIDT, K.: Essentials SNMP, Second Edition. Sebastopol: O'Reilly Media, 2005, ISBN: 0-596-00840-6.
[2] MILLER, A. M.: Managing Internetworks with SNMP. West Sussex: John Wiley & Sons, 1999, ISBN: 978-0764575181.
[3] WALSH, L.: SNMP MIB Handbook. Standwood: Wyndham Press, 2008, ISBN: 978-0-9814922-0-9.

Termín zadání:	29.1.2010	***Termín odevzdání:***	26.5.2010

Vedoucí práce: Ing. Jiří Hošek

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

UPOZORNĚNÍ:

ABSTRAKT

Hlavným cieľom tejto práce bolo naštudovať problematiku protokolu SNMP a databáz MIB, za účelom zamerania sa na správu a konfiguráciu databáze MIB pomocou protokolu SNMP. Táto práca sa zameriava na vytvorenie komunikačného modelu typu Manager – Agent v prostredí OPNET Modeler. V tomto modelovacom nástroji sa vytvára výmena informácií zo strany Managera na stranu Agenta prostredníctvom SNMP správ. Práca sa hlavne zameriava na vytvorenie správy SNMP podľa definovaných pravidiel, následne na vyčítanie informácii z databáze MIB na strane Agenta a ukladanie týchto informácií na strane Managera do vlastnej MIB databáze. Výmena informácií prebieha prostredníctvom typu správ GetRequest, GetNext a GetResponse. Tieto modeli sú naprogramované v jazyku C/C++ a následne implementované do prostredia OPNET Modeler.

KLÍČOVÁ SLOVA

SNMP, MIB, DiffServ, OPNET Modeler.

ABSTRACT

The main aim of this work was to study the issue of the SNMP protocol and MIB database in order to focus on management and configuration of MIB database using the SNMP protocol. The Master's thesis in detail describes especially creating a communications model of the Manager - Agent type in an OPNET Modeler. In this modeling tool we created exchange of information from the side of the Manager to the side of the Agent via SNMP packets. The Master's thesis is mainly focused on the creation of the SNMP packet according to defined rules, then on reading of information from the MIB database on the side of the Agent and saves this information to the Manager's MIB database. Exchange of informations is realized via the messages of type GetRequest, GetNext and GetResponse. These models are programmed in language C/C++ and then implemented into the environment of OPNET Modeler.

KEYWORDS

SNMP, MIB, DiffServ, OPNET Modeler

MACURA, M. *Správa a konfigurace zjednodušené databáze QoS MIB*.
Brno: Vysoké učení technické v Brne. Fakulta elektrotechniky a komunikačních technológii. Ústav telekomunikácií , 2010. 65 s., 7 s. príloh. Magisterská práce. Vedoucí práce byl Ing. Jiří Hošek.

Prohlášení

Prohlašuji, že svou magisterskou práci na téma „*Správa a konfigurace zjednodušené databáze QoS MIB*“ jsem vypracoval samostatně pod vedením vedoucího magisterské práce a s použitím odborné literatúry a ďalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatúry na konci práce.

Jako autor uvedenej magisterské práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne …….. …….

podpis autora

POĎAKOVANIE

Touto cestou by som chcel poďakovať vedúcemu magisterskej práce, Ing. Jiřímu Hoškovi , za motiváciu, cenné rady i pripomienky a priebežné konzultácie mojej práce.

V Brne dňa . . . . . ……
(podpis autora)

OBSAH

ZOZNAM OBRÁZKOV

1 Úvod

Pri súčasnom náraste používaných sieťových zariadení vznikajú nároky na ich správu a údržbu. Táto správa a údržba sa netýka iba pripojenia zariadenia do siete, správne nastavenie siete a detekcii chýb. Dôležité je taktiež priebežné monitorovanie prevádzky, sledovanie výkonu a využitie liniek s cieľom odhadnúť budúci vývoj a potrebu siete. Týmto monitorovaním získam informácie o výpadkoch, chybách, pretečení front a pod., ktoré môžu byť zasielané administrátorovi vo forme udalostí či alarmov. Monitorovaním siete a analýzou týchto dát zisťujeme odpovede potrebné pre chod prevádzky v sieti. Získavanie informácií o stavu siete, rýchlosti prenosu, spoľahlivosti spojenia nieje v oblasti komunikácii nič nového. Najpoužívanejším protokolom v súčasnosti pre riadenie a správu sieti je SNMP (Simple Network Management Protocol). Tento protokol je používaný v troch verziách a to SNMPv1, SNMPv2 a SNMPv3. Každý z týchto verzií má radu výhod a nevýhod. S ohľadom na porovnanie týchto výhod a nevýhod je najpoužívanejšia verzia protokolu pre správu sietí SNMPv2. Tento protokol dotazuje pomocou SNMP správ databázu MIB (Management Information Base) umiestenú na strane Agenta (smerovač) . V tejto databázy sú uložené monitorované informácie o siete. Niektorými z týchto monitorovacích informácií sú napr. priepustnosť siete, spoždenie siete, kolísanie siete a mnoho ďalších monitorovacích informácií. V tejto práci je popísaný postup vytvorenia štruktúry správy SNMPv2 v prostredí OPNET Modeler, následné naplnenie datami a konečné šifrovanie položky pre heslo v štruktúre správy a kódovaním aplikovaným na celú správu SNMPv2. Pre zabezpečenie prenášaného hesla je použitý symetrický šifrovací algoritmus DES (Data Encryption Standard) . Ako kódovací algoritmus je zvolený BER (Basic Encoding Rules). Cieľom kódovania je zápis zo syntaxe abstraktnej do syntaxe prenosovej. Po vytvorení správ na strane Agenta a Managera sú vytvorené zjednodušené databáze jak na strane Agenta, tak na strane Managera. Pomocou správ SNMP sa dotazuje databáza na strane Agenta pre získanie konkrétnych informácií, ktoré sú následne poslané Managerovi a uložené do jeho databázy. Výsledkom mojej diplomovej práce je vytvorenie reálnej správy SNMPv2 a prenos správ medzi Managerom a Agentom v prostredí OPNET Modeler, ktoré slúžia na získanie informácií z databáze MIB na strane Agenta.

2 SNMP

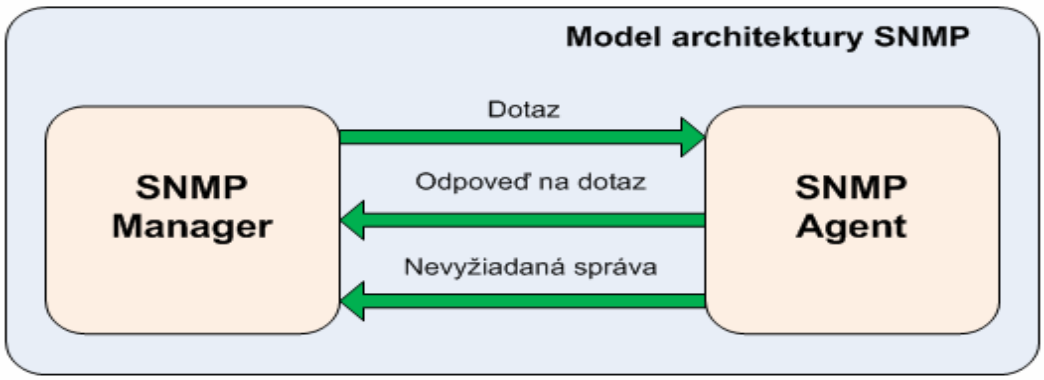
SNMP (Simple Network Management Protocol) je súčasťou sady internetových protokolov. Umožňuje priebežný zber najrôznejších dát pre potreby správy sietí a ich následné vyhodnocovanie. Na tomto protokolu je dnes založená väčšina prostriedkov a nástrojov pre správu sietí. SNMP[1] existuje v troch verziách: SNMPv1, v2, v3. Sledovanie sieti pomocou SNMP protokolu sa uskutočňuje formou komunikácie, typu otázka/odpoveď, medzi dvoma sieťovými entitami (MANGER-AGENT)[2].

2.1 SNMPv1

SNMPv1[1] je prvá implementácia SNMP protokolu. Je vysvetlená v RFC 1157 (Request For Comments 1157). Pracuje v medziach špecifikácie SMI (Structure of Managed Information). SNMPv1 pracuje nad protokolom UDP (User Datagram Protocol).

2.1.1 SNMPv1 operácie

Riadiaca stanica môže pomocou protokolu SNMP komunikovať so stanicami, na ktorých je spustený SNMP agent pomocou piatich správ. Slovo "Simple" v SNMP pochádza z toho faktu, že protokol používa pre komunikáciu len 5 správ[1]. Model architektúry SNMP je znázornený na obr. 2.1.



Obr 2.1 Model architektúry SNMP

GetRequest – žiadosť o informáciu, ktorú posiela Manager Agentovi k získavaniu informácie o stavu alebo hodnote istého objektu. Ide vlastne o príkaz "Read".

GetNextRequest – žiadosť o ďalšiu informáciu. Pretože informácie sú organizované hierarchicky, ide o žiadosť o informáciu na ďalšej, nižšej vrstve MIB štruktúry.

GetRespons – tento príkaz je vyslaný Agentom ako odpoveď na príkaz GetRequest, ktorým vracia vyžiadanú informáciu Managerovi.

SetRequest – Nastavuje hodnotu premennej v MIB Agentovi. Ide vlastne o príkaz "Write".

Trap – tento príkaz je vysielaný Agentom Managerovi ako oznámenie určitej významnej udalosti.

2.1.2 Nedostatky SNMPv1

Architektúra – Prvým nedostatkom SNMPv1 je jej samotná architektúra. Architektúra SNMPv1 používa pasívne agenty. Z toho vyplýva, že títo agenti sú aktivovaný len cez SNMP dotazy. Preto je v tejto komunikácie nutnosť pravidelného dotazovania tzv.polling. Táto komunikácia zaberá významnú šírku prenosového pásma, určeného pre užívateľské aplikácie.

Komunikácia Manager-Manager – SNMPv1 neposkytuje žiadne prostriedky pre komunikáciu medzi managery (manager-manager). Z toho vyplýva, že funkcia managementu nemôže byť distribuovaná medzi viacerými správcovskými konzolami (toto obmedzenie sťažuje použitie SNMP managementu vo veľkých spoločnostiach).

Objem dát – Nemožnosť získavania väčšieho množstva dát jediným dotazom, napr. celú smerovaciu tabuľku.

Komunikácia Agent – Manager – Agent síce môže upozorniť managera na problém, ale bez detailných informácií (manager sa potom musí dotazovať na podrobnosti, čo sa vlastne stalo) a naviac si nieje agent schopný overiť, že upozornenie bolo prijaté.

Slabé zabezpečenie protokolu – Systém autorizácie prístupu je veľmi primitívny. Heslá sa ukladajú pomocou takzvaného "community string", sú uložené a prenášané v nezašifrovanej podobe.

2.2 SNMPv2

Verzia SNMPv2[1] prekonala obmedzenia protokolu SNMPv1. Je taktiež známa ako SNMPv2 alebo SNMPv2p. Vznikla ako náhrada SNMPv1 a obsahuje zvýšenú úroveň bezpečnosti, schopnosť komunikácie MANAGER-MANAGER (príkaz Inform), umožňuje zasielať požiadavky na väčšie množstvo dat (GetBulk). Tento protokol ma početné výhody v porovnaní so SNMPv1. Kým SNMPv1 podporuje len 32-bit IP adresy, SNMPv2 podporuje 64-bit adresy. Protokol SNMPv1 podporuje len protokolové operácie (GetRequest, GetResponse, GetNext, Set a Trap), SNMPv2 zvýšil protokolové operácie o (Get Bulk, Inform). Dôležitou súčasťou protokolu SNMPv2 je otázka zabezpečenia. SNMPv2 neprenáša heslá v otvorenej podobe tak ako protokol SNMPv1, ale s určitým šifrovaním a tým je zabezpečená čiastočná ochrana odchytnutia hesiel počas komunikácie.

2.2.1 SNMPv2 operácie

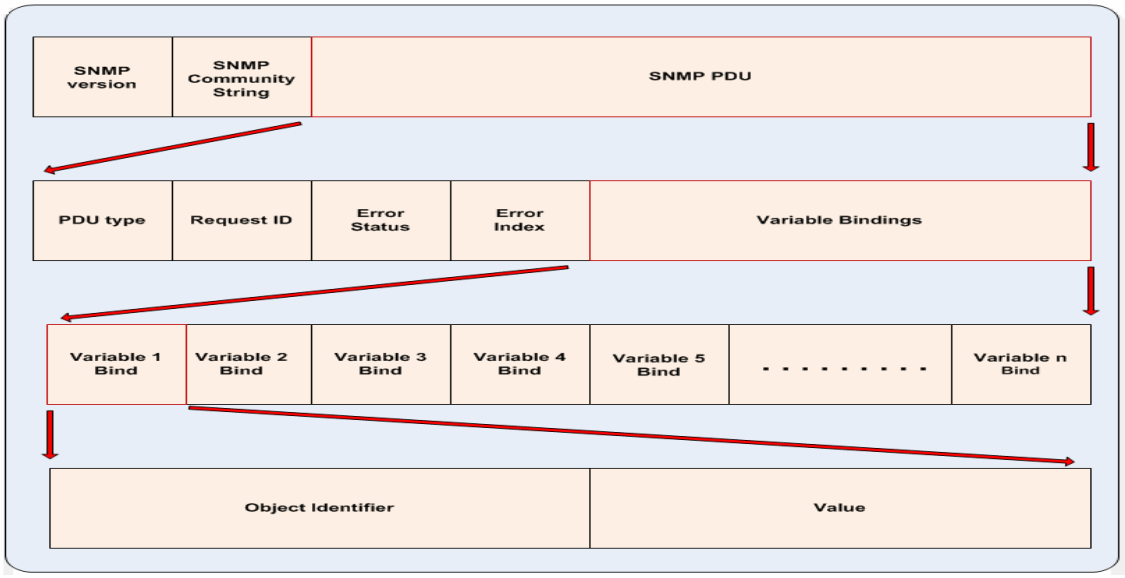
Protokol SNMPv2[1] implementuje nové správy ako GetBulk a Inform. Podporuje taktiež také isté správy ako protokol SNMPv2, len formáty týchto správ sú čiastočne zmenené[2].

GetBulk - operácia, ktorá je súčasťou SNMPv2. Umožňuje vyžiadať si k prečítaniu celú skupinu informácií z MIB (Management Information Base), čím sa urýchľuje komunikácia.

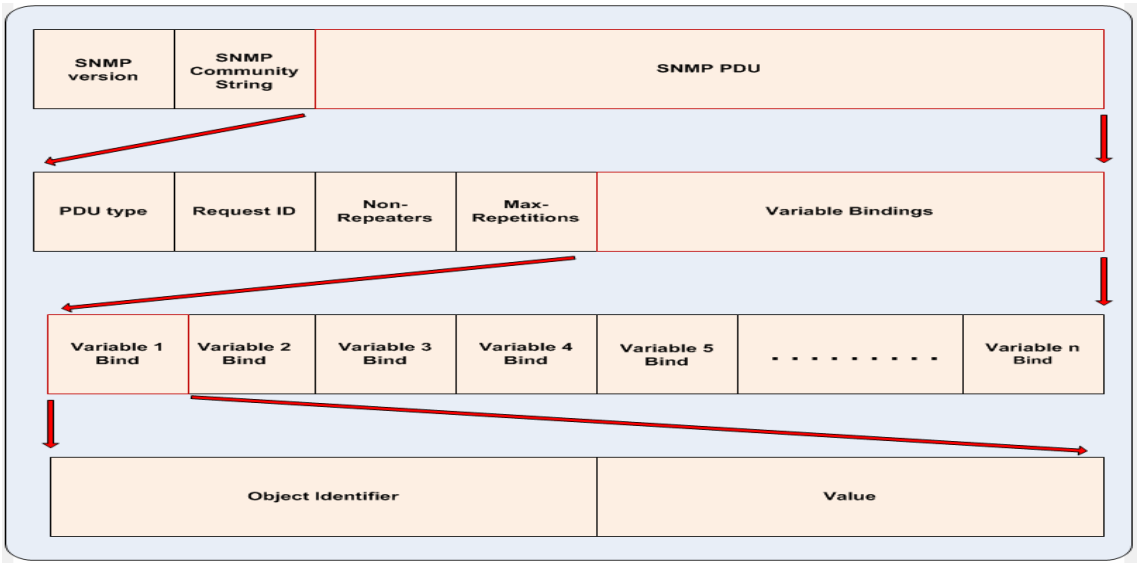
Inform - mechanizmus komunikácie typu Manager - Manager. Správa typu inform vyžaduje odpoveď príjemcu správy potvrdením o jej prijatí. Priebeh komunikácie správy inform sa podobá komunikácie správy typu get alebo set. Výhodou správy typu inform proti typu trap je potvrdenie o prijatí. Táto správa je nutná k prenosu medzi stanicami Manager – Manager, ak je potreba vzniku viac ako jednej NMS (Network management system) v sieti.

2.2.2 Formát správy SNMPv2

Formát SNMPv2[1] pre správy GetRequest, GetNext Request, GetResponse and Set ostal rovnaký ako u SNMPv1 (viď Obr. 2.2). U verzii SNMPv2 sa správa Trap používa na rovnaký účel ako u SNMPv1, ale má nový formát a je navrhnutý na nahradenie SNMPv1 Trap. Formát správy pre príkaz Trap je rovnaký ako pre správy GetRequest, GetNextReques, GetResponse, Inform a Set. U SNMPv2 sú implementované dve nové správy (GetBulk a Inform). Formát SNMPv2 pre správu Inform ostáva rovnaký ako pre správy GetRequest, GetNext, Response a Set (viď Obr. 2.2). Formát SNMPv2 pre správu GetBulk sa ale líši od ostatných správ. Tento formát správy zobrazuje Obr. 2.3.



Obr 2.2 Formát správ GetRequest, GetNext, Inform, GetResp, Set and Trap PDU



Obr 2.3 Formát správy GetBulk

2.3 SNMPv3

Špecifikácia protokolu SNMPv3[1] je popísaná v RFC 2571-2275. SNMPv3 bola prehlásená za štandard v roku 2004. Táto verzia konečne umožňuje uspokojivú autentifikáciu a šifrovanie pomocou mena a hesla. Pre návrh SNMPv3 bola použitá modulárna architektúra, kde každá SNMP entita obsahuje tzv. SNMP engine, ktorý implementuje funkcie odosielania správ, prijímania správ, šifrovania správ a riadiaci prístup ku spravovaným objektom. Tieto funkcie sú poskytované ako služby aplikáciám, ktoré spolu so SNMP engine tvoria vlastnú SNMP entitu. Sú tu doplnené mechanizmy overovania a šifrovania prenášaných dát (DES, MD5) k zamedzeniu možnosti „podstrčenia“ skôr odchytených správ pomocou časových značiek.

Medzi hlavné poskytované bezpečnostné vlastnosti patria[2]:

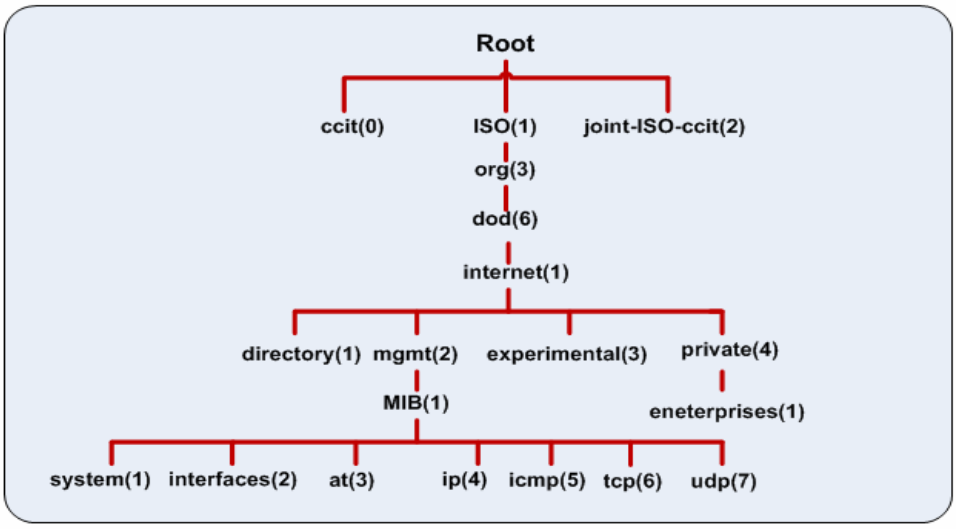
Integrita správ – pre overenie, či nebolo počas prenosu s paketom manipulováno.
Overenie identity – k zisteniu, či je správa z planého zdroja.
Šifrovanie – šifrovanie správ k zamedzeniu odposlechu komunikácie.

3 MIB

MIB (Management Information Base)[1] je hierarchicky organizovaná kolekcia informácií. Každé zariadenie je charakterizované svojimi špecifickými MIB objektmi. MIB databáza obsahuje tieto špecifické vlastnosti spravovaného zariadenia. K MIB objektom sa pristupuje pomocou network-management protokolu, v našom prípade prostredníctvom aplikačného protokolu SNMP. Objekty sú usporiadané do stromovej štruktúry, deliacu databázu do ľahko prehľadných časti. Keďže databáza má stromovú štruktúru, je možné základnú vetvu rozšíriť o ďalšie vetvy, odpovedajúce podporovaným funkciám a konkrétnym potrebám výrobcov. Databáza obsahuje na každej úrovni uzly, každý uzol a objekt v databáze má presne definovanú svoju polohu v stromovej štruktúre pomocou OID. OID je na každej pozícii označené nielen číslicou, ale i slovným popisom (viď Obr. 3.1). Popis súčasnej verzie MIB-2 je standartizován v RFC1213 a RFC 1155. MIB nieje databázou, iba definuje databázové vlastnosti pre uloženie a prácu s daty (niekedy je možné stretnúť sa s termínom Virtuálne dátové úložisko).

Koreňový uzol (root) je sám bez popisu, ale pod ním sú prinajmenšom tri dôležité uzly:

- iso(1) – spravovaný organizáciou ISO
- ccitt(0) – spravovaný organizáciou ITU-T (bývalé CCITT)
- joint-iso-ccitt(2) – spoločne spravovaný ISO a ITU-T



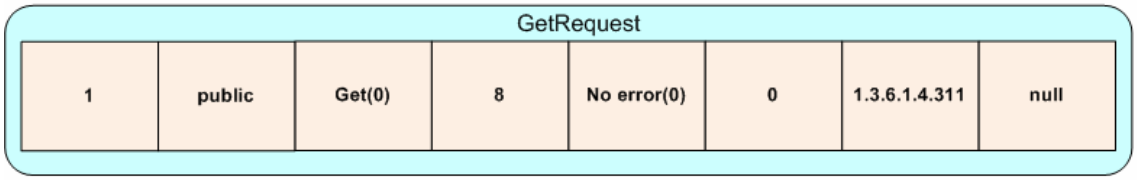
Obr 3.1 Stromová štruktúra MIB databáze

4 Správa databáze MIB

V tejto kapitole je popísaná správa databáze MIB (čítanie, zápis) prostredníctvom SNMP správ GetRequest, GetNextRequest, GetResponse a SetRequest. Prostredníctvom týchto správ prebieha dotazovanie databáze MIB na strane Agenta stranou Managera.

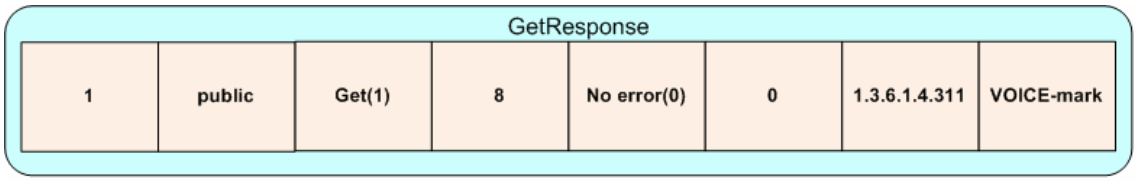
4.1 Realizácia sady príkazov v komunikačnom modeli

Pre zisťovanie a ukladanie uložených informácií v databáze MIB na strane Agenta slúžia správy GetRequest, GetNextRequest, GetResponse a Set. Význam týchto správ je možné nájsť v práci[1]. Ako bolo zmienené v kapitole 3, informácie v databáze MIB sú ukladané prostredníctvom identifikátorov „OID“, ktorým zodpovedajú dané hodnoty. V prípade, že sa dotazuje správca (Manager) na konkrétnu hodnotu v databáze MIB, musí poznať identifikátor (OID), pod ktorým je táto hodnota uložená. Ak sa administrátor na strane Managera dotazuje na jednu konkrétnu hodnotu, dotazuje sa prostredníctvom správy typu „GetRequest“. Štruktúra tejto správy je znázornená na Obr.2.2. Najdôležitejšia časť štruktúry tejto správy je položka „ObjectIdentifier“, „Value“ a „PDUtype“. Význam všetkých položiek štruktúry správy je vysvetlený v práci[1] a pre účel vysvetlenia sú ostatné položky správy nevýznamné. Na strane Managera je vytvorená správa „GetRequest“, ktorý žiada o konkrétnu hodnotu pod daným „OID“ ako je vidieť na Obr. 4.1. Manager žiada o hodnotu na stane Agenta, ktorá je uložená pod „OID = 1.3.6.1.4.311“. Pretože sa jedná o správu typu „GetRequest“, do položky typ správy je vložená „0“, ktorá identifikuje správu typu „GetRequest“. Položka správy „Value“ je prázdna z dôvodu, že sa jedná o získanie tejto hodnoty odpovedajúcej identifikátoru „OID = 1.3.6.1.4.311“.



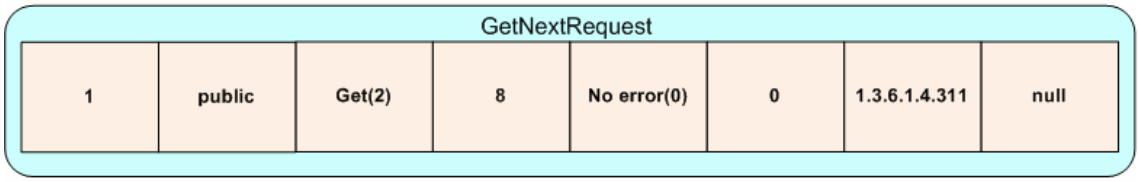
Obr 4.1 Správa typu GetRequest.

Po prijatí správy na stranu Agenta dochádza k vyčítaniu „OID“ zo štruktúry správy a dochádza k následnému prehľadávaniu databáze MIB s odpovedajúcim „OID“ a jemu priradenej hodnote. Agent zistí, že sa jedná o žiadosť o vyhľadávanie z databáze MIB podľa identifikátora v položke „PDUtype = 0“. Po vyhľadaní danej hodnoty z databáze MIB, Agent vytvorí štruktúru správy na strane Agenta, pričom do „OID“ vloží hľadané „OID = 1.3.6.1.4.311“ a jemu priradenú hodnotu „Value = Voice-mark“ ako je možné vidieť na Obr. 4.2. Že sa jedná o typ správy „GetResponse“ identifikuje položka „PDUtype = 1“.



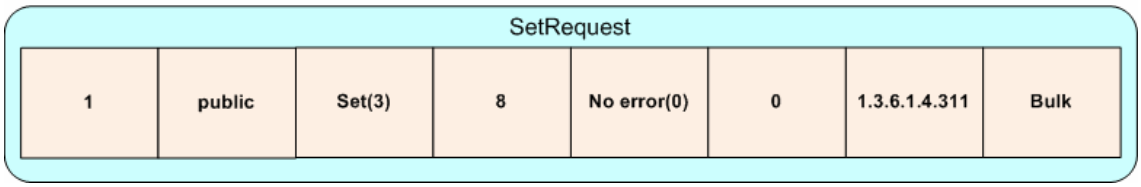
Obr 4.2 Správa typu GetResponse.

V prípade, že sa jedná o typ správy „GetNextRequest“, jedná sa o získanie viacerých informácií zo strany Managera. Touto správou sa umožňuje získavať viacero ďalších „OID“ v databáze MIB. Formát správy „GetNextRequest“ je vidieť na Obr.4.3. Pre zistenie typu správy odpovedá položka „PDUtype = 2“. Týmto je dosiahnuté informovanie Agenta o rôznych typov správ SNMP.



Obr 4.3 Správa typu GetNextRequest.

V predchádzajúcich príkladoch bol ukázaný príklad vyčítania (Read) z databáze MIB prostredníctvom správ typu „GetRequest, GetNextRequest“. V prípade, že administrátor (Manager) požaduje zápis do databáze MIB, udeje sa to prostredníctvom správy „SetRequest“. Do štruktúry správy sa nastaví dané „OID“, ktoré sa má ukladať do databáze na strane Agenta a jej odpovedajúca hodnota. Pre identifikáciu správy, ktorý slúži pre zápis (write) a nie (read) slúži položka správy „PDUtype = 3“. Podľa tohto identifikátora pozná Agent, že sa nejedná o vyčítanie hodnoty odpovedajúcej „OID = 1.3.6.1.4.311“ ale o to, že sa jedná o zápis v našom príklade „Value = Bulk“ do databáze MIB. Príklad tejto správy je možné vidieť na Obr. 4.4.



Obr 4.4 Správa typu SetRequest

V predchádzajúcich príkladoch boli teoreticky vysvetlené správy pre vyčítanie (Read) a pre zápis (Write) databáze na strane Agenta. Podľa teoretických poznatkov zmienených v predchádzajúcich kapitolách bude vytvorený v prostredí OPNET Modeler komunikačný model typu Manager – Agent, v ktorom budú implementované príkazy zmieňované vyššie.

5 Realizácia komunikačného modelu Manager - Agent

5.1 Realizácia štruktúry do komunikačného modelu

V tejto časti práce je opísaný postup realizácie aplikačného protokolu SNMPv2 v prostredí OPNET Modeler (OM). Protokol SNMPv2 je aplikovaný do komunikačného modelu OM vytvoreného v študentskej práci[3]. S použitím uvedeného postupu je možné navrhnúť štruktúru správy protokolu SNMPv2.

5.1.1 Simulačný program OPNET Modeler

OPNET Modeler[4] je softwarové vývojové prostredie vhodné pre návrh a analýzu komunikačných sietí. Patrí do celkového softwarového balíku Opnet (Optimum Network Performance) od americkej firmy Opnet Technologies.

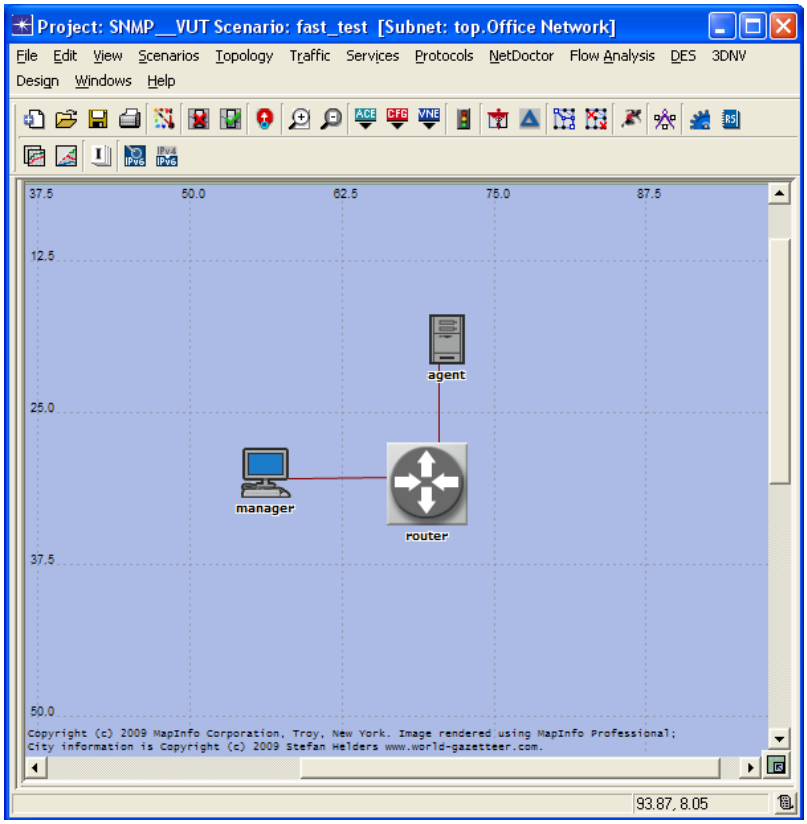
OM uľahčuje návrh komunikačnej siete s možnosťami dimenzovania sieťovej komunikácie, modelovania komunikačných protokolov a simuláciu chovania sieťových aplikácií atd. s veľkou možnosťou flexibility testovania. Veľkou výhodou programu je jeho výkonnosť a pružnosť.

Celý systém je hierarchicky a objektovo orientovaný, v grafickom prostredí názorne zobrazuje reálne rozmiestnenie jednotlivých sieťových komponentov a na najnižšej úrovni je chovanie jednotlivých modelov zapísané v jazyku C/C++. Správnosť zapísaného kódu je možné okamžite overiť na vstavanom mechanizme ladenia a hľadania chýb – Debugger. Ten poskytuje textovú aj vizuálnu odozvu konkrétnych dejov prebiehajúcej komunikácie na programovacej úrovni. Ďalej obsahuje široké možnosti v oblasti simulácie a následnej analýzy dosiahnutých výsledkov. Simulácia prebieha s určitou časovou kompresiou (zrýchlením), preto je možné mesačné chovanie siete nasimulovať za pár hodín.

K softwareu OM sú navyše dodávané objemné knižnice simulačných modelov so širokou škálou reálnych sieťových prvkov .Najväčšiu výhodu OPNET Modeler-u nájdeme, v otvorenosti zdrojového kódu všetkých simulačných modelov. Vďaka tomu môžeme ovplyvňovať funkciu a výsledné chovanie jednotlivých komponentov. Sieťové prvky môžeme ďalej upravovať, rôzne kombinovať alebo si naprogramovať nový podľa vlastných potrieb.

5.1.2 Model simulovanej komunikačnej siete

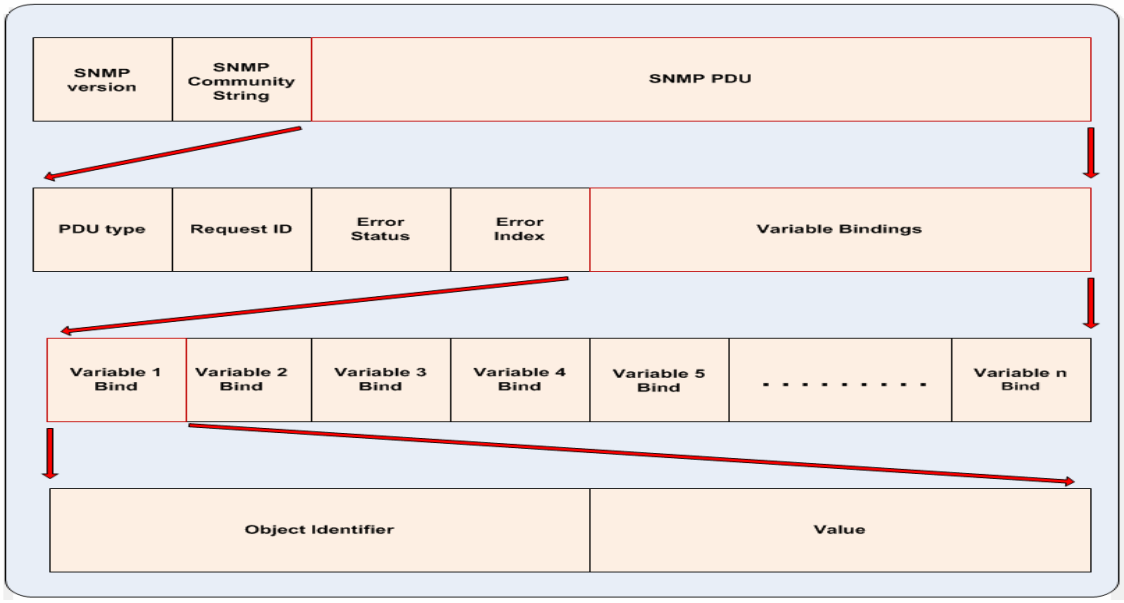
Postup vytvorenia simulačného modelu siete, pre komunikačné rozhranie, bude prezentovaný v programe OPNET Modeler, v použitej verzii 14.5.A PL5 (Build 4523), pod operačným systémom Windows1 XP Professional, verzia 2002, Service Pack 2. Vytvorenie modelu siete je popísané v práci[3]. Simulačný modul siete je zobrazený na Obr. 5.1, model siete je typu Manager – Klient, medzi ktorými sa budú vymieňať správy typu (Otázka - Odpoveď).



Obr 5.1 Typ simulačného modelu vytvorený v prostredí OM

5.1.3 Formát dátovej správy

Komunikácia v navrhnutom simulačnom modeli v prostredí OM je závislá na výmene správ. V našom modeli[3] prebieha výmena týchto správ prostredníctvom protokolu SNMPv2. Tento model simulačnej sieti je typu Manager – Agent. Manager posiela požiadavky Agentovi a ten mu nane odpovedá prostredníctvom SNMP správ[1]. Správa SNMP je vytváraná dynamicky, správa sa vždy jedinečne vytvorí a naplní užívateľskými datami. Štruktúra správy protokolu SNMPv2 bola navrhnutá podľa modelu správy zobrazeného na Obr. 5.2.



Obr 5.2 Formát správ GetRequest, GetNext, Inform, GetResp, Set and Trap PDU

Navrhnutá štruktúra je vložená do Header Bloku v Process Modelu OM. Táto štruktúra je zobrazená na Obr. 5.3.

```
typedef struct variablebin{
  char *ObjectID;
  char *ValueID
} VARIABLEBIN;


typedef struct snmppdu{
  int PDUtype;
  int RequestId;
  int ErrorStatus;
  int ErrorIndex;
  VARIABLEBIN *VariableBin;
} SNMPPDU;


typedef struct snmppaket{
  int Version;
  char *Community;
  SNMPPDU *SNMPpdu;
} SNMPPAKET;
```

Obr 5.3 Štruktúra správy SNMPv2

ObjectID – Identifikačné číslo žiadaného objektu.

ValueID – Hodnota vyčítaná z daného objektu.

PDUtype – Typ dátovej jednotky PDU.

RequestID – Identifikátor správy.

ErrorStatus – Identifikátor chyby.

ErrorIndex – Index chyby.

VariableBin – Štruktúra VARIABLEBIN.

Version – Verzia správy SNMP.

Community – Prenášané heslo.

SNMPpdu – Štruktúra SNMPPDU.

Teraz máme definovanú štruktúru SNMP správy. Ak chceme s touto štruktúrou pracovať, je nutné pre túto štruktúru alokovať pamäť. Do bloku TV priradíme kód na Obr. 5.4, ktorým jednoznačne hovoríme, že premenná datapaket bude odkazovať na nami vytvorenú štruktúru SNMPPAKET [3] .

```
SNMPPAKET* datapaket;
```

Obr 5.4 Ukazovateľ datapaket na štruktúru SNMPPAKET.

Pred samotným plnením nami vytvorenej štruktúry je nutné pre túto štruktúru alokovať pamäť. Alokácia pamäti sa v OM uskutoční pomocou funkcie op_prg_mem_alloc. Nasledujúci kód na Obr. 5.5 zobrazuje dynamickú alokáciu pamäti pre štruktúru SNMPPAKET.

```
datapaket = (SNMPPAKET *) op_prg_mem_alloc (sizeof (SNMPPAKET));
datapaket->SNMPpdu = (SNMPPDU *) op_prg_mem_alloc (sizeof (SNMPPDU));
datapaket->SNMPpdu->VariableBin = (VARIABLEBIN *) op_prg_mem_alloc (sizeof (VARIABLEBIN));
datapaket->Community = (char *) op_prg_mem_alloc (strlen(&community_string_m));
datapaket->SNMPpdu->VariableBin->ObjectID = (char *) op_prg_mem_alloc (strlen(""));
datapaket->SNMPpdu->VariableBin->ValueID = (char *) op_prg_mem_alloc (strlen(" "));
```

Obr 5.5 Alokácia pamäte pre štruktúru SNMPPAKET.

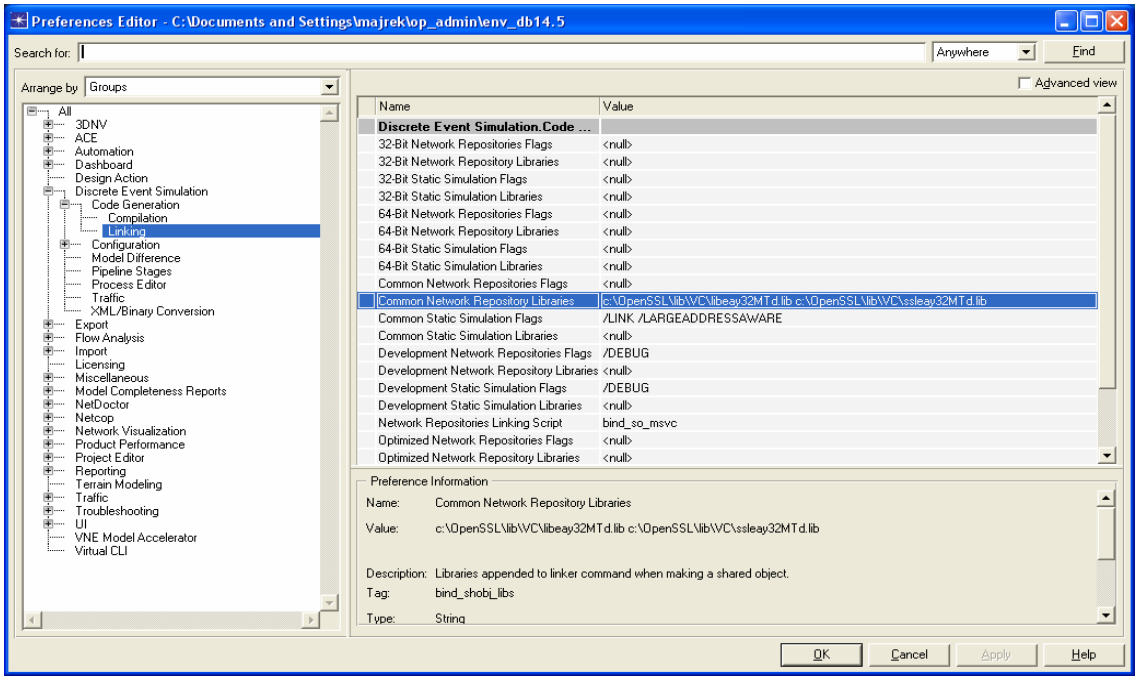
Po úspešnej alokácií štruktúry SNMPPAKET už môžeme túto štruktúru naplniť. Naplnenie tejto štruktúry zobrazuje nasledujúci Obr. 5.6.

```
datapaket->Version=1;
datapaket->Community = &community_string_m;
datapaket->SNMPpdu->PDUtype=1;
datapaket->SNMPpdu->RequestId=1;
datapaket->SNMPpdu->ErrorStatus=0;
datapaket->SNMPpdu->ErrorIndex=0;
sprintf (datapaket->SNMPpdu->VariableBin->ObjectV, ".1.3.6.1.2.1.97.1.2.6.1.3");
```

Obr 5.6 Naplnenie štruktúry SNMP datami.

5.2 Šifrovanie položky Community pomocou algoritmu DES

Spáva SNMPv2 má od svojej staršej verzie protokolu SNMPv1 zlepšenie v prenose hesla sieťou prostredníctvom šifrovania DES (Data Encryption Standard). Jedná sa o symetrický šifrovací algoritmus. V súčasnosti je táto šifra považovaná za nespoľahlivú, pretože používa kľúč len o dĺžke 64 bitov, z toho 8 je kontrolných a 56 efektívnych. Naviac obsahuje algoritmus slabiny, ktoré ďalej znižujú bezpečnosť šifry. Vďaka tomu je možné šifru prelomiť útokom hrubej sily za menej ako 24 hodín. Jedná sa teda o čiastočné zlepšenie autentizácie správy SNMPv2 oproti správe SNMPv1 v komunikačných sieťach. Položka Community u správy SNMPv2 je šifrovaná algoritmom DES prostredníctvom knižnice Win32OpenSSL-0_9_8k[5]. Pre správnu funkčnosť algoritmu DES je nutné nainštalovať a následne nastaviť v OM cesty k používanej knižnice Win32OpenSSL-0_9_8k. Je nutné nastaviť u položky Common network Respository Libraries v menu Preferencies cestu k dvom hlavným knižniciam a to libeay32MTd.lib a ssleay32MTd.lib. Na Obr. 5.7 je vidieť správne nastavenie.



Obr 5.7 Konfigurácia knižnice algoritmu DES

Adresár openssl musí byť nahraný na správne miesto v adresárovej štruktúre OPNET Modeler-u (OPNET\14.5.A\sys\include). Následne je nutné vložiť knižnicu algoritmu DES do položky Header Block v OM. Nasledujúci kód je znázornený na Obr.5.8.

```
#include <openssl/des.h>
```

Obr 5.8 Vloženie knižnice openssl pre DES v Header Blocku.

Pre správnu funkciu používaných príkazov algoritmu v OM je nutné vložiť kód na Obr. 5.9 do položky TV (Temporary Variables).

```
DES_key_schedule *schedule;
char *key;
```

Obr 5.9 Definovanie ukazovateľov v položke TV.

Po správnom nastavení OM môžeme následne vložiť kód šifrovania do simulačného prostredia, konkrétne to položky SEND v proces modelu OM. Nasledujúci kód zobrazuje správne zašifrovanie a následné dešifrovanie položky Community v správe SNMPv2[3]. Kód pre šifrovanie a dešifrovanie DES je vidieť na Obr. 5.10.

```
// Alokovanie pamäte pre šifrovací kľuč
schedule = op_prg_mem_alloc(sizeof(DES_key_schedule));
// Nastavenie šifrovacieho kľúča
key = "kluc klu";
// Dopočítanie kľúča „key“ na 8bitov
DES_key_sched((const_DES_cblock *)key, schedule);
// Šifrovanie reťazca Community
DES_ecb_encrypt((const_DES_cblock*)datapaket->Community,(const_DES_cblock*)datapaket-
>Community, schedule, 1);
// Dešifrovanie reťazca Community
DES_ecb_encrypt((const_DES_cblock*)datapaket->Community,(const_DES_cblock*)datapaket-
>Community, schedule, 0);
```

Obr 5.10 Implementácia DES do komunikačného modelu.

Po implementácii tohto kódu do prostredia OPNET Modeler je možné vidieť správnu funkčnosť tohto algoritmu výpisom v simulačnej konzoli. Kliknutím na Configure/ Run Disrete Event Simulation zvolíme dobu simulácie 1 minuta a následne kliknutím Run Simulation sa spustí simulácia tejto sieti. Následný výpis simulačnej konzoly je vidieť na Obr. 5.11.

Výpis na strane Managera:

Packet Version: 0
Packet Community: Opnet
Packet Community-DES: V¼n{Ø„ï-ƒ□
Packet PDUtype: 0
Packet RequestId: 1
Packet ErrorStatus: 0
Packet ErrorIndex: 0
Packet ObjectV: 1.3.6.1.4.1.26.80.1.2.7.3.2.0
Packet ValueV: aaaaaa

Výpis na strane Agenta:

Packet Version: 0
Packet Community-DES: V¼n{Ø„ï-ƒ□
Packet Community: Opnet
Packet PDUtype: 0
Packet RequestId: 1
Packet ErrorStatus: 0
Packet ErrorIndex: 0
Packet ObjectV: 1.3.6.1.4.1.26.80.1.2.7.3.2.0
Packet ValueV: aaaaaa

Obr 5.11 Výpis simulačnej konzoly pre DES.

Z výpisu je možné vidieť, že sa kóduje text „Opnet“ , ktorý je vidieť u položky Packet Community. Zakódované heslo „Opnet“ pomocou DES algoritmu je znázornené u položky „Packet Community-DES: V¼n{Ø„ï-‡□“. Tento výpis nám znázorňuje správnu funkčnosť implementácie algoritmu DES do prostredia OPNET Modeler.

5.3 Kódovanie správy SNMPv2 algoritmom BER

The Basic Encoding Rules (BER)[6] je jeden z kódovacích formátov definovaných ako časť ASN.1 špecifikovaných ITU v X.690. Najzákladnejšie pravidlo hovorí, že každé pole je zakódované v troch častiach: typ, dĺžka a data. Typ určuje dátový typ poľa pomocou jedného bajtu identifikátora. Identifikátor bajtu pre správu SNMP je znázornený na Obr. 5.12, ktorý znázorňuje niektoré dátové typy u správ SNMP a ich priradené identifikátory[7]. Dĺžka určuje dĺžku v bajtoch danej hodnoty a data je aktuálna hodnota (číslo, reťazec, OID, atď.).

Primitive ASN.1 Types	***Identifier in hex***
INTEGER	02
BIT STRING	03
OCTET STRING	04
NULL	05
OBJECT IDENTIFIER	06

Constructed ASN.1 type	***Identifier in hex***
SEQUENCE	30

Primitive SNMP application types	***Identifier in hex***
IpAddress	40
Counter (Counter32 in SNMPv2)	41
Gauge (Gauge32 in SNMPv 2)	42
TimeTicks	43
Opaque	44
NsapAddress	45
Counter64 (available only in SNMPv2)	46
Uinteger32 (available only in SNMPv2)	47

Context-specific types within an SNMP Message	***Identifier in hex***
GetRequest-PDU	A0
GetNextRequestPUD	A1
GetResponse-PDU (Response-PDU in SNMPv 2)	A2
SetRequest-PDU	A3
Trap-PDU (obsolete in SNMPv 2)	A4
GetBulkRequest-PDU (added in SNMPv 2)	A5
InformRequest-PDU (added in SNMPv 2)	A6
SNMPv2-Trap-PDU (added in SNMPv 2)	A7

Obr 5.12 Dátové typy SNMP a ich priradené identifikátory BER

Data sa v BER[6] kódujú vždy v štruktúre pomocou troch polí: typ dat, dĺžka a hodnôt, ktorá je vidieť na Obr. 5.13

Obr 5.13 Štruktúra kódovania BER

Pre implementáciu kódovania BER v prostredí OPNET Modeler bolo nevyhnutné vytvoriť nasledujúce funkcie pre správnu funkciu tohto kódovania. Vytvorené funkcie sú jak pre zakódovanie štruktúry správy, tak i pre rozkódovanie štruktúry správy[8].

Pre zakódovanie štruktúry SNMPv2 v OM sú implementované tieto funkcie:

- SaveIntToBuff
- SaveBERInteger
- SaveBERString
- SaveBEROID
- SaveBERVarBind
- SaveBERVarBindList
- SaveBERSNMPDU
- SaveBERSNPMMessage

Pre rozkódovanie štruktúry SNMPv2 v OM sú implementované tieto funkcie:

- GetBerInteger
- GetBERString
- GetBEROID
- GetBerVarBind
- GetBerVarBindList
- GetBerSNMPDU
- GetBerSNMPMessage

5.3.1 Funkcie použité pre kódovanie BER

Funkcia SaveIntToBuff (int SaveIntToBuff(char * buff, int i)**)**

Táto funkcia v prostredí OM zisťuje počet bajtov potrebných pre vyjadrenie číselnej hodnoty, ktorú potrebujeme vkladať do pola "Dĺžka dat" pri kódovaní BER. Touto funkciou vieme zistiť počet bajtov potrebných pre vyjadrenie daného čísla. Funkcia pri každej z týchto podmienok ukladá do príslušnej pamäte *(BUFF) jednotlivé čísla rozdelené po jednom bajte.

V prípade, že je číslo menšie ako – 255, vracia jeden bajt.
– 65535, vracia dva bajty.
– 16777215**,** vracia tri bajty.
– 4294967295, vracia štyri bajty.

V prípade, že sa jedná o 4 bytové číslo, ukladanie jednotlivých bytov do pola „buff" je znázornené na Obr. 5.14.

```
*(buff) = (char)((i&0xFF000000) / 0x1000000);// uložíme 1. byte do pola *buff.
*(buff + 1) = (char)((i&0xFF0000) / 0x10000);// uložíme 2. byte do pola *buff na pozíciu o jedna väčšiu.
*(buff + 2) = (char)((i&0xFF00) / 0x100);// uložíme 3. byte do pola *buff na pozíciu o dva väčšiu.
*(buff + 3) = (char)(i&0xFF);//uložíme 4. byte do pola *buff na pozíciu o tri väčšiu.
return 4 ;// vracia 4 byte.
```

Obr 5.14 Funkcia SaveIntToBuff.

Funkcia SaveBERInteger (int SaveBERInteger(char* buff, int i)**)**

Úlohou tejto funkcie je načítať vstupné číslo a následne uložiť toto číslo prostredníctvom vyjadrenia troch polí (typ dat, dĺžka a hodnota) do pamäti BUFF. Pomocou funkcie „SaveIntToBuff" zistíme počet potrebných alokovaných bajtov a zápis hodnôt do pamäte BUFF. Zapisovanie do pamäte sa deje prostredníctvom ukazovateľov a pohybom v pamäti. Ak sa jedná o číslo, do pamäti sa ukladá hodnota typu 0x02, ktorá je reprezentovaná podľa pravidiel v kódovaní BER. Nasledujúci kód tejto funkcie je znázornený na Obr. 5.15.

```
size = SaveIntToBuffM(buff+2, i); // do tretej pozície "hodnota" ukladáme číslo.
*buff = 0x02; //do prvého pola uložíme označenie typu: číslo.
*(buff+1) = size; //uložíme počet bytov na ktorých je číslo uložené.
return size+2; //vratíme velkosť bloku BER
```

Obr 5.15 Funkcia SaveBERInteger.

Funkcia SaveBERString (int SaveBERString(char* buff, char* str)**)**

Princíp tejto funkcie je podobný funkcie predchádzajúcej. Načítaný vstupný reťazec sa vyjadrí prostredníctvom vyjadrenia troch polí (typ dat, dĺžka a hodnota) do pamäti BUFF. Ak sa jedná o prázdny reťazec, do pamäti sa uloží hodnota typu 0x05 a následne počet bajtov. V prípade, že sa jedná o prázdny reťazec, kódovaním BER sa vyjadrí prázdny reťazec iba prostredníctvom dvoch bajtov. Hodnota 0x05 (NULL) reprezentuje prázdny reťazec, v prípade, že sa nejedná o prázdny reťazec, uloží sa hodnota typu 0x04 (Octet String) podľa pravidiel BER. Táto funkcia je znázornená na Obr. 5.16.

```
If (str == NULL)//ak je reťazec NULL {
                    *buff = 0x05; //nastavíme označenie typu NULL
                    *(buff + 1) = 0x00; //dĺžka je 0, hodnota sa neudáva
                    return 2; //vrátime velkosť bloku BER }
else {    buff = 0x04; //nastavíme označenie typu Text
                    *(buff + 1) = strlen(str) ;//uložíme dĺžku textu
                    memcpy((buff + 2), str, strlen(str)); //vložíme text do pola *buff
                    return strlen(str)+2;//vrátime velkosť bloku BER }
```

Obr 5.16 Funkcia SaveBERString.

Funkcia SaveBEROID (int SaveBEROID(char* buff, char* oid)**)**

Činnosť funkcie je založená na rovnakom princípe ako predchádzajúce funkcie. Vstupný reťazec reprezentuje OID číslo (typ pre BER 0x06). Pre toto OID je potrebná alokácia pamäte, preto vytvárame pamäť TMP do ktorej ukladáme OID číslo. Prostredníctvom cyklu FOR sa nastavuje hodnota nula namiesto bodiek v OID na ohraničenie znakov a jej následné spracovanie (nula prezentuje koniec reťazca, v našom prípade koniec jedného znaku). V prípade prvých dvoch znakov OID sa kódujú tieto prvé dve čísla ako jedno číslo, ktoré vypočítame ako (40*tmp1+tmp2), podľa pravidiel BER kódovania. Následne pomocou tohto cyklu sa prehľadávajú ostatné znaky OID, pričom pomocou funkcie „SaveIntToBuff" zisťujeme počet bajtov jednotlivých znakov v celom čísle OID. Kód tejto funkcie je značne objemný a z toho dôvodu je vysvetlený v prílohe[A1] s popiskami ku kódu.

Funkcia SaveBERVarBind (int SaveBERVarBind(char *buff ,VARBIN* vb)**)**

Podľa kódovacích pravidiel spadajú v našom prípade hodnoty Identifikátor objektu a jeho hodnota v kódovaní pod položku "VarBind". Táto položka je reprezentovaná identifikátorom (0x30- Sequence typ) a následne počtom bajtov potrebných pre vyjadrenie Identifikátora objektu a jej hodnoty. Táto funkcia vracia počet bajtov pre vyjadrenie Identifikátora objektu a jej hodnoty (vrátane dvoch bajtov potrebných pre vyjadrenie VarBind). Táto funkcia pracuje s funkciami „SaveBEROID" a „SaveBERString", ktoré sú popísané vyššie. Táto funkcia je znázornená na Obr. 5.17.

```
*buff = 0x30; //nastavíme typ VarBind.
*(buff+1)=0; //nastavíme dĺžku 0, budeme zvyšovať.
//uložíme OID do *buff a pričítame dĺžku bloku OID k dĺžke aktuálneho bloku.
*(buff+1)+=SaveBEROIDM(buff+2,vb->ObjectV);
// uložíme ValueV, nastavíme dĺžku aktuálneho bloku.
*(buff+1)+=SaveBERStringM(buff+2+*(buff+1),vb->ValueV);
return *(buff+1)+2; // vraciame počet bytov.
```

Obr 5.17 Funkcia SaveBERVarBind.

Funkcia SaveBERVarBindList (int SaveBERVarBindList(char *buff ,VARBIN* vb)**)**

Táto funkcia má rovnakú funkciu ako funkcia "SaveBERVarBind". Podľa kódovacích pravidiel popisuje VarBindList premenné typu VarBind. Táto položka je reprezentovaná identifikátorom (0x30- Sequence typ) a následne počtom bajtov potrebných pre vyjadrenie VarBind. Táto funkcia vracia počet bajtov položky VarBindList (vrátane dvoch bajtov VarBindList), ktorú je možné vidieť na Obr. 5.18.

```
*buff = 0x30; // nastavíme typ VarBind.
*(buff+1)=0; // nastavíme dĺžku 0, budeme zvyšovať.
*(buff+1)+=SaveBERVarBindM(buff+2,vb); // pripočítame dĺžku VarBind k dĺžke aktuálneho bloku.
return *(buff+1)+2; // vraciame počet bytov.
```

Obr 5.18 Funkcia SaveBERVarBindList.

Funkcia SaveBERSNMPDU (int SaveBERSNMPDU(char *buff ,snmppdu* pdu)**)**

Funkcia vracia počet bajtov vyhradených pre položky RequestID, ErrorStatus, ErrorIndex a VarBindList (vrátane dvoch bajtoch pre zápis SNMP PDU). Funkcia zapíše do pamäti identifikátor 0x30 a následne počet bajtov, ktoré sú vyhradené pre položky RequestID, ErrorStatus, ErrorIndex a VarBindList. Táto funkcia používa funkcie "SaveBERInteger" a "SaveBERVarBindList" pre zistenie počtu bajtov potrebných pre vyjadrenie podradenej štruktúry . Túto funkciu je vidieť na Obr. 5.19.

```
// uložíme RequestId a pripočítame dĺžku RequestId k dĺžke aktuálneho bloku.
*(buff+1)+=SaveBERIntegerM(buff+2+*(buff+1),pdu->RequestId);
// uložíme ErrorStatus a pripočítame dĺžku ErrorStatus k dĺžke aktuálneho bloku.
*(buff+1)+=SaveBERIntegerM(buff+2+*(buff+1),pdu->ErrorStatus);
// uložíme ErrorIndex a pripočítame dĺžku ErrorIndex k dĺžke aktuálneho bloku.
*(buff+1)+=SaveBERIntegerM(buff+2+*(buff+1),pdu->ErrorIndex);
// uložíme VariableBin a pripočítame dĺžku VariableBin k dĺžke aktuálneho bloku.
*(buff+1)+=SaveBERVarBindListM(buff+2+*(buff+1),pdu->VariableBin);
return *(buff+1)+2;
```

Obr 5.19 Funkcia SaveBERSNMPDU.

FukciaSaveBERSNPMMesage (int SaveBERSNPMMesage(char* buff,snmppaket* packet)**)**

Funkcia vracia počet bajtov vyhradených pre položky SNMPversion, SNMP community string a SaveBERSNMPDU (vrátane dvoch bajtoch pre zápis SNMP message). Funkcia zapíše do pamäti identifikátor 0x30 a následne počet bajtov, ktoré sú vyhradené pre položky SNMPversion, SNMP community string a SaveBERSNMPDU. Táto funkcia využíva funkcie "SaveBERInteger" a "SaveBERSNMPDU", ktoré sú popísané vyššie. Princíp funkcie je totožný s funkciou SaveBERSNMPDU, v tomto prípade sa však pripočítavajú bloky SNMPversion, SNMP community string a SaveBERSNMPDU k aktuálnemu bloku. Kód tejto funkcie je značne objemný a z toho dôvodu je vysvetlený v prílohe[A1] s popiskami ku kódu.

Funkcia getBERInteger (int GetBerInteger(char * buff, int * i)**)**

Funkcia načíta z BER (pamäte BUFF) číslo a vráti veľkosť prečítanej pamäte a do premennej "i" uloží načítané číslo. Pomocou cyklu sa skopírujú do pomocného pola čísla z BER. Význam hlavných častí funkcie je znázornený na Obr. 5.20.

```
//naplníme si dočastne pole bytami z pola *buff a doplníme nulu.
memset(a,0,sizeof(char)*4);
        for(j=l-1; j>=0; j--)
         a[l-j-1] = *(buff+2+j);
//dekódujeme číslo z BER a uložíme do premennej i.
*i = (0xff000000 & 0x01000000*a[3]) + (0x00ff0000 & 0x010000*a[2])
 + (0x0000ff00 & 0x0100*a[1]) + (0x000000ff & 0x01*a[0]);
return *(buff+1)+2;//vrátime prečítaný počet bytov z BER
```

Obr 5.20 Funkcia getBERInteger.

Funkcia GetBERString (int GetBERString(char* buff, char **str)**)**

Táto funkcia načíta reťazec znakov (string) z BER pamäte BUFF. Výsledkom tejto funkcie je vrátenie počtu bajtov a následne do **str (ukazovateľ pola znakov) uloženie reťazcu znakov. Táto funkcia je znázornená na Obr. 5.21.

```
*str = (char*)(op_prg_mem_alloc(sizeof(char)*(*(buff+1)+1)));  //načítame z BER textový reťazec
memset(*str, 0, (*(buff+1)+1));
memcpy(*str, buff+2, *(buff+1));
return *(buff+1)+2; //vrátime prečítaný počet bajtov z BER.
```

Obr 5.21 Funkcia GetBERString.

Funkcia GetBEROID (int GetBEROID(char * buff, char **oid)**)**

Táto funkcia načíta OID z BER pamäte BUFF. Výsledkom tejto funkcie je vrátenie počtu bajtov a následne do **oid (ukazovateľ pola znakov) uloženie OID. V prípade prvých dvoch znakov rozkóduje prvé dve OID čísla podľa podmienky stanovujúcej BER pre OID čísla. Po rozkódovaní prvých dvoch znakov dekódujeme zostávajúce OID čísla v pamäti. Z dôvodu veľkosti kódu tejto funkcie, je možné nájsť tento kód s príslušnými popiskami v prílohe[A2].

Funkcia GetBerVarBind (int GetBerVarBind(char *buff, VARIABLEBIN* vb)**)**

Funkcia pre načítanie položky VarBind z pamäti, ktorá následne vracia počet bajtov potrebných pre Identifikátor objektu a jej hodnoty. Táto funkcia vracia v ukazovateli pola znakov Identifikátor objektu a jej hodnoty. Výpis funkcie GetBerVarBind je znázornený na Obr. 5.22.

```
tmpbuff+=GetBEROIDM(tmpbuff, &(vb->ObjectV));//načítame OID.
tmpbuff+=GetBERStringM(tmpbuff, &(vb->ValueV));//načítame ValueV.
return *(buff+1)+2; // vracia počet bytov.
```

Obr 5.22 Funkcia GetBerVarBind.

Funkcia GetBerVarBindList (int GetBerVarBindList(char *buff,VARIABLEBIN*vb)**)**

Funkcia má podobnú funkciu ako funkcia „GetBerVarBind“, pouze vracia počet bajtov potrebných pre VarBind. Tento kód je možné nájsť s popiskami v prílohe[A2].

Funkcia GetBerSNMPDU (int GetBerSNMPDU(char *buff, snmppdu* pdu)**)**

Obdobná činnosť funkcie ako funkcii „GetBerVarBindList“ “, pouze vracia počet bajtov potrebných pre RequestID, ErrorStatus, ErrorIndex a položky VarBindList. Využíva funkcie „GetBerInteger“ a „GetBerInteger“, ktoré sú popísané vyššie. Vrátenie počtu bytov pre RequestID, ErrorStatus, ErrorIndex a položky VarBindList je znázornený na Obr. 5.23.

```
tmpbuff+=GetBerIntegerM(tmpbuff, &(pdu->RequestId)); //načítame RequestID.
tmpbuff+=GetBerIntegerM(tmpbuff, &(pdu->ErrorStatus)); // načítame ErrorStatus.
tmpbuff+=GetBerIntegerM(tmpbuff, &(pdu->ErrorIndex)) ;// načítame ErrorIndex.
tmpbuff+=GetBerVarBindListM(tmpbuff, (pdu->VariableBin)); // načítame VariableBin.
```

Obr 5.23 Funkcia GetBerSNMPDU.

Funkcia GetBerSNMPMesage (int GetBerSNMPMesage(char *buff,snmppaket** packet)**)**

Funkcia vracia počet bajtov celej štruktúry správy SNMP. Vytvára sa alokácia pamäti pre SNMP správu a podradné štruktúry. Touto funkciou načítame data do správy z BER. Princíp činnosti je podobný funkcie GetBerSNMPDU, rozdiel je v tom, že táto funkcia vracia počet bytov pre položky Version, Community a SNMPpdu. Kód tejto funkcie je značne objemný a z toho dôvodu je vysvetlený v prílohe[A2] s popiskami ku kódu

Po implementácii tohto kódu do prostredia OPNET Modeler je možné vidieť správnu funkčnosť tohto algoritmu výpisom v simulačnej konzoli. Kliknutím na Configure/ Run Disrete Event Simulation zvolíme dobu simulácie 1 minuta a následne kliknutím Run Simulation sa spustí simulácia tejto sieti. Následná implementácia kódovania BER na stranách Managera Obr. 5.24 a Agenta Obr. 5.25 a následný výpis simulačnej konzoly Obr. 5.26 je vidieť nižšie.

Implementácia BER kódovania na odosielaný „tmp_packet“ na strane Managera v položke SEND je znázornená na Obr. 5.24.

```
len = SaveBERSNPMMessageM(tmp_packet, datapaket);
```

Obr 5.24 Implementácia BER na strane Managera.

Implementácia BER dekódovania na prijatý „tmp_packet“ na strane Agenta v položke RECEIVE SEND je znázornená na Obr. 5.25.

```
GetBerSNMPMessage(tmp_packet, &datapaket);
```

Obr 5.25 Implementácia BER na strane Agenta.

Po implementácií kódovania BER je po odsimulovaní tejto siete vidieť výstup simulačnej konzoly, ktorej obsah je možný vidieť nižšie na Obr. 5.26

Výpis na strane Managera:

Packet Community: Opnet

0. manager: Naplnený a vyslaný SNMP paket z Managera:

Packet Version: 0

Packet Community: V¼n{Ø„ï-ƒ

Packet PDUtype: 0

Packet RequestId: 1

Packet ErrorStatus: 0

Packet ErrorIndex: 0

Packet ObjectV: 1.3.6.1.4.1.26.80.1.2.7.3.2.0

Packet ValueV: aaaaaa

Správa zakódovaná pomocou BER:

30 35 02 01 00 04 0A 56 BC 6E 7B D8 84 EF AD 87 02 A0
24 02 01 01 02 01 00 02 01 00 30 19 30 17 06 0D 2B 06 01
04 01 1A 50 01 02 07 03 02 00 04 06 61 61 61 61 61 61

Výpis na strane Agenta:

Packet Version: 0

Packet Community-DES: V¼n{Ø„ï-ƒ

Packet Community: Opnet

Packet PDUtype: 0

Packet RequestId: 1

Packet ErrorStatus: 0

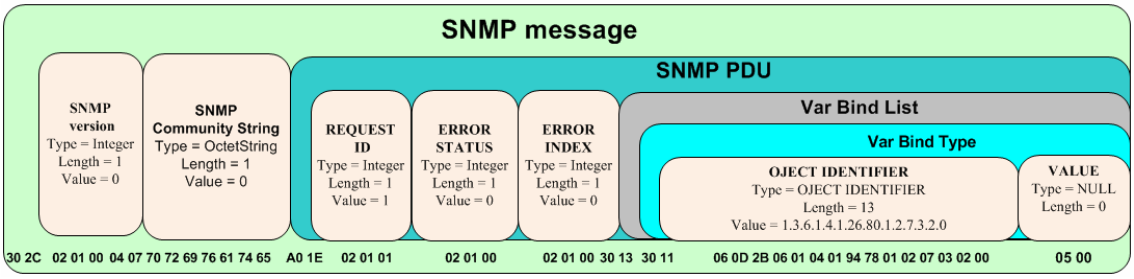
Packet ErrorIndex: 0

Packet ObjectV: 1.3.6.1.4.1.26.80.1.2.7.3.2.0

Packet ValueV: aaaaaa

Obr 5.26 Výpis simulačnej konzoly kódovania BER.

Z výpisu je možné vidieť, že sa kóduje text „Opnet“ , ktorý je vidieť u položky Packet Community. Zakódované heslo „Opnet“ pomocou DES algoritmu je znázornené u položky „Packet Community-DES : V¼n{Ø„ï-‡□“. Následne sa na celú štruktúru aplikuje BER kódovanie. Správa zakódovaná pomocou BER je znázornená v hexa tvare a je vidieť vo výpise Managera. Tento výpis nám znázorňuje správnu funkčnosť implementácie algoritmu BER do prostredia OPNET Modeler. Pre ľahšie pochopenie je znázornený Obr. 5.27, na ktorom vidieť priradenie štruktúry BER kódovania do správy SNMPv2.



Obr 5.27 BER kódovanie správy SNMPv2

5.4 Implementácia obojsmernej komunikácie Manager-Agent

Výsledkom študentskej práce v predchádzajúcich kapitolách bola vytvorená jednosmerná prevádzka z modulu Managera na modul Agenta. Pre dokončenie realizácie modelu komunikačnej siete je nutné doplniť komunikačnú sieť o spätnú väzbu zo strany Agenta na stranu Managera. V process modeli na strane Agenta je nutné opäť definovať štruktúru správy, definovať funkcie pre kódovanie BER a DES kódovania a následne tieto funkcie volať v modeli Agenta v položke „SEND“.

5.4.1 Formát datovej správy

Navrhnutá štruktúra je vložená do Header Bloku v Process Modelu OM na strane Agenta. Táto štruktúra je identická so štruktúrou na strane Managera. Definovaná štruktúra správy SNMP je zobrazená na Obr. 5.3. Význam jednotlivých položiek kódu je vysvetlený v kapitole 5.1.3.

Nyní máme definovanú štruktúru SNMP správy. Ak chceme s touto štruktúrou pracovať, je nutné pre túto štruktúru alokovať pamäť. Do bloku TV priradíme nasledujúci kód, ktorým jednoznačne hovoríme, že premenná datapaket bude odkazovať na nami vytvorenú štruktúru SNMPPAKET[3]. Následne je nutné do bloku TV priradiť premenné FValueV, ObjectV, ValueV, FObjectV, s ktorými pracuje štruktúra správy. Definícia doplneného kódu do bloku TV je znázornená na Obr. 5.28.

```
SNMPPAKET* datapaket;
char* FValueV;
char* ObjectV;
char* ValueV;
char* FObjectV;
```

Obr 5.28 Priradenie premenných do bloku Temporary Variable.

5.4.2 Realizácia zjednodušenej databáze MIB

V tejto kapitole bude popísaný postup vytvorenia zjednodušenej databáze MIB a následne managment tejto databáz. Táto databáza je zjednodušená z dôvodu zložitosti a veľkosti informácií reálnych MIB databáz. Pre managment nami vytvorených typov SNMP správ postačuje len zjednodušená databáza, ktorá neobsahuje vo svojej štruktúre všetky informácie, ktoré sa nachádzajú v reálnych MIB databáz. Pred samotným plnením nami vytvorenej štruktúry je nutné pre túto štruktúru alokovať pamäť. Alokácia pamäte sa v OM uskutoční pomocou funkcie „op_prg_mem_alloc“. Príslušný kód alokovania pamäte je zhodný s alokáciou pamäte na strane Managera, ktorý je vidieť v kapitole 5.1.3. Alokácia pamäte je odlišná len pri alokovaní pamäte pre položku „Community “ na strane Agenta. Doplnenie alokácie pamäte na strane Agenta je možné vidieť nižšie na Obr. 5.29.

```
datapaket->Community = (char *) op_prg_mem_alloc (strlen(&community_string_a));
```

Obr 5.29 Doplnenie alokácie pamäte na strane Agenta.

Po úspešnej alokácii štruktúry SNMPPAKET je nutné túto štruktúru naplniť. Pri dorazení správy na stranu Agenta je volaná MIB databáza, ktorá v podobe „.txt súbor" predstavuje zjednodušenú MIB databázu, v ktorej sa nachádzajú „OID" čísla a im priradené hodnoty pre daný sieťový prvok (AGENT). Z položky „ObjectV" vyčítanej zo správy dorazeného z MANAGERA sa vyčíta OID číslo a pomocou tohto čísla prebieha vyhľadávanie "ValueV" hodnoty v databáze MIB. Po vyhľadaní sa hodnota "ObjectV" uloží do štruktúry správy SNMPv2. Nasledujúci script je vidieť na Obr. 5.30.

```
F = fopen ("c:\\mib.txt", "r");
If ( f!= NULL)
{       while (fscanf (f, "%s %s", FObjectV, FValueV)==2)
        if{( strcmp( datapaket->SNMPpdu->VariableBin->ObjectV,FobjectV)== 0)
        sprintf(ValueV,"%s",FValueV);}
        fclose(f);
}
```

Obr 5.30 Získanie hodnoty z databáze MIB na strane Agenta.

Na strane Agenta v položke „Receive" je nutné alokovať pamäť pre prácu jak s premennou ObjectV tak i premennou ValueV, ktorá je vidieť na Obr. 5.31.

```
FValueV = op_prg_mem_alloc(sizeof(char)*100);
FObjectV = op_prg_mem_alloc(sizeof(char)*100);
```

Obr 5.31 Alokácia pamäte pre ObjectV na strane Agenta.

Na strane Agenta sú následne staticky uložené hodnoty pre PDUtype, RequestId, ErrorStatus a ErrorIndex. Zmenila sa pouze položka „PDUtype", ktorá informuje sieťovú komunikáciu o tom, že sa nejedná o typ požiadavku „Request = 0" ale o typ „Response = 1". Hodnoty ErrorStatus a ErrorIndex identifikujú pozíciu a tým chyby prenášanej správy. Tieto hodnoty sú nastavené na „0" pretože podľa pravidiel SNMP identifikujú bezchybnosť prenosu. Hodnota „RequestId" je nastavená na hodnotu „1", ktorá identifikuje danú správu. Tento výstup je možný vidieť na Obr. 5.32.

```
datapaket->SNMPpdu->PDUtype=1;
datapaket->SNMPpdu->RequestId=1;
datapaket->SNMPpdu->ErrorStatus=0;
datapaket->SNMPpdu->ErrorIndex=0;
```

Obr 5.32 Naplnenie položiek na strane Agenta statickými hodnotami.

5.4.3 Odoslanie správy z komunikačného modelu AGENT

Po alokovaní a následnom uložení hodnôt do štruktúry správy je nutné zakódovať položku „Community“ kódovaním DES a následne zakódovať celú správu kódovacími pravidlami BER. Boli použité rovnaké funkcie pre tieto kódovania ako pri kódovaní správy na strane MANAGERA, s tým rozdielom, že bolo nutné premenovať názvy funkcií na strane MANAGERA. OPNET Modeler nepodporuje použitie dvoch funkcií s rovnakým názvom v dvoch oddelených moduloch, z tohto dôvodu bolo nutné premenovať funkcie na strane MANAGERA. Za názov funkcie bolo pridané písmeno „m“ ako „Manager“. Prehľad a význam týchto funkcií je vidieť v kapitole 5.2 „DES“ a 5.3 „BER“. Po následnom zakódovaní bola naplnená správa poslaná zo strany Agenta na stranu Managera pomocou nasledujúcich príkazov v OpneteModeleri, ktoré je možné vidieť na Obr. 5.33. Význam príkazov je možné nájsť v práci [3].

```
op_pk_fd_set(pkptr,0,OPC_FIELD_TYPE_STRUCT,tmp_packet,1024,
op_prg_mem_copy_create, op_prg_mem_free, sizeof (SNMPPAKET));
op_ici_attr_set (ici_ptr, "local_port", loc_port);
op_ici_attr_set (ici_ptr, "rem_addr", remote_address);
op_ici_attr_set (ici_ptr, "rem_port", remote_port);
op_ici_install (ici_ptr);
op_pk_send (pkptr, UDPSTRM);
```

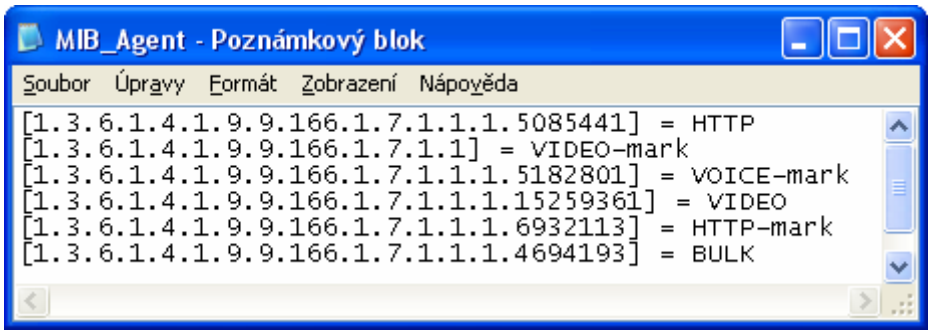
Obr 5.33 Odosielanie správy zo strany Agenta

5.4.4 Prijatie správy z komunikačného modelu Manager

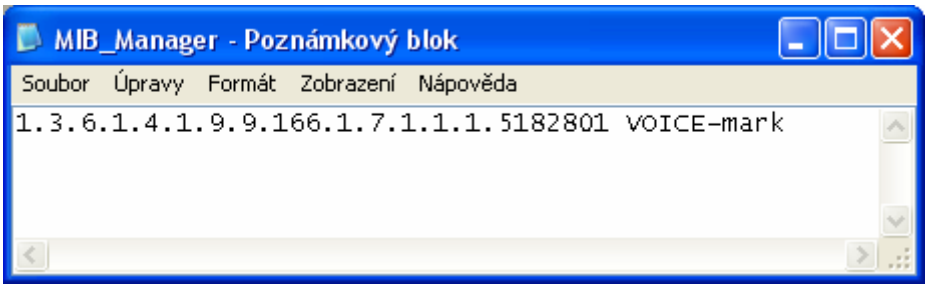
Po prijatí správy zo strany Agenta dochádza k rozkódovaniu správy pomocou funkcie „GetBerSNMPMessageM“, ktorej význam je vysvetlený v kapitole 5.3 a následne je rozkódovaný DES algoritmus pomocou funkcií vysvetlených v kapitole 5.2. Po rozkódovaní správy dochádza k vyčítaniu položiek ObjectV a ValueV zo štruktúry správy SNMPv2 do zjednodušenej databáze MIB na strane Managera. Následné uloženie „ObjectV“ a „ValueV“ do databáze MIB realizuje script na Obr. 5.34. Správnosť vyčítania z databáze overíme na Obr. 5.35, 5.36, keď na strane Managera zadávame „OID = 1.3.6.1.4.1.9.9.166.1.7.1.1.1.5182801“.

```
f = fopen("C:\\mib2.txt", "a");
If ( f!= NULL )
        {       fprintf (f, "%s %s \n", datapaket ->SNMPpdu ->VariableBin -> ObjectV,
                datapaket -> SNMPpdu -> VariableBin -> ValueV);
                fclose(f);
        }
```

Obr 5.34 Zapísanie hodnoty do databáze MIB na Managerovi



Obr 5.35 Databáza MIB na strane Agenta



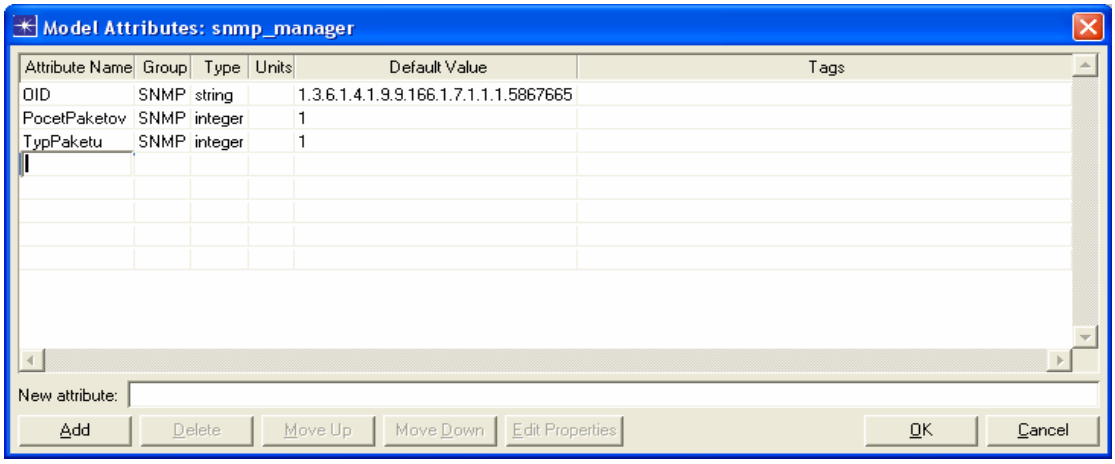
Obr 5.36 Databáza MIB na strane Managera.

5.5 Implementácia typu správy Get-NEXT do komunikačného modelu OM

Po vytvorení komunikačného modelu, pri ktorom sa posielajú správy typu „GetRequest“ zo strany Managera a následnom odosielaní správ typu „Response“ zo strany Agenta bolo nutné implementáciu rozšíriť o ďalší typ správy a to „GetNext“. Správa typu „GetNext“ pri odosielaní zo strany Managera žiada Agenta o doručenie nasledujúcich „OID“ vyčítaných Agentom zo svojej databáze MIB. V práci bolo požadované, aby počet týchto nasledujúcich správ bol ľubovoľný a preto bola vytvorená položka „PočetPaketov“ v atribútoch na strane Managera. Na strane Managera bola vytvorená v atribútoch taktiež položka „OID“ pre zadávanie hodnoty „OID“ a následne položka typu „TypPaketu“ pre interaktívnejšie zadávanie hodnôt do prostredia OPNET Modeler.

5.5.1 Implementácia na strane Managera.

V model Attributes bolo nutné definovať nasledujúce parametre, pre vytvorenie týchto atribútov. Všetky tri položky patria do skupiny SNMP, len datové typy sa menia. Položka „OID“ je typu „String“ a zostávajúce položky sú nastavené na datový typ „Int“. Nastavenie týchto atribútov je možné vidieť na Obr. 5.37.



Obr 5.37 Nastavenie atribútov OID, PočetPaketov a TypPaketu v Model Attributes.

Do položky State Variables bolo nutné zadefinovať premenné a datové typy správ. Definíciu je možné vidieť na Obr. 5.38.

```
char    \OID_m [100];
int \PocetPaketov;
int \TypPaketu;
```

Obr 5.38 Definícia položiek OID_m, PočetPaketov a TypPaketu v State Variables.

Posledným krokom pre vytvorenie atribútov v prostredí OPNET Modeler bolo nutné tieto premenné pridať do položky „Init“ v bloku „Send“. Pridanie premenných z grafického rozhrania do premenných s ktorými sa bude pracovať v prostredí „Manger“ je vidieť na Obr. 5.39.

```
op_ima_obj_attr_get (my_obj_id, "OID",                    &OID_m);
op_ima_obj_attr_get (my_obj_id, "TypPaketu",              &TypPaketu);
op_ima_obj_attr_get (my_obj_id, "PocetPaketov",           &PocetPaketov);
```

Obr 5.39 Priradenie premenných z grafického rozhrania do bloku Managera.

Po vytvorení atribútov bolo nutné hodnoty premenných priradiť do vytvorenej štruktúry SNMPv2, v záložke „SendManager“ . Do štruktúry správy sa pridalo pouze „OID“ a „TypPaketu“, pričom pre vytvorenie položky „OID“ sa musela dynamicky alokovať pamäť. Priradenie hodnôt do štruktúry je možné vidieť na Obr. 5.40.

```
datapaket->SNMP->VariableBin->ObjectV = op_prg_mem_alloc(sizeof(char)*(sizeof(OID_m)+1));
memcpy(datapaket->SNMPpdu->VariableBin->ObjectV, OID_m, sizeof(char)*(sizeof(OID_m)+1));
datapaket->SNMPpdu->PDUtype=TypPaketu;
```

Obr 5.40 Priradenie hodnôt OID a TypPaketu do štruktúry správy SNMPv2.

Na strane Managera v položke „Receive“ sa muselo pridať testovanie, či Manager má odosielať ďalšie „GetNext“ dotazy na stranu Agenta. To sa dosiahlo dekrementáciou premennej „PočetPaketov“ a následnom zisťovaní, či je počet správ väčší ako nula. V prípade, ak bola splnená podmienka, bol volaný príkaz „MNGR_SEND“ pre opätovné odosielanie správy SNMP od Managera k Agentovi. Ak podmienka splnená nebola, opätovné odosielanie sa neuskutočnilo. V prípade, že podmienka bola splnená, bolo nutné prepisovať premennú „OID_m“.

Je to nutné z toho dôvodu, aby sa opätovne odosielali požiadavky pre ďalšie správy, vždy s nasledujúcim „OID“ . Počet opakovaní bol vytvorený kódom, ktorý je vidieť na Obr. 5.41.

```
// Prepisovanie premennej pre získanie dalšieho GetNext v cykle
sprintf(OID_m, "%s", datapaket->SNMPpdu->VariableBin->ObjectV);

// Zistovanie počtu odosielaných paketov na stranu Agenta
PocetPaketov--;
if (PocetPaketov >0)
        op_intrpt_schedule_self (op_sim_time ()+10, MNGR_SEND);
```

Obr 5.41 Opätovné odosielanie správy GetNext na stranu Agenta.

Na strane Managera bolo poslednou časťou vytvoriť zapisovanie týchto hodnôt „OID“ a ich „VALUE“ do nami vytvorenej MIB na strane Managera. Pre zápis týchto hodnôt bolo nutné doplniť Managera, konkrétne časť do ktorej prichádza správa „Receive“ daným kódom, ktorý je vidieť na Obr. 5.42, ktorý zapisuje tieto hodnoty do MIB databáze.

```
// Zapisovanie aktuálneho OID a hodnoty do MIB databáze na strane Managera
f= fopen("c:\\mib2.txt", "a");
if (f!=NULL)
        {       fprintf(f, "%s %s\n", datapaket->SNMPpdu->VariableBin->ObjectV,
                datapaket->SNMPpdu->VariableBin->ValueV);
                fclose(f);
        }
```

Obr 5.42 Zápis OID a príslušnej hodnoty do databáze MIB na strane Managera.

5.5.2 Implementácia na strane Agenta

Pre implementáciu typu správy „GetNext“ bolo nutné na strane Agenta vytvoriť pomocnú štruktúru „TMibFile“. Pomocou tejto štruktúry je zoradená databáza MIB od najmenších „OID“ čísiel po najväčšie. Pre toto zoradenie bolo nutné vytvoriť funkciu „PorovnajOID“ ktorá porovnáva dvojicu „OID“ a zoraďuje ich podľa poradia. Následne bola načítaná celá štruktúra databáze MIB do štruktúry „TMibFile“ s ktorou sa následne pracovalo. Pomocou cyklu a podmienok sa zoraďovali „OID“ do danej štruktúry, z ktorej boli následne vyčítané hodnoty „Value“ podľa hľadaných „OID“ čísiel.

Ako bolo zmienené vyššie, potrebná časť riešenia nájdenia správy „GetNext“ je zoradenie „OID“ čísiel od najmenších čísiel po najväčšie čísla. Ako prvé je nutné túto samotnú štruktúru vytvoriť. Ako názov štruktúry bol zvolený „TMibFile“. Do tejto štruktúry sa budú ukladať pouze identifikátory „OID“ a im priradené hodnoty. Preto sa v tejto štruktúre vytvorili dve premenné a to „OID“ a k nemu príslušné hodnoty „Value“. Nasledujúci skript na Obr. 5.43 ukazuje vytvorenie tejto štruktúry. Tento kód bol vložený na strane Agenta do položky „Header Block“

```
typedef struct SMIBFile{
        char* OID;
        char* Value;
} TMibFile;
```

Obr 5.43 Vytvorenie štruktúry TMibFile.

Po vytvorení tejto štruktúry bolo nutné túto štruktúru naplniť datami. Napĺňanie štruktúry datami z databáze MIB sa uskutočňuje v položke „Init“, ktorá sa volá vždy raz. Na začiatku napĺňania sa pomocou cyklu „for“ vymazala daná štruktúra. Tento proces vymazania premenných je znázornený na Obr. 5.44.

```
// vymazanie premenných OID a Value v štruktúre TMibFile
for( i = 0; i< 10000; i++)
        {
                MibFile[i].OID = NULL;
                MibFile[i].Value = NULL;

        }
```

Obr 5.44 Vymazanie premenných v TMibFile.

Následne sme alokovali pamäť pre premenné „ObjectV“ a „FValue“. Po alokovaní pamäte sme vložili do štruktúry „TMibFile“ premenné „OID“ a „Value“ z databáze MIB postupne, pomocou cyklu „while“. Na Obr. 5.45 je vidieť v zjednodušenej podobe alokáciu pamäte a následné priradenie hodnôt „FOjectV-OID“ a „FValueV-Value“ v jednom cyklu. Do štruktúry sa ukladali OID hodnoty bez zátvoriek, ktoré je možné vidieť v naplnenej databáze MIB, preto sa pri alokovaní pamäte odpočítali a pripočítavali čísla, ako je možné vidieť na Obr. 5.45

```
MibFile[i].OID = op_prg_mem_alloc(sizeof(char)*(strlen(FObjectV)-1));
MibFile[i].Value = op_prg_mem_alloc(sizeof(char)*(strlen(FValueV)+1));
memcpy(MibFile[i].OID, FObjectV+1, strlen(FObjectV)-2);
MibFile[i].OID[strlen(FObjectV)-2] = 0;
memcpy(MibFile[i].Value, FValueV, strlen(FValueV)+1);
```

Obr 5.45 Alokácia a naplnenie štruktúry datami z databáze MIB.

Pri naplnení štruktúry datami z databáze MIB bolo následne nutné vytvoriť funkciu, ktorá bude zoraďovať nasledujúce „OID“ čísla v štruktúre „TMibFile“ vzostupne. Funkcia „PorovnajOID“ zoraďuje vždy dvojicu čísiel v dvoch susedných „OID“ číslach a podľa toho, ktoré číslo je väčšie, podľa toho dané číslo v OID zoradí. Táto funkcia je volaná v Agentovi v položke „Init“. Pomocou cyklu sa zoradia všetky „OID“ čísla a k nim priradené položky od najmenšieho čísla k „OID“ číslam s najväčšou hodnotou. Pomocou tohto cyklu a funkcie sa nám uložila databáza z hierarchicky usporiadanej do zoradenej, pomocou ktorej môžeme jednoducho vyhľadávať nasledujúce „OID“ čísla, ak príde požiadavka správy typu „GetNext“. Ukážka tohto zoradenia je možná vidieť na Obr. 5.46.

```
if(PorovnajOID(MibFile[i].OID, MibFile[i+1].OID)>0)
        {
                zmena = 1;
                 tmp_oid = MibFile[i+1].OID;
                 tmp_value = MibFile[i+1].Value;
                 MibFile[i+1].OID = MibFile[i].OID;
                 MibFile[i+1].Value = MibFile[i].Value;
                 MibFile[i].OID = tmp_oid;
                 MibFile[i].Value = tmp_value;
         }
```

Obr 5.46 Zoradenie OID v štruktúre

Po následnom naplnení a zoradení nami vytvorenej štruktúry, ktorých kód je možný vidieť na Obr. 5.45 a Obr. 5.46 zostáva na strane Agenta, v položke „Receive“ pouze vybrať dané „OID“ číslo a pomocou práce s poliami vybrať následné „OID“ číslo zo štruktúry a priradiť hodnotu tohto čísla do štruktúry správy SNMPv2, s ktorou bude pracovať Agent pri odosielaní správy na stranu Managera. Na strane Agenta sa testuje položka správy „PDUType“, ktorej kód je vidieť na Obr. 5.47 a Obr. 5.48.

V prípade, že Manager odosiela typ správy „PDUType = 0“, jedná sa o typ správy „GetRequest“. V tomto prípade sa hodnota správy u položky „ObjectV“ porovnáva s „OID“ hodnotou vytvorenej štruktúry „TMibFile“. Následne, ak sa daná položka nájde, priradí sa hodnota tohto „OID“ do položky „Value“, ktorú priradíme následne do datovej štruktúry SNMPv2 správy pri odosielaní zo strany Agenta. Nižšie na Obr. 5.47 je znázornená práca, v prípade, ak sa jedná o príkaz „GetRequest“.

```
if (strcmp(ObjectV, MibFile[i].OID)==0)
        {
                sprintf(ValueV, "%s", MibFile[i].Value);
                break;
        }
```

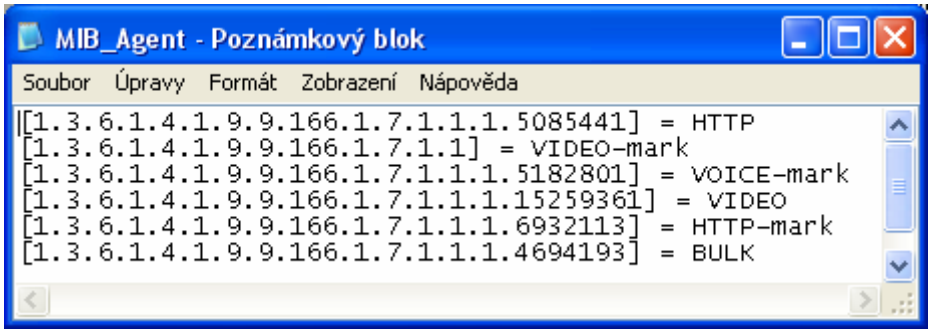
Obr 5.47 Práca so štruktúrou v prípade GetRequest.

V prípade, že Manager odosiela typ správy „PDUType = 2“, jedná sa o typ správy „GetNext“. V tomto prípade sa uloží do „ObjectV“ a „ValueV“ hodnota zo štruktúry v poli nasledujúcom, ako je aktuálne. Tento kód je možný vidieť na Obr. 5.48.

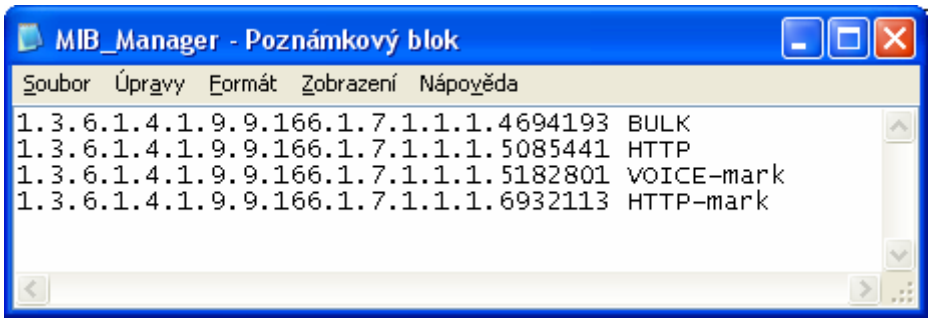
```
if (strcmp(ObjectV, MibFile[i].OID)==0)
        {
                sprintf(ObjectV, "%s", MibFile[i+1].OID);
                sprintf(ValueV, "%s", MibFile[i+1].Value);
                break;
        }
```

Obr 5.48 Práca so štruktúrou v prípade GetNextRequest.

Pre overenie správnosti je možné vidieť z Obr. 5.49 a Obr. 5.50 správne vytvorený komunikačný model v tejto práci. Na Obr. 5.49 je zobrazená zjednodušená MIB databáza, ktorá predstavuje databázu na reálnom školskom smerovači. V prípade, že chceme získať hodnotu z OID čísla „OID = 1.3.6.1.4.1.9.9.166.1.7.1.1“ s typom správy „PDUType = 2“ a následným počtom opakovaní „PočetPaketov = 4“, tak by sme sa mali presvedčiť o správne vytvorenom komunikačnom modeli. Na Obr. 5.50 môžeme vidieť výstup MIB databáze na strane Managera. Z výstupu je vidieť, že komunikačný model s navrhnutým „GetNext“ je implementovaný správne.



Obr 5.49 MIB databáza na školskom smerovači -Agent.



Obr 5.50 MIB databáza na Managerovi.

Týmto bola vytvorená komunikácia protokolu SNMP pri ktorej môžeme prenášať i viac informácií zo strany Agenta na stranu Managera. Nami vytvorený komunikačný model už umožňuje prenášať správy typu GetReguest, GetResponse a GetNextRequest.

5.6 Implementácia štruktúry MIB databáze na strane Managera

Pre zaistenie rovnakej datovej štruktúry MIB na strane Managera a Agenta, bolo nutné na strane Managera implementovať štruktúru databáze MIB. Do tejto štruktúry sa postupne napĺňajú správy prijaté zo strany Agenta, následne po skončení simulácie sa tieto data zotriedia od najmenších po najväčšie OID, ako to bolo implementované na strane Agenta. Touto implementáciou zaistíme rovnaké zoradenie MIB databáz v komunikačnom modeli Manager – Agent.

V tejto implementácie bolo nutné vytvoriť štruktúru, do ktorej sa budú prijaté data ukladať po príchode zo strany Agenta. Štruktúra je navrhnutá pre ukladanie dvoch hodnôt a to pre identifikátor hodnoty „OID“ a jemu priradenú hodnotu „Value“. Navrhnutá štruktúra je vložená do Header bloku na strane Managera. Štruktúra TMibFile vytvorená na strane Managera je vidieť na Obr. 5.51.

```
typedef struct SMIBFile{
        char* OID;
        char* Value;
} TMibFile;
```

Obr 5.51 Štruktúra TMibFile databáze MIB.

Po vytvorení štruktúry bolo nutné alokovať pamäť a následne vynulovať obsah tejto štruktúry. Kód znázornený na Obr. 5.52 bol vložený do inicializačného bloku na strane Managera.

```
MibFile = (TMibFile*)op_prg_mem_alloc(sizeof(TMibFile)*10000);
for( i = 0; i< 10000; i++)
        {
                MibFile[i].OID = NULL;
                MibFile[i].Value = NULL;
        }
```

Obr 5.52 Alokovanie pamäte pre štruktúru TMibFile.

Po alokácií štruktúry TMibFile bolo následne nutné alokovať pamäť pre hodnoty Value a OID, ktoré sa budú ukladať do nami vytvorenej štruktúry. Po alokácií pamäte pre tieto hodnoty bolo možné ukladať tieto hodnoty do štruktúry TMibFile. Táto alokácia a následné ukladanie hodnôt SNMP správ „OID“ a „Value“ je implementovaná na strane Managera v bloku „Receive“. Nasledujúci kód je možný vidieť na Obr. 5.53.

```
// Alokovanie pamäte pre OID.
MibFile[MibFileLen].OID=op_prg_mem_alloc(sizeof(char)*(strlen(datapaket->SNMPpdu-> VariableBin
->ObjectV)+1));
// Alokovanie pamäte pre Value.
MibFile[MibFileLen].Value=op_prg_mem_alloc(sizeof(char)*(strlen(datapaket->SNMPpdu->VaribleBin
- >ValueV) +1));
// Naplnenie štruktúry hodnotamy OID prijatými zo strany Agenta.
Memcpy (MibFile[MibFileLen].OID, datapaket->SNMPpdu->VariableBin->ObjectV, strlen(datapaket-
>SNMPpdu->VariableBin->ObjectV)+1);
// Naplnenie štruktúry hodnotamy Value prijatými zo strany Agenta.
Memcpy (MibFile[MibFileLen].Value, datapaket->SNMPpdu->VariableBin->ValueV, strlen(datapaket-
>SNMPpdu->VariableBin->ValueV) +1);
```

Obr 5.53 Alokácia pamäte a následné uloženie hodnôt do TMibFile.

Po naplnení štruktúry datami bolo nutné testovať koniec simulácie, po ktorej sa má vykonať funkcia na zoradenie OID od najmenšieho po najväčšie v nezoradenej štruktúre TMibFile. Zistenie konca simulácie sa implementovalo v OPNET Modeleri pomocou funkcie „op_intrpt_schedule_call“. Jedná sa o funkciu, ktorá spustí definovanú funkciu v presne definovaný čas. Pre zistenie času simulácie slúži konštanta v OPNET Modeleri „OPC_INTRPT_SCHED_CALL_ENDSIM“, do ktorej sa ukladá čas konca simulácie. Druhým vstupom do funkcie „op_intrpt_schedule_call“ je užívateľský definovaný číselný kód volanej funkcie. Tretím vstupom je funkcia, ktorá sa má vykonať v definovaný čas. Po skončení simulácie sa má spustiť funkcia „KonecTest“, ktorá vykonáva zoradenie OID v nezoradenej štruktúre TMibFile. Posledným vstupom do tejto funkcie je ukazovateľ na štruktúru, s ktorou funkcia pracuje. V našom prípade pracujeme so štruktúrou TMibFile. Funkciu so vstupnými parametrami je možné vidieť na Obr. 5.54.

```
op_intrpt_schedule_call (OPC_INTRPT_SCHED_CALL_ENDSIM, 0, KonecTest, &MibFile);
```

Obr 5.54 Volanie funkcie na konci simulácie.

Pre zoradenie tejto štruktúry sa používajú rovnaké algoritmy ako v kapitole 5.5.2, viď Obr. 5.46, pre zoradenie štruktúry na strane Agenta, z toho dôvodu nieje nutné ich v tejto kapitole popisovať. Po následnom zotriedení štruktúry TMibFile bolo poslednou úlohou uložiť obsah zoradenej štruktúry do textového súboru. Pre implementáciu vypísania tejto štruktúry slúži nasledujúci kód, ktorý je možné vidieť na Obr. 5.55.

```
//uloženie zotriedenej MIB databáze do súboru mib2.txt
f = fopen("c:\\mib2.txt", "w");

        for (i=0; i<9999; i++)
                {
                        if ((mf[i].OID == NULL))
                        break;
                        fprintf(f, "[%s] = %s\n", mf[i].OID, mf[i].Value);
                }
fclose(f);
```

Obr 5.55 Výpis zoradenej štruktúry do textového súboru mib2.txt

Výsledkom je zoradenie štruktúry jak na strane Managera, tak na strane Agenta. Na strane Managera a Agenta sa teraz nachádzajú rovnaké štruktúry zjednodušenej databáze MIB.

6 Záver

Pri tvorbe tejto diplomovej práce boli čerpané informácie predovšetkým z literatúry a internetových zdrojov, venovaných protokolom pre správu sietí, predovšetkým protokolu SNMP, šifrovania DES a kódovania BER.

Protokol SNMP je v súčasnej dobe najpoužívanejším protokolom pre správu sietí. Je súčasťou sady internetových protokolov. Umožňuje priebežný zber najrôznejších dát pre potreby správy sieti a ich následné vyhodnocovanie. SNMP protokol pozostáva z troch verzií, z ktorých najviac požívaná je verzia SNMPv2, ktorú zariadenia podporujú najviac. Tento protokol používa pre svoju činnosť databázu MIB, uloženú v pamäti sledovaného zariadenia (napr. smerovača). V databáze MIB sú uložené dôležité vlastnosti (informácie) zariadenia (napr. smerovača), z ktorej správca siete pomocou protokolu SNMP "vyčítava, nastavuje" dôležité informácie a tým zaisťuje správu siete.

Hlavným cieľom tejto diplomovej práce bolo navrhnúť štruktúru správy SNMPv2 v prostredí OPNET Modeler. Navrhnutú štruktúru bolo nutné doplniť o šifrovací algoritmus DES a kódovanie BER. Šifrovací algoritmus DES slúži pre zašifrovanie prenášaného hesla v štruktúre správy SNMPv2. Pri implementácii algoritmu DES sa v tejto práci vychádzalo z knižnice OpenSll, pomocou ktorej je zašifrované heslo v štruktúre správy SNMPv2. Následne sa aplikovala implementácia kódovania BER na štruktúru správy SNMPv2. Po implementácií týchto druhov kódovania bol vytvorený komunikačný model Manager – Agent, medzi ktorými boli prenášané naplnené a zakódované správy SNMPv2 (GetRequest, GetResponse). Po vytvorení typov správ SNMPv2 (GetRequest, GetResponse) sa vytvorila následne správa typu GetNextRequest, ktorá slúži na získanie väčšieho množstva správ z databáze MIB na strane Agenta. Na konci práce je implementovaná rovnaká štruktúra databáz MIB, jak na strane Managera, tak na strane Agenta. Týmto sú zoradené OID v týchto databáz od najmenších po najväčšie OID. Všetky algoritmy, ktoré sa v tejto práci vytvárali boli vytvárané v programovacom jazyku „C" a následne implementované do prostredia OPNET Modeler. Táto diplomová práca je súčasťou väčšieho projektu, ktorý rieši problematiku prenášania SNMP správ v prostredí OPNET Modeler.

7 Literatúra

[1] MACURA, M., *Protokoly pre správu sietí a zabezpečenie QoS.* Brno: Vysoké učení technické v Brne, Fakulta elektrotechniky a komunikačných technológií, 2007. Vedoucí bakalárske práce Ing. Jiří Hošek.

[2] WIKIPEDIA Simple Network Management Protocol. The Free Encyclopedia [on-line], apríl 2007.Dostupné na Internete: <http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol>

[3] ZEMAN, O., Implementace zjednodušené databáze DIFFSERV MIB. Brno: Vysoké učení technické v Brne, Fakulta elektrotechniky a komunikačných technológií, 2008. Vedoucí Diplomové práce Ing. Karol Molnár, PhD.

[4] OPNET Technologies, Inc.,OPNET Modeler Release 12 Product documentation 2007.

[5] OPEN SLL CONTRIBUTORS, *International Control Message Cryptographi and SSL/TLS.* [on-line], máj 2007. Dostupné na Internete: < http://www.openssl.org/docs/crypto/des.html>

[6] ASN.1 CONTRIBUTORS, *BER kódovaní.* [on-line], jún 2005. Dostupné na Internete: < http://man.och.cz/tcpipadns/Cd-II/CD-asn/asn02.htm>

[7] SNMP ASN.1 CONTRIBUTORS, *Based Endoded Rules.* [on-line], jún 2006. Dostupné na Internete: < http://www.et.put.poznan.pl/snmp/asn1/aber4.>

[8] HEROUT, P. Učebnice jazyka C. 3. uprav. Vyd. České Budějovice: Kooo, 2001. 269 s. ISBN 80-85828-21-9.

ZOZNAM SYMBOLOV A SKRATIEK

ASN.1	Abstrakt Syntax Notation number One – formálny jazyk pre popis štruktúrovaných dát pre komunikačné protokoly distribuovaných systémov
BER	Basic Encoding Rules – základné kódovacie pravidlá
DES	Data Encryption Standard – symetrická šifra v kryptografii
Diffserv	Differentiated Service – rozlíšené služby
FB	Function Block – blok deklarovania vlastných funkcií
HB	Header Block – blok hlavičky kódu
ISO	International Organization for Standardization – Medzinárodná organizácia pre normalizáciu
MIB	Management Information Base – databáza údajov a parametrov siete
OID	Object identifier – jedinečné označenie objektu, parametra v databáze MIB
OM	OPNET Modeler – simulačný program
QoS	Quality of Service – zaistenie dostatku sieťových prostriedkov potrebných k uspokojeniu požiadavky danej aplikácie
RFC	Request for Comments – zbierka dokumentov opisujúca výskumy, inovácie a metódy aplikované v Internetových technológiách
SNMP	Simple Network Management Protocol – protokol pre dohľad a správu siete
SNMP v1	Simple Network Management Protocol – prvá verzia
SNMP v2	Simple Network Management Protocol – druhá verzia
SNMP v3	Simple Network Management Protocol – tretia verzia
TCP	Transmission Control Protocol – spojovaný, spoľahlivý transportný protokol
UDP	User Datagram Protocol – nespoľahlivý, nespojovaný protokol.
NMS	Network management system – software, ktorý beží na Managerovi.

PRÍLOHY

A ZDROJOVÉ TEXTY

A.1 BER kódovanie

```
//  Funkcia, ktorá zistí počet bajtov potrebných pre vyjadrenie čísla a uloženie do pola *buff
int SaveIntToBuffM(char * buff, int i)
{
// ak je číslo 1 bytové, vracia 1 byte
	if(i<=0xFF)
	{
		//uložíme 1. byte do pola *buff
		*(buff) = (char)(i&0xFF);
		return 1;
	}
// ak je číslo 2 bytové, vracia 2 byte
	else if(i<=0xFFFF)
	{
		//uložíme 1. byte do pola *buff
		*(buff) = (char)((i&0xFF00) / 0x100);
		// uložíme 2. byte do pola *buff na pozíciu o jedna väčšiu
		*(buff + 1) = (char)(i&0xFF);
		return 2;
	}
	else if(i<=0xFFFFFF)
// ak je číslo 3 bytové, vracia 3 byte
	{
		//uložíme 1. byte do pola *buff
		*(buff) = (char)((i&0xFF0000) / 0x10000);
		// uložíme 2. byte do pola *buff na pozíciu o jedna väčšiu
		*(buff + 1) = (char)((i&0xFF00) / 0x100);
		// uložíme 3. byte do pola *buff na pozíciu o dva väčšiu
		*(buff + 2) = (char)(i&0xFF);
		return 3;
	}
// ak je číslo 4 bytové, vracia 4 byte
	else if(i<=0xFFFFFFFF
	{
		//uložíme 1. byte do pola *buff
		*(buff) = (char)((i&0xFF000000) / 0x1000000);
		// uložíme 2. byte do pola *buff na pozíciu o jedna väčšiu
		*(buff + 1) = (char)((i&0xFF0000) / 0x10000);
		// uložíme 3. byte do pola *buff na pozíciu o dva väčšiu
		*(buff + 2) = (char)((i&0xFF00) / 0x100);
		// uložíme 4. byte do pola *buff na pozíciu o tri väčšiu
		*(buff + 3) = (char)(i&0xFF);
		//funkcia vracia 4 – predstavuje počet bytov
		return 4;
	}
	return 0;
}
```

```
// Funkcia, ktorá vstupné číslo ukladá do pamäti prostredníctvom 3 položiek-typ,dĺžka,hodnota
int SaveBERIntegerM(char* buff, int i)
{
	int size = 0;
	// do tretej pozície "hodnota" ukladáme číslo
	size = SaveIntToBuffM(buff+2, i);
	// uložíme označenie typu číslo
	*buff = 0x02;
	//uložíme počet bytov na ktorých je číslo uložené
	*(buff+1) = size;
	//vrátime veľkosť bloku BER
	return size+2;
}


// Funkcia, ktorá  vstupný reťazec znakov ukladá do pamäti prostredníctvom 3 položiek-typ,dĺžka,hodnota
int SaveBERStringM(char* buff, char* str)
{
	//ak je reťazec NULL
	if(str == NULL)
	{
		//nastavíme označenie typu NULL
		*buff = 0x05;
		//dĺžka je 0, hodnota sa neudáva
		*(buff + 1) = 0x00;
		//vrátime veľkosť bloku BER
		return 2;
	}
	else
	{
		//nastavíme označenie typu Text
		*buff = 0x04;
		//uložíme dĺžku textu
		*(buff + 1) = strlen(str);
		//vložíme text do pola *buff
		memcpy((buff + 2), str, strlen(str));
		//vrátime veľkosť bloku BER
		return strlen(str)+2;
	}
}


// Funkcia, ktorá vstupné OID ukladá do pamäti prostredníctvom 3 položiek-typ,dĺžka,hodnota
int SaveBEROIDM(char* buff, char* oid)
{
	int len = strlen(oid);
//vytvoríme si dočasnú premennú, kam skopírujeme OID a "." nahradíme za 0 - ukončenie textového reťazca
	char *tmp = (char*)op_prg_mem_alloc(sizeof(char)*len);
	int i,j,tmp1, tmp2;
	memcpy(tmp, oid, len);
	//prechod celým OID
	for(i=0; i<len; i++)
	{
		// Bodku nahradíme znakom 0 – ukončenie textového reťazca
		if(tmp[i]=='.')
		{
			tmp[i] = 0;
		}
	}
```

```
	//uložíme si do dočastnej premennej tmp1 číselnú hodnotu prvého údaja z OID
	j = 1;
	tmp1 = atoi(tmp);
	//prejdeme dočasnú premennú tmp(pole reťazcov vytvorené z OID)
	for(i=0; i<len; i++)
	{
		//ak sme na konci daného bloku z OID
		if(tmp[i]==0)
		{
			//ak ide o prvé dve čísla tak je iný výpočet
			if (j == 1)
			{
				j=2;
				tmp2 = atoi(tmp+i+1);
				//uložíme do *buff číselnú hodnotu 40*x+y(x-prvé číslo,y-druhé číslo)
				j += SaveBERIntegerM(buff+j, 40*tmp1+tmp2);
			}
			//ak ide o ostatné čísla ako je 1. a 2.
			else
			{
				tmp2 = atoi(tmp+i+1);
				//uložíme do *buff číselnú hodnotu daného pola z OID
				j += SaveBERIntegerM(buff+j, tmp2);		}
		}
	}
	//nastavíme označenie typu OID
	*buff = 0x06;
	//nastavíme dĺžku bloku BER
	*(buff+1) = j-2;
	return j;
}


// Funkcia vracia počet bajtov pre vyjadrenie identifikátora objektu a jej hodnoty
int SaveBERVarBindM(char *buff ,VARIABLEBIN* vb)
{
	//nastavíme typ VarBind
	*buff = 0x30;
	//nastavíme dĺžku 0, budeme zvyšovať
	*(buff+1)=0;
	//uložíme OID do *buff a pričítame dĺžku bloku OID k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBEROIDM(buff+2,vb->ObjectV);
	// uložíme ValueV, nastavíme dĺžku aktuálneho bloku
	*(buff+1)+=SaveBERStringM(buff+2+*(buff+1),vb->ValueV);
	return *(buff+1)+2;
}

// Funkcia vracia počet bajtov pre vyjadrenie podradenej štruktúry-VarBind
int SaveBERVarBindListM(char *buff ,VARIABLEBIN* vb)
{
	//nastavíme typ VarBind
	*buff = 0x30;
	//nastavíme dĺžku 0, budeme zvyšovať
	*(buff+1)=0;
	// uložíme VarBind, nastavíme dĺžku aktuálneho bloku
	*(buff+1)+=SaveBERVarBindM(buff+2,vb);
	return *(buff+1)+2;
}
```

```
// Funkcia vracia počet bajtov pre vyjadrenie podradenej štruktúry-VarBindList
int SaveBERSNMPDUM(char *buff, SNMPPDU* pdu)
{
	// nastavíme typ PDU
	if (pdu->PDUtype == 0)
		*buff = 0xA0;
	else if (pdu->PDUtype == 1)
		*buff = 0xA1;
	else if (pdu->PDUtype == 2)
		*buff = 0xA2;
	*(buff+1)=0;
	// uloženie RequestId a pričítame dĺžku bloku RequestId k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBERIntegerM(buff+2+*(buff+1),pdu->RequestId);
	// uloženie ErrorStatus a pričítame dĺžku bloku ErrorStatus k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBERIntegerM(buff+2+*(buff+1),pdu->ErrorStatus);
	// uloženie ErrorIndex a pričítame dĺžku bloku ErrorIndex k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBERIntegerM(buff+2+*(buff+1),pdu->ErrorIndex);
	// uloženie VarBind a pričítame dĺžku bloku VarBind k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBERVarBindListM(buff+2+*(buff+1),pdu->VariableBin);
	return *(buff+1)+2;
}

// Funkcia vracia počet bajtov pre vyjadrenie podradenej štruktúry - SNMPversion, CommunityString,
SNMP_PDU
int SaveBERSNPMMessageM(char *buff, SNMPPAKET* packet)
{
	// nastavíme typ 0x30- Sequence typ
	*buff = 0x30;
	// nastavíme dĺžku, budeme zvyšovať
	*(buff+1)=0;
	// uloženie Version a pričítame dĺžku bloku Version k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBERIntegerM(buff+2+*(buff+1),packet->Version);
	// uloženie Community a pričítame dĺžku bloku Community k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBERStringM(buff+2+*(buff+1),packet->Community);
	// uloženie SNMPpdu a pričítame dĺžku bloku SNMPpdu k dĺžke aktuálneho bloku
	*(buff+1)+=SaveBERSNMPDUM(buff+2+*(buff+1),packet->SNMPpdu);
	return *(buff+1)+2;
}
```

A.2 BER dekódovanie

```
// Funkcia, ktorá načíta z pamäte BUFF číslo, vráti počet bajtov a následne uloží číslo do premennej "i"
int GetBerIntegerM(char * buff, int * i)
{
	int l = *(buff+1);
	char a[4];
	int j;
	// ak sa nejedná o číslo, vráti 0
	if (*buff != 0x02)
	{
		return 0;
	}

	//naplníme si dočasne pole bytmi z pola *buff a doplníme 0
	memset(a,0,sizeof(char)*4);
	for(j = l-1; j >= 0; j--)
		a[l-j-1] = *(buff+2+j);
	//dekódujeme číslo z BER a uložíme do premennej i
	*i = (0xff000000 & 0x01000000*a[3]) + (0x00ff0000 & 0x010000*a[2]) + (0x0000ff00 &
	0x0100*a[1]) + (0x000000ff & 0x01*a[0]);
	//vrátime prečítaný počet bytov z BER
	return *(buff+1)+2;
}


// Funkcia, ktorá načíta z pamäte BUFF reťazec znakov, vráti počet bajtov a následne uloží reťazec
znakov do premennej "str"
int GetBERStringM(char* buff, char **str)
{
	// ak neni NULL a zároveň reťazec, vráť 0
	if ((*buff != 0x05) && (*buff != 0x04))
	{
		return 0;
	}
	//ak je text NULL vrátime NULL
	if (*buff == 0x05)
	{
		*str = NULL;
		return *(buff+1)+2;
	}

	//načítame z BER textový reťazec
	*str = (char*)(op_prg_mem_alloc(sizeof(char)*(*(buff+1)+1)));
	memset(*str, 0, (*(buff+1)+1));
	memcpy(*str, buff+2, *(buff+1));
	//vrátime prečítaný počet bajtov z BER
	return *(buff+1)+2;
}
```

```
// Funkcia, ktorá načíta z pamäte BUFF reťazec OID, vráti počet bajtov a následne uloží OID do premennej "oid"
int GetBEROIDM(char * buff, char **oid)
{
	char *tmp_oid;
	int i;
	int tmp;
	int prvé = 0;
	// ak sa nejedná o OID
	if (*buff != 0x06)
		return 0;

	//alokujeme pamäť pre textovú reprezentáciu OID
	*oid = (char*)op_prg_mem_alloc(sizeof(char)*100);
	memset(*oid, 0, sizeof(char)*100);
	tmp_oid = *oid;

	//prejdeme OID zakódované BER
	for(i = 0; i < *(buff+1);)
	{
		i += GetBerIntegerM(buff+2+i, &tmp);
		//ak ide o prvú hodnotu dekódujeme ju ako 2 hodnoty OID
		if(prve++==0)
		{
			sprintf(tmp_oid,"%d.%d.",(tmp)/40,(tmp) % 40);
		}
		//dekódujeme ostatné hodnoty OID
		else
		{
			tmp_oid = *oid + strlen(*oid);
			if (i == *(buff+1))
				sprintf(tmp_oid,"%d",tmp);
			else
				sprintf(tmp_oid,"%d.",tmp);

		}
	}


	return *(buff+1)+2;
}

//Funkcia, ktorá vracia počet bajtov potrebných pre identifikátor objektu a jej hodnoty
int GetBerVarBindM(char *buff, VARIABLEBIN* vb)
{
	char* tmpbuff;
	// ak sa nejedná o sekvenčný typ
	if (*buff != 0x30)
		return 0;

	tmpbuff = buff;
	tmpbuff+=2;
	//načítame OID a zistíme aktuálnu veľkosť bloku
	tmpbuff+=GetBEROIDM(tmpbuff, &(vb->ObjectV));
	//načítame ValueV a zistíme aktuálnu veľkosť bloku
	tmpbuff+=GetBERStringM(tmpbuff, &(vb->ValueV));
	return *(buff+1)+2;
}
```

```
// Funkcia, ktorá  vracia počet bajtov potrebných pre VarBind
int GetBerVarBindListM(char *buff, VARIABLEBIN* vb)
{
	char* tmpbuff; // ak sa nejedná o sekvenčný typ, vráť 0
		if (*buff != 0x30)
		return 0;
	tmpbuff = buff;
	tmpbuff+=2; //načítame VarBind a zistíme aktuálnu veľkosť bloku
	tmpbuff+=GetBerVarBindM(tmpbuff, vb);
	return *(buff+1)+2;
}

// Funkcia, ktorá vracia počet bajtov pre RequestID, ErrorStatus, ErrorIndex a položky VarBindList
int GetBerSNMPDUM(char *buff, SNMPPDU* pdu)
{
	// ak sa nejedná o jeden zo správ  Request, RequestNextRequest a Response, vráť 0
	char* tmpbuff;
	if (((unsigned char)(*buff) != (unsigned char)0xA0) && ((unsigned char)(*buff) != (unsigned
	char)0xA1) && ((unsigned char)(*buff) != (unsigned char)0xA2))
		return 0;

	tmpbuff = buff;
	// Ak sa jedná o typ A0, jedná sa o typ správy Request
	if ((unsigned char)(*buff) == (unsigned char)0xA0)
		pdu->PDUtype = 0;
	// Ak sa jedná o typ A1, jedná sa o typ správy RequestNextRequest
	if ((unsigned char)(*buff) == (unsigned char)0xA1)
		pdu->PDUtype = 1;
	// Ak sa jedná o typ A2, jedná sa o typ správy Response
	if ((unsigned char)(*buff) == (unsigned char)0xA2)
		pdu->PDUtype = 2;

	tmpbuff+=2;
	tmpbuff+=GetBerIntegerM(tmpbuff, &(pdu->RequestId));//načítame RequestID
	tmpbuff+=GetBerIntegerM(tmpbuff, &(pdu->ErrorStatus));// načítame ErrorStatus
	tmpbuff+=GetBerIntegerM(tmpbuff, &(pdu->ErrorIndex));// načítame ErrorIndex
	tmpbuff+=GetBerVarBindListM(tmpbuff, (pdu->VariableBin));// načítame VariableBin
	return *(buff+1)+2;
}

// Funkcia vracia počet bajtov celej štruktúry SNMP-dekódovanie celej štruktúry
int GetBerSNMPMessageM(char * buff, SNMPPAKET** packet)
{
	// ak sa nejedná o sekvenčný typ, vráť 0
	char* tmpbuff;
	if ((unsigned char)(*buff) != (unsigned char)0x30)
		return 0;
	//alokujeme pamäť pre štruktúru SNMPPAKET, SNMPpdu a VariableBin
	*packet = (SNMPPAKET *) op_prg_mem_alloc (sizeof (SNMPPAKET));
	(*packet)->SNMPpdu = (SNMPPDU *) op_prg_mem_alloc (sizeof (SNMPPDU));
	(*packet)->SNMPpdu->VariableBin = (VARIABLEBIN *) op_prg_mem_alloc (sizeof
	(VARIABLEBIN));

	tmpbuff = buff;
	tmpbuff+=2;
	tmpbuff+=GetBerIntegerM(tmpbuff, &((*packet)->Version));//načítame Verziu
	tmpbuff+=GetBERStringM(tmpbuff, &((*packet)->Community));//načítame Community
	tmpbuff+=GetBerSNMPDUM(tmpbuff, ((*packet)->SNMPpdu));//načítame SNMPpdu

	return *(buff+1)+2; }
```