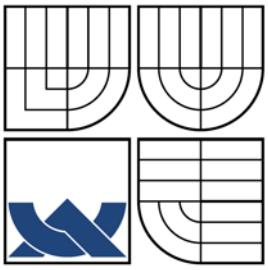
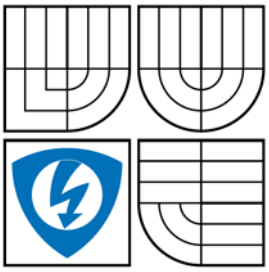


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

INFORMAČNÍ SYSTÉM PRO SPRÁVU BEZDRÁTOVÉ SÍTĚ S VYUŽITÍM ROUTERBOARDU MIKROTIK

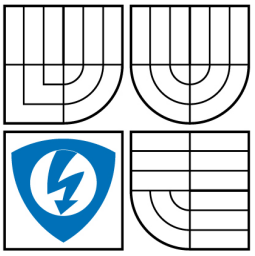
IINFORMATION SYSTEM FOR WIRELESS NETWORK MANAGEMENT BASED ON MIKROTIK ROUTERBOARD

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — Bc. PETR HROMÁDKO
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN MALÝ
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Petr Hromádko ***ID:*** 83725
Ročník: 2 ***Akademický rok:*** 2008/2009

NÁZEV TÉMATU:

Informační systém pro správu bezdrátové sítě s využitím routerboardu MikroTik

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte webový informační systém pro správu klientů bezdrátové sítě s využitím routerboardu MikroTik. Analyzujte možnosti propojení webového serveru s routerboardem a implementujte navržený informační systém na serveru Apache s využitím jazyka PHP a relační databáze MySQL. Systém otestujte z hlediska bezpečnosti a použitelnosti.

DOPORUČENÁ LITERATURA:

[1] Kabir, M. J.: Apache Server 2 Bible, Wiley 2002, ISBN: 9780764548215
[2] MikroTik: MikroTik RouterOs v2.9 Reference Manual, Mikrotikls SIA 2008
[3] Gutmans A., Bakken S.S., Rethans D.: Mistrovství v PHP 5, Computer Press 2007, ISBN: 9788025115190

Termín zadání: 9.2.2009 ***Termín odevzdání:*** 26.5.2009

Vedoucí práce: Ing. Jan Malý

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

UPOZORNĚNÍ:

Abstrakt

Diplomová práce se zabývá návrhem webového informačního systému pro správu klientů bezdrátové sítě s využitím routerboardu MikroTik. Jedná se o informační systém, prostřednictvím kterého bude mít poskytovatel internetového připojení trvalý přehled o všech klientech své sítě a zároveň bude moci prostřednictvím systému ovlivňovat parametry nastavení síťových prvků na platformě MikroTik. Kromě toho má možnost monitorovat stav signálu jednotlivých klientů či možnost sledovat množství přenesených dat klienty.

Práce je rozdělena na dvě části, první část se převážně věnuje popisu co je to vlastně informační systém, použitým platformám, zařízením, konfigurací serveru a služeb spuštěných na tomto serveru. Druhá část je již plně věnovaná analýze zabezpečení aplikace, případům užití, návrhu databázového schématu a funkčním formulářům. Největší důraz byl kladen v této části na možnosti propojení routerboardu s OS MikroTik s webovým serverem a vlastní otestování aplikace v reálném provozu.

Klíčová slova

webový informační systém, routerboard, MikroTik, PHP, MySQL, Apache, databáze, E-R digram, formulář, bezpečnost

Abstract

The master`s thesis deals with a concept of a web information system, designed to manage clients of a wireless network based on Mikrotik routerboards. The system allows the internet provider to have a great survey over all clients of the wireless network and to manage the settings of Mikrotik based network hardware. Furthermore, it offers an ability of signal strength monitoring and data accounting.

The thesis has been split into two sections. The first one describes the information system as such, used platforms, equipment, running services and server configuration. The second section is fully focused on the analysis of the application security, database schema design and functional forms. The strongest emphasis has been put to the conjunction of the Mikrotik OS based routerboards with a web server and to the testing in a real environment.

Keywords

web information system, routerboard, MikroTik, PHP, MySQL, Apache, database, E-R digram, form, security

Bibliografická citace

HROMÁDKO, P. *Informační systém pro správu bezdrátové sítě s využitím routerboardu MikroTik*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 78 s. Vedoucí diplomové práce Ing. Jan Malý.

Prohlášení

Prohlašuji, že svůj semestrální projekt na téma Informační systém pro správu bezdrátové sítě s využitím routerboardu MikroTik jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto projektu 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 následujícího 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

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu práce Ing. Janu Malému za konstruktivní kritiku a připomínky k této práci. Poděkování si rovněž zaslouží technici, kteří testovali informační systém v reálném prostředí. Děkuji také svým rodičům a partnerce za podporu během studia a psaní této práce.

Obsah

Seznam zkratek

AP - Access point

CA - Certifikační autorita

CASE - Computer-aided software

CSS - Cascading Style Sheets

DES - Data Encryption Standard

DSA - Digital Signature Algorithm

E-R diagram - Entity Relationship Diagram

FTP - File Transfer Protocol

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

IE6 - Internet Explorer 6

IP - Internet Protocol

IPX - Internet Packet Exchange

IS - Informační systém

ISP - Internet service provider

LAN - Local Area Network

LOV - List of values

MIB - Management information base

OID - Object identifier

OOD - Objektově orientované databáze

OS - Operační systém

PC - Personal computer

PHP - Hypertext Preprocesor

PKI - Public Key Infrastructure

RB - RouterBoard

RRD - Round Robin Databáze

RSA - Iniciály autorů Rivest, Shamir, Adleman

RSH - remote shell

SFTP - SSH File Transfer Protocol

SNMP - Simple Network Management Protocol

SŘBD - Systém řízení báze dat

SSH - Secure Shell

SSL - Secure Sockets Layer

TCP - Transmission Control Protocol

TLS - Transport Layer Security

UDP - User Datagram Protocol

UML - Unified Modeling Language

W3C - World Wide Web Consortium

XHTML - Extensible hypertext markup language

XML - eXtensible Markup Language

Úvod

Informační systém pro správu bezdrátové sítě je komplexní informační systém, který obsahuje funkce pro administraci a monitoring koncových klientů nebo také částečnou administraci síťových prvků routerboardů založených na platformě OS MikroTik. Diplomová práce popisuje takovýto informační systém z pohledu vývojáře a zároveň z pohledu poskytovatele internetu.

Informačních systémů pro administraci a řízení aktivních síťových prvků je na trhu několik, např. ISPadmin nebo Cibs. Hlavním důvodem pro vlastní implementaci informačního systému ve spolupráci s routerboardem s OS Mikrotik bylo, že aplikace od jiných vývojářů jsou komerční, a také obsahují nespočet funkcí, které by nebyly využity v bezdrátové síti do 200 klientů.

Práce je koncipována do dvou velkých celků, teoretické a praktické části. Teoretická část je pak členěna do čtyř kapitol, ve kterých se zabývám tím, co to je informační systém, vývojovými prostředky, zařízením a konfigurací serveru a služeb spuštěných na tomto serveru. Nejrozsáhlejší část je věnována architektuře, vývojovému prostředí a protokolům. Bez těchto znalostí bychom nebyli schopni realizovat samotný informační systém. Praktická část se již plně soustředí na analýzu a následnou realizaci informačního systému spolupracujícího s routerboardem MikroTik. Cílem není detailně popsat všechny skripty, jednotlivé funkce systému či HTML kód, ale podrobně analyzovat možná bezpečnostní rizika, definovat případy užití a kvalitně zpracovat návrh databázového schématu. Pokud bychom jeden z těchto bodů nedostatečně zpracovali či vynechali, mohlo by to mít katastrofální následky pro celý systém. Neméně podstatnou částí je i návrh jednotlivých funkčních formulářů (obrazovek), proto i jim je věnovaná kapitola. V poslední části práce jsou podrobně rozebrány možnosti komunikace mezi realizovaným informačním systémem a routerboardem prostřednictvím SSH kanálu a SNMP.

Samotný závěr je určen pro shrnutí hodnocení realizovaného informačního systému uživateli, kteří měli možnost tento systém několik týdnů testovat.

1. Informační systém

Pojem informační systém má mnoho výkladů i definic. Informační systémy jsou systémy určené pro sběr, zpracování a poskytování informací a dat. Zjednodušeně by se dalo říci, že IS transformuje zadané vstupní informace jedince do podoby určené pro ostatní uživatele.

Dobře fungující informační systém by měl vést jednak k celkové úspoře práce, k jejímu zefektivnění a také ke zvýšení výkonnosti lidí v organizaci.

1.1 Historie

První informační systémy se začaly objevovat v sedmdesátých letech minulého století, kdy vznikly první databázové systémy. Databázi lze přirovnat k datovému skladišti, kde jsou data a informace uloženy a zpracovány. Samotný přístup k datům je ovládán přes tzv. „systém řízení báze dat“ (SŘBD) [12]. Aplikační programy nepřistupují v databázi přímo k datům, ale dávají požadavky k SŘBD, která jim vrací požadovaná data.

Postupem času bylo vytvořeno několik databázových modelů pro ukládání dat, např. síťový, hierarchický či relační. Poslední zmiňovaný relační model je využíván dodnes. Relační databáze je sestavena z tabulek, jejich sloupce jsou vázány na sloupce jiných tabulek, vazby lze jednoduše vytvářet, modifikovat atd.

Nástupcem relačních databází měly být objektově orientované databáze (OOD), které umějí ukládat samotné objekty bez nutnosti je uspořádávat do tabulek. Ačkoliv se v dnešní době využívají objektově orientované programovací jazyky, tak OOD nalezly na trhu jen malé uplatnění [12][11]. Další etapou vývoje bylo oddělení uživatelského rozhraní od samotných procesů, což odloučilo programování logiky od programování prezentace výsledků.

1.2 Požadavky na IS

Při návrhu informačního systému je nutná znalost daného prostředí. Požadavky na informační systém získá vývojář systému od zadavatele, který by měl poskytnout pro přesné zadání dostatek informací k vývoji IS. Dále si musíme uvědomit, že IS nebude používat pouze jeden člověk, ale že je určen pro větší okruh uživatelů, přičemž každý z nich na něj bude mít své vlastní požadavky. Všechny tyto požadavky je nutné v první fázi návrhu IS zaznamenat a později zanalyzovat.

Při samotné analýze systému již pracujeme s daty získanými ve specifikaci požadavků. Nejdříve musíme vyhodnotit, které požadavky jsou reálné a realizovatelné v daném programovacím prostředí. Dále musíme vytvořit seznam funkčních požadavků, seznam událostí a reakcí na ně. Po těchto krocích dojde k vytvoření logického a datového modelu, k modelaci entit a životních cyklů [5].

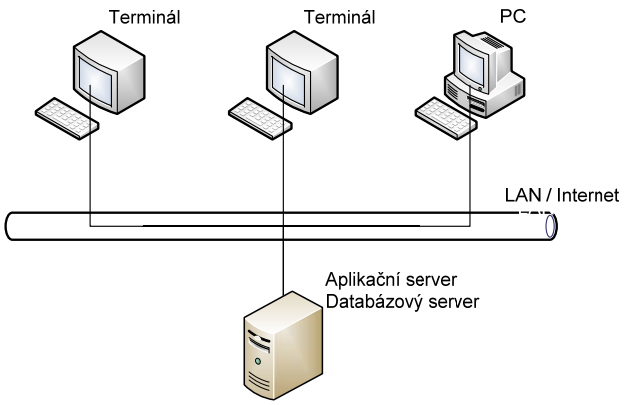
1.3 Grafický návrh systému

Pro samotný návrh systému lze využít software umožňující práci v jazyku UML (Unified Modelling Language), který je standardem při návrhu informačních systémů. Cílem grafického návrhu systému a samotného jazyka UML je ukázat pohled na systém v grafické podobě, což umožňuje reálnější představu o IS z hlediska jeho funkčnosti. Proto je tento jazyk vhodný jako nástroj pro komunikaci mezi žadatelem a vývojářem [20].

2. Architektura, programovací jazyky, protokoly

2.1 Třívrstvá architektura

Webové aplikace postavené na platformě PHP, ASP.NET, JAVA atd. běžně využívají tzv. třívrstvou architekturu. Tento standard vývoje aplikací vznikl z předešlé dvouvrstvé architektury odvozené od modelu klient/server. Klient/server neboli tenký klient funguje na principu méně výkonného počítače, který slouží jako terminál, a serveru (obr. 1). Terminál slouží jako nástroj pro prezentaci dat uživateli, kdy veškeré programové aplikace a data byla umístěna na serveru. Jakmile je terminál spuštěn, stáhne si ze serveru potřebná data pro start. Veškeré operace se dějí na straně serveru, klient na něj pouze směruje požadavky a v zápětí od něj dostává odpovědi. Takto podobně pracuje i dvouvrstvá architektura. Místo terminálu se používá plnohodnotný počítač, na kterém jsou nainstalovány potřebné aplikace, a na serveru jsou umístěna data, případně další aplikace. Jednou z hlavních nevýhod této architektury je malá rychlost aplikací a velké množství přenesených dat přes síť – to z toho důvodu, že jakmile chceme v souboru o velikosti několika megabytů vyhledat nějakou informaci, musíme celý tento soubor stáhnout na PC a teprve poté ho zpracovat. Obdobně probíhá komunikace i při nepatrné modifikaci dat, kdy se musí celý modifikovaný soubor přenášet tam a zpět.



Obr. 1: Architektura klient/server

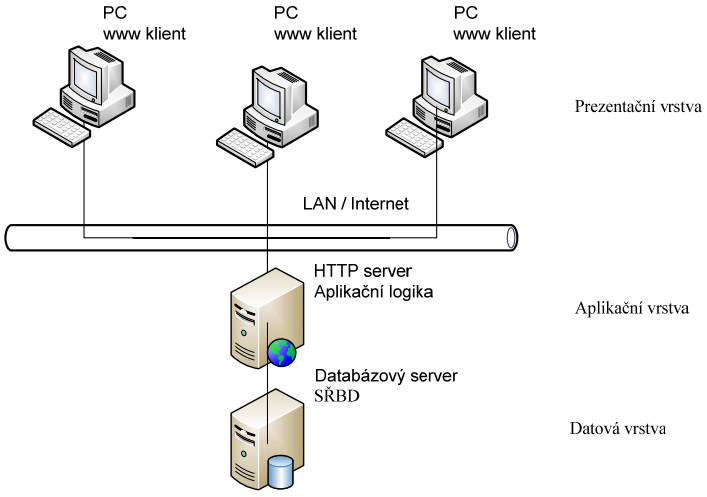
Ačkoliv přenosová rychlost sítí i internetu narůstá, tato architektura byla do budoucnosti neperspektivní,a proto vznikla třívrstvá architektura (obr. 2). Zde je hlavní myšlenkou zpracovávat data tam, kde jsou, a nikam je zbytečně nepřenášet. Aplikace se tak musely rozdělit na dvě části, jedna zpracovávala data na serveru a druhá část takto zpracovaná data publikovala uživateli. Je-li aplikace správně navržena, tak je přes síť přeneseno minimum dat. Charakteristickou vlastností serveru je jeho pasivita. Naslouchá na síti a reaguje na žádosti připojených autorizovaných klientů, při zpracování těchto žádostí bývá výpočetně mnohem více vytížen. Klient na druhou stranu je aktivní po celou dobu interakce s uživatelem, kdy s ním nejenom komunikuje přes grafické rozhraní, ale také posílá požadavky na server a přijímá odpovědi.

Model třívrstvé architektury rozlišuje tyto vrstvy:

- Prezentační vrstva – obsahuje funkce uživatelského rozhraní. Obvykle existuje několik prezentačních vrstev pro různé druhy zařízení, platformy a prostředí. V našem případě se jedná o webový prohlížeč využívající tuto vrstvu.
- Aplikační vrstva – tvoří prostředníka mezi vrstvou prezentační a vrstvou datovou . V této vrstvě dochází k transformací dat mezi vstupně / výstupními požadavky a datovou vrstvou. Nejčastěji se jedná o aplikační webový server s nainstalovaným skriptovacím jazykem např. PHP.
- Datová vrstva – obsahuje funkce pro přístup k informacím v datovém úložišti. Pro datové úložiště se volí samostatný databázový server, který komunikuje s aplikačním serverem prostřednictvím dotazů a odpovědí.

Strana klienta je reprezentována webovým prohlížečem, který je dostupný všem uživatelům přistupujícím na internet. Jedná se doslova o tenkého klienta, který nejenže

zobrazuje www stránky, ale dokáže si ze serveru stáhnout případně i klientský skript nebo jeho část, např. JAVA applet.



Obr. 2: Třívrstvá architektura

Mezi výhody oddělení prezentační vrstvy od aplikační a datové řadíme snadnou údržbu serveru. Staráme se pouze o serverovou část a případné opravy, modernizace, nebo přemístění serveru se dějí, aniž by to klienti poznali, nebo tím byli nějak ovlivněni. Další velkou výhodou je, že data jsou uložena na serveru daleko bezpečněji než u samotných klientů, protože se můžeme zaměřit na dobře fungující zabezpečení jednoho serveru a ne mnoha klientů.

2.2 Popis programovacích jazyků

Pro vývoj webového informačního systému existuje celá řada programovacích jazyku, frameworků a platforem. V této podkapitole se budu věnovat programovacím, skriptovacím a ostatním jazykům použitých při realizaci mého řešení informačního systému.

2.2.1 Unifikovaný jazyk UML

Jazyk UML není programovacím jazykem, nýbrž jazykem informativním, unifikovaným pro grafickou vizualizaci návrhu programovaného systému. Za vznikem jazyka UML stojí Grady Booch, Jim Rumbaught a Ivar Jacobson z firmy Rational SoftWare. Snažili se vytvořit jazyk určený pro objektově orientovanou analýzu a návrh systémů. V roce 1997 byl přijat standard UML v. 1.1.

Jedná se o unifikovaný jazyk - to znamená, že používá jednotných výrazových prostředků. UML jazyk nevyužijeme pouze u programování webových aplikací, ale najde hojné využití u všech nyní už objektových programovacích jazyků. UML umožňuje a usnadňuje návrh a vizualizaci různých typů aplikací. Aby byl jazyk UML opravdu univerzální, implementovali autoři přímo mechanizmy rozšíření, které usnadňují deklaraci a definici nových elementů jazyka. Další výhodou unifikovaného jazyka je, že ať se jedná o malý či velký projekt, sdílený mezi dvěma nebo deseti vývojáři, všichni se v něm bez větších problémů orientují z důvodu jednotné formální syntaxe.

Na jazyku UML jsou postaveny CASE nástroje, kdy vývojář nakreslí UML diagramy, ze kterých přímo vygeneruje spustitelný kód. Jazyk UML je tvořen třemi základními bloky, které jsou reprezentovány grafickými značkami.

1) Předměty: Předměty jsou na pracovní ploše UML reprezentovány jako obdélníky, elipsy, kružnice a jiné. Vyjadřují tzv. strukturní abstrakce UML modelu (programové třídy, komponenty atd.). Do předmětů patří i chování mezi objekty, které je značeno pomocí různých šipek či propojovacích čar. Objekty nejenže jdou propojovat, ale je možné je i seskupovat do diagramu na nižší úrovni, kdy se jedná o tzv. balíčky reprezentovány složkou. Posledním typem, který se řadí do předmětů jsou poznámky specifikující vlastnosti a chování elementů UML diagramu.
2) Relace: Vztahy mezi jednotlivými předměty specifikují relace. Mezi základní typy relací v jazyce UML řadíme:
   - Asociace: spojení mezi třídami. Závislost propojení prvků, kdy při změně jednoho prvku se tato změna projeví i u druhého.
   - Generalizace: jeden prvek dědí vlastnosti prvku druhého.
   - Realizace: prvek je odpovědný za prvek jiný
3) Diagramy: Diagramy graficky zobrazují aspekty modelového systému, zachycují prvky a jejich vztahy. Digramů existuje velké množství a každý obsahuje odlišné grafické značky.

Ačkoliv je UML jazyk velice uznávaný při návrhu a dokumentaci programů, tak se ale také může stát, že jeden styl může znamenat ve dvou různých typech diagramu různou vlastnost. Dále obsahuje mnoho schémat a konstrukcí, které jsou nadbytečné či zřídka používány [20].

2.2.2 Skriptovací jazyk PHP

PHP je zkratkou pro PHP Hypertext Preprocesor, avšak původní význam byl „Personal Home Page“, a šlo o balíčky skriptů v jazyce Perl [3].

Jedná se o skriptovací jazyk zabudovaný na straně serveru, to znamená, že HTML kód je doplněný o výkonné příkazy (PHP skripty), které se před odesláním dokumentu provedou. Jazyk PHP byl stejně jako např. ASP.NET přímo vyvíjen pro tvorbu webových aplikací a služeb. Skriptovací jádro PHP má optimalizovanou dobu odezvy potřebnou ve webových aplikacích [3][7].

Jádro od verze PHP4 tvoří tzv. Zend Engine, díky kterému se zvýšil výkon jazyka a tím se i zrychlilo zpracování samotných skriptů. S příchodem PHP5 bylo jádro vylepšeno o plnou podporu objektového programování s označením Zend Engine 2.

Jazyk PHP se skládá z obvyklých doplňků matematických, přes podporu databází, práce se soubory a s adresáři, komprese a dekomprese dat, funkce pro práci s elektronickou poštou až po funkce pro čtení informací ze síťových zařízení pomocí protokolu SNMP. Další důležitou vlastností je, že výstupem nemusí být nutně pouze HTML kód, ale při použití specializovaných knihoven můžeme generovat obrázky, PDF soubory atd. [7].

Pro PHP od verze 4 také existují frameworky, které usnadňují vývojářům práci, kdy se nemusejí zabývat například navigací mezi stránkami, ale můžou se plně věnovat vývoji zadané aplikace. Jedním z frameworků pro PHP je i Zend framework, který je vyvíjen přímo tvůrci PHP.

PHP je zcela open source, tedy software s otevřeným zdrojovým kódem dostupný pro všechny zdarma. Proto je tento skriptovací jazyk ve spojení s webovým serverem Apache a operačním systémem Linux tolik rozšířený a oblíbený u široké veřejnosti.

2.2.3 Skriptovací jazyk JavaScript

JavaScript vznikl v roce 1995 ve společnosti NCC[1]. Navzdory svému jménu JavaScript nikterak nesouvisí s jazykem Java, jak by se podle názvu mohlo zdát. Oba programovací jazyky jsou však obchodní značkou firmy SUN Microsystems. JavaScript je rychlý skriptovací jazyk vkládaný do webové stránky. Dříve docházelo k načítání JavaScriptu do prohlížeče přímo s požadovanou webovou stránkou, nyní s příchodem

[1] NNC – Netscape Communications Corporation

technologie Web 2.0 a Ajaxu je možné JavaScript načítat do prohlížeče nezávisle na stahování a obnovování webové stránky v prohlížeči. Samotný program napsaný v jazyce JavaScript je vykonán až přímo u klienta v prohlížeči [9][25][10]..

JavaScript společně s HTML, CSS a XML tvoří základ tzv. „Dynamic HTML“, jedná se o dynamické a interaktivní webové stránky, které svojí funkčnost nezajišťují applety. Takto vytvořené webové stránky je možné obsahově, nebo graficky měnit i po načtení ze serveru.

2.2.4 Značkovací jazyk HTML

HTML není programovací, nýbrž značkovací jazyk. Vkládá se přímo do textu a určuje, jak se budou jednotlivé části textu zobrazovat ve webovém prohlížeči, případně vkládá odkazy na externí soubory (obrázky, zvuky). Tento jazyk prošel dlouhým vývojem, přičemž nejvýznamnější verze jsou 3.2 (schválená organizací W3C jako standard v lednu 1997) a 4.0 (schválená v prosinci 1997). Mezi oběma verzemi jsou poměrně velké rozdíly: verze 4.0 vypouští některé značky používané v předchozích verzích a nahrazuje je použitím kaskádových. Jak jsem již uvedl, HTML není programovací jazyk. Co má tedy společného s programováním? V počátcích webu se v nových technologiích orientovalo jen málo lidí a vytvoření vlastní stránky vyžadovalo poměrně hodně technických znalostí. Web prostě přinesl mnoho nového, na co se ne každý byl schopen rychle adaptovat [12].

Tvorbu webových stránek od počátku ztěžovalo množství různých implementací HTML. Samotný jazyk je standardem přesně specifikovaný - mj. pomocí tzv. DTD (Document Type Definition), což je soubor, který popisuje, které značky (tagy) a jejich atributy jsou povolené a v jakém pořadí se mohou vyskytovat. Nicméně různí výrobci prohlížečů přidávali svoje vlastní značky či atributy, které nebyly implementovány v jiných produktech. Navíc jednotlivé prohlížeče zobrazovaly stejný kód různě. Vznikal tak zmatek a tvůrce stránek musel (a stále ještě musí) kontrolovat výsledný vzhled v různých prohlížečích [8].

2.2.5 Značkovací jazyk XHTML

XHTML (extensible hypertext markup language*)* je rozšířený hypertextový značkovací jazyk, který vznikl jako nástupce jazyka HTML. Jelikož však vývoj HTML byl obnoven, tak vývoj obou značkovacích jazyků pokračuje paralelně. XHTML

využívá vlastností HTML a také XML, před vlastním dokumentem musí být uvedena deklarace XML. Na rozdíl od HTML je nutné striktně dodržet specifikaci, například:

- všechny tagy musejí být ukončené a to i nepárové
- tagy i jejich atributy musejí být psány malými písmeny
- hodnoty atributů musejí být uzavřeny do uvozovek

Díky těmto jednoduchým, ale zato velice přísným pravidlům může počítač, respektive webový prohlížeč XHTML velmi snadno automatizovaně zpracovávat [18].

2.2.6 Styly CSS

CSS (Cascading Style Sheets) neboli tabulka kaskádových stylů je jazyk pro grafickou úpravu webových stránek. V roce 1996 vznikla první specifikace CSS1 vydána konsorciem W3C (World Wide Web Consortium). CSS odděluje jádro webových stránek (text, obrázky atd.) od jeho vizuální prezentace. Toto oddělení umožňuje jednoduše přidávat, odstraňovat nebo aktualizovat obsah dokumentu aniž by jsme se museli starat o vizuální stránku nového textu. Jelikož CSS vychází ze stromové struktury HTML, dochází k dědění vlastnosti prvku blíže ke kořenu na další prvky (potomky). Jednoduší je také s použitím CSS měnit například styl písma či barvu, už nemusíme hledat v celém obsahu webové prezentace tag <*font*>, ale vše změníme v jedné definici písma. Předpis CSS do HTML dokumentu lze vložit třemi způsoby:

1. připojení externího souboru k dokumentu tagem <*link*>, jedná se o nejvýhodnější řešení, kdy předpis je načten pro celý dokument při první návštěvě webové stránky.
2. umístění předpisu do hlavičky dokumentu v tagu <*style*>, nevýhodou je, že při každém načtení dokumentu se musí načíst i jeho předpis.
3. vložení předpisu přímo do samotného tagu, využívá se pouze ojediněle pro tagy, které není nutné takřka vůbec měnit.

Navíc oddělením dvou vrstev a použitím externího souboru můžeme jednoduše a rychle změnit celkový vzhled webové prezentace aniž bychom se dotkli samotného obsahu. Pouze se vytvoří nová šablona [4].

2.3 Protokoly

V této podkapitole jsou popsány síťové protokoly použité při komunikaci ať už mezi HTTP serverem a routerboardem, ale i mezi webserverem Apache a klientským webovým prohlížečem.

2.3.1 Protokol HTTP

HTTP (Hypertext Transfer Protocol) [16][19] používá TCP port 80, prvotně byl vyvinut pro přenos hypertextových dokumentů. S postupem vývoje internetu se vylepšoval i http, v současné době je schopen přenést i další informace, případně i s rozšířením MIME přenášet soubory. HTTP je tzv. „bezestavový protokol“, to znamená, že funguje na principu dotaz-odpověď. Uživatel zašle ve formě textu informace o své konfiguraci a akceptovaných formátech dokumentů serveru, server poté zašle žadateli odpověď obsahující informace o požadovaném dokumentu a samotná data.

HTTPS [16][19] je nástavbou protokolu http, zvyšuje bezpečnost přenášených dat přes internet mezi webovým serverem a prohlížečem. Data jsou přenášena pomocí stejného protokolu http a navíc jsou šifrována pomocí SSL nebo TLS, což zabraňuje jejich odposlechnutí v nešifrované podobě. Ovšem šifrování dat nám nezabrání podvržení dat od útočníka, proto se používají ještě navíc certifikáty a certifikační autority, které určí identitu komunikujících protistran. Organizace provozující webovou stránku si nechá podepsat certifikát certifikační autoritou, ta garantuje, že žadatel o certifikát skutečně existuje a je držitelem domény, která má být v certifikátu uvedena. Uživatel si poté může zkontrolovat, jestli byl certifikát vystaven opravdu na danou webovou stránku, kterou provozuje uvedená organizace. Protokol https implicitně komunikuje na portu TCP 443.

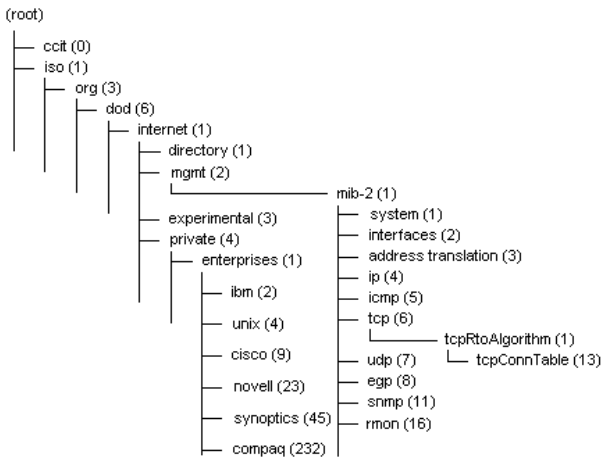
2.3.2 Protokol SNMP

SNMP neboli „Simple Network Managment Protocol“ označuje síťový protokol pro management TCP/IP a IPX sítě. Byl vyvinut pro nejrůznější sběr dat z aktivních síťových prvků a jejich následné vyhodnocení. Historie tohoto protokolu sahá až do roku 1990, kdy byl SNMP schválen jako standard RFC1157 [2], následně nato vznikla verze SNMP-2 a nyní se používá SNMP-3. Největším nedostatkem SNMP-1,2 byla bezpečnost, zabezpečení přístupu k SNMP agentovi byla řešená prostřednictvím hesla, které bylo zasíláno vždy s každým paketem v otevřené formě, to je v řetězci „community string“, který si mohl útočník jednoduše přečíst. Tento problém se podařilo

vyřešit až ve verzi SNMP-3, kdy je ochrana dat realizovaná algoritmem DES. SNMP je založen na protokolu UDP a pro komunikaci využívá privilegované porty 161 a 162. Protokol SNMP je asynchronní, transakčně orientovaný protokol obsahující 2 primární elementy *manager* a *agents*. Strana, která posílá požadavky a vykonává management sítě je manager, též někdy označován SNMP klient. Naopak strana, která na požadavky odpovídá, se označuje agents nebo také SNMP agent. Agent ukládá informace o aktuálních stavech do tzv. informační báze MIB. Oba elementy mohou běžet buď odděleně na různých zařízeních, nebo v rámci jednoho zařízení [1].

Mnoho programovacích jazyků (PHP, AAJS, ASP, C, Java atd.) mají v sobě již implementovanou podporu protokolu SNMP, proto je možné je využít pro zpracování a zobrazení dat získaných přes SNMP.

MIB (Management Information Base) je databáze obsahující informace (objekty), které manager i agent může získat a předávat dále (obr. 3)[1]. Aby to bylo možné, je nutná znalost struktury MIB jak pro managera, tak i pro agenta. Jedná se o stromovou strukturu a informace jsou uloženy v listech stromu, což zpřehledňuje celou databázi objektů. Každý uzel stromu je jednoznačně označen a to jak slovně, tak i číselně. Pro vyhledání dané informace ve stromové struktuře se používá identifikátor objektu tzv. OID, který jednoznačně určuje cestu od kořene až po pozici uzlu ve stromu. Kromě číselného OID lze použít i slovní ekvivalent.



Obr. 3: Ukázka MIB databáze

Abychom byli schopni číst informace z RouterBoardu přes protokol snop musíme znát strukturu databáze MIB, jak už bylo řečeno. Pomocí této databáze a identifikátoru OID jsme schopni se dotázat SNMP serveru (agenta) přímo na požadovanou informaci.

Jedná se o velice složitou a rozsáhlou databázi [17]. Každý síťový prvek, od různých výrobců, má nepatrně odlišnou strukturu MIB. Potřebné OID lze získat několika způsoby. První možností je vyhledat ho pomocí internetového vyhledávače zadáním výstižné kombinace slov hledaného výrazu. Výrobci hardwaru většinou zveřejňují MIB databáze na svých internetových stránkách, kde je možné si je prohlédnou případně stáhnout, to je další možnost jak najít požadované OID. Třetí a nejelegantnějším způsobem jak získat OID je použít ať už komerční nebo volně šiřitelný program, který se dokáže připojit k danému prvku a stáhnout celou MIB databázi. Poté tuto databázi dokáže přehledně zobrazit ve stromové struktuře a my jsme schopni se „proklikat“ až k požadovanému OID. V případě OS MikroTik [14] lze ještě získat OID i příkazem v příkazové řádce, kdy například příkazem

```
interface wireless print oid
```

vypíšeme OID WiFi rozhraní.

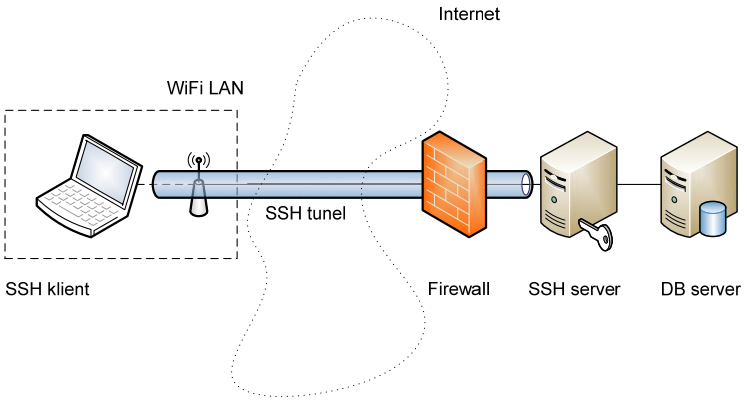
2.3.3 Protokol SSH

SSH (Secure Shell) znamená v překladu „bezpečná příkazová řádka“, název byl odvozen z programu RSH (Remote Shell). Ani v jednom uvedeném případě se nejedná ve skutečnosti o příkazový interpret shell. SSH je jak program, tak i protokol pracující na portu TCP/22. Historie tohoto protokolu sahá do roku 1995, kdy po útoku odchytáváním hesel na počítačovou síť univerzity v Helsinkách vznikla první verze protokolu SSH-1. V roce 1996 byla vydána už komerční verze protokolu SSH-2, který bohužel nebyl zpětně kompatibilní s SSH-1 [19][16].

Protokol a samotný program SSH se používá pro bezpečnou komunikaci mezi dvěma počítači, která je šifrovaná. Komunikaci můžeme nazvat bezpečnou, jsou-li splněny tři základní požadavky: autentizace obou účastníků, šifrování a integrita přenášených dat.

Celý proces navázání komunikace je následující: celá komunikace pracuje na architektuře typu klient/server (obr. 4). Klient nejdříve naváže nezabezpečené připojení na server a vyžádá si jeho veřejný RSA/DSA klíč. Ke klíči veřejnému je i v páru klíč soukromý, který zná a vlastní pouze server. Jelikož si klíče může vygenerovat takřka každý, je nutné, chceme-li být považování za důvěryhodný server, si tyto klíče nechat podepsat certifikační autoritou (CA). Ta ověří, například podle dokladu, že za

certifikátem (klíčem) se vydává opravdu ta osoba (organizace), která je uvedena v certifikátu. Pokud je certifikát (klíč) podepsán certifikační autoritou, které klient věří, říkáme, že je certifikát (klíč) autentický. Pokud je klíč podepsán sám sebou (Self-signed Certificate), tak je také autentický, ale nemáme jistotu, kdo ho vydal. Jakmile dostaneme veřejný klíč, porovnáme ho se svým seznamem známých serverů, ověříme, zda-li je podepsán známou certifikační autoritou, a tím ověříme, že server je opravdu tím, za koho se vydává. Proto, pokud se přihlašujeme k novému serveru poprvé, jsme dotázáni, zda chceme server přidat do seznamu, případně jsme varováni, že klíč není podepsán CA. Pokud se k danému serveru připojujeme vícekrát a jeho identifikace se změní, je to znamení, že něco není v pořádku a měli bychom si opětovně ověřit ke komu se skutečně připojujeme. Dále klient vygeneruje klíč spojení, který zakóduje veřejným klíčem serveru a odešle ho serveru. Klient a server se domluví na použití šifry (IDEA, Blowfish, 3DES, DES, AES) a pomocí vybrané šifry a dohodnutého klíče sestaví zabezpečené připojení. Klient se poté autentizuje například pomocí hesla, které je už ovšem přenášeno zabezpečeným kanálem a útočník ho nemůže odposlechnout.



Obr. 4: Komunikace přes SSH

Služba SSH neslouží jenom k práci v příkazové řádce na vzdáleném počítači, ale můžeme jí využít i pro transfer dat podobně jako FTP jen bezpečněji. SFTP neboli SSH File Transfer Protocol využívá pro zabezpečení a autentizaci právě SSH-2. Kromě komerční platformy SSH byla vydána i Open Source varianta OpenSSH postavená na knihovně OpenSSL umožňující šifrování a ověřování totožnosti.

3. Zařízení

Abychom mohli realizovat webový informační systém, je nutné použít zařízení, na kterém bude tento systém nainstalován a spuštěn, jedná se o HTTP server. Druhým zařízením popsaným v této kapitole je routerboard s OS MikroTik.

3.1 HTTP server

Webserver neboli webový server je softwarová instance běžící na operačním systému (OS) počítače zapojeného do počítačové sítě. Tento server přijímá a zpracovává požadavky, vrací odpovědi nejčastěji webovému prohlížeči. Dalo by se říci, že se jedná o interaktivní software reagující na požadavky uživatele prostřednictvím webového prohlížeče. Operačním systémem může být ať už UNIX, LINUX nebo WINDOWS, v tomto projektu je vše postavené na OS linux a v tomto případě se jedná o linuxovou distribuci OS Debian.

Webových serverů je na trhu několik: Apache, Internet Information Services či Sun Java Systém Web server, některé jsou zdarma, jiné jsou komerční. Pro náš HTTP server použijeme volně šiřitelný webserver Apache jenž se na internetu používá z více jak 50%[2].

Tímto máme připraven HTTP server, ovšem abychom na něm mohli spustit informační systém, je nutné ještě doinstalovat aplikace pro podporu skriptovacího jazyka PHP, databázový systém MySQL či openSSH pro bezpečnou komunikaci mezi klientem a webserverem. Instalace aplikací pro běh HTTP serveru je popsána v kapitole 7 Instalace a konfigurace HTTP serveru.

3.1.1 Web server Apache

Apache je v současné době nejoblíbenějším a nejrozšířenějším[2] web serverem. Tento web server byl vyvinut jako pokračovatel staršího NCSA serveru z dílny National Centre for Supercomputer Aplications.

Název apache vznikl ze spojení „A patchy server“, protože apache vznikl ze záplat (patche) pro jiný webový server - NCSA HTTPd.

Apache je vyvíjen jako public domain, jeho zdrojové texty, binární distribuce a dokumentace je možné získat z oficiálních webových stránek. Architektura Apache je

[2] http://news.netcraft.com/archives/2008/11/19/november_2008_web_server_survey.html

modulární, což znamená, že je možné nainstalovat přídavné moduly, které zajistí funkce proxy serveru, autentifikaci a podobně [3][12].

3.1.2 Databáze MySQL

MySQL je velmi výkonný relační databázový systém s architekturou klient/server. Za jeho vznikem stojí firma MySQL AB a nyní SUN Microsystems. V roce 1995 vznikla první verze určená pro vnitřní potřebu firmy, postupem času vznikl vysoce výkonný relační databázový systém, který dokáže pojmout velké množství dat, aniž by přitom ztratil mnoho ze svého výkonu. Navíc není vázán na jedinou platformu, ale můžeme jej využívat téměř na všech dnes používaných platformách. MySQL je open-source projekt, který je šířen pro nekomerční použití zdarma. V případě komerčního využití je sice nutno zaplatit určitou cenu, ale i tak se dostaneme na zlomek ceny konkurenčních produktů. Navíc rozhodneme-li se využít MySQL komerčně, získáme jeden „bonus“ – možnost zásahu do zdrojových kódů, zatímco při nekomerčním využití smíme do zdrojových kódů pouze nahlédnout pro studijní účely. Tímto způsobem získáme verzi, která bude maximálně vyhovovat všem požadavkům [22].

V dnešní době snad každá firma ukládá svá cenná data v elektronické podobě. Někdy jsou tato data určena k interním účelům, jindy zase k veřejnému publikování. Ať tak či onak, užití MySQL je opět na místě. Díky MySQL budou data vhodným způsobem zabezpečena proti možnému zneužití. K těmto datům je pak možno přistupovat pomocí různých rozhraní. V MySQL jsou přímo dostupná rozhraní pro C/C++, Perl, Javu, Python, PHP či ODBC. V případě potřeby je možno další rozhraní naprogramovat [11].

Od verze 5.0 MySQL podporuje pohledy a úložné procedury, to je výhodné pro programátory klientů, protože není potřeba pracovat přímo s tabulkami a mohou se spolehnout na uložené procedury. Další velkou inovací bylo zavedení tzv. triggerů, které server automaticky spouští při určitých operacích. A nesmíme zapomenout na replikaci, která umožňuje kopírovat databáze na více serverů a zvýší odolnost proti výpadku databázového serveru.

3.2 Routerboard s OS MikroTik

Historie operačního systému MikroTik sahá až do roku 1995, kdy byl spuštěn v Lotyšsku vývoj a prodej bezdrátových systémů pro poskytovatele internetového připojení. Celé to bylo zastřešeno pod obchodním jménem MikroTik.

Během tohoto vývoje vzniklo několik řad routerboardů (RB), které se lišily především výkonem, spotřebou elektrické energie, rychlostí CPU, velikostí zařízení a s tím souvisejícím počtem rozhraní routerboardu. Ať už se jedná o jakoukoliv verzi RB, vždy obsahuje sériový port pro komunikaci a konfigurace, jeden ethernetový port umožňující taktéž konfiguraci přes LAN síť, ale také v dnešní době velice oblíbené napájení zařízení po LAN síti tzv. „Power of Ethernet“. Deska RB je osazena nejčastěji několika miniPCI sloty pro přídavné karty, ethernet porty a už zmiňovaným RS323 konektorem.

Operačním systémem RB nemusí být nutně již zmiňovaný OS Mikrotik, ale můžeme do RB nahrát například OS Linux distribuce Debian. Ovšem musíme brát v úvahu hardwarové nároky takového systému a to především velikost datového prostoru, kdy RB obsahuje integrovanou flash paměť bez možnosti rozšířit jakkoliv diskovou kapacitu zařízení. Pro běžné správce bezdrátových sítí je ovšem zcela dostačující dodávaný operační systém MikroTik [15].

OS MikroTik je postaven na velice stabilním linuxovém jádře a založený na uClibc[3] a BusyBoxu[4]. Ačkoliv se hlásí k GNU software, tak tuto myšlenku naopak zatracuje neposkytnutím zdrojových kódů. K routerboardu s OS MikroTik (ROS) je možné se připojit několika způsoby a to pomocí telnetu, což se příliš z bezpečnostních důvodů nedoporučuje, či přes SSH. V obou případech kromě bezpečnosti dosáhneme stejného výsledku. Po připojení se dostaneme do klasické linuxové konzole, neboli do příkazového interpretu shell. Kdo už někdy pracoval v linuxu bez grafického rozhraní si velmi rychle zvykne.

Příklad nastavení administrátorského hesla přes SSH:

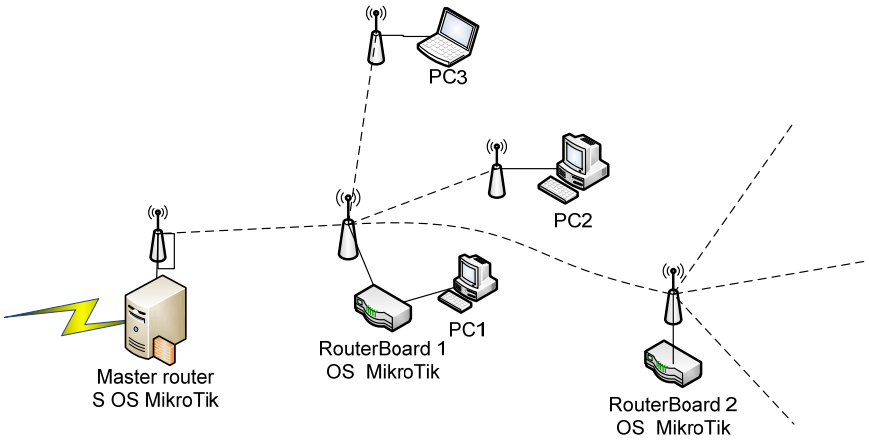
```
user edit admin password
```

Další možností jak ovládat RB je prostřednictvím windows aplikace Winbox. Jedná se o „malý“ program, který nepotřebuje instalaci a proto ho je možné mít na flashdisku

[3] http://www.uclibc.org/about.html
[4] http://www.busybox.net/

vždy při sobě. Prostřednictvím tohoto programu se můžeme vzdáleně připojit k ROS se znalostí jeho IP či MAC adresy a hesla. Jedná se o intuitivní grafické rozhraní, kde jsme po několika minutách osvojování schopni nastavit základní konfiguraci RB.

V případě, že bychom chtěli využívat RB jako hlavní router s firewall či proxy serverem jako hlavní vstupní (hraniční) aktivní prvek u rozsáhlejší bezdrátové sítě, tak nám nebude ani nejvýkonnější řada RB výkonově stačit (obr. 5). I na toto vývojáři pomysleli a je možné implementovat OS Mikrotik i na běžná výkonná PC, která lze díky tomu použít jako router s OS Mikrotik, protože je dostupný i pro x86 architekturu. Nejjednodušší cestou jak tohoto dosáhnout, je objednat si compact flash kartu od výrobce či distributora uzpůsobenou k připojení do IDE slotu v PC.



Obr. 5: WiFi síť z routerboardu

Spojením routerboardu a OS MikroTik dostaneme velmi výkonný a propracovaný router, který nemusí být použit výhradně pro bezdrátové sítě, ale také jako aktivní prvek LAN sítě.

4. Instalace, konfigurace HTTP serveru

Jak už jsem psal dříve, jako operační systém HTTP serveru byl zvolen OS Linux z distribuce Debian. Jedná se o volně šiřitelný software vhodný pro serverovou instalaci, který má nespočet rozšiřujících balíčků. Doporučené hardwarové požadavky pro OS Linux jsou následující[5]:

- architektura procesoru: Alpha, AMD64, ARM, HP PA-RISC, Intel x86, Intel IA-64, MIPS (big endian), MIPS (little endian), PowerPC, IBM S/390, SPARC
- procesor: alespoň Pentium 4

[5] http://www.debian.org/releases/stable/i386/ch03s04.html.cs#id2532656

- operační paměť: 512MB
- pevný disk: 5GB

Nejsou kladeny žádné velké nároky na grafickou kartu, protože se budeme pohybovat pouze v příkazovém řádku, proto postačí integrovaná grafická karta. Jelikož se jedná o HTTP server obsahující i databázový server MySQL, budou kladeny velké nároky na procesor a velikost operační paměti systému, a to z toho důvodu, že server může v jednom okamžiku zpracovávat velké množství dat. Vyplatí se proto investovat do kvalitního a výkonného hardwaru, protože server bude v provozu 24 hodin denně, 365 dní v roce.

4.1 Instalace

Pro instalaci OS Debian si stáhneme z internetových stránek [21] obraz CD, které obsahuje základní data nutná pro instalaci, toto CD si vypálíme a poté z něho nainstalujeme systém. Při stahování obrazu máme na výběr z několika možností. Vybereme a stáhneme, pokud máme trvalé připojení k internetu, minimální síťovou instalaci, která umožňuje započetí instalace, a potřebný zbytek dat se dodatečně stáhne z internetu během instalace.

Instalace OS Debian není nikterak složitá, po „nabootování“ z instalačního CD máme na výběr z typu instalace (grafická, textová), pokud vybereme instalaci s grafickým průvodcem, tak je instalace OS Debian stejně uživatelsky přívětivá jako například instalace OS Windows.

Po instalaci operačního systému je nutné nainstalovat služby, které na daném HTTP serveru poběží. Pro instalaci služeb použijeme softwarové balíčky určené pro OS Debian, běžně se instalují příkazem *apt-get install balíček*, my ovšem použijeme příkaz *aptitude install baliček*. Je to z důvodu, že při této instalaci jsou k danému softwaru doinstalovány i další vázané balíčky s instalovanou aplikací.

Nejdříve nainstalujeme webový server Apache

```
aptitude install apache2
```

Kromě samotného apache se nám automaticky nainstaluje i skriptovací jazyk perl. Pro nezasvěcené do problematiky webových serverů by se mohlo zdát, že nainstalováním serveru apache je vše hotovo, to ale není pravda. Je dále potřeba nainstalovat stejným

příkazem skriptovací jazyk PHP, MySQL databázi, správce databáze phpMyAdmin a také SSH pro bezpečný přenos dat a přihlašování na server.

```
aptitude install php5
aptitude install mysql-server
aptitude install phpMyAdmin
aptitude install ssh
aptitude install openssl
aptitude install rrdtool
aptitude install expect
```

Poslední instalovanou aplikací byla aplikace Expect, jedná se o program určený pro komunikaci s interaktivními aplikacemi, jako jsou telnet, ftp, ssh atd. V závislosti na předaném skriptu Expect ví, jaký výstup má od programu očekávat a jakou má dát programu odpověď. Expect může být buď zavolán jako program a nebo se vytvoří „expectovský“ skript, který je spouštěn například prostřednictvím aplikace cron [23]. Po nainstalování tohoto programu není nutná žádná jeho další konfigurace, konfiguraci ostatních nainstalovaných programů si popíšeme v další kapitole.

4.2 Konfigurace služeb

Tímto máme nainstalovány základní služby pro běh HTTP serveru. Aby však služby pracovaly korektně, je nutná ještě jejich konfigurace.

4.2.1 Konfigurace Apache

Nejdříve nakonfigurujeme webový server Apache. Jelikož server Apache umožňuje vytvářet virtuální domény, tak si pro náš informační systém vytvoříme virtuální doménu na portu 443, na kterém funguje protokol https. Další virtuální doménu můžeme vytvořit standardně na portu 80 http, či na jakémkoliv jiném neprivilegovaném portu. Tak to definované virtuální domény můžeme použít například na přesměrování www požadavků neplatících klientů na naši virtuální doménu s varováním. Nastavení virtuálních domén na webovém serveru Apache se provádí v souboru */etc/apache2/sites-available*

```
<VirtualHost *:8080>
NameVirtualHost *:8080
DocumentRoot /var/www/neplatici/

 DocumentRoot /var/www/neplatici/
        <Directory />
```

```
                Options FollowSymLinks
                AllowOverride All
        </Directory>
        <Directory /var/www/neplatici/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride All
                Order allow,deny
        </Directory>
</VirtualHost>
```

Zde je vidět vytvoření virtuální domény pro neplatící klienty, pokud v informačním systému označíme klienta statutem „Odpojený klient“ je automaticky při pokusu o navázání spojení přes webový prohlížeč přesměrován na webovou prezentaci v adresáři */var/www/neplatici/*. O přesměrování na adresu HTTP serveru a portu 8080 se stará hlavní router. Při každé změně konfigurace Apache je nutné službu restartovat.

4.2.2 Konfigurace PHP

U PHP je nutné správně nastavit direktivy z důvodu bezpečnosti. Nejdříve si ukážeme, které direktivy zakázat:

- display_errors = off – zobrazením chyb, dáváme útočníkovi možnost odhalit bezpečnostní chyby naší aplikace
- safe_mode = off – jedná se o nespolehlivý bezpečnostní režim
- magic_quotes_sybase = off – mění ošetření speciálních znaků u SQL dotazů
- register_globals = off – útočník by mohl přes URL přidat novou globální proměnnou s libovolnou hodnotou, což je velice nebezpečné

[3]

S výjimkou první direktivy se s ostatními ve verzi PHP6 nepočítá. Nyní si ukážeme, které direktivy by měly být naopak povoleny:

- open_basedir - slouží k zamezení přístupu mimo zadaný adresář
- disable_functions – tato direktiva definuje, které funkce není možné spustit, např. exec() – spustil by externí program
- session.use_only_cookies – tato direktiva zamezuje předání ID sessions jinak než přes cookie
- session.save_path a upload_tmp_dir – nastavení adresářů pro uložení dočasných dat

[13]

Takto nastavené prostředí v PHP nám zajistí částečnou bezpečnost ze strany příchozích požadavků na HTTP server. Ovšem musíme pamatovat na to, že i nejlépe zabezpečený server může mít bezpečnostní hrozbu v podobě špatně naprogramované aplikace.

4.2.3 Konfigurace phpMyAdmin

V neposlední řadě je nutné také nastavit webovou aplikaci phpMyAdmin, která slouží jako klientské rozhraní pro řízení databáze MySQL. Pro základní spuštění je nejnutnější správně nastavit adresu MySQL serveru, toto nastavení najdeme v souboru /usr/share/phpmyadmin/config.inc.php:

```
$cfg['Servers'][1]['host'] = 'localhost';
```

Ve výchozím stavu je adresa nastavena na *localhost,* kdybychom měli oddělený databázový server od webového serveru, museli bychom uvést IP adresu databázového serveru. Jelikož nejsou v tomto souboru informace šifrována, z důvodu bezpečnosti nevyplňujeme žádná hesla, heslo se zadává až při přihlášení do phpMyAdmin.

4.2.4 Konfigurace MySQL

U MySQL databáze je vhodné vytvořit zvláštní účet pro PHP skripty, které přistupují k databázi. Tento účet by měl mít omezené oprávnění pouze na čtení, vkládání, opravu či mazání dat z databáze vytvořené pouze pro tento účet. V případě útoku na databázi takto zajistíme, že útočník modifikuje „pouze“ data, která pravidelně zálohujeme, a ne strukturu databáze či databázi jinou.

4.2.5 Vytvoření SSH kanálu

Pro vytvoření bezpečné komunikace mezi HTTP serverem a routerboardem je potřeba vytvořit zabezpečený komunikační SSH kanál. Abychom toho mohli dosáhnout, musíme si nejdříve vygenerovat certifikáty, pomocí kterých se budeme na routerboard přihlašovat. Certifikáty si vygenerujeme na HTTP serveru pomocí programu ssh-keygen.

```
ssh-keygen –t dsa
```

Po zadání příkazu budeme dotázáni na jméno klíče a heslo, ani jeden údaj nesmí být vyplněn, abychom poté při přihlášení nemuseli zadávat heslo. Klíče jsou uloženy

v domovském adresáři uživatele, pod kterým byly klíče vygenerovány. Nyní stačí veřejný klíč z dvojice klíčů nahrát mezi soubory na routerboardu a poté v záložce *Users* a *SSH Keys* naimportovat klíč danému uživateli. Nyní se můžeme přes SSH přihlásit na routerboard, aniž by po nás systém vyžadoval heslo.

5. Praktická část

Praktická číst je věnována důkladné analýze návrhu informačního systému především z hlediska bezpečnosti a správného návrhu datového modelu. Druhá část je věnována podrobné studii skriptů pro vzájemnou komunikaci mezi serverem a routerboardem.

5.1 Analýza možností přihlašování a ukládání hesel

Po předchozí kapitole, kde jsme se věnovali základnímu nastavení zabezpečení skriptovacího jazyku PHP a šifrovanému kanálu SSH, přichází na řadu kapitola věnovaná základním ověřovacím mechanismům a způsoby uložení uživatelských hesel na serveru.

5.1.1 Způsoby přihlášení do IS

Budeme se věnovat dvěma základním ověřovacím mechanizmům definovaným ve specifikaci HTTP standartu RFC[24]. Nepomineme ani ověření pomocí certifikátů a také webové ověřovací mechanizmy použité i v této práci.

HTTP Basic ověření [24]

Ověřovací mechanizmus Basic je nejzákladnějším ověřovacím mechanizmem pro webové aplikace. Byl poprvé standardizován ve specifikací HTTP. Tento mechanizmus již není z bezpečnostních důvodů využíván a důvod si popíšeme v následujících řádcích.

Obr. 6: Formulář pro přihlášení přes IE6

Při vstupu na webovou stránku, která je chráněna heslem, pošle prohlížeč na server požadavek pro přístup. Server vrátí odpověď s parametrem ***HTTP Reason:*** *Unauthorized*

```
HTTP - Hyper Text Transfer Protocol[6]
  HTTP Version:           HTTP/1.1
  HTTP Status:            401
  HTTP Reason:            Unauthorized
  Cache-Control:          private
  Connection:             close
  Date:                   Sat, 14 Mar 2009 19:30:56 GMT
  Content-Type:           text/html
  Expires:                Sat, 14 Mar 2009 19:30:56 GMT
  Server:                 Microsoft-IIS/6.0
  X-Powered-By:           ASP.NET
  WWW-Authenticate:       Basic realm="server.cz"
```

Po přijetí odpovědi prohlížeč zobrazí přihlašovací formulář (obr. 6). Tento formulář není vytvořen operačním systémem, ale webovým prohlížečem. Prohlížeče mají v sobě zaimplementovány podpůrné programy, které reagují na odpovědi od serveru např. přihlašovacím formulářem.

Uživatel vyplní potřebné údaje a prohlížeč vzápětí odešle serveru stejnou žádost jako poprvé, pouze s pozměněnou hlavičkou o položku „*Autorization*“, ve které je uloženo přihlašovací jméno a heslo.

[6] Výstup z analyzátoru datových toků OmniPeek

```
HTTP - Hyper Text Transfer Protocol[7]
  HTTP Command:          GET
  URI:                   /?cls=login
  HTTP Version:          HTTP/1.1
  Host:                  server.cz
  User-Agent:            Mozilla/5.0 …
  Accept:                text/html,application/xhtml+xml…
  Accept-Language:       cs,en-us;q=0.7,en;q=0.3
  Accept-Encoding:       gzip,deflate
  Accept-Charset:        UTF-8,*
  Keep-Alive:            300
  Connection:            keep-alive
  Referer:               http://server.cz/
  Authorization:         Basic bWNvbjJEjkr0umprNTQ4NA==
```

Uživatelské jméno a heslo je na první pohled přenášeno bezpečně v zašifrované podobě, ale u metody Basic to není úplně pravda. Jméno a heslo je zakódováno algoritmem Base 64, což vede k jistému pocitu bezpečí, ale tento algoritmus je možné dekódovat za několik málo vteřin, tudíž by se dalo říci, že je přenášeno takřka v otevřené formě.

Další bezpečnostní hrozbou u metody Basic je opakované zasílání jména a hesla při dalších požadavcích na webový server. To znamená, i když přihlášení provedeme pomocí SSL tedy HTTPS a další část naší webové prezentace již není takto zabezpečena, je možné přihlašovací údaje lehce získat.

HTTP Digest ověření [24]

Ověřovací mechanizmus *Digest* je koncepčně podobný jako mechanizmus Basic, byl standardizován ve specifikaci HTTP/1.0. Samotný ověřovací mechanizmus je založen stejně jako Basic na schématu požadavek-odpověď s tím rozdílem, že heslo není přenášeno v nechráněné či nedostatečně chráněné podobě.

Jak už bylo řečeno, princip je založený na metodě Basic, prohlížeč zašle žádost pro vstup na zabezpečenou stránku s hlavičkou bez uživatelského hesla a jména. Server negativně odpoví s hlavičkou paketu obsahující stavový kód „*401 Unauthorized*“. Modifikovaná *Digest* hlavička oproti Basic hlavičce obsahuje v poli „WWW-*Authenticate*“ navíc pro nás zajímavou hodnotu „*nonce*“ a „*algorithm*“. Pomocí těchto dvou hodnot jsme schopni vytvořit z uživatelského jména a hesla „*hash*“ řetězec. Jedná se o jednosměrnou kryptografickou funkci, kterou lze snadno vypočítat jedním směrem, pro mechanizmus *Diges* se používá nejčastěji *hašovací* funkce MD5[8]. Jakmile prohlížeč

[7] Řetězec Basic je z bezpečnostních důvodů pozměněn

[8] V roce 2004 byly nalezeny v algoritmu MD5 dvě závažné chyby, proto se od použití postupně upouští http://www.security-portal.cz/clanky/md5-cracking.html

přijme tento packet, zobrazí uživateli přihlašovací formulář (obr. 6). Po zadání přihlašovacích údajů uživatelem je vypočítán *message digest* z těchto údajů a hodnoty nonce, algoritmus je určen serverem v poli *algorithm* a celý řetězec je poslán zpět s žádostí o přístup na zabezpečené stránky serveru.

Hodnota *nonce* zvyšuje odolnost kryptografické *hashovací* funkce proti dešifrování, protože existují databáze[9] obsahující *hashe* (otisky) nejrůznějších řetězců, kdy stačí námi odposlechnutý řetězec s touto databází porovnat a pokud existuje shoda dostaneme původní řetězec. Jako hodnota *nonce* může být použita IP adresa, MAC adresa serveru, časové razítko či náhodně generovaný alfa-numerický řetězec.

Po přijmutí *packetu* obsahující *message digest* serverem je stejným algoritmem jako u klienta vypočítán *hash* na straně serveru a porovnán s *hashem* od klienta. Pokud se shodují oba *hashe* je klientovi přístup povolen, v opačném případě je opět zaslán *packet* s hlavičkou obsahující stavový kód „*401 Unauthorized*".

Ačkoliv se jedná o poměrně zastaralé a nebezpečné metody, stále v praxi existuje mnoho webových aplikací využívající těchto metod. Správci webových aplikací si buď nechtějí připustit hrozící nebezpečí, nebo nechtějí investovat do aktualizace a vyššího zabezpečení například prostřednictvím SSL.

Certifikáty

Ověřovací mechanizmy využívající certifikáty byly vyvinuty z bezpečnostních důvodů. Mnoho uživatelů si neuvědomuje rizika spojená s nedostatečně robustním heslem – to by mělo mít alespoň 10 znaků a obsahovat kromě alfanumerických znaků také znaky speciální (tj. @, #, $, %, ^, &, * atd.). Proto vznikají dvě situace:

- Heslo je příliš krátké a triviální, avšak uživatel si ho pamatuje
- Heslo odpovídá bezpečnostním standardům, ale je tak složité na zapamatování, že si uživatel heslo poznamená na papír, do mobilu atd.

V prvním případě jsme schopni heslo prolomit pomocí dostupných nástrojů v reálném čase, v druhém případě si většina uživatelů heslo poznamená na papírek a připevní na monitor, poté stačí přijít k PC, přečíst si heslo a přihlásit se.

Aby k obdobným scénářům nedocházelo, byly vytvořeny ověřovací mechanizmy pomocí certifikátů využívající kryptografii s veřejným klíčem a digitálním certifikátem pro ověření identity uživatele (PKI).

[9] http://md5.rednoize.com/

Princip je následující: klient se osobně dostaví k vydavateli certifikátů, kde nejčastěji dostane kartu obsahující certifikát a čtečku karet. Po připojení čtečky karet k PC a připojení se k webové aplikaci je zkontrolován veřejný klíč, který si aplikace stáhne ze serveru nebo z čipové karty a ověří platnost u certifikační autority. Pokud je vše v pořádku, z dat, která mají být zaslána na server, se udělá *hash* a zašifrují se soukromým klíčem uloženým na čipové kartě. Naopak na straně serveru se dešifruje přijatý *hash* veřejným klíčem serveru a porovná se s *hashem* vytvořeným na straně serveru, souhlasí-li oba *hashe*, je identita klienta ověřena.

Nevýhodou tohoto mechanizmu je neustálá nutnost mít při sobě čipovou kartu se čtečkou, avšak pokud tento mechanizmus doplníme o ověření heslem, stává se tato metoda velice robustní a bezpečnou. V dnešní době si našla uplatnění především u internetového bankovnictví.

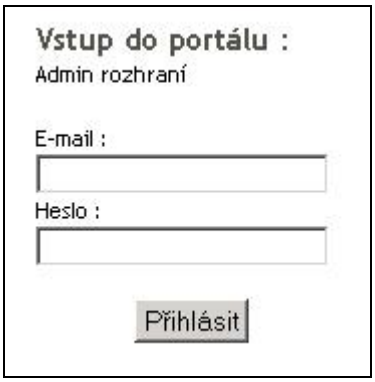
Formulářové ověření

Doposud jsme popsali ověřovací mechanizmy specifikované ve standardu HTTP či využívající protokol SSL, nyní se zaměříme na mechanizmus ověření identity prostřednictvím webového formuláře – mechanizmus je využit i v této práci.

Jedná se o přihlašovací mechanizmus vložený přímo do webové prezentace pomocí HTML tagů <FROM> a <INPUT>.

```
        <FORM ACTION="log_in.php" METHOD="post" NAME="formular">
E-mail :<INPUT TYPE="text" name="EMAIL">
Heslo  :<INPUT TYPE="password" name="PASS">
        <INPUT TYPE="submit" value="Přihlásit>
        </FROM>
```

Takto vytvořený formulář můžeme vidět na obr. 7.



Obr. 7: Webový formulář pro přihlášení

Poté co vyplníme formulář, jsou data z formuláře odeslána na server a tam podle naprogramované logiky zpracována. Komunikaci mezi serverem a klientem si nyní popíšeme podrobněji. Pro názorný popis mějme na serveru uloženy dva soubory: login.php a secret.php, webová stránka secret.php je zabezpečena a vstup na ní je pouze možný po přihlášení se přes stránku login.php. Pokud se budeme snažit zobrazit webovou stránku na adrese www.server.cz/secret.php, dostaneme od serveru negativní odpověď HTTP/1.1 302 Found

```
HTTP - Hyper Text Transfer Protocol
  HTTP Version:           HTTP/1.1
  HTTP Status:            302
  HTTP Reason:            Found
  Date:                   Sun, 15 Mar 2009 17:14:17 GMT
  Server:                 Apache/2.2.3 (Debian)
  Location:               http://www.server.cz/login.php
  Cache-Control:          max-age=0
  Expires:                Sun, 15 Mar 2009 17:14:17 GMT
  Content-Length:         275
```

a budeme přesměrováni na webovou stránku www.secret.cz/login.php s přihlašovacím formulářem. Po vyplnění formuláře přihlašovacími údaji, odešleme data z formuláře na server.

```
HTTP - Hyper Text Transfer Protocol
  HTTP Command:           POST
  URI:                    /login.php
  HTTP Version:           HTTP/1.1
  Host:                   www.server.cz
  User-Agent:             Mozilla/5.0 (Windows; U; Windows...
  Accept:                 text/html,application/xhtml+xml…
  Accept-Language:        cs,en-us;q=0.7,en;q=0.3
  Accept-Encoding:        gzip,deflace
  Accept-Charset:         UTF-8,*
  Keep-Alive:             300
  Connection:             keep-alive
  Referer:                http://server.cz/login.php
  Cookie:                 centposts=a%3A20%3A%7Bi%3A1538%3Bi%...
  Content-Type:           application/x-www-form-urlencoded
  Content-Length:         91
  Line  1:                log=user&pwd=heslo&...
```

Zde si všimněme, že data jsou na server přenášena metodou *POST* a také, pokud není implementováno SSL, jsou jméno a heslo přenášeny v otevřené formě jako text. Na serveru jsou přihlašovací údaje zkontrolovány a v případě shody je zpět prohlížeči zaslán paket s hlavičkou HTTP/1.1 200 OK a parametrem Location obsahující původní požadavek. V případě, že shoda nebyla nalezena, je zaslána hlavička HTTP/1.1 302 Found s přesměrováním buď na chybovou stránku nebo opět na přihlašovací formulář.

5.1.2 Uložení uživatelských hesel

Uživatelská hesla máme možnost uložit do prostého souboru či do databáze. Není důležité, kam hesla uložíme, ale spíše jak je zabezpečíme proti zneužití. Pokud vytváříme jednoduchou aplikaci s ověřením klienta, tak si vystačíme s uložením jak hesla, tak i uživatelského jména do souboru. Ovšem máme-li složitější webový projekt s návazností na různé moduly a přístupové úrovně, nevyhneme se využití databáze pro uložení jmen a hesel, jako je tomu i v této diplomové práci.

Pokud ponecháme hesla uložena v otevřené podobě, vystavujeme nejen hesla, ale také zabezpečení serveru a v neposlední řadě i klienty nebezpečí odcizení a zneužití hesel. Nyní se podívejme na důvody proč nepoužívat hesla v otevřené podobě:

1. Pokud pracujeme s hesly jako s prostým textem, lze jednoduše hesla odposlechnout během komunikace mezi prohlížečem a serverem. Tento příklad byl ukázán v předchozí podkapitole.
2. Většina klientů používá stejné heslo do více aplikací, to znamená pokud se útočník zmocní uživatelských jmen a hesel z našeho serveru, je ohrožen každý server, kde klient použil stejná hesla.
3. Útočník, který získal nezabezpečená hesla, se může již bez sebevětších problémů přihlásit do našeho systému, kde má vetší šanci způsobit nejenom finanční škody.
4. V rámci důvěryhodnosti našeho systému a serveru by neměl ani administrátor znát hesla klientů uložených v otevřené formě na serveru.

Abychom zabránili první hrozbě, použijeme pro komunikaci mezi webovým klientem a vzdáleným serverem šifrované spojení protokolem HTTPS. Tím zajistíme, že přenosový kanál bude šifrován a data nebudou přenášena v čitelné podobě..

Jakmile bezpečně přeneseme hesla na server, měli bychom z nich vytvořit *hash* a zpracovávat již pouze tyto otisky. To znamená ať už uložení hesel do databáze či souboru v podobě vytvořeného *hashe*, tak i následné porovnání při ověřování klienta při přístupu do aplikace. Jako *hashovací* funkci můžeme použít třeba již zmiňovaný algoritmus MD5 či SHA-1,2. Například použijeme-li algoritmus MD5 a vytvoříme *hash* ze slova „heslo“, dostaneme otisk „955db0b81ef1989b4a4dfeae8061a9a6“[10]. Bohužel i při použití *hashovací* funkce je možné z *hashe* získat původní heslo. Tyto techniky

[10] http://www.maxiorel.cz/md5-online-generator

používají dostupné tabulky[11] obsahující již vygenerované řetězce i s jejich otisky. Poté již stačí porovnat ukradený *hash* s příslušnou tabulkou a pokud se *hashe* shodují, útočník získal původní heslo. Aby k tomuto nedocházelo, je nutné volit dostatečně dlouhé a silné heslo či při *hashovaní* použít i tzv. „SALT“. Jedná se o řetězec (např. IP adresa serveru, uživatelské jméno atd.), kterým je doplněno původní heslo a až poté *zahashováno*. Tímto získáme dostatečně dlouhý a atypický řetězec, který útočníka odradí nebo alespoň zbrzdí při hledání původního hesla.

Stane-li se, že útočník pronikne do databáze a zmocní se jejího obsahu, není schopen v reálném čase při použití *hashovacích* funkcí získat čitelná hesla uživatelů a použít je při útoku na jiné servery.

5.2 Analýza případů užití

Nežli začneme programovat jednotlivé moduly našeho informačního systému, je vhodné se sejít s budoucími uživateli tohoto systému a poznamenat si jejich požadavky. Abychom si je nemuseli pamatovat a pracně později „lovit“ z paměti použijeme tzv. *případy užití*, jedná se o jednoduchou tabulku interakce mezi uživateli a budoucím systémem. Po sestavení případů užití (jednotlivých scénářů) je budoucí uživatel schopen posoudit, které požadavky by chtěl doplnit a naopak programátor má po schválení případů užití jasně definované hranice budovaného systému.

V prvním kroku je nutné si ujasnit, kolik bude aktérů (rolí) používající informační systém (v našem případě to budou čtyři aktéři – technik, administrátor, administrativa, klient), v dalším kroku navrhneme společně s jednotlivými aktéry scénáře případů užití a v posledním kroku předáme jednotlivé scénáře programátorům. Při navrhování jednotlivých scénářů musíme být důkladní, neboť v budoucím systému budou obsaženy pouze ty požadavky, které jsou poznamenány ve scénářích.

Nyní přejděme k jednotlivým scénářům, jak už bylo řečeno, jedná se o nejdůležitější část analýzy celého projektu. Nejdříve si popíšeme roli ***technik,*** který bude do našeho systému zadávat nejčastěji data (informace). Pro tuto roli byly sestaveny tři scénáře a to:

[11] http://ophcrack.sourceforge.net/

Založení nového klienta s novou přípojkou

1. Technik Přihlásí se do informačního systému
2. Systém Zobrazí aktivní klienty
3. Technik Zvolí nabídku nový zákazník
4. Systém Zobrazí prázdný formulář pro zadání iniciálu zákazníka
5. Technik Vyplní iniciály a odešle formulář
6. Systém Založí nového zákazníka a zobrazí formulář pro registraci připojení
7. Technik Vyplní údaje (IP adresa, MAC adresa …)
8. Systém Založí záznam o nové přípojce a vygeneruje nastavení do AP

Registrace nové přípojky stávajícímu klientovi

1. Technik Přihlášení do informačního systému
2. Systém Zobrazí aktivní klienty
3. Technik Vyhledá požadovaného klienta
4. Systém Zobrazí klienta
5. Technik Zvolí nové připojení u stávajícího klienta
6. Systém Zobrazí prázdný formulář pro registraci připojení
7. Technik Vyplní údaje (IP adresa, MAC adresa …)
8. Systém Založí záznam o nové přípojce a vygeneruje nastavení do AP

Evidence nového zařízení

1. Technik Přihlášení do informačního systému
2. Systém Zobrazí aktivní klienty
3. Technik Zvolí nabídku přidat nové zařízení (Access point, server….)
4. Systém Zobrazí prázdný formulář pro zaevidování zařízení
5. Technik Vyplní požadované údaje
6. Systém Zavede zařízení do emailu

Mohlo by se zdát, že uvedené scénáře pro roli technik jsou nedostatečné a daly by se doplnit o další, avšak my to neuděláme z důvodu popsaného dále.

Druhou nepostradatelnou rolí je role ***administrativa***, jejímž hlavním úkolem je dohled nad řádným placením za poskytované služby, tisk smluv a v neposlední řadě

také změna kontaktních údajů klientů. Stejně jako pro roli technik i u role administrativa máme tři scénáře:

Editace kontaktních údajů

1. Administrativa Přihlášení do informačního systému
2. Systém Zobrazí aktivní klienty
3. Administrativa Vyhledá požadovaného klienta
4. Systém Zobrazí klienta
5. Administrativa Vyedituje kontaktní údaje klienta
6. Systém Zobrazí formulář pro změnu údajů
7. Administrativa Opraví kontaktní údaje a odešle formulář
8. Systém Uloží změněné údaje

Změna parametrů sjednané služby

1. Administrativa Přihlášení do informačního systému
2. Systém Zobrazí aktivní klienty
3. Administrativa Vyhledá požadovaného klienta
4. Systém Zobrazí klienta
5. Administrativa Zvolí nabídku pro změnu stávající služby připojení
6. Systém Zobrazí formulář pro změnu služby
7. Administrativa Změní službu a odešle
8. Systém Uloží změněné údaje a vygeneruje nové nastavené do AP

Vytvoření administrativních položek

1. Administrativa Přihlášení do informačního systému
2. Systém Zobrazí aktivní klienty
3. Administrativa Zvolí nabídku přidat nové administrativní položky
4. Systém Zobrazí požadovaný formulář
5. Administrativa Vyplní formulář a odešle
6. Systém Zaeviduje novou položku

Jelikož by se některé scénáře u více rolí shodovaly, například založení klienta, je vytvořena imaginární role ***zaměstnanec***, která tyto společné scénáře pro role technik a administrativa zastupuje. Těmito scénáři jsou:

Založení nového klienta

1. Zaměstnanec Přihlášení do informačního systému
2. Systém Zobrazí aktivní klienty
3. Zaměstnanec Zvolí nabídku pro založení nového klienta
4. Systém Zobrazí formulář pro nového klienta
5. Zaměstnanec Vyplní formulář potřebnými údaji a odešle
6. Systém Založí nového zákazníka do systému

Změna stavu klienta

1. Zaměstnanec Přihlášení do informačního systému
2. Systém Zobrazí všechny klienty
3. Zaměstnanec Vybere požadovaného klienta
4. Systém Zobrazí detail klienta
5. Zaměstnanec Povolí nebo zablokuje klienta (např. když nezaplatil…)
6. Systém Uloží změnu stavu klienta a vygeneruje nové nastavení do AP

Stejně jako je ve firmách hierarchie nadřízených a podřízených, i my vytvoříme scénář pro roli ***administrátor***, která bude nadřazená nad rolemi technik a administrativa. Bude mí stejná oprávnění a navíc bude spravovat zaměstnanecké účty a jejich práva v informačním systému.

Správa zaměstnaneckých účtů

1. Administrátor Přihlášení do informačního systému
2. Systém Zobrazí aktivní klienty
3. Administrátor Zvolí položku nový zaměstnanec
4. Systém Zobrazí formulář pro nového zaměstnance
5. Administrátor Vyplní kontaktní údaje a přiřadí zaměstnanci roli

   POKUD role neexistuje, vytvoří ji

   1. Systém Zobrazí formulář pro novou roli
   2. Administrátor Vyplní požadovaná práva
   3. Systém Uloží novou roli

   JINAK administrátor vybere požadovanou roli

6. Systém Uloží nového zaměstnance a odešle informativní e-mail

Na závěr nesmíme zapomenou na uživatelskou roli ***klient***, samotný klient by měl mít možnost si zkontrolovat parametry nesmlouvané služby, případně změnit kontaktní údaje nebo zjistit například množství přenesených dat. Proto vytvoříme scénář i pro roli klient, aby mohli programátoři realizovat klientský modul:

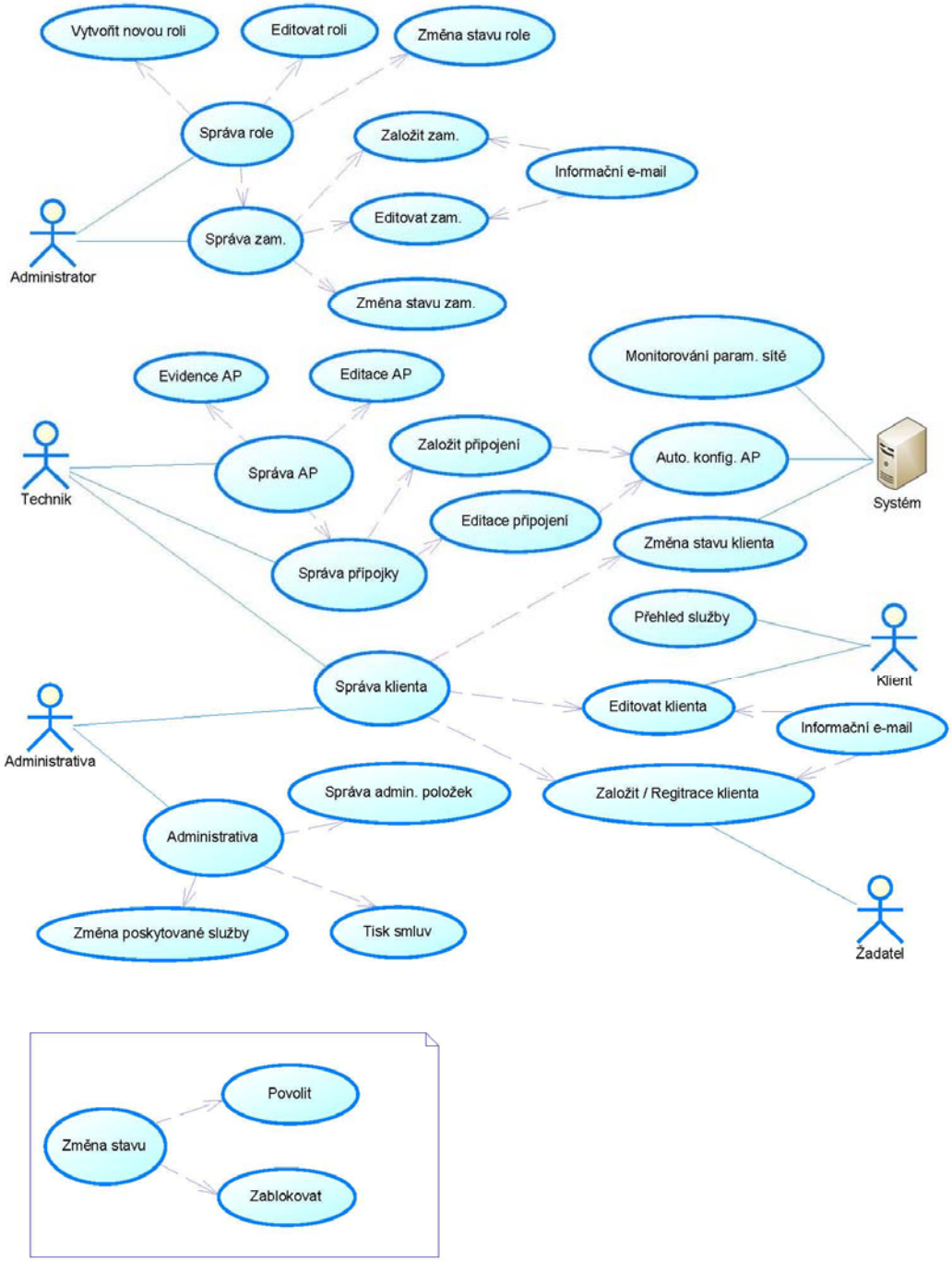
Uživatelský přehled

1.	Klient	POKUD klient nemá registraci, zaregistruje
		JINAK klient se přihlásí do informačního systému
2.	Systém	Zobrazí kontaktní údaje o přihlášeném klientovi
3.	Klient	Zvolí položku editace kontaktních údajů
4.	Systém	Zobrazí formulář pro změnu kontaktních údajů
5.	Klient	Provede vyplnění formuláře aktuálními údaji
6.	Systém	Uloží změnu a vygeneruje nové nastavení do AP

1.	Klient	POKUD klient nemá registraci, zaregistruje
		JINAK klient se přihlásí do informačního systému
2.	Systém	Zobrazí kontaktní údaje o přihlášeném klientovi
3.	Klient	Zvolí položku zobrazit statistiku odebírané služby
		(velikost přenesených dat, kvalita signálu)
4.	Systém	Zobrazí okno s požadovanými grafy

Kromě samotných případů užití v tabulkové formě, lze popsat případy užití pomocí diagramů. Ačkoliv psaná forma má největší užitnou hodnotu, grafické znázornění, jak ho můžeme vidět na obr. 8, činí celou analýzu přehlednější.

Z obrázku je také patrné, že aktérem nemusí být pouze uživatel, ale také samotný systém, který vykonává automaticky nadefinované úkoly administrátorem serveru / systému.



Obr. 8: Případy užití

5.3 Analýza návrhu databáze, E-R diagram

Srdcem každého informačního systému je databáze a databázová struktura, v tomto projektu tomu nebude jinak. Před samotným návrhem struktury pro uložení dat si probereme potřebnou teoretickou část s praktickými příklady, bez které bychom nebyli schopni kvalitně a optimálně databázovou strukturu sestavit.

5.3.1 Primární klíč, cizí klíč, index, normalizační pravidla a vztahy mezi relacemi

Než začneme navrhovat databázovou strukturu pro naši aplikaci, je nutné si objasnit některé pojmy a pravidla související s tímto návrhem. Vysvětlíme si pojmy jako je primární a cizí klíč, index, jaké jsou vztahy mezi relacemi a co je to normalizace a k čemu je dobré znát tzv. normální formy.

Primární klíč, cizí klíč, index

Každá tabulka obsahuje *primární klíč*, jehož úkolem je urychlit vyhledávání příslušného záznamu v tabulce. Primárním klíčem může být číselná hodnota, nebo může být složen z hodnot z více sloupců tabulky. Při definování primárního klíče bychom se měli držet těchto doporučení:

1. Primární klíč musí být unikátní, nesmí se stát, že sloupeček s primárním klíčem bude obsahovat dvě stejné hodnoty.
2. Nejčastěji se jako primární klíč používají 32bitová celá čísla. Pokud navíc použijeme vlastnost sloupce „autoincrement“, nemusíme si dělat starosti s unikátností, čísla jsou generována automaticky vzestupně.
3. Primární klíče by měly být kompaktní, jednak z toho důvodu, že fungují i jako indexy, a také proto, že primární klíče mohou sloužit jako cizí klíče v jiných tabulkách.

Jestliže chceme tabulky mezi sebou propojit, použijeme primární a *cizí klíč*. Co je primární klíč už víme, úkolem cizího klíče je pak odkazovat na záznamy v hlavní tabulce. Můžeme tedy říci, že cizí klíč v tabulce B je primárním klíčem v hlavní tabulce A. Cizí klíč už nemůže být definován pomocí funkce databáze „autoincrement“, ale je definován hodnotou, na kterou cizí klíč ukazuje v hlavní tabulce.

Cizí klíč je také spojen s referenční integritou, neboli omezení cizího klíče. Nesmí se stát, že smažeme záznam v tabulce A a v tabulce B nám zůstane záznam závislý na smazaném záznamu v tabulce A; potom říkáme, že byla porušena referenční integrita databáze. Abychom nebyli závislí na tom, jak programátor vyřeší tento problém v samotné aplikaci, hlídá si většina databázových systémů referenční integritu sama. Při definici cizího klíče definujeme, co se má stát, pokud by byla porušena integrita:

- operace se neprovede, je zobrazeno chybové hlášení
- jsou ovlivněny všechny související záznamy tj. je smazán záznam v tabulce A a také záznam v tabulce B

Úkolem *indexů* je urychlit vyhledávání a ukládání dat v rozsáhlé struktuře databáze. Ke stávajícím tabulkám s uloženými daty je vytvořena další tabulka pro uložení hodnoty indexu a odkazu na záznam v odpovídající tabulce. Jakmile uložíme nový záznam do tabulky, je vytvořen index s odkazem na daný záznam. Při vyhledávání projdeme tabulku indexů a pomocí odkazu najdeme požadovaný záznam. Databázové systémy používají pro indexy nejčastěji stromové struktury.

Normalizační pravidla

Normalizace neboli zjednodušování a optimalizace databázových struktur vede k odstranění redundancí. Správně navržená databázová struktura by měla splňovat tzv. normální formy a právě tyto formy si popíšeme v následujících podkapitolách.

První normální forma

Aby tabulka mohla být zařazena do první normální formy musí splnit několik kriterií:

1. Tabulka nesmí mít sloupce se stejným obsahem. Jak můžeme vidět v názorné tabulce (tab. 1), tuto podmínku splňujeme.
2. V jednom sloupci nemůže být obsaženo více druhů údajů, například jméno a ulice.
3. Každý záznam musí být jednoznačně identifikovatelný.

Tab. 1: První normální forma

ID	Jméno	Ulice	Město	IP adresa	MAC adresa
1	Jan Novák	Orlí 5	Brno	192.168.1.100	b4:d3….
2	Jana Krátká	Úvoz 59	Brno	192.168.1.120	c3:e7….

Už při předběžném návrhu bychom měli myslet v první normální formě, ulehčíme si tím pozdější práci.

Druhá normální forma

Jak můžeme vidět v tabulce 1, některé hodnoty ve sloupcích se nám opakují, tento problém řeší druhá normální forma, která říká:

- Opakují-li se v některých sloupcích v různých řádcích data, je nutné tuto tabulku rozdělit a propojit cizím klíčem.

Abychom tuto podmínku splnili, rozdělíme naší tabulku na dvě části, jedna bude obsahovat kontaktní údaje klienta a jeho IP a MAC adresy, druhá bude obsahovat názvy měst. Na závěr obě tabulky propojíme pomocí jejich klíčů.

Tab. 2a: Druhá normální forma

ID	Jméno	Ulice	ID mesto	IP adresa	MAC adresa
1	Jan Novák	Orlí 5	1	192.168.1.100	b4:d3….
2	Jana Krátká	Úvoz 59	1	192.168.1.120	c3:e7….

5.3.2

Tab. 2b: Druhá normální forma

ID mesto	Město
1	Brno

Podle definice druhé normální formy jsme propojili *hlavní tabulku* s tabulkou *mesto* pomocí primárního klíče *ID mesto*.

Třetí normální forma

Abychom splnili třetí normální formu, musíme z tabulky odstranit všechny sloupečky, které nejsou přímo závislé na primárním klíči, v tomto případě se jedná o IP

a MAC adresu. Jakmile provedeme toto rozdělení, všimneme si, že na počátku jsme měli pouze jednu tabulku a rázem máme tabulky čtyři.

Tab. 3a: Třetí normální forma

ID klient	Jméno	Ulice	ID mesto
1	Jan Novák	Orlí 5	1
2	Jana Krátká	Úvoz 59	1

ID mesto	Město
1	Brno

Tab. 3b: Třetí normální forma

ID adresa	IP adresa	MAC adresa
1	192.168.1.100	b4:d3….
2	192.168.1.101	b1:a3….
3	192.168.1.120	c3:e7….

ID klient	ID adresa
1	1
1	2
2	3

Jakmile přidáme další záznam již stávajícímu klientovi, nezatížíme tím tabulku klientů opakujícími se kontaktními údaji. A naopak pokud klient zruší smlouvu, my pouze vymažeme jeho IP a MAC adresu a v databázi nám zůstanou jeho kontaktní údaje pro možné budoucí zpracování.

Čtvrtá normální forma

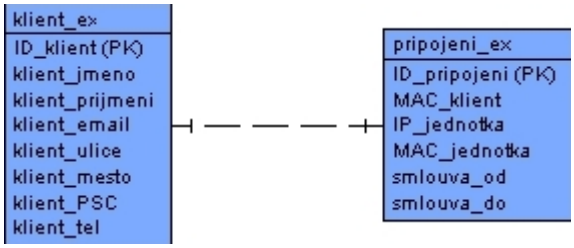
Pro úplnost ještě dodám, že je známá i čtvrtá normální forma, která popisuje příčinnou souvislost, ovšem při návrhu se zohledňuje jen zřídka.

Vztahy mezi relacemi

Abychom mohli provést v předchozím kroku normalizaci tabulek, jsou potřeba znalosti o možnostech propojení mezi tabulkami.

Vztahy 1:1

Prvním a nejjednodušším je vztah jeden-na-jednoho, kde každému řádku první tabulky existuje záznam i ve druhé tabulce. Tento vztah není příliš častý, protože bychom mohli obě tabulky sloučit do jedné.

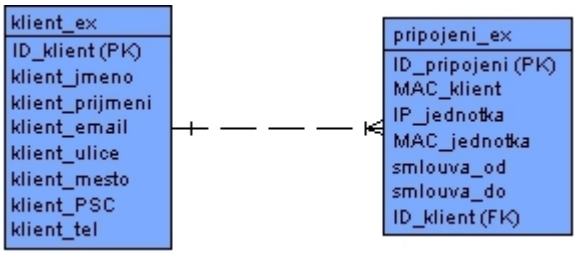


Obr. 9: Relace 1:1

Význam tohoto vztahu lze vidět pouze pokud bychom chtěli každou tabulku jinak zabezpečit (přiřadit různá oprávnění), nebo pokud by jako celek měla mnoho sloupců, z nichž používáme jen malou část pro zpracování.

Vztahy 1:n

Druhým a nejčastěji využívaným vztahem je jedna-na-mnoho a na tomto vztahu je postavená i databázová struktura našeho projektu. Vztahem 1:n se rozumí, že ke každému záznamu z první tabulky odpovídá jeden nebo více záznamů v druhé tabulce. V literatuře se můžeme dočíst i o vztahu n:1, ale jedná se stále o vztah 1:n akorát z opačného pohledu.

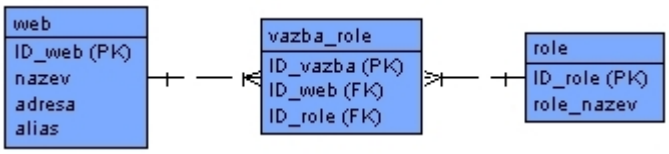


Obr. 10: Relace 1:n

V našem případě může mít klient jedno připojení, ale také dvě a více jak ukazuje obr. 9.

Vztahy n:m

Tento vztah v databázových schématech nenajdeme, jelikož se jedná o natolik komplikovaný vztah, že musí být rozdělen na několik vztahů typu 1:n, jak můžeme vidět i na obr. 11. Vztah mnoho-na-mnoho znamená, že jednomu záznamu v hlavní tabulce odpovídá několik záznamů v druhé a naopak.



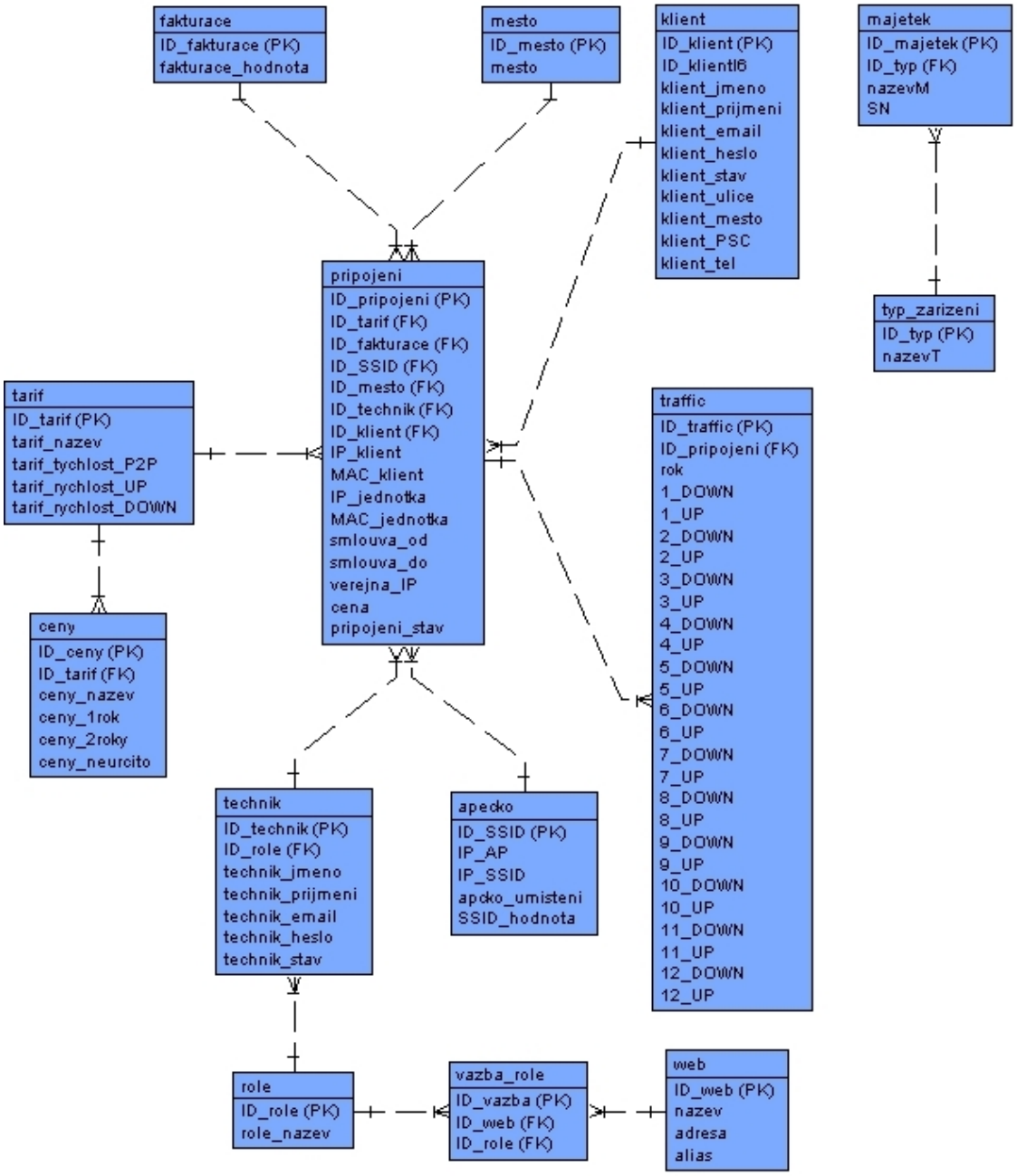
Obr. 11: Relace n:m

Vztahy jsou realizovány prostřednictvím sloupců v tabulkách obsahujících primární a cizí klíče. Primární klíč je obsažen v hlavní tabulce a cizí klíč v dílčí tabulce, pomocí cizího klíče (FK) se okazujeme do hlavní tabulky na záznam obsahující odpovídající primární klíč (PK).

Vztah mezi primárním klíčem a cizím klíčem nemusí existovat pouze mezi tabulkami, ale i v rámci jedné tabulky, kdy se odkazujeme pomocí cizího klíče na další záznam v téže tabulce.

5.3.3 Návrh E-R diagramu

Jakmile jsme se dostatečně seznámili s předchozími normalizačními pravidly, vztahy a atributy, jsme schopni navrhnout E-R diagram pro naší aplikaci. Na obrázku 12 můžeme vidět výsledný návrh obsahující 15 tabulek navržený ve třetí normální formě.



Obr. 12: Entity-relationship diagram (ERD)

Nežli z E-R diagramu vygenerujeme SQL skript pro import do MySQL databáze, měli bychom si celý návrh ještě jednou zkontrolovat, jednak podle případů užití a jednak prostřednictvím normálních forem. Případné chyby budou mít vliv na efektivitu programování a údržbu. Chybný návrh by mohl mít katastrofální následky, pokud by byly chyby odhaleny až před dokončením aplikace.

Popis jednotlivých tabulek, atributů a vztahů v tomto případě postrádá smysl, veškeré pojmy jsme si vysvětlili na praktických ukázkách z našeho projektu v předchozí podkapitole. Všechny položky v E-R diagramu jsou jednoznačně a jasně pojmenovány, mají o sobě vypovídající hodnotu. Proto můžeme vygenerovat SQL dotaz a celou strukturu vložit do připravené databáze.

5.4 Realizace informačního systému

Jak už bylo napsáno v teoretické části práce, pro chod celého systému je nutné mít na serveru nainstalovaný webový server Apache 2.0 s rozšiřujícími moduly mod_php5, mod_ssl, mod_rewrite a mod_snmp. Dále samozřejmě skriptovací jazyk PHP5, ve kterém je celá aplikace napsána, a databázový systém MySQL. Všechny tři aplikace jsou multiplatformní, tudíž lze použít jak server na bázi UNIXu, tak i Windows server.

Nebudu zde popisovat kousek po kousku zdrojového kódu, ale zaměřím se na popis bezpečnostních prvků realizovaných v tomto IS, popíši základní typy obrazovek (formulářů), které by měl každý systém obsahovat a na závěr detailněji rozeberu dva typy komunikace mezi webovým serverem a routerboardem.

5.4.1 Bezpečnost

Jeden z největších důrazů byl kladen na bezpečnost, přeci jenom se jedná o citlivá data uživatelů a také bychom nechtěli narušení systému cizí osobou či pouze špatně zadaným nebo modifikovaným dotazem do databáze.

První bezpečnostní hrozbou je neustálá komunikace mezi webovým prohlížečem a serverem přes otevřený kanál, kudy jsou přenášena veškerá data. Aby útočník při odposlouchávání komunikačního kanálu nemohl tato data analyzovat, je celá komunikace vedena šifrovaně za pomocí protokolu HTTPS. V příloze č. 1 je popsán návod jak nastavit HTTP server, aby komunikoval s webovým prohlížečem přes HTTPS.

Pro přihlášení do informačního systému je použit webový přihlašovací formulář pro zadání uživatelského jména a hesla. Právě hesla jsou nejvíce střežena – jak už bylo

řečeno, většina uživatelů používá stejné heslo ve většině aplikací. Proto nejsou hesla v databázi uložena v textové podobě, ale jsou z nich vytvořeny hashe (otisky) a ty poté uloženy. Pro hashovaní se používají jednoduché jednosměrné hashovací funkce MD5 a SHA-1, ačkoliv jsou již známé metody pro prolomení těchto funkci pomocí tabulkových útoků. Opakovanou kombinací těchto dvou algoritmů docílíme dostatečně silného „zahashování“ uživatelských hesel.

Při realizaci byl využit i skriptovací jazyk JavaScript, který oznamuje uživateli systémová oznámení, kontroluje vyplnění formulářů atd. Z toho vyplývá, že zákaz JavaScriptu ve webovém prohlížeči by mohl mít katastrofální následky pro celý IS systém. Proto uživatel není schopen se přihlásit bez povoleného JavaScriptu v prohlížeči.

Protože se jedná o informační systém, je samozřejmostí použití formulářů pro vkládání požadavků a dat pro zpracování webovým serverem, např. přihlašovací formulář. Aby bylo zabráněno jak úmyslným tak náhodným útokům na náš systém prostřednictvím těchto formulářů, jsou tyto formuláře 2x kontrolovány:

1. Jakmile uživatel zadá požadované informace do formuláře, jsou jednotlivá pole zkontrolována pomocí JavaScriptových funkcí na správný formát zadaných dat a hlavně na obsah nebezpečných znaků. V případě, že by uživatel měl ve svém webovém prohlížeči JavaScript vypnut, nebude mu umožněno formulář odeslat
2. V druhém kroku jsou vstupní data od uživatele „očištena“ od nebezpečných znaků prostřednictvím PHP funkci addslashes() [12] , strip_tags()[13] a htmlspecialchars()[14]

Nedílnou součástí všech informačních a jiných systémů nejen webových jsou uživatelé a hlavně uživatelské role. Každému uživateli bychom měli přidělit v systému jen takové pravomoci, které opravdu pro svojí činnost potřebuje, a náš systém není výjimkou. Po přihlášení je uživateli sestaveno navigační menu s položkami, do kterých má oprávnění vstoupit. Pokusí-li se dostat do sekce, například pozměněním URL, kam nemá oprávněný přístup, je automaticky odhlášen z celého systému.

Dalším možným zabezpečením je omezení přístupu k informačnímu systému pouze z určitých IP adres. Toto řešení je vhodné pro použití v intranetových aplikacích či místních lokálních sítích. Nemáme důvod zpřístupňovat náš informační systém

[12] http://cz2.php.net/addslashes
[13] http://cz2.php.net/manual/en/function.strip-tags.php
[14] http://cz2.php.net/manual/en/function.htmlspecialchars.php

z internetu, když naši uživatelé (klienti, zaměstnanci) jsou připojení přes naši bezdrátovou síť, tudíž přístup je omezen pouze na IP adresy z našeho rozsahu. V případě, že chceme mít dohled nad našim systémem i z jiné sítě, použijeme například VPN připojení.

5.4.2 Základní typy formulářů

Jakmile máme navrženou strukturu databáze a připraveny scénáře užití, můžeme se pustit do implementace samotného informačního systému. Při návrhu jednotlivých modulů vycházíme z pěti základních typů formulářů (obrazovek), které spolu logicky souvisí. Jsou to:

Record List

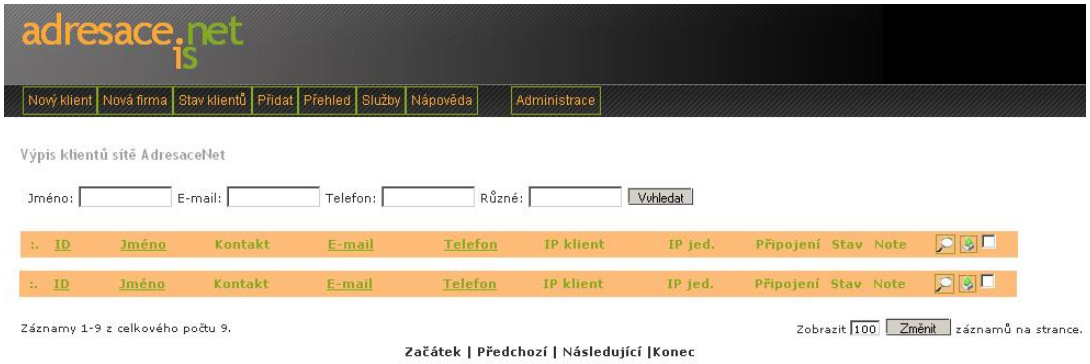
Nejzákladnějším typem formuláře je record list, jedná se zobrazení dat v řádkovém režimu bez možnosti vkládat či upravovat zobrazené informace. Z record listu je možné se dále navigovat pomocí odkazů na další typy formulářů.

:.	ID	Jméno	Kontakt	E-mail	Telefon	IP klient	IP jed.	Připojení	Stav	Note
	1	Petr Hromádko		modruss@modruss.eu	789456987	192.168.11.10	192.168.11.1	1	?	
	2	Josef Hlaváč		josef@centrum.cz	777 199 946	192.168.200.231	192.168.205.131	1	✓	
	3	Zdeněk Kusala		tomas@centrum.cz	732 936 184	192.168.70.245	192.168.75.133	1	✓	
	4	Radek Dostalík		radek@seznam.cz	776 078 742	192.168.40.191	192.168.45.135	1	✓	
	5	Petra Klímková		petra@centrum.cz	602 883 946	192.168.70.175	192.168.75.223	1	✓	
	6	Milan Lišaník		milan@centrum.cz	772 742 569	192.168.10.189	192.168.15.138	1	✓	
	7	Martin Žemžula		martin@volny.cz	774 589 145	192.168.90.8	192.168.95.197	1	✓	
	9	Pavel Fránek		pavel@seznam.cz	609 547 258	192.168.200.232	192.168.205.146	1	✓	
	10	Svatopluk Hudeček		svatopluk@seznam.cz	606 154 874	192.168.140.4	192.168.145.10	1	✓	
:.	ID	Jméno	Kontakt	E-mail	Telefon	IP klient	IP jed.	Připojení	Stav	Note

Obr. 13: Record list

Query Form

Před samotným zobrazením informací v record listu bychom si měli být nejdříve schopni určit, podle jakých kriterií chceme data zobrazit. K tomuto účelu slouží formulář query form, do kterého můžeme zadat, kolik záznamů chceme zobrazit, vyfiltrovat podle určitých pravidel atd.



Obr. 14: Query form

Record list a query form se někdy spojují do jednoho celku, kdy si nejdříve necháme zobrazit všechny záznamy a až posléze nadefinujeme pravidla pro odfiltrování nežádoucích záznamů.

View Form

Pro prohlížení dat v rozšířeném režimu se používá view form, který zobrazí kompletní informace o daném záznamu. Tento formulář může zobrazit buď detail jednoho záznamu (obr. 15), nebo rozšířené informace více záznamů ve více řádcích.

Obr. 15: View form

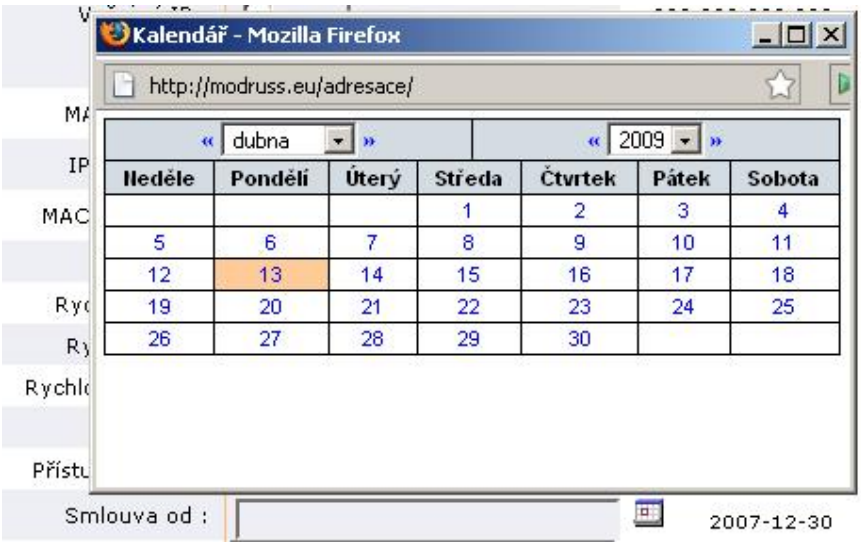
Insert Form

Jelikož nechceme data pouze zobrazovat, ale také vkládat, musíme definovat formulář pro vkládání, tzv. insert form. Opět jako v předchozím případě se může jednat o vkládání jednoho záznamu na jedné obrazovce (obr. 16), nebo lze vkládat na jedné obrazovce více záznamů.

Obr. 16: Insert form

LOV (seznamy hodnot)

Speciálním typem formuláře jsou tzv. seznamy hodnot (obr. 17), určené pro vkládání povolených hodnot pouze z těchto formulářů. Typickým příkladem LOV jsou kalendáře.



Obr. 17: List of values

5.4.3 Struktura rozhraní

Jelikož se jedná o interaktivní webový informační systém, kdy uživatel zadává pokyny a server vrací odpověď, je zde důležitá funkční logika. Nebudu zde popisovat jednotlivé stránky a v nich umístěné PHP skripty, ale vyberu z nich jeden ukázkový, na kterém popíši systém tvorby jednotlivých modulů.

Jak můžeme vidět na obrázkových ukázkách v předchozí kapitole, nejedná se o statické webové stránky, ale uživatel si volí dle požadavků, co mu má server zobrazit. Abychom vytvořili takto interaktivní informační systém, je nutné do HTML kódů přidat PHP skripty. Na následující ukázce je vytvořen zdrojový kód pro zobrazení všech zaměstnanců používající náš IS. Hned zpočátku je zinicializovaná třída *technik()*, která obsahuje funkce pro načtení zaměstnanců z databáze. Tyto hodnoty z DB si vzápětí uložíme do proměnné pro pozdější zpracování a následuje HTML kód pro definici hlavičky a samotného těla dokumentu.

```
<?php
//-- inicializace tridy
$survey = new technik();
//-- nacteni dat z databaze do tabulky technici
$pocetTechnik = $survey->pocetTech();
$resultTechnik = $survey->resultTech();
?>

<HTML>
<HEAD>…
</HEAD>
<BODY>
<fieldset id="tab2">
 <legend>Technik</legend>
 <br>
  <TABLE cellpadding=4 cellspacing=0 border=0>
    <TR BGCOLOR=#FFBC79>
        <TD>:.</TD>
        <TH>Jméno</TH>
       <TH>Příjmení</TH>
       <TH>E-mail</TH>
       <TH>Telefon</TH>
       <TD> .: </TD>
    </TR>
      <?php
//-- kolik je techniku, tolik bude radku
            for($i=0; $i<$pocetTechnik; $i++){
//-- vypis jednotlivých radku tabulky
```

Samotné vykreslení řádků již obstarává PHP skript, který do HTML kódu vkládá údaje získané z databáze.

```
                print "<TR>\n";
                print "<TD><IMG SRC=\"images/person.gif\"
                      WIDTH=\"16\" HEIGHT=\"16\" BORDER=0 ALT=\"\">
                        </TD>\n";
                print "<TD>".mysql_Result($resultTech, $i,
                      "technik_jmeno")."</TD>\n";
                print "<TD>".mysql_Result($resultTech, $i,
                       "technik_prijmeni").
                       "</TD>\n";
                print "<TD><A HREF='mailto:" .
                     mysql_Result($resultTech,$i,
                     "technik_email")."'>" .
                     mysql_Result($resultTech,$i, "technik_email").
                        "</TD>\n";
                print "<TD>".mysql_Result($resultTech, $i,
                        "technik_telefon"). "</TD>\n";
                 print "<TD nowrap ALIGN=CENTER><IMG
                       SRC=\"images/person.gif\" WIDTH=\"16\"
                       HEIGHT=\"16\" BORDER=0 ALT=\"\"></TD>\n";
                print "</TR>\n";
               }
        ?>
  <TR BGCOLOR=#FFBC79>
        <TD>:.</TD>
        <TH>Jméno</TH>
      <TH>Příjmení</TH>
      <TH>E-mail</TH>
      <TH>Stav</TH>
      <TD> .: </TD>
  </TR>
</TABLE>
</fieldset>

</BODY>
</HTML>
```

Pro každou část informačního systému je v adresáři *./class* vytvořena třída, která obsahuje funkční logiku dané části. Pro naši ukázku je vybrána třída obsluhující funkce pro práci se zaměstnanci obsluhující IS. Jak si můžeme všimnout, třída obsahuje dva druhy funkcí, *private* - která obsluhuje požadavky na databázi a *public* - která zpřístupňuje privátní funkci ostatním objektům kódu PHP.

```
<?php
class technik() {
      private function Tech() {
            $resultTech = mysql_query(
                        "SELECT  ID_technik, technik_prijmeni,
                                 technik_jmeno, technik_email,
                                 technik_telefon, technik_stav,
                                 role_nazev
                          FROM   technik, role
                          WHERE  technik.ID_roleR = role.ID_role");
            if (!$resultTech)
            die('Interni chyba');
            return $resultTech;
      }
```

```
        public function resultTechnik() {
                return $this->Tech();
        }

        public function pocetTechnik() {
                return mysql_num_rows($this->Tech());
        }
}
?>
```

Stejným způsobem, jakým byl vytvořen tento příklad pro výpis zaměstnanců, je tvořen celý navržený informační systém.

5.4.4 Skript pro vzdálenou konfiguraci RouterBoardu HTTP serverem

Abychom nemuseli při změně osobních údajů, změně rychlosti připojení nebo odpojení klienta překonfigurovávat routerboard (RB) a zároveň tyto změny aktualizovat i v informačním systému, můžeme propojit HTTP server s RB. Toto řešení nám nejenom usnadní práci, ale tyto úkony může provádět například i méně kvalifikovaný člověk, kterému umožníme pouze omezený vstup do IS, aby se nedostal přímo ke konfiguraci RB.

Nejdříve si ukážeme a popíšeme terminálové příkazy OS MikroTik, kterými se definují klienti, a pravidla, která se na ně aplikují. RouterBoard s OS MikroTik (ROS) zajišťuje mnoho služeb, kromě základního jako je dynamické routovaní či DHCP server také zajišťuje kvalitu služeb QoS, Proxy server, VPN, Pokročilý paketový firewall či FTP server a další. Výstavba počítačové sítě na aktivních prvcích RouterBoard a jejich konfigurace není obsahem této práce, proto se budeme soustředit pouze na položky námi využívané. Jedná se především o bezdrátové rozhraní (Wireless), firewall a řízení toku dat (Queue Tree).

Nejdříve je potřeba přidat bezdrátovou jednotku, respektive MAC adresu klienta do access listu. Jedná se o seznam, který určuje, kdo se může připojit k přístupovému bodu. Pokud bychom tento krok neprovedli, klient, respektive klientská jednotka by při žádosti o připojení k přístupovému bodu byla odmítnuta.

```
interface wireless access-list add mac-address=00:4F:62:0F:22:A4
interface=wifi.net comment="Klient Ulice CP"
```

Jedná se o standardní zabezpečení proti neoprávněnému připojení, které je většinou doplněné mechanizmem WPA2.

Jakmile je klient přidaný v access listu, je nutné ho ještě zapsat do address listu ve firewallu. Jedná se o seznam připojených klientů a jejich IP adresy, kde klientovi přidáme příznak, např. zda-li se jedná o platícího klienta, který se neprovinil proti pravidlům sítě, nebo zda bude mít příznak např. „neplatici“ a budou jeho požadavky přesměrovány na virtuální doménu apache serveru, jak už bylo popsáno dříve.

```
ip firewall address-list add list=klient address=192.168.140.20
disabled=no comment="Klient Ulice CP"
```

Abychom mohli aplikovat řízení datových toků (Queue Tree) je nutné nejdříve tzv. „označkovat„ průchozí pakety routerem. Protože máme překladovou síť neboli „NAT“, je nutné pro označkování použít trojici pravidel. Nejdříve označkujeme samotné spojení z dané IP adresy do vnější sítě.

```
ip firewall mangle add chain=Down action=mark-connection new-
connection-mark=192.168.140.19_Vse_con passthrough=yes src-
address=192.168.140.20
```

S takto označkovaným spojením můžeme dále pracovat, k tomu využijeme tzv. „mark-packet“, kdy už přímo rozlišujeme procházející packety označkovaným spojením. Díky tomu můžeme rozeznávat, zda-li se jedná o příchozí nebo odchozí provoz, či o jakou komunikaci (na jakém protokolu) se jedná podle zdrojových, respektive cílových portů.

```
ip firewall mangle add chain=Up src-address=192.168.140.19 in-
interface=vnitrni action=mark-packet new-packet-
mark=192.168.140.20_Vse_Up passthrough=no
```

Action	Src. Address	In. Inter...	D...	Protocol	New Packet Mark	New Connection Mark
mark packet	192.168.140.20	vnitrni			192.168.140.20_P2P_Up	
mark packet	192.168.140.20	vnitrni	80	6 (tcp)	192.168.140.20_HTTP_Up	
mark packet	192.168.140.20	vnitrni		1 (icmp)	192.168.140.20_ICMP_Up	
mark packet	192.168.140.20	vnitrni			192.168.140.20_Vse_Up	
mark connection	192.168.140.20					192.168.140.20_Vse_con
mark connection	192.168.140.20					192.168.140.20_P2P_con
mark connection	192.168.140.20		80	6 (tcp)		192.168.140.20_HTTP_con
mark connection	192.168.140.20			1 (icmp)		192.168.140.20_ICMP_con
mark packet		vnejsi			192.168.140.20_Vse_Down	
mark packet		vnejsi			192.168.140.20_P2P_Down	
mark packet		vnejsi			192.168.140.20_HTTP_Down	
mark packet		vnejsi			192.168.140.20_ICMP_Down	

Obr. 18: Značkování packetů

Na obr. 18[15] je vidět aplikace značkovacích pravidel, které rozeznávají příchozí a odchozí provoz, tento provoz v jednotlivých směrech je dále rozdělen na veškerou, P2P, HTTP a ICMP komunikaci.

Jakmile máme označkovanou danou komunikaci klienta s vnější sítí, můžeme plynule přejít k samotnému řízení datového toku. Jedná se o stromovou strukturu (obr. 7[15]), kdy nejdříve definujeme hlavní datové toky jednotlivými směry – down, up. Tyto toky opět rozdělíme podle přístupových bodů (hlavní_AP, AP_skola) tak, aby měly všechny stejnou prioritu. Každý přístupový bod je opět rozvětven na jednotlivé klienty přistupující k danému bodu a jednotlivým klientům je dále datový tok rozdělen podle označkovaných packetů.

```
queue tree add name=Klient_192.168.140.20_P2P_Up
parent=Klient_192.168.140.19_Vse_Up packet-mark=192.168.140.19_P2P_Up
limit-at=100 queue=default priority=8 max-limit=235929.6 burst-limit=0
burst-threshold=0 burst-time=0s disabled=no
```

Na uvedeném příkladu definice toku dat můžeme vidět omezení P2P provozu z vnitřní sítě do internetu.

Name	Packet Mark	Limit At (b...	Max Limit...
down		1M	13M
hlavni_AP_down		100k	5M
Michajlovicova_192.168.140.20_Vse_Down		0	3585024
Michajlovicova_192.168.140.20_HTTP_Down	192.168.140.20_HTTP_Down	100	2924184
Michajlovicova_192.168.140.20_ICMP_Down	192.168.140.20_ICMP_Down	100	100k
Michajlovicova_192.168.140.20_Oth_Down	192.168.140.20_Vse_Down	100	2924184
Michajlovicova_192.168.140.20_P2P_Down	192.168.140.20_P2P_Down	100	235929
AP_skola_down		100k	5M
up		1M	13M
hlavni_AP_up		100k	3M
Michajlovicova_192.168.140.20_Vse_Up		0	3585024
Michajlovicova_192.168.140.20_HTTP_Up	192.168.140.20_HTTP_Up	100	2924184
Michajlovicova_192.168.140.20_ICMP_Up	192.168.140.20_ICMP_Up	100	100k
Michajlovicova_192.168.140.20_Oth_Up	192.168.140.20_Vse_Up	100	2924184
Michajlovicova_192.168.140.20_P2P_Up	192.168.140.20_P2P_Up	100	235929
AP_skola_up		100k	3M

Obr. 19: Queue Tree

Na obr. 19 můžeme vidět, že součet maximální rychlosti ve větvi klienta, například pro download, je vyšší než maximální propustnost celkového downloadu daného klienta. Takto se to řeší proto, že když klient například sleduje streamované

[15] Obrázek z programu pro vzdálenou konfiguraci RouterBoardu WinBox

video z internetu přes webový prohlížeč a zároveň ještě prohlíží webové stránky, je jeho dostupná přenosová rychlost až 2,9Mb/s. Jakmile však začne využívat jinou aplikaci komunikující přes internet například VoIP, tak maximální přenosová rychlost stoupá též k 2,9Mb/s, ale celková rychlost nemůže přesáhnout 5Mb. Proto musíme při definici Queue Tree nastavit priority. Pokud tedy HTTP_Down má menší prioritu než Oth_Down (např. VoIP) začne se přenosová rychlost webu zmenšovat na rychlost maximálně 2Mb/s a VoIP zvyšovat na 2,9Mb/s, což je maximálně 5Mb/s. Po ukončení VoIP se opět začne navyšovat rychlost pro HTTP_Down.

Nyní jsme vytvořili nového klienta i s pravidly v RB přes terminál programu WinBox nebo přes SSH, stejným způsobem bychom ručně vytvořili a přidali do RB další klienty. My ovšem chceme, aby se tyto příkazy vkládaly přes SSH samy prostřednictvím HTTP serveru, a toho docílíme pomocí dříve vytvořeného komunikačního kanálu SSH a programu Expect. Nejdříve stvoříme v programovacím jazyku PHP funkci, která bude například z databáze MySQL číst klienty s nastavením jejich IP adres a automaticky do souboru generovat konfigurační skripty popsané v této kapitole. Abychom mohli z tohoto souboru načíst a provést konfigurační příkazy prostřednictvím programu expect, je nutné do záhlaví skriptu vložit hlavičku s IP adresou nebo DNS názvem ROS a naopak na konci skriptu nesmíme také celou komunikaci, respektive SSH kanál a program expect ukončit.

Na následujících několika řádcích je ukázka zkráceného konfiguračního souboru vygenerovaného pomocí PHP skriptu.

```
spawn ssh -l skript router.net "
/ ip firewall address-list add list=klient address=192.168.140.20
  disabled=no comment="Klient Ulice CP"
                                    .
                                    .
                                    .
/ queue tree add name=Klient_192.168.140.20_P2P_Down
  parent=Klient_192.168.140.20_Vse_Down packet-
  mark=192.168.140.20_P2P_Down limit-at=100 queue=default priority=8
  max-limit=235929.6 burst-limit=0 burst-threshold=0 burst-time=0s
  disabled=no
/quit
"
expect "#"
```

5.4.5 Monitorování množství přenesených dat uživatelů

Pomocí SNMP jsme schopni získat od ROS nejrůznější informace, od jména zařízení, umístění, vytížení procesoru až po počet připojených klientů, MAC adresy

klientů, sílu signálu u WiFi atd. Abychom toho byli schopni i v našem případě, je nejprve nutné aktivovat SNMP přímo v ROS zadáním (ať už do terminálu nebo přes SSH) následujícího příkazu:

```
snmp set enabled=yes contact="Petr Hromadko" location=" Brno"
```

Tímto jsme aktivovali SNMP na RB v Brně a kontaktní osobou je Petr Hromadko. Pokud bychom chtěli vytvořit nový *LogIn* či zadat IP adresu, ze které by bylo pouze možné zadávat SNMP dotazy, zadali bychom toto:

```
snmp community add address="192.168.132.128" name="LogIn"
```

Nyní ROS naslouchá a vyčkává na příchozí SNMP dotazy z IP adresy 192.168.132.128. Protože nainstalovaný skriptovací jazyk PHP na HTTP serveru nemá ve výchozím stavu nainstalovanou podporu SNMP, je nutné jí doinstalovat.

```
aptitude install php5-snmp
```

Po zapnutí podpory SNMP u ROS a doinstalovaní modulu pro PHP na HTTP server, jsme schopni prostřednictvím PHP skriptu kontaktovat routerboard. Celá komunikace je prováděna systémem klient-server. Pro vzájemnou komunikaci jsme vytvořili novou třídu *traffic*:

```
<?php
class traffic {
protected $UP_DB = '4_UP';
protected $DOWN_DB = '4_DOWN';
protected $year = '2009';
protected $snmp_data;
```

Jak už bylo řečeno, základem celé komunikace je protokol SNMP a PHP funkce *snmpwalkoid*, která načte do pole požadované informace. Bohužel požadovaná data nejsou umístěna ve stejné koncové větvi stromové struktury MIB databáze, proto musíme provést dva dotazy. Při prvním dotazu uložíme do proměnné *$snmp_name* jména všech klientů připojených do naší sítě a registrovaných na hlavním routerboardu. Druhým dotazem načteme do pole *$snmp_data* informace o množství přenesených dat u

jednotlivých klientů. Následuje ještě třetí dotaz, který spustí v routerboardu skript (příloha č. 2) pro vynulování množství přenesených dat klientů.

```
public function monitoring() {
// nacteme klienty z roterboardu
      $snmp_name = snmpwalkoid('192.168.1.1', 'adresace',
 'SNMPv2-SMI::enterprises.14988.1.1.2.2.1.2');
// nacteme mnozstvi prenesenych dat
      $this->snmp_data = snmpwalkoid('192.168.1.1',
 'adresace', 'SNMPv2-SMI::enterprises.14988.1.1.2.2.1.5');
// smazeme data v routerboardu
      snmpset('192.168.1.1', 'adresace',
 '1.3.6.1.4.1.14988.1.1.8.1.1.3.3', 's', '1');
```

Jak jsme si ukázali v předchozí kapitole o nastavení routerboardu serverem, vytvořili jsme pro každého klienta několik pravidel pro upload i download. Jelikož chceme ovšem počítat data pouze pro celkový download i upload klienta, musíme projít pole $snmp_name a vyfiltrovat pouze požadovaná data.

```
      foreach ($snmp_name as $oid=>$klient)
      {
// Vytvorime nove pole do ktereho umistime casti retezce
      $dummy = explode(" ", $klient);
// Odmazeme uvozovky
      $klient = trim(strTr($dummy[1], "\"", " "));
// Podle hodnoty Down/Up ulozime data
            if(strpos($klient, 'Vse_Down')) {
                  $this->search($oid, $klient, $DOWN_DB);
            }
            if(strpos($klient, 'Vse_Up')) {
                  $this->search($oid, $klient, $UP_DB);
            }
      }
}
```

Jakmile cyklus *foreach* narazí na požadovaný záznam, je volána funkce *search()*, která vyhledá k požadovanému klientovi v poli *$snmp_dat* příslušný záznam obsahující množství přenesených dat. Tato data jsou sečtena s daty již uloženými v databázi a následně je provedena funkce pro update položky v databázi s aktualizovanými daty.

```
private function search($oid, $klient, $down_up) {
      // Rozdelime pole
      $IP = explode('_', $klient);
      // Nacteme data z DB podle IP adrresy
      $data_DB = $this->select_data($IP[1]);
      // Ziskame potrebne ID z snmp dotazu
      $oid_search = '14988.1.1.2.2.1.2';
      $skip = strpos($oid, $oid_search) +
            strlen($oid_search) + 1;
```

```
            $id = substr($oid, $skip);
            // Vybereme z pole pozadovany zaznam
            $traffic = $this->snmp_data['SNMPv2-
                    SMI::enterprises.14988.1.1.2.2.1.5.' . $id];
            // Rozdelime pole na typ a hodnotu
            $data = explode(" ", $traffic);
            // Secteme data z DB a z routerboardu
            $data_all = ($data_DB[$down_up] + $data[1]);
            // Vytvorime dotaz do DB
            $data_all = $down_up . " = '" . $data_all . "'";
            // Ulozime nova data do DB
            $this->save_traffic($data_all, $data_DB['ID']);
      }
```

Poslední dvě funkce v třídě *traffic* se starají o obsluhu databáze, to je načítání uložených dat z požadované tabulky v databázi a následné uložení aktualizovaných dat funkci *search()*.

```
      private function select_data($ip) {
            $result = mysql_query("SELECT ID, $UP_DB, $DOWN_DB
                                   FROM   traffic, pripojeni
                                   WHERE  traffic.ID_pripojeniP =
                                          pripojeni.ID_pripojeni
                                   AND    IP_klient = '$ip'
                                   AND    rok = '$year'");
          if (!$result)
          die('Interni chyba.');
          return mysql_fetch_array($result);
      }

      private function save_traffic($data, $id) {
            $result = mysql_query("UPDATE traffic
                                   SET    $data
                                   WHERE  ID = '$id'");
          if (!$result)
          continue;
      }
```

Na závěr musíme celou třídu ukončit.

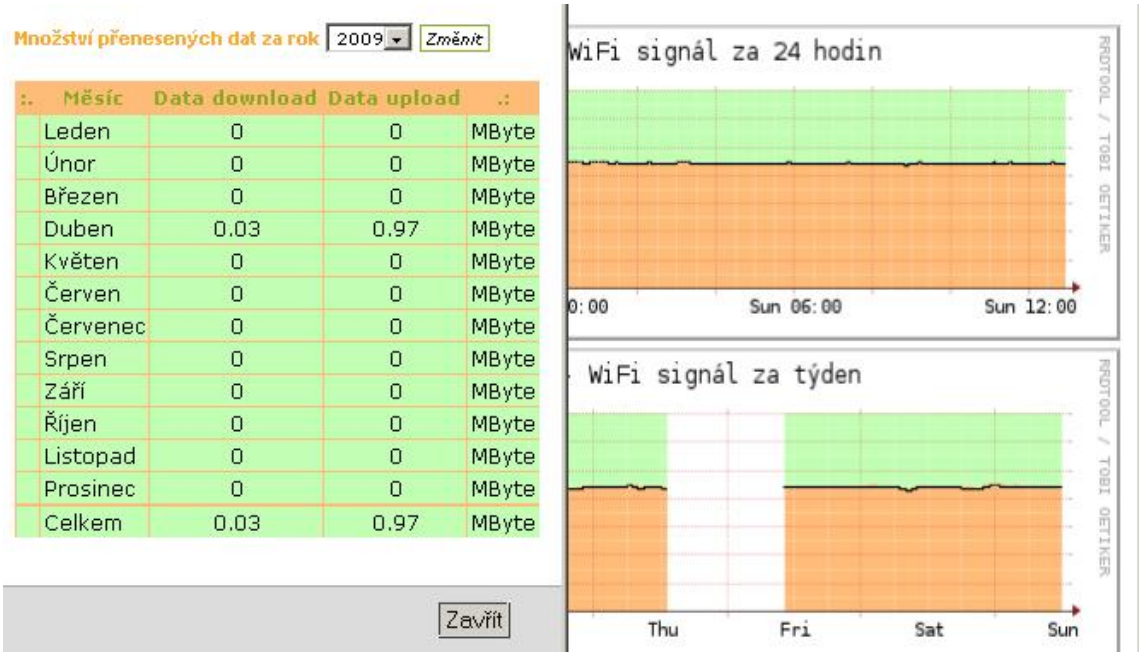
```
}
?>
```

Jakmile máme takto napsaný funkční skript, je nutné ještě nastavit server, aby každou hodinu tento skript spustil. Kdyby se tak nestalo, žádná data by se automaticky neaktualizovala a my bychom se museli co hodinu přihlásit do systému a skript spustit

ručně. To ovšem nechceme a proto vložíme záznam pro daemona *crond* do tabulky *crontab*[16]:

```
* 1-23  * * *   root    /var/www/adresace/admin/cron/stahni_data
```

Tímto je každou hodinu volán automaticky soubor *stahni_data* obsahující příkaz pro spuštění třídy *traffic*. Následně můžeme získaná data zpracovat ve formě tabulky nebo grafu (obr. 20).



Obr. 20: Zpracovaní dat

Pro uložení množství přenesených dat jednotlivých klientů je využita primární MySQL databáze. Ovšem abychom byli schopni uložit a následně zobrazit graf síly signálu klientů (obr. 20), je použita RRD databáze (Round Robin Databáze) a nástroj RRDTool [6]. Její předností oproti MySQL je především jednoduché vytvoření (v rámci jednoho klienta jeden soubor), její následné cyklické plnění daty v časové rovině a vykreslení grafů v rámci definovaného časového rozsahu. Využití najde především tam, kde je potřeba monitorovat průběh veličiny měnící se v čase a toto je i případ modulu sledující WiFi signál připojených klientů. Použití není nikterak složité, nejdříve je nutné

[16] http://www.manpagez.com/man/5/crontab/

vytvořit samotný soubor databáze, kde určíme název tabulky, do které se budou data zapisovat v daných časových odstupech.

```
//Vytvorime novou DB s krokem 300s
$prikaz  ='/usr/bin/rrdtool create ' . $klient . ' --step 300 ';
//Definice nove DB, nazev: signal, GAUGE zapise aktualni hodnotu, cas
mezi vzorky:
$prikaz .='DS:Signal:GAUGE:600:-100:0 ';
//Vytvorime archivy pro sber dat co:
//5 minut pro 288 vzorku, coz je 24 hodin (24*60)/5
$prikaz .='RRA:AVERAGE:0.5:1:288 ';
//1 hodina pro 2016 vzorku coz je tyden po 1 hodine 24*7
$prikaz .='RRA:AVERAGE:0.5:12:168 ';
//pul den pro 180 vzorku coz jsou 3 mesice
$prikaz .='RRA:AVERAGE:0.5:72:180 ';
```

Při následném zadávání nových hodnot použijeme jednoduchý příkaz *update*.

```
$prikaz  = '/usr/bin/rrdtool update ' . $klient . ' N:' . $sig;
```

Jakmile máme databázi naplněnou daty, můžeme následně z těchto hodnot jednoduše vykreslit graf. Jediným povinným parametrem je umístění souboru, ze kterého se má generovat graf. To by nám jistě nestačilo, proto k tomu máme nepřeberný počet volitelných parametrů, kterými definujeme titulek, barvu grafu, barvu pozadí, rozsah os atd.

```
$img = '/var/www/adresace/admin/rrd/images/' . $klient . $time .
       '.png";
$rrd_file = "/var/www/adresace/admin/rrd/' . $klient . '.rrd';

$prikaz ='rrdtool graph ' . $img;
//Titulek
$prikaz .='-t WiFi signál ';
$prikaz .='-h 100 -w 550 ';
//Format vystupu
$prikaz .='-a PNG ';
//Urcuje velikost osy Y
$prikaz .='-r ';
$prikaz .='-u -30 ';
$prikaz .='-l -100 ';
//Urcuje od kdy se ma zobrazit graf
$prikaz .='-s now-' . $time . ' ';
//Urcuje do kdy ma byt zobrazen graf
$prikaz .='-e now ';
//Popis Y osy
$prikaz .='-v dBm ';
//Bila barva pozadi
$prikaz .='--color BACK#ffffff ';
//Barva pozadi grafu
$prikaz .='--color CANVAS#ffffff ';
//Brva mrizky
```

```
$prikaz .='--color GRID#ffffff ';

$prikaz .='DEF:sig=$rrd_file:Signal:AVERAGE ';
//Barva nad krivkou
$prikaz .='AREA:sig#C2FFB5:: ';
//Barva pod krivkou
$prikaz .='AREA:sig#FFBC79::STACK ';
//Barva kriky
$prikaz .='LINE1:sig#000000:: ';
//Vykoname prikaz
exec($prikaz);
```

Výsledkem příkazu je obrázek ve zvoleném formátu (jpeg, png), který následně můžeme zpracovat pomocí HTML tagu <*img*>.

5.5 Zhodnocení realizovaného systému

Realizovaný informační systém byl několik týdnů testován v reálném prostředí u ISP (Internet service provider) s 200 klienty. Mezi největší přednosti, které uživatelé uvedli, patřily jednoduchost, rychlost jakýchkoli změn a možnost nahlížení klientů do tabulky přenesených dat. Administrativní zaměstnanci viděli výhodu především v nutnosti zadávat všechny kontaktní údaje a tudíž odpadlo pracné dohledávání kontaktních údajů při fakturaci či jiné administrativní činnosti u starších klientů. Největším přínosem pro organizaci používající informační systém je časové a tudíž i finanční měřítko, neboť bylo praktickými testy dokázáno, že nový klient je připojen i se zadáváním kontaktních údajů do systému během dvou minut. Naopak pokud se muselo vše konfigurovat ručně, trvala tato operace až deset minut, v případě změny služby klienta tato změna byla ještě podstatnější, kdy stačí pouze pozměnit údaj v IS a vše je automaticky nastaveno. Dříve se muselo provést několik změn na dvou až třech přístupových bodech. Dále byl na testovací síti simulován výpadek jednoho AP (access point) a následná konfigurace nového. Než technik nakonfiguroval nový ručním zadáváním hodnot, uplynulo průměrně 30 minut, automatickou konfigurací prostřednictvím IS bylo veškeré nastavení hotovo do 2 minut. V celkovém hodnocení lze tedy říci, že organizace vlastnící informační systém ušetří mnoho prostředků za administrativu, rychleji může reagovat na vzniklé situace a v očích klienta bude serioznější při dvouminutovém výpadku oproti několika desítkám minut.

6. Závěr

V rámci této práce jsem se zabýval popisem třívrstvé architektury a jejím rozdílem oproti architektuře klient/server. Dále jsem popsal webové technologie, které budou použity pro implementaci informačního systému jako PHP, JavaScript, (X)HTML, CSS. Stručně jsem popsal použití jazyka UML pro definici návrhu IS.

V druhé části teoretického úvodu jsem se věnoval charakteristice síťových protokolů http, https a především SSH a SNMP, které budou použity pro komunikaci mezi HTTP serverem a ROS.

Praktická část byla rozdělena na dva důležité celky. V první části jsem se důkladně věnoval analýze návrhu datového modelu. V případě špatně navrženého E-R diagramu bychom mohli v průběhu realizace informačního systému zjistit podstatné nedostatky, což by mohlo mít zásadní vliv na další pokračovaní v implementaci. Kdyby tato situace nastala, museli bychom se rozhodovat zda pozměnit stávající schéma a pokračovat nebo začít zcela od začátku. Proto je velký úsek praktické části věnovaný správnému návrhu struktury databáze, aplikaci normálních forem a jsou analyzovány vztahy mezi tabulkami. Nedílnou součástí analýzy je i zabezpečení celé aplikace, proto jsem se v části bezpečnost snažil nastínit nejdůležitější pravidla, jak se vyhnout bezpečnostním hrozbám a jak se jim bránit.

V závěrečné části jsem nastínil podobu webového informačního systému – jak by měl vypadat, z jakých prvků se skládá a jakou formou je realizován. Nesnažil jsem se popsat všechny skripty, funkce, HTML kód atd., protože se jedná o tisíce řádků a není to předmětem této práce. Naopak jsem podrobně popsal možnosti realizace propojení HTTP serveru s routerboardem MikroTik prostřednictvím kanálu SSH a protokolu SNMP. A také využití jiného typu databáze (RRD) pro uložení velkého množství dat získaných z routerboardu MikroTik. Právě RRD databáze je vhodná pro uložení dat z monitorování např. signálu, využití CPU, teploty atd., protože umožňuje uložení dat v čase a lze jí využít ve skriptech psaných v Perl, Python, Ruby, TCL nebo PHP jazycích. Pro jasné pochopení jsou uvedeny podrobně okomentované zdrojové skripty.

Celá aplikace informačního systému byla nasazena do reálných podmínek u poskytovatele internetového připojení a její zhodnocení přímo uživateli systému je obsaženo v předchozí podkapitole Zhodnocení realizovaného systému.

Literatura

1. Bouška Petr: *SNMP - Simple Network Management Protocol* [online]. 2006 [cit. 2008-12-07]. Dostupný z WWW: <http://www.samuraj-cz.com/clanek/snmp-simple-network-management-protocol/>.
2. Case, Fedor, Davin, Schoffstall: *RFC 1157* [online]. 1990 [cit. 2008-12-15]. Dostupný z WWW: <http://www.ietf.org/rfc/rfc1157.txt>.
3. Castagnetto Jesus: Programujeme PHP profesionálně. Vyd. 1. Praha: Computer Press, 2001. xxiv, 656 s. :. ISBN: 80-7226-310-2.
4. Drift Jeff, Lloyd Ian, Rubin Dan: Mistrovství v CSS : pokročilé techniky pro webové designéry a vývojáře ; [překlad Josef Bábík]. Vyd. 1.. Brno : Computer Press, 2007. -- 409 s. :. ISBN: 978-80-251-1705-7.
5. Duyne Douglas K. van, Landay James A., Hong Jason I.: Návrh a tvorba webů : vytváříme zákaznicky orientovaný web. Vyd. 1.. Brno : Computer Press, 2005. 672 s. :. ISBN: 80-251-0508-3.
6. GIGEL , Milan . *Prechádzame na RRDTOOL* [online]. 2001 [cit. 2009-04-06]. Dostupný z WWW: <http://www.root.cz/clanky/prechadzame-na-rrdtool/>.
7. Gutmans Andi, Bakken Stig Saether, Rethans Derick: Mistrovství v PHP 5. Vyd. 1.. Brno : Computer Press, 2005. 655 s. :. ISBN: 80-251-0799-X.
8. Hlavenka Jiří: Vytváříme www stránky. 5. aktualiz. vyd.. Praha: Computer Press, 2001. xiii, 437 s. :. ISBN: 80-7226-494-X.
9. Holzner Steven: JavaScript profesionálně : [kompletní referenční příručka]. Praha : Mobil Media, c2003. 1071 s. :. ISBN: 80-86593-40-1.
10. Chapman Stephen J.: Začínáme programovat v jazyce Java. Vyd. 1.. Praha : Computer Press, 2001. ix, 307 s. :. ISBN: 80-7226-472-9.
11. Kotlet Michael: Mistrovství v MySQL 5 : [kompletní průvodce webového vývojáře]. Vyd. 1.. Brno : Computer Press, 2007. 805 s. :. ISBN: 978-80-251-1502-2.
12. Lacko Ľuboslav: Web a databáze. -- Vyd. 1.. Praha : Computer Press, 2001. xii, 250 s. :. ISBN: 80-7226-555-5.
13. Malý, J.; Kacálek, J.: Zabezpečení webových aplikací III. - ostatní útoky, nastavení prostředí. *Access Server,* 2007, roč. 2007, č. 8, s. 1-6. ISSN: 1214-9675.

14. Mikrotikls SIA. *SNMP Service* [online]. 1999-2006 , 15.09.2004 [cit. 2008-12-07]. Dostupný z WWW: <http://www.mikrotik.com/testdocs/ros/2.9/root/snmp.php>.
15. Mikrotikls. *MikroTik Routers and Wireless* [online]. 2000-2006 [cit. 2008-12-07]. Dostupný z WWW: <http://www.mikrotik.com/>.
16. Nemeth Evi, Snyder Garth, Hein Trent R.: Linux : kompletní příručka administrátora. Vyd. 1.. Brno : Computer Press, 2004. xxviii, 828 s. :. ISBN: 80-7226-919-4.
17. *Net-SNMP* [online]. [2005] , 02-Mar-2007 [cit. 2008-12-07]. Dostupný z WWW: <http://www.net-snmp.org/>.
18. Powell Thomas A.: Web design : kompletní průvodce; [překlad Petr Matějů]. Vyd. 1.. Brno : Computer Press, 2004. xxviii, 818 s. :. ISBN: 80-7226-949-6.
19. Shah Steve, Soyinka Wale: Administrace systému Linux : překlad čtvrtého vydání. 1. vyd.. Praha : Grada, 2007. 426 s. ;. ISBN: 978-80-247-1694-7.
20. Schmuler Joseph: Myslíme v jazyku UML [i.e. Schmuller]. 1. vyd.. Praha: Grada, 2001. 359 s. :. ISBN: 80-247-0029-8.
21. SPI. *Debian -- Univerzální operační systém* [online]. 1997-2008 [cit. 2008-12-15]. Dostupný z WWW: <http://www.debian.org/>.
22. Sun Microsystems. *MySQL :: The world\'s most popular open source database* [online]. 1995-2008 [cit. 2008-12-07]. Dostupný z WWW: <http://www.mysql.com/>.
23. *The Expect Home Page* [online]. 2006 [cit. 2008-12-09]. Dostupný z WWW: <http://expect.nist.gov/>.
24. The Internet Society. *RFC 2617 HTTP Authentication* [online]. 1999 [cit. 2009-03-14]. Dostupný z WWW: <http://www.ietf.org/rfc/rfc2617.txt?number=2617>.
25. Zakhour Sharon: Java 6 : výukový kurz. Vyd. 1.. Brno : Computer Press, 2007. 534 s. :. ISBN: 978-80-251-1575-6

Seznam obrázků

Seznam tabulek

Příloha č. 1

Generování certifikátů

V prvním kroku si musíme vytvořit vlastní certifikační autoritu, kterou si následně podepíšeme certifikát. Samozřejmě si můžeme nechat certifikát podepsat i od kvalifikované certifikační autority, ale budeme za to muset zaplatit.

```
openssl genrsa -out my-ca.key 2048
openssl req -new -x509 -days 3650 -key my-ca.key -out my-ca.crt
```

V dalším kroku vygenerujeme samotný certifikát pro server, následně podáme žádost o podepsání vlastní certifikační autoritou a na závěr certifikát podepíšeme.

```
openssl genrsa -out server.key 1024
openssl req -new -key server.key -out server.csr
openssl x509 -req -in server.csr -out server.crt -sha1 -CA my-
ca.crt -CAkey my-ca.key -
     CAcreateserial -days 3650
```

Při zadávání názvu serveru musíme zadat přesný název, jinak se bude uživatelům zobrazovat varování, že certifikát nepatří navštívenému serveru.

Nastavení Apache serveru

V prvním kroku je nutné povolit v konfiguraci Apache port 443. Do souboru */etc/apache2/ports.conf* doplníme „Listen 443“, čímž bude server naslouchat na portu TCP 443.

V druhém kroku doplníme do souboru */etc/apache2/sites-enabled/default* následující záznam. Takto můžeme nakonfigurovat Apache server pro různé domény na jednom fyzickém serveru či naslouchající na různém portu.

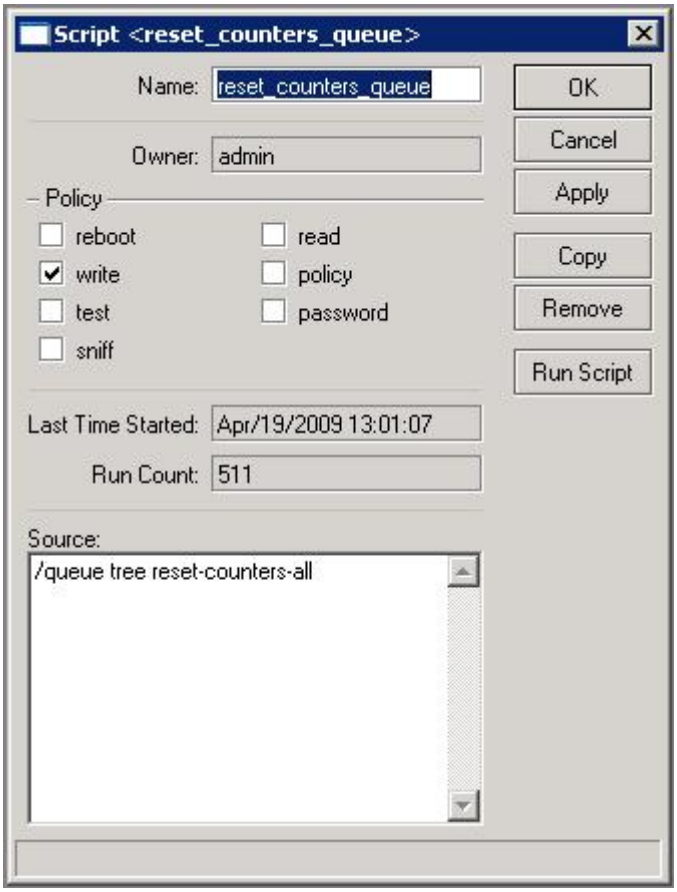
```
<VirtualHost *:443>
        ServerAdmin webmaster@localhost

        # Zapnuti jádra SSL
        SSLEngine on
        SSLCertificateFile /cesta/k/certifikatu/server.crt
        SSLCertificateKeyFile /cesta/k/certifikatu/server.key
        SSLProtocol All -SSLv2
        SSLCipherSuite HIGH:MEDIUM

        DocumentRoot /var/www/
```

```
        <Directory />
                Options FollowSymLinks
                AllowOverride All
        </Directory>
        <Directory /var/www/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride All
                Order allow,deny
                allow from all
                RedirectMatch ^/$ /adresace
        </Directory>
</VirtualHost>
```

Tímto jsme vytvořili nové virtuální hostování pro naši aplikaci na portu TCP 443 se zapnutou podporou SSL. Na závěr je nutné Apache server restartovat pro aktualizaci nastavení.

Příloha č. 2

Jakmile v routerboardu MikroTik vytvoříme skript pro spuštění, jsme schopni ho spustit vzdáleně prostřednictvím SNMP. V tomto případě se jedná o jednoduchý příkaz pro vynulování množství přenesených dat klientů v *queue tree*.



Obr. P1: Reset counters queue

Příloha č. 3

Navržený informační systém je možné si prakticky otestovat v On-line demu na adrese:
URL: **http://www.modruss.eu/adresace/admin**
Jméno: **adresace@modruss.eu**
Heslo: **vut**
Po přihlášení se návštěvník dostane do administrátorské sekce, kde má oprávnění *administrátor* bez možnosti blokovat zaměstnance.

nebo
URL: **http://www.modruss.eu/adresace/**
Jméno: **modruss@modruss.eu**
Heslo: **vut**
Po přihlášení se návštěvník dostane do klientské sekce.