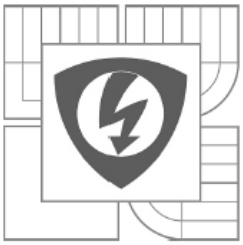
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A IMPLEMENTACE SYSTÉMU PRO VÝMĚNU STATISTICKÝCH INFORMACÍ O SÍŤOVÉM PROVOZU MEZI PŘÍSTUPOVÉ BODY WLAN SÍTĚ

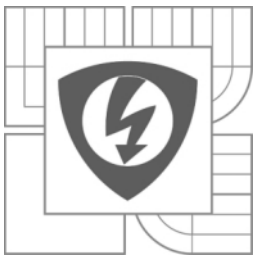
DESIGN AND IMPLEMENTATION OF A SYSTEM FOR STATISTICAL DATA EXCHANGE BETWEEN ACCESS POINTS ON A WLAN NETWORK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETER LENK
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. KAROL MOLNÁR, Ph.D.
SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Peter Lenk	***ID:***	83532
Ročík:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Návrh a implementace systému pro výměnu statistických informací o síťovém provozu mezi přístupové body WLAN sítě

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte aplikaci pro operační systém OpenWRT, zprovozněný v zařízení Mikrotik RB433, která zajistí distribuci paketů přes bezdrátovou síť využitím unicastového a multicastového (příp. broadcastového) přenosu. Přenášené zprávy budou obsahovat informace o krátkodobém a dlouhodobém zatížení WLAN rozhraní daného zařízení.
Umožnete konfiguraci způsobu přenosu i parametrů statistik pomocí textového souboru.
Následně rozšiřte aplikaci o možnost komunikace s větším počtem zařízení, aby přístupový bod mohl získat statistická data od všech svých sousedů, na kterých běží daný program.
Dále rozšiřte aplikaci tak, aby na základě informací o vytíženosti a intenzity signálu dokázala vybrat nejvhodnějšího souseda pro handover.

DOPORUČENÁ LITERATURA:

[1] JURČÍK, M. Nízkoúrovňové řízení a sběr dat v přístupovém bodu: semestrální projekt. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010.

[2] Hallian, C.: Embedded Linux Primer: A Practical Real-World Approach. Indiana: Prentice Hall, 2006. 576 s. ISBN: 0-13-167984-8.

Termín zadání:	6.2.2011	***Termín odevzdání:***	24.5.2011

Vedoucí práce: doc. Ing. Karol Molnár, Ph.D.

Konzultanti diplomové práce:

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

ABSTRAKT

Diplomová práca sa zaoberá možnosťami implementácie metódy na zber a distribúciu štatistických informácií o sieťovej prevádzke v prístupovom bode MikroTik pri použití virtualizovaného systému OpenWRT. Prvá časť práce obsahuje stručný popis vlastností zariadenia a rozšírenie jeho možností pomocou virtualizácie operačného systému. Obsahom druhej časti je návrh skriptov slúžiacich na zber dát priemerného zaťaženia sieťových rozhraní zariadenia. Ďalej sa práca zaoberá možnosťami prenosu získaných štatistických dát a popisuje implementáciu navrhnutých programov v jazyku C. Záver práce pojdenáva o implementácii uceleného programu na zber, odosielanie a príjem dát s možnosťou konfigurácie parametrov a automatickým získaním konfigurácie.

KĽÚČOVÉ SLOVÁ

prístupový bod, OpenWRT, skriptovací jazyk, zber dát, sokety

ABSTRACT

This thesis explores the possibilities of implementation of methods for data acquisition in wireless access point MikroTik while using virtualized OpenWRT system. First part of the paper contains short introduction into device's features and extention of the features by virtualizing the operation system. Design and implementation of the scripts for average load of interfaces data acquisition is in the second part of the paper. The next part deals with the transmission of the gathered data and implementation of the proposed programs in C language. The last part of the paper covers the implementation of a complete program to collect, send and receive data with the option of parameters configuration, and automatic acquisition of the configuration.

KEYWORDS

access point, OpenWRT, scripting language, data acquisition, sockets

LENK, P. *Návrh a implementace systému pro výměnu statistických informací o síťovém provozu mezi přístupové body WLAN sítě.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 52 s. Vedoucí diplomové práce doc. Ing. Karol Molnár, Ph.D..

PREHLÁSENIE

Prehlasujem že svoju diplomovú prácu na tému „Návrh a implementace systému pro výměnu statistických informací o síťovém provozu mezi přístupové body WLAN sítě“ som vypracoval samostatne pod vedením vedúceho diplomovej práce s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky uvedené v zozname použitej literatúry na konci práce. Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnotsných a som si plne vedomý následkov porušenia ustanovenia §11 a nasledujúcich autorskych zákonov č.121/2000 Sb., včetne možných tresnoprávnych dôsledkov vyplyvajúcich z ustanovania §152 trestného zákona č.140/1961 Sb.

V Brne dňa . . . .

(podpis autora)

POĎAKOVANIE

Ďakujem vedúcemu práce doc. Ing. Karolovi Molnárovi, Ph.D. za koordináciu a odborné vedenie behom mojej práce a za možnosť nahliadnuť do sveta sieťového programovania. Ďalej sa chcem poďakovať svojej rodine za neustálu podporu a porozumenie.

V Brne dňa . . . .

(podpis autora)

Obsah

ÚVOD

Bezdrôtové technológie hrajú v živote moderného človeka veľkú rolu. Počet zariadení a služieb využívajúcich vzduch ako prenosové médium stále stúpa, pretože ich veľkou výhodou je mobilita poskytovaná účastníkovi. Myšlienka použitia systémov automatizovaného dohľadu bezdrôtových sietí je teda aktuálna a cieľom ich vývoja je vylepšenie služby a uľahčenie manažmentu sietí. Neoddeliteľnou súčasťou takého systému je zber nízkoúrovňových dát v jednotlivých prístupových bodoch, ich následné spracovanie a distribúcia informácií v sieti. Diplomová práca sa zaoberá práve týmito možnostami zberu dát a ich spracovania v prístupovom bode MikroTik pri použití virtualizovaného systému OpenWRT.

Prístupový bod MikroTik je je vysokorýchlostný prístupový bod ktorý môže slúžiť ako bezdrôtový most alebo smerovač a jeho operačným systémom je RouterOS. Jeho výhodou je možnosť použiť vlastný virtualizovaný operačný systém fungujúci nad pôvodným systémom, a to za účelom rozšírenia vlastností prístupového bodu o možnosť použiť pokročilé funkcie linuxového systému alebo implementáciu vlastných programov. V práci je použitý model MikroTik RB433 a jej prvá časť sa venuje nastaveniu prístupového bodu a inštalácii virtualizovaného systému OpenWRT.

Ďalšia časť práce pojednáva o možnostiach zberu dát o sieťovej prevádzke na jednotlivých rozhraniach prístupového bodu a popisuje implementáciu navrhnutých skriptov. Je využitý skriptovací jazyk systému OpenWRT a na prácu s textovými súbormi je zvolený jazyk AWK. Metóda zberu je ďalej rozšírená o možnosť získania priemerných údajov o prevádzke počas zvoleného časového intervalu.

Návrhom spôsobu prenosu nazbierných dát a popisom príslušných programov navrhnutých v jazyku C sa zaoberá ďalšia časť práce. Programy pracujú na princípe klient-server a implementácia využíva programové konštruky nazývané sokety, ktoré umožňujú odosielať a prijímať správy od viacerých susedných zariadení. Je využitý protokol UDP a multicastové vysielanie k viacerým členom multicastovej skupiny. Pri vytváraní programov pre OpenWRT systém je nutné použiť krížovú kompiláciu a teda aj tento problém je v tejto kapitole popísaný .

Záver práce popisuje ucelený program, ktorého jadro tvorí riadiaci skript manažujúci jednotlivé skripty a programy navrhnuté v predchádzajúcich kapitolách. Program zároveň umožňuje konfiguráciu parametrov prenosu a taktiež aj automatické získanie configurácie od zariadení na ktorých beží daný program.

1 STRUČNÝ ÚVOD DO PROBLEMATIKY WLAN SIETÍ

WLAN siete (Wireless Local Area Networks - bezdrôtové lokálne siete) sú špecifikované v normách IEEE 802, ktoré definujú fyzickú a linkovú vsrstvu OSI modelu. WLAN sú konkrétne zastrešené štandardmi 802.11, ktorých dnes existuje niekoľko typov. Všetky typy majú spoločnú metódu prístupu k médiu ale líšia sa v riešení fyzickej vrstvy. Využíva sa metóda viacnásobného prístupu s predchádzaním kolíziám (Carrier Sense Multiple Access/Collision Avoidance - CSMA/CA) , ktorej princíp spočíva v použití potvrdzovania prijatia vysielaných dát. Protokol je ďalej zodpovedný aj za mechanizmus počiatočného spojenia a opätovného spojenia koncovej stanice s prístupový bodom.

Bezdrôtová lokálna sieť môže pracovať v dvoch konfiguráciách:

a) nezávislá konfigurácia (ad hoc) – stanice medzi sebou komunikujú priamo a nie je potrebné inštalovať žiadnu podpornú infraštruktúru. Takáto konfigurácia je mimoriadne vhodná pre náhodné usporiadanie (trvajúce podľa potreby hodiny, ale aj mesiace), ale nehodí sa pre rozsiahle riešenia.

b.) konfigurácia s využitím infraštruktúry v podobe prístupových bodov (Access Point, AP), ktoré fungujú ako základňové rádiové stanice a súčasne ako dátové mosty. V takomto prípade je centrom siete prístupový bod na ktorý sa jednotlivé klientské stanice pripájajú na základe signálu pravidelne vysielaného prístupovým bodom.

Prenosová rýchlosť a dosah v rámci WLAN závisí od použitej technológie. Štandard IEEE 802.11 je rozdelený na:

802.11b : pracovná frekvencia 2,4 GHz s prenosovou rýchlosťou 11 Mbps
802.11a : pracovná frekvencia 5 GHz s prenosovou rýchlosťou 54 Mbps (nie je kompatibilný s 802.11b)
802.11g : pracovná frekvencia 2,4 GHz, maximálna prenosovú rýchlosť 54 Mbps (kompatibilný s 802.11b)

ostatné štandardy: 802.11i (zvýšená bezpečnosť), 802.11 (manažment spektra a energie), 802.11e (kvalita služieb)

Medzi výhody použitia bezdrôtových lokálnych sietí patria rýchlosť a jednoduchosť inštalácie, prispôsobivosť, cenová efektivita, variabilnosť pri konfigurácii, alebo využitie bezlicenčného pásma 2,4 GHz. [8]

2 PRÍSTUPOVÝ BOD A SYSTÉM OpenWRT

Prístupový bod je zariadenie, ktoré navzájom prepája bezdrôtové sieťové komunikačné zariadenia, čím vytvára bezdrôtovú sieť. Bezdrôtový prístupový bod funguje ako fyzický opakovač (repeater) alebo ako smerovač (router). Zariadenie môže poskytovať vlastný DHCP server, možnost NAT (preklad adries), preklad portov do vnútornej siete (port forwarding), autentifikáciu klientov pomocou RADIUS serveru, rôzne úrovne šifrovania a podobne. Prístupové body vystupujú v niekoľkých rôznych úlohách, ktoré sú dané nielen požiadavkami na štruktúru siete, ale aj schopnosťami týchto zariadení. Schopnosti mnohých bezdrôtových zariadení sú ľahko rozšíriteľné pomocou zmeny softvérového vybavenia.

2.1 Prepojenie jednotlivých prístupových bodov

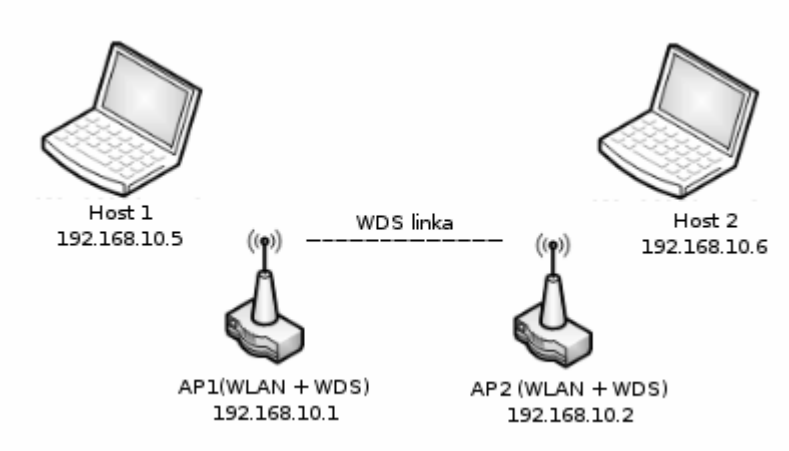
Podľa charakteru sieťe je možné pripojiť prístupové body viacerými spôsobmi. Konfigurácia ktorá spočíva v spojení LAN ethernet segmentov prostredníctvom prístupových bodov využíva bezdrôtové spojenie na komunikáciu medzi nimi, a toto spojenie slúži ako premostenie medzi jednotlivými ethernet sieťami. Ďalším možným zapojením je prepojenie prístupových bodov prostredníctvom ethernet linky kde jednotlivé segmenty pripojené k daným AP využívaju bezdrôtovú či ethernet technológiu na pripojenie klientských staníc. Táto práca však predpokladá čisto bezdrôtovú sieť v ktorej sa klienti pripájajú k prístupovým bodom bezdrôtovo a kde jednotlivé prístupové body slúžia na rozšírenie rozsahu siete bez potreby používať kabeláž na ich prepojenie.

2.1.1 Wireless distribution system (WDS)

Wireless distribution systém (bezdrôtový distribučný systém, WDS) je systém ktorý umožňuje bezdrôtové pripojenie prístupových bodov. Prístupový bod v takejto sieti môže slúžiť ako hlavná, medziľahlá alebo vzdialená stanica, pričom hlavná stanica je zvyčajne pripojená do siete ethernet. Každý z týchto bodov, alebo staníc, môže okrem komunikácie so susednými bodmi prijímať aj bezdrôtových klientov. Medziľahlé a vzdialené stanice preposielajú dáta medzi vzdialenými klientskými stanicami, ďalšími medziľahlými stanicami alebo hlavnou stanicou. Všetky prístupové body v sieti musia používať rovnaký rádiový kanál a v prípade použitia zabezpečenia aj rovnaké kľúče. WDS nie je IEEE certifikovaný štandard a teda každý výrobca hardwaru môže mať vlastnú implementáciu systému. Nie je preto zaručená vzájomná konektivita staníc v prípade že používajú rozdielny hardware, aj napriek tomu že môžu mať rovnaký operačný systém.

Zariadenie MikroTik umožňuje využitie WDS použitím bezdrôtového rozhrania Atheros a jeho nastavnie prebieha v operačnom systéme RouterOS, pričom je možné využiť už popísanú utilitu WinBox. V rámci nastavenia je nutné povoliť bezdrôtvé

rozhranie a vytvoriť most na ktorom bude WDS pracovať. Vzhľadom na to, že je použitý virtualizovaný systém OpenWRT, je nutné k mostu aj priradiť príslušné virtuálne rozhranie. Nastavenie na ostatných prístupových bodoch prebehne rovnako, pričom je nutné rozhraniam priradiť IP adresy rovnakej podsiete. Jednoduchú situáciu ilustruje obrázok 2.1.



Obr.2.1: Príklad zapojenia WDS v bezdrôtovej sieti

2.2 Prístupový bod Mikrotik RB 433

MikroTik RB433 je vysokorýchlostný prístupový bod ktorý môže slúžiť ako bezdrôtový most alebo smerovač. Jadrom dosky je procesor Atheros s pracovnou frekvenciou 300MHz. Doska je osadená pamäťovým modulom 64MB DDR SDRAM a obsahuje tri 10/100 Mb ethernet porty a 3 miniPCI porty slúžiace na pripojenie prídavných zariadení. K dispozícii je aj asynchrónny sériový port RS232C. Doska ďalej disponuje 64MB NAND pamäťou slúžiacou na uloženie dát.

Operačným systémom je RouterOS vďaka ktorému je možné zariadenie využívať ako smerovač, firewall či na manažment prenosového pásma. [7]

2.3 Operačný systém

Zariadenie Mikrotik RB433 obsahuje predinštalovaný operačný systém RouterOS. Je to samostatný systém založený na linuxovom jadre v2.6. Poskytuje všetky bežné služby ako smerovanie, firewall, manažment prenosového pásma, funkciu bezdrôtového prístupového bodu, Virtual Private Network server a iné. [7]

Konfigurácia systému je možná viacerými spôsobmi, medzi ktoré patrí lokálny prístup prostredníctvom klávesnice a monitora, použitie sériovej konzole či prístup prostredníctvom Telnetu a SSH. Ďalšou možnosťou je použitie webového

konfiguračného rozhrania alebo konfiguračného nástroja WinBox, prípadne využitie poskytovaného API za účelom vytvorenia vlastnej aplikácie. V ďalšom texte bude konfigurácia prebiehať prostredníctvom nástroja WinBox a príkazovej konzoly.

2.3.1 Základná konfigurácia systému

Nástroj WinBox sa do systému pripája na základe MAC adresy portu ku ktorému je pripojený klientský počítač. Po prihlásení je možné prostredníctvom grafického rozhrania spravovať všetky aspekty systému. Menu je logicky členené na jednotlivé oblasti a samotné rozhranie je zväčša intuitívne. Konzola sa spúšta tlačítkom *New terminal* z menu a predstavuje možnosť konfigurácie spôsobom príkazového riadku. Samotné príkazy sú členené do hierarchickej štruktúry kde koreňom je cesta , / '. Jednotlivé podpríkazy môžeme chápať ako adresáre ktoré obsahujú ďalšie súvisiace možnosti. Príkladom môže byť výpis sieťových adries pridelený jednotlivým rozhraniam, viď obrázok 2.2. V tomto prípade je koreňovým príkazom príkaz *ip* ktorý obsahuje podpríkazy súvisiace z protokolom IP a jeho nastaveniami. Príkaz *address* slúži na konfiguráciu jednotlivých adries, kde medzi podpríkazy patrí napríklad *add,* pomocou ktorého sa priraďujú nové adresy alebo príkaz *print* ktorý vypíše ich zoznam.

```
[admin@MikroTik] /interface> /ip
[admin@MikroTik] /ip> address
[admin@MikroTik] /ip address> print
Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         BROADCAST       INTERFACE
 0   192.168.99.1/24    192.168.99.0    192.168.99.255  ether3
 1 D 192.168.15.117/24  192.168.15.0    192.168.15.255  ether1
```

Obr.2.2: Výpis IP adries jednotlivých rozhraní

Pri základnej konfigurácii systému sa využívajú príkazy */interface* na nastavenie jednotlivých rozhraní, */ip address* na nastavenie sieťových adries, a */ip route* za účelom nastavenia smerovania.

2.3.2 Virtualizácia a metarouter

Systém RouterOS umožňuje pomocou funkcie MetaROUTER vytvoriť virtualizovaný systém bežiaci nad hostiteľským systémom. Takto vytvorený systém môže plne alebo čiastočne nahradiť pôvodný hostiteľský systém a to napríklad za účelom zvýšenia bezpečnosti, vytvorenia viacerých administračných domén alebo za účelom testovania. Táto funkcia zároveň umožnuje pripojeným klientom prístup k ich vlastnému smerovaču a jeho konfigurácii.

MetaROUTER podporuje až osem virtualizovaných systémov s maximálne ôsmimi virtuálnymi rozhraniami. Ďalšie rozšírenie počtu rozhraní je možné využitím virtuálych lokálnych sietí. Každý z týchto systémov má rovnaké nároky na pamäť ako hostiteľský systém, kde v prípade použitia zabudovaného RouterOS je to minimálne

16MB. K fyzickým rozhraniam zariadenia má virtualizovaný systém prístup prostredníctvom virtuálnych rozhraní ktoré musia byť premostené (bridged) s danými fyzickými portmi. [7]

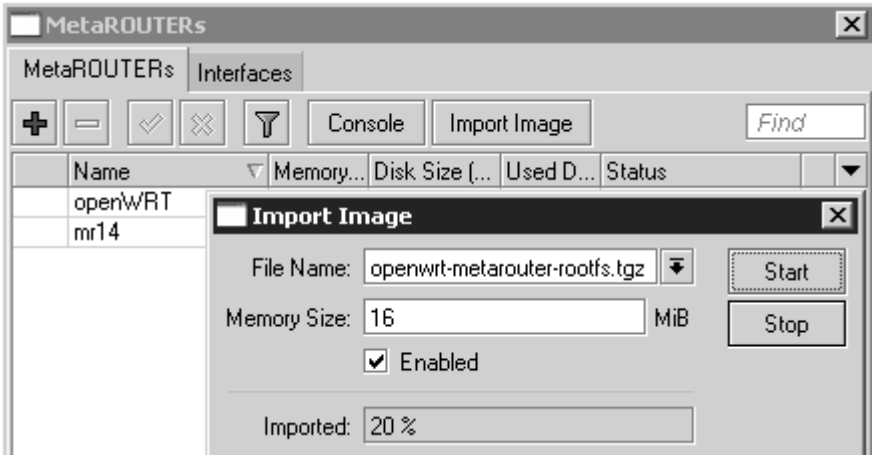
2.4 Operačný systém OpenWRT

Okrem využitia RouterOS ako virtualizovaného systému sa naskytuje aj možnosť využiť vlastný obraz skompilovaného systému, napríklad OpenWRT. Teto systém je zmenšená verzia linuxovej distribúcie, ktorý poskytuje kompletný zapisovateľný súborový systém a tiež aj manažment balíčkov. Použitie tohto systému umožňuje nielen využitie mnohých známych balíčkov ako PhP či Asterisk PBX, ale aj vytvorenie vlastnej aplikácie.

Obraz OpenWRT je možné buď skompilovať v prípade využitia vlastných balíčkov, alebo stiahnuť už pripravený z oficiálnych stránok Mikrotiku.

2.4.1 Inštalácia OpenWRT

Po získaní obrazu systému je potrebné preniesť ho do prístupového bodu. Toto je možné prostredníctvom WinBoxu jednoduchým pretiahnutím súboru do položky *files* alebo použitím FTP protokolu. Ďalším krokom je vytvorenie virtuálneho systému a to prostredníctvom okna sekcie MetaROUTER do ktorého sa v nástroji WinBox dostaneme z hlavného menu. Otvorené okno obsahuje všetky nainštalované virtuálne systémy. Systém typu RouterOS sa vytvorí jednoduchým pridaním nového MetaROUTERu tlačítkom *plus*. V našom prípade je potrebné zvoliť funkciu *Import Image* (obrázok 2.3).



Obr.2.3: Vytvorenie MetaROUTERU importom OpenWRT obrazu

Po úspešnom vložení obrazu je tento systém automaticky spustený a je možné k nemu pristupovať prostredníctvom konzole spustitelnej v okne MetaROUTER.

2.4.2 Nastavenie sieťových rozhraní

Na ďalšie nastavenia využijeme konzolu hostiteľského systému. Je potrebné vytvoriť virtuálne rozhranie prostredníctvom ktorého bude virtuálny systém komunikovať so systémom hostiteľským. Ďalej je potrebné toto rozhranie premostiť s fyzickým rozhraním prostredíctvom takzvaného *mostu (bridge).* Tieto kroky ilustrujú obrázky 2.4 a 2.5, kde dochádza k premosteniu fyzického rozhrania *ether2* a virtuálneho *vif1* .

```
[admin@MikroTik] /metarouter interface> add virtual-machine=openWRT type=dynamic
[admin@MikroTik] /metarouter interface> print
Flags: X - disabled, A - active
 #   VIRTUAL-MACHINE         TYPE     STATIC-INTERFACE         VM-MAC-ADDRESS
 0 A openWRT                 dynamic                           02:AE:9B:0C:F8:30
```

Obr.2.4: Priradenie nového virtuálneho rozhrania MetaROUTERu

```
[admin@MikroTik] > /interface bridge
[admin@MikroTik] /interface bridge> add
[admin@MikroTik] /interface bridge> print
Flags: X - disabled, R - running
 0  R name="bridge1" mtu=1500 l2mtu=65535 arp=enabled mac-address=00:00:00:00:00:00
      protocol-mode=none priority=0x8000 auto-mac=yes admin-mac=00:00:00:00:00:00
      max-message-age=20s forward-delay=15s transmit-hold-count=6 ageing-time=5m
[admin@MikroTik] /interface bridge> port
[admin@MikroTik] /interface bridge port> add interface=ether2 bridge=bridge1
[admin@MikroTik] /interface bridge port> add interface=vif1 bridge=bridge1
[admin@MikroTik] /interface bridge port> print
Flags: X - disabled, I - inactive, D - dynamic
 #    INTERFACE                 BRIDGE                    PRIORITY PATH-COST  HORIZON
 0 I  ether2                    bridge1                   0x80     10         none
 1    vif1                      bridge1                   0x80     10         none
[admin@MikroTik] /interface bridge port>
```

Obr.2.5: Vytvorenie nového mostu a premostenie rozhraní

Následne je nutné nastaviť rozhrania vo virtualizovanom systéme OpenWRT a to prostredníctvom konzoly. Je možné buď editovať súbor */etc/config/network* pomocou editora *vi* alebo použiť príkazy na nastavenie parametrov jednotlivých rozhraní, viď obrázok 2.6.

```
root@OpenWrt:/# uci set network.lan.ipaddr=192.168.1.1
root@OpenWrt:/# uci set network.lan.netmask=255.255.255.0
root@OpenWrt:/# uci commit
root@OpenWrt:/# reboot
```

Obr.2.6: Nastavenie rozhrania virtuálneho systému prostredníctom konzoly

3 METÓDA ZBERU INFORMÁCIÍ O SIEŤOVEJ PREVÁDZKE

3.1 Princíp navrhnutej metódy

Postup zberu štatistických informácií navrhovaný v [5] je založený na využití skriptovacieho jazyka poskytovaného systémom RouterOS. V tejto práci je však využitý virtualizovaný systém OpenWRT ktorý nahrádza pôvodný operačný systém a teda zber dát prebehne v tomto systéme, ale na podobnom princípe.

Samotný zber dát môže byť riešený viacerými spôsobmi. Jednou možnosťou je sledovanie sieťovej prevádzky na danom rozhraní a následný zber dát pomocou analýzy príchodzích/odchodzích paketov. Toto riešenie je však náročné na výpočtový výkon, zvlášť v prípade kedy je potrebné dáta na smerovači ďalej spracovávať. Ďalšou možnosťou je využitie príjmu informácii od tretej strany v podobe zariadenia monitorujúceho prevádzku v sieti a zasielajúceho správy priľahlým smerovačom. S takým zariadením sa ale v tejto práci nepočíta a preto je využitá posledná možnosť, ktorá spočíva v získavaní štatistických dát o prevádzke priamo od operačného systému.

Jadro operačného systému OpenWRT udržiava informácie o stave systému a procesoch v takzvanom súborovom systéme procesov *(/proc)* . Tieto údaje sú automaticky obnovované a užívateľ má k nim prístup jednak pomocou dostupných objektov alebo prostredníctvom príkazov z príkazového riadku. Vzhľadom na to, že zber informácií o jednotlivých rozhraniach nie je v tomto prípade náročný na procesor, je táto metóda vhodná na implementáciu v smerovači. Takto získané údaje je nutné pred spracovaním ukladať, a to buď do pamäte, alebo fyzicky do súborového systému. Uskladnenie v pamäti je z hľadiska rýchlosti výhodnejšie, ale v prípade získavania dát o dlhodobom zaťažení systému, kde sa následné operácie s dátami vykonajú až po skončení sledovacieho intevalu, by mohli mať negatívny dopad na funkciu smerovača. V práci je preto zvolený prístup s uskladnením dát do textových súborov ktoré sú programovo vytvorené a naplnené. Súbory je nutné čítať a zapisovať na začiatku sledovacieho intervalu, keď dochádza k záznamu počiatočných hodnôt, a na jeho konci, kedy sú hodnoty využívané na výpočet priemerného zaťaženia a získanie objemu prenesených dát či údajov o chybovosti.

3.2 Príkazový riadok OpenWRT a BASH

Príkazový riadok systému linuxového typu, ktorým je aj OpenWRT, je program ktorý vykonáva príkazy čítané zo štandarného vstupu akým je klávesnica alebo súbor. Nazýva sa *shell* a nie je súčasťou samotného jadra ale využíva jeho funkcie. Existuje viacero druhov ako BASH, KSH a TCSH, pričom v tejto práci bude využitý BASH, respektíve jeho zmenšená verzia ASH ktorá je k dispozícii v systéme OpenWRT.

BASH je v podstate prekladač ktorý rozumie nielen zabudovaným príkazom, ale umožňuje aj vlastné programovanie. Je možné vytvárať užívateľské premenné, využívať vetvenie programu či vytváranie smyčiek pomocou štandardných príkazov *if* a *while* , ako aj používať mnoho iných programových koštrukcií známych z iných programovacích jazykov. BASH ďalej umožňuje presmerovávať výstupy/vstupy príkazov, spúšťat iné programy a vytvárať skripty. Bližšie oboznámenie s jeho funkciami a syntaxou je možné napríklad na stránkach manuálu. [1]

Možnosť vytvárania skriptov je využitá aj v tejto práci, pretože umožňuje spojit bežné príkazy príkazového riadku s programovým kódom za účelom vytvorenia funkčného logického celku. Skript sa vykonáva sekvenčne a pri jeho spustení je možné zadať argumenty ku ktorým má samotný program v tele skriptu prístup cez pole argumentov. Jednotlivé navrhnuté skripty sú popísané v ďalšom texte a je možné ich nájsť v prílohe.

3.3 AWK – Jazyk spracovania textu

Ako už bolo spomenuté, navrhovaná aplikácia pracuje s textovými súbormi a je preto nutné používať vhodný spôsob na efektívne získanie už uložených dát. Na tento účel bol zvolený jazyk *awk* prekladaný (interpretovaný) rovnomennou utilitou *awk* ktorá je prítomná v systéme OpenWRT. Dokáže nielen vykonávať zložité textové operácie, ale umožňuje aj využiť matematické operácie nad numerickými dátami. [3]

Zaťial čo programové konštrukcie BASHu sú vcelku všeobecné a nepotrebujú bližšie vysvetlenie, jazyk *awk* sa laikovi môže zdať kryptický. Je preto na mieste vysvetliť základy jazyka ktoré sú nutné na pochopenie navrhnutých skriptov aplikácie ktorou sa zaoberá tento dokument.

3.3.1 Syntax jazyka

Jednoduchý všeobecný awk program má nasledovnú formu:

```
awk <vyhľadávací_vzor > {<program>} vstupný_súbor
```

Program prehľadáva vstupný súbor a pre každý riadok odpovedajúci vyhľadávaciemu vzoru vykoná zadaný program. V tomto programe je možné využívať premenné značené $1 až $n, kde *n* je počet stĺpcov v danom riadku. Stĺpec v tomto prípade reprezentuje skupinu znakov bez medzery (slovo). V tele programu je možné využit viacero funkcií ako napríklad *print,* ktorá pošle na výstup žiadané hodnoty, alebo *split,* ktorá rozdelí záznam daného poľa podľa poskytnutého reťazca. Ďalej je možné testovať podmienky pomocou *if* alebo vykonať matematické operácie v prípade číselnych hodnôt daných polí. Viaceré z týchto funkcií sú použité v samotných navrhnutých skriptoch.

Rozšírená forma programu má nasledovnú štruktúru :

```
awk 'BEGIN                {<inicializácia>}
<vyhľadávací_vzor1 >      {<program>}
<vyhľadávací_vzor2 >      {<program>}
...
END                       {<program>}'
```

Blok *BEGIN* umožňuje vykonať inicalizáciu ešte pred tým než je prehľadaný vstupný súbor. Po inicializácii je podľa jednotlivých vyhľadávacích vzorov prehľadaný súbor a pre každý nájdený riadok je vykonaný patričný blok programu. Na konci sú v bloku *END* uskutočnené dodatočné operácie, napríklad poskladanie výstupného reťazca z hodnôt získaných či vypočítaných v predošlých blokoch.

Na záver je uvedený jednoduchý príklad, ktorý ilustruje niektoré spomenuté vlastnosti programu. V tomto prípade sa z textového súboru *adresy.txt* pošle na výstup druhé slovo z riadku obsahujúceho reťazec "MAC" :

```
awk '/MAC/ { print $2 }' adresy.txt
```

3.4 Implementácia metódy

V rámci navrhnutej metódy sa pracuje s výstupom príkazu *ifconfig* , ktorý je dostupný z príkazového riadku a teda aj zo skriptu. Výstupom príkazu sú informácie o všetkých rozhraniach systému obsahujúce okrem sieťových a fyzických adries aj údaje o celkovom počte prenesených paketov, prenesenom objeme dát a o chybovosti a to ako pre vysielací tak aj pre prijímací smer. V prípade že je pri vyvolaní príkazu zadaný argument predstavujúci názov rozhrania, sú zobrazené len informácie o tomto rozhraní. Táto možnosť je využitá v navrhnutom skripte, kde sa do dočasného súboru tieto informácie uložia a následne spracujú pomocou *awk* programu. Program prehľadáva jednotlivé riadky súboru z ktorých sú získané hodnoty celkového počtu prenesených paketov, celkového počtu prenesených bytov, počtu chýb a počtu zahodených paketov. Tieto údaje sú následne vložené do textového reťazca a zaslané na výstup programu. Ďalším krokom je uloženie dát do súboru, ktorého názov obsahuje názov analyzovaného rozhrania (napr.: eth0.dat). K tomuto súboru je ďaľej možno pristupovať a v prípade zberu dát o dlhodobom zaťažení ich použiť na výpočet priemerných hodnôt počas daného intervalu. Tento proces je popísaný v ďalšom texte.

Samotný skript pre zber informácií o požadovanom rozhraní má názov *get_if_data.sh* a je uvedený a bližšie popísaný v prílohe. Výsledné dáta sú na výstup posielané vo forme jedného riadku a to z dôvodu zjednodušenia ich ďalšieho spracovania (obrázok 3.1).

```
root@OpenWrt:/spsi# ./get_if_data.sh eth0
rxp: 40857 txp: 8372 rxb: 61016501 txb: 480088 rxe: 0 txe: 0 rxd: 0 txd: 0
```

Obr.3.1: Výstup skriptu get_if_data.sh

4 ZBER DÁT PRIEMERNÉHO ZAŤAŽENIA A KONFIGURÁCIA PARAMETROV

4.1 Priemerné zaťaženie sieťových rozhraní

Rozšírenie predošlej metódy o možnosť zberu štatistických dát priemerného zaťaženia jednotlivých rozhraní je otázkou implementácie riadiaceho skriptu obsahujúceho časovač. Parametrami tohoto skriptu sú v takomto prípade názvy jednotlivých rozhraní ktoré chceme monitorovať a časový interval určujúci dĺžku sledovacej doby. V našom prípade sa využíva aj premenná indikujúca počet po sebe nasledujúcich zberov priemerných dát, z ktorých je následne vypočítaná priemerná hodnota za celú sledovanú dobu a výslené dáta sú odoslané okolitým prístupovým bodom.

Medzi sledované informácie o rozhraniach patrí napríklad priemerná hodnota dát prenesených za sekundu či počet chýb alebo zahodených paketov počas daného intervalu. V prípade hodnôt, ktoré sú vyjadrené v závislosti na čase, je nutné brať do úvahy aj vplyv dĺžky sledovacieho intervalu na výsledné dáta. V prípade dlhého intervalu nie je možné detekovať prítomnosť náhleho nárastu hodoty, respektíve špičiek v prevádzke, pretože vysledná hodnota sa rozloží počas tohto intervalu. Príliš krátka doba zas nemusí vyhovovať požiadavkam implementovanej metódy na čas potrebný na vykonanie zberu dát v rámci samotného skriptu, čo môže mať za následok simultánny prístup k datovým súborom a neuspešné pokusy o zápis do týchto súborov.

4.1.1 Implementácia riadiaceho skriptu

V rámci krátkodobého či dlhodobého zberu dát bol navrhnutý skript, ktorý využíva už popísaný skript *get_if_data.sh* a to za účelom zberu informácií o jednom rozhraní. Názov skriptu je *monitor_interfaces.sh* a ako už bolo spomenuté, jeho vstupnými argumentami sú jednotlivé sieťové rozhrania ktoré sa budú monitorovať a dĺžka sledovacieho intervalu. Dĺžka intervalu, je súčasťou konfiguračného súboru. Po spustení skriptu dochádza pomocou cyklu k zapísaniu počiatočných informácií o jednotlivých rozhraniach do súborov. Pre každé rozhranie je vytvorený jeden súbor a ten obsahuje ako počiatočné dáta, tak aj dáta zaznamenané na konci sledovacieho intervalu. Zároveň obsahuje aj samotnú dĺžku tohto intervalu a po skončení procesu zberu sa do neho zapíšu priemerné hodnoty o prenesenom množstve paketov a dát, ako aj absolútne hodnoty chybovosti počas daného intervalu.

Jadrom skriptu je príkaz *sleep* ktorý dočasne odloží proces mimo procesor po počiatočnom zbere dát. Táto doba je reprezentovaná práve dĺžkou sledovacieho intervalu, a po jej skončení sa vykoná konečný zber nových hodnôt a výpočet priemerných. O túto úlohu sa stará *awk* program prehľadávajúci jednotlivé súbory

uložené v podadresári */data*. Dátové súbory sa skladajú z troch riadkov, kde prvý riadok predstavuje počiatočné hodnoty, druhý riadok obsahuje konečné hodnoty a v treťom je zaznamenaná dĺžka intervalu potrebná pre konečný výpočet (obrázok 4.1).

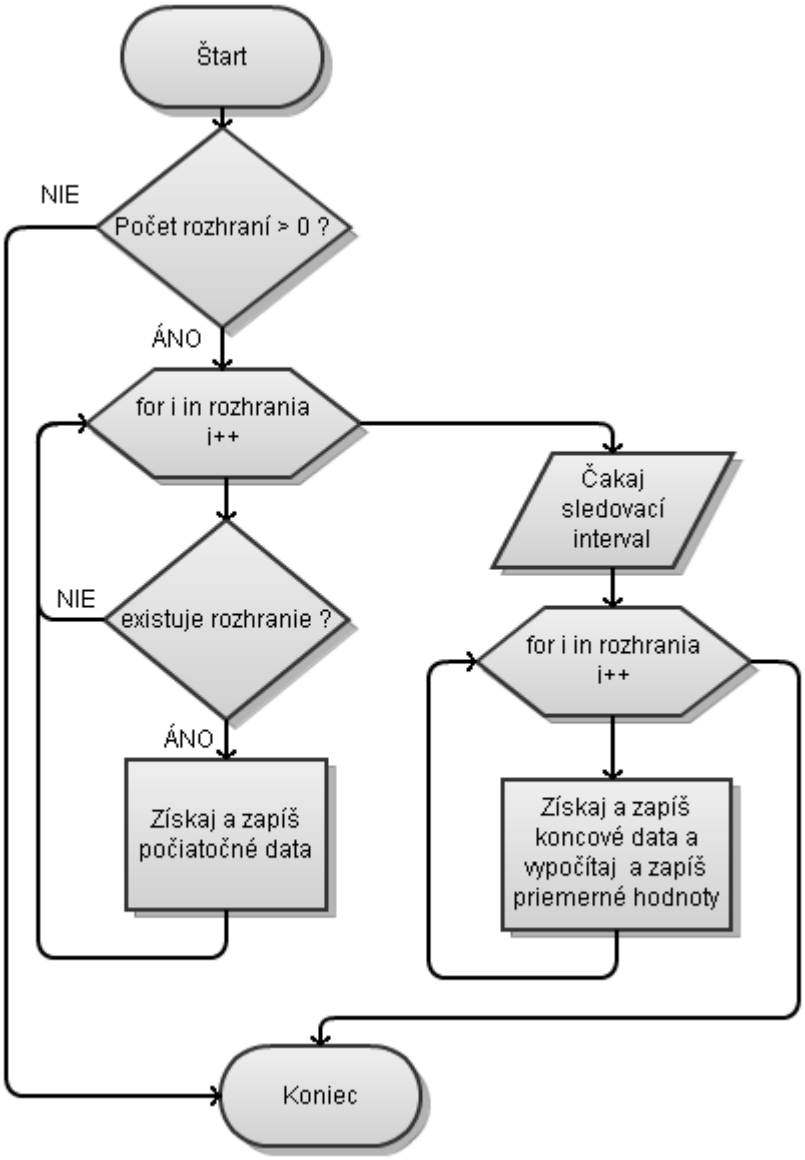
```
root@OpenWrt:/spsi# cat data/eth0.dat
init rxp: 37496 txp: 7795 rxb: 55987623 txb: 447626 rxe: 0 txe: 0 rxd: 0 txd: 0
final rxp: 40857 txp: 8372 rxb: 61016501 txb: 480088 rxe: 0 txe: 0 rxd: 0 txd: 0
secs 60
RXPps: 56.0167
TXPps: 9.61667
RXBps: 83814.6
TXBps: 541.033
RXEpp: 0
TXEpp: 0
RXDpp: 0
TXPpp: 0
```

Obr.4.1: Obsah datového súboru pre rozhranie eth0 po minútovom intervale

Tieto hodnoty sú do súboru zapisované dvojfázovo a to na začiatku a na konci intervalu, pričom v záverečnej fáze sú do súboru dopísané vypočítané priemerné hodnoty. Tieto hodnoty však môžu byť zapísané aj do iného súboru , prípadne inak spracované. Samotný skript je bližšie popísaný v prílohe a jeho vývojový diagram je na obrázku 4.2. V prípade viacnásobného zberu dát počas dlhšej periódy je možné na jeho opakované spúšťanie využívať iný nadradený skript so samostatným časovačom, a práve táto možnosť je využitá a popísaná v ďalšom texte.

4.2 Parametre systému

Aby bol systém dostatočne flexibilný je nutné implementovať metódu, ktorá umožňuje nastavenie vybraných parametrov zberu dát. Tieto parametre sa môžu týkať buď vlastných zbieraných dát, alebo nastavenia vlastností samotného programu. Do prvej kategórie patrí možnosť nastavovať typ zachytených dát, a teda určiť či sa má sledovať chybovosť, počet prenesených paketov alebo bitov a podobne. Do druhej kategórie spadá napríklad nastavenie dĺžky intervalu celkového zberu dát, nastavenie dĺžky intervalu medzi jednotlivými vzorkami, alebo frekvencia zasielania správ k susedným prístupovým bodom.



Obr.4.2: Vývojový diagram skriptu *monitor_interfaces.sh*

4.2.1 Metódy nastavenia parametrov

Samotné nastavenie parametrov programu je možné vykonať viacerými spôsobmi. Z programového hľadiska je narýchlejšie a najjednoduchsie nastavovať parametre formou argumentov pri spúštaní programu na príkazovom riadku. V takomto prípade by bolo nutné v rámci programu pred samotným zberom dát parametre vyčítať z poľa argumentov spusteného riadiaceho skriptu. Nevýhodou riešenia je však jeho neprehľadosť a neprenosnoť nastavenia. V konečnom dôsledku by bolo riešenie zároveň nepohodlné, pretože zadávané parametre by buď museli byť v dopredu známom usporiadaní ktoré by slúžilo ako referencia pri ich zadávaní, alebo by bolo potrebné v tele programu celé pole prehľadávať a zisťovať prítomnosť jednotlivých parametrov a ich nastavení. Okrem toho zadávanie niekoľkých desiatok hodnôt na príkazovom riadku je nepraktické.

Druhá metóda spočíva v použití konfiguračných súborov uložených v systémovom súbore zariadenia. Tento spôsob umožňuje prehľadné upravovanie konfigurácie a tiež možnosť zálohovať ju, alebo ju preniesť do iného prístupového bodu. Nevýhodou prístupu je nutnosť v tele programu čítať a spracovávať textové súbory, čo sa však prejaví len pri spustení programu, ale po uložení hodnôt do pamäte nemá ďalej vplyv na jeho priebeh. Metóda zároveň poskytuje možnosť aplikovať niektoré z viacerých predošlých uložených konfigurácií, a to využitím argumentu pri spustení programu. Argumentom by v takomto prípade bola cesta ku konfiguračnému súboru. Vzhľadom na spomenuté pozitívne vlastnosti je v programe zvolená práve táto metóda.

4.2.2 Konfigurácia pomocou textových súborov

Konfigurácia parametrov programu je uložená v súbore nazvanom *config* a jeho obsah je uvedený v prílohe. Každý parameter je uložený na samostatnom riadku a súbor bol navrhnutý tak, aby bolo čítanie jednotlivých hodnôt ľahko vykonateľné pomocou programu *awk*. Obsahom súboru je multicastová IP adresa a príslušný port, ktoré sú využívané programom medziprocesovej komunikácie za účelom zasielania správ medzi jednotlivými prístupovými bodmi. Konfigurácia ďalej obsahuje údaj o dĺžke intervalu zberu v sekundách, ktorý je použitý na zber počiatočných a koncových hodnôt, a teda aj na výpočet priemerného zaťaženia rozhraní počas daného intervalu, a údaj o počte zberov dát na rozhraní, ktoré majú byť vykonané pred tým než sa nazbierané dáta odošlú susedným prístupový bodom. V neposlednom rade súbor obsahuje informácie o tom, ktoré z parametrov sa majú v správach k susedným bodom prenášať. Jedná sa o údaj o počte prijatých a odoslaných paketov za sekundu, počte prijatých a odoslaných bajtov za sekundu, počte chýb zaznamenaných na danom rozhraní počas sledovacieho intervalu a počte zahodených paketov na rozhraní. Konfiguračný súbor je samozrejme možné rozšíriť, avšak v takom prípade by bola nutná i úprava skriptov. Samotné využitie konfiguračného súboru a čítanie konfigurácie je popísané v kapitole 6.

5 PROGRAMY PRE ODOSIELANIE A PRIJÍMANIE SPRÁV

V rámci vytvorenia systému, ktorý by plnil funkciu automatického dohľadu nad WLAN sieťou, je nutné implementovať funkciu odosielania a prijímania správ jednotlivými prístupovými bodmi ktoré sú súčasťou tejto siete. Správy môžu obsahovať buď štatistické dáta sieťovej prevádzky získané na jednotlivých prístupových bodoch, alebo napríklad môžu slúžiť aj na prenos konfigurácie v prípade automatického pripojenia do multicastovej skupiny.

V prvej fáze návrhu programu, ktorý zabezpečí prenos krátkych správ medzi prístupovými bodmi, je potrebné zvoliť vhodný programovací jazyk a nástroje. V tejto práci je zvoleným jazykom jazyk *C*, a za účelom medziprocesovej komunikácie program využíva programové konštrukty nazývané sokety. Soket je v podstate jeden z koncov komunikačného kanálu a každý z komunikujúcich procesov vytvára vlastný soket. Jedna komunikujúca strana má rolu klienta a druhá pracuje ako server. Programové rozhranie pre sokety poskytuje viacero funkcií na vytvorenie soketu, odosielanie dát a prijímanie dát, ktoré budú bližšie popísané v ďalšom texte v rámci návrhu samotného programu.

5.1 Kompilácia programov pre OpenWRT

Jednotlivé programy nie je vhodné vyvíjať natívne v prístupovom bode, pretože medzi obmedzenia takéhoto prístupu patrí nedostatok miesta, pamäte a nižšia rýchlosť jeho procesora. S týmito obmedzeniami súvisí aj neprítomnosť kompiléra v operačnom systéme prístupového bodu. Preto bude využitá metóda takzvanej krížovej-kompilácie (cross-compilation) pri ktorej sa program vyvinie na plnohodnotnom operačnom systéme s linuxovým jadrom a následne prenesie do prístupového bodu.

5.1.1 Vytvorenie SDK

Architektúra OpenWRT sa líši od architektúry bežného linuxového systému a je preto nutné v hostiteľskom systéme nainštaľovať balíček na vývin programov pre Open WRT – OpenWRT SDK. Tento balíček je možné stiahnuť z repozitára OpenWRT, ale vzhľadom na vyšší počet rôznych architektúr a verzií systému je vhodnejšie vytvoriť si vlastné SDK. Zvlášť to platí pre vývoj programov pre prístupový bod Mikrotik, kde skupina stojaca za vývojom systému poskytuje opravu (patch), ktorú je potrebné aplikovať na zdrojové kódy OpenWRT pred jeho kompiláciou a prenosom do prístupového bodu. Tento súbor je možné nájst na stránkach Mikrotiku v sekcii virtualizácia [2] .

Prvým krokom k vytvoreniu SDK je získanie zdrojových kódov OpenWRT, ktoré je možné, po nainštaľovaní *subversion* v hostiteľskom systéme, získať príkazom:

```
svn co svn://svn.openwrt.org/openwrt/trunk
```

Následne je nutné získať a aplikovať spomínanú opravu pre Mikrotik zariadenie:

```
cd trunk/
wget http://www.mikrotik.com/download/metarouter/openwrt-metarouter-
1.2.patch
patch -p0 <openwrt-metarouter-1.2.patch
```

Takto upravené zdrojové kódy môžeme skompilovať pomocou *make*. Pred samotnou kompiláciou je však nutné spustiť *make menuconfig* a zvoliť vhodnú architektúru cieľového zariadenia a tiež aj zvoliť voľbu vytvorenia SDK. Pre RB433 je cieľová architektúra *mipsbe*. Po úspešnej kompilácii sa SDK nachádza v adresári */bin* ako skomprimovaný súbor, ktorý je potrebné pred použitím rozbaliť.

5.1.2 Kompilácia programu a inštalácia v prístupovom bode

Zdrojový súbor programu je potrebné umiestniť do adresárovej štruktúry SDK a to do adresára :

```
/package/<nazov_programu>/src/
```

V tomto adresári je tiež potrebné vytvoriť štandardné Unixové *Makefile*, čo je súbor obsahujúci inštrukcie pre kompiláciu programu. Ďalej je v aresári o úroveň vyššie nutné vytvoriť *Makefile* súbor, ktorý bude SDK používať pri vytváraní balíčku pre náš cieľový systém. Tento súbor obsahuje popis vytváraného balíčku, destinácie pre vytvorené súbory a spôsob inštalácie v prístupovom bode. Použité súbory je možné nájsť v prílohe práce.

Kompiláciu je možné spustiť z koreňového adresára SDK použitím príkazu *make*. Po úspešnej kompilácii je potrebné súbor preniesť do virtualizovaného OpenWRT prostredníctvom FTP protokolu alebo ho umiestniť na web a stiahnuť pomocou *wget*. Keďže je program pripravený ako balíček, je možné ho nainštaľovať príkazom:

```
opkg install <nazov_programu>
```

Spustitelné súbory programu inštalátor umiestni do adresára určeného súborom *Makefile* použitého pri kompilácii pomocou SDK.

5.2 Programové rozhranie pre sokety

Ako už bolo spomenuté, soket je v podstate jeden z koncov komunikačného kanálu, pričom každý z komunikujúcich procesov vytvára vlastný soket. Komunikácia je

založená na princípe client-server. Na klientskej strane je vytvorenie spojovo orientovaného soketu nasledovné :

1. Vytvorenie soketu systémovým volaním *socket()*
2. Pripojenie soketu na adresu serveru použitím volania *connect()*
3. Príjem a vysielanie dát pomocou volaní *read()* a *write()*

Na strane serveru je vytvorenie soketu podobné :

1. Vytvorenie soketu systémovým volaním *socket()*
2. Spojenie soketu s adresou použitím volania *bind()*. Adresa obsahuje číslo portu hostiteľskej stanice
3. Naslúchanie spojeniam pomocou volania *listen()*
4. Prijatie spojenia pomocou volania *accept()*. Typicky toto volanie vyvolá blokovanie pokiaľ sa klient nespojí so serverom
5. Príjem a odosielanie dát

Pri vytvorení soketu je nutné zadať adresnú doménu a typ soketu, pričom komunikovať navzájom môžu len procesy používajúce rovnaký typ soketu a adresnú doménu. Procesy komunikujúce v rámci jedného súborového systému používajú Unixovú doménu a procesy komunikujúce v rámci sieťe používajú Internetovú doménu. Adresy soketov v Unixovej doméne sú tvorené reťazcom znakov, ktorý je zapísaný v súborovom systéme. Adresy Internetovej domény sa skladajú z IP adresy hostiteľského zariadenia a čísla portu patriacemu danému procesu. Sokety ďalej môžu byť takzvané prúdové sokety, ktoré komunikujú spojovo orientovaným spôsobom a využívajú protokol TCP, alebo datagramové sokety ktoré využívajú nespojovaný protokol UDP. Pri použití tohoto UDP soketu klient nevytvára spojenie so serverom ale namiesto toho proste zašle datagram použitím *sendto* funkcie, ktorej parametrom je adresa cieľového servera. Server zas neprijíma spojenia od klinetov ale namiesto toho volá funkciu *recvfrom* ktorá čaká, pokým nie sú prijaté dáta od klienta. Funkcia vracia protokolovú adresu klienta, spolu s prijmutým datagramom, vďaka ktorej môže server poslať klientovi odpoveď. [9]

Rozhranie pre programovanie aplikácií (Application programming interface - API) umožňuje prácu so sieťovými adresami vo forme štruktúr a väčšina príslušných volaní očakáva adresu práve v takomto tvare. Pre reprezentáciu adries v rámci protokolu IPv4 slúži štruktúra *sockaddr_in* :

```
struct sockaddr_in {
  uint8_t         sin_len;      // dĺžka štruktúry
  sa_family_t     sin_family;   // typ adresy (AF_INET)
  in_port_t       sin_port;     // 16-bitové číslo TCP alebo UDP portu
  struct in_addr  sin_addr;     // 32-bitová IPv4 adresa
  char            sin_zero[8];  // nepoužité
};
```

Štruktúra obsahuje polia pomocou ktorých je možné pristupovať k typu adresy, portu a k samotnej adrese zariadenia ktorá je uložená v štruktúre

```
struct in_addr {
  in_addr_t   s_addr;           // 32-bitová IPv4 adresa
};
```

Na vytvorenie soketu sa používa volanie

```
int socket (int domain, int type, int protocol);
```

ktorého návratovou hodnotou je takzvaný deskriptor soketu. Prostredníctvom tejto hodnoty je ďalej možné so soketom pracovať. Jedným z parametrov volania je doména, ktorá určuje spôsob komunikácie procesov. Pre použitie IPv4 sa doména určí konštantou *AF_INET*. Parameter typ určuje typ soketu, *SOCK_DGRAM* pre datagramovú službu, a protokol môže byť TCP,UDP alebo SCTP. Konštanta *IPPROTO_UDP* určuje UDP protokol (v prípade že je hodnota nastavená na 0 je protokol určený systémom podľa typu soketu).

Na spojenie adresy servera a portu s vytvoreným soketom sa používa volanie

```
int bind (int sock_dsc, const struct sockaddr *serv_addr, socklen_t addrlen);
```

ktoré pracuje s deskriptorom soketu *sock_dsc*. Cieľová, či zdrojová adresa zariadenia je v rámci API určená štruktúrou typu *sockaddr_in*. Ďalšími z parametrov volania sú práve ukazatel na takúto adresu, ktorá reprezentuje adresu servera, a veľkosť tejto štruktúry. V ďalšom texte budú využité volania na odosielanie a prijem dát, ako aj iné pomocné funkcie ktoré API poskytuje. Toto programové rozhranie je vcelku rozsiahle a možné sa s ním bližšie oboznámiť napríklad v [9].

5.3 Voľba transportného protokolu

Programové rozhranie pre sokety umožňuje využitie ako TCP tak aj UDP transportného protokolu. Pri voľbe je nutné zohladniť charakter vyvíjaného programu a tiež aj typ správ ktoré bude potrebné prenášať. Aplikácia, ktorou sa zaoberá tento text, podporuje prenos správ multicastovým prenosom a prenášané správy sú pomerne krátke. Z charakteru TCP protokolu vyplýva, že ním poskytovaná spojovo orientovaná služba takejto aplikácii nevyhovuje. Pri prenose krátkych správ viacerím cieľovým staniciam by bolo potrebné pre každú z nich vytvoriť spojenie, čo do celého procesu prináša nadbytočnú réžiu. Na druhej strane UDP protokol poskytuje nespoľahlivú službu, avšak tento nedostatok je možné čiastočne nahradiť zasielaním potvrdenia prijatej správy cieľovou stanicou a tiež aj využívaním sekvenčného čísla v prípade nutnosti zasielania správy vo viacerých paketoch. Vzhľadom na tieto fakty je v tejto práci ako transportný protokol zvolený protokol UDP.

Pri vytvorení soketu je použitý protokol určený buď pomocou parametra vo funkcii *socket()*, alebo pri nastavení tejto hodnoty do nuly ho systém určí sám na základe typu použitého soketu, ktorý je tiež jeden z parametrov funkcie. Protokol UDP je použitý pri

sokete typu *SOCK_DGRAM,* a priamo určiť protokol je možné pomocou konštanty *IPPROTO_UDP*.

5.4 Využitie multicastového prenosu

Vzhľadom na fakt, že zasielané správy od jednotlivých prístupových bodov bude potrebné spracovať vo viacerých, prípadne všetkých ostatných prístupových bodoch, je vhodné využiť namiesto unicastu alebo broadcastu práve multicastové vysielanie. Takéto vysielanie spočíva vo vysielaní paketov viacerým cieľovým staniciam, pričom dáta zo zdroja sú kopírované v jednotlivých sieťových segmentoch podľa potreby a doručované staniciam.

5.4.1 Multicastové adresy

Multicastová adresa umožňuje adresáciu k viacerím cieľovým staniciam, ktoré sú členmi multicastovej skupiny a kombinácia multicastovej IP adresy a UDP portu tvorí takzvanú reláciu. V IPv4 je pre multicastové adresy vyhradený adresný priestor v rozsahu 224.0.0.0 do 239.255.255.255. Dolných 28 bitov adresy slúži ako identifikátor multicastovej skupiny a celá 32 bitová adresa sa nazýva skupinová adresa. Multicastové IP adresy sú mapované na Ethernetové multicast adresy a to takým spôsobom, že horých 24 bitov Ethernetovej adresy má vždy hodnotu 01:00:5e, nasledujúci bit je 0 a zvyšných 23 bitov je tvorených dolnými 23 bitmi mapovanej IP adresy. Rámec s takouto adresou je ignorovaný stanicami, ktoré nie sú členmi žiadnej multicastovej skupiny. Je to z toho dôvodu, že rozhranie nie je nastavené na prijímanie multicastu a linková adresa nemá tvar broadcastovej ani unicastovej adresy patriacej danému rozhraniu. [1]

5.4.2 Použitie multicastu v programe

Aby mohol proces prijať multicastový datagram, je potrebné aby sa pridal do multicastovej skupiny a zároveň vykonal spojenie UDP soketu s číslom portu, ktorý bude slúžiť ako cieľový port pre datagramy zaslané do tejto skupiny. Pripojenie do skupiny pripraví sieťovú a linkovú vrstvu stanice na príjem takýchto datagramov a spojenie portu so soketom umožní procesu tieto datagramy získavať. Je možné tiež s daným soketom prepojiť aj multicastovú adresu skupiny, čo zabezpečí príjem len multicastových datagramov danej skupiny.

Programové API pre sokety poskytuje spôsob ako nastaviť vlastnosti soketu, prostredníctvom ktorého sa budú získavať dáta zaslané v rámci multicastovej skupiny. Na zaslanie informácii systémovému jadru sa využíva volanie

```
int setsockopt(int socket_dsc, int level, int optname, const void* optval,
int optlen);
```

ktorého návratová hodnota je 0 v prípade úspechu a -1 v prípade neúspechu. Parametrami volania sú deskriptor soketu, úroveň identifikujúca vrstvu ktorej sa nastavenie týka, idetifikátor voľby, hodnota voľby a jej dĺžka. Úroveň môže byť všeobecná pre sokety alebo sa môže týkať konkrétneho protokolu ako IPv4 alebo TCP. Pre multicast sa úroveň nastaví konštantou IPPROTO_IP a voľby týkajúce sa multicastu sú stručne uvedené v tabuľke 5.1.

IP_MULTICAST_LOOP	Zapína/vypína preposielanie odoslaných dát na lokálne rozhranie
IP_MULTICAST_TTL	Určuje veľkosť TTL odoslaných datagramov
IP_MULTICAST_IF	Udáva výstupné rozhranie pre daný soket
IP_ADD_MEMBERSHIP	Umožňuje pripojenie do multicastovej skupiny
IP_DROP_MEMBERSHIP	Umožňuje odhlásenie z multicastovej skupiny

Tab.5.1: Voľby pre volanie *setsockopt()* týkajúce sa multicastu

Pred samotným prihlásením sa do multicastovej skupiny je potrebné vyplniť štruktúru typu *ip_mreq,* ktorá obsahuje adresu multicastovej skupiny a IP adresu lokálneho rozhrania. Štruktúra má nasledovný tvar

```
struct ip_mreq
{
     struct in_addr imr_multiaddr; //IP adresa multicastovej skupiny
     struct in_addr imr_interface; //IP adresa lokálneho rozhrania
};
```

a vo volaní je pri použití voľby *IP_ADD_MEMBERSHIP* nutné ako parameter *optval* použiť ukazateľ na vyplnenú instanciu tejto štruktúry. Použitie je možné vidieť v zdrojovom kóde programu pre server. Využitie multicastového prenosu v programoch je otázkou nastavenia volieb soketu a zmeny typu adries. Tieto nastavenia sa v podstate týkajú len programu servera, pretože klient prijíma len správu z unicastovej adresy vo forme odpovede zo servera.

5.5 Program server

Zdrojový kód navrhnutého programu slúžiaceho ako server je miestnený v súbore *stats_server.c*. Úlohou servera je vytvoriť socket datagramového typu, prihlásiť sa do multicastovej skupiny a čakať na prichádzajúce spojenie na danom porte. Po prijatí spojenia dochádza k čítaniu typu správy, ktorý je zaznamenaný na začiatku správy ako reťazec troch znakov. Tento program očakáva správu typu ‚*upd*'(update – aktualizácia), ktorá obsahuje dáta získané v odosielajúcom prístupovom bode. Následne je jeho úlohou tieto dáta spracovať a teda zapísať do súboru, ktorého názov tvorí IP adresa odosielajúceho zariadenia. Funkcia a hodnoty jednotlivých volaní programu sú zrejmé

z komentárov v zdrojovom texte. Niektoré z funkcií, ktoré priamo súvisia s programovaním soketov, je ale vhodné bližšie vysvetliť.

V úvode programu dochádza k čítaniu multicastovej adresy a čísla portu z poľa argumentov programu. Následne je naplnená štruktúra adresy pre použitie v ďalšom programe a dochádza k príprave soketu pre príjem. Tento krok pozostáva z vytvorenia samotného soketu volaním *socket()* a jeho prepojením s pripravenou multicastovou adresou pomocou volania *bind()*. Pripojenie do multicastovej skupiny je dosiahnuté volaním *setsockopt()* ktoré bolo popísané v predošlej kapitole.

Jadrom programu je nekonečná smyčka *while*, ktorá zaistí sekvenčné prijímanie správ na zadanom sokete. Program využíva volanie

```
ssize_t recvfrom(int sock_dsc, void *buff, size_t nbytes, int flags,
struct sockaddr *from, socklen_t *addrlen);
```

ktoré slúži na príjem správy. Výstupom volania je počet prečítaných bytov. Parametrami sú deskriptor soketu, ukazateľ na adresu kam sa majú zapisovať prijímané dáta, maximálny počet zapísaných či prečítaných bytov, príznaky a ukazatele na cieľovú a zdrojovú sieťovú adresu. V prípade úspešného prijatia správy má server možnosť zaslať odpoveď na adresu klienta, ktorej štruktúra je naplnená volaním *recvfrom()*.

Volanie blokuje proces dovtedy, kým nedôjde k príjmu znakov kedy sa vykoná ich načítanie do zásobníka. Pomocou volaní *fopen()* a *fprintf()* je vytvorený a naplnený súbor pre zdrojovú IP adresu:

```
inet_ntop(AF_INET, &(client_addr.sin_addr), sender_IP,
            INET_ADDRSTRLEN);            //prevod IP adresy
char filename[30];                       //pole pre nazov suboru
sprintf(filename, "data/%s",sender_IP);  //vytvorenie nazvu suboru
FILE *file;                              //ukazatel na subor
file=fopen(filename, "w"); //vytvorenie a otvorenie suboru
fprintf(file,"%s",buffer); //zapis prijatych dat zo zasobniku
```

Jeho obsahom sú teda posledné známe dáta nazbierané na zdrojom prístupovom bode. Na získanie IP adresy odosielateľa v čitateľnom formáte je použitá funkcia

```
char *inet_ntop(int af, const void *restrict src,char *restrict dst,
      socklen_t size);
```

ktorá prevádza numerickú adresu uloženú v poli *src* na textový reťazec v tvare xxx.xxx.xxx.xxx . Parameter *af* udáva verziu IP adresy a *dsc* je ukazateľ na pole kam sa má získaný reťazec uložiť.

5.6 Program klient

Program klient slúži na odoslanie správy na vopred známu adresu servera, ktorá je daná štruktúrou *server_addr* . Zdrojový súbor programu má názov *stats_client.c* a jeho obsah je uvedený v prílohe. Jedinou hlavnou odlišnosťou od programu servera je to, že pri príprave soketu sa nevolá volanie *bind()*. Na naplnenie štruktúry adresy servera program využíva volanie

```
int inet_aton(const char *strptr, struct in_addr *addrptr);
```

ktoré sa stará o prevod adresy z ASCII reťazca do binárneho tvaru vhodného na použitie v sieti, a následne túto adresu uloží do štruktúry na ktorú ukazuje parameter *addrpntr*.

Na rozdiel od servera klientský program neobsahuje smyčku, pretože program sa volá za účelom jednorázového odoslania správy multicastovej skupine. Správa sa vytvorí kopírovaním obsahu dátového súboru */data/recent_data.dat* do zásobníku a využitím volania

```
ssize_t sendto(int sock_dsc, const void *buff, size_t nbytes, int flags, const struct sockaddr *to, socklen_t addrlen);
```

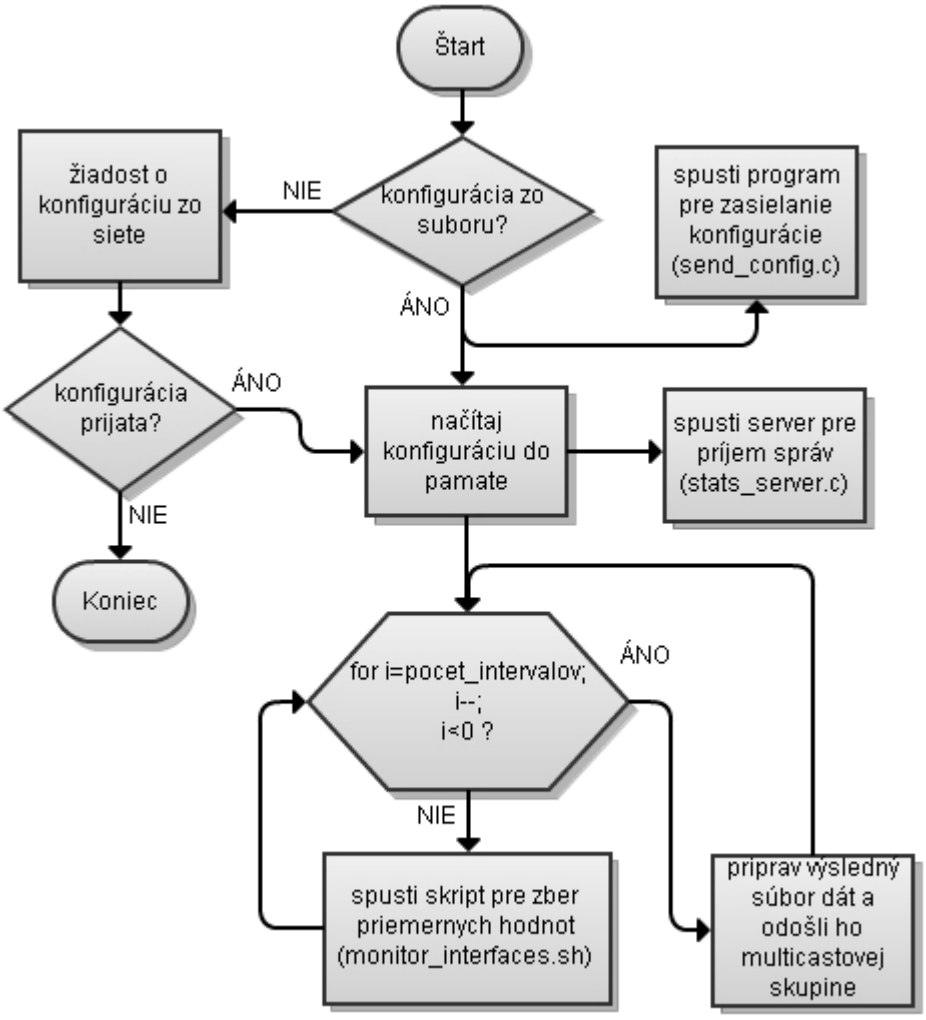
ktoré slúžia na samotné odoslanie (*sendto*) správy. Parametre volania sú prakticky zhodné s už popísaným volaním pre príjem s tým rozdielom, že štruktúra adresy obsahuje cieľovú adresu.

6 DISTRIBÚCIA KONFIGURÁCIE A ŠTATISTÍK A ICH VYHODNOTENIE

Záverečnou fázou návrhu systému je spojenie jednotlivých programov a skriptov do funkčného celku. Za týmto účelom bol navrhnutý skript, ktorého úkolom je buď načítať, alebo získať konfiguráciu systému a volať jednotlivé skripty a programy podľa potreby. Skript má názov *stats_manager.sh* a ďalší text sa bude zaoberať jeho implementáciou.

6.1 Manažér skript

Skript pozostáva z dvoch hlavných častí – bloku konfigurácie a bloku spracovania dát a odosielania správ a jeho vývojový diagram je zobrazený na obrázku 6.1.



Obr.6.1: Vývojový diagram skriptu *stats_manager.sh*

Argumentmi skriptu sú jednotlivé rozhrania zariadenia, ktoré je potrebné sledovať. V úvode skript kontroluje správnosť zadaných rozhraní a ich existeciu a to pomocou *ifconfig* súboru. V prípade, že bolo zadané aspoň jedno rozhranie, prikročí sa k načítaniu konfigurácie. Spúštanie programu v rámci väčšej siete bolo zjednodušené umožnením zaslať požiadavku na zaslanie konfigurácie a jej následný príjem. Toto je dosiahnuté využitím broadcastového vysielania, kde o odoslanie požiadavky sa stará program *get_config* , a naopak zasielanie konfigurácie má na starosti program *send_config*, ktorý pracuje ako server. Po načítaní konfiguračného súboru dochádza k neustálemu zberu aktuálnych priemerných hodnôt parametrov rozhraní a tieto dáta sú ukladané do príslušných súborov a v istých intervaloch, daných konfiguráciou, zasielané ostatnýcm členom multicastovej skupiny. Automatické spustenie skriptu, napríklad po reštarte systému, je možné vykonať napríklad zahrnutím skriptu v súbore *etc/rc.local* v súborovom systéme operačného systému.

6.2 Načítanie konfigurácie

Typ načítania konfigurácie je daný buď absenciou konfiguračného súboru v adresári programu, alebo nastavením hodnoty *#k* v konfiguračnom súbore. V prípade že sa hodnota rovná reťazcu *„file“* , vykoná sa načítanie konfigurácie zo súboru. Zároveň sa predpokladá že aktuálna konfigurácia bola manuálne nastavená, a teda je žiadúca, a spustí sa program *send_config* , ktorý umožní distribúciu tejto konfigurácie k ostatným prístupovým bodom lokálnej siete. V prípade, že je *#k* nastavené na inú hodnotu, alebo konfiguračný súbor chýba, spustí sa program *get_config* , ktorý odošle požiadavku na broadcastovú adresu 255.255.255.255 a čaká na príjem konfigurácie.

Na získanie nakonfigurovaných hodnôt je použitých viacero blokov programu *awk* , ktoré príslušné hodnoty parametrov ukladajú do premenných skriptu.

6.2.1 Program send_config.c

Program je typu server a teda je založený na predošlom programe *stats_server*. Rzodiel je ale v nastavení soketu pre príjem dát. Zariadenie, ktoré vysiela žiadosť o konfiguráciu nepozná adresu multicastovej skupiny, a teda v programe servera je nemožné využívať soket nakonfigurovaný pre multicastový príjem. Pri nastavení soketu je teda namiesto multicastovej adresy použitá konštanta *INADDR_ANY* , ktorá udáva, že sa budú prijímať správy na ktoromkoľvek rozhraní zariadenia. Program zároveň daný soket využíva aj na príjem správ a aj na odosielanie konfigurácie. Čítanie konfiguračného súboru a odosielanie dát do soketu je takmer identicke s postupom popísaným v kapitole 5.6.

6.2.1 Program get_config.c

Program je typu klient a preto je nemálo podobný už prebranému *stats_client* . Vzhľadom na to, že využíva odosielanie správ na broadcastovú adresu, je nutné na tento fakt upozorniť operačný systém, aby bolo vôbec možné správu odoslať. Toto je dosiahnuté prostredníctvom volania *setsockopt()* :

```
// nastavenie moznosti zasielat pakety na broadcastovu adresu
int optval = 1;
setsockopt(socket_dsc, SOL_SOCKET, SO_BROADCAST, &optval,
      sizeof(optval));
```

Volanie slúži na bližšie nastavenie soketu, v tomto prípade je využitá konštanta *SO_BROADCAST*, ktorá pri nastavení hodnoty 1 udáva, že na danom sokete je možné vysielať datagramy na broadcastovú adresu. Parameter *socket_dcs* predstavuje deskriptor nastavovaného soketu a *optval* obsahuje nastavovanú hodnotu parametru. Aj v tomto prípade je na vysielanie a prijímanie použitý jeden soket. Program konfiguráciu neukladá priamo do súboru, ale prijaté dáta odošle na výstup, kde ich odchytí nadradený skript, ktorý sa postará o správne vytvorenie súboru.

6.3 Odosielanie a príjem dát

Po úspešnom načítaní konfigurácie dochádza k spusteniu serveru pričom proces je spustený a odoslaný do pozadia. Skript následne získa jeho identifikátor pomocou ktorého môže overiť či bol skutočne spustený :

```
#---- SPUSTENIE PROGRAMU SERVER
echo "Spustam server na multicastovej adrese ${mc_ip} ..."
#spustenie serveru v pozadi, ulozenie jeho PID
./stats_server ${mc_ip} ${port} &
PID=$!
```

Vo volaní programu parametre *mc_p* a *port* predstavujú multicastovú ip adresu a port získané z konfiguračného súboru. Program server beží v pozadí a v prípade že obdrží správu s dátami tak ju spracuje podľa postupu uvedeného v kapitole 5.5.

K zberu dát a odosielaniu správ okolitým prístupovým bodom dochádza v nekonečnej smyčke *while*. Zber je rozdelený na dve fázy. V prvej fáze sú pre každé rozhranie a nakonfigurované parametre získané krátkodobé priemerné hodnoty dát. Údaje sú zapisované do patričných súborov, kde každému rozhraniu patrí jeden súbor. Dlhodobejšie hodnoty sú získané opakovaním zberu, pričom počet opakovaní udáva parameter *#u* v konfiguračnom súbore. Pre každý zber sa v súbore rozhrania vytvorí nový riadok s priemernými hodnotami a na konci tohto viacnásobného zberu dochádza

k čítaniu hodôt a získaniu dlhodobejšieho priemeru. Obsah súboru po viacnásobnom zbere vyzerá nasledovne :

```
root@OpenWrt:/statistics/data# cat eth1_total.dat
9.5 3.5 822 223.5 0 0 0 0
4.2 2.1 431 103.7 0 0 0 0
7.2 5.3 622 189.1 0 0 0 0
12.8 8.2 987 246 0 0 0 0
```

Hodnoty sú zoradené v poradí v akom sú používane v celom programe a teda nie je potrebné zbytočne do súboru zapisovať o ktoré parametre prevádzky sa jedná. Prvé dve hodnoty predstavujú prijaté a odoslané pakety, druhé dve prijaté a odoslané bajty, ďalšie dve obsahujú hodnoty o chybovosti a posledné dve o statovosti paketov. Takýto spôsob zápisu umožňuje následné spracovanie programom *awk*, ktorý pre každý stĺpec vypočíta priemernú hodnotu, ktorú zapíše do súboru *data/recent_data.dat* , ktorý je spoločný pre všetky rozhrania a kde každému rozhraniu patrí práve jeden riadok :

```
root@OpenWrt:/statistics/data# cat 192.168.2.2
upd
eth1 8.5 4 423 251 0 0 0 0
```

Súbor zároveň tvorí správu, ktorá je pravidelne odosielaná ostatným členom multicastovej skupiny. Z toho dôvodu obsahuje v prvom riadku príznak „upd“, ktorý príjemcovi signalizuje že sa jedná o aktualizáciu nazbieraných hodnôt. Výber odosielaných parametrov podľa konfigurácie vykonáva nasledovný blok programu *awk:*

```
awk -v t=$interval_t -v i=$i -v xp=$xp -v xb=$xb -v xe=$xe -v xd=$xd
'{
      if (xp==1){RXP+=$1;TXP+=$2;}
      if (xb==1){RXB+=$3;TXB+=$4;}
      if (xe==1){RXE+=$5;TXE+=$6;}
      if (xd==1){RXD+=$7;TXD+=$8;}
      }
END{
      if (xp==1){printf "%s %s ", RXP/t,TXP/t;}
      if (xb==1){printf "%s %s ", RXB/t,TXB/t;}
      if (xe==1){printf "%s %s ", RXE,TXE;}
      if (xd==1){printf "%s %s ", RXD,TXD;}
}' data/${i}_total.dat >> data/recent_data.dat
```

Vstupnými parametrami bloku sú, okrem časového intervalu sledovania a názvu rozhrania, práve jednotlivé parametre zberu, podľa ktorých program but číta dané hodnoty, albo ich vynecháva. Po uložení dát dochádza k vyslaniu správy sputením programu *stats_client* s vhodnými parametrami :

```
# spustenie klientskeho programu na odoslanie datoveho
# suboru multicastovej skupine
echo "Zasielam spravu na multicast adresu ${mc_ip}.."
```

```
./stats_client ${mc_ip} ${port} &
```

Nasledujúci vypis znázorňuje funkciu programu po spustení manažéra pre rozhranie dané parametrom, kde vidno všetky popísané fáze programu :

```
root@OpenWrt:/statistics# ./stats_manager.sh eth0
Monitorovane rozhrania:  eth0
Spustam server na multicastovej adrese 239.255.1.1 ...
Server spusteny (PID:4252).
Spustam zber dat s parametrami:
  sledovaci interval: 2s
  pocet zberov pred zaslanim spravy: 4
  zber informacii: xp=1 xb=1 xe=1 xd=1
Prebieha zber dat..
[server]Nastavena adresa: 239.255.1.1 Nastaveny port: 50000
Zasielam spravu na multicast adresu 239.255.1.1..
Prebieha zber dat..
[client]Odosielam data..
[client]Sprava odoslana.
```

6.4 Návrh procesu voľby suseda pre handover

Pri voľbe vhodného suseda na handover je potrebné zohľadniť viacero parametrov prevádzky. Aktuálne zaťaženie rozhraní, úroveň prijímaného signálu či chybovosť majú rôzny vplyv na kvalitu linky. Z toho dôvodu je vhodné zaviesť systém váh, kedy každému parametru prevádzky je priradená číselná hodnota ktorou je aktuálna hodnota parametru násobená a následne použitá pri výbere kandidáta.

6.4.1 Implementácia skriptu

Skript navrhnutý za účelom výberu vhodného kandidáta na handover má názov *select_for_handover.sh* a je uvedený v prílohe. Skript nie je súčasťou vyššie uvedeného programu a to z toho dôvodu, že zber údaju o sile signálu, a všeobecne zber parametrov týkajúcich sa vyslovene bezdrôtového rozhrania zariadenia bol znemožnený použitím virtualizovaného systému. Virtualizovaný systém totiž nemá prístup k fyzickým zariadeniam hostiteľského systému [7] a teda konfigurácia bezdrôtového rozhrania v OpenWRT skončila neúspešne. Skript je teda čisto teoretický a používa hodnoty parametrov nazbieraných v rámci predošlého programu.

Argumentami skriptu sú dátové súbory prijaté v rámci výmeny štatistík medzi členmi multicastovej skupiny. Je to súbor ktorý je vytvorený programom *stats_server* a teda jeho názov je IP adresa zdrojového zariadenia. Skript podľa nastavených váh vykonáva sumu všetkých hodnôt parametrov pre jednotlivé rozhrania a následne ako kandidáta na handover vyberie zariadenie s tou IP adresou, ktorého celková suma má najnižšiu hodnotu. Blok programu, ktorý vykonáva výber je opäť založený na *awk ,* pretože na vykonávanie matematických operácií je tento vhodnejší než skriptovací jazyk shellu :

```
curr_sum=$(awk -v xpw=$xpw -v xbw=$xbw -v xew=$xew -v xdw=$xdw 'NR != 1 {
      sum[$1]=0; sum[$1]+=$2*xpw;   sum[$1]+=$3*xpw;
      sum[$1]+=$4*xbw; sum[$1]+=$5*xbw; \
      sum[$1]+=$6*xew;sum[$1]+=$7*xew;sum[$1]+=$8*xdw;
      sum[$1]+=$9*xdw;
      } END {
      low_sum=999999999;
      for(s in sum) {if(sum[s]< low_sum){low_sum=sum[s]}}
                              print low_sum;
      }' data/${i})
```

Po získaní najnižších hodnôt pre každé zariadenie je vykonané porovnanie patričných súm pomocou ďalšieho *awk* bloku a následne skript pošle na výstup IP adresu toho zariadenia, ktoré je považované za najlepšieho kandidáta na handover.

Samozrejme v reálnej situácii by bolo potrebné skript rozšíriť o zahrnutie úrovne prijímaného signálu susedných bodov a hodnoty jednotlivých váh by bolo vhodné určit experimentálne.

7 ZÁVER

V texte práce je ilustrovaný postup nastavenia zariadenia MikroTik RB433, ako aj inštalácia virtualizovaného systému OpenWRT a jeho nastavenie. V rámci zadania boli vyvinuté dva skripty ktoré spolu umožňujú zber dát sieťovej prevádzky na daných rozhraniach prístupového bodu. Na implementáciu samotného zberu bol zvolený skriptovací jazyk BASH, ktorý je plnohodnotným programovacím jazykom vhodným práve na takéto účely. Medzi výhody takéhoto prístupu patrí priamy prístup k štatistickým dátam, ktorých hodnoty poskytuje sám operačný systém, čo má oproti získavaniu dát v reálnom čase zo sieťovej prevádzky niekoľko výhod. Patrí medzi ne hlavne rýchlosť a nízka záťaž na procesor prístupového bodu. Za účelom spracovania dát bol zvolený jazyk AWK, jazyk spracovania textu, ktorý sa stará o čítanie dát z vytvorených dátových súborov. Navrhnutá metóda zberu ďalej využíva časovač, ktorý sa stará o uloženie procesu mimo procesor zariadenia počas intervalu zberu. Počiatočné dáta sú zapísané do patričných dátových súborov na začiatku intervalu, a po vypršaní časovača je vykonaný zber koncových dát a vypočítané a uložené priemerné hodnoty.

Možnosti nastavenia parametrov systému sú popísané v neskorších kapitolách, kde bol zvolený prístup s uložením konfigurácie do textových súborov. Takýto prístup umožňuje ich pohodlné a prehľadné nastavenie, prenositeľnosť nastavení a tiež aj ich uloženie pre budúce potreby. Za účelom prenosu správ medzi jednotlivými zariadeniami bola navrhnutá sada programov v jazyku C, ktoré sú následne spolu s predošlími skriptami spojené do jedného programu zastrešeného riadiacim skriptom. Tento ucelený program sa stará o zber dát, ich ukladanie a výpočet priemerných hodnôt a tiež aj o čítanie konfigurácie. Vzhľadom na požiadavku použitia väčšieho počtu prístupových bodov bola v skripte zahrnutá aj možnosť automatického získania konfigurácie od susedných prístupových bodov a to spôsobom zaslania broadcastovej správy do lokálnej siete a čakania na odpoveď v podobe konfiguračného súboru, ktorý je následne uložený lokálne.

Použitie virtualizovaného systému OpenWRT na vývoj programu pre prístupový bod je vhodná voľba ale jeho použitie má aj nepriaznivé účinky. Takto virtualizovaný systém nie je vhodný na použitie do skutočnej prevádzky pretože jednak predstavuje zbytočnú záťaž na procesor prístupového bodu a jednak nemá prístup k hardwaru hosťujúceho zariadenia. RouterOS síce umožňuje premostiť virtuálne rozhrania s fyzickými portmi ale problém nastáva pri použití sériového portu, usb rozhrania či napríklad aj bezdrôtovej karty, kedy virtualizovaný systém vidí virtuálne rozhranie ako bežný ethernet port, a nie je teda možné pristupovať k špeciálnym parametrom fyzického rozhrania. RouterOS poskytuje programové rozhranie na prístup k jednotlivým častiam systému, ale také riešenie nie je vhodné pre systém na zber dát o sieťovej prevádzke pretože by zbytočne dochádzalo k ďalšej sieťovej komunikácii medzi dvoma systémami. Lepším riešením by bolo úplne nahradiť pôvodný operačný

systém permanentným OpenWRT systémom alebo iným systémom založeným na linuxovom jadre. Tieto okolnosti mali nepriaznivý dopad aj na samotný program vývýjaný v rámci práce, pretože pri zbere štatistických dát nebolo možné zahrnúť hodnotu úrovne prijímaného signálu prístupových bodov. Úprava programu na zahrnutie týchto dát by však pri použití plnohodnotného OpenWRT systému nebola nijak zložitá a ani jeho chovanie by sa nijak nezmenilo.

POUŽITÁ LITERATÚRA A ZDROJE

[1] *Bash Reference Manual* [online]. 2010 [cit. 2011-10-25]. Dostupné z WWW: <http://www.gnu.org/software/bash/manual/bashref.html>.

[2] GITE, Vivek. *Linux Shell Scripting Tutorial (LSST)* [online]. 2.0. 2010 [cit. 2011-11-16]. Dostupné z WWW: <http://bash.cyberciti.biz/guide/Main_Page>.

[3] GOEBEL, Greg. *A Guided Tour Of Awk* [online]. 2010 [cit. 2011-10-25]. Dostupné z WWW: <http://www.vectorsite.net/tsawk_1.html>.

[4] HALLINAN, Christopher. *Embedded Linux Primer : A Practical, Real-World Approach*. 2. Indianopolis : Prentice Hall, 2011. 616 s. ISBN 978-0-13-701783-6.

[5] JURČÍK, Michal. *Nízkoúrovňové řízení a sběr dat v bezdrátovém přístupovém bodu MikroTik*. Brno, 2011. 62 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[6] KOCHAN, Stephen G. *Programming in C*. Vyd. 1. Indianapolis: Sams Publishing, 2005, 543 s. 3. ISBN 06-723-2666-3.

[7] *MikroTik Wiki* [online]. 2011 [cit. 2011-10-14]. Dostupné z WWW: <http://wiki.mikrotik.com/wiki/Main_Page>.

[8] ROSHAN, Pejman; LEARY, Jonathan. *802.11 Wireless LAN Fundamentals*. 1. Indianapolis (Indiana) : Cisco Press, 2010. 312 s. ISBN 978-1587142246.

[9] STEVENS, W. Richard; FENNER, Bill; RUDOFF, Andrew M. *UNIX® Network Programming Volume 1 : The Sockets Networking API*. 3. Boston (USA) : Addison Wesley, 2003. 1024 s. ISBN 0-13-141155-1.

ZOZNAM POUŽITÝCH SKRATIEK

AP	(Access Point) - prístupobvý bod
API	(Application programming interface) - rozhranie pre programovanie aplikácií
CSMA/CA	(Carrier Sense Multiple Access/Collision Avoidance) - metóda viacnásobného prístupu s predchádzaním kolíziám
DHCP	(Dynamic Host Configuration Protocol) - protokol slúžiaci na automatickú konfiguráciu klientov v sieti IP
FTP	(File Transfer Protocol) – protokol pre prenos súborov
ICMP	(Internet Control Message Protocol) - sieťový protokol zväčša využívaný na diagnostické účely
IGMP	(Internet Group Management Protocol) - sieťový protokol na vytváranie multicastových skupín
LAN	(Local Area Network) - lokálna sieť pokrývajúca malé územie
MAC	(Media Acess Control adress) - adresa fyzickej vrstvy OSI modelu
NAT	(Network Address Translation) - preklad sieťových adries na rozhraniach smerovača
PIM	(Protocol Independent Multicast) - skupina multicastových protokolov pre IP siete
RADIUS	(Remote Authentication Dial In User Service) - sieťový protokol poskytujúci autentifikáciu, autorizáciu a účtovanie pre klientov za účelom využívania sieťových služieb
RouterOS	(Router Operation System) - operačný systém používaný v smerovačoch MikroTik
SSH	(Secure Shell) - sieťový protokol pre zabezpečenú komunikáciu
WLAN	(Wireless Local Area Network) - lokálna bezdrôtová sieť
WDS	(Wireless Distribution System) - systém ktorý umožňuje bezdrôtové pripojenie prístupových bodov

ZOZNAM PRÍLOH

Príloha 1 : CD príloha:
- PDF súbor tohto textu – xlenkp00_LMST.pdf
- navrhnuté programové súbory
- balíček skompilovaného programu ‚statistics' pre MikroTik OpenWRT systém, bleeding edge, revízia 30728

Príloha 2 : Zdrojové kódy skriptov a programov

ZDROJOVÉ KÓDY

monitor_interfaces.sh

```
#!/bin/bash
# vstupne parametre: [retazec_rozhrani interval]
# Pre kazde rozhranie uvedene v retazci prveho argumentu pri spusteni
# skriptu sa ziskaju infomacie o jeho stave prostrednictvom get_if_data.sh
# skriptu. Data su dalej ulozene do suboru a po uplynuti sledovacej doby
# proces prebehne odznova s tym ze su vypocitane a ulozene priemerne
# hodnoty. Pre kazde rozhanie je vytvoreny subor v adresari /data .
if test $# -eq 2 # Ak boli zadane argumenty prebehne skript
then
	interfaces=$1 # Retazec existujucich rozhrani
	interval=$2 # Interval sledovania
	for i in $interfaces # Pre kazde zadane rozhranie sa ziskaju data
	do
		# Ak rozhranie existuje zapisu sa pociatocne data do suboru
		echo -n 'init ' > data/${i}.dat
		./get_if_data.sh $i >> data/${i}.dat
	done
	if test $interval -gt 0 # Ak je nastaveny interval vykona sa zber
	then
		sleep $interval
		# Caka sa na uplynutie sledovacej doby
		# Pre kazde rozhranie z retazca rozhrani sa vykona zber
		# a ulozenie dat na konci sledovacej doby
		for i in $interfaces
		do
			echo -n 'final ' >> data/${i}.dat
			./get_if_data.sh $i >> data/${i}.dat
			echo "secs ${interval}" >> data/${i}.dat
#-- Blok awk programu – vypocet priemernych hodnot, pre prehladnost bolo odobrane
#-- odsadenie textu

# Nacitanie pociatocnych hodnot z riadku init
# Nacitanie konecnych hodnot z riadku final
# Nacitanie dlzky sledovacej doby z riadku secs
# Zapis dat do suboru:
#   v riadku su hodnoty <RXPps TXPps RXBps TXBps RXEpp TXEpp RXDpp TXPpp>
#   oddelene medzerou
awk '/init/
{rxp_i=$3;txp_i=$5;rxb_i=$7;txb_i=$9;rxe_i=$11;txe_i=$13;rxd_i=$15;txd_i=$17}
/fina/
{rxp_f=$3;txp_f=$5;rxb_f=$7;txb_f=$9;rxe_f=$11;txe_f=$13;rxd_f=$15;txd_f=$17}
/secs/ {secs=$2}
END {print (rxp_f-rxp_i)/secs" "(txp_f-txp_i)/secs" "\
    (rxb_f-rxb_i)/secs" "(txb_f-txb_i)/secs" "(rxe_f-rxe_i)" "\
    (txe_f-txe_i)" "(rxd_f-rxd_i)" "(txd_f-txd_i);
}' data/${i}.dat >> data/${i}_total.dat
#-- Koniec awk programu
		done
	fi
else
echo "Neboli zadane ziadne argumenty!"
fi
```

get_if_data.sh

```
#!/bin/bash
# Skript ktory zisti informacie sietovej prevadzky pre dane rozhranie
# urcene vstupnym parametrom.
# Data su ulozene do suboru /data/<nazov_rozhrania>.dat .
# Legenda k vystupnym datam :
#   rxp / txp – pocet prijatych/odoslanych paketov
#   rxb / txb – pocet prijatych/odoslanych bajtov
#   rxe / txe – pocet chyb v prichodzom/odchodzom smere
#   rxd / txd – pocet zahodenych paketov v prichodzom/odchodzom smere

# Ziskaju sa udaje z ifconfig pre dane rozhranie (prvy parameter - $1)
# a ulozia sa do docasneho suboru pre dalsie spracovanie.
ifconfig $1 > data/$1_ifconfig.tmp

# Prikazom awk sa z docasneho suboru ziskaju pozadovane data a poslu
# sa na vystup pomocou print
awk '
/RX p/ {split($2,i,":"); rxp=i[2]} /TX p/ {split($2,i,":"); txp=i[2]} \
/RX b/ {split($2,i,":"); rxb=i[2]} /RX b/ {split($6,i,":"); txb=i[2]} \
/RX p/ {split($3,i,":"); rxe=i[2]} /TX p/ {split($3,i,":"); txe=i[2]} \
/RX p/ {split($4,i,":"); rxd=i[2]} /TX p/ {split($4,i,":"); txd=i[2]} \
END {print "rxp: " rxp " txp: " txp " rxb: " rxb " txb: " txb \
" rxe: " rxe " txe: " txe " rxd: " rxd " txd: " txd}' data/$1_ifconfig.tmp
```

stats_manager.sh

```
#!/bin/bash
# Manazer skript
# Argumenty : [nazvy_rozhrani]
# Pouzitie: ./stats_manager eth0 wlan1

#kontrola argumentov (rozhrani ktore sa maju sledovat)
if test ! $# -eq 0 # Ak boli zadane argumenty prebehne skript
then
	interfaces=""
	for i in $@ # Kontrola nazvov rozhrani
	do
		# Ak rozhranie existuje je jeho nazov pridany do retazca rozhrani
		if ifconfig $i > /dev/null 2>&1
		then
			interfaces="${interfaces} ${i}"
		else
		echo "Rozhranie $i neexistuje!"
		fi
	done
	#ak nebolo zadane ziadne korektne rozhranie skript sa ukonci
	if test -z $interfaces
	then
		echo "Ziadne zo zadanych rozhrani neexistuje! Ukoncujem skript."
	else
		echo "Monitorovane rozhrania: ${interfaces}"
		# nacitanie konfiguracie - ak je #k v 'config' subore nastavene na
		# inu hodnotu ako 'net', alebo subor neexistuje, tak sa pristupi k
		# ziskaniu konfiguracie od susednych AP pomocou broadcastu
```

```
if test -e config #ak existuje 'config' subor
then
	#ziskanie hodnoty
	config_type=$(awk '/#k=/ {split($1,i,"="); config_type=i[2]}
			END {print config_type}' config)
	if [ $config_type != "file" ]
	then    # konfiguracia zo siete
		echo "Cakam na prijem konfiguracie.."
		./get_config > config
	else
		# ak je lokalna konfiguracia ziaduca tak sa spusti
		# server program ktory caka na vyziadanie konfiguracie
		# susednymi AP, po prijatej ziadosti je danemu
		# zariadeniu zaslany konfiguracny subor
		./send_config ${mc_ip} ${port} >/dev/null 2>&1 &
	fi
else
echo "Cakam na prijem konfiguracie.."
./get_config > config
fi
#----- CITANIE KONFIGURACNEHO SUBORU
# multicast IP adresa
mc_ip=$(awk '/#i=/ {split($1,i,"="); ip=i[2]} END {print ip}' config)
#port
port=$(awk '/#p=/ {split($1,i,"="); p=i[2]} END {print p}' config)
# sledovaci interval
interval_t=$(awk '/#t=/ {split($1,i,"="); t=i[2]}
		END {print t}' config)
# pocet intervalov po ktorych dojde k zaslaniu spravy
intervals_u=$(awk '/#u=/ {split($1,i,"="); u=i[2]}
		END {print u}' config)
# parametre zberu ( hodnota je odosielana ak prislusny parameter=1 )
xp=$(awk '/#xp=/ {split($1,i,"="); xp=i[2]} END {print xp}' config)
xb=$(awk '/#xb=/ {split($1,i,"="); xb=i[2]} END {print xb}' config)
xe=$(awk '/#xe=/ {split($1,i,"="); xe=i[2]} END {print xe}' config)
xd=$(awk '/#xd=/ {split($1,i,"="); xd=i[2]} END {print xd}' config)
prs=$(echo xp=$xp xb=$xb xe=$xe xd=$xd)

#---- SPUSTENIE PROGRAMU SERVER
echo "Spustam server na multicastovej adrese ${mc_ip} ..."
#spustenie serveru v pozadi, ulozenie jeho PID
./stats_server ${mc_ip} ${port} &
PID=$!
kill -0 ${PID} # test existencie spusteneho procesu
if [ $? ]
then
	echo "Server spusteny (PID:${PID})."
	#---- ZBER STATISTIK
	echo "Spustam zber dat s parametrami:"
	echo "  sledovaci interval: ${interval_t}s"
	echo "  pocet zberov pred zaslanim spravy: ${intervals_u}"
	echo "  zber informacii: ${prs}"

	#---- Hlavna smycka
	while [ 1 ]
	do
		echo "Prebieha zber dat.."
		for i in $interfaces #priprava suborov rozhrani
		do
			echo "" > data/${i}_total.dat
```

```
                        done

                        i=$intervals_u
                        # zber sa vykona vykona 'i'-krat
                        while [ $i -gt 0 ]
                        do
                        ./monitor_interfaces.sh "$interfaces" "$interval_t"
                         i=$(( $i - 1 ))
                        done

                        # pre vsetky rozhrania sa vypocitaju priemerne hodnoty z
                        # dat jednotlivych zberov a zapisu sa do suboru
                        # data/recent_data.dat. Tento subor sa nasledne prenasa
                        # k zariadeniam prihlasenym do multicastovej skupiny
                        echo "upd" > data/recent_data.dat
                        for i in $interfaces
                        do
                               echo -ne "${i} " >> data/recent_data.dat
                               #zahrnutie parametrov podla konfiguracie
                               awk -v t=$interval_t -v i=$i -v xp=$xp -v xb=$xb
                                      -v xe=$xe -v xd=$xd '{
                                      if (xp==1){RXP+=$1;TXP+=$2;}
                                      if (xb==1){RXB+=$3;TXB+=$4;}
                                      if (xe==1){RXE+=$5;TXE+=$6;}
                                      if (xd==1){RXD+=$7;TXD+=$8;}
                               }
                               END{
                                      if (xp==1){printf "%s %s ", RXP/t,TXP/t;}
                                      if (xb==1){printf "%s %s ", RXB/t,TXB/t;}
                                      if (xe==1){printf "%s %s ", RXE,TXE;}
                                      if (xd==1){printf "%s %s ", RXD,TXD;}
                               }' data/${i}_total.dat >> data/recent_data.dat
                               echo "" >> data/recent_data.dat
                        done

                        # spustenie klientskeho programu na odoslanie datoveho
                        # suboru multicastovej skupine
                        echo "Zasielam spravu na multicast adresu ${mc_ip}.."
                        ./stats_client ${mc_ip} ${port} &
                 done
          else
                 echo "Nepodarilo sa spustit server! Ukoncujem skript."
          fi
    fi
else
echo "Neboli zadane ziadne argumenty! Ukoncujem skript."
Fi
```

select_for_handover.sh

```
#!/bin/bash
# Skript pre vyhodnotenie najvhodnejsieho suseda na handover
# Skript prehlada subory v adresari /data ktorych nazvy(ip adresy)
# su predane v argumentoch skriptu

#priklad hodnot vah parametrov
xpw=1        #pre prijate/osodlane pakety
xbw=0.1      #pre prijate/osodlane bajty
xew=100      #pre pocet chyb na rozhrani
xdw=100      #pre pocet zahodenych paketov na rozhrani
```

```
if test $# -gt 1 # Ak boli zadane minimalne dva argumenty prebehne skript
then
	lowest_sum=999999999 #pociatocne velke cislo
	for i in $@
	do
		# pre danu ip adresu zariadenia su nacitane data a jednotlive hodnoty
		# su vynasobene vahou a navzajom scitane. Najnizsia suma z
		# jednotlivych rozhrani sa ulozi pre porovnanie s hodnotami dalsieho
		# zariadenia. Najnizsia hodnota znaci najvhodnejsie zariadenie
		# pre handover

		# jeden riadok na rozhranie, vystupom je najnizsia suma
		curr_sum=$(awk -v xpw=$xpw -v xbw=$xbw -v xew=$xew -v xdw=$xdw
			'NR != 1 {sum[$1]=0; sum[$1]+=$2*xpw; sum[$1]+=$3*xpw; \
			sum[$1]+=$4*xbw; sum[$1]+=$5*xbw; \
			sum[$1]+=$6*xew; sum[$1]+=$7*xew; sum[$1]+=$8*xdw; \
			sum[$1]+=$9*xdw;
			} END {
				low_sum=999999999;
				for(s in sum) {if(sum[s]< low_sum){low_sum=sum[s]}}
				print low_sum;
			}' data/${i})

		#porovnanie aktualnej najnizsej hodnoty s hotnotou daneho zariadenia
		lowest_sum=$(echo "$lowest_sum $curr_sum" | awk
			'{if ($1 > $2) print $2; else print $1}')

		# ak bola najnizsia hodnota nastavena na aktualnu tak je
		# ulozena ip noveho kandidata na handover
		if [ $lowest_sum == $curr_sum ]
		then
			ho_candidate=$(echo $i)
		fi
	done
	echo "Kandidatom je "$ho_candidate" s hodnotou "$lowest_sum
fi
```

config

```
/** Konfiguracny subor.
/** #k : typ konfiguracie, zo siete alebo tohto suboru (net/file)
/** #i : multicast ip adresa
/** #p : cislo portu
/** #t : sledovaci interval
/** #u : pocet intervalov po ktorych sa odosle sprava
/** parametre zberu (hodnoty 0/1):
/** #xp: zber poctu prijatych a odoslanych paketov
/** #xb: zber poctu prijatych a odoslanych bajtov
/** #xe: zber poctu chyb na rozhrani
/** #xd: zber poctu zahodenych paketov na rozhrani
#k=file
#i=239.255.1.1
#p=50000
#t=2
#u=4
#xp=1
#xb=1
```

```
#xe=1
#xd=1
```

get_config.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int main(int argc, char *argv[])
{
//---- * DEKLARACIA PREMENNYCH
int socket_dsc;           // deskriptor soketu;
int ret;                  // navratova hodnota sendto/recvfrom volani
socklen_t addr_length;    // dlzka adresy
int port;                 // cislo portu
struct sockaddr_in addr;         // struktura obsahujuca adresu
struct sockaddr_in response_addr; // adresa odpovedajuceho

//---- * NASTAVENIE ADRESY A PORTU
char *IP="255.255.255.255";      //broadcast IP adresa
port = 50001;

//---- * PRIPRAVA SOKETU
socket_dsc= socket(AF_INET, SOCK_DGRAM, 0); // vytvorenie noveho datagram socketu
if (socket_dsc < 0){      // ak sa nepodarilo vytvorit socket
       fprintf(stderr, "CHYBA: chyba otvarania socketu\n");
       exit(0);
}
// nastavenie moznosti zasielat pakety na broadcastovu adresu
int optval = 1;
setsockopt(socket_dsc, SOL_SOCKET, SO_BROADCAST, &optval, sizeof(optval));

addr_length=sizeof(struct sockaddr_in); // dlzka struktury adresy
bzero(&addr,addr_length);               // naplnenie adresy nulovymi hodnotami
addr.sin_family=AF_INET;                // nastavenie typu adresy (IPv4)
inet_aton(IP,&addr.sin_addr);           // nastavenie IP adresy broadcastu
addr.sin_port=htons(port);              // nastavenie portu

//---- * ODOSLANIE SPRAVY
char request[3];
strncpy(request,"cfg",3);               // typ spravy cfg – ziadost o konfiguraciu
char buffer[max_dlzka_spravy];          // zasobnik pre prijem spravy
bzero(buffer,max_dlzka_spravy);         // vynulovanie zasobniku

// odoslanie spravy na server
ret=sendto(socket_dsc,request,3,0,(const struct sockaddr *)&addr,addr_length);
if (ret > 0) {      // v pripade uspechu zaslania cakanie na odpoved zo servera
       ret = recvfrom(socket_dsc,buffer,1024,0,(struct sockaddr *)&response_addr,
              &addr_length);
       if (ret > 0) {
              //vypise prijatu konfiguraciu na vystup kde je
```

```
            //spracovana bash skriptom
            write(1,buffer,ret);
      }
}
else
{
      fprintf(stderr, "CHYBA: nepodarilo sa odoslat ziadost o konfiguraciu\n");
}

close(socket_dsc);  // uzavretie soketu

return 0;
}
```

send_config.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
int main(int argc, char *argv[])
{
//---- * DEKLARACIA PREMENNYCH
int socket_dsc;           // deskriptor soketu;
socklen_t addr_length;    // dlzka adries;
int ret;                  // navratova hodnota sendto/recvfrom volani
int server_port;          // cislo portu servera
struct sockaddr_in server_addr;  // struktura obsahujuca adresu servera
struct sockaddr_in client_addr;  // struktura obsahujuca adresu klienta
server_port = 50001;

//---- * PRIPRAVA SOKETU PRE PRIJEM
socket_dsc=socket(AF_INET, SOCK_DGRAM, 0); // vytvorenie noveho datagram socketu
if (socket_dsc < 0){      // ak sa nepodarilo vytvorit socket
      fprintf(stderr, "CHYBA: chyba otvarania socketu\n");
      exit(0);
}
addr_length = sizeof(struct sockaddr_in);     // dlzka struktury adresy
bzero(&server_addr,addr_length);       // naplnenie struktury nulovymi hodnotami
server_addr.sin_family=AF_INET;        // nastavenie typu adresy (IPv4)
server_addr.sin_addr.s_addr=INADDR_ANY;// nastavenie IP adresy servera
server_addr.sin_port=htons(server_port); // nastavenie portu servera

// spojenie daneho socketu s adresou a portom servera
if (bind(socket_dsc,(struct sockaddr *)&server_addr,addr_length)<0)
{
      fprintf(stderr, "CHYBA: chyba bind()\n");
      exit(0);
}

//---- * SPRACOVANIE PRICHODZICH SPRAV A ODOSLANIE KONFIGURACIE
char buffer[1024];        // zasobnik pre prijem spravy
// nekonecna smycka
while(1){
      // zablokovanie procesu a cakanie na prichodziu spravu
      ret = recvfrom(socket_dsc,buffer,max_dlzka_spravy,0,(struct sockaddr
```

```
            *)&client_addr,&addr_length);
      if (ret > 0){       // ak boli prijate znaky
            // ziskanie typu spravy kde cfg=ziadost o konfiguraciu
            char msg_type[3];
            strncpy(msg_type,buffer,3);
            msg_type[3]= '\0';
            if(!strcmp(msg_type, "cfg"))
            {
                  //otvorenie konfiguracneho suboru v binarnm mode na citanie
                  FILE *cfg_file;
                  long lSize;
                  char *buffer;
                  cfg_file = fopen ( "config" , "rb" );
                  if( cfg_file ) {
                        // ziskanie dlzky suboru
                        fseek( cfg_file , 0L , SEEK_END);
                        lSize = ftell( cfg_file );
                        rewind( cfg_file );
                        // priprava zasobniku pre obsah suboru
                        buffer = calloc( 1, lSize+1 );
                        if( buffer )
                        {
                              // kopirovanie obsahu do zasobniku
                              if( fread( buffer , lSize, 1 , cfg_file) == 1 )
                              {
                        // zaslanie konfiguracie
                        ret = sendto(socket_dsc,buffer,lSize, 0,(struct sockaddr
                              *)&client_addr,addr_length);
                              }
                              fclose(cfg_file);
                              free(buffer);
                        }
                  }
            }
            else
            {
                  ret = sendto(socket_dsc,"Nerozoznana sprava!\n",21,0,(struct
                        sockaddr *)&client_addr,addr_length);
            }
      }
}
return 0;
}
```

stats_client.c

```
/* UDP KLIENT argumentmi su multicast IP adresa a port */
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
```

```
{
//---- * DEKLARACIA PREMENNYCH
int socket_dsc;           // deskriptor soketu;
int ret;                  // navratova hodnota sendto/recvfrom volani
char multicast_IP[16];    // multicastova IP adresa
socklen_t addr_length;    // dlzka adresy
int multic_port;          // cislo portu pre multicast
struct sockaddr_in multic_addr;  // struktura obsahujuca adresu multicastu

// spracovanie argumentov
if(argc != 3) //ocakavaju sa tri argumenty (nazov programu,IP a port)
{
      fprintf(stderr, "[client]CHYBA: nespravny pocet argumentov (ip_adresa
            port)\n");
      exit(0);
}
else{
      strncpy(multicast_IP,argv[1],15); //naplnenie premennych
      multic_port = atoi(argv[2]);
}
//---- * PRIPRAVA SOKETU
socket_dsc= socket(AF_INET, SOCK_DGRAM, 0); // vytvorenie noveho datagram socketu
if (socket_dsc < 0){      // ak sa nepodarilo vytvorit socket
      fprintf(stderr, "[client]CHYBA: chyba otvarania socketu\n");
      exit(0);
}
addr_length=sizeof(struct sockaddr_in); // dlzka struktury adresy
bzero(&multic_addr,addr_length);       // naplnenie adresy nulovymi hodnotami
multic_addr.sin_family=AF_INET;        // nastavenie typu adresy (IPv4)
inet_aton(multicast_IP,&multic_addr.sin_addr);// nastavenie IP adresy multicastu
multic_addr.sin_port=htons(multic_port);      // nastavenie portu multicastu

//---- * ODOSLANIE SPRAVY
printf("[client]Odosielam data..\n");

//otvorenie datoveho suboru v binarnom mode na citanie
FILE *data_file;
long lSize;
char *buffer;

data_file = fopen ( "data/recent_data.dat" , "rb" );
if( data_file ) {
      // ziskanie dlzky suboru
      fseek( data_file , 0L , SEEK_END);
      lSize = ftell( data_file );
      rewind( data_file );
      // priprava zasobniku pre obsah suboru
      buffer = calloc( 1, lSize+1 );
      if( buffer )
      {
            // kopirovanie obsahu do zasobniku
            if( fread( buffer , lSize, 1 , data_file) == 1 )
            {
                  // zaslanie dat
                  ret = sendto(socket_dsc,buffer,lSize,0,(const struct sockaddr
                        *)&multic_addr,addr_length);
                  if (ret > 0) {
                        printf("[client]Sprava odoslana.\n");
                  }
                  else
```

```
                {
                        fprintf(stderr, "[client]CHYBA: nepodarilo sa odoslat
                                spravu\n");
                }
                close(socket_dsc);  // uzavretie soketu
        }
        fclose(data_file);
        free(buffer);
    }
}
return 0;
}
```

stats_server.c

```
/* UDP SERVER argumentmi su multicast IP adresa a port */
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int main(int argc, char *argv[])
{
//---- * DEKLARACIA PREMENNYCH
int in_socket_dsc;        // deskriptor soketu pre prijem;
int out_socket_dsc;       // deskriptor soketu pre vysielanie;
socklen_t addr_length;    // dlzka adries;
int ret;                  // navratova hodnota sendto/recvfrom volani
char multicast_IP[16];    // multicastova IP adresa
char sender_IP[16];       // IP adresa odosielatela prijatej spravy
int server_port;          // cislo portu servera
struct sockaddr_in multic_addr;  // struktura obsahujuca multicastovu adresu
struct sockaddr_in server_addr;  // struktura obsahujuca adresu servera
struct sockaddr_in client_addr;  // struktura obsahujuca adresu klienta
struct ip_mreq multic_req;       // struktura obsahujuca udaje o multicaste

// spracovanie argumentov
if(argc != 3)
{
       fprintf(stderr, "[server]CHYBA: nespravny pocet argumentov (ip_adresa
              port)\n");
       exit(0);
}
else{
       strncpy(multicast_IP,argv[1],15); //naplnenie premennych
       server_port = atoi(argv[2]);
}
// nastavenie struktury multicast IP adresy
addr_length = sizeof(struct sockaddr_in);     // dlzka struktury adresy
bzero(&multic_addr,addr_length);       // naplnenie struktury nulovymi hodnotami
if(inet_aton(multicast_IP,&multic_addr.sin_addr)){   // ak sa podaril prevod
       multic_addr.sin_family=AF_INET;        // nastavenie typu adresy (IPv4)
       multic_addr.sin_port=htons(server_port);      // nastavenie portu servera
       multic_req.imr_multiaddr = multic_addr.sin_addr; //nastavenie adresy
```

```
	multic_req.imr_interface.s_addr = INADDR_ANY; // nastavenie rozhrania
}
else
{
	fprintf(stderr, "[server]CHYBA: chyba konfiguracie (IP adresa)\n");
	exit(0);
}
printf("[server]Nastavena adresa: %s Nastaveny port:
	%u\n",multicast_IP,server_port);

//---- * PRIPRAVA SOKETU PRE PRIJEM
// vytvorenie noveho datagram socketu
in_socket_dsc=socket(AF_INET, SOCK_DGRAM, 0);
if (in_socket_dsc < 0){   // ak sa nepodarilo vytvorit socket
	fprintf(stderr, "[server]CHYBA: chyba otvarania socketu\n");
	exit(0);
}
// spojenie daneho socketu s multicastovou adresou
if (bind(in_socket_dsc,(struct sockaddr *)&multic_addr,addr_length)<0)
{
	fprintf(stderr, "[server]CHYBA: chyba bind() pre prijem\n");
	exit(0);
}
// prihlasenie do multicastovej skupiny
setsockopt(in_socket_dsc, IPPROTO_IP, IP_ADD_MEMBERSHIP, &multic_req,
	sizeof(multic_req));

//---- * SPRACOVANIE PRICHODZICH SPRAV
char buffer[1024];  // zasobnik pre prijem spravy
// nekonecna smycka
while(1){
	// zablokovanie procesu a cakanie na prichodziu spravu
	ret = recvfrom(in_socket_dsc,buffer,1024,0,(struct sockaddr
		*)&client_addr,&addr_length);
	if (ret > 0){	// ak boli prijate znaky
		// ziskanie typu spravy z prvych troch znakov prijatej spavy
		// server caka na spravu typu 'upd'
		char msg_type[3];
		strncpy(msg_type,buffer,3);
		msg_type[3]= '\0';
		if(!strcmp(msg_type, "upd")) // spracovanie prijatych dat
		{
			// vytvori sa subor ktoreho nazov tvori IP adresa odosielatela
			// spravy, tento subor obsahuje posledne data od zdrojoveho AP
			inet_ntop(AF_INET, &(client_addr.sin_addr), sender_IP,
				INET_ADDRSTRLEN);
			char filename[30];
			sprintf(filename, "data/%s",sender_IP);
			FILE *file;
			file=fopen(filename, "w");
			fprintf(file,"%s",buffer);
			fclose(file);
		}
	}
}
return 0;
}
```

SDK\package\Statistics\src\Makefile

```
// kompilacia jednotlivych C programov

all: stats_client stats_server get_config send_config

stats_client: stats_client.o
	$(CC) $(LDFLAGS) stats_client.o -o stats_client

stats_client.o: stats_client.c
	$(CC) $(CFLAGS) -c stats_client.c

stats_server: stats_server.o
	$(CC) $(LDFLAGS) stats_server.o -o stats_server

stats_server.o: stats_server.c
	$(CC) $(CFLAGS) -c stats_server.c

get_config: get_config.o
	$(CC) $(LDFLAGS) get_config.o -o get_config

get_config.o: get_config.c
	$(CC) $(CFLAGS) -c get_config.c

send_config: send_config.o
	$(CC) $(LDFLAGS) send_config.o -o send_config

send_config.o: send_config.c
	$(CC) $(CFLAGS) -c send_config.c


clean:
	rm *.o stats_client stats_server get_config send_config
```

SDK\package\Statistics\Makefile

```
// vytvorenie balicku pre instalaciu v OpenWRT

include $(TOPDIR)/rules.mk

# nazov a verzia balicku
PKG_NAME:=statistics
PKG_RELEASE:=1

# adresar pre vytorenie balicku
PKG_BUILD_DIR := $(BUILD_DIR)/$(PKG_NAME)

include $(INCLUDE_DIR)/package.mk

# udaje o balicku
define Package/statistics
	SECTION:=utils
	CATEGORY:=Utilities
	TITLE:=Statistics program
endef

# kopirovanie zdrojovych suborov do adresara pre vytvorenie balicku.
```

```
define Build/Prepare
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef

# specifikuje kam sa ma program nainstalovat – vytvorenie adresarov a kopirovanie
# suborov
define Package/statistics/install
	$(INSTALL_DIR) $(1)/statistics
	$(INSTALL_DIR) $(1)/statistics/data
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/stats_server $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/stats_client $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/get_config $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/send_config $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/monitor_interfaces.sh $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/get_if_data.sh $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/stats_manager.sh $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/stats_terminate.sh $(1)/statistics/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/config $(1)/statistics/
Endef

$(eval $(call BuildPackage,statistics))
```