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 KOMUNIKAČNÍHO PROTOKOLU PRO VÝMĚNU ZPRÁV MEZI PŘÍSTUPOVÉ BODY WLAN SÍTĚ PŘES IPV6

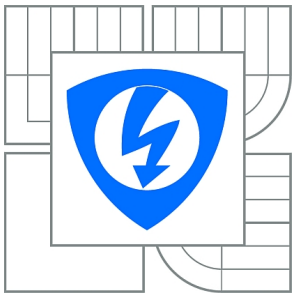
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE BORIS BARTAL
AUTHOR

BRNO 2012

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 KOMUNIKAČNÍHO PROTOKOLU PRO VÝMĚNU ZPRÁV MEZI PŘÍSTUPOVÉ BODY WLAN SÍTĚ PŘES IPV6

DESIGN AND IMPLEMENTATION OF COMMUNICATION PROTOCOL ALLOWING MESSAGE EXCHANGE BETWEEN WLAN ACCESS POINTS USING IPV6

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

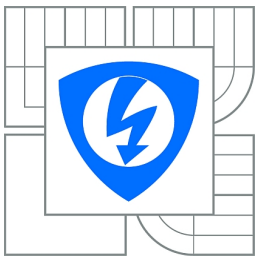
AUTOR PRÁCE — BORIS BARTAL

AUTHOR

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

SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Boris Bartal	***ID:***	125362
Ročník:	3	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Návrh a implementace komunikačního protokolu pro výměnu zpráv mezi přístupové body WLAN sítě přes IPv6

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte aplikaci pro operační systém OpenWRT, zprovozněný v zařízení Mikrotik RB433, která zajistí komunikaci mezi WLAN přístupové body využitím unicastového a multicastového přenosu využitím IPv6. Rozšiřte aplikaci tak, aby základní parametry přenosu byly nastaveny využitím textového konfiguračního souboru. Nastavte obsah zpráv tak, aby přenášely informace o vytížení bezdrátového síťového rozhraní. Proveďte zachytávání provozu generovaného danou aplikací a analyzujte obsah zachycených dat.

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, Ústav telekomunikací, 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:	6.2.2012	***Termín odevzdání:***	31.5.2012

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

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

UPOZORNĚNÍ:

ABSTRAKT

Bakalářská práce se zabývá programováním síťových aplikací v programovacím jazyku C a jeho implementací v přístupovém bodě. V první části práce jsou teoreticky popsány použité technologie, vysvětleno nastavení přístupového bodu a instalace operačního systému OpenWrt, který je nezbytný pro možnost použití aplikací v přístupovém bodě. V druhé části je popsán a vysvětlen postup programování samotné aplikace a její implementaci do přístupových bodů.

KLÍČOVÁ SLOVA

přístupový bod, OpenWrt, socketové programování, IPv6, multicast

ABSTRACT

The bachelor's thesis is about programming of network application in programming language C a its implementation in an access point. In the first part of the thesis, theory of used technologies is described, along with access point configuration and installation of operating system OpenWrt, which is necessary for usage of the applications in an access point. The second part describes and explains the process of programming of the aplication and its implementation to access points.

KEYWORDS

access point, OpenWrt, socket programming, IPv6, multicast

BARTAL, Boris *Návrh a implementace komunikačního protokolu pro výměnu zpráv mezi přístupové body WLAN sítě přes IPv6*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 54 s. Vedoucí práce byl doc. Ing. Karol Molnár, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Návrh a implementace komunikačního protokolu pro výměnu zpráv mezi přístupové body WLAN sítě přes IPv6“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Karolovi Molnárovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

V dnešní době se počítačové sítě vyvíjejí s velkou rychlostí, a hlavně internet je dnes součástí každodenního života. S rozvojem internetu se však musí vyvíjet i nové technologie, aby se stihli uspokojit čím dál větší nároky lidí. Ne vždy jsou dostupné technologie ideálním řešením problému přizpůsobení sítě k určitému účelu. Naprogramováním vlastní aplikace je však možno dostupné technologie libovolně kombinovat a vyřešit potřebný problém efektivněji.

Systém OpenWrt umožňuje efektivní prostředí pro běh aplikací v přístupovém bodě. Protože je tento systém distribuce Linuxu, dá se přizpůsobit téměř jakýkoliv požadavkům. Je výhodný z hlediska podpory různých technologií, zejména IPv6.

Výsledkem programování bude pár programů, jeden bude určen k získáni dat potřebných pro vysílání a samotné vysílání do multicastových skupin, druhý bude mít za úkol přihlásit se do multicastové skupiny a přijímat datagramy pro ni určené. Tento pár aplikací by měl na zařízení běžet současně, aby nebylo potřeba řešit střídání vysílání a příjímání datagramů a riskovat únik dat. Celá aplikace bude převedena pro podporu IPv6.

1 IPV6

IPv6 (Internet Protocol version 6) je sada protokolů, nástupce IPv4. Zajišťuje komunikaci a informace o adresaci pro směrovaní paketů v počítačové síti a především pro internet nové generace. Pracuje na síťové vrstvě – stará se o to, jak jsou data posílána z jednoho zařízení do druhého, přes počítačovou sít (např. Internet). IPv6 poskytuje oproti IPv4 více výhod, tou hlavní je však zvětšení adresního prostoru, který byl pro současné tempo rozvoje internetu příliš malý. Další výhody jsou :

- podpora QoS (Quality of Service)
- design odpovídající vysokorychlostním sítím
- bezpečnost
- podpora mobilních zařízení
- automatická konfigurace - nižší nároky na správu
- kooperace s IPv4 - co nejhladší přechod od stávajícího protokolu k novému

1.1 IPv6 adresování

Délka IPv6 adresy je čtyřnásobek původní IP adresy používané protokolem IPv4, má tedy 128 bitů. Počet různých IP adres je tedy v řádu 10^{38}. S takto velkým adresovým prostorem je možné přiřadit jedinečnou adresu každému a jakémukoliv zařízení připojenému do sítě. Toto je výhodné hlavně s nárůstem různých zařízení, které podporují přístup na Internet, jako například mobilní telefony, PDA (Personal Digital Assistant) a jiné, bez potřeby použití metod jako například překlad síťových adres, na to aby jsme se vyhnuli vyčerpaní těchto adres.

1.1.1 Zápis IPv6 adresy

V IPv6 se adresy zapisují v šestnáctkové soustavě a jednotlivé dvojice bajtů (čtveřice šestnáctkových číslic) se od sebe oddělují dvoutečkama. IPv6 adresa může vypadat například takto:

2001:0db8:ac10:fe01:0000:0000:0000:0000

pro zkrácení zápisu lze vynechat nuly v čtveřicích:

2001:0db8:ac10:fe01:0:0:0:0

skupiny nul následujících po sobě lze nahradit dvojtečkou, pokud by se tam vyskytovala jen jednou:

2001:0db8:ac10:fe01::

URL adresa bude pak:

http://[2001:0db8:ac10:fe01:0000:0000:0000:0000]/

Prefixy se zapisují stejně jako u IPv4 adres *adresa/délka*. Adresa určuje jenom začátek adresy (nevýznamné bity se nulují) a délka definuje počet významných bitů. Prefixu ff00::/8 vyhoví každá adresa která má prvních 8 bitů jedničky.

1.1.2 Typy IPv6 adres

Tab. 1.1: Tabulka prefixů IPv6 adres

Prefix	**Význam**
::/96	Adresy kompatibilní s protokolem IPv4
::/128	Nespecifikované adresy, slouží na adresovaní v rámci softwaru
::1/128	Lokální smyčka - reference na lokálního hosta,aplikace která odešle paket na tuto adresu ho dostane zpátky, ekvivalent v IPv4 je 127.0.0.1
2001:db8::/32	Dokumentace - všechny příklady IPv6 adres, by ideálně měly používat tento prefix, aby bylo možné rozeznat, že jde jen o příklad.
fec0::/10	Lokální adresa.Tento prefix značí, že adresa je platná jen v lokální organizaci
fc00::/7	ULA(Unique Local Adress) - tyto adresy jsou směrovány jen lokálně. (jsou pravděpodobně jednoznačné i globálně)
ff00::/8	Skupinové adresy
fe80::/10	Lokální linkové adresy - platné jen v rámci fyzické linky

Adresy mají tří kategorie:

1. Individuální adresy (unicast)
2. Skupinové adresy (multicast)
3. Výběrové adresy (anycast)

Individuální adresy slouží jako označení pro jediné rozhraní. Paket poslán na individuální adresu, bude doručen na rozhraní označené touto adresou.

Skupinové adresy slouží jako označení pro skupinu rozhraní které můžou patřit

různým zařízením. Paket označen touto adresou je doručen do více rozhraní najednou.

Výběrové adresy slouží jako skupinové označení, paket označen takovouto adresou však bude poslán pouze na jedno rozhraní patřící do této skupiny. Klient může poslat paket s obecnou adresou a ujme se ho některý ze serverů.

Oproti IPv4 tak zmizeli vysílací(broadcast) adresy a jejich roli převzaly skupinové adresy, které jsou upravené pro obecnější adresaci.

Jednotlivé druhy adres pro různé účely jsou mezi sebou rozlišeny pomocí prefixů. Lze je rozpoznat podle počáteční skupiny bitů. Výběrové adresy nemají svůj vlastní prefix. Jsou to globální individuální adresy, s jiným zpracováním v směrovacích algoritmech.

V tabulce 1.1 jsou uvedeny prefixy IPv6 adres a jejich významy.

1.2 Struktura IPv6 datagramu

Struktura hlavičky paketu IPv6 je na obrázku 1.1.

Obr. 1.1: Hlavička IPv6 paketu

- *Verze* – určuje verzi protokolu IP, u IPv6 je to binární číslo 0110(6)
- *Třída provozu* – specifikuje požadavky na vlastnosti sítě
- *Značka toku* – identifikace toku - proud datagramů od jednoho uživatele ke stejnému cíli ze stejnými vlastnostmi
- *Délka dat* – určuje množství přenesených dat

- *Další hlavička* – určuje typ rozšiřující hlavičky
- *Max.skoků* – určuje kolika směrovači může datagram projít (chrání před zacyklením)

Typy rozšiřujících hlaviček

Volby pro všechny	0	informace zajímavé pro každého po cestě (např. upozornění směrovače, že paket nese data, která by jej mohla zajímat)
Směrování	43	datagram musí projít předepsanou cestou
Fragmentace	44	při fragmentaci paketu nese informace nutné pro jeho složení do původní podoby
Šifrování obsahu (ESP)	50	obsah datagramu je zašifrován, ESP hlavička nese odkaz na parametry pro dešifrování
Autentizace (AH)	51	data pro ověření totožnosti odesilatele a obsahu
Poslední hlavička	59	nic dalšího nenásleduje
Volby pro cíl	60	informace určené příjemci datagramu (např. domácí adresa mobilního uzlu)
Mobilita	135	pro potřeby komunikace s mobilními zařízeními

2 MULTICAST

Multicast je definován v dokumentu RFC 1112. Multicast (neboli skupinové vysílání) je přenos datagramu do cílové skupiny pozůstávající z nula nebo víc klientů identifikovaných jedinou IP cílovou adresou. Multicastový datagram je doručen všem členům cílové skupiny, se stejnou spolehlivostí s jakou by byl doručen unicastový datagram.

Členství v multicastové skupině je dynamické, klienti se do této skupiny můžou přidat nebo ji opustit kdykoliv. Není žádné omezení ohledně počtu klientů ve skupině nebo jejich umístnění v síti. Klient může být členem více skupin zároveň a nemusí být členem takovéto skupiny na to, aby do ní mohl posílat datagramy.

Multicastová skupina může být dočasná nebo trvalá. Trvalá skupina má známou administrativně přidělenou adresu. Trvalá je jenom adresa, na rozdíl od členství ve skupině. Trvalá skupina může mít jakýkoliv počet klientů (i nula). Adresy, které nejsou přiděleny trvalým skupinám se využívají pro dočasné skupiny, které existují, jen dokud mají nějaké členy.

Multicast je realizován multicastovými směrovači. Pokud takovýto směrovač dostane multicastový datagram odešle ho všem sousedícím klientům, patřícím do cílové skupiny. Pokud má tento datagram time-to-live (u IPv6 Hop Limit - počet přeskoků) větší než 1, postará se směrovač o poslání datagramu všem dalším sítím, které mají klienty patřící do cílové skupiny.

Multicastové adresy
Multicastové skupiny mají přiřazený vlastní rozsah IP adres. U protokolu IPv4 mají rozsah 224.0.0.0 - 239.255.255.255. U protokolu IPv6 jsou multicastové adresy odlišeny předponou 0xFF.

2.1 Protokol IGMP

IGMP (Internet Group Management Protocol) se používá k řízení a zprávě multicastových skupin[reference BMTD]. Pracuje na síťové vrstvě a využívá IP datagramy.

Jeho funkcí je informovat nejbližší směrovač o připojení a odpojení klienta k multicastové skupině.

V dnešní době známe tři verze protokolu IGMP a v současnosti jsou užívány hlavně IGMPv2 a IGMPv3

2.1.1 IGMPv1

Využívá 2 typy zpráv:

- Membership Report - klient oznamuje své přihlášení na adresu požadované multicastové skupiny
- Membership Query - zpráva periodicky posílána směrovačem na adresu 224.0.0.1 (skupina všech klientů). V případě, že klient nebude mít zájem být členem skupiny, neodpoví směrovači zprávou Membership report

2.1.2 IGMPv2

Oproti IGMPv1 poskytuje lepší odhlašovací mechanizmus a menší zpoždění. Využívá tři typy zpráv:

- Membership Query
  - General Query - pomocí těchto zpráv směrovač zjistí, které skupiny mají v síti členy
  - Group Specific Query - směrovač posílá dotaz pro konkrétní skupinu, například po obdržení zprávy zjišťuje zda-li je ještě v skupině nějaký klient
- Membership Report
- Leave Group - využívá se k odhlášení z multicastové skupiny

2.1.3 IGMPv3

Tato verze je rozšířena o možnost specifikace zdroje, ze kterého mají být vysílána data. Používají se dva módy:

- Include - specifikován seznam zdrojů, od kterých je zájem přijímat vysílání do dané skupiny
- Exclude - specifikován seznam zdrojů, od kterých nemá být přijímáno multicastové vysílání

Pomocí této verze protokolu IGMP lze přijímat vysílaní od více zdrojů, a odhlašovat se lze z multicastové skupiny pouze od některých zdrojů.

2.2 Modely multicastu

Multicastové vysílání je zastoupeno dvěma skupinami:

- Any Source Multicast (ASM)
- Source Specific Multicast (SSM)

2.2.1 Any Source Multicast

ASM má strukturu sdíleného stromu - kořen stromu je směrovač, tzv. Rendezvous Point (RP). Tento směrovač má za úkol oznamovat kde v síti se nachází zdroj multicastového vysílání.

Komunikace v ASM probíhá v následujících krocích:

- Klient požádá nejbližší směrovač o přijímání vysílání ze zvolené adresy pomocí paketu IGMP
- Směrovač žádá RP o příjem multicastu
- RP začne vysílat klientovi multicastová data
- Z prvním přijatým paketem, koncové zařízení může vypočítat nejkratší cestu k zdroji, data pak budou směrovány touto cestou (nemusí být využito RP)

2.2.2 Source Specific Multicast

SSM má stromovou strukturu, přičemž zdroj multicastového vysílání je kořenem. Komunikace v SSM probíhá:

- Klient zažadá nejbližší směrovač o multicastové vysílání a zadá zdroj tohoto vysílání
- Je sestaven strom od zdroje ke klientovi

3 ZAŘÍZENÍ

Budeme používat zařízení MikroTik RouterBoard RB433. Samotný RouterBoard je jenom základní deska. Když tuto desku rozšíříme o několik prvků, jako např. bezdrátová karta, dostaneme vysokorychlostní přístupový bod. Součástí zařízení RB433 je operační systém RouterOS.

3.1 Technické parametry

Zařízení obsahuje mikroprocesor Atheros s frekvencí 300 MHz založený na architektuře MIPS-BE (Microprocessor without Interlocked Pipeline Stages), operační pamětí o velikosti 64 MB a vnitřní pamětí 64 MB. Obsahuje taky 3 LAN porty a 3 miniPCI porty použitelné pro rozšíření vnitřní paměti a především pro instalaci bezdrátové karty. Použitá bezdrátová karta je R52Hn podporující standardy 802.11a/b/g/n.

3.2 Připojení k RouterBoardu

Existuje více způsobů, jak se připojit do RouterBoardu s RouterOS. Můžeme použít IP adresu nebo sériový kabel. K samotnému příkazovému řádku se dostaneme pomocí WinBoxu, Telnetu, SSH nebo sériového kabelu. Připojení přes sériový kabel se dá provést přes HyperTerminal nebo Putty. Pro přístup je vždy potřeba sériový kabel, IP adresa, nebo popřípadě MAC adresa.

Telnet je protokol používaný k připojení k vzdálenému zařízení. Je součástí operačního systému Windows i Linuxových distribucí. Pro připojení do RouterOS lze použít příkaz:

```
telnet <ip adresa směrovače>
```

SSH (Secure Shell) je protokol podobný telnetu s rozdílem, že poskytuje pro připojení zabezpečení šifrováním. Je součástí Linuxových distribucí v systému Windows ho lze využívat pomocí různých klientů např. Putty. Jeho součástí je i prpgram `scp`(secure copy, který umožňuje šifrovaný přenos dat. Pro připojení lze použít příkaz:

```
ssh admin@<ip adresa směrovače
```

3.2.1 RouterBOOT

RouterBOOT je součástí RouterBoardu. Načte se po přípojení směrovače do elektrické sítě. Při připojení přes sériový kabel lze číst výstupní informace a upravovat nastavení pro bootování (proces zavádění operačního systému při startu) směrovače.

Důležité nastavení, které je vhodné změnit, je to odkud se bude směrovač bootovat. Buďto z paměti NAND (výchozí nastavení), nebo přes ethernetový port. Dále lze nastavit ještě několik méně důležitých parametrů. Výstup RouterBootu:

```
RouterBOOT booter 2.15

RouterBoard 433

Authorization: Passed
CPU frequency: 300 MHz
    Memory size:  64 MB

Press any key within 2 seconds to enter setup


RouterBOOT-2.15
What do you want to configure?
       d - boot delay
       k - boot key
       s - serial console
       o - boot device
       u - cpu mode
       f - cpu frequency
       r - reset booter configuration
       e - format nand
       g - upgrade firmware
       i - board info
       p - boot protocol
       x - exit setup
your choice:
```

3.2.2 WinBox

Po změně nastavení na bootovaní z Ethernet portu se lze do směrovače připojit přes WinBox. Je to konfigurační utilita od Mikrotiku, která se muže připojit na směrovač pomocí IP nebo MAC adresy. Po připojení se WinBox pokusí stáhnout ze směrovače všechny pluginy (jen když se připojuje poprvé). Po stažení se otevře hlavní okno, ve kterém je možné provádět konfiguraci různých parametrů. V podstatě je to grafické rozhraní pro RouterOS. WinBox usnadňuje a zpřehledňuje práci s RouterOS.

3.3 RouterOS

Základní operační systém pro RouterBoardy od Mikrotiku založen na GNU/Linuxu. Pro tuto práci byla použita verze 5.0. Je reprezentován příkazovým řádkem nebo GUI (Graphical User Interface - WinBox). Hlavní výhodou je jeho stabilita. Obsahuje všechny prvky potřebné pro tvorbu a spravovaní sítě.
Hlavní příkazy v příkazovém řádku RouterOS jsou *?* a *print*. *print* vypíše seznam položek menu, ve kterém se uživatel nachází a *?* vypíše seznam dostupných příkazů pro dané menu. Menu funguje jako adresářová struktura, kde pro vstup do adresáře se napíše jeho název pro návrat zpět "..", a do kořene se dá dostat pomocí "/".

3.3.1 IPv6 v RouterOS

Aby jsme mohli odesílat pakety protokolu IPv6, potřebujeme v systému RouterOS zprovoznit IPv6. K získaní podpory IPv6, v zařízeni s operačním systémem RouterOS, lze použít metodu 6to4[9]. Je to mechanizmus, který nám umožní posílat IPv6 data po IPv4 síti, bez potřeby dodatečné konfigurace. Poskytne IPv4 adrese zodpovídající přiřazení v IPv6 prostoru.

Funguje v několika krocích. Jako první první potřebujeme transformovat IPv4 adresu:

1. Konvertujeme adresu do hexadecimálního vyjádření (192.0.2.1 – C0000201)
2. Přidáme prefix 2002: (2002:C000:0201)
3. Doplníme adresu jako /48 blok (2002:C000:0201::/48)

Niní lze pomocí několika příkazů zprovoznit podporu 6to4 v systému RouterOS.

```
[admin@MikroTik]/interface 6to4
[admin@MikroTik]add disabled=np local-address=192.0.2.1 name=6to4
remote-address=unspecified
[admin@MikroTik]/ipv6 address
[admin@MikroTik]add address=2002:C000:0201::1/16 advertise=no
comment=6to4public disabled=no eui-64=no interface=6to4
[admin@MikroTik]add address=2002:C000:0201::1/64 advertise=yes
comment=6to4 subnet disabled=no eui-64=no interface=LAN1
[admin@MikroTik]/ipv6 nd prefix default
[admin@MikroTik]set autonomous=yes preferred-lifetime=2m
valid-lifetime=5m
[admin@MikroTik]/ipv6 route
[admin@MikroTik]add disabled=no distance=1 dst-address=2000::/3
gateway::192.88.99.1%6to4
```

3.3.2 Metarouter

MetaRouter je funkcí operačního systému RouterOS, umožnující vytvoření virtuálního zařízení. Toto zařízení umožňuje uživatelům s nižší prioritou konfigurovat parametry směrovače podle svých potřeb, bez zásahu do hlavní konfigurace směrovače nebo potřeby samostatného zařízení. Dále je možné doplnit tento virtuální směrovač o jiný operační systém, v našem případe OpenWrt.

Každá instance MetaRouteru využívá tolik prostředků jako samotná instalace RouterOS. Minimální požadavek pro každé virtuální zařízení je 16 MB RAM plus dalších 16 MB pro směrovač na kterém MetaRouter běží. Vytvoření virtuálního zařízení probíhá v adresářovém menu *metarouter* pomocí příkazu *add*. Je nutno zadat jméno virtuálního zařízení a je možné hodnoty virtuální paměti,v megabajtech, a pevné paměti,v kilobajtech , kterou bude instance MetaRouteru využívat.

Vytvoření MetaRouteru:

```
[admin@MikroTik] /metarouter> add name=mr0 memory-size=32 disk-size=32000
disabled=no
```

4 OPENWRT

OpenWrt je operační systém založený na jádře GNU/Linuxu, který umožňuje plné přizpůsobení síťového zařízení podle požadavků uživatele, pomocí balíčků, je ho tak možno přizpůsobit jakékoliv funkci. Oproti RouterOS má pro účel této práce výhodu hlavně v lepší možnosti programování a implementování vlastních programů do virtuálního zařízení.

4.1 Instalace

Instalace systému spočívá ve dvou krocích. Jako první je potřeba získat obraz OpenWrt. Buďto je možné si stáhnout balíčky ze stránek OpenWrt a přizpůsobit ho pro aplikaci do virtuálního zařízení - instance MetaRouteru. Na toto je potřeba aplikovat doplňkový balíček. Aby se to dalo provést je potřeba provést kompilaci jádra. Jednodušší řešení je stáhnout si obraz OpenWrt přizpůsobený pro použití na MetaRouter, který je dostupný na fóru MikroTiku[2]. Balíček připravený pro použití stačí nakopírovat do směrovače, což se jednoduše provede přes WinBox. Balíček uložen v směrovači pak stačí naimportovat

```
[admin@MikroTik] /metarouter> import-image file-name="jmeno obrazu OpenWrt"
```

a tím vytvořit nové virtuální zařízení. Před vstupem do virtuálního systému, mu je potřeba umožnit vstup do sítě, vytvořením přemostění na některém rozhraní nebo mu přidělit rozhraní staticky. Například na prvním ethernetovém portu. Lze to provést pomocí příkazového řádku nebo v programu WinBox.

```
[admin@MikroTik] /metarouter> / interface bridge
[admin@MikroTik] /interface bridge> add
[admin@MikroTik] /interface bridge> port
[admin@MikroTik] /interface bridge port> add bridge=bridge1 interface=ether1
```

Pak přidáme do přemostění virtuální rozhraní. Tento krok lze taky provést pomocí WinBoxu.

```
[admin@MikroTik] /interface bridge port> / metarouter
[admin@MikroTik] /metarouter interface>
[admin@MikroTik] /metarouter interface> add virtual-machine=mr2
dynamic-bridge=bridge1 type=dynamic
```

Do systému OpenWrt se pak dostaneme příkazem *console* kde vykonáme potřebné nastavení.

```
[admin@MikroTik] /metarouter> console mr2
```

4.2 Nastavení sítě

Nastavovaní síťových parametrů v operačním systému OpenWrt se provádí změnou konfiguračních souborů ve složce */etc/config/*. Jsou to soubory:

- */etc/config/wireless*
- */etc/config/dhcp*
- */etc/config/firewall*

V souboru *network* se provádí centrální nastavení sítě. Zahrnuje virtuální lokalní sítě (VLAN), konfigurace rozhraní, a směrování v síti. Soubor*wireless* nastavuje bezdrátovou síť a obsahuje různé nastavení podle standardu IEEE 802.11. Soubor *dhcp* ovládá nastavení DHCP (Dynamic Host Configuration Protocol) a DNS (Domain Name System) servrů. Soubor *firewall* obsahuje nastavení firewallu. Nastavovaní těchto souborů se provadí v textovém editoru *vi*, který je součastí OpenWrt. Je možné měnit nebo přidávat různé parametry. Po změně souboru je nutné vykonat příkaz */etc/init.d/network restart* aby se konfigurační soubor načetl. Pomocí těchto konfiguračních souborů lze v zařízení používajícím OpenWRT nakonfigurovat veškeré potřebné nastavení sítě.

Podpora IPv6 systémem OpenWrt je automatická a nastavuje se při kompilaci jádra, pokud jádro podporuje IPv6, rozhraní dostanou svou IPv6 adresu automaticky. To se dá zkontrolovat pomocí příkazu *ifconfig*.

5 VÝVOJ APLIKACE

Pro vytvoření spojení mezi zařízeními, je nutno vytvořit programy které budou vzájemně komunikovat. Spojení bude probíhat pomocí datagramů a datagramových socketů. Využijeme pár programů - listener a talker. Talker bude odesílat datagramy s daty, a listener bude čekat na datagramy a přijímat je. Program bude napsán v programovacím jazyku C, pomocí funkcí, které umožňují přístup k síťovému rozhraní.

5.1 Struktury

Velká část proměnných které se v programu vyskytnou budou součástí struktur. První strukturou bude `struct addrinfo`. Tato struktura reprezentuje jméno hostitele a adresu.

```
struct addrinfo {
         int              ai_flags;
         int              ai_family;
         int              ai_socktype;
         int              ai_protocol;
         size_t           ai_addrlen;
         struct sockaddr  *ai_addr;
         char             *ai_canonname;
         struct addrinfo  *ai_next;
        };
```

Tato struktura se nahraje použitím funkce `getaddrinfo()`. Jeho návratovou hodnotou bude ukazatel na lineární seznam (každý prvek obsahuje taky odkaz na následující prvek seznamu) těchto struktur naplněných požadovanými informacemi. V proměnné `ai_family` se nastavuje typ protokolu IP. Výchozí hodnota je `AF_UNSPEC`, v tomto případe se nastaví `AF_INET6` aby program podporoval jen protokol IPv6. V položce `ai_socktype` se nastavuje druh socketu, v tomto případe `SOCK_STREAM`. Pole `ai_flags` nastavuje specifikuje další možnosti. Bude stačit když se nastaví `AI_PASSIVE`. Ostatní pole vyplní funkce `getaddrinfo()`.

Další použitou strukturou je `struct sockaddr`. Uchováva informace o adrese socketu.

```
struct sockaddr {
         unsigned         short sa_family;
         char             sa_data[14];
        };
```

`sa_family` bude `AF_INET6` a `sa_data` obsahuje cílovou adresu a číslo portu. Aby se nemuselo pole `sa_data` vyplňovat ručně a zdlouhavě, byla vytvořena struktura `struct sockaddr_in` a její verze pro IPv6, `sockaddr_in6`. Tyto struktury mohou mít úlohu ukazatele na strukturu `struct sockaddr` a naopak.

```
struct sockaddr_in6 {
         u_int16_t        sin6_family; // AF_INET6
         u_int16_t        sin6_port;
         u_int32_t        sin6_flowinfo;
         struct in6_addr  sin6_addr;
         u_int32_t        sin6_scope_id;
      };

struct in6_addr {
         unsigned char    s6_addr[16];  };
```

Pole `sin6_port` je číslo portu(datový typ `u_int16_t` znamená unsigned 16 bit integer). Struktura `struct in6_addr` obsahuje IPv6 adresu. `sin6_flowinfo` je identifikátor toku a `sin6_scope_ID` je indetifikátor rozsahu IPv6 adresy.

5.2 Sockety

Sockety slouží jako způsob komunikace mezi procesy použitím Unixového deskriptoru souborů. Když mají programy vU nixu (GNU) vstup nebo výstup, provádějí ho pomocí čtení nebo zapisování do deskriptoru souboru. Deskriptor souboru je celé číslo (integer) přiděleno otevřenému souboru. Tento soubor může ale být i síťové připojení. Když chceme komunikovat s jiným programem po síti, provede se to přes deskriptor souboru. Tento deskriptor pro síťovou komunikaci (deskriptor socketu) získáme voláním funkce `socket()`.

Existuje několik druhů socketů. Dvěma základními jsou streamové sockety a datagramové sockety. Dále existují ještě raw sockety(`SOCK_RAW`), které umožňují přímý přístup k IP vrstvě. Streamové sockety (`SOCK_STREAM`) poskytují spolehlivou, obousměrnou, spojově orientovanou komunikaci. Data jsou doručovány v pořadí v jakém byli odeslány. Streamové sockety používají protokol TCP (Transmission Control

Protocol), který zabezpečuje vysokou úroveň kvality přenášení dat a velmi nízkou chybovost. Datagramové sockety (`SOCK_DGRAM`) poskytují nespolehlivou, nespojovanou komunikaci. Používají UDP (User Datagram Protokol). Posílají datagramy s IP hlavičkou která obsahuje adresu příjemce.

5.3 Základní funkce

V programu bude použitých několik funkcí které nám umožní přístup k síti a tím nám umožní zprovoznit komunikaci server-klient . Při volání těchto funkcí, jádro systému automaticky vykoná zadanou práci.

`getaddrinfo()`

Tato funkce slouží hlavně k práci ze strukturami popsanými výše. Úzce souvisí zejména ze strukturou `struct addrinfo`. Slouží na přeměnu textových řetězců reprezentujících IP adresy nebo domény na dynamicky přiřazené lineární seznamy struktur `struct addrinfo`. Přiřazení se uvolní funkcí `freeaddrinfo()`.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char            *node,
                const char            *service,
                const struct addrinfo *hints,
                struct addrinfo       **res);

void freeaddrinfo(struct addrinfo *res);
```

Zadají se tři vstupní parametry a funkce navrátí ukazatel na lineární seznam výsledků, `res`. Parametr `node` je jméno hostitele nebo jeho IP adresa. Další parametr, `service`, může být buď číslo portu, nebo jméno konkrétní služby (http, ftp, telnet...). Parametr `hints` ukazuje na strukturu `struct addrinfo`, která už je naplněna podstatnými informacemi.

Opačná funkce k `getaddrinfo()`je `getnameinfo()`, která vykonává přeměnu binárních reprezentací IP adres ve formě `struct addrinfo` na textové řetězce reprezentující jméno nebo IP adresu hostitele.

socket()

Touto funkcí získáme deskriptor socketu.

```
#include <sys/types.h>
#include <sys/socket.h>

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

Argumenty značí jaký socket požadujeme. Proměnnou `domain` se nastavuje verze protokolu IP, v tomto případe `PF_INET6`, `type` značí typ socketu, `SOCK_STREAM` nebo `SOCK_DGRAM` a `protocol` (TCP nebo UDP),se při nastavení 0, zvolí automaticky podle typu socketu. Tyto data však netřeba nastavovat explicitně, ale lze použít hodnoty výsledků funkce `getaddrinfo()`.

```
s = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
```

Funkce `socket()` navrátí celé číslo typu integer dále označované jako `sockfd`.

bind()

Po vytvoření socketu ho musíme přidružit k portu na lokálním zařízení. Vykoná se to pomocí funkce `bind()`.

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

`sockfd` je deskriptor socketu, `my_addr` je ukazatel na `struct sockaddr`, `addrlen` je délka adresy v bajtech.

connect()

Funkce `connect()` slouží pro připojení k vzdálenému zařízení pomocí IP adresy a čísla portu.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

`sockfd` je deskriptor socketu, `serv_addr` je struktura `struct sockaddr` obsahující cílový port a IP adresu a `addrlen` je délka adresy hostitele v bajtech.

close()

Funkcí `close()` se uzavře spojení a zabrání dalšímu čtení a zapisovaní do deskriptoru

```
close(sockfd);
```

Při chybě všechny zmíněné funkce navrátí hodnotu -1 a přiřadí určitou hodnotu proměné `errno`. Tato hodnota značí která chyba nastala.

5.4 Odesílání a přijímání dat

Na odesílání a přijímaní dat slouží dvojice funckcí `send()`, `recv()` (používají se u TCP socketů, nebo u UDP za použití funkce `connect()`) respektive jejich modifikace pro datagramový přenos `sendto()` a `recvfrom()`.

send()

```
int send(int sockfd, const void *msg, int len, int flags);
```

`sockfd` je deskriptor socketu, na který posíláme data, `msg` je ukazatel na odesílané data a `len` je délka dat v bajtech. Parametr `flags`, který specifikuje způsob jakým se data posílají, postačí nastavit na 0.

Návratová hodnota funkce `send()` je počet skutečně odeslaných dat v bajtech. Když je tato hodnota jiná, než délka dat `len`, musí se zbytek dat poslat dodatečně.

recv()

```
int recv(int sockfd, void *buf, int len, int flags);
```

`sockfd` je deskriptor, ze kterého se čte,`buf` je vyrovnávací paměť do které se načtou informace, `len` je maximální velikost paměti a `flags` opět postačí nastavit na 0 (má stejnou funkci jako u `send()`).

sendto()

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags,
           const struct sockaddr *to, socklen_t tolen);
```

Podobá se funkci `send()` se dvěma novýma parametrama to je ukazatel na strukturu `struct sockaddr`, která obsahuje cílovou adresu a port a `tolen` je délka struktury `struct sockaddr` a může být nastavena na `sizeof *to`.

recvfrom()

```
int recvfrom(int sockfd, void *buf, int len, unsigned int flags,
             struct sockaddr *from, int *fromlen);
```

Taky se podobá funkci `recv()` - `from` je struktura `struct sockaddr_storage` naplněna IP adresou a portem odesílatele paketu a `fromlen` je délka této struktury

5.5 Implementace multicastu

Aby aplikace podporovala multicastové vysílání je potřeba zařídit přidaní klienta do multicastové skupiny a připravit socket pro přijímání datagramů s multicastovou adresou.

Nastavení socketu se budou provádět pomocí funkce `setsockopt()`. Tato funkce slouží k nastavení různých možností pomocí, kterých jde socket přizpůsobit k mnoho účelům.

```
#include <sys/types.h>
#include <sys/socket.h>

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

`s` je deskriptor nastavovaného socketu, `level` je úroveň protokolu které se nastavení týká (např. pro nastavení na úrovni socketu se používá `SOL_SOCKET`, pro nastavení multicastu `IPPROTO_IP`, respektive `IPPROTO_IPV6` pro IPv6). `optname` určuje, jaké nastavení chceme provést. `optval` je ukazatel na data, které jsou potřeba pro nastavení konkrétní možnosti `optname` a `optlen` je délka těchto dat.

Hodnoty parametru `optname` pro nastavení multicastu jsou při úrovni `IPPROTO_IPV6` následující:
`IPV6_ADD_MEMBERSHIP` - připojí socket do multicastové skupiny na určeném rozhraní jako argument `optval` používá strukturu `ipv6_mreq`, která obsahuje adresu

požadované multicastové skupiny
`IPV6_DROP_MEMBERSHIP` - slouží pro opuštění multicastové skupiny upřesněné v struktuře `ipv6_mreq`
`IPV6_MULTICAST_HOPS` - nastavuje hop limit (počet přeskoků) multicastového paketu
`IPV6_MULTICAST_LOOP` - povolí (nebo zakáže) odesílání paketů na lokální smyčku, když je zařízení součástí cílové multicastové skupiny
`IPV6_MULTICAST_IF` - nastaví rozhraní pro odesílání multicastových paketů
Struktura `ipv6_mreq` uchovává adresu multicastové skupiny u IPv6

```
struct ipv6_mreq {
  struct in6_addr  ipv6mr_multiaddr;
  unsigned int     ipv6mr_interface;
};
```

`ipv6mr_multiaddr` je adresa multicastové skupiny a `ipv6mr_interface` je rozhraní na kterém má být připojení a odpojení z skupiny vykonáno (0 pro výchozí).

5.6 Načítaní argumentů ze souboru

Pro jednodušší práci se souborem, je vhodné načíst vstupní parametry ze souboru. Tuto problematiku lze řešit mnoha způsoby, nejjednodušší je pomocí textového souboru, s proměnnými zapsanými ve tvaru klíč=hodnota. Textový soubor se bude číst po řádcích a pomocí klíče se rozhodne do jaké proměnné se hodnota přiřadí. IP adresy lze ukládat jako textové řetězce, které nemusí mít přesnou délku (zkracovaní IP adres u IPv6), stačí pak využít funkci `inet_pton()`, která převádí adresu z čitelného tvaru do tvaru v jakém se zapisuje do požadovaných struktur (její opačná funkce je `inet_ntop()`).

```
#include <arpa/inet.h>

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

int inet_pton(int af, const char *src, void *dst);
```

`af` je address family (`AF_INET`, `AF_INET6`), `src` je zdrojová adresa, `dst` cílová proměnná do které se adresa uloží, `size` je maximální velikost cílového řetězce. Pro¨ převod z textového řetězce do typu integer lze použít funkci `atoi()`.

Údaje které se můžou načítavat ze souboru:

- typ spojení - výběr mezi multicastem a unicastem
- IP adresa
- port
- interval posílání zpráv

Tvar konfiguračního souboru a část programu pro jeho načítaní jsou uvedeny v přílohách.

5.7 Implementace aplikace do OpenWrt

Protože směrovač nemá potřebnou kapacitu paměti a dost výkoný procesor na to aby jeho operační systém mohl obsahovat kompilátor, nelze vyvíjet aplikaci přímo v prostředí OpenWrt, ale musí se použít tzv. křížová kompilace. Ta spočívá v tom, že se aplikace vytvoří v jednom systému a zkompiluje se pro použití v druhém systému. Základem pro zkompilování je operační systém obsahující GCC (GNU Compiler Collection). V tomto systému je pak nutno vytvořit SDK přesně pro architekturu operačního systému, ve kterém chceme aplikaci používat. SDK se také automaticky vytváří při kompilaci jádra. Postup vytváření je popsán v [2].

Pro vytvoření SDK pro OpenWrt bežící na virtuálním stroji v přístupovém bodě Mikrotik je potřeba nejdřív stáhnout aktuální zdrojový kód OpenWrt pomocí svn (subversion). V dalším kroku je potřebné stáhnout do adresáře se zdrojovým kódem záplatu pro použití na MetaRouter a aplikovat ji.

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

cd trunk/

wget http://www.mikrotik.com/download/metarouter/openwrt-
metarouter-1.2.patch

patch -p0 <openwrt-metarouter-1.2.patch
```

Když se záplata aplikuje, je potřeba nastavit konfiguraci, tento krok vyžaduje několik balíčků, které je potřeba doinstalovat v případě, že ještě v operačním systému nejsou.

```
make menuconfig
```

V okně které se následně otevře se musí nastavit cílový systém (target system) na Mikrotik MetaROUTER MIPS a označit vytvoření SDK (Build OpenWrt SDK).

Nasledně se spustí příkaz `make` (vytvoří se obraz OpenWrt, toto vytváření může trvat několik hodin). Po dokončení se v adresáři */bin* vytvoří archiv SDK, který po rozbalení bude sloužit ke kompilaci aplikací přenesitelných do systému OpenWrt. Pro kompilaci se program umístní do adresáře */SDK/packages/<nazev_programu>*, tento adresář bude obsahovat zdrojový kód ve složce */src* a dva soubory makefile. Soubory makefile obsahují instrukce pro kompilátory a určují postup kompilace. Jeden makefile umístněn ve složce */src* určuje kompilaci programu a druhý o složku víš určuje kompilaci balíčku. Po upravě souborů makefile (přizpůsobení některých hodnot) se vykoná kompilace spuštěním příkazu `make` v adresáři SDK. Vytvoří se balíček v adresáři */bin/mr_mips/packages*. Po přenesení do OpenWrt se nainstaluje příkazem:

```
opkg install <nazev_programu>
```

6 SBĚR INFORMACÍ O SÍŤOVÉM PROVOZU

Pro účely této práce je potřeba používat program, který monitoruje provoz síťových rozhraní. Je třeba uvažovat využití v prostředí OpenWrt, kde je systém omezen pamětí a procesorem směrovače, proto je třeba využít metodu, která není výpočtově náročná. Další nutností je zapsání dat do textového souboru pro další využití.

Jednou z možností jak tento program realizovat je pomocí příkazového interpreteru, tzv. shell, který je součástí systému založených na GNU/Linux. Příkazový interpreter čte a vykonává příkazy čtené ze standardního vstupu (klávesnice, textový soubor).Při ovládání z klávesnice má funkci příkazového řádku. Součástí systému OpenWrt je příkazový řádek typu ASH. Příkazový interpreter umožňuje programování tzv. skriptů, ve kterých lze spojit základní prvky programování (proměnné, smyčky, větvení) a zabudované příkazy shellu a vytvořit tak konstrukce, které po spuštění vykonají požadované operace.

6.1 Program awk

Skript pro sběr informací o síťovém provozu bude pracovat s textovými soubory. Hlavní funkcí programu *awk* je právě hledání v textových souborech po určitém vzoru, který je zadán v při spouštění programu. Další výhoda programu *awk* je práce s numerickými daty a možnost vykonávání matematických operací.

Syntaxe programu *awk* spočívá v zadání vzoru(textový řetězec) který se bude vyhledávat a následně akce, která se provede, když se vzor najde.

```
BEGIN   {<inicializace>}
<vzor>  {<akce>}
<vzor>  {<akce>}
 ...
END     {<finalní akce>}
```

6.2 Implementace skriptu pro sběr dat

Princip sběru informací o síťovém provozu spočívá v práci s výstupem příkazu *ifconfig*. Výstupem tohoto příkazu jsou informace o všech dostupných síťových rozhraních, včetně počtu přenesených paketů, bajtů, chybovosti pro oba směry. Při volání příkazu s názvem rozhraní jako parametrem jsou výstupem pouze informace o zadaném rozhraní(6.1).

```
root@OpenWrt:/# ifconfig br-lan
br-lan    Link encap:Ethernet  HWaddr 02:83:F4:FB:4E:52
          inet addr:192.168.3.1  Bcast:192.168.3.255  Mask:255.255.255.0
          inet6 addr: fe80::cca7:b7ff:feel:bc49/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:503 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:35105 (34.2 KiB)  TX bytes:1020 (1020.0 B)
```

Obr. 6.1: Vystup příkazu *ifconfig*

To bude použito v skriptu pro sběr informací o síťovém provozu, kde se výstupní informace uloží do souboru a program awk získá jednotlivé hodnoty o počtu přenesených dat. Výsledný skript (uveden v příloze) s názvem *get_if_data.sh* bude mít jako výstup, řetězec obsahující počet přijatých a odeslaných paketů a bitů (6.2).

```
root@OpenWrt:/# ./get.sh br-lan
rxp: 950 txp: 8 rxb: 67866 txb: 1020
```

Obr. 6.2: Vystup skriptu *get_if_data.sh*

6.3 Průměrné zatížení

Pro použití v praxi však nestačí pouze počet paketů a bitů, protože systém může běžet dlouhou dobu a údaje nemusí vypovídat o aktuálním stavu rozhraní. Proto je třeba zavést nový skript, který bude obsahovat časovač a bude získávat informace z rozhraní jen určitou dobu, případně vypočítal průměrné hodnoty. Parametry tohoto skriptu pak budou název rozhraní a časový interval, který bude představovat časový úsek, za který proběhne několik sběrů dat a bude z nich vypočten průměr nebo interval mezi dvěma sběry dat.

6.3.1 Implementace řídícího skriptu

Skript *monitor_interfaces.sh* byl navržen pro sběr dat v určitém časovém rozmezí a využívá skript *get_if_data.sh* za účelem získání informací o jednom rozhraní. Vstupní argumenty tohoto skriptu je seznam rozhraní, které se mají monitorovat. Pro každé rozhraní bude vytvořen soubor obsahující data z počátku a konce sledovacího intervalu, délku sledování, a po uplynutí sledovací doby do něho budou zapsána průměrná přenesená data.

Skript teda funguje tak, že po spuštění vykoná sběr dat a zapíše je do souboru, pak se uspí pomocí příkazu *sleep* na dobu definovanou jako sledovací interval. Po uplynutí doby vykoná sběr zase, data a délku intervalu zapíše do souboru a z počátečních a koncových dat vypočte průměrné hodnoty.

7 WDS

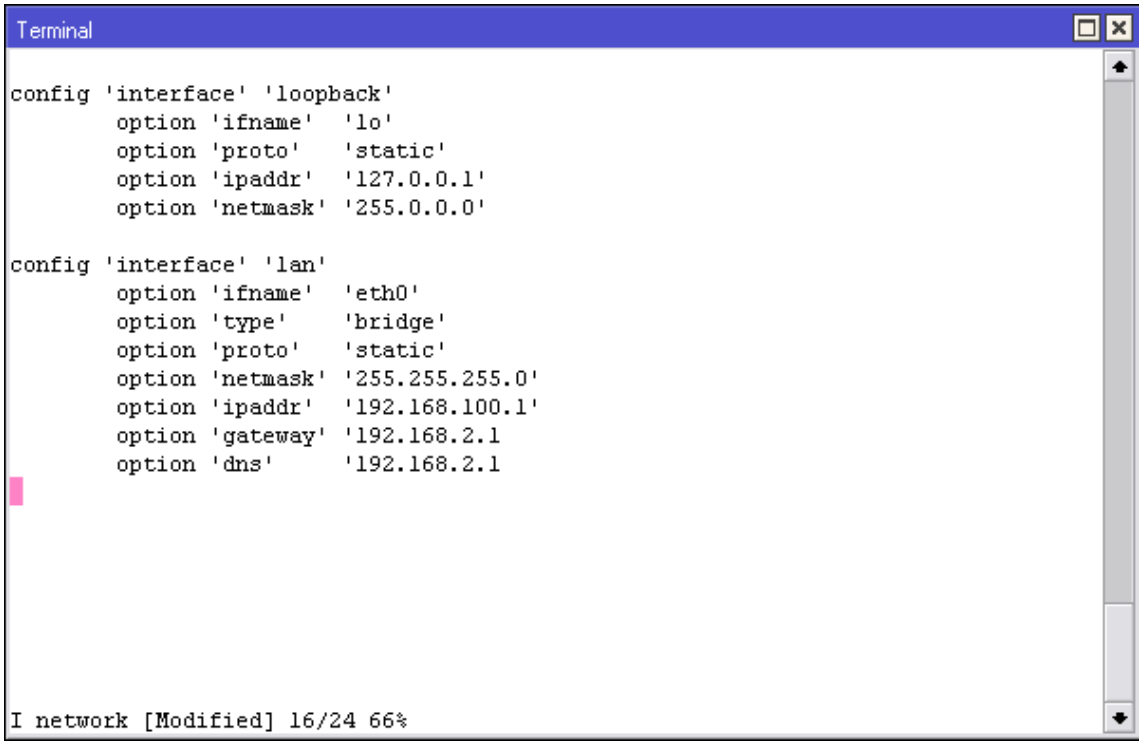
WDS (Wireless Distribution System) je způsob bezdrátového přemostění přístupových bodů. Funguje na principu vyhrazení části přenosového pásma pro komunikaci mezi přístupovými body navzájem, místo komunikace mezi přístupovým bodem a klientem. Úlohou je rozšíření rozsahu sítě bez potřeby zavedení kabelů, což lze využit například v dočasných sítích nebo v místech kam by bylo zavedení ethernetového kabelu obtížné. Také lze použít tento systém na propojení více lokálních sítí. Přístupové body musí být ve svém dosahu a pracovat na stejném kanále. Nevýhodou tohoto systému je snížení výkonu bezdrátové sítě, proto je vhodné tento systém používat v sítích bez nároku na velkou přenosovou rychlost a kapacitu. Další nevýhodou je to, že systém WDS je nestandardizovaný, proto je možnost, že zařízení od různých výrobců budou z hlediska WDS nekompatibilní.

7.1 Konfigurace WDS v OpenWrt

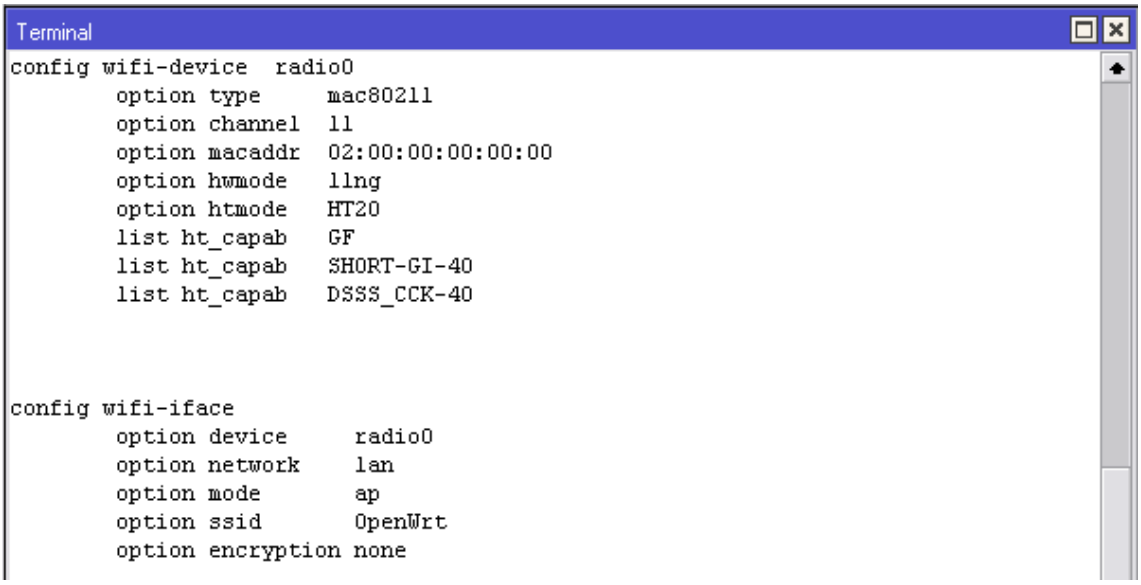
Pro bezdrátové propojení směrovačů ze systémem OpenWrt lze nakonfigurovat WDS přímo ve virtuálním stroji. Konfigurace probíhá v několika krocích, změnou síťových konfiguračních souborů.

Jako první je potřeba nastavit směrovač do módu přemostěného přístupového bodu (bridged AP), tím, že se přemostí lokální síť s bezdrátovou sítí směrovače.

Jako první se v souboru */etc/config/network* nadefinuje nová sekce pro rozhraní.



Dále je potřeba v souboru */etc/config/wireless* v sekci wifi-iface změnit síť aby souhlasila s názvem vytvořeného rozhraní.



A nakonec je třeba zablokovat soubor *dnsmasq* a zapnout rozhraní:

```
/etc/init.d/dnsmasq disable/
root@OpenWrt:/# ifup lan
```

Pro samotnou konfiguraci WDS je potřeba zjistit MAC adresy virtuálních směrovačů pomocí příkazu *ifconfig*. Dalším krokem je přidání nové bezdrátové sítě v souboru */etc/config/wireless*. BSSID se nastaví na MAC adresu směrovače, se kterým bude vytvořeno bezdrátové propojení.

```
config wifi-iface
        option device   radio0
        option network  lan
        option mode     wds
        option bssid    02:BC:57:DF:E1:9D
        option encryption none
```

Další nastavení se provedou v souboru */etc/config/firewall*.

```
config zone
        option name             lan
        option input    ACCEPT
        option output   ACCEPT
        option forward  ACCEPT
```

V souboru */etc/config/network* se musí povolit protokol stp(spanning tree)

```
config 'interface' 'lan'
        option 'ifname'   'eth0'
        option 'type'     'bridge'
        option 'proto'    'static'
        option 'netmask'  '255.255.255.0'
        option 'ipaddr'   '192.168.100.1'
        option 'stp'      '1'
```

Nakonec na jednom ze zařízení třeba zablokovat dhcp server. Ostatní změny se musí vykonat na obou zařízeních.

```
config dhcp lan
        option interface        lan
        option start            100
        option limit            150
        option leasetime        12h
        option ignore           1
```

Pro další WDS vlákna se musí všechny nastavení vykonat pro nové bezdrátové rozhraní.

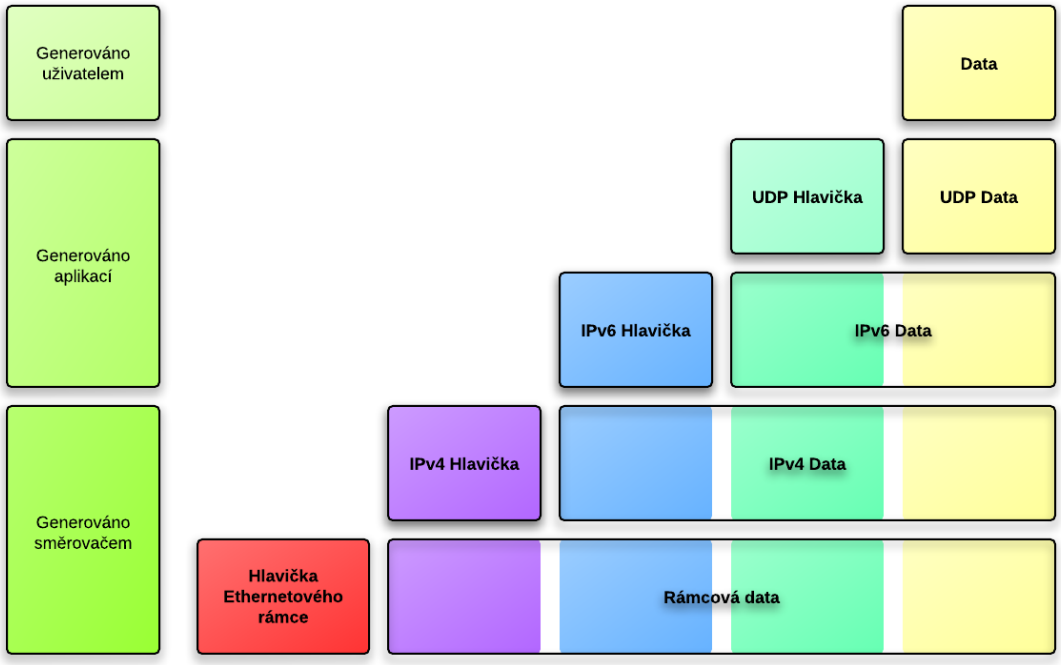
8 ANALÝZA

8.1 Wireshark

Pro ověření správné funkce počítačové sítě je potřeba síťový provoz analyzovat na to lze využít analyzátor Wireshark. Tento program má hodně funkcí. Dokáže rozpoznat velké množství síťových protokolů a poskytnout tak obraz o struktuře jednotlivých paketů. Jeho hlavní funkce je zachytávání provozu na určeném rozhraní. Pomocí zachycených paketů lze rekonstruovat komunikaci mezi dvěma zařízeními. Umožňuje taky filtrovat zobrazování paketů na základě protokolů nebo jiného parametru.

8.2 Zachytávání provozu generovaného aplikací

Unicastová aplikace přizpůsobená pro IPv6 generuje jen jeden druh paketů. Jsou to pakety přenášející data. Samotná data jsou zapouzdřena pod několika hlavičkami (8.1). Tento typ paketu se odesílá vždy, když se přenášejí data.



Obr. 8.1: Zapouzdření dat

Samotná data dává uživatel aplikaci jako vstup, případně je aplikace načte ze souboru (aplikační vrstva). UDP hlavička se vytváří automaticky při použití datagramového socketu na transportní vrstvě. Na síťové vrstvě je přidána hlavička

protokolu IPv6. Protože jsou data přenášeny po IPv4 síti, musí být přidaná další hlavička IPv4 generována směrovačem za použití metody 6to4. Hlavička ethernetového rámce je generována síťovou kartou vždy, když jsou data přenášena po ethernetové síti.

U multicastového spojení je generován další druh paketu, protokolu IGMPv2. Je to zpráva membership report a slouží k přihlášování do multicastové skupiny. Tato správa je odesílána v určitých časových intervalech.

9 ZÁVĚR

Seznámení se s Mikrotik RB433. Vytvoření instance Metarouter s operačním systémem OpenWrt. Vytvoření aplikace pro výměnu zpráv. Vysvětlení použitých funkcí a struktur. Přizpůsobení aplikace pro IPv6. Implementace multicastu do aplikace. Prostudování a implementace skriptů pro sběr dat na požadovaném rozhraní. Bezdrátové propojení přístupových bodů pomocí WDS. Implementace aplikace do systému OpenWrt.

Aplikace se skládá z dvou programů. První má za úkol rozesílat informace o vytížení určeného rozhraní do multicastové skupiny, do které se připojí ostatní směrovače v síti. Druhý program se připájí do multicastové skupiny a má za úkol přijímat data přicházející v pravidelných intervalech. Spojení je realizováno pomocí datagramových socketů, takže programy běží nezávisle na sobě. Odesílání datagramů může probíhat i když nejsou ostatní zařízení v síti dostupná. Přístupové body jsou spojeny bezdrátově pomocí WDS, to umožňuje rozšíření sítě bez potřeby spojení kabely.

Pro použití v praxi by bylo vhodné přidat vyhodnocování přijatých dat a přizpůsobit aplikaci pro využití těchto dat k zabezpečení efektivnějšího využití sítě.

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, Ústav telekomunikací, 2010.

[2] *Wiki.mikrotik.com* [online]. 2008, 17th October 2011 [cit. 2011-12-15]. Manual:Metarouter. Dostupné z WWW: <http://wiki.mikrotik.com/wiki/Manual:Metarouter>.

[3] *TNET Wiki* [online]. 12.7.2009, 18.7.2009 [cit. 2011-12-15]. RouterOS. Dostupné z WWW: <http://wiki.tresnovec.net/index.php?title=RouterOS>.

[4] ŠTRAUCH, Adam. *Root.cz* [online]. 22.7.2009 [cit. 2011-12-15]. OpenWRT: MetaROUTER v RouterOS. Dostupné z WWW: <http://www.root.cz/clanky/openwrt-metarouter-v-routeros/>.

[5] *OpenWRT : Wireless freedom* [online]. 2005 [cit. 2011-12-15]. Dostupné z WWW: <openwrt.org>.

[6] *IPv6.com*[online]. 2008 [cit. 2011-12-15]. IPv6 addressing. Dostupné z WWW: <http://ipv6.com/articles/general/IPv6-Addressing.htm>.

[7] *IPv6* [online]. 5.10.2008 [cit. 2011-12-15]. Dostupné z WWW: <www.ipv6.cz>.

[8] HALL, Brian. *Beej's Guide to Network Programming : Using Internet Sockets* [online]. Version 3.0.14. September 8, 2009 [cit. 2011-12-15]. Dostupné z WWW: <http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html>.

[9] *The (unofficial) Mikrotik site* [online]. September 4, 2010 [cit. 2011-12-15]. IPv6 and Mikrotik – Using 6to4. Dostupné z WWW: <http://www.mikrotik-routeros.com/?p=112>.

[10] ČÍKA, P. *Multimediální služby.* Skriptum VUT v Brně. 2007. s. 1-106.

[11] 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. 30 s. Vedoucí semestrální práce doc. Ing. Karol Molnár, Ph.D.

[12] WINDOWS. *Dev Center - Desktop* [online]. c 2012 [cit. 2012-05-28]. Dostupné z: http://msdn.microsoft.com/en-us/library/windows/desktop/

[13] CASTRO, Eva M. Porting applications to IPv6 HowTo. *Laboratories Over Next Generation Networks* [online]. 2002 [cit. 2012-05-28]. Dostupné z: `http://long.ccaba.upc.es/long/045Guidelines/eva/ipv6.html`

[14] The GNU Awk User's Guide. FREE SOFTWARE FOUNDATION, Inc. *GNU's Not Unix!* [online]. Edition 4. ©1989-2011 [cit. 2012-05-28]. Dostupné z: `http://www.gnu.org/software/gawk/manual/gawk.html`

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

IPv6 (*Internet Protocol version 6*) – protokol sloužící pro adresování v sítích, nová verze, která bude mít využití v budoucnosti, z přechodem síti na nové adresy

IPv4 (*Internet Protocol version 4* – protokol sloužící pro adresování používaný v současných sítích

QoS (*Quality of Service*) – služba se zaručenou kvalitou

PDA (*Personal Digital Asistant*) – multimediální zařízení známe též jako kapesní počítač

ULA (*Unique Local Address*) – jednoznačná lokální adresa

MIPS (*Microprocessor without Interlocked Pipeline Stages*)

RB (*RouterBoard*) – deska směrovače, označení pro základní desky, vyráběné firmou Mikrotik, které po přidání dalších doplňků slouží jako přístupové body nebo směrovače

SSH (*Secure Shell*) – síťový protokol sloužící pro zabezpečený přístup k vzdálenému zařízení

MAC (*Media Access Control*) – fyzická adresa přidělena výrobcem, celosvětově jedinečná pro každou síťovou kartu

OS (*Operating System*) – operační systém

GNU (*GNU's Not Unix*) – označuje operační systémy tzv. Unix-like, ale neobsahují žádný originální kód Unixu, mezi tyto systémy patří např. Linux

GUI (*Graphical User Interface*) – grafické uživatelské rozhraní

RAM (*Random Access Memory*) – druh paměti, určen pro dočasné uskladnění dat, za účelem spracovaní procesorem

DHCP (*Dynamic Host Configuration Protocol*) – síťový protokol bežící na DHCP serveru slouží na dynamické přidělování adres klientům

VLAN (*Virtual Local Area Network*) – virtuální lokální síť

DNS (*Domain Name Systém*) – systém sloužicí k překladu URL adres na číselné vyjádření adresy

API (*Aplication Programming Interface*) – rozhraní pro programování aplikací, sbírka funkcí, které jsou součástí nějaké knihovny nebo systému a není je teda třeba programovat

WDS (*Wireless Distribution System*) – systém bezdrátové distribuce, slouží k bezdrátovému propojení směrovačů

IGMP (*Internet Group Management Protocol*) – protokol pro správu internetových skupin

ASM (*Any Source Multicast*)

SSM (*Source Specific Multicast*)

RP (*Rendezvous Point*) – směrovač se speciálním účelem používán u ASM

SSID (*Service Set Identifier*) – identifikátor bezdrátové sítě

BSSID (*Basic Service Set Identifier*) – jedinečný identifikátor BSS stanice

TCP (*Transmission Control Protocol*) – spojově orientovaný spolehlivý protokol pro přenos dat

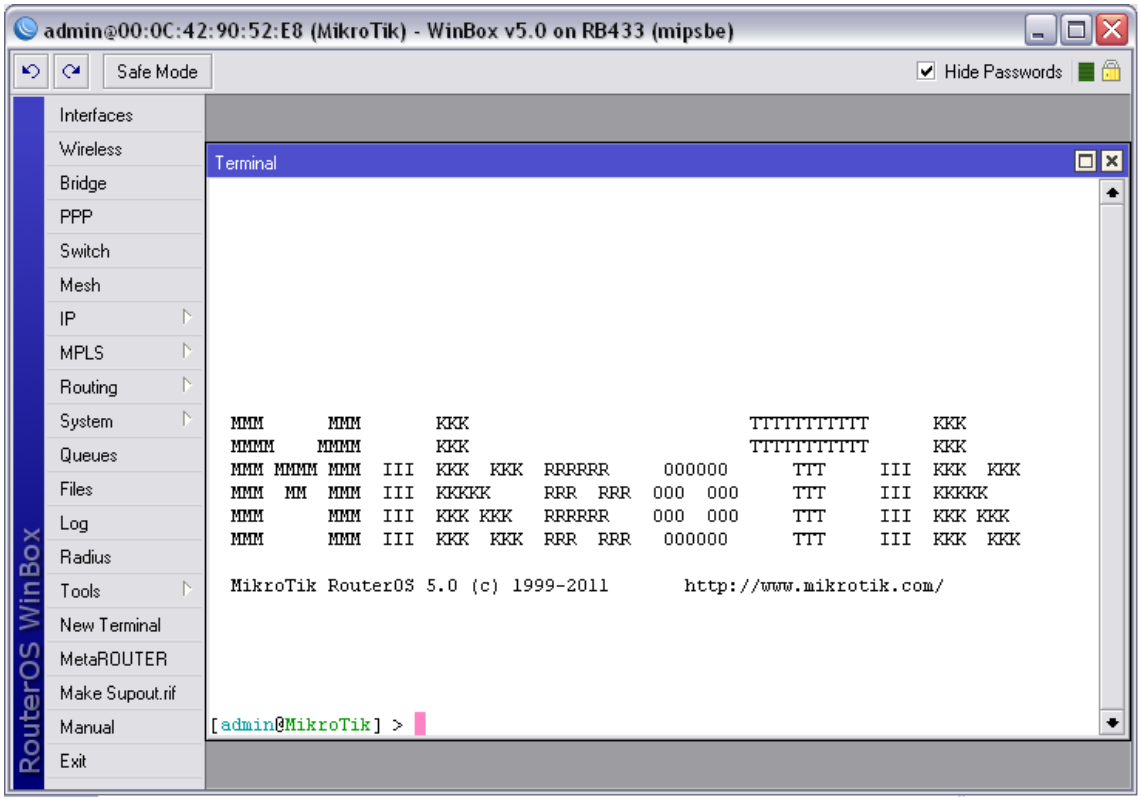
UDP (*User Datagram Protocol* – nespojově orientovaný nespolehlivý protokol pro přenos dat

GCC (*GNU Compiler Collection*) – sada kompilátorů GNU, kompilátory používáné v systémech GNU

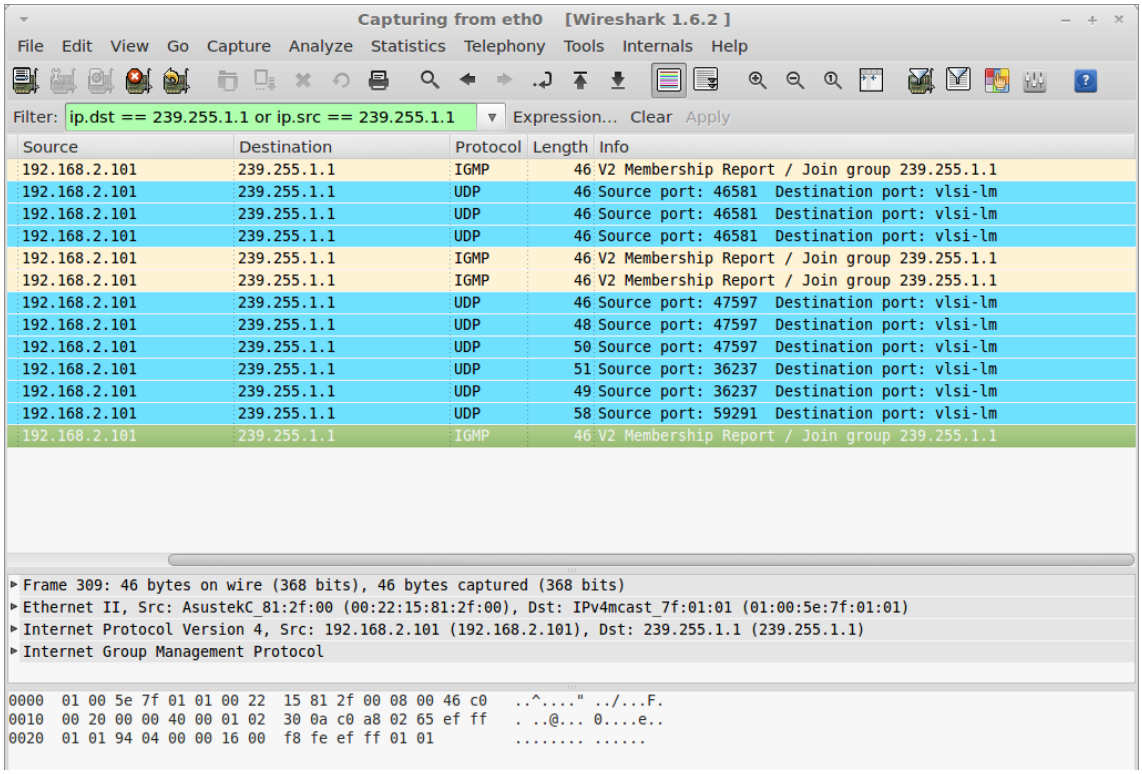
SDK (*Software Development Kit*) – sada nástrojů, která umožňuje kompilaci aplikace pro určité systémy

SEZNAM PŘÍLOH

A OBRAZKY



Obr. A.1: Okno winboxu



Obr. A.2: Okno programu Wireshark zobrazující multicastovou komunikaci u IPv4

B KONFIGURAČNÍ SOUBOR

B.1 Syntaxe konfiguračního souboru

```
@i=ff01::1111
@p=50500
@t=m
@v=30

//i - ip adresa (mc skupina)
//p - cislo portu
//t - typ spojeni (multicast/unicast)
//v - interval posilani zprav(delka sledovaciho intervalu)
//    v sekundach
```

B.2 Načítání parametrů z konfiguračního souboru

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

main()
{

int PORT; //PORT
int Sleep_time; //interval odesilani zprav
char mcast_IP[39]; // ipv6 adresa(v tomto pripade multicastova)
char t; //typ prenosu

FILE *f;
char radek[40]; //max delka radku

if(f = fopen("config.txt","r")) //otevreni souboru r- cteni
{
        printf("Nacitavani konfigurace ze souboru");

        while( fgets(radek, 60, f) != NULL)
                {
                        switch (radek[1]) //zkoumani ktery parametr uklada
                        {
                         case 'p':
                         PORT = atoi(radek+3);
                         break;
                         case 't':
                         t=radek[3];
                         break;
                         case 'i':
                         strncpy(mcast_IP,(radek+3),39);
                         break;
                         case 'v':
                         Sleep_time = atoi(radek+3);
                         break;
                        }
                }
                fclose(f); //uzavreni souboru
        }
        else
        {
                printf("Chyba nacitani souboru");
                exit(0);
        }
printf("\nPORT:%d\nTYP:%c\nadresa:%sx\ninterval:%d\n",PORT,t,mcast_IP,Sleep_time);
}
```

Obr. B.1: Část programu pro načítání parametrů ze souboru

C SKRIPTY PRO SBĚR DAT O SÍŤOVÉM PROVOZU

monitor_interfaces.sh

```
#!/bin/bash
# Pre kazde rozhranie uvedene ako argument 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 .

interval=3 # Delka sledovaciho intervalu
interfaces="" # Rozhrani, pro ktere se bude provadet sber ve forme retezce

if test ! $# -eq 0 # Ak boli zadane argumenty prebehne skript
then
	for i in $@ # Ziskavani dat pro kazde rozhrani
	do
		#Pokud rozhrani existuje vykona se pro nej sber dat a
		#bude pridan do retezce rozhrani pro zpracovani
		#po skonceni sledovaciho intervalu
		if ifconfig $i > /dev/null 2>&1
		then
			echo -n 'init ' > data/${i}.dat
			./get_if_data.sh $i >> data/${i}.dat
			interfaces="${interfaces} ${i}"
		else
			echo "Rozhranie $i neexistuje!"
		fi
	done
		if test $interval -gt 0 # kdyz je sledovaci doba 0 jedna se jen
					# jednoduchy sber dat, bez vypoctu prumeru
		then
			sleep "$interval" # Uspani skriptu do konce sledovaci doby

			# Vykonani sberu pro kazde rozhrani z retezce
			# a ulozeni dat do radku final
			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 prumernych hodnot
awk \
'/init/ \ # Nacitani hodnot ze zacatku intervalu - radek 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/ \ # Nacitani hodnot z konce intervalu - radek final
{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=$2} \ # Nacitani doby trvani intervalu
END {  print "RXPps: "(rxp_f-rxp_i)/secs; \ # Vypocet a odeslani prumernych hodnot
       print "TXPps: "(txp_f-txp_i)/secs; \ # a jejich odeslani na vystup
       print "RXBps: "(rxb_f-rxb_i)/secs; \
       print "TXBps: "(txb_f-txb_i)/secs; \
       print "RXEpp: "(rxe_f-rxe_i)/secs; \
       print "TXEpp: "(txe_f-txe_i)/secs; \
       print "RXDpp: "(rxd_f-rxd_i)/secs; \
       print "TXPpp: "(txd_f-txd_i)/secs; \
}' data/${i}.dat >> data/${i}.dat # Zapis dat do souboru

		done
		fi
	else
		echo "Nebyly zadany argumenty!"
	fi
```

get_if_data.sh

```
#!/bin/bash
# Legenda k vystupnym datum :
# rxp / txp - prijate/odeslane pakety (packet)
# rxb / txb - prijate/odeslane bajty (byte)
# rxe / txe - pocet chyb v obou smerech (error)
# rxd / txd - pocet zahodenych paketu v obou smerech (dropped)
# Ziskani dat z prikazu ifconfig pro prvni parametr - $1
# a ulozeni do docasneho souboru.
ifconfig $1 > data/$1_ifconfig.tmp

# Prohledavani docasneho souboru pomoci awk a odeslani na vystup
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
```

D PROGRAMY

```
/*talker.c*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>


/*funkce pro vypis chybovych hlaseni*/
static void DieWithError(const char* errorMessage)
{
  fprintf(stderr, "%s\n", errorMessage);
  exit(10);
}

int main(int argc, char *argv[])
{
/*inicializace promennych */
  int    sock;                        /* Socket */
  char*     multicastIP;              /* multicastová adresa */
  char*     multicastPort;            /* port */
  char*     sendString;               /* data k poslani */
  size_t    sendStringLen;            /* delka posilanych dat */
  struct addrinfo * multicastAddr;    /* struktura pro MC adresu */
  struct addrinfo  hints       = { 0 };  /* struktura hints pro funkci getaddrinfo */
  int sleep_time;                     /* interval odesilani */
 FILE *dat;
 char line[60];

/*nacitani promennych z prikazoveho radku, lze zamenit za nacitani ze souboru*/
  if ( argc != 3 )
  {
     fprintf(stderr, "Volani:  %s <Multicastova Adresa> <Port> \n", argv[0]);
     exit(10);
  }
   multicastIP   = argv[1];
   multicastPort = argv[2];
   sleep_time = 30;

  /* ziskani adresy - multicast */
  hints.ai_family  = PF_INET6;
  hints.ai_socktype = SOCK_DGRAM;
  hints.ai_flags    = AI_NUMERICHOST;
  if (getaddrinfo(multicastIP, multicastPort, &hints, &multicastAddr) != 0) DieWithError("getaddrinfo()");

  /* vytvoření  deskriptoru pro multicastovou adresu */
  if ((sock = socket(multicastAddr->ai_family, multicastAddr->ai_socktype, 0)) == -1) DieWithError("socket()");

  const int scope_id = 1;    /*0 nebo 1 podle rozhrani */
 if (setsockopt(sock, IPPROTO_IPV6, IPV6_MULTICAST_IF, (const char *) &scope_id, sizeof(scope_id)) != 0)
DieWithError("setsockopt(MULTICAST_IF) ");

  for (;;) /* nekonecny beh */
  {
     system("./monitor_interfaces.sh") ;/*pristup k prikazovemu radku a volani skriptu   */
     if (dat=fopen("data\wlan.dat", "r") {
     sendString=0;                        /*nacitani souboru do promenne k poslani */
      while(fgets(line, 60, f) !=NULL)
       { strcat(sendString, line);
            strcat(sendString,"\n",);
       }
      fclose(dat);
  }
 else
 {
  printf("Chyba nacitani datoveho souboru");
  exit(0);
 }
  sendStringLen = strlen(sendString);  /*zjisteni delky retezce k poslani */

/*odesilani dat*/
     int sendLen = sendto(sock, sendString, sendStringLen, 0, multicastAddr->ai_addr, multicastAddr->ai_addrlen);
     if (sendLen == sendStringLen )
     {
        printf("Poslano [%s] (%i bytu) na %s, port %s\n", sendString, sendLen,  multicastIP, multicastPort);
     }
     else
     {
        printf("sendto() : poslan jiny pocet bytu");
     }
     sleep(sleep_time); /* uspani na zadanou dobu  - interval posilani dat */
  }
  /* v pripade nedostupnosti*/
  freeaddrinfo(multicastAddr);
  close(sock);
  return 0;
}
```

```
/*listener*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

/*funkce pro vypis chybovych hlaseni */
void DieWithError(const char* errorMessage)
{
  fprintf(stderr, "%s\n", errorMessage);
  exit(10);
}

int main(int argc, char* argv[])
{
/*inicializace */
  int    sock;                /* Socket */
  char*    multicastIP;           /* multicastova adresa */
  char*    multicastPort;         /* port */
  struct addrinfo *  multicastAddr;         /* struktura pro MC adresu */
  struct addrinfo *  localAddr;           /* lokalni adresa pro propojeni (bind)*/
  struct addrinfo   hints       = { 0 };  /* struktura hints pro funkci getaddrinfo */


/*nacitani z prikazoveho radku , lze nahradit ctenim ze souboru*/
  if ( argc != 3 )
  {
    fprintf(stderr,"Volani: %s <Multicast IP> < Port>\n", argv[0]);
    exit(10);
  }

  multicastIP  = argv[1];
  multicastPort = argv[2];

  /* overeni mc adresy ... getaddrinfo */
  hints.ai_family = PF_INET6;
  hints.ai_flags  = AI_NUMERICHOST;
  if ( getaddrinfo(multicastIP, NULL, &hints, &multicastAddr) != 0 ) DieWithError("getaddrinfo() mc");

  /* ziskani lokalni adresy na zaklade multicastove*/
  hints.ai_family  = multicastAddr->ai_family;
  hints.ai_socktype = SOCK_DGRAM;
  hints.ai_flags   = AI_PASSIVE; /* navrati adresu pro bind */
  if ( getaddrinfo(NULL, multicastPort, &hints, &localAddr) != 0 )
  {
    DieWithError("getaddrinfo() local");
  }

  /* vytvoreni deskriptoru */
  if ( (sock = socket(localAddr->ai_family, localAddr->ai_socktype, 0)) == -1 )
  {
    DieWithError("socket()");
  }

  /* Bind na port multicastu */
  if ( bind(sock, localAddr->ai_addr, localAddr->ai_addrlen) != 0 )
  {
    DieWithError("bind() ");
  }
  /* pripojeni do MC skupiny.  */
  if ((multicastAddr->ai_family == PF_INET6)&&(multicastAddr->ai_addrlen == sizeof(struct sockaddr_in6)))
  {
    struct ipv6_mreq multicastRequest;  /* Multicast address join structure */

    /* specifikace mc skupiny */
    memcpy(&multicastRequest.ipv6mr_multiaddr, &((struct sockaddr_in6*)(multicastAddr->ai_addr))->sin6_addr,
sizeof(multicastRequest.ipv6mr_multiaddr));

    /* prijimani MC na jakemkoliv rozhrani */
    const int scope_id = 1;   /*podle rozhrani */
    multicastRequest.ipv6mr_interface = scope_id;

    /* samotne pripojeni do skupiny*/
    if ( setsockopt(sock, IPPROTO_IPV6, IPV6_JOIN_GROUP, (char*) &multicastRequest, sizeof(multicastRequest)) != 0 )
DieWithError("setsockopt(IPV6_JOIN_GROUP) ");
  }
  else DieWithError("neni IPv6");

  freeaddrinfo(localAddr);
  freeaddrinfo(multicastAddr);
```

```
  for (;;) /* nekonecny beh */
  {
    char   recvString[500];    /* zasobnik pro prijate data */
    int    recvStringLen;      /* delka prijatych dat*/

    /* Receive a single datagram from the server */
    if ((recvStringLen = recvfrom(sock, recvString, sizeof(recvString) - 1, 0, NULL, 0)) < 0)
    DieWithError("recvfrom() ");
    recvString[recvStringLen] = '\0';

    /* vystup - prijate data */
    printf("Received string [%s]\n", recvString);
  }
  /* nedostupny */
  close(sock);
  exit(EXIT_SUCCESS);
}
```