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 LABORATORNÍCH ÚLOH V SIMULAČNÍM PROSTŘEDÍ NS3

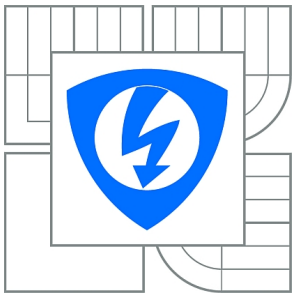
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MATEJ ŠTEFANČÍK
AUTHOR

BRNO 2015

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 LABORATORNÍCH ÚLOH V SIMULAČNÍM PROSTŘEDÍ NS3

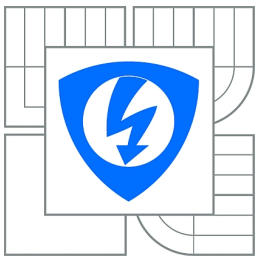
CREATION OF LABORATORY EXERCISES IN THE NS3 ENVIROMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MATEJ ŠTEFANČÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JOSEF POLÁK
SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Matej Štefančík	***ID:***	125661
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Návrh laboratorních úloh v simulačním prostředí NS3

POKYNY PRO VYPRACOVÁNÍ:

Seznamte s možnostmi simulačního prostředí NS3, včetně nástrojů pro získání dat ze simulací a možnostmi následné analýzy výsledků simulací. Dále se seznamte s obsahem předmětu Pokročilé komunikační techniky a v rámci diplomové práce navrhněte čtyři nové laboratorní úlohy včetně návodu vhodného pro studenty doplněného samostatnými úkoly a kontrolními otázkami použitelné pro laboratorní výuku tohoto předmětu. Časová náročnost jednotlivých úloh je přibližně dvě hodiny, včetně plnění samostatných úkolů a doplňujících otázek. Doporučená témata laboratorních úloh jsou: použití protokolů IPv4, IPv6, směrovací protokoly, aplikační protokoly, ToS, multicast, unicast.

DOPORUČENÁ LITERATURA:

[1] NS3 Consortium: Dokumentace k Network Simulator 3 (NS3), 2014. Dostupné online <http://www.nsnam.org/documentation/>, citováno 19.9.2014.
[2] JEŘÁBEK, J. Pokročilé komunikační techniky. Skriptum FEKT Vysoké učení technické v Brně, 2014. s. 1-189.

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: Ing. Josef Polák
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Diplomová práca poskytuje zoznámenie sa so sieťovým simulátorom ns-3. Popisuje základné abstrakcie a súčasti tohto nástroja. Práca ďalej rozoberá teóriu, ktorá bola aplikovaná pri vytváraní simulácie. Konkrétne je rozobraný pojem unicast, problematika IP adresovania a smerovanie v sieťach. Praktická časť je sústredená na vytvorenie štyroch laboratórnych úloh do predmetu Pokročilé Komunikačné Techniky. Prvá úloha sa venuje problematike unicastu a statického smerovania v sieti. Objasňuje vplyv metriky na proces smerovania. Nasledujúca úloha porovnáva jednotlivé typy správ protokolov ICMPv4 a ICMPv6. Tretia úloha rozoberá ďalšie funkcie protokolu ICMPv6. Konkrétne sa zameriava na päť rôznych typov ICMPv6 správ definovaných protokolom NDP (Neighbor Discovery Protocol) a ich úlohou v sieťovej prevádzke. Posledná úloha sa sústreďuje na zabezpečenie kvality služieb (QoS) v bezdrôtových sieťach, nastavením priority a tým rozdelením prevádzky do jednotlivých tried. Pozornosť je zameraná na parametre ako priepustnosť, oneskorenie a kolísanie oneskorenia.

KĽÚČOVÉ SLOVÁ

Sieťový simulátor, Ns-3, Unicast, IPv4, Smerovanie, ICMPv6, Aplikácia, IPv6, Laboratórna úloha, Uzol, QoS

ABSTRACT

Diploma thesis provides an introduction to the ns-3 network simulator. It describes the basic abstactions and components of the instrument. This work further discusses the theory, which was applied during simulation creation. The concept of unicast, the issue of IP addressing and routing in networks are specificaly analyzed. The practical part is focused on the creation of four laboratory tasks within the scope of Advanced Communication Techniques. The first task is dedicated to the the issue of static unicast routing in the network. It clarifies the impact of metrics on the process of routing. The following task compares the different types of ICMPv4 and ICMPv6 messages. The third task discusses additional functions of the ICMPv6. Specifically, it focuses on five different types of ICMPv6 messages defined by NDP (Neighbor Discovery Protocol) and their role in the network traffic. The last task focuses on ensuring the quality of services (QoS) in wireless networks, by setting priorities and thereby dividing the traffic into different classes. Attention is focused on parameters such as throughput, delay and jitter.

KEYWORDS

Network simulator, Ns-3, Unicast, IPv4, Routing, ICMPv6, Application, IPv6, Laboratory assignment, Node, QoS

ŠTEFANČÍK, Matej *Návrh laboratorních úloh v simulačním prostředí NS3*: diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 102 s. Vedúci práce bol Ing. Josef Polák

PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „Návrh laboratorních úloh v simulačním prostředí NS3“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád by som sa poďakoval vedúcemu diplomovej práce pánovi Ing. Josefovi Polákovi, za odborné vedenie, trpezlivosť a podnetné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

1 Úvod **13**

2 Sieťová vrstva s IPv4 protokolom **15**
2.1 Unicast . . . 15
2.2 IPv4 adresovanie . . . 15
2.2.1 Maska siete . . . 15
2.2.2 Triedy sieťových adries . . . 16
2.2.3 Špeciálne IPv4 adresy . . . 16
2.3 Smerovanie . . . 18
2.3.1 Statické smerovanie . . . 19
2.3.2 Dynamické smerovanie . . . 19
2.3.3 Smerovacia tabuľka . . . 19
2.3.4 Smerovanie na sieťovej vrstve . . . 20
2.3.5 Prepojenie pomocou smerovača . . . 20

3 Ns-3 **22**
3.1 Základný popis . . . 22
3.2 Prehľad . . . 23
3.3 Waf . . . 23
3.4 Vývojové prostredie . . . 24
3.5 Hlavné abstrakcie . . . 24
3.5.1 Uzol . . . 24
3.5.2 Aplikácia . . . 24
3.5.3 Kanál . . . 25
3.5.4 Sieťové zariadenie . . . 25
3.5.5 Pomocníci tvorby topológie . . . 25
3.6 Zdroje . . . 25

4 Úloha 1: Unicast a statické směrování **27**
4.1 Zadání . . . 27
4.2 Teoretický úvod . . . 28
4.2.1 Unicastová komunikace . . . 28
4.2.2 Statické směrování . . . 28
4.3 Popis zdrojového souboru . . . 28
4.4 Vypracování . . . 31
4.4.1 Aplikace bodu sedm zadání . . . 40
4.5 Kontrolní otázky . . . 41

5 Úloha 2: Porovnání ICMPv4 a ICMPv6 **42**
5.1 Zadání . . . 42
5.2 Teoretický úvod . . . 43
5.2.1 Internet Control Message Protocol . . . 43
5.2.2 Internet Control Message Protocol version 6 . . . 43
5.3 Vypracování . . . 44
5.3.1 První část . . . 45
5.3.2 Druhá část . . . 54
5.4 Kontrolní otázky . . . 62

6 Úloha 3: Protokol ICMPv6 **63**
6.1 Zadání . . . 63
6.2 Teoretický úvod . . . 64
6.2.1 Globální adresy . . . 64
6.2.2 Lokální linkové adresy . . . 65
6.2.3 Lokální smyčka . . . 65
6.2.4 Nespecifikovaná adresa . . . 66
6.2.5 Lokální unikátní adresy . . . 66
6.2.6 Skupinové adresy . . . 66
6.2.7 Internet Control Message Protocol version 6 . . . 67
6.2.8 Neighbour Discovery Protocol . . . 67
6.2.9 Address Resolution Protocol . . . 68
6.3 Vypracování . . . 69
6.3.1 První část . . . 69
6.3.2 Druhá část . . . 75
6.4 Kontrolní otázky . . . 80

7 Úloha 4: Wifi s podporou QoS **81**
7.1 Zadání . . . 81
7.2 Teoretický úvod . . . 82
7.2.1 Wifi . . . 82
7.2.2 Distribuovaná Koordinační Funkce . . . 82
7.2.3 Hybridní Koordinační Funkce . . . 83
7.3 Vypracování . . . 84
7.4 Kontrolní otázky . . . 93

8 Záver **94**

Literatúra **96**

Zoznam symbolov, veličín a skratiek **98**

Zoznam príloh **101**

A Príloha **102**

A.1 Obsah CD 102

A.1.1 Elektronický text diplomovej práce xstefa11.pdf 102

A.1.2 uloha1 102

A.1.3 uloha2 102

A.1.4 uloha3 102

A.1.5 uloha4 102

ZOZNAM OBRÁZKOV

2.1 Porovnanie unicastu a multicastu. [6] 15
4.1 Zadaná topologie první ulohy. 29
4.2 Výpis v terminálu pro On / Off aplikaci využívající protokol TCP. . . 37
4.3 Zobrazení komunikace na prvním uzlu pro On / Off aplikaci využívající protokol TCP. 38
4.4 Zobrazení simulace v programu NetAnim pro On / Off aplikaci využívající protokol TCP. 38
4.5 Obsah souboru static-routing pro On / Off aplikaci využívající protokol TCP. 39
4.6 Velikost paketů na uzlu Node0 za čas chodu aplikace On / Off využívající protokol TCP. 40
5.1 Základní formát ICMPv6 zprávy. 44
5.2 Zadaná topologie pro první část. 45
5.3 Zobrazení simulace v programu NetAnim. 51
5.4 Zobrazení komunikace na uzlu R1. 51
5.5 Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server. 52
5.6 Zobrazení chybové komunikace aplikace UDP Echo Client a Server. . 53
5.7 Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server. 53
5.8 Zadaná topologie pro druhou část. 54
5.9 Zobrazení simulace v programu NetAnim. 60
5.10 Zobrazení komunikace na uzlu R1. 60
5.11 Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server. 61
5.12 Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server. 62
6.1 Základní typy IPv6 adres. 64
6.2 Struktura globální individuální adresy. 65
6.3 Základní formát ICMPv6 zprávy. 67
6.4 Zadaná topologie pro první část. 69
6.5 Zobrazení komunikace na uzlu SOU pro aplikaci ping. 73
6.6 Soubor se směrovacími tabulkami pro jednotlivé uzly. 74
6.7 Zadaná topologie pro druhou část. 75
6.8 Zobrazení komunikace na uzlu R1. 78
6.9 Zobrazení komunikace na uzlu R2. 79
7.1 Zadaná topologie. 84

7.2 Výpis v terminálu pro On / Off aplikaci využívající protokol UDP. . . 92
7.3 Zobrazení komunikace na prvním uzlu pro On/Off aplikaci využívající protokol UDP. 92
7.4 Zobrazení simulace v programu NetAnim pro On/Off aplikaci využívající protokol UDP. 93

ZOZNAM TABULIEK

2.1 Triedy sieťových adries. [2] 16
2.2 Výpočet adresy siete. 16
2.3 Výpočet všesmerovej adresy. 17
2.4 Privátne sieťové rozsahy. [2] 18
5.1 Vybrané typy ICMP zpráv. 44
6.1 Členění adresního prostoru IPv6. 66
7.1 Přehled standardů IEEE 802.11. 83

1 ÚVOD

Dnešná doba tlačí na rozvoj telekomunikačných sietí. Dennodenne prichádzame do kontaktu s týmito sieťami v práci, doma a v súčasnosti sa dá povedať na každom kroku. Využívame ich pre prístup k informáciám, na komunikáciu, pre zábavu a mnohé ďalšie. S postupným rozvojom a navyšovaním počtu pripojení rastú požiadavky na dané siete. Preto je potrebné reagovať na túto situáciu. Riešením je úprava starých a vývoj nových lepších možností sieťovej komunikácie, ktoré zvýšia jej efektivitu. Ako príklad si uvedieme protokol IPv4, ktorý už nepostačoval dnešným nárokom (vyčerpanie adresného priestoru, bezpečnosť) a preto sa začal vývoj novej verzie IPv6. Implementácia nových protokolov však nie je jednoduchá a spája sa s ťažkosťami. Z toho dôvodu je nutné testovanie týchto protokolov pred ich použitím v reálnych sieťach.

Uvedieme iný príklad sieťového administrátora spravujúceho malú sieť, ktorá sa značne rozrastie a on musí zaistiť správne fungovanie vhodnou voľbou smerovacích protokolov, zabezpečenia pred neoprávneným prístupom a postarať sa, aby oprávnení užívatelia mali prístup k požadovaným službám. Pred samotným použitím je nutné overiť ako sa bude dané riešenie správať v skutočnej prevádzke.

Testovanie sietí majú na starosti sieťové simulátory ako je ns-3. Jedná sa o projekt sieťového simulátoru diskrétnych udalostí, určený prevažne pre výskum a vývoj. Snahou tohto projektu je vytvorenie otvoreného simulačného prostredia pre výskum sietí. Softvér ns-3 je voľne dostupný.

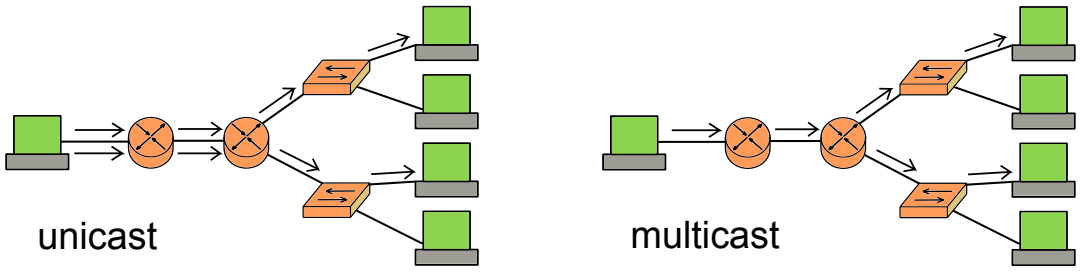
Cieľom diplomovej práca je použitie simulátora ns-3 pre tvorbu laboratórnych úloh z predmetu Pokročilé Komunikačné Techniky. Najskôr je v práci rozobraná teória z oblasti sietí. Následne sú vytvorené štyri laboratórne úlohy, ktoré overia tieto teoretické znalosti. Prvá úloha sa sústreďuje na unicastovú komunikáciu a statické smerovanie. Bližšie sa venuje vplyvu metriky na proces smerovania a doplnená je o aplikácie, graf a animáciu. Cieľom ďalšej úlohy je porovnanie protokolov ICMPv4 a ICMPv6. Vytvorené sú dva samostatné zdrojové súbory. Jeden sústreďujúci sa na ICMPv4 a druhý zameraný na ICMPv6. Porovnané medzi týmito dvoma protokolmi sú typy správ Echo Request (Požiadavka na odozvu), Echo Reply (Odozva na požiadavku), Destination Unreachable (Cieľ nedostupný) a Time Exceeded (Čas prekročený). Tretia úloha pokračuje protokolom ICMPv6. Porovnaných v úlohe je päť typov ICMPv6 správ definovaných protokolom NDP (Neighbor Discovery Protocol), ktorými sú Neighbor Solicitation (Výzva susedovi), Neighbor Advertisement (Ohlásenie suseda), Router Solicitation (Výzva smerovaču), Router Advertisement (Ohlásenie smerovača) a Redirect (Presmerovanie). Zabezpečenie kvality služieb (QoS) v bezdrôtovej sieti je náplňou poslednej úlohy. Vytvorené sú aplikácie, u ktorých nastavujeme prioritu, na základe ktorej je prevádzka rozdelená do tried, ktoré majú

definované špecifické zaobchádzanie s paketmi. Zameriavame sa na parametre priepustnosť, oneskorenie a kolísanie oneskorenia. V závere každej úlohy sú uvedené kontrolné otázky.

2 SIEŤOVÁ VRSTVA S IPV4 PROTOKOLOM

2.1 Unicast

Unicast (obr. 2.1) definujeme ako prenos od jedného zdroja dát k jednému príjemcovi. Slúži pre priamu komunikáciu medzi dvoma uzlami v sieti. Efektivita tohto prenosu je pri viacerých príjemcoch zhodných dát malá, keďže je potrebné zaslanie takého počtu správ od zdroja, aký je počet adresátov. Toto spôsobuje značné zaťaženie prenosových zdrojov. [6]



Obr. 2.1: Porovnanie unicastu a multicastu. [6]

2.2 IPv4 adresovanie

Sieťová adresa je 32-bitová logická adresa zariadenia v IP sieti. Vyjadrená býva v binárnej sústave v podobe núl a jednotiek. Pre človeka sa však nejedná o zrozumiteľný zápis a preto bola zavedená konvencia označená ako desiatková notácia. IP adresa je rozdelená na štyri časti nazývané oktety, pričom každý je tvorený ôsmymi bitmi, ktoré sú oddelené bodkou. Táto adresa sa zapisuje v dekadickej forme. Adresný rozsah je od 0.0.0.0 po 255.255.255.255, hovoríme však o teoretickom rozsahu. [12]

2.2.1 Maska siete

Umožňuje rozdelenie IP adresy na adresu podsiete a adresu konkrétneho zariadenia v tejto podsieti. Zápis je totožný s IP adresami, ale platné sú len hodnoty v binárnom tvare s jednotkami zľava a nulami sprava. Počet jednotiek udáva počet bitov určených pre adresu podsiete v IP adrese a zvyšok z masky udáva počet bitov pre adresáciu stanice v podsieti. Maska môže byť zapísaná aj v podobe dĺžky prefixu, ktorá definuje počet jednotiek v maske. Formálny zápis je pomocou lomky s daným počtom jednotkových bitov za IP adresou. [12]

2.2.2 Triedy sieťových adries

Adresný priestor protokolu IPv4 je rozdelený do piatich základných tried (tab. 2.1). Podľa počtu bitov určených pre adresáciu siete a aj počtu bitov pre adresovanie stanice, sa delia IP adresy do týchto tried. Pri komunikácii neboli potrebné masky vďaka pevne daným adresám. Označenie sietí s triednym adresovaním je classful siete. Adresy triedy A, B a C sa využívajú na bežné adresovanie, trieda D je určená pre multicast a adresy triedy E sú experimentálne. Rozdelenie adries do jednotlivých tried sa ukázalo ako neefektívne a preto bola nasadená technika podsieťovania. Nazýva sa Variable Length Subnet Masking (VLSM) a umožňuje rozdeliť IP adresný priestor na hierarchiu podsietí rôznej veľkosti. Dosiahne sa lepšie využitie adresného priestoru bez zbytočného plytvania veľkého počtu adries a vytvorenie podsietí s rozdielnym počtom staníc. [6] [12]

Trieda	Určujúce bity	Rozsah adries	Maska	Prefix
A	0	0 - 127.x.x.x	255.0.0.0	/8
B	10	128 - 191.x.x.x	255.255.0.0	/16
C	110	192 - 223.x.x.x	255.255.255.0	/24
D	1110	224 - 239.x.x.x		
E	1111	240 - 255.x.x.x	rezervované	

Tab. 2.1: Triedy sieťových adries. [2]

2.2.3 Špeciálne IPv4 adresy

Adresa siete (tab. 2.2) vznikne nahradením všetkých bitov z časti pre adresu stanice nulami, popisuje prvú adresu z danej siete. Logickým súčinom (AND) jednotlivých bitov IP adresy a masky siete vypočítame adresu siete.

IP adresa:	10.217.123.7/20
Maska:	255.255.240.0
IP adresa binárne:	00001010 11011001 01111011 00000111
Maska binárne:	11111111 11111111 11110000 00000000
Adresa siete binárne:	00001010 11011001 01110000 00000000
Adresa siete:	10.217.112.0

Tab. 2.2: Výpočet adresy siete.

Všesmerová (broadcast) adresa (tab. 2.3) umožňuje súčasne poslať dátové jednotky všetkým staniciam v danej podsieti. Vznikne nahradením bitov z časti pre adresu stanice jednotkami. Matematicky je operácia popísaná ako bitový súčet (OR) IP adresy a invertovanej masky podsiete. Všesmerová adresa je posledná adresa v danej sieti.

Adresy koncových staníc v sieti sú všetky adresy v rozsahu medzi adresou siete a všesmerovou adresou v rámci danej siete.

Lokálna slučka (loopback) je virtuálne sieťové rozhranie vo vnútri počítača, ktoré nie je pripojené k žiadnemu hardvéru. Slučka je využívaná pri medziprocesorovej komunikácii a má adresu 127.x.x.x/8. Dátové jednotky s touto adresou neopustia počítač, sú spätne smerované na toto rozhranie.

IP adresa:	10.217.123.7/20
Maska:	255.255.240.0
Maska binárne:	11111111 11111111 11110000 00000000
Inverzná maska binárne:	00000000 00000000 00001111 11111111
IP adresa binárne:	00001010 11011001 01111011 00000111
Všesmerová adresa binárne:	00001010 11011001 01111111 11111111
Všesmerová adresa:	10.217.127.255

Tab. 2.3: Výpočet všesmerovej adresy.

Neverejné (privátne) adresy (tab. 2.4) sú IP adresy určené pre použitie v lokálnych sieťach. Nie je možné sa pomocou týchto adries priamo pripojiť k Internetu. Potrebné je využiť funkciu prekladu privátnych adries na verejné (NAT), prostredníctvom preposlania komunikácie na smerovač alebo L3 (pracujúci na tretej vrstve) prepínač, ktorý podporuje danú funkciu. Privátne adresy sa môžu opakovať v jednotlivých lokálnych sieťach po celom svete, lebo sú skryté za hraničnými smerovačmi. Výhodou tohto systému je navýšenie bezpečnosti. [6]

Lokálna linková adresa je automaticky alokovaná, keď nebola stanici nastavená statická IP adresa alebo sa jej nepodarí získať adresu z DHCP serveru. Rozsah týchto adries je 169.254.0.0 až 169.254.255.255 a stanica si sama nastaví sieťovú adresu z rozsahu. Lokálne linkové adresy sa vyznačujú unikátnosťou len v danej podsieti a smerovače ich ďalej nedistribujú. Medzi ďalšie špecifické adresy patrí rozsah 0.0.0.0/8 určený pre lokálnu identifikáciu staníc a adresa 0.0.0.0/32 vyhradená pre vlastnú identifikáciu stanice, v prípade ako komunikácia s DHCP serverom pred získaním adresy. [6]

Sieť	Adresa siete	Všesmerová adresa	Adresy staníc
10.0.0.0/8	10.0.0.0	10.255.255.255	10.0.0.1 - 10.255.255.254
172.16.0.0/12	172.16.0.0	172.31.255.255	172.16.0.1 - 172.31.255.254
192.168.0.0/16	192.168.0.0	192.168.255.255	192.168.0.1 - 192.168.255.254

Tab. 2.4: Privátne sieťové rozsahy. [2]

2.3 Smerovanie

Prezývané routing je proces, pri ktorom sa hľadá cesta z jedného bodu do iného bodu v rámci prepojených sietí. Tento proces býva realizovaný pomocou smerovačov (routerov) alebo L3 prepínačov (switchov). Podstatou smerovania je vyhľadanie optimálnej cesty z bodu A do bodu B. Optimálnu cestu môžeme definovať ako najkratšiu, najrýchlejšiu, najspoľahlivejšiu a to v závislosti na použitom smerovacom protokole a nastavení jednotlivých parametrov. [6]

Siete sú hierarchicky delené na jednotlivé podsiete (subnety), ktoré sú prepojené pomocou smerovačov. Pokiaľ chce zariadenie (napr. stanica) v jednej podsieti komunikovať so zariadením v druhej podsieti, tak sú dáta zasielané na smerovač, ktorý zariadi ich doručenie. Dĺžka cesty je určená pomocou počtu skokov (hopov), čo je prechod zo zariadenia na zariadenie. Jedná sa o počet smerovačov na ceste plus jedna alebo inak povedané počet liniek na ceste. Používaný je termín ďalší skok (next hop), ktorý označuje adresu ďalšieho smerovača na ceste. [4]

Pri smerovaní dátových jednotiek sa každý smerovač lokálne rozhoduje, kam dáta smerovať. Smerovač je zariadenie pracujúce na tretej vrstve referenčného modelu OSI (sieťová vrstva). Keď hovoríme o dátových jednotkách, ktoré posiela, jedná sa o pakety. Môže existovať viacero ciest k cieľu a smerovač volí vhodné výstupné rozhranie (interface). Lokálne rozhodovanie spočíva na znalosti globálnej topológie, čo charakterizuje primárny problém smerovania. Zložitosť globálnej topológie a jej zmeny v čase sťažujú zber informácii potrebných k smerovaniu. Správne fungovanie smerovania je dosiahnuté pomocou nasledujúcich údajov:

- adresy cieľového zariadenia,
- identifikácie ciest do vzdialených sietí,
- zvolenej najlepšej cesty do cieľovej siete,
- susedných smerovačov, s ktorými si vymieňa informácie o cestách,
- spôsob ako sa dozvedieť o cestách a ako udržiavať dané informácie aktuálne.

2.3.1 Statické smerovanie

Dosiahnuté je manuálne zadanými smerovacími informáciami, ktoré presne stanovujú cestu. Nevýhodou tohto smerovania je, že nereflektuje zmeny v topológii siete. A v prípade zmien alebo porúch je nutná manuálna úprava, inak nebude smerovanie korektné. Klasické použitie je v malých sieťach a pri nastavení predvolenej brány (default gateway alebo gateway of last resort). V prípade, že smerovač nenájde v smerovacej tabuľke vhodnú cestu, posiela pakety na predvolenú bránu. Ako predvolená brána obvykle slúži smerovač, ktorý posiela dáta z lokálnej siete do ďalšej siete. Toto zariadenie sa označuje ako hraničný smerovač, teda na rozhraní sietí. Typické použitie predvolenej brány je pri pripojení do internetu. Jedná sa o poslednú variantu, ktorá sa použije pri smerovaní. Dôležité je nastavenie statickej cesty v oboch smeroch.

2.3.2 Dynamické smerovanie

Spočíva vo výmene smerovacích informácií prostredníctvom smerovacích protokolov. Sieť reaguje na aktuálne potreby a automaticky sa prispôsobuje zmenám v topológii (napr. výpadok linky). Využívajú sa algoritmy (napr. Dijkstrov algoritmus u OSPF) na výpočet vhodnej cesty. Nevýhoda dynamického smerovania, čiže využitie smerovacieho protokolu spočíva v zanášaní oneskorenia do smerovacieho procesu v dôsledku zaťaženia procesoru smerovača v súvislosti so spracovaním paketov. Ďalšie negatívum je vyššie vyťaženie linky z dôvodu výmeny smerovacích informácii medzi smerovačmi v sieti. [6]

2.3.3 Smerovacia tabuľka

Smerovacie informácie sú zaznamenávané do smerovacej tabuľky lokalizovanej v pamäti každého smerovača. Tabuľka pozostáva z nasledujúcich polí:

- cieľ – môže byť adresa podsiete, zariadenia alebo predvolenej trasy,
- maska siete – slúži k určeniu rozsahu sieťových adries, pre ktoré platí záznam v tabuľke,
- brána – udáva adresu najbližšieho priamo pripojeného smerovača (next hop), na ktorý budú pakety odosielané,
- rozhranie – určuje rozhranie, ktoré bude použité pre prenos paketov na nasledujúci smerovač alebo do cieľového zariadenia,
- metrika – udáva relatívnu cenu pre prenos paketov do cieľa pri použití danej trasy, slúži k výberu efektívnej cesty, teda pokiaľ existuje viacero ciest je využitá trasa s najnižšou metrikou. [11]

2.3.4 Smerovanie na sieťovej vrstve

V sieťach TCP/IP je smerovanie založené na adresách podsietí, aby bola dosiahnutá minimalizácia počtu záznamov smerovacích tabuliek. Značná veľkosť by spôsobila zväčšené pamäťové nároky na smerovač, ako aj navýšenie doby vyhľadávania záznamu v tabuľke. Internet je z pohľadu sieťovej vrstvy tvorený jednotlivými podsieťami. Na základe IP adresy príjemcu môže odosielateľ rozpoznať, či sa nachádza v zhodnej (lokálnej) podsieti alebo nie. V prípade, že sú odosielateľ aj adresát v jednej podsieti, je paket zaslaný priamo. Pri nezhode podsiete odosielateľa a príjemcu je dátová jednotka predaná najbližšej predvolenej bráne. Rozhodovanie brány spočíva len na adrese siete príjemcu, ktorá je vypočítaná z cieľovej adresy a masky siete získaných zo smerovacej tabuľky. Paket je predaný ďalšej bráne na základe vyhľadanej zhody adresy cieľovej siete. Danému záznamu v tabuľke smerovača je pridelený nasledujúci skok (next hop), ktorým je dátová jednotka ďalej smerovaná. Keď sa paket dostane do cieľovej siete brána využije zvyšnú časť IP adresy príjemcu (adresu konkrétneho zariadenia v sieti) a doručí paket konečnému adresátovi. [6]

2.3.5 Prepojenie pomocou smerovača

V prípade, že máme dve stanice v rozdielnych podsieťach prepojené pomocou smerovača bude prebiehať doručenie paketu nasledujúcou postupnosťou:

1. Na sieťovej vrstve dôjde k zapúzdreniu segmentu prípadne datagramu do paketu, ktorý bude opatrený hlavičkou obsahujúcou zdrojovú a cieľovú adresu stanice.
2. Zdrojová stanica najskôr otestuje, či sa príjemca nenachádza v rovnakej podsieti, na základe cieľovej adresy.
3. Potom vrstva sieťového rozhrania vytvorí hlavičku ethernetového rámca, kde vloží stanica svoju (zdrojovú) MAC adresu. Cieľová MAC adresa sa priradí k adresátovej IP adrese pomocou dotazu ARP protokolu. Pokiaľ sa príjemca nachádza v zhodnej podsieti ako odosielateľ, tak sa jedná o fyzickú adresu príjemcu. Ak sa nenachádza príjemca v danej podsieti, bude použitá fyzická adresa smerovača (next hop).
4. Rámec je odoslaný zo stanice.
5. Smerovač obdrží rámec a podľa fyzickej adresy vie, že mu patrí.
6. V rámci prepíše zdrojovú MAC adresu na tú svoju a cieľovú MAC adresu na adresu príjemcu. To na základe IP adresy cieľa v hlavičke paketu.
7. Smerovač skontroluje paket (prepočet kontrolného súčtu záhlavia) a zníži hodnotu doby života paketu (TTL), čo je počet skokov.
8. Opätovný test, či sa nachádza adresát v lokálnej sieti niektorého z rozhraní smerovača. Dôjde k vyhľadaniu vhodnej cesty v smerovacej tabuľke a vyslaniu

paketu daným rozhraním.

9. Cieľová stanica príjme rámec podľa MAC adresy.
10. Skontroluje paket a IP adresu. Následne je predaný vyššej vrstve. [3]

3 NS-3

Obsahom tejto kapitoly je priblíženie konkrétnych vlastností sieťového simulátoru ns-3. Na začiatku je základný popis prostredia ns-3, doplnený históriou, vývojom, architektúrou a cieľom jeho použitia. Nasledovaný je prehľadom, ktorý rozoberá štruktúru ns-3 tvorenú softvérovými knižnicami. Ďalšia podkapitola sa venuje kompilátoru Waf a jeho úlohe pri simulácii. Vývojové prostredie uvádza, pre ktorú platformu je sieťový simulátor určený. Jednotlivo sú popísané aj hlavné abstrakcie, čiže aplikácia, kanál, sieťové zariadenie a pomocník tvorby topológie, ktoré tvoria základ pri tvorbe simulácie. Na záver sú uvedené zdroje informácií, primárne sústredené na webových stránkach simulátoru, doplnené detailnou dokumentáciou, taktiež sústredenou na webových stránkach.

3.1 Základný popis

Ns-3 (network simulator) je sieťový simulátor určený prevažne pre výskumné a vzdelávacie účely. Jedná sa o open-source program dostupný pod licenciou GNU GPLv2. Vývoj tohto programu začal v roku 2006. Hlavný cieľ, ktorý si program kladie, je tvorba simulácií v súlade s potrebami moderného sieťového výskumu. Softvér ns-3 umožňuje vývoj realistických simulačných modelov. Podporovaná je aj emulácia sietí v reálnom čase a implementácie protokolov a ich opätovné použitie. [13]

Na vývoji sieťového simulátora sa podieľala skupina ľudí, medzi ktorými boli Tom Henderson a George Riley, ktorí sa snažili vytvoriť náhradu za simulátor ns-2. Na projekte spolupracovalo výskumné centrum INRIA. Prvá vydaná verzia bola ns-3.1 v júni roku 2008 a každý štvrťrok pokračovalo vydanie novej verzie softvéru.

Sieťový simulátor je postavený na základe programovacích jazykov C++ a Python. Líši sa od simulátoru ns-2, ktorý používa jazyk OTcl namiesto jazyka Python. Ns-3 nie je spätne kompatibilný s ns-2 a nepodporuje aplikačné programové rozhranie (API).

Ns-3 poskytuje modely, ktoré ukazujú ako pracujú paketové siete. Taktiež umožňuje realizáciu náročnejších alebo ťažko uskutočniteľných štúdií na reálnych systémoch pre testovanie správania systému vo vysoko kontrolovanom prostredí. Možné je reprodukovanie simulácie pre zistenie ako pracuje daná sieť. Dostupné modely v ns-3 sa sústreďujú na modelovanie internetových protokolov, tie však nelimitujú použitie simulátora.

Dizajn ns-3 pozostáva zo sady vzájomne kombinovateľných knižníc, ktoré sú spojiteľné s ďalšími externými softvérovými knižnicami. Grafické rozhranie ns-3 je modulárne a tvorené niekoľkými externými animačnými a vizualizačnými nástrojmi.

Podporované sú aj nástroje pre analýzu dát. Primárne je simulátor využívaný na Linuxových systémoch. Nie je oficiálne podporovaný produkt žiadnej spoločnosti.

Simulátor ns-3 je celý napísaný v jazyku C++, s možnou väzbou na Python. Z toho vyvodzujeme, že zdrojový kód môže byť napísaný v jazyku C++ alebo Python. Keďže ns-3 generuje trasovacie pakety typu pcap, pre analýzu trasy môžu byť použité aj ďalšie programy. [10]

Tvorba kvalitného sieťového simulátora s dostatočným počtom kvalitných a udržovaných modelov vyžaduje značné množstvo práce. Ns-3 simulátor sa snaží rozdeliť prácu medzi veľkú komunitu používateľov a vývojárov, aby dosiahla tento cieľ.

Každé tri mesiace je vydaná nová stabilná verzia ns-3 s novo vytvorenými modelmi, ktoré sú zdokumentované a udržiavané tímom výskumníkov. [13]

3.2 Prehľad

Základ simulátora ns-3 tvorí spolupracujúci systém softvérových knižníc. Užívateľské programy je možné napísať prepojením s týmito knižnicami alebo importovaním z nich. Keďže ns-3 je distribuovaný v podobe zdrojového kódu, je nevyhnutné, aby mal systém prostredie pre vývoj softvéru určené na budovanie knižníc a následne užívateľského programu. [10]

Ns-3 má simulačné jadro a modely implementované v jazyku C++. Zdrojový kód je uložený prevažne v zložke `src`. Simulačné jadro je vložené v `src/core`. Pakety sú základné objekty sieťového simulátora a sú implementované v zložke `src/network`. Programy simulátora ns-3 môžu pristupovať ku všetkým aplikačným programovým rozhraniam (API) priamo alebo môžu použiť tzv. helper API, ktoré poskytujú vhodné zapúzdrenie nízkoúrovňových volaní API. [8]

Podstatné je uvedenie rozdielu medzi modulmi a modelmi. Softvér ns-3 je organizovaný do separátnych modulov, z ktorých každý je budovaný ako samostatná softvérová knižnica. Individuálne programy môžu prepojiť moduly, ktorých simulácie potrebujú vykonať. Na druhú stranu modely sú abstraktné reprezentácie reálnych objektov, protokolov, zariadení atď. Hocijaký ns-3 modul môže pozostávať z viacerých modelov. [9]

3.3 Waf

Zdrojový kód je nutné po stiahnutí skompilovať pre vytvorenie použiteľných programov. Kompilovanie, inak povedané preklad, je transformáciou zdrojového kódu napísaného v programovacom jazyku do strojového kódu, čiže nižšieho jazyka. Existuje mnoho nástrojov určených pre túto činnosť. Známym je nástroj make, avšak pre

jeho zložité použitie sú vyvíjané alternatívy. Jednou z nich je Waf vyvinutý použitím jazyka Python. [10]

3.4 Vývojové prostredie

Skriptovanie v ns-3 je prostredníctvom jazyka C++ a Python. Modely sú napísané v C++ a väčšina aplikačného programového rozhrania simulátora je dostupná v jazyku Python. Systém ns-3 používa niekoľko súčastí z GNU sady nástrojov (gcc, gdb), ale nepoužíva nástroj make ani GNU kompilačný nástroj. Pre tento prípad slúži Waf.

Ns-3 je určený pre Linuxové prostredie. Typické je pre používateľov operačného systému Windows inštalovanie virtuálneho stroja a do neho nainštalovanie Linuxu. [10]

3.5 Hlavné abstrakcie

3.5.1 Uzol

Ns-3 je sieťový simulátor a nejedná sa špecificky o simulátor Internetu. Výpočtové zariadenie pripojené do siete sa bežne označuje ako hostiteľ (host), ale tento názov sa vzťahuje na sieť Internet a protokoly s ním spojené. My budeme používať pojem uzol (node), ktorý je všeobecnejší a používaný aj u iných simulátorov. Abstrakciou základného výpočtového zariadenia je teda uzol, ktorý je reprezentovaný triedou `Node` v C++. Pre riadenie výpočtového zariadenia v simulácii nám slúžia metódy triedy `Node`. Uzol si môžeme predstaviť ako počítač s pridanými funkciami, ako sú aplikácie alebo protokolovými zásobníkmi. [10]

3.5.2 Aplikácia

Počítačový softvér sa rozdeľuje na systémový a aplikačný softvér. Hranica, ktorá ich rozdeľuje je vytýčená zmenou privilegovanej úrovne (levelu), ktorá sa deje vo volaniach operačného systému (traps). Ns-3 nepracuje s reálnym konceptom operačného systému, privilegovanými úrovňami alebo systémovými volaniami. Rovnako ako aplikácie bežia na reálnych počítačoch, bežia ns-3 aplikácie na uzloch. Aplikácia je základnou abstrakciou pre užívateľský program, ktorý vytvára aktivitu pre simuláciu. Trieda `Application` reprezentuje túto abstrakciu v jazyku C++. Trieda `Application` obsahuje metódy pre správu užívateľských aplikácii v simuláciách. [10]

3.5.3 Kanál

Médium, pomocou ktorého sa dáta dostávajú do siete, označujeme ako kanály (channels). Pripojením sieťového kábla do zásuvky, pripájame počítač ku komunikačnému kanálu. V ns-3 simulácii pripájame uzol k objektu reprezentujúcemu komunikačný kanál. Abstrakcia základnej podsiete pre komunikáciu sa nazýva kanál a je určená triedou `Channel` v C++. Táto trieda poskytuje metódy pre riadenie objektov podsiete a pripája k nim uzly. Kanál v ns-3 môže byť použitý pre reprezentáciu jednoduchej linky, avšak môže modelovať aj zariadenie ako je prepínač. [10]

3.5.4 Sieťové zariadenie

Z hardvérového hľadiska potrebujeme pre pripojenie počítača do siete špecifický sieťový kábel a zariadenie nazývané karta sieťového rozhrania (NIC) alebo bežne len sieťová karta, inštalovaná v počítači. Sieťová karta potrebuje pre svoje fungovanie a riadenie hardvéru softvérový ovládač. U operačného systému Linux je sieťová karta označená ako sieťové zariadenie (network device), ktoré je riadené pomocou ovládačov sieťových zariadení, spoločne nazývaných ako sieťové zariadenia (net devices). Príkladom takéhoto sieťového zariadenia u OS Linuxu je eth0.

Pojem sieťové zariadenie (net device) v simulátore ns-3 zastrešuje softvérový ovládač ako aj hardvér (NIC). Úlohou sieťového zariadenia (net device) je umožniť uzlu komunikáciu s ostatnými uzlami v simulácii cez kanál (channel), preto je inštalovaný v uzle. Uzol (node) môže byť pripojený ku viacerým kanálom (channel) prostredníctvom viacerých sieťových zariadení (net devices). Trieda `NetDevice` obsahuje metódy riadenia pripojenia k uzlu a kanálu. Abstrakcia sieťového zariadenia je určená touto triedou v jazyku C++. [10]

3.5.5 Pomocníci tvorby topológie

Vo veľkých simulovaných sieťach je potrebné zabezpečiť veľké množstvo spojení medzi uzlami, sieťovými zariadeniami a kanálmi. Následne je nutné nastavenie IP adries a mnohých ďalších operácií. Ns-3 poskytuje pomocníkov tvorby topológie (topology helpers) pre zjednodušenie týchto úkonov. [10]

3.6 Zdroje

Primárnym zdrojom informácií pre užívateľov ns-3 simulátora je webová stránka `http://www.nsnam.org`. Poskytuje prístup k hlavným informáciám o ns-3 simulátore. Prostredníctvom ďalšej webovej stránky `http://www.nsnam.org/documentation/` sa dostaneme k detailnej dokumentácii a popisu systémovej architektúry.

Wiki webové stránky obsahujú: podrobné návody ako riešiť problémy, často kladené otázky užívateľov a vývojárov, zdrojové kódy a rôzne iné príspevky.

Na stránke `http://code.nsnam.org/` nájdeme zdrojový kód ns-3, ktorý môžeme študovať. V zložke `ns-3 DCE` sa nachádza aktuálny vývojový strom. A tiež je možné dohľadať staršie vydania alebo aj experimentálne zdroje od rôznych vývojárov.

Na stránke popisujúcej dokumentáciu sa nachádza odkaz pre „API Documentation“, ktorý nás privedie na stránku dokumentácie aplikačného programového rozhrania. Prostredníctvom nástroja pre generovanie online dokumentácie zo zdrojového kódu je zaznamenaná API dokumentácia a organizovaná naprieč rôznymi modulmi. [10]

4 ÚLOHA 1: UNICAST A STATICKÉ SMĚROVÁNÍ

4.1 Zadání

1. Prostudujte si teoretický úvod a zdrojový soubor.
2. Zdrojový soubor *uloha1.cc* dokončete ve zvoleném textovém editoru podle vypracování.
3. Vhodně doplňte cesty na jednotlivé uzly do pasáže statického směrování. Pro zajištění datového toku z uzlu n0 do uzlu n4 přes n3.
4. Upravte metriku, aby byla veškerá komunikace směrována přes uzel n3. Vyzkoušejte si, jak nastavení parametru metriky ovlivní směřování.
5. Změňte nastavení metriky, aby datový tok z uzlu n0 na uzel n4 směřoval přes n2 a tok z uzlu n4 na uzel n0 směřoval přes n3.
6. Vytvořte On / Off aplikaci využívající protokol UDP a použijte port 5060. Zdroj bude Node4 a cíl Node1 přes uzel n2 i n3. Jako zdrojovou IP adresu na uzlu n1 zvolte 192.168.1.2. Nastavte atributy podle vlastního uvážení, čas spuštění (2) a ukončení (7) aplikace. Použijte třídu `UdpSocketFactory`. Tvorba aplikace je velmi podobná aplikaci využívající protokol TCP. Doplňte potřebné směrovací záznamy.
7. Podle vypracování doplňte UDP Echo Server a Client aplikace. Pro správné fungování aplikací přidejte směrovací záznamy na uzly. V závěru vypracování je popis vytvoření aplikací.
8. Zobrazte do terminálového okna střední hodnotu zpoždění „Mean Delay“ a střední hodnotu kolísání zpoždění „Mean Jitter“ úpravou zdrojového souboru sekce FlowMonitor. (Mean Delay = delaySum / rxPackets; Mean Jitter = JitterSum / (rxPackets − 1))
9. Prohlédněte si obsah souboru static-routing se směrovacími tabulkami a zkontrolujte vaše nastavení.
10. V programu Wireshark si prostudujte komunikaci mezi uzly. Všimněte si, jak se zobrazuje číslo portu 80. Prohlédněte si, jak probíhala komunikace u TCP protokolu (spolehlivý, spojově orientovaný protokol) a protokolu UDP.
11. Otevřete NetAnim, načtěte soubor *uloha1.xml*, spusťte simulaci a sledujte průběh komunikace. Změňte barvu vybraného uzlu v animaci a upravte jména jednotlivých uzlů (Node0, Node1, . . .). Pomoc najdete na webových stránkách wiki simulátoru ns-3.
12. Odpovězte na otázky na konci úlohy.

4.2 Teoretický úvod

4.2.1 Unicastová komunikace

Definována je jako přenos mezi jedním odesílatelem a jedním příjemcem. Na rozdíl od broadcastu nebo multicastu, kde je více příjemců daného paketu. Při vysílání totožných dat většímu počtu příjemců je nutné vyslat tolik paketů, kolik je příjemců. V daném případě se nejedná o efektivní způsob, protože každé síťové připojení spotřebovává výpočetní zdroje a vyžaduje vyhradit určitou šířku pásma pro přenos.

4.2.2 Statické směrování

Je proces, při kterém síťový administrátor ručně nakonfiguruje směrovací informace pro úspěšné zasílání paketů. Mluvíme o manuálním přidání cest do směrovací tabulky. Výhoda spočívá v tom, že nedochází k výměně směrovacích informací. Ušetří se šířka pásma a nezatěžují se procesory směrovačů režií. Nevýhodou, kterou přináší statické směrování, je neschopnost dynamicky reagovat na změny v topologii. Například při výpadku linky je nutné, aby administrátor upravil záznamy ve směrovací tabulce, jinak bude vzdálená síť nedostupná.

4.3 Popis zdrojového souboru

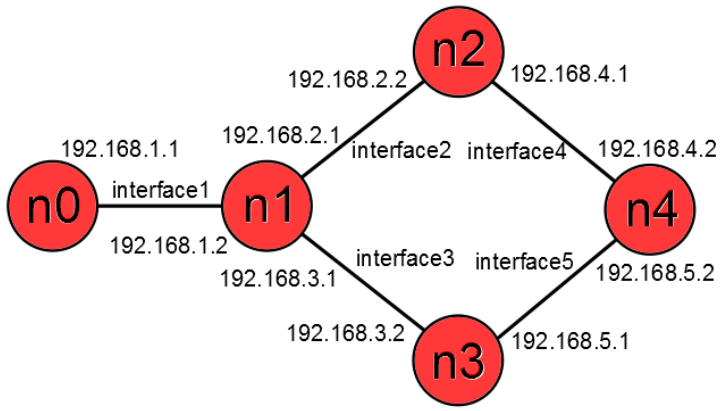
Pro vypracování simulace byl použit operační systém Linux distribuce Ubuntu 14. 04.1 LTS a síťový simulátor ns-3.21. Zdrojový soubor je vytvořen ve zvoleném textovém editoru (např. nano, gedit, . . .) a uložen s příponou *.cc*. My použijeme dopředu vytvořenou topologii (obr. 4.1) uloženou v souboru s názvem *uloha1.cc*. Pro budování ns-3 můžeme využít skript build.py nebo nástroj bake. Pokročilá konfigurace a práce s ns-3 simulátorem, však zahrnuje použití nativního ns-3 nástroje Waf, který využijeme v úlohách. Běh skriptu pod kontrolou nástroje Waf dovolí systému zajistit správné nastavení cest ke sdíleným knihovnám a dostupnost knihoven v průběhu simulace. Poznámky se ve zdrojovém souboru zapisují za znaky `//`.

Vytvořený soubor *uloha1.cc* již obsahuje část kódu, který je následně popsán. Prohlédněte si popis, ve kterém se zaměřte na vytvořené zařízení, rozhraní a jim přidělené adresy. Pochopení vám umožní správně dopracování úlohy. Nekopírujte části kódu z rámečků, které se již nacházejí ve zdrojovém souboru!

Kód začíná s prohlášeními `#include`, čili s přidanými hlavičkovými soubory. Protože nechceme, aby kód obsahoval velké množství přidaných souborů, sdružíme tyto soubory do velkých modulů. Použijeme jeden soubor, který načte všechny z přidaných souborů použitých v každém modulu. Nebude třeba vyhledat konkrétní

hlavičkový soubor, což usnadní psaní skriptu. Všechny přidané soubory jsou umístěny do podadresáře *ns3* (v adresáři *build*), v průběhu kompilace, aby nedošlo ke kolizi jmen přidaných souborů.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
```



Obr. 4.1: Zadaná topologie první ulohy.

Po přidaných souborech následuje deklarace jmenného pole (`namespace`). Implementace ns-3 projektu je v jazyce C++ ve jmenném poli `ns3`. Tato deklarace zajistí, abychom nemuseli psát `ns3::` operátor před celý ns-3 kód, před jeho použitím.

```
using namespace ns3;
```

Dalším řádkem je záznamová funkce, která slouží k zobrazení zpráv. Standardně jsou všechny záznamy zablokovány a tento zápis povolí pozdější selektivní povolování záznamových funkcí.

```
NS_LOG_COMPONENT_DEFINE ("Uloha1LOG");
```

Následuje deklarace funkce programu s názvem `main`, která bude spuštěna jako první. Taktéž je deklarována proměnná `verbose` typu *boolean*, která je přímo inicializována. Může nabývat hodnoty pravda nebo nepravda (`true` / `false`).

```
int main (int argc, char *argv[])  // Deklarace funkce main
{

bool verbose = true;
```

Podle hodnoty proměnné `verbose` jsou povoleny nebo zakázány záznamy pro aplikace Echo Client a Echo Server. Konkrétně se jedná o výpis zpráv zobrazující odesílané a přijímané pakety v průběhu simulace.

```
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

Dopracovali jsme se k samotné tvorbě topologie. Následující řádky kódu definují vytvoření uzlů, tedy objektů třídy `Node`. Pomocník tvorby topologie `NodeContainer` nám umožní pohodlné vytvoření, řízení a přístup k uzlům vytvořeným pro tuto simulaci. První řádek deklaruje objekt třídy `NodeContainer` s názvem `n`. Ve druhém řádku je volána metoda `Create` na objekt `n`, aby vytvořila námi definovaný počet uzlů (pět). Následující řádek vytvoří ukazatel `Node0` na první uzel. Je třeba si uvědomit, že index prvního uzlu je nula. Obdobná je tvorba ukazatelů pro všechny další uzly.

```
NodeContainer n;  // Vytvoření kontejneru uzlů
n.Create (5);
Ptr<Node> Node0 = n.Get (0);  // Vytvoření ukazatele na uzel
```

Máme vytvořené samotné uzly. Konstrukce sítě pokračuje propojením uzlů pomocí linek typu point–to–point. Znovu použijeme pomocníka tvorby topologie `PointToPointHelper` pro vytvoření objektu nazvaného `pointToPoint1`. Daná třída nám pomůže nakonfigurovat a propojit objekty `PointToPointNetDevice` a `PointToPointChannel`. Nadefinujeme vlastnosti vytvořeného objektu `pointToPoint1` pomocí atributů „DataRate“ (přenosová rychlost) a „Delay“ (zpoždění).

```
// Vytvoření point-to-point linek
PointToPointHelper pointToPoint1;
pointToPoint1.SetDeviceAttribute ("DataRate", StringValue ("5Mbps")
    );
pointToPoint1.SetChannelAttribute ("Delay", StringValue ("3ms"));
```

Obdobně jako jsme použili pomocníka `NodeContainer` pro vytvoření uzlů, nám pomůže `NetDeviceContainer` při vytvoření, konfiguraci a instalaci zařízení. Pojem zařízení v podstatě označuje síťovou kartu. Kontejner `NetDeviceContainer` bude obsahovat list všech vytvořených zařízení. Pro každý uzel v kontejneru uzlů je vytvořen `PointToPointNetDevice` a uložen v kontejneru zařízení. Dvě `PointToPointNetDevice` zařízení jsou připojena na vytvořený `PointToPointChannel` kanál. Po volání metody `Install` bude na obou uzlech (ukazatele na první dva uzly) instalované zařízení a kanál. Postup je replikován i pro ostatní zařízení.

```
NetDeviceContainer devices1;
// Nainstalování zařízení do uzlů
devices1 = pointToPoint1.Install (Node0, Node1);
```

Protokolová sada je dalším bodem. Pomocná třída `InternetStackHelper` nám zjednoduší instalaci internetové sady protokolů na uzly v kontejneru.

```
InternetStackHelper stack;
// Nainstalování protokolové sady na uzly z kontejneru
stack.Install (n);
```

Přidělení IP adres zařízením na vytvořených uzlech je řešeno pomocnou třídou `Ipv4AddressHelper`. Přitom je deklarován objekt této třídy a IP adresy jsou alokovány ze sítě 192.168.1.0 s maskou 255.255.255.0. První adresa přidělena z rozsahu bude 192.168.1.1 (interface1 -> devices1 -> Node0) a následována bude IP adresou 192.168.1.2 (interface1 -> devices1 -> Node1). Ns-3 systém zajišťuje, aby nedošlo k přidělení jedné adresy vícekrát, a kdyby nastala daná situace, systém vykáže chybovou hlášku. Prostřednictvím objektu `Ipv4Interface` vytvoříme spojení mezi IP adresou a zařízením. List objektů `Ipv4Interface` je uložen v příslušném kontejneru rozhraní `Ipv4InterfaceContainer`.

```
Ipv4AddressHelper address1;
address1.SetBase ("192.168.1.0", "255.255.255.0");
// Propojení IP adresy a rozhraní zařízení
Ipv4InterfaceContainer interface1 = address1.Assign (devices1);
```

4.4 Vypracování

Přiložený soubor *uloha1.cc* doplňte do vyznačených oblastí zdrojovým kódem z rámečků. Vytvořili jsme síť a potřebujeme zajistit směrování v této síti. Máme ukazatel ukazující na první uzel, který má přičleněnou IPv4 implementaci. S pomocí volání metody `GetObject<IPv4>` získáme ukazatel na IPv4 objekt předtím přičleněn uzlu. Nadefinujeme typ směrovacího protokolu, tedy nastavíme statické směrování. Na jednotlivých uzlech povolíme statické směrování. Metoda `GetStaticRouting` zajistí vyhledání statického směrování. Pro vytvoření cesty v globální směrovací tabulce na daném uzlu použijeme metodu `AddHostRouteTo`. První IP adresa definuje cíl cesty a druhá IP adresa další skok (next hop), tedy nejbližší zařízení na trase do cíle. Číslo za IP adresami definuje rozhraní určené k zaslání paketů do cíle. Poslední parametr udává metriku, jejíž standardní hodnota je nula. Nutné je přidat hlavičkový soubor do zdrojového kódu (je přidán).

```
Ptr<Ipv4> ipv4N0 = Node0->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4N1 = Node1->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4N2 = Node2->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4N3 = Node3->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4N4 = Node4->GetObject<Ipv4> ();
```

```
// Povolení statického směrování
Ipv4StaticRoutingHelper ipv4RoutingHelper;

Ptr<Ipv4StaticRouting> staticRoutingN0 = ipv4RoutingHelper.
    GetStaticRouting (ipv4N0);
Ptr<Ipv4StaticRouting> staticRoutingN1 = ipv4RoutingHelper.
    GetStaticRouting (ipv4N1);
Ptr<Ipv4StaticRouting> staticRoutingN2 = ipv4RoutingHelper.
    GetStaticRouting (ipv4N2);
Ptr<Ipv4StaticRouting> staticRoutingN3 = ipv4RoutingHelper.
    GetStaticRouting (ipv4N3);
Ptr<Ipv4StaticRouting> staticRoutingN4 = ipv4RoutingHelper.
    GetStaticRouting (ipv4N4);

staticRoutingN0->AddHostRouteTo (Ipv4Address ("192.168.4.2"),
    Ipv4Address ("192.168.1.2"), 1);  // Cesta z n0 do n4 přes n1
staticRoutingN0->AddHostRouteTo (Ipv4Address ("192.168.5.2"),
    Ipv4Address ("192.168.1.2"), 1);  // Cesta z n0 do n4 přes n1

staticRoutingN1->AddHostRouteTo (Ipv4Address ("192.168.4.2"),
    Ipv4Address ("192.168.2.2"), 2, 5);  // Cesta z n1 do n4 přes n2
staticRoutingN1->AddHostRouteTo (Ipv4Address ("192.168.5.2"),
    Ipv4Address ("192.168.2.2"), 2, 5);  // Cesta z n1 do n4 přes n2
staticRoutingN2->AddHostRouteTo (Ipv4Address ("192.168.5.2"),
    Ipv4Address ("192.168.4.2"), 2);  // Cesta z n2 do n4 přes n4

staticRoutingN4->AddHostRouteTo (Ipv4Address ("192.168.1.1"),
    Ipv4Address ("192.168.4.1"), 1, 5);  // Cesta z n4 do n0 přes n2
staticRoutingN2->AddHostRouteTo (Ipv4Address ("192.168.1.1"),
    Ipv4Address ("192.168.2.1"), 1);  // Cesta z n2 do n0 přes n1
```

Aplikace je jedna z hlavních abstrakcí systému ns-3. V našem skriptu použijeme On / Off aplikaci vytvořenou pomocí třídy `OnOffHelper`. Při této aplikaci byl použit TCP protokol. `Ns3::TcpSocketFactory` je název protokolu, který slouží aplikaci k odeslání dat. Tento řetězec identifikuje typ továrny socketu, použité k vytvoření socketu pro aplikace. `TcpSocketFactory` je třída používaná instancí protokolu transportní vrstvy pro vytvoření TCP socketu. Socket je definován IP adresou a číslem portu. Příjemcem dat je pátý uzel (Node4), z čehož vyplývá, že se jedná o `devices4` a tedy o `interface4`. Jako cílový port, byl zvolen port 80. IP adresu rozhraní získáme pomocí metody `GetAddress`. Důležité je nastavení atributů přenosová rychlost „DataRate“ a velikost přenášených uživatelských dat (payload) paketu „PacketSize“. Zapnutá bude aplikace stále „OnTime“ a vypnutá nebude nikdy „OffTime“. Aplikaci nainstalujeme na první uzel (ukazatel Node0) a nastavíme začátek a konec jejího běhu. PacketSinkHelper vytváří socket (IP adresa zdroje dat

a cílový port) použitím třídy `TcpSocketFactory`. Jedná se o příjemce paketů pro TCP spojení na pátém uzlu (ukazatel Node4) s určeným časem pro začátek a konec běhu aplikace.

```
// Vytvoření On/Off aplikace využívající TCP protokol
OnOffHelper onoffTCP ("ns3::TcpSocketFactory", Address (
    InetSocketAddress (interface4.GetAddress (1), 80)));

onoffTCP.SetAttribute ("DataRate", StringValue ("0.3Mbps"));
onoffTCP.SetAttribute ("PacketSize", UintegerValue (512));
onoffTCP.SetAttribute ("OnTime", StringValue ("ns3::
    ConstantRandomVariable[Constant=1]"));
onoffTCP.SetAttribute ("OffTime", StringValue ("ns3::
    ConstantRandomVariable[Constant=0]"));

ApplicationContainer appTCP = onoffTCP.Install (Node0);
appTCP.Start (Seconds (1.0));
appTCP.Stop (Seconds (10.0));

// Vytvoření příjemce paketů
PacketSinkHelper sinkTCP ("ns3::TcpSocketFactory", Address (
    InetSocketAddress (Ipv4Address::GetAny (), 80)));
appTCP = sinkTCP.Install (Node4);
appTCP.Start (Seconds (1.0));
appTCP.Stop (Seconds (10.0));
```

Sledovací systém nám zpřístupní detailní informace o posílaných paketech. Výstupem sledování bude ASCII soubor s názvem *uloha1.tr*. Umístění těchto řádků kódu je těsně nad funkci `Simulator::Run`. Volání metody `EnableAsciiAll` povolí ASCII sledování trasy na všech zařízeních point–to–point a zapisování informací o pohybu paketů v ASCII formátu.

Trasovací (sledovací) soubory lze vytvořit i ve formátu *pcap*, což znamená packet capture (zachycení paketů). Vhodný program pomocí, kterého si zobrazíme výstup je Wireshark. Další varianta je tcpdump. Do zdrojového kódu jsme vložili pouze název souboru bez přípony, protože parametrem není celý název souboru, no jen předpona. Vytvořený bude trasovací soubor pro každé point–to–point zařízení v topologii. Název vytvořeného trasovacího souboru bude poskládán z předpony, čísla uzlu, čísla zařízení a přípony „.pcap“. Dohromady bude vytvořeno deset souborů ve formátu *pcap*.

```
// Vytvoření trasovacích souborů
AsciiTraceHelper ascii;
pointToPoint1.EnableAsciiAll (ascii.CreateFileStream ("uloha1.tr"))
    ;
pointToPoint1.EnablePcapAll ("uloha1");
```

Pro výpis směrovací tabulky všech uzlů v určitý čas použijeme metodu `Print RoutingTableAllAt`. Výstup bude zapsán do souboru s názvem *static-routing*.

```
// Výpis směrovací tabulky do souboru
Ipv4GlobalRoutingHelper table;
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper
    > ("static-routing", std::ios::out);
table.PrintRoutingTableAllAt (Seconds (10), routingStream);
```

Pro vytvoření grafu s minimálním počtem řádků kódu slouží třída `Gnuplot Helper`, která vytvoří tři soubory (datový soubor „.dat“, kontrolní soubor „.plt“ a shell skript pro generování gnuplot „.sh“) na konci simulace. V prvé řadě je třeba nakonfigurovat graf pomocí metody `ConfigurePlot`. Definované jsou následující parametry: název souboru, název grafu, popis osy X, popis osy Y a typ výstupního souboru (standardně png). Dále připojíme sondu (Probe), jejíž argumenty jsou: typ sondy, cesta pro přístup sondy, připojen výstup sondy, popis v legendě, poloha legendy v grafu. Metoda `PlotProbe` je použita pro vykreslení hodnot generovaných sondou. Připojené budou dvě sondy stejného typu (první argument v metodě `PlotProbe`). První sonda bude zachycovat pakety odeslané (`Tx`) z uzlu Node0 (`/NodeList/0/`) a druhá sonda bude zachycovat pakety přijaté (`Rx`) na uzel Node0 (`/NodeList/0/`). Jedná se o druhý argument v metodě `PlotProbe`, tedy cesta pro přístup sondy.

```
GnuplotHelper plotHelper;

plotHelper.ConfigurePlot ("gnuplotPackets",
                          "Velikost paketů na uzlu Node0 za čas
                             aplikace On/Off protokolu TCP",
                          "Čas [s]",
                          "Velikost paketu [B]",
                          "png");

plotHelper.PlotProbe ("ns3::Ipv4PacketProbe",
                      "/NodeList/0/$ns3::Ipv4L3Protocol/Tx",
                      "OutputBytes",
                      "Pakety odeslané z uzlu Node0",
                      GnuplotAggregator::KEY_BELOW);

plotHelper.PlotProbe ("ns3::Ipv4PacketProbe",
                      "/NodeList/0/$ns3::Ipv4L3Protocol/Rx",
                      "OutputBytes",
                      "Pakety přijaté na uzel Node0",
                      GnuplotAggregator::KEY_BELOW);
```

Modul Flow Monitor poskytuje flexibilitu při měření výkonu síťových protokolů. Používá sondy instalovány v uzlech na sledování výměny paketů mezi uzly a měření specifických parametrů. Pakety jsou členěny prostřednictvím toků, do kterých patří na základě charakteristiky sondy (protokolu, zdrojové, cílové IP adresy, čísla portu). Mezi základní parametry se řadí identifikace toku „Flow ID“, počet přijatých paketů „Rx Packets“, počet odeslaných bajtů „Tx Bytes“ a další. `NS_LOG_UNCOND` umožňuje výpis parametrů do terminálového okna. Sestavením vztahů ze seznamu parametrů můžeme vypočítat další charakteristiky například propustnost „Throughput“. Použití modulu je zjednodušeno pomocí třídy `FlowMonitorHelper` a výstup je uložen do souboru *uloha1Flowmon* ve formátu *xml*. Nesmíme zapomenout na přidání hlavičkového souboru do zdrojového kódu (je přidán).

```
FlowMonitorHelper flowHelper;
Ptr<FlowMonitor> flowMonitor;
flowMonitor = flowHelper.InstallAll();

Simulator::Stop (Seconds (11));  // Zastavení simulace

// NetAnim
  // Vložení kódu pro NetAnim
// End NetAnim

Simulator::Run ();

flowMonitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier
   > (flowHelper.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = flowMonitor->
   GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
   stats.begin (); i != stats.end (); ++i)
  {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->
       first);
    NS_LOG_UNCOND( "\n  Flow ID: " << i->first << "   Src Addr: "
       << t.sourceAddress << "   Dst Addr: " << t.
       destinationAddress);
    NS_LOG_UNCOND( "  Tx Bytes: " << i->second.txBytes);
    NS_LOG_UNCOND( "  Rx Bytes: " << i->second.rxBytes);
    NS_LOG_UNCOND( "  Tx Packets: " << i->second.txPackets);
    NS_LOG_UNCOND( "  Rx Packets: " << i->second.rxPackets);
    NS_LOG_UNCOND( "  Mean Delay: " << i->second.delaySum.
       GetSeconds () / (i->second.rxPackets) * 1000 << " ms");
    NS_LOG_UNCOND( "  Mean Jitter: " << i->second.jitterSum.
       GetSeconds () / (i->second.rxPackets - 1) * 1000 << " ms");
```

```
    NS_LOG_UNCOND( "  Throughput: " << i->second.rxBytes * 8.0 / (i
       ->second.timeLastRxPacket.GetSeconds () - i->second.
       timeFirstTxPacket.GetSeconds ()) / 1024  << " Kbps");
    }
flowMonitor->SerializeToXmlFile("uloha1Flowmon", true, true);
```

Animace je důležitým nástrojem pro síťovou simulaci. Ns-3 neobsahuje žádný standardní grafický animační nástroj. NetAnim je jedna z možností, jak poskytnout animaci. Jedná se o samostatný software používající sledovací soubory, generované v průběhu simulace pro zobrazení topologie a animace toků paketů mezi uzly. Třída `AnimationInterface` je zodpovědná za vytváření sledovacího XML souboru *uloha1.xml*. Potřebné je nastavit pozici jednotlivých uzlů v animaci. Slouží k tomu metoda `SetConstantPosition`, kde první atribut je ukazatel na uzel a zbylé dva udávají pozici na osách X a Y. Další užitečná metoda je `EnablePacketMetadata` zobrazující meta informace o paketech. Konkrétně se jedná o zobrazení IP adres zdroje a cíle pro On / Off aplikace a čísla portů pro Echo Client a Echo Server aplikace. Nezbytné je přidání hlavičkového souboru do zdrojového kódu (je přidán). Po zkompilování se v terminálovém okně zobrazí varování, že nebyl použit model mobility a je třeba pro jednotlivé uzly nastavit konstantní polohu. Protože jsme konstantní polohu nastavili, netřeba se znepokojovat daným výpisem.

```
AnimationInterface anim (animFile);

anim.SetConstantPosition (Node0, 0, 10);  // Nastavení pozice uzlu
anim.SetConstantPosition (Node1, 5, 10);
anim.SetConstantPosition (Node2, 10, 5);
anim.SetConstantPosition (Node3, 10, 15);
anim.SetConstantPosition (Node4, 15, 10);
anim.EnablePacketMetadata (true);
```

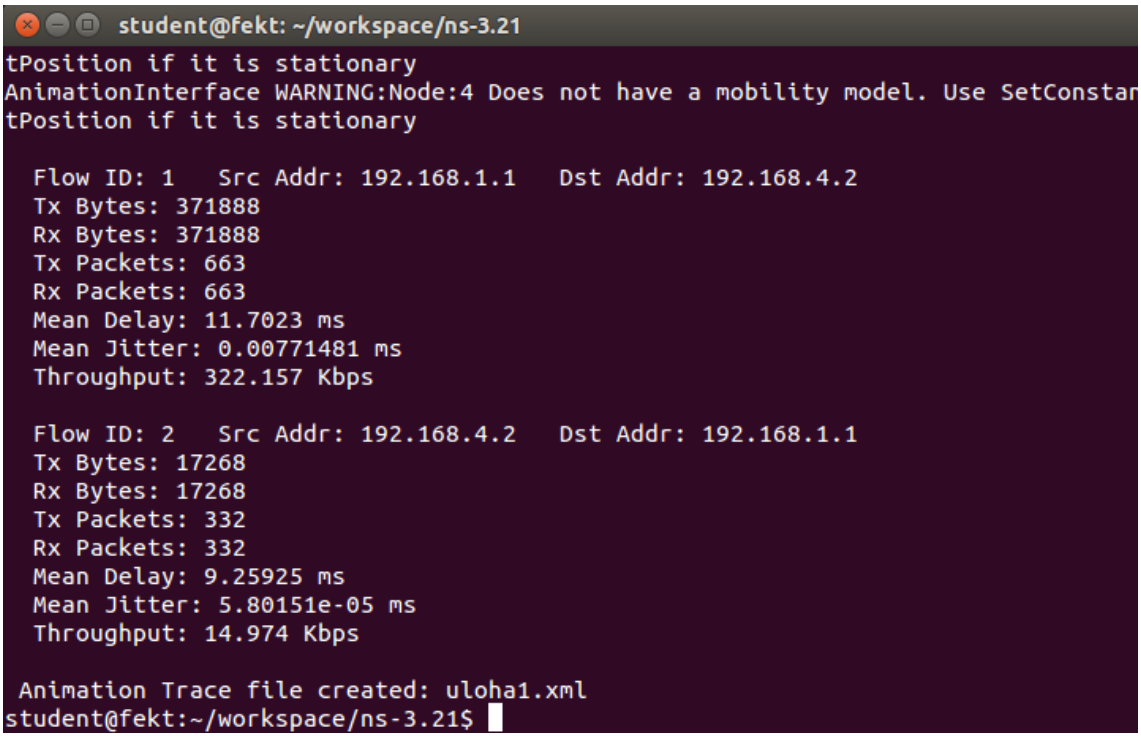
Spuštění samotné simulace realizujeme pomocí globální funkce `Simulator::Run`. Voláním metod `Start` a `Stop` u klienta a serveru aplikace jsme načasovali události v simulátoru. Když zavoláme funkci `Simulator::Run`, systém prohlédne list naplánovaných událostí a provede jej. Na začátku simulace naváže klient spojení se serverem. Po dokončení přenosu dat, v našem případě definovaném časovým údajem Stop, přejde simulace do stavu nečinnosti. Další plánovaná událost je v tomto okamžiku jen zastavení aplikace pro server i klienta. Protože již není nic víc naplánováno, simulace se ukončí. Posledním zbývajícím úkolem je úklid, který má na starosti globální funkce `Simulator::Destroy`. Vymaže všechny objekty vytvořené během simulace a my nemusíme udržovat přehled o vytvářených objektech.

Kompilace vytvořeného skriptu je relativně jednoduchá. V prvé řadě je třeba změnit adresář, ve kterém se nacházíme. Nesmíme opomenout vložení souboru

uloha1.cc do adresáře *scratch* (*/workspace/ns-3.21/scratch*). Následně provedeme kompilaci v terminálovém okně pomocí nástroje Waf (musíme se nacházet v adresáři ns-3.21). Po úspěšné kompilaci dostaneme do terminálového okna výpis `'build' finished successfully`. Simulaci spustíme pomocí názvu souboru bez přípony.

```
$ cd wokrspace/ns-3.21
$ ./waf
$ ./waf --run scratch/uloha1
```

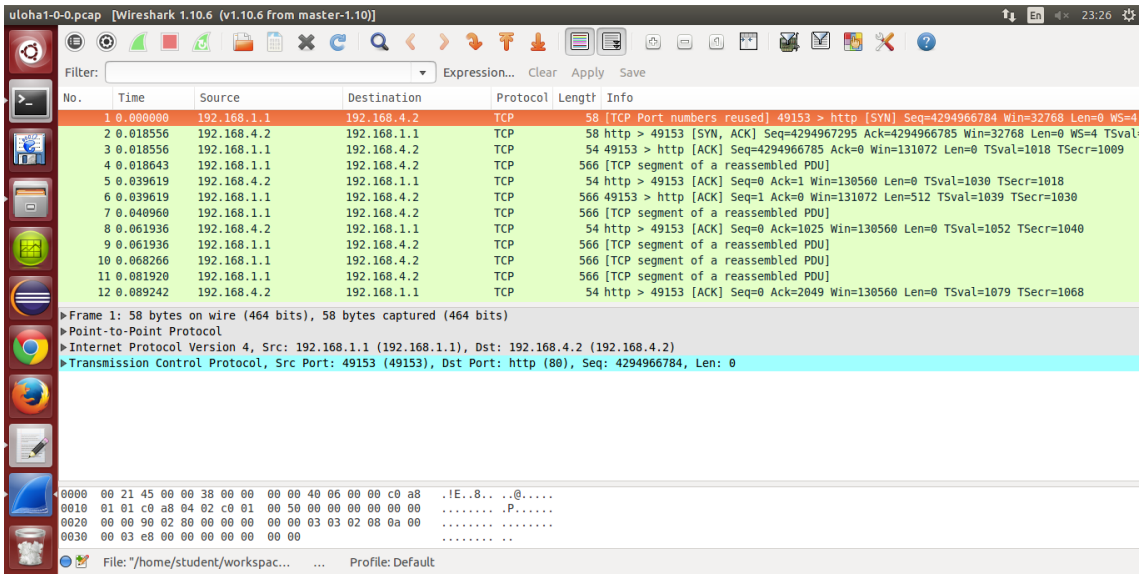
V terminálovém okně (obr. 4.2) vidíme dva datové toky (Flow ID 1 a 2) On / Off aplikace využívající protokol TCP. Z vypracování by mělo být zřejmé, o jaké toky se jedná. První definuje přenos z uzlu n0 do uzlu n4 a druhý přenos z uzlu n4 do uzlu n0.



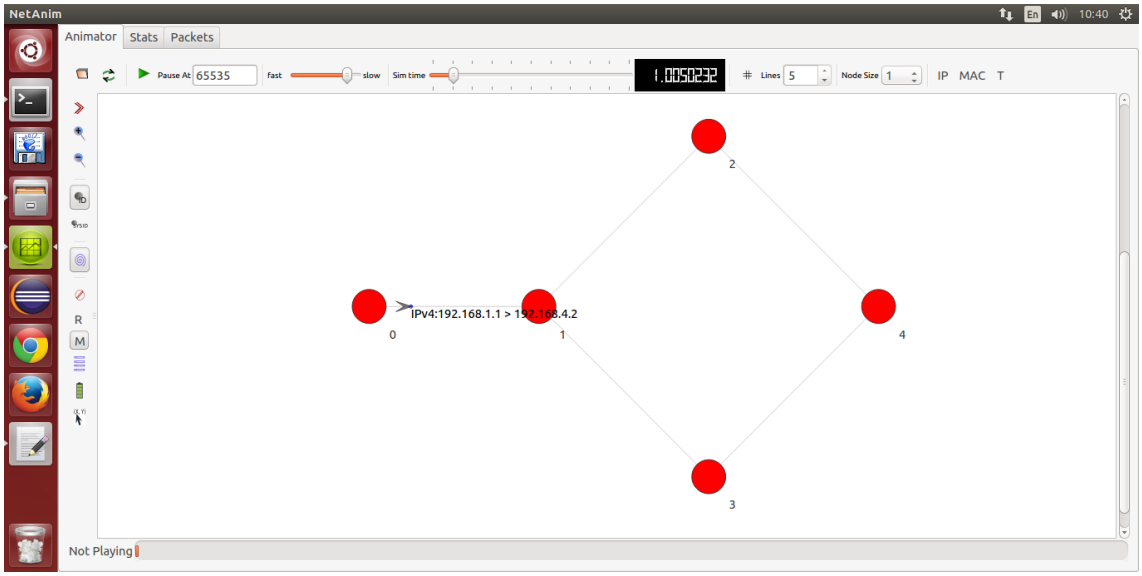
Obr. 4.2: Výpis v terminálu pro On / Off aplikaci využívající protokol TCP.

V adresáři ns-3.21 se vytvořilo deset trasovacích souborů typu *pcap* (všech zařízení a uzlů), trasovací soubor *uloha1.tr*, spustitelný soubor *uloha1.xml* pro NetAnim, *gnuplotPackets.dat*, *gnuplotPackets.plt*, *gnuplotPackets.sh* a soubor *static-routing* obsahující směrovací tabulky jednotlivých uzlů. Prohlédněte si trasovací soubory (obr. 4.3) pro jednotlivé uzly ve Wiresharku (uzly n0 a n4). Soustřeďte se na IP adresy, typ protokolu, čísla portů a samotný průběh komunikace. Otevřete si program NetAnim a spusťte v něm soubor *uloha1.xml* (obr. 4.4). Prohlédněte si, jak

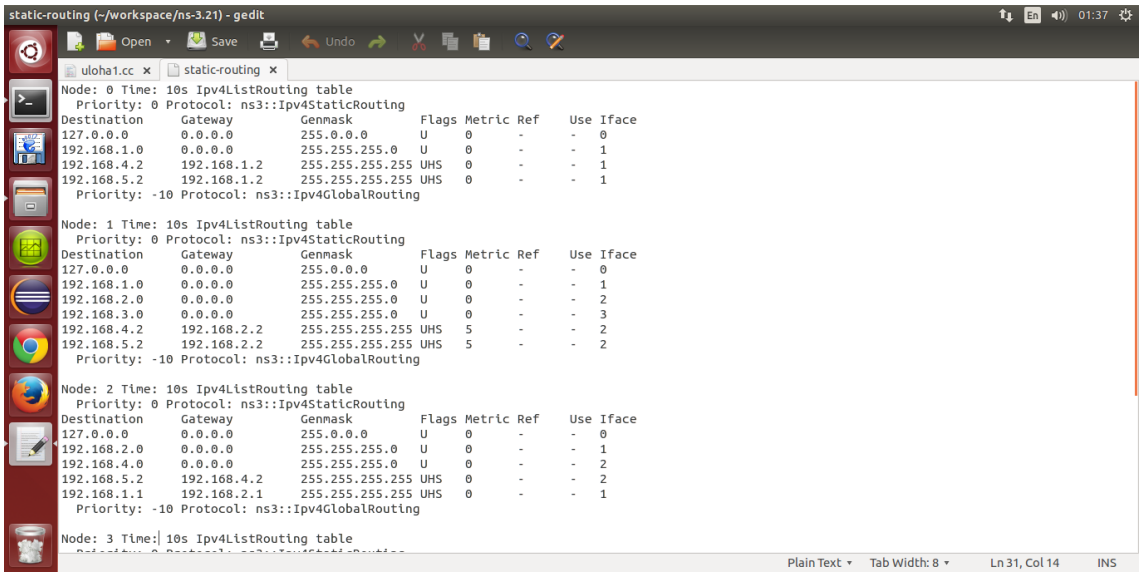
probíhá posílání dat mezi jednotlivými uzly sítě. Pro lepší pochopení klikněte na záložku Packets a následně analyzujte probíhající komunikaci. Otevřete si soubor *static-routing* (obr. 4.5) a projděte si směrovací tabulky uzlů.



Obr. 4.3: Zobrazení komunikace na prvním uzlu pro On / Off aplikaci využívající protokol TCP.



Obr. 4.4: Zobrazení simulace v programu NetAnim pro On / Off aplikaci využívající protokol TCP.

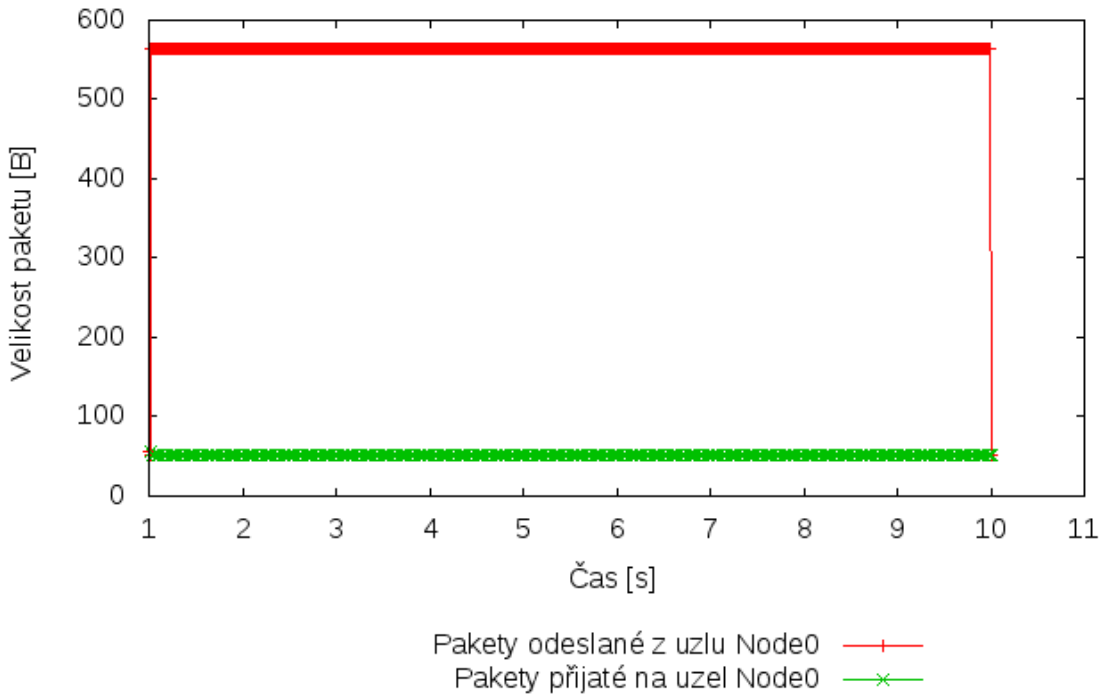


Obr. 4.5: Obsah souboru static-routing pro On / Off aplikaci využívající protokol TCP.

Pro vytvoření soubor *gnuplotPackets.png* s grafem je nutné zadat příkaz `gnuplot gnuplotPackets.plt` do terminálového okna. Vytvořený graf (obr. 4.6) zobrazuje velikost paketů v bajtech za čas na uzlu Node0 pro aplikaci On / Off využívající protokol TCP. Pakety odeslané z uzlu Node0 jsou zobrazeny červenou barvou a pakety přijaté na uzel Node0 mají zelenou barvu. Protože se jedná o protokol TCP je nejdříve sestaveno spojení. Zaslaný je požadavek SYN (o velikosti 58 bajtů) z uzlu Node0 na uzel Node4 přičemž odpovědí je platný požadavek SYN se SYN/ACK (o velikosti 58 bajtů) a nakonec uzel Node0 odpoví potvrzením ACK (o velikosti 54 bajtů). Toto navázání spojení je označeno jako „3-way handshake“. Ukončení spojení je pomocí požadavku FIN (o velikosti 54 bajtů) se strany příjemce i odesílatele. Komunikaci vidíme ve Wiresharku, ze souboru *uloha1-0-0.pcap*.

Na začátku komunikace kolísá velikost paketů při sestavování spojení (pozorujeme u odeslaných paketů zobrazených červeně) a následně jsou z uzlu Node0 odesílané pakety konstantní velikosti (566 bajtů) nesoucí přenášená data. Velikost příchozích paketů se nemění (54 bajtů). Jedná se o potvrzení ACK, které hovoří o úspěšném doručení paketu z uzlu Node0 na uzel Node4 a o sekvenčním čísle následujícího paketu vyslaného z uzlu Node0. Při kopírování obsahu z rámečků dbejte, aby byla dodržena korektní syntaxe!

```
$ cd wokrspace/ns-3.21
$ gnuplot gnuplotPackets.plt
```



Obr. 4.6: Velikost paketů na uzlu Node0 za čas chodu aplikace On / Off využívající protokol TCP.

4.4.1 Aplikace bodu sedm zadání

Tvorba aplikací bodu sedm zadání. Aplikace, které použijeme, jsou specializace třídy `Application` nazývané `UdpEchoServerApplication` a `UdpEchoClientApplication`. S použitím pomocných objektů `UdpEchoServerHelper` a `UdpEchoClientHelper` se nám snadněji nakonfigurují základní objekty. Vytvoříme UDP Echo Server aplikaci na zvoleném uzlu. `UdpEchoServerHelper` je objekt, který nám pomůže vytvořit aplikaci. Tomuto objektu dodáme atribut číslo portu, na základě čehož můžeme volit, o jakou aplikaci se jedná. Metoda `Install` objektu `UdpEchoServerHelper` zajistí připojení aplikace na zvolený uzel. Pomocí metod `Start` a `Stop` nadefinujeme začátek a konec generování provozu aplikace.

```
// Definuje číslo portu, na který se dotazuje client (port serveru)
UdpEchoServerHelper echoServer (80);

// Definuje uzel, na kterém je server
ApplicationContainer serverApps = echoServer.Install (Node3);
serverApps.Start (Seconds (3.0));
serverApps.Stop (Seconds (10.0));
```

Realizace Echo Client aplikace je podobná serveru. K dispozici je objekt `UdpEchoClientHelper` pro zprávu aplikace `UdpEchoClientApplication`. Počet atributů se však liší od serveru. První dva definují vzdálenou IP adresu a vzdálené číslo portu. Pro udržení seznamu IP adres přidělených jednotlivým zařízením jsme použili kontejner `Ipv4InterfaceContainer`. Čtvrtému rozhraní v kontejneru rozhraní bude odpovídat IP adresa čtvrtého uzlu v kontejneru uzlů. Klientovi nastavíme vzdálenou IP adresu (adresa, na kterou bude zasílat dotazy), která je přidělena čtvrtému uzlu (ukazatel Node3). Na tomto uzlu je nastaven server. Nutné je definovat i číslo vzdáleného portu, na který bude klient posílat dotazy. Je to námi určený port serveru. Nastavíme i ostatní atributy. Maximální počet paketů poslaných během simulace „MaxPackets“, doba čekání mezi zasláním dvou paketů „Interval“ a velikost přenášených uživatelských dat (payload) paketu „PacketSize“ udaná v bajtech. Určíme i začátek a konec provozu klientské aplikace.

```
UdpEchoClientHelper echoClient (interface3.GetAddress (1), 80);

echoClient.SetAttribute ("MaxPackets", UintegerValue (7));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

// Definuje uzel, na kterém je client
ApplicationContainer clientApps = echoClient.Install (Node0);
clientApps.Start (Seconds (3.0));
clientApps.Stop (Seconds (10.0));
```

4.5 Kontrolní otázky

1. Vysvětlete rozdíl mezi unicastem a multicastem.
2. Jaký je význam metriky při směrování a co ovlivňuje?
3. Jakou formu mají záznamy ve směrovací tabulce?
4. Porovnejte výhody a nevýhody statického směrování.
5. Uveďte základní rozdíl mezi UDP a TCP protokolem a prezentujte jej pomocí programu Wireshark.
6. Jaký protokol aplikační vrstvy používá port 80?

5 ÚLOHA 2: POROVNÁNÍ ICMPV4 A ICMPV6

5.1 Zadání

1. Prostudujte si teoretický úvod.
2. Vytvořte pomocí vypracování soubor *uloha2a.cc*.
3. Spusťte soubor *icmpv4.xml* v NetAnime a zobrazte, jak probíhá komunikace. Ověřte přítomnost konfigurovaných statických cest ve směrovacích tabulkách R1 (Node0) a R2 (Node2) souboru *icmpv4-routing*.
4. Zaměřte se na strukturu IPv4 paketu ve Wiresharku v souboru *icmpv4-0-0.pcap*. V záložce Internet Protocol Version 4 si všimněte pole Protocol, které určuje jaká datová jednotka vyšší vrstvy je v IPv4 paketu zapouzdřená. Zjistěte číselnou hodnotu v poli Protocol pro ICMP zprávu.
5. V programu Wireshark, souboru *icmpv4-0-0.pcap*, pro jednotlivé typy zpráv Echo Request (Požadavek na odezvu) a Echo Reply (Odezva na požadavek) najděte jejich číselné hodnoty v záložce označené Internet Control Message Protocol v poli Type a Code pro jednotlivé datové jednotky.
6. Změnou čísla vzdáleného portu u UDP Echo Client aplikace dosáhnete zaslání zprávy ICMP Destination Unreachable (Cíl nedostupný). Projděte si pozměněnou animaci v NetAnimu.
7. Prohlédněte si trasovací soubory a pro typ zpráv Destination Unreachable najděte číselné hodnoty v záložce označené Internet Control Message Protocol v poli Type a Code pro jednotlivé datové jednotky v programu Wireshark.
8. Úpravou vzdálené adresy UDP Echo Client aplikace, dosáhnete zaslání chybové zprávy Time Exceeded (Čas překročen). Sledujte, jak se změnil průběh komunikace pomocí NetAnimu.
9. Pro jednotlivé uzly si projděte komunikaci v programu Wireshark. Najděte číselné hodnoty pro typ zpráv Time Exceeded, podobně jako u zpráv Destination Unreachable. Zjistěte také ze zobrazeného trasovacího souboru *icmpv4-0-0.pcap* počáteční hodnotu Time to live (počet skoků).
10. Vytvořte pomocí vypracování soubor *uloha2b.cc*.
11. Spusťte soubor *icmpv6.xml* v NetAnime a zobrazte, jak probíhá komunikace. Ověřte přítomnost konfigurovaných výchozích cest ve směrovacích tabulkách R1 (Node0) a R2 (Node2) souboru *icmpv6-routing*. V programu Wireshark si projděte komunikaci z trasovacích souborů formátu *pcap*.
12. Soustřeďte se na strukturu IPv6 paketu ve Wiresharku. V záložce Internet Protocol Version 6 si všimněte pole Next header, které určuje jaká datová jednotka vyšší vrstvy je v IPv6 paketu zapouzdřená. Zjistěte číselnou hodnotu

v poli Next header pro ICMPv6 zprávu.

13. V programu Wireshark, souboru *icmpv6-0-0.pcap*, pro jednotlivé typy zpráv Echo Request (Požadavek na odezvu) a Echo Reply (Odezva na požadavek) najděte jejich číselné hodnoty v záložce označené Internet Control Message Protocol Version 6 v poli Type a Code pro jednotlivé datové jednotky.
14. Změnou čísla vzdáleného portu u UDP Echo Client aplikace dosáhnete zaslání zprávy ICMPv6 Destination Unreachable (Cíl nedostupný). Projděte si pozměněnou animaci v NetAnimu.
15. Podrobně si prohlédněte trasovací soubory a pro typ zpráv Destination Unreachable najděte číselné hodnoty v záložce označené Internet Control Message Protocol Version 6 v poli Type a Code pro jednotlivé datové jednotky v programu Wireshark.
16. Úpravou vzdálené adresy UDP Echo Client aplikace, dosáhnete zaslání chybové zprávy Time Exceeded (Čas překročen). Sledujte, jak se změnil průběh komunikace pomocí NetAnimu.
17. Pro jednotlivé uzly si projděte komunikaci v programu Wireshark. Najděte číselné hodnoty pro typ zpráv Time Exceeded, podobně jako u zpráv Destination Unreachable. Zjistěte také ze zobrazeného trasovacího souboru *icmpv6-0-0.pcap* počáteční hodnotu Hop limit (počet skoků).
18. Odpovězte na otázky na konci úlohy.

5.2 Teoretický úvod

5.2.1 Internet Control Message Protocol

ICMP je jedním ze základních protokolů sady internetových protokolů. Úloha tohoto protokolu spočívá v posílaní chybových zpráv například o nedosažitelnosti cíle nebo o nedostupnosti služby. Není určen k výměně uživatelských dat na rozdíl od protokolů UDP a TCP. ICMP zprávy (tab. 5.1) se využívají k signalizaci mimořádných událostí a nestandardních stavů, dále ke diagnostickým účelům. Generuje se při zahození paketu, ale pouze informuje odesílatele a nestará se o nápravu. Ačkoli jsou ICMP zprávy obsaženy v standardních IP paketech, jsou ICMP zprávy obvykle zpracovány jako speciální případ a ne jako normální sub-protokol IP (nejedná se o běžné zapouzdření). [5]

5.2.2 Internet Control Message Protocol version 6

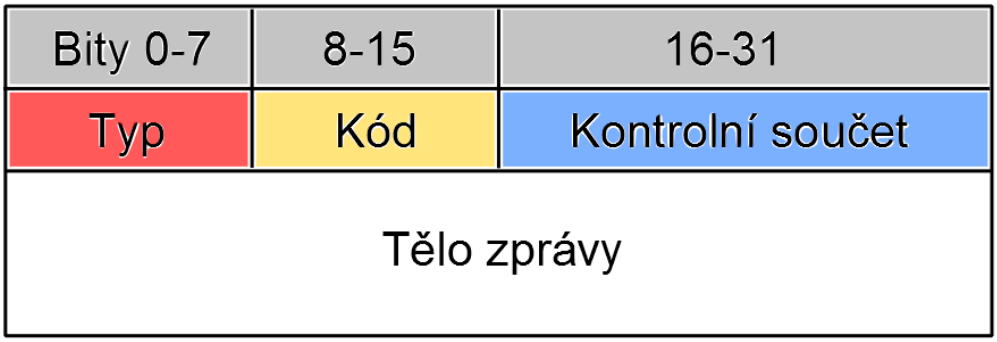
ICMPv6 patří do sady internetových protokolů. Základní úkoly tohoto protokolu jsou oznamování chybových stavů, diagnostika (testování dostupnosti) a v rámci

Typ	Funkce	Popis
0	Echo Reply (Odezva na požadavek)	odpověď na výzvu
3	Destination Unreachable (Cíl nedostupný)	paket byl zahozen
8	Echo Request (Požadavek na odezvu)	výzva k zaslání odpovědi
11	Time Exceeded (Čas překročen)	TTL dosáhlo hodnotu 0

Tab. 5.1: Vybrané typy ICMP zpráv.

IPv6 se využívá ještě k podpoře zprávy multicastových skupin, objevování sousedů, překladu adres a zajištění mobility. ICMPv6 je nezbytnou součástí IPv6, bez níž není funkční. Na rozdíl od protokolu ICMP pro IPv4 má verze pro IPv6 implementovány bezpečnostní opatření. Jedná se o nastavení určitých kvantitativních parametrů, jako povolení průměrného počtu ICMP zpráv za jednotku času nebo maximální podíl zpráv na celkové šířce pásma. Další možnost je nastavení autentizace komunikujících stran v záhlaví. [5]

Základní formát ICMPv6 zprávy (obr. 5.1) je tvořen z polí: typ, kód, kontrolní součet a tělo zprávy. Položka typ udává druh zprávy a kód blíže specifikuje danou zprávu.



Obr. 5.1: Základní formát ICMPv6 zprávy.

5.3 Vypracování

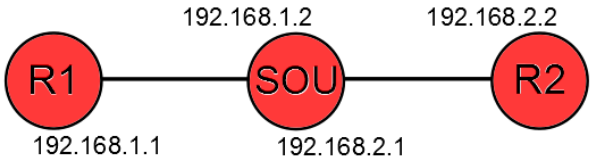
Úloha se skládá ze dvou částí tvořených samostatnými soubory *uloha2a.cc* a *uloha 2b.cc*.

První část se zaměřuje na protokol ICMPv4. Konkrétně se jedná o zprávy Echo Request (Požadavek na odezvu), Echo Reply (Odezva), Destination Unreachable (Nedostupný cíl) a Time Exceeded (Čas překročen). Ve správně fungující topologii (obr. 5.2), která je popsána níže jsou provedeny úpravy pro dosažení chybových stavů. Těmito úpravami jsou změna čísla portu a IPv4 adresy, které vyvolají zaslání zpráv ICMPv4.

Druhá část se soustřeďuje na ICMPv6 zprávy Echo Request (Požadavek na odezvu), Echo Reply (Odezva), Destination Unreachable (Nedostupný cíl) a Time Exceeded (Čas překročen). Ve zprávně fungující topologii (obr. 5.8), jsou provedeny úpravy pro dosažení chybových stavů. Konkrétně se jedná o změnu IPv6 adresy a čísla portu, které vyvolají zaslání daných ICMPv6 zpráv.

Pro správné zadání adresy je důležité si uvědomit, že uzly R1 a R2 mají každý jedno rozhraní a uzel SOU má dvě rozhraní. Každému rozhraní je přidělena jedna MAC adresa. Pro uzel R1 je to 00:00:00:00:00:01 a číslování pokračuje za sebou. Na uzlu SOU se společným rozhraním s uzlem R1 je MAC adresa 00:00:00:00:00:02. Dále je na uzlu SOU se společným rozhraním s uzlem R2 MAC adresa 00:00:00:00:00:03. Poslední použita MAC adresa je 00:00:00:00:00:04 na uzlu R2. Od tohoto faktu se odvíjejí i IPv6 adresy.

5.3.1 První část



Obr. 5.2: Zadaná topologie pro první část.

Kód souboru *uloha2a.cc* (obr. 5.2) začíná přidáním hlavičkových souborů pomocí prohlášení `#include`.

```
#include <fstream>
#include "ns3/core-module.h"
```

```
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/netanim-module.h"
```

Následuje deklarace jmenného pole `ns3`, povolení selektivního zobrazení záznamů pomocí „Icmpv4LOG“ záznamové funkce a hlavní funkce `main`.

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Icmpv4LOG");

int main (int argc, char **argv)  // Deklarace funkce main
{
```

Animace je poskytnuta grafickým animačním nástrojem NetAnim a nutná je definice výstupního XML souboru s názvem „icmpv4“.

```
std::string animFile = "icmpv4.xml";  // Výstupní XML soubor
```

Voláním metody `CreateObject` vytvoříme uzel třídy `Node` a třídou `Ptr` dotvoříme ukazatel na uzel R1. Doplňte zbývající uzly (SOU a R2) pomocí uvedeného příkladu. Tyto uzly vložíme do kontejneru uzlů s názvem `all` v pořadí R1, SOU a R2, pro snadnější manipulaci. **Pořadí uzlů neměňte!**

```
Ptr<Node> R1 = CreateObject<Node> ();  // Vytvoření uzlu

NodeContainer all (R1, SOU, R2);  // Vložení uzlů do kontejneru
```

Uzly propojíme pomocí linek typu point–to–point. Pomocná třída `PointToPoint Helper` nám umožní nakonfigurovat a propojit objekty `PointToPointNetDevice` a `PointToPointChannel`. Následně zadefinujeme hodnoty atributů přenosová rychlost (9 Mbps) a zpoždění (6 ms).

```
PointToPointHelper pointToPoint;  // Vytvoření point-to-point linek
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("9Mbps"))
    ;
pointToPoint.SetChannelAttribute ("Delay", StringValue ("6ms"));
```

Vytvoříme kontejner zařízení nazvaný `dev1` a na uzly R1 a SOU naistalujeme síťová zařízení. Totožným postupem vytvoříme druhý kontejner zařízení `dev2` a síťová zařízení nainstalujeme na uzly SOU a R2. Každý z dvojice uzlů bude mít instalováno point–to–point síťové zařízení (net device) a bude je spojovat point–to–point kanál (channel).

```
// Nainstalování zařízení do uzlů
NetDeviceContainer dev1 = pointToPoint.Install (R1, SOU);
NetDeviceContainer dev2 = pointToPoint.Install (SOU, R2);
```

Sadu internetových protokolů nainstalujeme na vytvořené uzly, zařízení a kanály. Poslouží nám k tomu kontejner uzlů `all`.

```
InternetStackHelper stackIPv4;
// Nainstalování protokolové sady na uzly z kontejneru
stackIPv4.Install (all);
```

Pomocná třída `Ipv4AddressHelper` nám usnadní přidělení IP adres zařízením na vytvořených uzlech. Metodou `SetBase` nastavíme IPv4 adresu sítě a masku. První adresa 192.168.1.1 bude přidělená rozhraní zařízení na uzlu R1 a následující adresa ze stejné sítě 192.168.1.2 bude přidělena rozhraní zařízení SOU.

```
Ipv4AddressHelper address1;
address1.SetBase ("192.168.1.0", "255.255.255.0");
Ipv4AddressHelper address2;
address2.SetBase ("192.168.2.0", "255.255.255.0");
```

Propojení mezi IPv4 adresou a rozhraním zařízení vytvoříme prostřednictvím metody `Assign`.

```
// Propojení IP adresy a rozhraní zařízení
Ipv4InterfaceContainer int1 = address1.Assign (dev1);
Ipv4InterfaceContainer int2 = address2.Assign (dev2);
```

Metodou `GetObject<IPv4>` získáme ukazatel na IPv4 objekt přičleněn uzlu. Tedy získáme ukazatel na IPv4 objekt přičleněn uzlu R1.

```
Ptr<Ipv4> ipv4R1 = R1->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4SOU = SOU->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4R2 = R2->GetObject<Ipv4> ();
```

Povolíme statické směrování na jednotlivých uzlech. Metodou `GetStaticRouting` zajistíme vyhledání statického směrování.

```
// Povolení statického směrování
Ipv4StaticRoutingHelper ipv4RoutingHelper;

Ptr<Ipv4StaticRouting> routingR1 = ipv4RoutingHelper.
    GetStaticRouting (ipv4R1);
Ptr<Ipv4StaticRouting> routingSOU = ipv4RoutingHelper.
    GetStaticRouting (ipv4SOU);
Ptr<Ipv4StaticRouting> routingR2 = ipv4RoutingHelper.
    GetStaticRouting (ipv4R2);
```

Pro vytvoření cesty v globální směrovací tabulce na daném uzlu použijeme metodu `AddNetworkRouteTo`. První parametr definuje IP adresu cílové sítě následovaný maskou dané sítě. Druhá IP adresa určuje další skok (next hop), tedy nejbližší zařízení na trase do cíle. Číslo za IP adresami definuje rozhraní určené k zaslání paketů do cíle.

```
routingR1->AddNetworkRouteTo (Ipv4Address ("192.168.2.0"), Ipv4Mask
    ("255.255.255.0"), Ipv4Address ("192.168.1.2"), 1);
routingR2->AddNetworkRouteTo (Ipv4Address ("192.168.1.0"), Ipv4Mask
    ("255.255.255.0"), Ipv4Address ("192.168.2.1"), 1);
```

Aplikace ping je vytvořena pomocí třídy `V4PingHelper`. Jako první je definována vzdálená adresa (192.168.2.2), na kterou se budou posílat pakety ping aplikace. Specifikovat budeme dva atributy a to interval „Interval“ mezi zasílanými datovými jednotkami a jejich velikost „Size“. Ping aplikaci nainstalujeme na uzel R1 pomocí metody `Install`. Nadefinujeme začátek a konec aplikace, během níž se bude generovat provoz.

```
// Vzdálená adresa
V4PingHelper ping = V4PingHelper (Ipv4Address ("192.168.2.2"));

ping.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
ping.SetAttribute ("Size", UintegerValue (56));

ApplicationContainer appPing = ping.Install (R1);  // Zdroj
appPing.Start (Seconds (1.0));
appPing.Stop (Seconds (5.0));
```

Dalšími aplikacemi, které vytvoříme jsou `UdpEchoServerApplication` a `UdpEchoClientApplication`. Pomocí objektů `UdpEchoServerHelper` a `UdpEchoClientHelper` se nám snadněji nakonfigurují základní objekty. Vytvoříme UDP Echo Server aplikaci na uzlu R2. Objektu `UdpEchoServerHelper` dodáme atribut číslo portu 80, na základě čehož můžeme volit, o jakou aplikaci se jedná. Metoda `Install` objektu `UdpEchoServerHelper` zajistí připojení aplikace na zvolený uzel. Pomocí metod `Start` a `Stop` nadefinujeme začátek a konec generování provozu aplikace.

```
// Definuje číslo portu, na který se dotazuje client (port serveru)
UdpEchoServerHelper echoServer (80);

// Definuje uzel, na kterém je server
ApplicationContainer serverApps = echoServer.Install (R2);
serverApps.Start (Seconds (6.0));  // Start serveru
serverApps.Stop (Seconds (10.0));  // Stop serveru
```

Realizace Echo Client aplikace je podobná serveru. Pro zprávu aplikace `UdpEchoClientApplication` je k dispozici objekt `UdpEchoClientHelper`. Počet atributů se

od serveru liší. První dva definují vzdálenou IP adresu a vzdálené číslo portu. Klientovi tedy nastavíme vzdálenou IP adresu, na kterou bude zasílat dotazy. Konkrétně se jedná o adresu přidělenou uzlu R2, kterou můžeme zadat přímo nebo pomocí metody `GetAddress` z kontejneru rozhraní `int2`. Na tomto uzlu je nastaven server. Nutné je definovat i číslo vzdáleného portu, na který bude klient posílat dotazy. Je to námi určený port serveru. Nastavíme i ostatní atributy. Maximální počet paketů poslaných během simulace „MaxPackets“, doba čekání mezi zasláním dvou paketů „Interval“ a velikost přenášených uživatelských dat (payload) paketu „PacketSize“ udaná v bajtech. Určíme i začátek a konec provozu klientské aplikace.

```
UdpEchoClientHelper echoClient (Ipv4Address ("192.168.2.2"), 80);
//UdpEchoClientHelper echoClient (int2.GetAddress (1), 80);
echoClient.SetAttribute ("MaxPackets", UintegerValue (3));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

// Definuje uzel, na kterém je client
ApplicationContainer clientApps = echoClient.Install (R1);
clientApps.Start (Seconds (6.0));  // Start clienta
clientApps.Stop (Seconds (10.0));  // Stop clienta
```

Výpis směrovacích tabulek všech uzlů si v určitý čas (14 sekunda) zobrazíme metodou `PrintRoutingTableAllAt` do souboru *icmpv4-routing*.

```
// Výpis směrovacích tabulek do souboru
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper
    > ("icmpv4-routing", std::ios::out);
ipv4RoutingHelper.PrintRoutingTableAllAt (Seconds (14),
    routingStream);
```

Detailní informace o posílaných datech získáme metodou `EnablePcapAll`. Výstupem je řada trasovacích souborů ve formátu *pcap*.

```
// Vytvoření trasovacích souborů
pointToPoint.EnablePcapAll ("icmpv4");
```

Pro zobrazení animace použijeme NetAnim. Jedná se o standardní grafický animační nástroj. Samostatný software využívající sledovací soubory, generované v průběhu simulace pro zobrazení topologie a animace toků paketů mezi uzly. Třída `AnimationInterface` je zodpovědná za vytváření sledovacího XML souboru *icmpv4.xml*. Potřebné je nastavit pozici jednotlivých uzlů v animaci. Slouží k tomu metoda `SetConstantPosition`, kde první atribut je ukazatel na uzel a zbylé dva udávají pozici na osách X a Y. Další užitečná metoda je `EnablePacketMetadata` zobrazující meta informace o paketech. Konkrétně se jedná o zobrazení čísel portů pro aplikace využívající protokol UDP. Metoda `UpdateNodeDescription` slouží ke změně názvu

jednotlivých uzlů v NetAnimu. Po zkompilování se v terminálovém okně zobrazí varování, že nebyl použit model mobility a je třeba pro jednotlivé uzly nastavit konstantní polohu. Protože jsme konstantní polohu nastavili, netřeba se znepokojovat daným výpisem.

```
Simulator::Stop (Seconds (15));  // Zastavení simulace

AnimationInterface anim (animFile);
anim.SetConstantPosition (R1, 2, 5);  // Nastavení pozice uzlu
anim.SetConstantPosition (SOU, 8, 5);
anim.SetConstantPosition (R2, 14, 5);
anim.EnablePacketMetadata (true);
anim.UpdateNodeDescription (0, "R1");  // Změna názvu uzlu
anim.UpdateNodeDescription (1, "SOU");
anim.UpdateNodeDescription (2, "R2");
```

Standardní nastavení ns-3 je, že není generován kontrolní součet. Pole pro kontrolní součet je prázdné, čili vyplněné nulami. Pomocí nastavení atributu „Checksum Enabled“ na hodnotu `true`, povolíme výpočet kontrolního součtu.

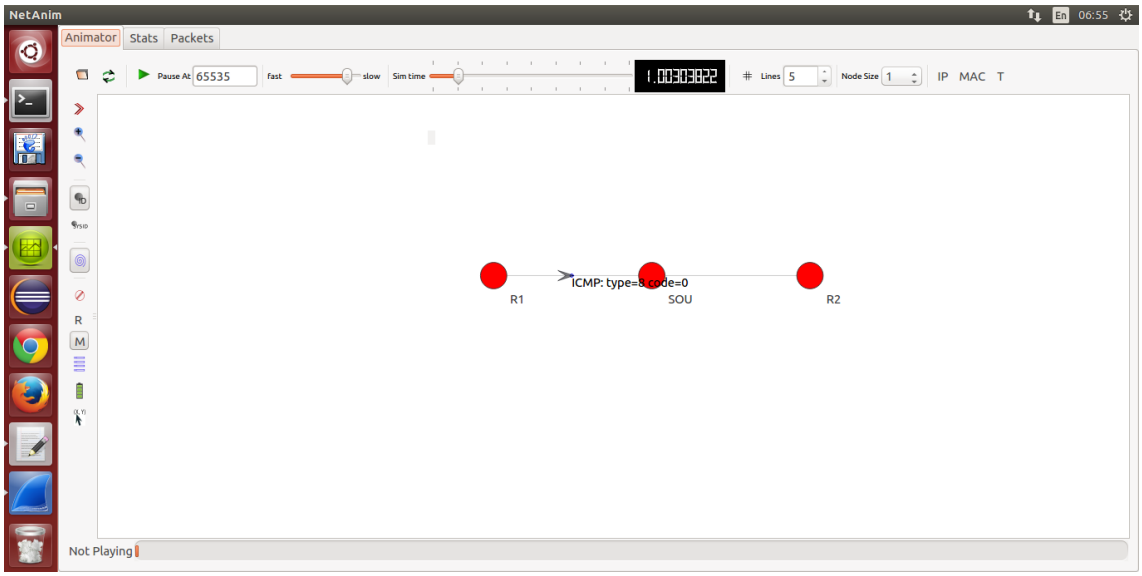
```
// Výpočet kontrolního součtu povolen
GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
```

Spuštění samotné simulace realizujeme pomocí globální funkce `Simulator::Run`. Na začátku simulace naváže klient spojení se serverem. Po dokončení přenosu dat, v našem případě definovaném časovým údajem stop, přejde simulace do stavu nečinnosti. Další plánovaná událost je v tomto okamžiku jen zastavení aplikací. Protože již není nic víc naplánováno, simulace se ukončí. Posledním zbývajícím úkolem je úklid, který má na starosti globální funkce `Simulator ::Destroy`. Vymaže všechny objekty vytvořené během simulace a my nemusíme udržovat přehled o vytvářených objektech.

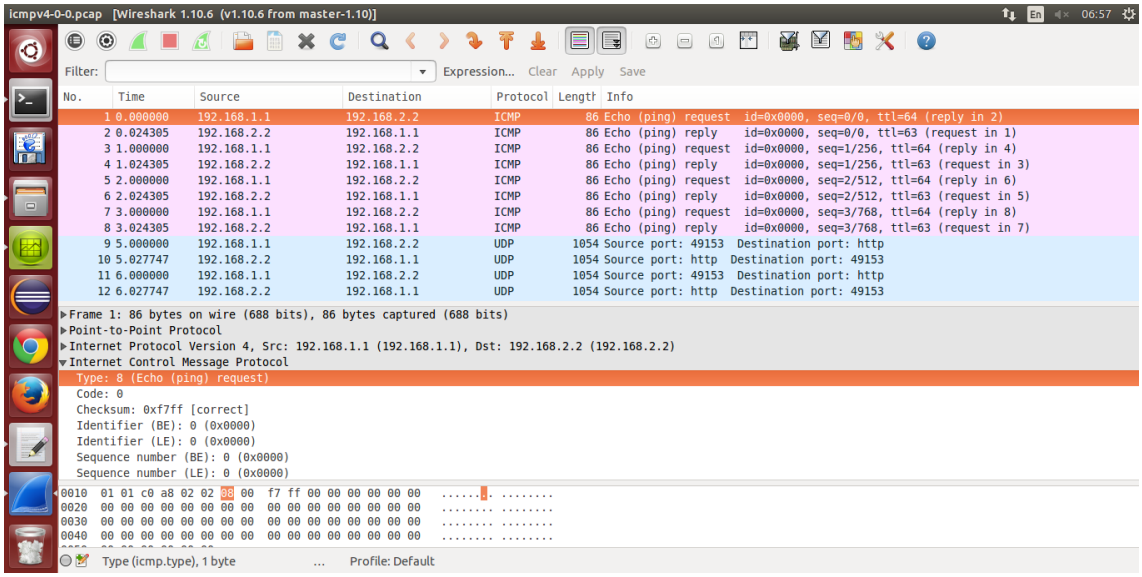
```
Simulator::Run ();  // Běh simulace
Simulator::Destroy ();  // Vymazání objektů
}
```

Kompilace vytvořeného skriptu je jednoduchá. V prvé řadě je třeba změnit adresář, ve kterém se nacházíme. Nesmíme opomenout vložení souboru *uloha2a.cc* do adresáře *scratch* (*/workspace/ns-3.21/scratch*). Následně provedeme kompilaci v terminálovém okně pomocí nástroje Waf (musíme se nacházet v adresáři *ns-3.21*). Po úspěšné kompilaci dostaneme do terminálového okna výpis `'build' finished successfully`. Simulaci spustíme pomocí názvu souboru bez přípony.

```
$ cd wokrspace/ns-3.21
$ ./waf
$ ./waf --run scratch/uloha2a
```



Obr. 5.3: Zobrazení simulace v programu NetAnim.



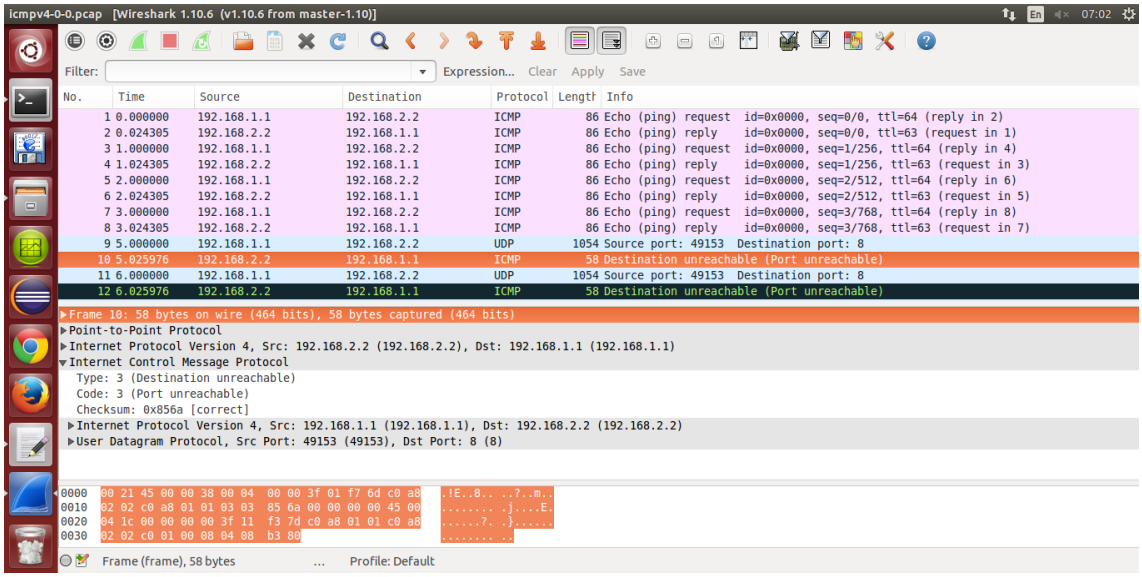
Obr. 5.4: Zobrazení komunikace na uzlu R1.

Vytvořili jsme funkční IPv4 topologii, která poslouží jako základ. Spuštěním souboru *icmpv4.xml* v NetAnimu (obr. 5.3) sledujeme průběh komunikace mezi uzly R1 a R2. Aplikace ping posílá (ping Echo Request a ping Echo Reply) pakety od 1.0 sekundy s pravidelností jedna sekunda. Počátek komunikace ve Wiresharku, soubor *icmpv4-0-0.pcap*, je v čase 0.0, což odpovídá první sekundě ve skriptu. Datové jednotky jsou v animaci označeny ICMP s hodnotou typu a kódu ICMP zprávy. Aplikace UDP Echo Client posílá pakety od 6.0 sekundy (ve Wiresharku 5.0) s pra-

videlností jedna sekunda a počtem paketů nastaveným na tři (třikrát UDP Echo Client z portu 49153 na port 80 a třikrát UDP Echo Server z portu 80 na 49153). Datové jednotky mají v animaci označení UDP s číslem odesílaného a přijímaného portu. Při správném fungování aplikací dojde k zaslání datové jednotky z uzlu R1 na uzel R2 a po doručení zprávy, uzel R2 odpoví (ping a UDP Echo Server). Co pozorujeme při otevření souboru *icmpv4-0-0.pcap* ve Wiresharku (obr. 5.4). V souboru *icmpv4-routing* vidíme předem vytvořené cesty, které jsme využili ke směrování.

Změnou čísla vzdáleného portu u UDP Echo Client aplikace na 8, tedy aby se nerovnalo číslu portu u UDP Echo Server aplikace, dosáhneme zaslání zprávy ICMP Destination Unreachable (Cíl nedostupný). Dojde k zaslání UDP datové jednotky z uzlu R1, avšak odpovědí je ICMP zpráva o nedostupnosti portu (obr. 5.5) z uzlu R2. Situace se bude opakovat pro každý paket UDP aplikace. V NetAnimu pozorujeme, že odpověď není označena, jako UDP s čísly portů, ale jako ICMP Dst. Unreachable (obr. 5.6). U aplikace ping nepozorujeme žádnou změnu.

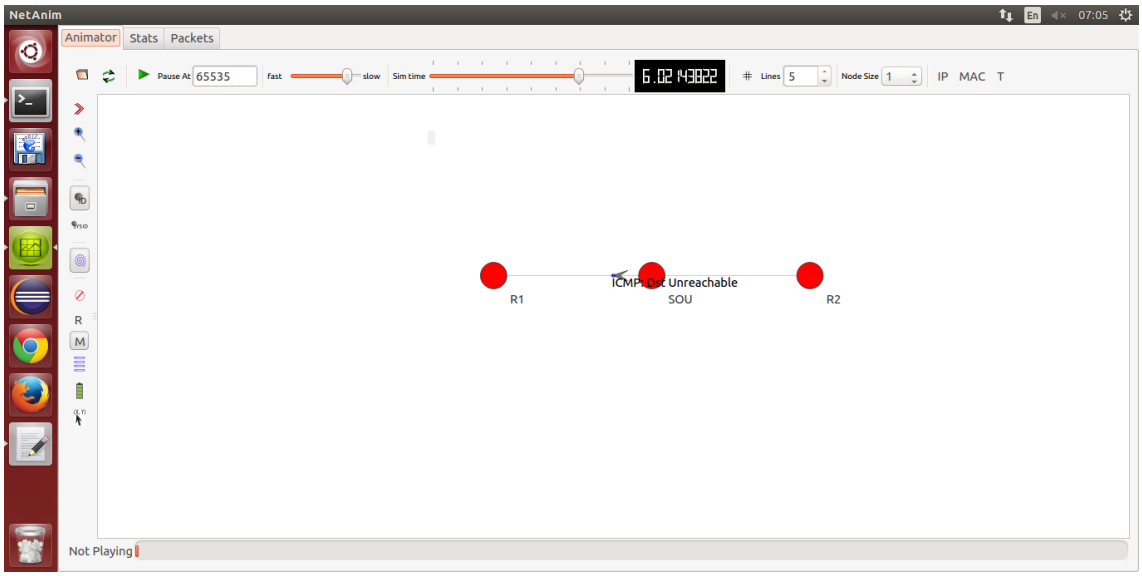
```
UdpEchoClientHelper echoClient (Ipv4Address ("192.168.2.2"), 8);
```



Obr. 5.5: Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server.

Změnou atributu „IpTtl“ v třídě `UdpSocket` pozměníte TTL (Time To Live) na hodnotu např. 6 (defaultně 64), abychom minimalizovali počet přeposílání mezi uzly SOU a R2. Tento řádek kódu umístíme za deklaraci hlavní funkce `main`.

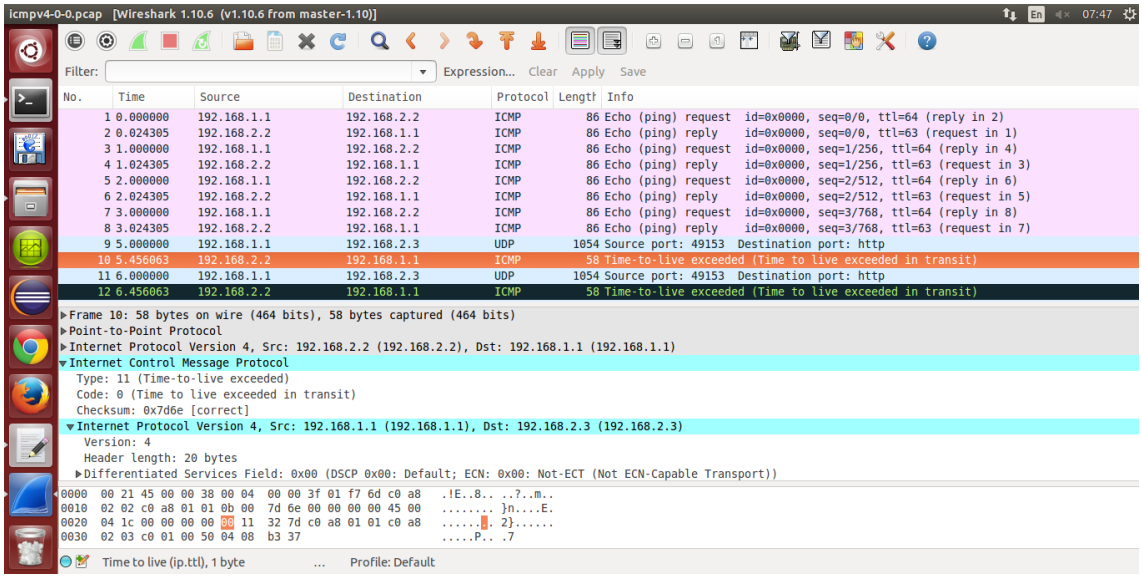
```
Config::SetDefault ("ns3::UdpSocket::IpTtl", UintegerValue (6));
   // Změna hodnoty TTL
```



Obr. 5.6: Zobrazení chybové komunikace aplikace UDP Echo Client a Server.

Úpravou vzdálené adresy UDP Echo Client aplikace na adresu např. 192.168.2.3, která není přidělena žádnému konfigurovanému rozhraní a patří síti 192.168.2.0, dosáhneme zaslání chybové zprávy Time Exceeded (Čas překročen).

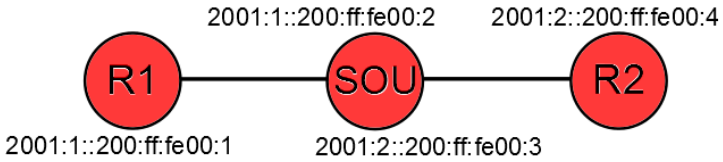
```
UdpEchoClientHelper echoClient (Ipv4Address ("192.168.2.3"), 80);
```



Obr. 5.7: Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server.

Průběh komunikace si zobrazíme z trasovacího souboru ve formátu *pcap* ve Wiresharku (obr. 5.7). Uzel R1 vyšle paket, který je směrován na adresu 192.168.2.3. Díky statické cestě nakonfigurované na R1 bude paket směrován do sítě 192.168.2.0, tedy na R2. To však není cílový adresát a proto bude znovu poslán pomocí výchozí cesty na uzel SOU, odkud je opětovně přeposlán na R2. Situace se bude opakovat až do vyčerpání hop limitu (počtu skoků) = time to live (TTL) doby života paketu, která se dekrementuje při každém průchodu uzlem. Když dosáhne nuly, dojde k zaslání ICMP zprávy Time Exceeded (Čas překročen) z uzlu R2.

5.3.2 Druhá část



Obr. 5.8: Zadaná topologie pro druhou část.

Soubor *uloha2b.cc* (obr. 5.8) začíná prohlášeními `#include` pro přidání hlavičkových souborů.

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"
#include "ns3/netanim-module.h"
```

Dále je deklarované jmenné pole `ns3`, záznamová funkce „Icmpv6LOG“ povolující selektivní zobrazení záznamů a hlavní funkce `main`, jako v předchozí části.

```
using namespace ns3;
```

```
NS_LOG_COMPONENT_DEFINE ("Icmpv6LOG");

int main (int argc, char *argv[])  // Deklarace funkce main
{
```

Použitý je nástroj NetAnim a proto je nutné definovat výstupní XML soubor s názvem „icmpv6“.

```
std::string animFile = "icmpv6.xml";  // Výstupní XML soubor
```

Voláním metody `CreateObject` vytvoříme uzel třídy `Node` a třídou `Ptr` dotvoříme ukazatel na uzel R1. Vytvoříme tři uzly s názvy R1, SOU a R2. Tyto uzly vložíme do kontejneru uzlů s názvem `all` v pořadí R1, SOU a R2, pro snadnější manipulaci. **Pořadí uzlů neměňte!**

```
Ptr<Node> R1 = CreateObject<Node> ();  // Vytvoření uzlu
Ptr<Node> SOU = CreateObject<Node> ();
Ptr<Node> R2 = CreateObject<Node> ();

NodeContainer all (R1, SOU, R2);  // Vložení uzlů do kontejneru
```

Pro propojení uzlů uplatníme point–to–point linky. Třída `PointToPointHelper` nám pomůže nakonfigurovat a propojit objekty `PointToPointNetDevice` a `PointToPointChannel`. Následně zadefinujeme hodnoty atributů přenosová rychlost (9 Mbps) a zpoždění (6 ms).

```
PointToPointHelper pointToPoint;  // Vytvoření point-to-point linek
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("9Mbps"))
    ;
pointToPoint.SetChannelAttribute ("Delay", StringValue ("6ms"));
```

Vytvoříme kontejner zařízení nazvaný `dev1` a naistalujeme síťová zařízení na uzly R1 a SOU. Stejným postupem vytvoříme druhý kontejner zařízení `dev2` a síťová zařízení nainstalujeme na uzly SOU a R2. Každý z dvojice uzlů bude mít instalováno point–to–point síťové zařízení (net device) a bude je spojovat point–to–point kanál (channel).

```
// Nainstalování zařízení do uzlů
NetDeviceContainer dev1 = pointToPoint.Install (R1, SOU);
NetDeviceContainer dev2 = pointToPoint.Install (SOU, R2);
```

Nainstalujeme protokolovou sadu na vytvořené uzly, zařízení a kanály. Poslouží nám k tomu kontejner uzlů `all`.

```
InternetStackHelper stackIPv6;
// Nainstalování protokolové sady na uzly z kontejneru
stackIPv6.Install (all);
```

Pomocí třídy `Ipv6AddressHelper` přidělíme IP adresy rozhraním zařízení. Metodou `SetBase` nastavíme adresu IPv6 sítě a velikost prefixu. Následně spojíme IPv6 adresu a rozhraní zařízení metodou `Assign`. Pomocí metody `SetForwarding` automaticky nastavíme všechna rozhraní uzlů do stejného přeposílacího stavu. Číslo udává index uzlu a `true` specifikuje, že se uzel bude chovat jako směrovač. Poslední řádek (`SetDefaultRouteInAllNodes`) nastaví výchozí cestu (default route) pro všechna zařízení kromě samotného směrovače.

```
Ipv6AddressHelper address1;
Ipv6AddressHelper address2;

address1.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix (64));
Ipv6InterfaceContainer intc1 = address1.Assign (dev1);
intc1.SetForwarding (0, true);  // R1
intc1.SetForwarding (1, true);  // SOU
intc1.SetDefaultRouteInAllNodes (1);  // Výchozí cesta (default
    route) z R1 do SOU

address2.SetBase (Ipv6Address ("2001:2::"), Ipv6Prefix (64));
Ipv6InterfaceContainer intc2 = address2.Assign (dev2);
intc2.SetForwarding (0, true);  // SOU
intc2.SetForwarding (1, true);  // R2
intc2.SetDefaultRouteInAllNodes (0);  // Výchozí cesta (default
    route) z R2 do SOU
```

Pomocná třída `Ipv6StaticRoutingHelper` povolí statické směrování.

```
// Povolení statického směrování
Ipv6StaticRoutingHelper ipv6RoutingHelper;
```

Tvorba ping6 aplikace je pomocí třídy `Ping6Helper`. Lokální adresu nastavíme adresu uzlu R1 a jako vzdálenou adresu nastavíme adresu uzlu R2. Můžeme nastavit přímo adresu nebo vybrat adresu z kontejneru rozhraní pomocí metody `GetAddress`. Dále specifikujeme vlastnosti dané aplikace: maximální počet paketů zaslaných v průběhu simulace „MaxPackets“, doba čekání mezi zasláním dvou paketů „Interval“ a velikost přenášených uživatelských dat (payload) paketu „PacketSize“. Metodou `Install` připojíme aplikaci na zvolený uzel R1. Nadefinujeme začátek a konec aplikace, během níž se bude generovat provoz.

```
Ping6Helper ping6;

// Zdroj R1
ping6.SetLocal (Ipv6Address ("2001:1::200:ff:fe00:1"));
ping6.SetRemote (Ipv6Address ("2001:2::200:ff:fe00:4"));  // Cíl R2
//ping6.SetLocal (intc1.GetAddress (0, 1));  // Zdroj R1
//ping6.SetRemote (intc2.GetAddress (1, 1));  // Cíl R2
```

```
ping6.SetAttribute ("MaxPackets", UintegerValue (4));
ping6.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
ping6.SetAttribute ("PacketSize", UintegerValue (58));

ApplicationContainer appPing = ping6.Install (R1);
appPing.Start (Seconds (1.0));
appPing.Stop (Seconds (5.0));
```

Dalšími aplikacemi, které vytvoříme jsou `UdpEchoServerApplication` a `UdpEchoClientApplication`. Pomocí objektů `UdpEchoServerHelper` a `UdpEchoClientHelper` se nám snadněji nakonfigurují základní objekty. Vytvoříme UDP Echo Server aplikaci na uzlu R2. Objektu `UdpEchoServerHelper` dodáme atribut číslo portu 80, na základě čehož můžeme volit, o jakou aplikaci se jedná. Metoda `Install` objektu `UdpEchoServerHelper` zajistí připojení aplikace na zvolený uzel. Pomocí metod `Start` a `Stop` nadefinujeme začátek a konec generování provozu aplikace.

```
// Definuje číslo portu, na který se dotazuje client (port serveru)
UdpEchoServerHelper echoServer (80);

// Definuje uzel, na kterém je server
ApplicationContainer serverApps = echoServer.Install (R2);
serverApps.Start (Seconds (6.0));  // Start serveru
serverApps.Stop (Seconds (10.0));  // Stop serveru
```

Realizace Echo Client aplikace je podobná serveru. Pro správu aplikace `UdpEchoClientApplication` je k dispozici objekt `UdpEchoClientHelper`. Počet atributů se od serveru liší. První dva definují vzdálenou IP adresu a vzdálené číslo portu. Klientovi tedy nastavíme vzdálenou IP adresu, na kterou bude zasílat dotazy. Konkrétně se jedná o adresu přidělenou uzlu R2, kterou můžeme zadat přímo nebo pomocí metody `GetAddress` z kontejneru rozhraní `intc2`. Při metodě `GetAddress` je první parametr index rozhraní a další je index adresy (typ adresy). Na tomto uzlu je nastaven server. Nutné je definovat i číslo vzdáleného portu, na který bude klient posílat dotazy. Je to námi určený port serveru. Nastavíme i ostatní atributy. Maximální počet paketů poslaných během simulace „MaxPackets“, doba čekání mezi zasláním dvou paketů „Interval“ a velikost přenášených uživatelských dat (payload) paketu „PacketSize“ udaná v bajtech. Určíme i začátek a konec provozu klientské aplikace.

```
UdpEchoClientHelper echoClient (Ipv6Address ("2001:2::200:ff:fe00:4
    "), 80);
//UdpEchoClientHelper echoClient (intc2.GetAddress (1, 1), 80);
echoClient.SetAttribute ("MaxPackets", UintegerValue (3));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
// Definuje uzel, na kterém je client
ApplicationContainer clientApps = echoClient.Install (R1);
clientApps.Start (Seconds (6.0));  // Start clienta
clientApps.Stop (Seconds (10.0));  // Stop clienta
```

Směrovací tabulky pro jednotlivé uzly si ve čtrnácté sekundě zobrazíme metodou `PrintRoutingTableAllAt` do souboru *icmpv6-routing*.

```
// Výpis směrovacích tabulek do souboru
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper
    > ("icmpv6-routing", std::ios::out);
ipv6RoutingHelper.PrintRoutingTableAllAt (Seconds (14),
    routingStream);
```

Sledování poskytující detailní informace o posílaných datech nastavíme metodou `EnablePcapAll`. Výstupem je řada trasovacích souborů ve formátu *pcap*.

```
// Vytvoření trasovacích souborů
pointToPoint.EnablePcapAll ("icmpv6");
```

Pro zobrazení animace použijeme NetAnim. Jedná se o standardní grafický animační nástroj. Samostatný software využívající sledovací soubory, generované v průběhu simulace pro zobrazení topologie a animace toků paketů mezi uzly. Třída `AnimationInterface` je zodpovědná za vytváření sledovacího XML souboru *icmpv6.xml*. Potřebné je nastavit pozici jednotlivých uzlů v animaci. Slouží k tomu metoda `SetConstantPosition`, kde první atribut je ukazatel na uzel a zbylé dva udávají pozici na osách X a Y. Další užitečná metoda je `EnablePacketMetadata` zobrazující meta informace o paketech. Konkrétně se jedná o zobrazení čísel portů pro aplikace využívající protokol UDP. Metoda `UpdateNodeDescription` slouží ke změně názvu jednotlivých uzlů v NetAnimu. Po zkompilování se v terminálovém okně zobrazí varování, že nebyl použit model mobility a je třeba pro jednotlivé uzly nastavit konstantní polohu. Protože jsme konstantní polohu nastavili, netřeba se znepokojovat daným výpisem.

```
Simulator::Stop (Seconds (15));  // Zastavení simulace

AnimationInterface anim (animFile);
anim.SetConstantPosition (R1, 2, 5);  // Nastavení pozice uzlu
anim.SetConstantPosition (SOU, 8, 5);
anim.SetConstantPosition (R2, 14, 5);
anim.EnablePacketMetadata (true);
anim.UpdateNodeDescription (0, "R1");  // Změna názvu uzlu
anim.UpdateNodeDescription (1, "SOU");
anim.UpdateNodeDescription (2, "R2");
```

Standardní nastavení ns-3 je, že není generován kontrolní součet. Pole pro kontrolní součet je prázdné, čili vyplněné nulami. Pro IPv6 je toto pole povinné, není možné jej vypnout. Proto při zobrazení zachycené komunikace v programu Wireshark se kontrolní součet zobrazuje jako chybný. Pomocí nastavení atributu „Check sumEnabled“ na hodnotu `true`, povolíme výpočet kontrolního součtu.

```
// Výpočet kontrolního součtu povolen
GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
```

Spuštění samotné simulace realizujeme pomocí globální funkce `Simulator::Run`. Na začátku simulace naváže klient spojení se serverem. Po dokončení přenosu dat, v našem případě definovaném časovým údajem stop, přejde simulace do stavu nečinnosti. Další plánovaná událost je v tomto okamžiku jen zastavení aplikací. Protože již není nic víc naplánováno, simulace se ukončí. Posledním zbývajícím úkolem je úklid, který má na starosti globální funkce `Simulator::Destroy`. Vymaže všechny objekty vytvořené během simulace a my nemusíme udržovat přehled o vytvářených objektech.

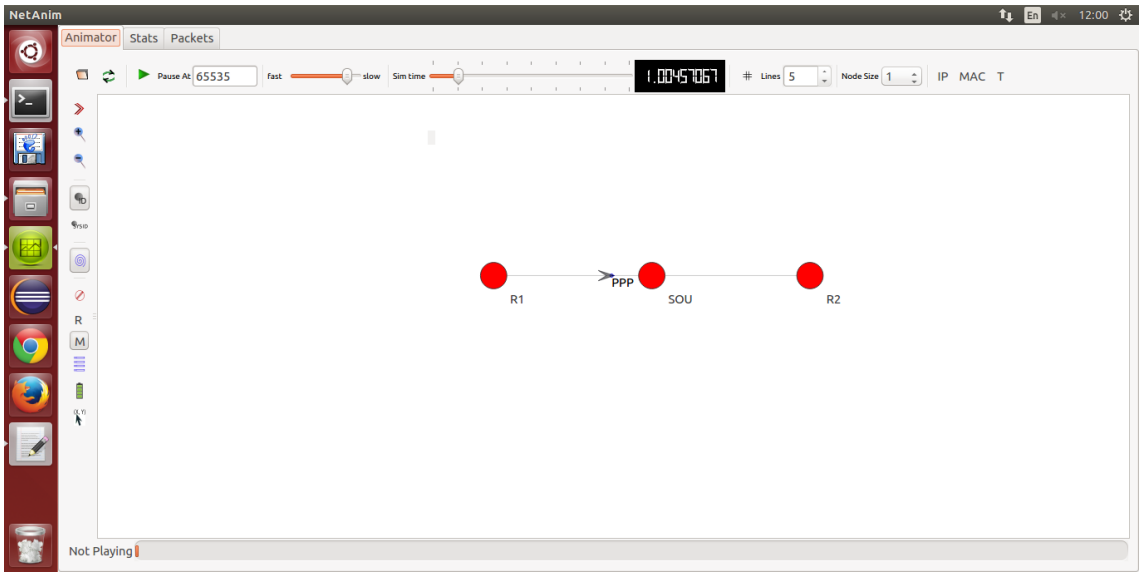
```
Simulator::Run ();  // Běh simulace
Simulator::Destroy ();  // Vymazání objektů
}
```

Kompilace vytvořeného skriptu je jednoduchá. V prvé řadě je třeba změnit adresář, ve kterém se nacházíme. Nesmíme opomenout vložení souboru *uloha2b.cc* do adresáře *scratch* (*/workspace/ns-3.21/scratch*). Následně provedeme kompilaci v terminálovém okně pomocí nástroje Waf (musíme se nacházet v adresáři *ns-3.21*). Po úspěšné kompilaci dostaneme do terminálového okna výpis `'build' finished successfully`. Simulaci spustíme pomocí názvu souboru bez přípony.

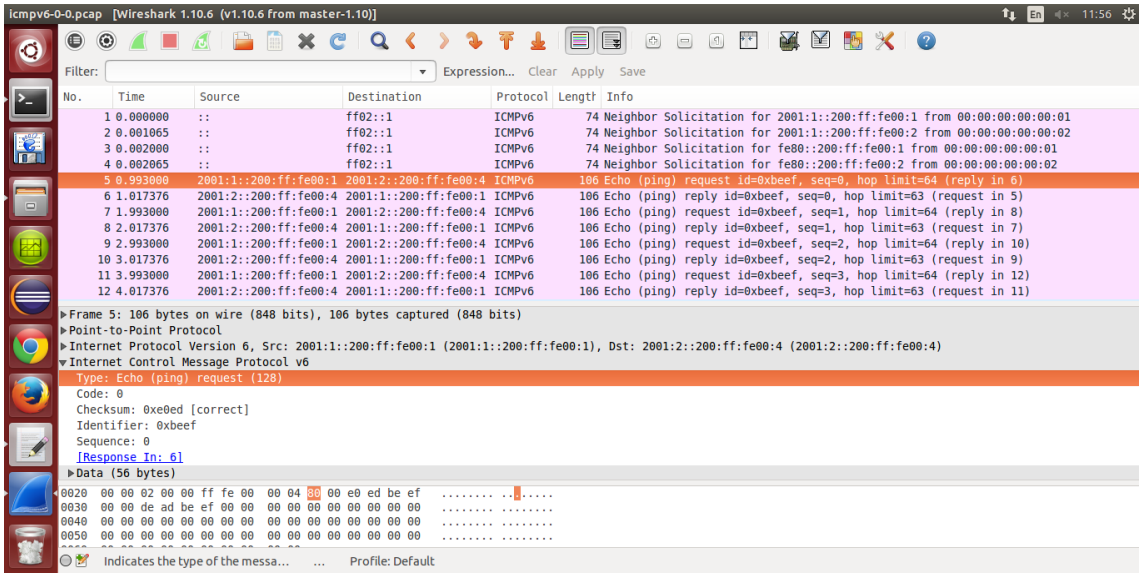
```
$ cd wokrspace/ns-3.21
$ ./waf
$ ./waf --run scratch/uloha2b
```

V tomto okamžiku máme funkční IPv6 topologii, která poslouží jako základ. Při spuštění *icmpv6.xml* souboru v NetAnimu (obr. 5.9) je třeba pár vteřin počkat dokud dojde k výměně Neighbor Solicitation zpráv. Samotné aplikace si začnou vyměňovat pakety až následně. Aplikace ping6 posílá pakety od 1.0 sekundy s pravidelností jedna sekunda a počet paketů je nastavených na čtyři (čtyřikrát ping Echo Request a čtyřikrát odpověď ping Echo Reply). V animaci mají datové jednotky označení PPP, protože pro zapouzdření byl použit protokol PPP (Point–to–Point Protocol) linkové vrstvy. UDP Echo Client aplikace posílá pakety od 6.0 sekundy s pravidelností jedna sekunda a počet paketů je nastavených na tři (třikrát UDP Echo Client z portu 49153 na port 80 a třikrát UDP Echo Server z portu 80 na 49153). Datové jednotky mají v animaci označení UDP s číslem odesílaného a přijímaného portu.

Při správném fungování aplikací dojde k zaslání datové jednotky z uzlu R1 na uzel R2 a po doručení zprávy, uzel R2 odpoví (ping6 a UDP Echo Server). Co pozorujeme při otevření jednoho ze souborů s příponou *pcap* ve Wiresharku (obr. 5.10). V souboru *icmpv6-routing* vidíme výchozí cesty, které jsme nastavili a využívají se ke směrování.



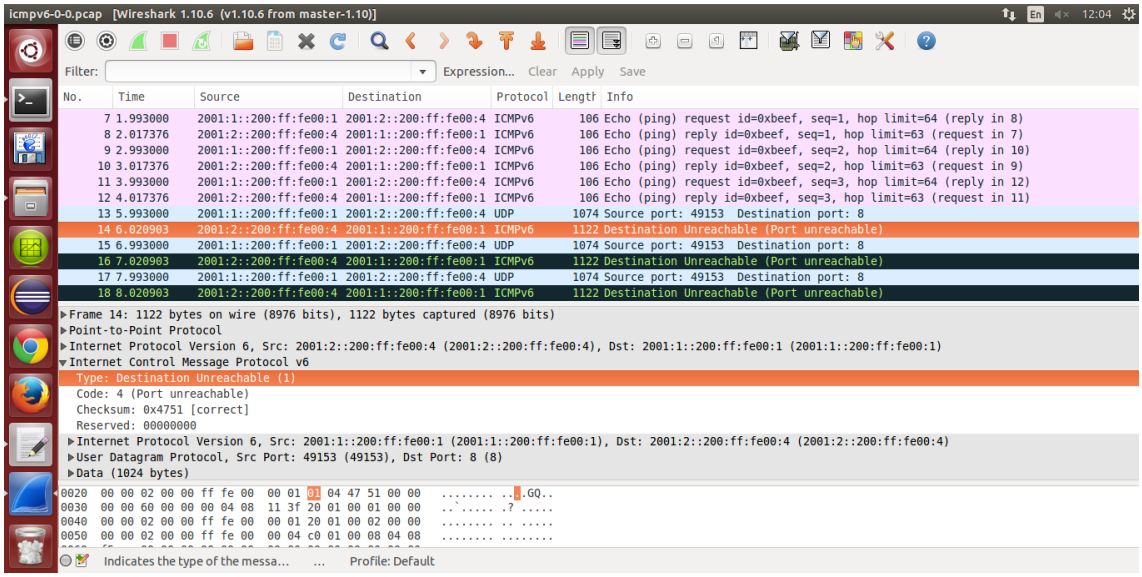
Obr. 5.9: Zobrazení simulace v programu NetAnim.



Obr. 5.10: Zobrazení komunikace na uzlu R1.

Změnou čísla vzdáleného portu u UDP Echo Client aplikace na 8, tedy aby se nerovnalo číslu portu u UDP Echo Server aplikace, dosáhneme zaslání zprávy ICMPv6 Destination Unreachable (Cíl nedostupný). Dojde k zaslání UDP datové jednotky z uzlu R1, avšak odpovědí je ICMPv6 zpráva o nedostupnosti portu (obr. 5.11) z uzlu R2. Situace se bude opakovat pro každý paket aplikace využívající protokol UDP. V NetAnimu pozorujeme, že odpověď není označena, jako UDP s čísly portů, ale jako PPP. U aplikace ping6 nepozorujeme změnu.

```
UdpEchoClientHelper echoClient (Ipv6Address ("2001:2::200:ff:fe00:4
    "), 8);
```



Obr. 5.11: Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server.

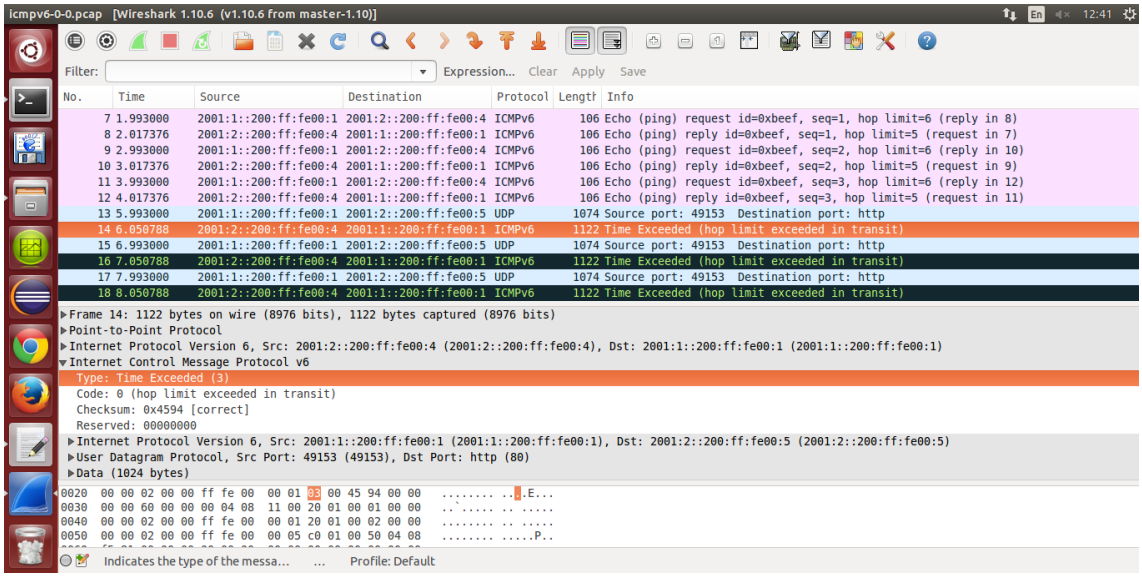
Změnou hodnoty atributu „DefaultTtl“ v třídě `Ipv6L3Protocol` pozměníme TTL na hodnotu např. 6 (defaultně 64), abychom minimalizovali počet přeposílání mezi uzly SOU a R2. Tento řádek kódu umístíme za hlavní funkcí `main`.

```
Config::SetDefault ("ns3::Ipv6L3Protocol::DefaultTtl",
    UintegerValue (6));  // Změna hodnoty TTL
```

Úpravou vzdálené adresy UDP Echo Client aplikace na adresu např. 2001:2::200:ff:fe00:5, která neodpovídá žádné z použitých adres, dosáhneme zaslání chybové zprávy Time Exceeded (Čas překročen). Průběh komunikace si zobrazíme z trasovacího souboru ve formátu *pcap* ve Wiresharku (obr. 5.12). Uzel R1 vyšle paket, který je směrován na adresu 2001:2::200:ff:fe00:5. Díky výchozím cestám nakonfigurovaným na R1 a R2 bude paket směrován do sítě 2001:2::, tedy na R2. To však není

cílový adresát a proto bude znovu poslán na uzel SOU, odkud je opětovně přeposlán na R2. Situace se bude opakovat až do vyčerpání hop limitu (počtu skoků) = time to live (TTL) doby života paketu, která se dekrementuje při každém průchodu uzlem. Když dosáhne nuly, dojde k zaslání ICMPv6 zprávy Time Exceeded (Čas překročen) z uzlu R2.

```
UdpEchoClientHelper echoClient (Ipv6Address ("2001:2::200:ff:fe00:5
    "), 80);
```



Obr. 5.12: Zobrazení chybové komunikace na uzlu R1 pro aplikace UDP Echo Client a Server.

5.4 Kontrolní otázky

1. Jaký je hlavní úkol ICMP protokolu?
2. Uveďte funkce jednotlivých ICMP zpráv, se kterými jste se seznámili v této laboratorní úloze.
3. Která pole z hlavičky protokolů IPv4, IPv6 využívají zprávy ICMPv4, ICMPv6 a jaké jsou hodnoty těchto polí?

6 ÚLOHA 3: PROTOKOL ICMPV6

6.1 Zadání

1. Prostudujte si teoretický úvod.
2. Vytvořte pomocí vypracování soubor *uloha3a.cc*.
3. V směrovacích tabulkách (soubor *icmpv6a-routing*) uzlů SOU, R1 a R2 vidíme výchozí cesty (default routes) a statické cesty. O které cesty se jedná? Na základě čeho jich rozeznáte? Proč a jaká cesta, přibyla ve směrovací tabulce uzlu SOU (Node 0)?
4. Otevřete si v programu Wireshark soubor *icmpv6a-0-0.pcap* a zobrazte, jak probíhala komunikace. Soustřeďte se na zprávu Redirect (Přesměrování) zaslanou z uzlu R1 na SOU a jaký mělo vliv zaslání této zprávy na komunikaci mezi uzly SOU a host2 (ping). Ve zprávě Redirect ve Wiresharku v záložce ICMP v6 v poli Target Address a Destination Address si ověřte, že Target Address udává adresu dalšího skoku (lokální linkovou adresu uzlu R2) a Destination Address udává adresu cíle (globální IPv6 adresu uzlu host2).
5. Ověřte si v programu Wireshark pro jednotlivé typy zpráv Echo Request (Požadavek na odezvu), Echo Reply (Odezva na požadavek) a Redirect (Přesměrování) jejich číselné hodnoty. Naleznete je v záložce označené Internet Control Message Protocol v poli Type pro jednotlivé datové jednotky.
6. Podle pokynů ve vypracování vytvořte soubor *uloha3b.cc*.
7. V programu Wireshark si otevřete soubor *icmpv6b-0-0.pcap* a pro zprávy Neighbor Solicitation (Výzva sousedovi) ověřte, že v záložce Internet Control Message Protocol v6 v poli Link-layer address se nachází fyzická adresa uzlu odesílatele. Dále si zobrazte, že pro zprávu Neighbor Solicitation se v záložce Internet Control Message Protocol v6 v poli Target Address nachází IPv6 adresa uzlu, které unikátnost ověřujeme.
8. Dále zjistěte ze souboru *icmpv6b-0-0.pcap* ve zprávě Router Advertisement (Ohlášení směrovače) v záložce Internet Control Message Protocol v6 v poli Router lifetime, dobu života prefixu a v poli ICMPv6 Option prefix a jeho délku.
9. Prohlédněte si průběh komunikace a výměnu zpráv Router Advertisement a Router Solicitation (Výzva směrovači). Jak se liší komunikace mezi soubory *icmpv6b-2-0.pcap* (uzel R2) a *icmpv6b-0-0.pcap*?
10. Jaký je typ adresy zdroje a jaký je typ adresy cíle při výměně zpráv Router Advertisement a Router Solicitation (od 1.0 s)?
11. Ověřte si pro jednotlivé typy zpráv RA (Router Advertisement) a RS (Router

Solicitation) jejich číselné hodnoty. Postupujte jako u ostatních ICMPv6 zpráv.

12. Odpovězte na otázky na konci úlohy.

6.2 Teoretický úvod

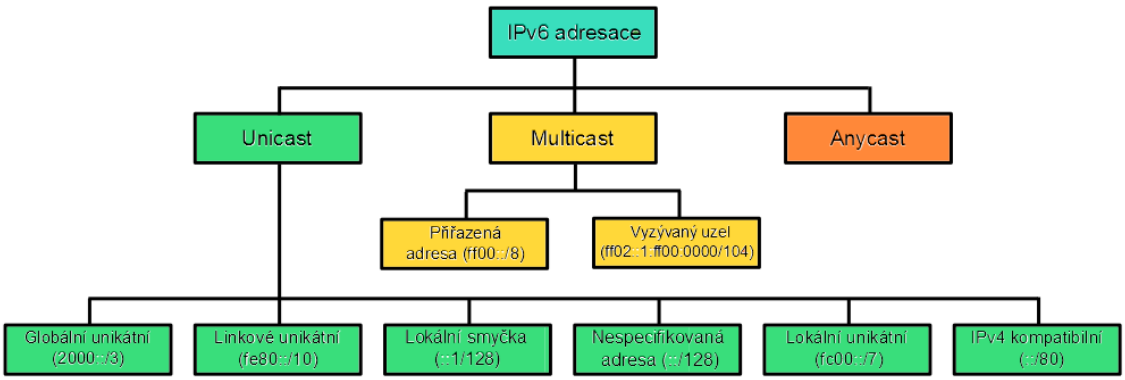
IPv6 (Internet Protocol version 6) používá 128-bitové adresy z čehož vyplývá, že adresní prostor má rozsah 2^{128}. Adresy jsou reprezentovány osmi skupinami čtyř hexadecimálních čísel oddělených dvojtečkou.

2000:0000:0000:0000:1000:8a2e:0370:7334

Každá skupina představuje 16 bitů. Běžné je použití zkráceného zápisu pomocí znaků „::“, v případě že adresa obsahuje souvislý blok nul. Možné je tento zápis použít pouze jednou v jedné adrese.

2000::1000:8a2e:0370:7334

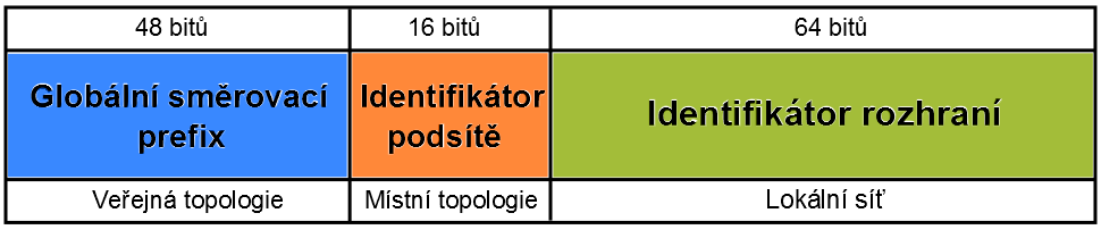
Jako u IPv4 i u IPv6 je adresa tvořena z prefixu definujícího adresu sítě (podsítě) a identifikátoru koncových stanic v rámci lokální sítě (adresa stanice). V rámci adresního rozsahu IPv6 jsou definovány speciální typy adres (obr.6.1).



Obr. 6.1: Základní typy IPv6 adres.

6.2.1 Globální adresy

Ve své podstatě zastupují IPv4 veřejné adresy. Slouží k identifikaci určitého síťového rozhraní v celém Internetu. Přidělování těchto adres má na starost organizace IANA



Obr. 6.2: Struktura globální individuální adresy.

(Internet Assigned Numbers Authority). Globální individuální adresy mají pevnou strukturu (obr. 6.2).

Globální směrovací prefix odpovídá adrese sítě v IPv4. Počet prefixů je 248.

Identifikátor podsítě je určen k rozlišení jednotlivých podsítí v rámci celé sítě. Každá síť může mít 216 podsítí, přičemž rozdělení na podsítě je regulováno organizací, jejíž síť patří.

Identifikátor rozhraní slouží k odlišení koncových stanic v rámci lokální sítě. Celkový počet stanic v koncové síti je 264. Stanovení hodnoty této části adresy je vždy závislé až na koncovém zákazníkovi. Existují 2 varianty:

- Standard IEEE EUI-64 – umožňuje použití identifikátoru na rozhraní (fyzické adresy) doplněné na příslušnou délku jako identifikátor rozhraní z adresy. Identifikátor rozhraní (64 bitů) = MAC adresa rozhraní (48 bitů) + konstanta fffe (16 bitů).
- Náhodně generovaná adresa – kde je identifikátor rozhraní generován náhodně (RFC 4941) v koncové stanici. Adresa má omezenou platnost a po určitém časovém intervalu se bude generovat nová adresa.

6.2.2 Lokální linkové adresy

Jsou adresy (fe80::/10) s lokální platností v rámci dané linky. Nastaveny jsou u každého aktivního rozhraní. Datové jednotky s danou adresou neprojdou přes žádný směrovač. Tato adresa je vytvořena stanicí samostatně a automaticky. Z toho vyplývá, že je vždy k dispozici. Adresy slouží k lokální komunikaci mezi stanicemi, k automatické konfiguraci pomocí protokolu DHCPv6.

6.2.3 Lokální smyčka

Umožňuje počítači komunikovat sám se sebou, čili pakety s touto adresou neopustí počítač. Na rozdíl od IPv4 se jedná o jednu adresu a ne celý adresný rozsah.

6.2.4 Nespecifikovaná adresa

Je využita pokud nebyla přiřazena platná adresa.

6.2.5 Lokální unikátní adresy

Tyto adresy (fc::/7) jsou převážně lokálně generované s prefixem fd::/8. Za prefixem následuje globální identifikátor, což je 40 bitové náhodně vygenerované číslo. Následuje 16 bitů adresy podsítě a adresa je zakončena 64 bitovým identifikátorem konkrétního rozhraní. Platnost adres je lokální a nepředpokládá se směrování internetem. Příklad použití je tam, kde je několik lokálních sítí v různých lokalitách, které se tváří jako jedna lokální síť. Jedná se o obdobu privátních adres z rozsahu IPv4.

6.2.6 Skupinové adresy

Určení těchto adres (ff00::/8) je pro distribuci multimediálního obsahu v reálném čase, jako je obraz a zvuk. Struktura adresy je tvořena z částí. Prvních 8 bitů jsou jedničky, tedy ff hexadecimálně. Následují čtyři bity volby, dále doplněny čtyřmi bity definujícími dosah platnosti adresy a zbylých 112 bitů je určených pro adresu skupiny.

Členění IPv6 adresního rozsahu z hlediska využití prvních dvou bajtů je zobrazeno v tabulce (tab. 6.1).

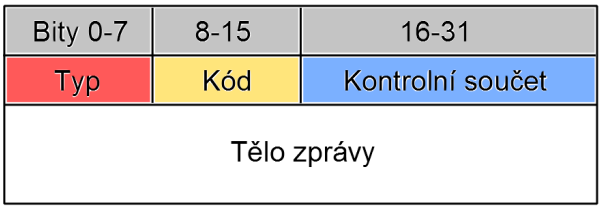
Adresy od do (první dva bajty)	**Význam**
0000 až 00ff	Nespecifikované, lokální smyčka, IPv4 kompatibilní adresy
0100 až 01ff	Nepřiřazeno
0200 až 03ff	Speciální adresy
0400 až 1fff	Nepřiřazeno
2000 až 3fff	Agregovatelné globální unicastové adresy
4000 až fbff	Nepřiřazeno
fc00 až fdff	(Individuální) lokální unikátní adresy
fe00 až fe7f	Nepřiřazeno
fe80 až febf	(Individuální) lokální linkové adresy
fec0 až feff	Site local adresy (zrušeny)
ff00 až ffff	Skupinové (multicast) adresy

Tab. 6.1: Členění adresního prostoru IPv6.

6.2.7 Internet Control Message Protocol version 6

ICMPv6 patří do sady internetových protokolů. Základní úkoly tohoto protokolu jsou oznamování chybových stavů, diagnostika (testování dostupnosti) a v rámci IPv6 se využívá ještě k podpoře zprávy multicastových skupin, objevování sousedů, překladu adres a zajištění mobility. ICMPv6 je nezbytnou součástí IPv6, bez níž není funkční. Na rozdíl od protokolu ICMP pro IPv4 má verze pro IPv6 implementovány bezpečnostní opatření. Jedná se o nastavení určitých kvantitativních parametrů, jako povolení průměrného počtu ICMP zpráv za jednotku času nebo maximální podíl zpráv na celkové šířce pásma. Další možnost je nastavení autentizace komunikujících stran v záhlaví. [5]

Základní formát ICMPv6 zprávy (obr. 6.3) je tvořen z polí: typ, kód, kontrolní součet a tělo zprávy. Položka typ udává druh zprávy a kód blíže specifikuje danou zprávu.



Obr. 6.3: Základní formát ICMPv6 zprávy.

6.2.8 Neighbour Discovery Protocol

NDP, čili protokol pro objevování sousedů. Pracuje na lokální úrovni. Jedná se o protokol linkové vrstvy a je zodpovědný za nalezení logických adres na základě znalosti fyzické adresy (obdobně jako ARP protokol v sítích IPv4). Dále je určen k autokonfiguraci uzlů, objevování uzlů na lince, zjišťování fyzických adres sousedních uzlů, odhalování duplicity adres, přesměrování, hledání směrovačů, zjišťování prefixů adres a udržování informací o dostupnosti sousedních uzlů. Ve všech případech jsou k přenosu využívány zprávy ICMPv6. Hledání sousedů funguje podobně jako u protokolu ARP. Zprávy, které se při tomto procesu využívají, jsou Neighbor Solicitation (Výzva sousedovi) a Neighbor Advertisement (Ohlášení souseda). Neighbor Solicitation je posílána, když chce uzel zjistit fyzickou adresu sousedního uzlu na lokálním spoji. Adresa zdrojového uzlu při zprávě Neighbor Solicitation je IPv6 adresa odesílatele a cílová adresa je multicastové adresa korespondující IPv6 adrese uzlu příjemce. MAC (fyzická) adresa zdrojového uzlu je zahrnuta ve zprávě. Po přijetí

zprávy Neighbor Solicitation, cílový uzel odpoví zasláním zprávy Neighbor Advertisement. Jako zdrojová adresa je IPv6 adresa uzlu odesílatele a cílová adresa je IPv6 adresa uzlu příjemce. V datové části zprávy Neighbor Advertisement je zahrnuta fyzická adresa uzlu odesílatele. Neighbor Solicitation zprávy jsou využívány k ověření dostupnosti souseda po tom, co je identifikována fyzická adresa souseda, nebo při změně fyzické adresy uzlu. Zprávy Neighbor Solicitation jsou také používány při procesu bezestavové autokonfiguraci pro ověření jedinečnosti unicatové IPv6 adresy, předtím než jsou přiděleny rozhraní. Ověření duplicity adresy je provedeno na nové lokální linkové IPv6 adrese, předtím než je adresa přidělena rozhraní. Nová adresa zůstává v nezávazném stavu, pokud je ověřována duplicita adresy. Uzel posílá Neighbor Solicitation zprávu s nespecifikovanou zdrojovou adresou a nezávaznou lokální linkovou adresou v těle zprávy. [5]

Automatická konfigurace adres podporována v IPv6 se označuje jako bezestavová. Spočívá na principu opakovaného vysílání potřebných informací směrovačem. Vysílání je realizované za pomoci zpráv ICMPv6 Router Advertisement (Ohlášení směrovače). Stanice může o zaslání těchto informací požádat pomocí zprávy ICMPv6 Router Solicitation (Výzva směrovači). Na základě získaných informací si stanice sama určí svou globální IPv6 adresu. Pro správné fungování bezestavové automatické konfiguraci musí být vysílána délka prefixu ve zprávě Router Advertisement 64 bitů. Zpráva Router Advertisement běžně obsahuje: jeden nebo více IPv6 prefixů používaných k autokonfiguraci IPv6 adres uzlů, informace o době života prefixu, sadu značek označujících typ automatické konfigurace, informace o výchozím zařízení, další informace pro uživatele. Cílová adresa ve zprávě Router Solicitation je multicastová adresa všech směrovačů (ff02::2) a cílová adresa ve zprávě Router Advertisement je multicastová adresa všech uzlů (ff02::01). [7]

6.2.9 Address Resolution Protocol

ARP řeší transformaci síťových adres na adresy linkové vrstvy. Konkrétně se jedná o nalezení odpovídající fyzické (MAC) adresy k IP adrese. Když posíláme datagram na uzel v lokální síti a známe jen IP adresu příjemce. Pomocí ARP protokolu získáme fyzickou adresu daného příjemce. Informace o adresách se ukládají do tabulky, kde jsou IP adresám přiřazeny odpovídající fyzické adresy. Životnost záznamů v tabulce je omezena jen na několik minut a následně jsou vymazány z důvodu neaktuálnosti. V opačném případě by tabulka vyžadovala velkou paměťovou kapacitu a vyhledání konkrétního záznamu by bylo časově náročné. [5]

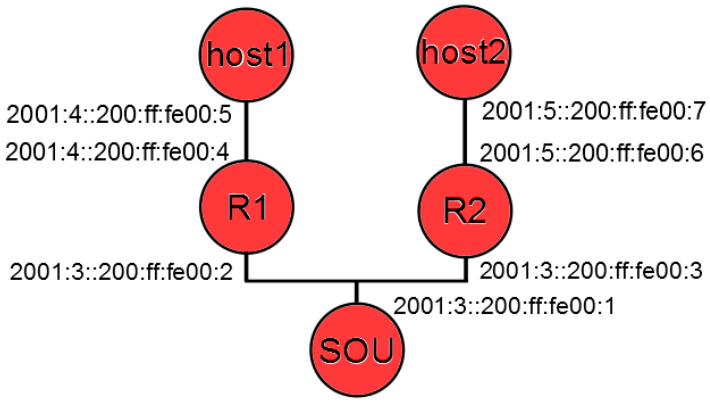
6.3 Vypracování

Úloha se skládá ze dvou částí tvořených samostatnými soubory *uloha3a.cc* a *uloha 3b.cc*.

První část se zaměřuje na zpávu ICMPv6 Redirect (Přesměrování). Podstata úkolu je, že zdroj SOU, vyšle Echo Request (žádost) na zařízení označené jako host2, avšak pro poslání se použije default route (výchozí cesta) směřující na uzel R1. R1 obdrží zprávu Echo Request od zdroje SOU a přepošle ji na uzel R2 statickou cestou. Uzel R1 pošle ICMPv6 Redirect zprávu na zdroj SOU s cílem host2 a směrováním přes uzel R2. Další Echo Request z uzlu SOU na host2 bude přímo poslán přes R2.

Druhá část se zaměřuje na posílání zpráv Router Advertisement. Uzel SOU vysílá periodicky Router Advertisement zprávy na zařízení R1, ovšem neposílá tyto zprávy na zařízení R2. Proto musí R2 zaslat zprávu Router Solicitation, čímž si vyžádá zaslání Router Advertisement.

6.3.1 První část



Obr. 6.4: Zadaná topologie pro první část.

Kód souboru *uloha3a.cc* (obr. 6.4) začíná prohlášeními `#include`, pomocí kterých se přidávají hlavičkové soubory.

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"
```

Následuje deklarace jmenného pole `ns3`, záznamové funkce povolující selektivní zobrazení záznamů a deklarování hlavní funkce `main`.

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Icmpv6aLOG");

int main (int argc, char **argv)  // Deklarace funkce main
{
```

Voláním metody `CreateObject` vytvoříme uzel třídy `Node` a třídou `Ptr` dotvoříme ukazatel na uzel R1. Doplňte zbývající uzly (R1, R2, host1, host2) pomocí uvedeného příkladu.

```
Ptr<Node> SOU = CreateObject<Node> ();  // Vytvoření uzlu
```

Pro lepší manipulaci s uzly je po skupinách vložíme do daných kontejnerů uzlů.

```
// Vložení uzlů do kontejneru
NodeContainer all (SOU, R1, R2, host1, host2);
NodeContainer ncm (SOU, R1, R2);
NodeContainer nc1 (R1, host1);
NodeContainer nc2 (R2, host2);
```

Sestavení topologie pokračuje propojením uzlů pomocí linek typu csma. Pro tvorbu a propojení csma zařízení a kanálů nám poslouží pomocná třída `CsmaHelper` obdobně jako u point–to–point linky. Na rozdíl od point–to–point linky má csma linka definovanou přenosovou rychlost „DataRate“ kanálovým atributem a nikoliv atributem zařízení. Protože v sítích csma nelze míchat různá zařízení na daném kanálu. Definované jsou hodnoty atributů přenosová rychlost „DataRate“, zpoždění „Delay“ a maximální velikost přenosové jednotky „Mtu“.

```
CsmaHelper csma;  // Vytvoření csma linek
csma.SetChannelAttribute ("DataRate", DataRateValue (7500000));
csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (4)));
csma.SetDeviceAttribute ("Mtu", UintegerValue (1480));
```

Pro udržování seznamu zařízení vytvořených pomocí třídy `CsmaHelper` vytvoříme kontejner zařízení `NetDeviceContainer`. Následně zavoláme metodu `Install` pro nainstalování zařízení do uzlů z daného kontejneru.

```
// Nainstalování zařízení do uzlů z daného kontejneru
NetDeviceContainer devm = csma.Install (ncm);
NetDeviceContainer dev1 = csma.Install (nc1);
NetDeviceContainer dev2 = csma.Install (nc2);
```

Další bod je instalování protokolové sady na vytvořené uzly, zařízení a kanály. Kontejner `all` obsahuje všechny uzly z naší sítě.

```
InternetStackHelper stackIPv6;
// Nainstalování protokolové sady na uzly z kontejneru (all)
stackIPv6.Install (all);
```

Přidělíme IP adresy rozhraním našich zařízení pomocí třídy `Ipv6AddressHelper`. Metodou `SetBase` nastavíme adresu IPv6 sítě a velikost prefixu. Následně spojíme IPv6 adresu a rozhraní zařízení metodou `Assign`. Pomocí metody `SetForwarding` automaticky nastavíme všechna rozhraní uzlů do stejného přeposílacího stavu. Číslo udává index uzlu a `true` specifikuje, že se uzel bude chovat jako směrovač. Poslední řádek nastaví výchozí cestu (default route) pro všechna zařízení kromě samotného směrovače.

```
Ipv6AddressHelper address1;
Ipv6AddressHelper address2;
Ipv6AddressHelper address3;

address1.SetBase (Ipv6Address ("2001:3::"), Ipv6Prefix (64));
Ipv6InterfaceContainer intcm = address1.Assign (devm);
intcm.SetForwarding (1, true);  // R1
intcm.SetForwarding (2, true);  // R2
intcm.SetDefaultRouteInAllNodes (1);  // Výchozí cesta do R1

address2.SetBase (Ipv6Address ("2001:4::"), Ipv6Prefix (64));
Ipv6InterfaceContainer intc1 = address2.Assign (dev1);
intc1.SetForwarding (0, true);  // R1
intc1.SetDefaultRouteInAllNodes (0);  // Výchozí cesta do R1

address3.SetBase (Ipv6Address ("2001:5::"), Ipv6Prefix (64));
Ipv6InterfaceContainer intc2 = address3.Assign (dev2);
intc2.SetForwarding (0, true);  // R2
intc2.SetDefaultRouteInAllNodes (0);  // Výchozí cesta do R2
```

Voláním metody `GetObject <IPv6>` získáme ukazatel na IPv6 objekt přičleněný uzlu R1. Pomocná třída `Ipv6StaticRoutingHelper` přidává objekty `Ipv6StaticRouting` a povoluje statické směrování. Metoda `GetStaticRouting` zajistí vyhledání statického směrování.

```
Ptr<Ipv6> ipv6R1 = R1->GetObject<Ipv6> ();
Ipv6StaticRoutingHelper routingHelper;
Ptr<Ipv6StaticRouting> routingR1 = routingHelper.GetStaticRouting (
    ipv6R1);
```

K vytvoření statické cesty ve směrovací tabulce na daném uzlu použijeme metodu `AddHostRouteTo`. První parametr je cílová adresa, druhý je adresa dalšího

skoku (next hop) a třetím je index rozhraní. Uvedené jsou dvě možnosti tvorby statické cesty. První je konkrétní zadání adresy a druhá je pomocí metod `GetAddress` a `GetInterfaceIndex`. Při metodě `GetAddress` je první parametr index rozhraní a další je index adresy (typ adresy), přičemž nula je určena pro lokální linkovou adresu. Metoda `GetInterfaceIndex` slouží k získání indexu daného rozhraní.

```
// Manuální přidání statické cesty na R1 do host2 přes R2
routingR1->AddHostRouteTo (Ipv6Address ("2001:5::200:ff:fe00:7"),
    Ipv6Address ("fe80::200:ff:fe00:3"), 1);
// routingR1->AddHostRouteTo (intc2.GetAddress (1, 1), intcm.
    GetAddress (2, 0), intcm.GetInterfaceIndex (1));
```

Tvorba ping aplikace je pomocí třídy `Ping6Helper`. Lokální adresu nastavíme adresu uzlu SOU a jako vzdálenou adresu nastavíme adresu uzlu host2. Adresu můžeme nastavit přímo nebo ji vybrat z kontejneru rozhraní pomocí metody `GetAddress`. Dále specifikujeme vlastnosti dané aplikace: maximální počet paketů zaslaných v průběhu simulace „MaxPackets“, doba čekání mezi zasláním dvou paketů „Interval“ a velikost přenášených uživatelských dat (payload) paketu „PacketSize“. Metodou `Install` připojíme aplikaci na zvolený uzel SOU. Nadefinujeme začátek a konec aplikace, během níž se bude generovat datový provoz.

```
Ping6Helper ping6;

// Lokální adresa (uzel SOU)
ping6.SetLocal (Ipv6Address ("2001:3::200:ff:fe00:1"));
// ping6.SetLocal (intcm.GetAddress (0, 1));
// Vzdálená adresa (uzel host2)
ping6.SetRemote (Ipv6Address ("2001:5::200:ff:fe00:7"));
// ping6.SetRemote (intc2.GetAddress (1, 1));
ping6.SetAttribute ("MaxPackets", UintegerValue (10));
ping6.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
ping6.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer appPing = ping6.Install (SOU);
appPing.Start (Seconds (3.0));
appPing.Stop (Seconds (15.0));
```

Vytištění směrovacích tabulek pro uzly R1, R2 a SOU ve třetí vteřině a pro uzel SOU ve čtrnácté vteřině do souboru *icmpv6a-routing*.

```
// Vytištěný směrovacích tabulek do souboru
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper
    > ("icmpv6a-routing", std::ios::out);
routingHelper.PrintRoutingTableAt (Seconds (3.0), R1, routingStream
    );
```

```
routingHelper.PrintRoutingTableAt (Seconds (3.0), R2, routingStream
    );
routingHelper.PrintRoutingTableAt (Seconds (3.0), SOU,
    routingStream);
routingHelper.PrintRoutingTableAt (Seconds (14.0), SOU,
    routingStream);
```

Předposledním bodem je nastavení sledování, které poskytuje detailní informace o komunikaci. Výstupem je řada trasovacích souborů ve formátu *pcap*, jejichž název bude složený z předpony „icmpv6a“, čísla uzlu, čísla zařízení a přípony *.pcap*.

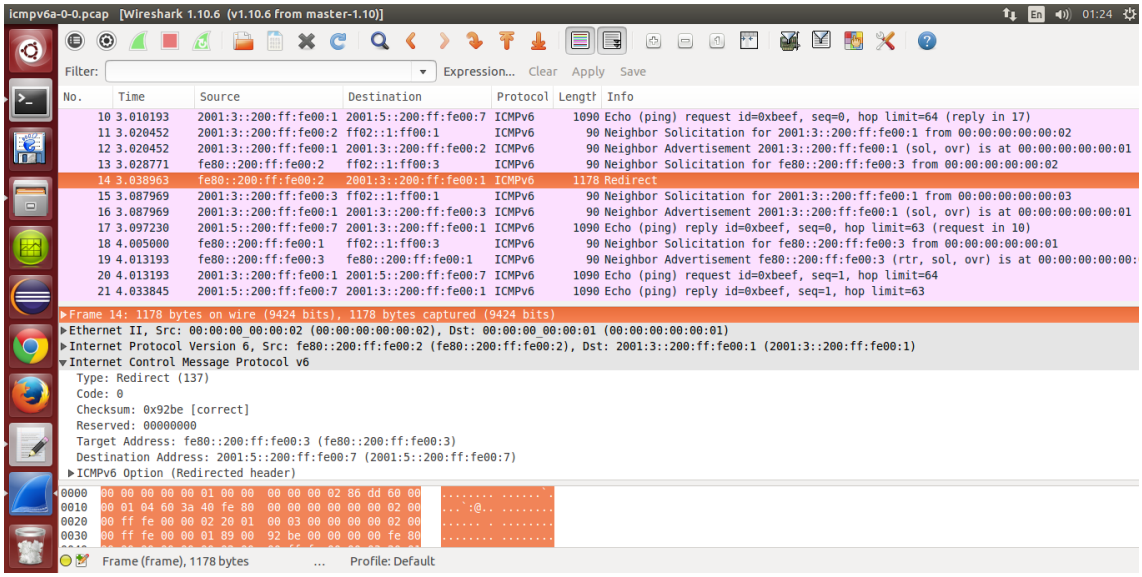
```
csma.EnablePcapAll ("icmpv6a");  // Vytvoření trasovacích souborů
```

Na závěr je nutné zadat globální funkci, která realizuje běh simulace. Po ukončení simulace má na starosti vymazání všech vytvořených objektů jiná globální funkce.

```
Simulator::Run ();  // Běh simulace
Simulator::Destroy ();  // Vymazání objektů
}
```

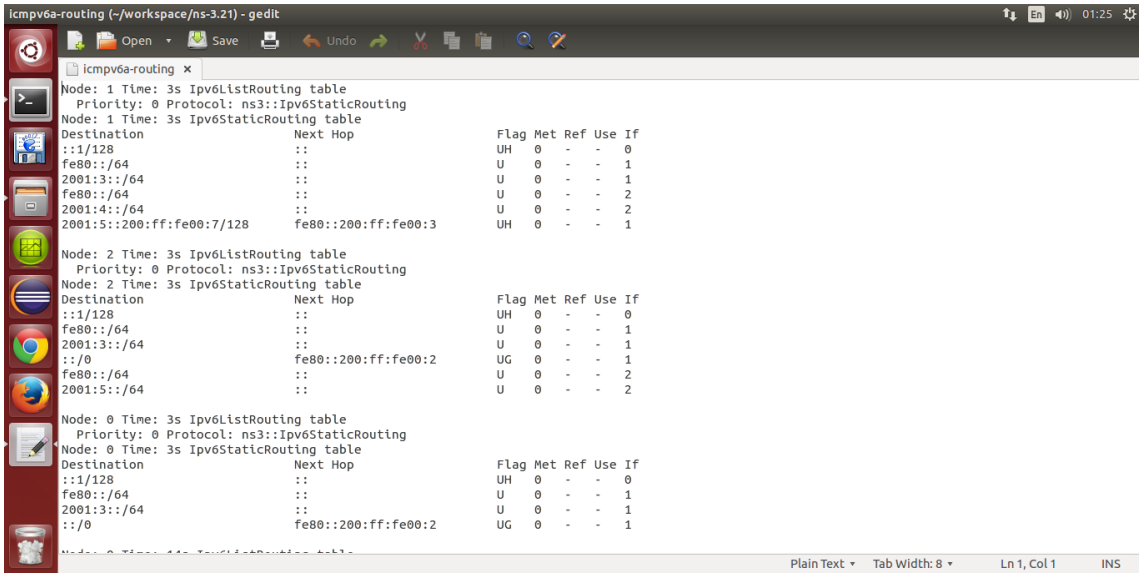
Podmínkou spuštění simulace je změna pracovního adresáře, vložení souboru *uloha3a.cc* do adresáře *scratch* a kompilace vytvořeného skriptu.

```
$ cd wokrspace/ns-3.21
$ ./waf
$ ./waf --run scratch/uloha3a
```



Obr. 6.5: Zobrazení komunikace na uzlu SOU pro aplikaci ping.

Výstupem simulace jsou trasovací soubory (icmpv6a) formátu *pcap* a soubor *icmpv6a-routing* obsahující směrovací tabulky. Prohlédnutím trasovacího souboru *icmpv6a-0-0.pcap* (obr. 6.5), v programu Wireshark, vidíme průběh komunikace. První důležitou událostí je, zaslání zprávy Echo Request z uzlu SOU s adresou 2001:3::200:ff:fe00:1, ve třetí vteřině, což je start aplikace ping. Cílem je uzel host2 s adresou 2001:5::200:ff:fe00:7. Tento paket je však výchozí cestou směrován na uzel R1. V souboru *icmpv6a-routing* (obr. 6.6) vidíme směrovací tabulky pro uzly R1 (Node1), R2 (Node2) a pro uzel SOU (Node0) ve třetí a čtrnácté sekundě. Zmíněnou výchozí cestu vidíme v tabulce pro uzel SOU (Node0) ve třetí vteřině. Uzel R1 po obdržení zprávy Echo Request zašle zprávu Redirect na uzel SOU a přepošle Echo Request na uzel R2 pomocí statické cesty (směrovací tabulka pro Node1), kterou jsme nakonfigurovali na uzlu R1. Uzel R2 přepošle zprávu na uzel host2 a ten zašle odpověď Echo Reply. Na uzel SOU je doručena zpráva Redirect, která zabezpečí přidání statické cesty na SOU, aby veškerá další komunikace s uzlem host2 směřovala přímo přes uzel R2. Tuto nově přidanou statickou cestu vidíme ve směrovací tabulce pro Node0 generované ve čtrnácté vteřině. Všechny další Echo Request zprávy jsou směrovány přímo přes uzel R2, jak pozorujeme na 20. řádku.



Obr. 6.6: Soubor se směrovacími tabulkami pro jednotlivé uzly.

Síťový hardware může provádět pokročilé funkce, jako je IP výpočet kontrolního součtu, také známý jako checksum offloading. Síťové drivery nebudou počítat samotný kontrolní součet, ale budou pouze předávat prázdný (nulami nebo neplatnými daty vyplněný) kontrolní součet hardwaru.

Kontrolní součty (Checksum offloading) často způsobují zmatek, když jsou

přenášené síťové pakety předány Wiresharku, před tím než jsou kontrolní součty skutečně vypočteny. Wireshark dostane tyto „prázdné“, kontrolní součty a zobrazuje je za neplatné, a to i přesto, že pakety budou obsahovat platné kontrolní součty, když opustí síťový hardware později.

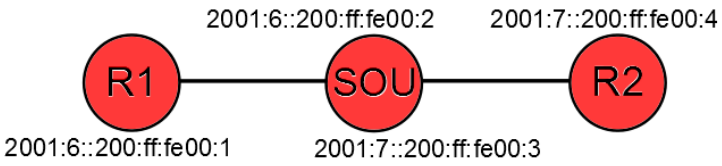
Pokud by FCSs (Frame Check Sequences) byly nesprávné, rámce by byly zahozeny a komunikace by selhala. Komunikace tedy byla úspěšná. Ve výchozím nastavení je kontrolní součet v ns-3 zakázán pro zrychlení simulace.

Wireshark -> Edit (horní lišta) -> Preferences -> Protocols (levý sloupec) -> Ethernet -> Unheck „Validate the Ethernet checksum if possible:“ -> Apply -> Ok.

6.3.2 Druhá část

Pro vypracování druhé části (obr. 6.7) můžeme použít jako základ soubor *uloha3a.cc*, ale bude třeba jej upravit. Z důvodu zachování původního souboru, ho uložíme pod novým názvem *uloha3b.cc* a právě tento soubor budeme upravovat. Zdrojový kód doplníme v prví řadě o prohlášení `#include` s hlavičkovými soubory.

```
#include "ns3/radvd.h"
#include "ns3/radvd-interface.h"
#include "ns3/radvd-prefix.h"
```



Obr. 6.7: Zadaná topologie pro druhou část.

Ponecháme tři uzly, které přejmenujeme R1, SOU, R2 a zbývající odstraníme. Důležité je dodržení pořadí uzlů R1, SOU, R2 ve zdrojovém souboru! Namísto stávajících kontejnerů uzlů vytvoříme tři. První bude obsahovat všechny tři uzly v pořadí R1, SOU, R2 a nazveme ho `all`. Druhý se bude jmenovat `nc1` a bude obsahovat uzly R1 a SOU. Třetí nazveme `nc2` a jeho uzly budou SOU a R2. Definované vlastnosti kanálu linky ponecháme („DataRate“, „Delay“), rovněž i atribut zařízení („Mtu“).

Z kontejnerů zařízení nám postačí dva nazvané `dev1` a `dev2`. Nainstalovanou protokolovou sadu na všech uzlech, zařízeních a kanálech ponecháme. Rovněž ponecháme objekty (`address1` a `address2`) třídy `Ipv6AddressHelper`, pomocí které přidělujeme IP adresy rozhraním našich zařízení.

Od tohoto řádku se kód liší (`Ipv6AddressHelper address2;`). Je vhodné, zbytek kódu odstranit a pokračovat podle postupu. Nejdříve je definovaná podsíť mezi uzly R1 a SOU. Nastavíme adresu a prefix pro nás již známou metodou `SetBase`. V dalším bodě vytvoříme nový kontejner zařízení a metodou `Add` přidáme obsah předem vytvořeného kontejneru zařízení `dev1` na konec nového kontejneru zařízení `tdev1`. My však z kontejneru `dev1` získáme ukazatel na konkrétní zařízení instalované na uzlu R1. Metoda `AssignWithoutAddress` zajistí, že zařízení je přiděleno rozhraní, ale adresa mu není přidělena. To je z důvodu pozdějšího přidělení adresy. Zopakujeme vytvoření nového kontejneru zařízení `tdev2` a ze stávajícího kontejneru `dev1`, získáme ukazatel na zařízení instalované na uzlu SOU. Tento vložíme do nového kontejneru zařízení. Prostřednictvím nového rozhraní vytvoříme spojení mezi zařízením a IP adresou. Metoda `SetForwarding` automaticky nastaví rozhraní uzlu SOU do stejného přeposílacího stavu. Tento uzel bude vystupovat jako směrovač. Zařízením na rozhraní nastavíme výchozí cestu (default route) metodou `SetDefaultRouteInAllNodes`, kromě samotného uzlu SOU (směrovače). Nakonec přidáme pár IP adresa a rozhraní. Analogicky je definovaná podsíť mezi uzly SOU a R2.

```
// Podsíť mezi R1 a SOU
address1.SetBase (Ipv6Address ("2001:6::"), Ipv6Prefix (64));
NetDeviceContainer tdev1;
tdev1.Add (dev1.Get (0));  // Zařízení na uzlu R1
// Rozhraní na uzlu R1
Ipv6InterfaceContainer intca = address1.AssignWithoutAddress (tdev1
    );

NetDeviceContainer tdev2;
tdev2.Add (dev1.Get (1));  // Zařízení na uzlu SOU
// Staticky nastaveny rozhraní na uzlu SOU
Ipv6InterfaceContainer intcra = address1.Assign (tdev2);
intcra.SetForwarding (0, true);
intcra.SetDefaultRouteInAllNodes (0);
intca.Add (intcra);

// Podsíť mezi SOU a R2
address2.SetBase (Ipv6Address ("2001:7::"), Ipv6Prefix (64));
NetDeviceContainer tdev3;
tdev3.Add (dev2.Get (0));  // Zařízení na uzlu SOU
```

```
// Rozhraní na uzlu SOU
Ipv6InterfaceContainer intcrb = address2.Assign (tdev3);
intcrb.SetForwarding (0, true);
intcrb.SetDefaultRouteInAllNodes (0);

NetDeviceContainer tdev4;
tdev4.Add (dev2.Get (1));  // Zařízení na uzlu R2
// Rozhraní na uzlu R2
Ipv6InterfaceContainer intcb = address2.AssignWithoutAddress (tdev4
    );
intcb.Add (intcrb);
```

Díky pomocné třídě `RadvdHelper` (Router advertisement daemon Helper), můžeme nastavit periodické posílání zpráv Router Advertisement bez vyžádání. Vysílané zprávy budou obsahovat prefix nastaven metodou `AddAnnouncedPrefix`. Zprávy budou zasílány z uzlu SOU směrem na uzel R1. Vysílán IPv6 prefix bude 2001:6::0 a jeho délka bude mít hodnotu 64. Analogicky bude nastaveno posílání zpráv Router Advertisement na uzel R2 z SOU, kde vysílaný IPv6 prefix bude 2001:7::0 a hodnota délky prefixu bude 64. Nicméně další řádek kódu zakáže periodické posílání zpráv. Proto, když bude chtít uzel R2 obdržet zprávu Router Advertisement musí o ni požádat zasláním Router Solicitation zprávy na uzel SOU.

```
RadvdHelper radvdHelper;  // Router advertisement daemon Helper
radvdHelper.AddAnnouncedPrefix (intca.GetInterfaceIndex (1),
    Ipv6Address("2001:6::0"), 64);
radvdHelper.AddAnnouncedPrefix (intcb.GetInterfaceIndex (1),
    Ipv6Address("2001:7::0"), 64);
radvdHelper.GetRadvdInterface (intcb.GetInterfaceIndex (1))->
    SetSendAdvert (false);  // Zákaz zasílání RA na uzel R2
```

Pokračujeme nainstalováním aplikace appRadvd na uzel SOU a nastavením začátku a konce běhu této aplikace.

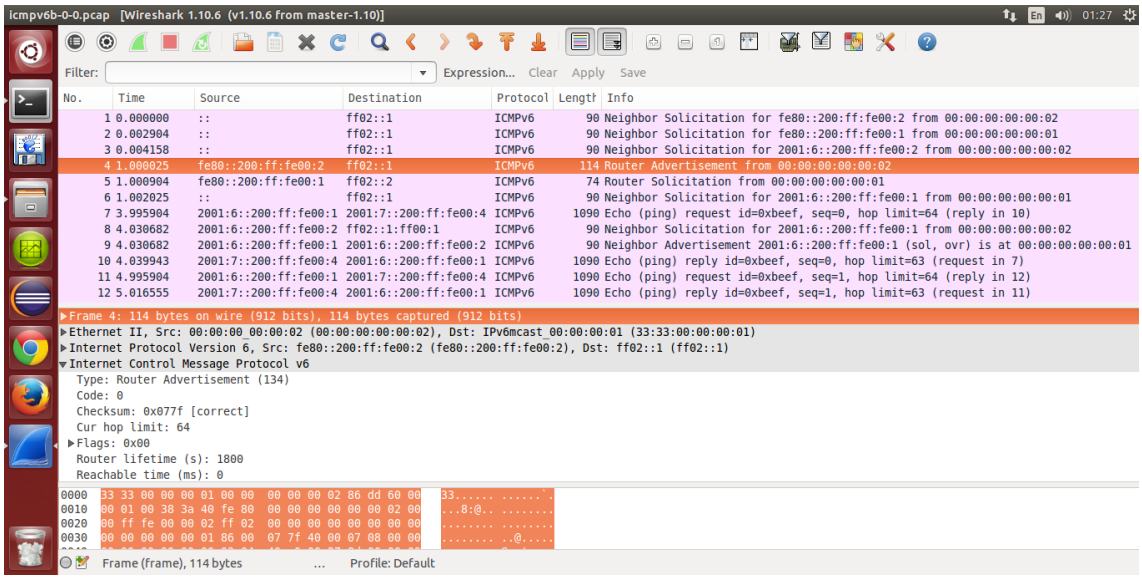
```
ApplicationContainer appRadvd = radvdHelper.Install (SOU);
appRadvd.Start (Seconds (1.0));
appRadvd.Stop (Seconds (15.0));
```

Pro správné zadání adresy je důležité si uvědomit, že uzly R1 a R2 mají každý jedno rozhraní a uzel SOU má dvě rozhraní. Každému rozhraní je přidělena jedna MAC adresa. Pro uzel R1 je to 00:00:00:00:00:01 a číslování pokračuje za sebou. Na uzlu SOU se společným rozhraním s uzlem R1 je MAC adresa 00:00:00:00:00:02. Dále je na uzlu SOU se společným rozhraním s uzlem R2 MAC adresa 00:00:00:00:00:03. Poslední použita MAC adresa je 00:00:00:00:00:04 na uzlu R2. Od tohoto faktu se odvíjejí i IPv6 adresy.

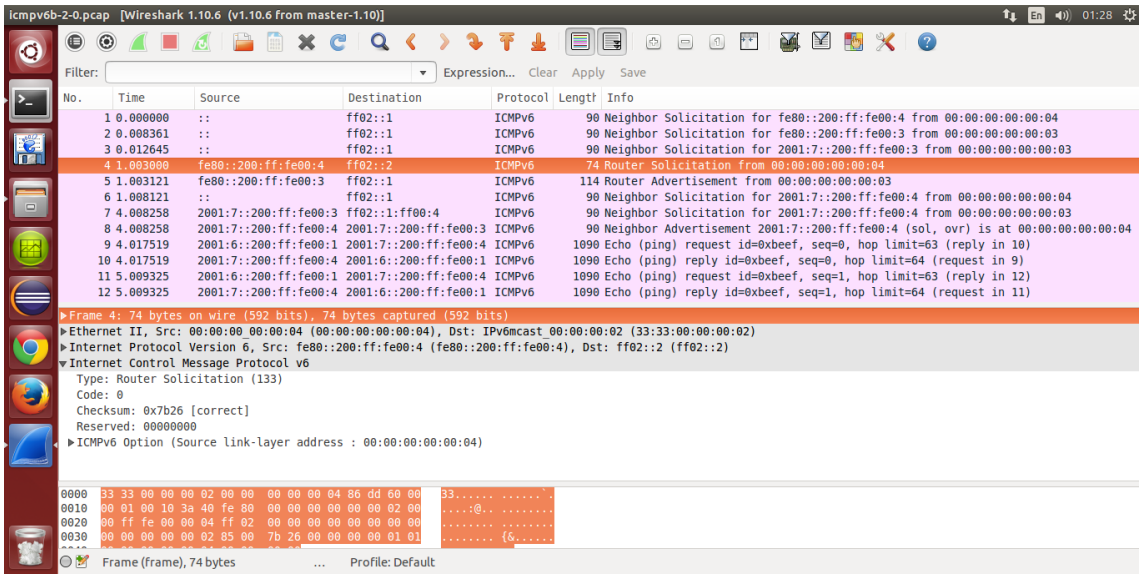
Doplníme i aplikaci ping, se kterou jsme se již seznámili. Lokální adresa bude zodpovídat adrese na uzlu R1. Vzdálená adresa bude po autokonfiguraci 2001:7::200:ff:fe00:4 a odvodíme ji na základě MAC (fyzické) adresy přidělené tomuto zařízení na uzlu R2. Atributy „MaxPackets“, „Interval“ a „PacketSize“ nastavte podle první části úlohy. Metodou `Install` připojte aplikaci ping na uzel R1. Spuštění nastavte na hodnotu čtyři a ukončení na hodnotu patnáct.

```
Ping6Helper ping6;

// Zdroj R1
ping6.SetLocal (Ipv6Address ("2001:6::200:ff:fe00:1"));
ping6.SetRemote (Ipv6Address ("2001:7::200:ff:fe00:4"));  // Cíl R2
// MaxPackets
// Interval
// PacketSize

// Install
// Start
// Stop
```

Nakonec přidejte podobné sledování, jako v první části úlohy (`EnablePcap` („název souboru“, id uzlu, id zařízení)) pro uzly R1 („icmpv6b“, 0, 0) a R2 („icmpv6b“, 2, 0). Výstupem bude soubor (icmpv6b) formátu *pcap*. Dopište globální funkce pro spuštění simulace a vymazání vytvořených objektů po ukončení simulace. Zdrojový kód zakončete složenou závorkou.



Obr. 6.8: Zobrazení komunikace na uzlu R1.



Obr. 6.9: Zobrazení komunikace na uzlu R2.

Zkompilujte vytvořený skript, spusťte simulaci a následně si projděte výsledky. Výstupem druhé části úlohy jsou sledovací soubory (icmpv6b) formátu *pcap*. V programu Wireshark sledujeme komunikaci na jednotlivých uzlech. Rozdíl spočívá mezi uzly R1 a R2. V souboru *icmpv6b-0-0.pcap* (obr. 6.8), tedy na uzlu R1 vidíme, že jako první byla zaslaná zpráva Neighbor Solicitation v čase 0.0, z nespecifikované IPv6 adresy a fyzické adresy 00:00:00:00:00:02, z uzlu SOU na IPv6 multicastovou adresu ff02::1 všech uzlů a fyzickou adresou 33:33:ff:00:00:02 (uzel R1). Tato zpráva je určena pro proces bezestavové automatické konfigurace k ověření duplicity adres na nové lokální linkové IPv6 adrese předtím než je přidělena rozhraní. V těle zprávy se nachází nezávazná lokální linková adresa fe80::200:ff:fe00:2 uzlu SOU. Stejným způsobem je ověřena jedinečnost lokální linkové adresy fe80::200:ff:fe00:1 uzlu R1. Dále je ověřena jedinečnost globální IPv6 adresy 2001:6::200:ff:fe00:2 uzlu SOU shodným postupem, jako v předchozích případech lokálních linkových adres, až na to, že v těle zprávy se přenáší globální IPv6 adresa. Následuje zpráva Router Advertisement z lokální linkové adresy fe80::200:ff:fe00:2 uzlu SOU na multicastovou adresu všech uzlů ff02::1 (uzlu R1) a následné uzel R1 zašle z lokální linkové adresy fe80::200:ff:fe00:1 zprávu Router Solicitation na multicastovou adresu všech směrovačů ff02::2 (uzlu SOU). Zpráva Router Advertisement byla doručena uzlu R1 bez vyžádání a nese IPv6 prefix, který použije uzel R1 na lokálním spoji pro automatickou konfiguraci globální IPv6 adresy. Stejně tato zpráva nese informace o době života prefixu. Při pohledu na komunikaci v souboru *icmpv6b-2-0.pcap* (obr. 6.9), uzel R2, vidíme rozdíl. Nejdříve byly zaslány dvě zprávy Neighbor Solicitation ověřující

jedinečnost lokálních linkových adres uzlu R2 a SOU. Následně byla zaslána zpráva Neighbor Solicitation ověřující jedinečnost globální IPv6 adresy uzlu SOU. Dále byla zaslána zpráva Router Solocitation z lokální linkové adresy fe80::200:ff:fe00:4 uzlu R2 na multicastovou adresu všech směrovačů ff02::2 (uzlu SOU) a až následně poslal uzel SOU z lokální linkové adresy fe80::200:ff:fe00:3 zprávu Router Advertisement na multicastovou adresu všech uzlů ff02::1 (uzlu R2). Uzel R2 si vyžádal zaslání Router Advertisement pomocí zprávy Router Solicitation. Zpráva Router Advertisement nesla IPv6 prefix, který použil uzel R2 na lokálním spoji pro automatickou konfiguraci globální IPv6 adresy.

6.4 Kontrolní otázky

1. Jaké typy IPv6 adres se nacházely ve směrovací tabulce uzlů? (unicastové, multicastové)
2. Jaké jsou základní funkce protokolu ICMPv6?
3. Uveďte některé typy ICMPv6 zpráv.
4. Jaký je název protokolu definujícího zprávu ICMPv6 typu Redirect? Tento protokol nahrazuje a vylepšuje funkce protokolu odpovědného za nalezení IPv4 (síťové) adresy na základě znalosti MAC (fyzické) adresy.
5. Jaká je úloha zpráv Redirect, Router Advertisement a Router Solicitation (na co slouží)? Na základě porovnání obsahu směrovacích tabulek, předveďte primární roli zpráv Router Advertisement. Doplňte zdrojový kód, tak abyste do souboru *icmpv6b-routing* zobrazily směrovací tabulky v čase 0.1 a 15. Využijte metodu `PrintRoutingTableAllAt`. Po uložení a zkompilování kódu si otevřete vytvořený soubor a porovnejte směrovací tabulky pro jednotlivé uzly.

7 ÚLOHA 4: WIFI S PODPOROU QOS

7.1 Zadání

1. Prostudujte si teoretický úvod.
2. Vytvořte pomocí vypracování soubor *uloha4.cc*.
3. Prohlédněte si výpis parametrů On / Off aplikace používající protokol UDP v terminálovém okně.
4. Otevřete si v programu Wireshark soubor *wifiQos-0-0.pcap* a zobrazte, jak probíhala komunikace. Soustřeďte se na nastavení hodnoty priority (přístupové kategorie) paketu. Všimněte si, jak se zobrazuje číslo portu 80.
5. Zobrazte si průběh simulace v Netanimu.
6. Změňte prioritu (přístupovou kategorii), čili hodnotu atributu „AccessClass“ na hodnotu 1 ve vytvořené aplikaci.
7. Vytvořte druhou On / Off aplikaci využívající protokol UDP a použijte port 5004. Zdroj bude uzel station1 a cíl uzel station3. Jako vzdálenou adresu zvolte 192.168.1.3. Nastavte atributy podle předchozí aplikace kromě „AccessClass“ atributu, jeho hodnota bude 5. Čas spuštění (5) a čas ukončení (25).
8. Vytvořte třetí On / Off aplikaci využívající protokol UDP a použijte port 5060. Zdroj bude uzel station1 a cíl uzel station4. Jako vzdálenou adresu zvolte 192.168.1.4. Nastavte atributy podle předchozí aplikace kromě „AccessClass“ atributu, jeho hodnota bude 6. Čas spuštění (5) a čas ukončení (25).
9. Vytvořte On / Off aplikaci využívající protokol TCP a použijte port 20. Zdroj bude uzel station1 a cíl uzel station5. Jako vzdálenou adresu zvolte 192.168.1.5. Nastavte atributy podle předchozí aplikace kromě „AccessClass“ atributu, jeho hodnota bude 0. Čas spuštění (5) a ukončení (25) aplikace ponechte. Použijte třídu TcpSocketFactory. Tvorba aplikace je velice podobná On / Off aplikaci využívající protokol UDP.
10. Po opětovné kompilaci a spuštění souboru *uloha4.cc* si prohlédněte výstup v terminálovém okně. Jednotlivé datové toky zodpovídají aplikacím. Výjimkou je On / Off aplikace využívající protokol TCP, která má dva datové toky. Rozpoznání těchto datových toků příslušejících jednotlivým On / Off aplikacím proveďte na základě IP adres zdroje (Src Address) a cíle (Dst Address). Jak se liší hodnoty parametrů průměrná doba zpoždění „Mean Delay“ a průměrná doba kolísaní zpoždění „Mean Jitter“ pro jednotlivé aplikace? Uvažujte nad souvislostmi mezi nastavením priorit a hodnotami těchto parametrů.
11. Otevřete si v programu Wireshark soubor *wifiQos-0-0.pcap* a zobrazte, jak probíhala komunikace. Soustřeďte se na nastavenou hodnotu priority (přístupové

kategorie) paketu. Všimněte si, jak se zobrazují čísla jednotlivých portů.

12. Pomocí dvojitého lomítka „//“ zakomentujte v každé aplikaci řádek, který nastavuje hodnotu „AccessClass“ atributu. Opětovně zkompilujte a spusťte soubor *uloha4.cc*. Jak se liší hodnoty parametrů průměrná doba zpoždění „Mean Delay“ a průměrná doba kolísaní zpoždění „Mean Jitter“ pro jednotlivé aplikace bez nastavení priority (přístupové kategorie)? V programu Wireshark si ověřte nastavení priority u jednotlivých aplikací. Porovnejte odlišnosti parametrů při použití kvality služeb QoS a bez použití kvality služeb QoS.
13. Odpovězte na otázky na konci úlohy.

7.2 Teoretický úvod

7.2.1 Wifi

Wifi je sada standardů IEEE 802.11 popisujících bezdrátovou komunikaci v počítačových sítích. Tato technologie využívá bez licenční pásmo o frekvenci 2,4 a 5 GHz, proto se jedná o levné řešení. Nevýhoda spočívá v rušení domácími spotřebiči, jako mikrovlnná trouba, pracujícími na stejné frekvenci. Silné zarušení příslušného frekvenčního spektra nastává v městských oblastech z důvodu hustého překrývání. Další nevýhoda oproti kabelovým spojům je nižší bezpečnost, protože útočník nepotřebuje fyzické připojení.

Identifikátor SSID (Service Set Identifier) je řetězec až 32 ASCII znaků, kterými jednoznačně identifikuje a tedy odlišuje jednotlivé sítě. Vysílán je SSID identifikátor v pravidelných intervalech jako broadcast (všesměrové vysílání), takže všichni klienti si mohou snadno zobrazit dostupné bezdrátové sítě, ke kterým je možné se připojit. Pro umožnění komunikace mezi zařízeními různých výrobců existují mezinárodní standardy IEEE 802.11 (tab. 7.1). IEEE 802.11 je sada specifikací MAC (Media Access Control) a fyzické vrstvy pro implementaci bezdrátové lokální sítě v pásmu 2,4 a 5 GHz.

Stanice je označení zařízení kombinujícího počítač a bezdrátovou síťovou kartu. Jeden rádiový komunikační kanál je sdílený všemi stanicemi v dosahu.

IEEE 802.11e je standard, který definuje kvalitu služeb QoS určenou pro bezdrátové lokální sítě prostřednictvím modifikace podvrstvy MAC (Media Access Control). Vrstva MAC je podvrstvou linkové vrstvy.

7.2.2 Distribuovaná Koordinační Funkce

Základem 802.11 MAC podvrstvy je distribuovaná koordinační funkce DCF (Distributed Coordination Function), která sdílí medium mezi více komunikujícími

Standard	Pásmo [GHz]	Maximální rychlost [Mbps]	Modulace
IEEE 802.11	2,4	2	DSSS, FHSS
IEEE 802.11a	5	54	OFDM
IEEE 802.11b	2,4	11	DSSS
IEEE 802.11g	2,4	54	OFDM, DSSS
IEEE 802.11n	2,5 / 5	50	OFDM, MIMO
IEEE 802.11ac	5	866	OFDM, MIMO

Tab. 7.1: Přehled standardů IEEE 802.11.

stranami. DCF spočívá na metodě CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), co je metoda mnohonásobného přístupu s detekcí nosné a vyhýbání se kolizím. Pro vylepšení řízení přístupu mohou zprávy obsahovat informaci, která udává, po jakou dobu bude ještě stanice obsazovat přenosové médium. Pokud se mnoho stanic pokusí komunikovat současně, dojde k velkému počtu srážek, které sníží dostupnou šířku pásma. Není tady garance QoS, protože DCF neobsahuje žádný mechanizmus pro prioritizaci přístupu k médiu. [1]

Distribuovaná koordinační funkce pracuje při přístupu k médiu s oknem soutěžení (Contention Window). Jeho velikost je pro každou stanici a třídu provozu dáno intervalem CWmin a CWmax. Jedná se o minimální a maximální hodnoty, kterých může velikost okna dosáhnout. Pokud má stanice data k odeslání, detekuje, zda je médium volné. Pokud ano, vygeneruje náhodné číslo v intervalu <0, w - 1>, kdy w je rovno CWmin. Poté začne od tohoto čísla odpočítávat. Během celého odpočtu neustále kontroluje, zda je médium volné. Pokud není, odpočítávání je zastaveno.

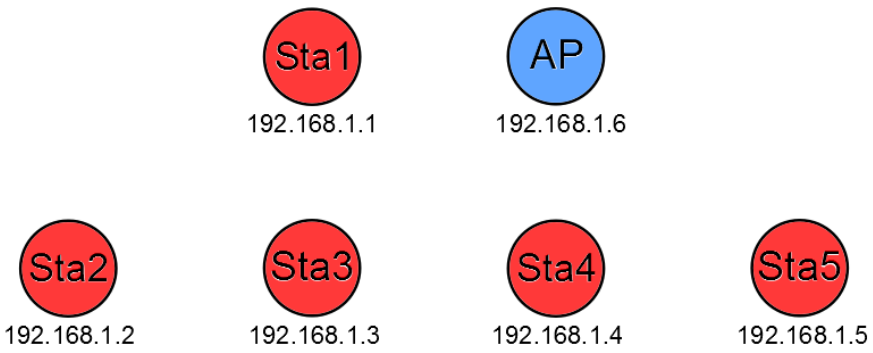
7.2.3 Hybridní Koordinační Funkce

802.11e vylepšuje DCF prostřednictvím nové koordinační funkce HCF (Hybrid Coordination Function). HCF obsahuje dvě metody přístupu na kanál: HCCA (HCF Controlled Channel Access) a EDCA (Enhanced Distributed Channel Access). Obě metody definují kategorie provozu. Vyšší priorita provozu u EDCA značí vyšší šance na zaslání, nežli nižší priorita provozu. V průměru stanice s vyšší prioritou čeká kratší dobu na zaslání, oproti stanici s nižší prioritou. Zaručeno je to přes protokol TCMA, který je obdobou CSMA/CA s použitím kratšího rozhodování u mezi rámcové mezery pro datové jednotky vyšší priority. Přesná hodnota závisí na fyzické vrstvě použité k přenosu dat. EDCA navíc poskytuje přístup na kanál během periody nazývané vysílací příležitost. Stanice tento časový interval využije na

zaslání tolika rámců kolik je možné. Úrovně priorit jsou v EDCA označovány, jako přístupové kategorie ACs (Access Categories). Okno soutěžení může být nastaveno v závislosti na očekávaném provozu v každé přístupové kategorii. Širší okno je potřebné pro kategorie s větším provozem. [1]

7.3 Vypracování

Úloha *uloha4.cc* spočívá ve vytvoření wifi (bezdrátové) sítě (obr. 7.1) standardu 802.11b tvořené v našem případě pěti stanicemi a jedním přístupovým bodem. Mezi stanicemi bude probíhat datový přenos realizovaný prostřednictvím aplikací. V síti uplatníme kvalitu služeb QoS (Quality of Service), která zabezpečí roztřídění komunikace do tříd podle priorit nastavených u jednotlivých aplikací. Priority jsou nazývány přístupové kategorie (Access Categories = ACs) a budeme rozlišovat čtyři typy: Best Effort (AC_BE), Background (AC_BK), Video (AC_VI), Voice (AC_VO). V této úloze budeme pozorovat jaký vliv má na komunikaci využití QoS.



Obr. 7.1: Zadaná topologie.

Kód souboru *uloha4.cc* začíná prohlášeními `#include`, pomocí kterých se přidávají hlavičkové soubory.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
```

```
#include "ns3/internet-module.h"
#include "ns3/config-store-module.h"
#include "ns3/olsr-helper.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/ipv4-list-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
```

Následuje deklarace jmenného pole `ns3`, záznamové funkce povolující selektivní zobrazení záznamů a deklarování hlavní funkce `main`.

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("WifiQosLOG");

int main (int argc, char **argv)  // Deklarace funkce main
{
```

Použitý je grafický animační nástrojem NetAnim a proto je nutné definovat výstupní XML soubor s názvem „wifiQos“.

```
std::string animFile = "wifiQos.xml";  // Výstupní XML soubor
```

Pomocník tvorby topologie `NodeContainer` nám umožní pohodlné vytvoření, řízení a přístup k uzlům. První řádek deklaruje objekt třídy `NodeContainer` s názvem `wifiNodes`. Ve druhém řádku je volána metoda `Create` na objekt `wifiNodes`, aby vytvořila námi definovaný počet uzlů (pět), tedy stanic v kontejneru. Stejným způsobem vytvoříme uzel AP (přístupový bod) v novém kontejneru uzlů. Další řádky slouží k vytvoření ukazatelů na jednotlivé uzly v příslušných kontejnerech uzlů.

```
NodeContainer wifiNodes;  // Vytvoření kontejneru uzlů
wifiNodes.Create (5);
NodeContainer wifiApNode;
wifiApNode.Create (1);
// Vytvoření ukazatele na uzel
Ptr<Node> station1 = wifiNodes.Get (0);
Ptr<Node> station2 = wifiNodes.Get (1);
Ptr<Node> station3 = wifiNodes.Get (2);
Ptr<Node> station4 = wifiNodes.Get (3);
Ptr<Node> station5 = wifiNodes.Get (4);
Ptr<Node> AP = wifiApNode.Get (0);
```

Konstrukce wifi kanálu je realizována pomocnou třídou `YansWifiChannelHelper` s ponecháním výchozích nastavení propagačních strat a propagačního zpoždění. Pomocná třída `YansWifiPhyHelper` je využita ke konfiguraci fyzické vrstvy. Jako při

wifi kanálu, tak při phy se nastaví výchozí nastavení pomocí funkce `Default`. Nakonec asociujeme vytvořený kanálový objekt s objektem fyzické vrstvy phy, aby všechny objekty fyzické vrstvy sdíleli stejný kanál (bezdrátové medium).

```
// Konstrukce wifi kanálu
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
// Konfigurace fyzické vrstvy
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());
```

Od fyzické vrstvy se přesouváme k podvrstvě MAC (Media access control). Zadefinujeme wifi standard pomocí třídy `WifiHelper`. Ve výchozím nastavení je použitý standard 802.11a my použijeme standard 802.11b. Metodou `SetRemoteStation Manager` nastavíme algoritmus kontroly rychlosti na konstantní, tedy konstantní rychlost 2 Mbps pro přenosy dat (atribut „DataMode“) a konstantní rychlost 1 Mbps pro přenosy RTS (Request to Send) řídících paketů („ControlMode“). Použita technika bude DSSS (Direct Sequence Spread Spectrum) přímého rozprostřeného spektra.

```
WifiHelper wifi;
// Definice wifi standardu
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "
    DataMode",StringValue ("DsssRate2Mbps"),
"ControlMode",StringValue ("DsssRate1Mbps"));
```

Další bod je spuštění podpory (atribut „QosSupported“ nastaven na hodnotu `true`) a definice kvality služeb QoS zajištěných třídou `QosWifiMacHelper`. Objekt této třídy ponecháme ve výchozím nastavení. Nakonfigurujeme vlastní SSID (Service Set Identifier) identifikátor bezdrátové sítě. Metodou SetType se volí konkrétní typ MAC podvrstvy, se kterou se bude pracovat. Zvolena byla podvrstva `StaWifiMac`, kterou určuje atribut „ns3::StaWifiMac“. Dalšími atributy jsou „Ssid“ (námi definováno SSID) a „AcitveProbing“ nastaven na hodnotu `false` (výchozí nastavení), co značí, že nebudou vysílány probe žádosti. Potřebné je uzlům přiradit síťové zařízení. Proto vytvoříme kontejner zařízení `staDevices` a metodou `Install` přiradíme zařízení příslušným uzlům (stanicím) s patřičnými fyzickými a MAC vlastnostmi, které jsme nadefinovali. Stejně jako pro stanice (`wifiNodes`), tak i pro přístupový bod AP (`wifiApNode`) zvolíme typ MAC podvrstvy, nastavíme atributy a přidáme síťové zařízení.

```
// Definice kvality služeb
QosWifiMacHelper mac = QosWifiMacHelper::Default ();

Ssid ssid = Ssid ("wifiQos");
```

```
mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "
    ActiveProbing", BooleanValue (false));
NetDeviceContainer staDevices;
// Volba konkrétního typu podvrtsvy MAC
staDevices = wifi.Install (phy, mac, wifiNodes);

mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
NetDeviceContainer apDevice;
apDevice = wifi.Install (phy, mac, wifiApNode);
```

Pomocí mobility modelu je možné nastavit mobilitu stanic a přiradit pozici uzlům. V této úloze jsme nastavili konstantní pozici všem uzlům jak stanicím, tak i přístupovému bodu AP.

```
MobilityHelper mobility;  // Vytvorění modelu mobility
Ptr<ListPositionAllocator> positionAlloc = CreateObject<
    ListPositionAllocator>();
// Nastavení pozice uzlu
positionAlloc->Add (Vector (5.0, 5.0, 0.0));
positionAlloc->Add (Vector (0.0, 10.0, 0.0));
positionAlloc->Add (Vector (5.0, 10.0, 0.0));
positionAlloc->Add (Vector (10.0, 10.0, 0.0));
positionAlloc->Add (Vector (15.0, 10.0, 0.0));
positionAlloc->Add (Vector (10.0, 5.0, 0.0));
mobility.SetPositionAllocator (positionAlloc);
mobility.Install (wifiNodes);
mobility.Install (wifiApNode);
```

Směrovací protokol OLSR (Optimized Link State Routing) je povolen v simulaci použitím třídy `OlsrHelper`. Povoleno je i statické směrovaní pomocí třídy `Ipv4StaticRoutingHelper`. Metodou `Add` je přidáno statické směrování i OLSR do objektu třídy `Ipv4ListRoutingProtocol` s prioritou 0 pro staticRouting a 10 pro olsr. Co znamená prioritní použití OLSR před statickým směrováním.

```
OlsrHelper olsr;  // Povolení směrovacího protokolu OLSR

// Povolení statického směrování
Ipv4StaticRoutingHelper staticRouting;

Ipv4ListRoutingHelper list;
list.Add (staticRouting, 0);
list.Add (olsr, 10);
```

Protokolová sada je dalším bodem. Pomocná třída `InternetStackHelper` zjednoduší instalaci internetové sady protokolů na uzly v kontejneru. Předtím metoda `SetRoutingHelper` přidá do sady protokolů směrovací protokol.

```
InternetStackHelper stack;
// Přidání směrovacího protokolu do sady protokolů
stack.SetRoutingHelper (list);
// Nainstalování protokolové sady na uzly stanic
stack.Install (wifiNodes);
stack.Install (wifiApNode);
```

Pomocí třídy `Ipv4AddressHelper` přidělíme IP adresy rozhraním zařízení. Metodou `SetBase` nastavíme IP adresu a masku sítě. Následně spojíme IP adresu a vytvořené rozhraní zařízení metodou `Assign`.

```
Ipv4AddressHelper address;
address.SetBase ("192.168.1.0", "255.255.255.0");

Ipv4InterfaceContainer wifiNodesInterfaces;
Ipv4InterfaceContainer apNodeInterface;
// Propojení IP adresy a rozhraní zařízení
wifiNodesInterfaces = address.Assign (staDevices);
apNodeInterface = address.Assign (apDevice);
```

Ve skriptu použijeme On / Off aplikaci vytvořenou pomocí třídy `OnOffHelper`. Při této aplikaci byl použit UDP protokol. Atribut „Ns3::UdpSocketFactory“ je název protokolu, který slouží aplikaci k odeslání dat. Tento řetězec identifikuje typ továrny socketu použité k vytvoření socketu pro aplikaci. Socket je definován IP adresou a číslem portu. Dalším atributem po protokolu je adresa vzdáleného uzlu, tedy příjemce dat. Je jím druhý uzel (station2) s IP adresou 192.168.1.2. a cílovým portem 80. Důležité je nastavení atributů přenosová rychlost „DataRate“ a velikost přenášených uživatelských dat (payload) paketu „PacketSize“. Zapnutá bude aplikace stále „OnTime“ (constant=1) a vypnutá nebude nikdy „OffTime“ (constant=0). Atribut s názvem „AccessClass“ definuje prioritu tedy přístupovou kategorii (AC_BE=0, AC_BK=1, AC_VI=5, AC_VO=6), která je využita při třídění datových toků do jednotlivých tříd pro zabezpečení kvality služeb QoS. Aplikaci nainstalujeme na první uzel (station1) a nastavíme začátek a konec jejího běhu. PacketSinkHelper vytváří socket (IP adresa zdroje dat a cílový port) použitím třídy `UdpSocketFactory`. Jedná se o přijímač paketů pro UDP spojení na druhém uzlu (station2) s určeným časem pro začátek a konec běhu aplikace.

```
OnOffHelper onoffhelper ("ns3::UdpSocketFactory", Address (
    InetSocketAddress (Ipv4Address ("192.168.1.2"), 80)));

onoffhelper.SetAttribute ("DataRate", StringValue ("0.16Mbps"));
onoffhelper.SetAttribute ("PacketSize", UintegerValue (512));
onoffhelper.SetAttribute ("OnTime", StringValue ("ns3::
    ConstantRandomVariable[Constant=1]"));
```

```
onoffhelper.SetAttribute ("OffTime", StringValue ("ns3::
    ConstantRandomVariable[Constant=0]"));
onoffhelper.SetAttribute ("AccessClass", UintegerValue (
    UintegerValue (0)));  // AC_BE = best effort

ApplicationContainer appUDP = onoffhelper.Install (station1);
appUDP.Start (Seconds (5.0));
appUDP.Stop (Seconds (25.0));

PacketSinkHelper sinkPacketUDP ("ns3::UdpSocketFactory", Address (
    InetSocketAddress (Ipv4Address::GetAny (), 80)));
appUDP = sinkPacketUDP.Install (station2);
appUDP.Start (Seconds (5.0));
appUDP.Stop (Seconds (25.0));
```

Výpis směrovacích tabulek protokolu OLSR pro všechny uzly si v určitých časových okamžicích (2. sekunda a 25. sekunda) zobrazíme metodou `PrintRoutingTableAllAt` do *souboru wifiQos.routes*.

```
// Vytištěný směrovacích tabulek do souboru
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper
    > ("wifiQos.routes", std::ios::out);
olsr.PrintRoutingTableAllAt (Seconds (2), routingStream);
olsr.PrintRoutingTableAllAt (Seconds (25), routingStream);
```

Detailní informace o posílaných datech získáme metodou `EnablePcap`. Výstupem je trasovací soubor „wifiQos“ ve formátu *pcap* pro první uzel station1.

```
// Vytvoření trasovacích souborů
phy.EnablePcap ("wifiQos", staDevices.Get (0));
```

Modul Flow Monitor poskytuje flexibilitu při měření výkonu síťových protokolů. Používá sondy instalovány v uzlech na sledování výměny paketů mezi uzly a měření specifických parametrů. Pakety jsou členěny prostřednictvím toků, do kterých patří na základě charakteristiky sondy (protokolu, zdrojové, cílové IP adresy, čísla portu). Mezi základní parametry se řadí identifikace toku „Flow ID“, počet přijatých paketů „Rx Packets“, počet odeslaných paketů „Tx Packets“ a další. `NS_LOG_UNCOND` umožňuje výpis parametrů do terminálového okna. Sestavením vztahů ze seznamu parametrů můžeme vypočítat další charakteristiky například propustnost „Throughput“. Použití modulu je zjednodušeno pomocí třídy `FlowMonitorHelper` a výstup je uložen do souboru „wifiQosFlowmon“ ve formátu *xml*.

```
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll ();

Simulator::Stop (Seconds (30.0));  // Zastavení simulace
```

```
// NetAnim
  // Vložení kódu pro NetAnim
// End NetAnim

Simulator::Run ();

monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier
   > (flowmon.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->
   GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
   stats.begin (); i != stats.end (); ++i)
  {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->
           first);
        NS_LOG_UNCOND( "\n Flow ID: " << i->first << " (Src Addr: "
            << t.sourceAddress << " -> Dst Addr: " << t.
           destinationAddress << ")");
        NS_LOG_UNCOND( "  Tx Packets: " << i->second.txPackets);
        NS_LOG_UNCOND( "  Rx Packets: " << i->second.rxPackets);
        NS_LOG_UNCOND( "  Mean Delay: " << i->second.delaySum.
           GetSeconds () / (i->second.rxPackets));
        NS_LOG_UNCOND( "  Mean Jitter: " << i->second.jitterSum.
           GetSeconds () / (i->second.rxPackets - 1));
        NS_LOG_UNCOND( "  Throughput: " << i->second.rxBytes * 8.0
           / (i->second.timeLastRxPacket.GetSeconds () - i->second.
           timeFirstTxPacket.GetSeconds ()) / 1024  << " Kbps");
  }
monitor->SerializeToXmlFile("wifiQosFlowmon", true, true);
```

Animaci zobrazíme pomocí animačního nástroje NetAnim. Jedná se o samostatný software používající sledovací soubory, generované v průběhu simulace pro zobrazení topologie a animace toků paketů mezi uzly. Třída `AnimationInterface` je zodpovědná za vytváření sledovacího XML souboru *wifiQos.xml*. Potřebné je nastavit pozici jednotlivých uzlů v animaci. Slouží k tomu metoda `SetConstantPosition`, kde první atribut je ukazatel na uzel a zbylé dva udávají pozici na osách X a Y. Další užitečná metoda je `EnablePacketMetadata` zobrazující meta informace o paketech. Konkrétně se jedná o zobrazení IP adres zdroje a cíle pro On / Off aplikace a čísla portů pro aplikace využívající protokol UDP. Metoda `UpdateNodeDescription` (pořadové číslo uzlu, „zobrazený název uzlu“) slouží ke změně názvu jednotlivých uzlů a metoda `UpdateNodeColor` (uzel, R, G, B) ke změně barvy uzlu v NetAnimu.

```
AnimationInterface anim (animFile);
```

```
//anim.SetConstantPosition (station1, 5, 5);
//anim.SetConstantPosition (station2, 0, 10);
//anim.SetConstantPosition (station3, 5, 10);
//anim.SetConstantPosition (station4, 10, 10);
//anim.SetConstantPosition (station5, 15, 10);
//anim.SetConstantPosition (AP, 10, 5);
anim.EnablePacketMetadata (true);
anim.UpdateNodeDescription (0, "Sta1");  // Změna názvu uzlu
anim.UpdateNodeDescription (1, "Sta2");
anim.UpdateNodeDescription (2, "Sta3");
anim.UpdateNodeDescription (3, "Sta4");
anim.UpdateNodeDescription (4, "Sta5");
anim.UpdateNodeDescription (5, "AP");
anim.UpdateNodeColor (AP, 0, 0, 255);  // Změna barvy uzlu
```

Na závěr je nutné zadat globální funkci, která po ukončení simulace má na starosti vymazání všech vytvořených objektů.

```
Simulator::Destroy ();  // Vymazání objektů

return 0;
}
```

Před spuštěním simulace je třeba nahradit soubory *onoff-application.cc* a *onoff-application.h* nacházející se ve složce *model* (*workspace/ns-3.21/src/applications/model*) za poskytnuté soubory. Stejně tak je potřebné nahradit soubor *wscript* ve složce *applications* (*workspace/ns-3.21/src/applications*) za poskytnutý soubor. Je to z důvodu přidání atributu „AccessClass“ do On / Off aplikace (zdrojového souboru) a s tím souvisejících změn.

Podmínkou spuštění simulace je změna pracovního adresáře, vložení souboru *uloha4.cc* do adresáře *scratch* a kompilace vytvořeného skriptu.

```
$ cd wokrspace/ns-3.21
$ ./waf
$ ./waf --run scratch/uloha4
```

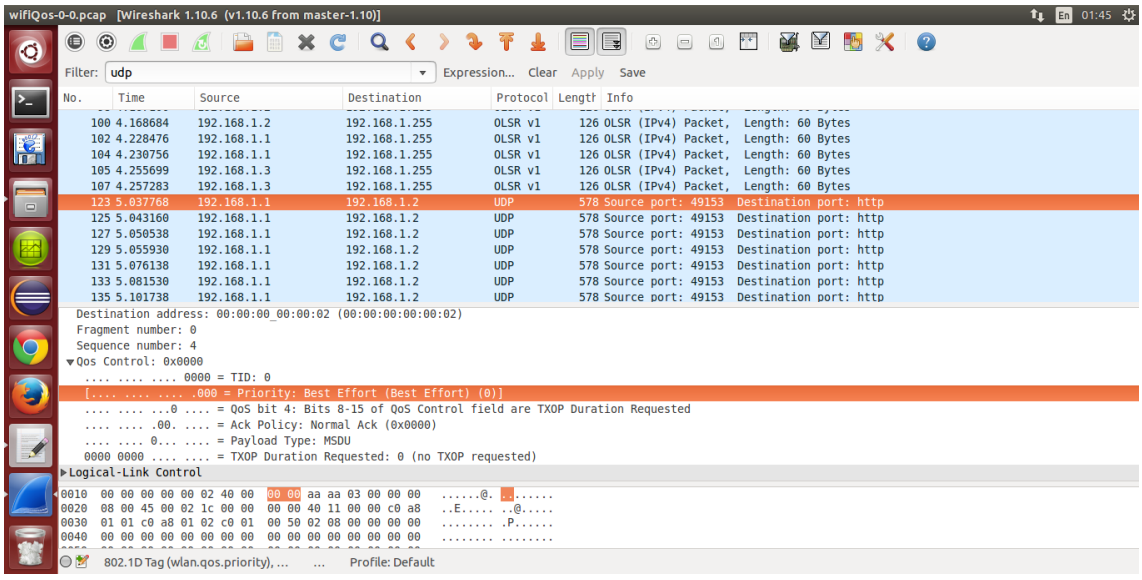
V terminálovém okně (obr. 7.2) vidíme jeden datový tok (Flow ID 1) On / Off aplikace využívající protokol UDP. Poznáme to podle uvedené zdrojové (Src 192.168.1.1) a cílové adresy (Dst 192.168.1.2). Jedná se tedy o datový tok z uzlu station1 do uzlu station2. Této aplikaci jsme nastavili prioritu, tedy přístupovou kategorii Best Effort (AC_BE = 0). Tato priorita je výchozí nastavení. Dále pozorujeme v terminálovém okně počet odeslaných (Tx), přijatých (Rx) paketů, střední dobu zpoždění (Mean Delay) a střední dobu kolísaní zpoždění (Mean Jitter).

Výstupem simulace je trasovací soubor formátu *pcap* (wifiQos-0-0.pcap), soubor *wifiQos.routes* obsahující směrovací tabulky všech uzlů a soubor *wifiQos.xml*.

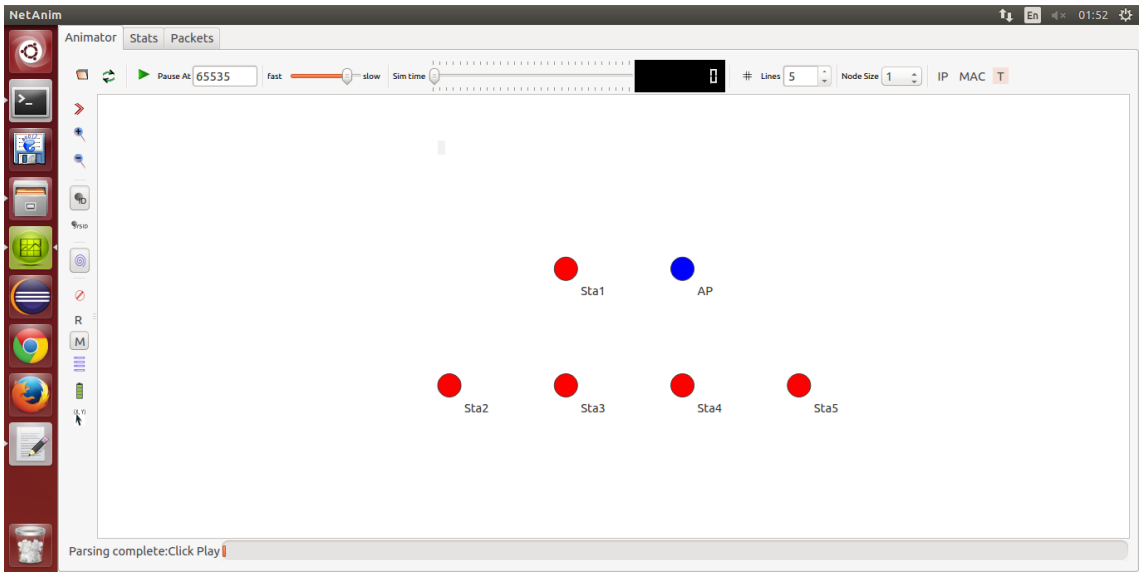
```
student@fekt:~/workspace/ns-3.21$ ./waf --run scratch/uloha4
Waf: Entering directory `/home/student/ns-allinone-3.21/ns-3.21/build'
Waf: Leaving directory `/home/student/ns-allinone-3.21/ns-3.21/build'
'build' finished successfully (4.407s)

 Flow ID: 1 (Src Addr: 192.168.1.1 -> Dst Addr: 192.168.1.2)
  Tx Packets: 781
  Rx Packets: 781
  Mean Delay: 5.42316 ms
  Mean Jitter: 0.0458492 ms
  Throughput: 164.961 Kbps
student@fekt:~/workspace/ns-3.21$
```

Obr. 7.2: Výpis v terminálu pro On / Off aplikaci využívající protokol UDP.

Prohlédnutím trasovacího souboru *wifiQos-0-0.pcap* (obr. 7.3), v programu Wireshark, vidíme průběh komunikace. Pokud chceme vidět komunikaci přehledněji je vhodné vyfiltrovat UDP protokol, zadáním „udp“ (bez uvozovek) do pole Filter a potvrzení tlačítkem Apply. Zajímají nás rámce aplikace využívající protokol UDP, konkrétněji pole „QoS Control“ v záložce „IEEE 802.11 QoS Data“. Pozorujeme nastavenou hodnotu priority (přístupové kategorie) Best Effort. Po otevření souboru *wifiQos.xml* (obr. 7.4) v NetAnime můžeme sledovat, jak probíhá komunikace mezi jednotlivými uzly a od 5 sekundy vidíme i zasílaní dat, námi definovanou aplikací z uzlu station1 (Sta1) na uzel station2 (Sta2).



Obr. 7.3: Zobrazení komunikace na prvním uzlu pro On/Off aplikaci využívající protokol UDP.



Obr. 7.4: Zobrazení simulace v programu NetAnim pro On/Off aplikaci využívající protokol UDP.

7.4 Kontrolní otázky

1. Jakým protokolům odpovídají námi definované čísla portu u jednotlivých aplikací (80, 5004, 5060, 20)? Dohledejte ve Wiresharku pro soubor *wifiQos-0-0.pcap*.
2. Jaká je souvislost těchto protokolů (k čemu slouží) s kvalitou služeb QoS?
3. Co je podstatou QoS v síti? (Jaké jsou požadavky na VoIP, video stream?
4. Proč má On / Off aplikace používající protokol TCP obousměrný datový tok?

8 ZÁVER

Diplomová práca sa zaoberá teóriou z oblasti sietí a návrhom štyroch laboratórnych úloh vzťahujúcich sa k tejto problematike. Z teórie sa konkrétne jedná o popis unicastu, IP adresovania, kde je vysvetlená IP adresa a maska siete. Ďalej sú uvedené triedy sieťových adries a špeciálne IP adresy. Nasledujúca časť je zameraná na smerovanie. Objasnený je rozdiel medzi statickým a dynamickým smerovaním, popísaná smerovacia tabuľka a vysvetlené samotné smerovanie na sieťovej vrstve. Keďže praktická časť práce je vytvorená v sieťovom simulačnom nástroji, tak nasleduje popis tohto simulátora. Uvedená je základná charakteristika, súčasti, hlavné abstrakcie použité pri simulácii a zdroje, z ktorých bolo čerpané pri tvorbe úloh.

Po oboznámení sa so simulátorom nasleduje realizácia praktickej časti. Použitý je sieťový simulátor ns-3, v ktorom sú vytvorené štyri laboratórne úlohy. Úloha číslo jedna je zameraná na unicastové statické smerovanie. Zostavená je topológia podľa vypracovania, doplnená o statické smerovanie a jednotlivé cesty. Pridané sú ďalej aplikácie zodpovedné za generovanie dátového toku. Výstupom zo simulácie sú sledovacie súbory obsahujúce záznamy komunikácie, smerovacia tabuľka a graf. Samotná simulácia je zobrazená vo vizualizačnom nástroji NetAnim. Overená je úprava smerovania pomocou nastavenia metriky. Výstupná komunikácia je podrobne analyzovaná v programe Wireshark.

Úloha číslo dva porovnáva protokoly sieťovej vrstvy ICMPv4 a ICMPv6. Podstatou úlohy je oboznámenie sa s informačnými správami Echo Request (Požiadavka na odozvu), Echo Reply (Odozva na požiadavku) a chybovými správami Destination Unreachable (Cieľ nedostupný), Time Exceeded (Čas prekročený). Na základe vypracovania je zostavená funkčná topológia s aplikáciami ping, UDP Echo Server a UDP Echo Client. Informačné ICMP správy Echo Request a Echo Reply sme obdržali u aplikácie ping. Pre dosiahnutie chybových ICMP správ sme vykonali zmeny v aplikáciách (zmena čísla portu a zmena adresy). Postup sme replikovali aj u druhého súboru zameraného na protokol ICMPv6. Na záver úlohy sme porovnali označenia jednotlivých ICMP správ.

Tretia úloha sa sústreďuje na protokoly ICMPv6 a NDP, ktoré tvoria nevyhnutnú súčasť IPv6. Úloha pozostáva z dvoch samostatných súborov. Podľa pokynov vo vypracovaní je zostavená topológia a overuje sa funkcia správ ICMPv6. Konkrétne sa jedná o správy typu Redirect, Echo Request a Echo Reply. Potom je vytvorená ďalšia topológia, na ktorej sa overuje funkcia správ Neighbor Advertisement, Neighbor Solicitation, Router Advertisement a Router Solicitation pri bezstavovej automatickej konfigurácii adries. Program Wireshark je použitý na rozbor zaznamenanej komunikácie. Dôležitú úlohu zohrali aj výstupy smerovacích tabuliek pri názornom vysvetlení fungovania jednotlivých správ.

Posledná úloha sa venuje zabezpečeniu kvality služieb (QoS) v bezdrôtovej sieti. Zostavená je topológia podľa vypracovania s jednou On / Off aplikáciou využívajúcou protokol UDP. Dopracovaním ďalších troch aplikácií (dve využívajúce protokol UDP a jedna využívajúca protokol TCP) sme dostali dostatočnú vzorku. Nastavením rôznej priority v každej aplikácii, na základe čísla portu definujúceho špecifický protokol došlo k rozdeleniu dátových tokov do jednotlivých tried. V terminálovom okne sme porovnávali hodnoty parametrov (priepustnosť, oneskorenie a kolísanie oneskorenia) a na ich základe sme potvrdili rozdielne zaobchádzanie s odlišnými triedami, čo je podstatou fungovania QoS.

LITERATÚRA

[1] ADIBI, Sasan. *Quality of Service Architectures for Wireless Networks: Performance Metrics and Management* [online]. 2010, [cit. 2015-03-24]. ISBN 16-152-0681-7. Dostupné z: <https://books.google.cz/books?id=YUyswy1Q19kC&pg=PA599&dq=wifi+qos&hl=en&sa=X&ei=Qe9jVdeqKoyP7Ab2_oPYAg&ved=0CEMQ6AEwBA#v=snippet&q=edca&f=false>.

[2] BOUŠKA, Petr. TCP/IP: adresy, masky, subnety a výpočty. *SAMURAJ-cz.com: počítačové sítě, Cisco, Microsoft, VMware, administrace* [online]. 05.09.2007, 11.08.2008 [cit. 2014-11-15]. Dostupné z: <http://www.samuraj-cz.com/clanek/tcpip-adresy-masky-subnety-a-vypocty/>.

[3] BOUŠKA, Petr. TCP/IP a ethernet: cesta v síti, aktivní síťové prvky. *SAMURAJ-cz.com: počítačové sítě, Cisco, Microsoft, VMware, administrace* [online]. 17.09.2007 [cit. 2014-11-18]. Dostupné z: <http://www.samuraj-cz.com/clanek/tcpip-a-ethernet-cesta-v-siti-aktivni-sitove-prvky/>.

[4] BOUŠKA, Petr. TCP/IP: Routing – směrování. *SAMURAJ-cz.com: počítačové sítě, Cisco, Microsoft, VMware, administrace* [online]. 21.09.2007 [cit. 2014-11-16]. Dostupné z: <http://www.samuraj-cz.com/clanek/tcpip-routing-smerovani/>.

[5] FALL, Kevin R a W STEVENS. *TCP/IP Illustrated, Volume 1: The Protocols*. 2nd. Upper Saddle River, NJ: Addison-Wesley, 2011, ISBN 03213363131.

[6] JEŘÁBEK, Jan. *Pokročilé komunikační techniky*. Brno, 2014, 177 s.

[7] *Neighbor Discovery for IP version 6 (IPv6)* [online]. 2007 [cit. 2014-04-03]. Dostupné z: <https://tools.ietf.org/html/rfc4861>.

[8] Ns-3 Manual: Release ns-3.21. *Ns-3 Manual* [online]. 2014 [cit. 2014-11-06]. Dostupné z: <http://www.nsnam.org/docs/release/3.21/manual/ns-3-manual.pdf>.

[9] Ns-3 Model Library: Release ns-3.21. *Ns-3 Model Library* [online]. 2014 [cit. 2014-11-09]. Dostupné z: <http://www.nsnam.org/docs/release/3.21/models/ns-3-model-library.pdf>.

[10] Ns-3 Tutorial: Release ns-3.21. *Ns-3 Tutorial* [online]. 2014 [cit. 2014-11-04]. Dostupné z: <http://www.nsnam.org/docs/release/3.21/tutorial/ns-3-tutorial.pdf>.

[11] Princip směrovací tabulky IP. *Resources and Tools for IT Professionals | TechNet* [online]. © 2014 [cit. 2014-11-19]. Dostupné z: <http://technet.microsoft.com/cs-cz/library/cc787509(v=ws.10).aspx>.

[12] Vytváření podsítí s proměnnou maskou v prostředí IPv4. *Access server* [online]. 2009 [cit. 2014-11-21]. Dostupné z: <http://access.feld.cvut.cz/view.php?cisloclanku=2009030001>.

[13] What is ns-3. *Ns-3* [online]. © 2011-2014 [cit. 2014-11-12]. Dostupné z: <http://www.nsnam.org/overview/what-is-ns-3/>.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

AC prístupová kategória určujúca prioritu – Access Category

API rozhranie pre programovanie aplikácií – Application Programming Interface

ARP protokol prekladu sieťovej adresy na fyzickú – Address Resolution Protocol

ASCII kódovací systém znakov anglickej abecedy – American Standard Code for Information Interchange

CSMA/CA metody viacnásobného prístupu s počúvaním nosnej – Carrier Sense Multiple Access with Collision Avoidance

DFC zdieľanie média s predchádzaním kolízie – Distributed Coordination Function

DHCP protokol automatickej konfigurácie pripojených staníc – Dynamic Host Configuration Protocol

DSSS technika priameho rozprestreného spektra – Direct Sequence Spread Spectrum

EDCA metóda prioritizovanej kvality služieb – Enhanced Distributed Channel Access

FHSS technika frekvenčného skákania – Frequency Hopping Spread Spectrum

GNU slobodný operačný systém – GNU's Not Unix

GPLv2 všeobecná verejná licencia verzie 2 – General Public License version 2

HCCA metóda garantujúca kvalitu služieb vo WLAN – HCF Controlled Channel Access

HCF hybridná metóda prístupu k médiu – Hybrid Coordination Function

IANA organizácia dohliadajúca na prideľovanie IP adries – Internet Assigned Numbers Authority

ICMPv6 protokol používaný k prenosu sieťových informácií – Internet Control Message Protocol version 6

IEEE organizácia vyvíjajúca štandardy v elektronike – Institute of Electrical and Electronics Engineers

IP internetový protokol – Internet Protocol

IPv4 internetový protokol verzie 4 – Internet Protocol version 4

IPv6 internetový protokol verzie 6 – Internet Protocol version 6

LTS dlhodobá podpora – Long Term Support

MAC fyzická adresa sieťového zariadenia – Media Access Control Address

MAC podvrstva linkovej vrstvy – Media Access Control Layer

MIMO model pre multi-anténny komunikačný systém – Multiple-input Multiple-output

NAT protokol prekladu adries – Network Address Translation

NDP protokol pre objavovanie susedov – Neighbor Discovery Protocol

NIC karta sieťového rozhrania – Network Interface Card

ns-2 sieťový simulátor verzie 2 – network simulator version 2

ns-3 sieťový simulátor verzie 3 – network simulator version 3

OFDM ortogonálny multiplex s frekvenčným delením – Orthogonal Frequency Division Multiplexing

OLSR smerovací protokol pre mobilné siete – Optimized Link State Routing Protocol

OS operačný systém – Operating System

OSI referenčný komunikačný model rozdeľujúci komunikáciu do vrstiev – Open Systems Interconnection model

OSPF adaptívny hierarchický distribuovaný smerovací protokol – Open Shortest Path First

OTcl objektovo orientovaný skriptovací jazyk – object oriented Tool Command Language

PCAP trasovacie pakety – Packet Capture

PPP komunikačný protokol linkovej vrstvy – Point–to–Point Protocol

QoS garancia kvality služby – Quality of Service

RA správa oznamujúca IP adresu rozhrania – Router Advertisement

Radvd daemon pre ohlasovanie smerovača – Router Advertisement Daemon

RFC dokumenty vzťahujúci sa k internetu – Request for Comments

RS správa k lokalizácii smerovačov na linke – Router Solicitation

RTS mechanizmus redukujúci kolízie rámcov u bezdrôtových sietí – Request to Send

SSID identifikátor bezdrôtovej siete – Service Set Identifier

TCP spojovo orientovaný transportný protokol – Transmission Control Protocol

TTL maximálny počet skokov – Time To Live

UDP nespojovo orientovaný transportný protokol – User Datagram Protocol

VLSM technika podsieťovania – Variable Length Subnet Masking

Wifi bezdrôtová sieťová technológia – Local Area Wireless Computer Networking

XML rozšíriteľný značkovací jazyk – Extensible Markup Language

ZOZNAM PRÍLOH

A Príloha **102**

A.1 Obsah CD 102

A.1.1 Elektronický text diplomovej práce xstefa11.pdf 102

A.1.2 uloha1 102

A.1.3 uloha2 102

A.1.4 uloha3 102

A.1.5 uloha4 102

A PRÍLOHA

A.1 Obsah CD

A.1.1 Elektronický text diplomovej práce xstefa11.pdf

A.1.2 uloha1

- Zdrojový súbor kompletný `uloha1.cc`
- Zdrojový súbor pre študentov `uloha1studenti.cc`
- `Unicast a statické směrování.docx`
- `Kontrolní otázky Laboratorní úloha číslo 1.docx`

A.1.3 uloha2

- Zdrojový súbor pre prvú časť `uloha2a.cc`
- Zdrojový súbor pre druhú časť `uloha2b.cc`
- `Porovnání ICMPv4 a ICMPv6.docx`
- `Kontrolní otázky Laboratorní úloha číslo 2.docx`

A.1.4 uloha3

- Zdrojový súbor pre prvú časť `uloha3a.cc`
- Zdrojový súbor pre druhú časť `uloha3b.cc`
- Zdrojový súbor pre druhú časť `uloha3bKontrolniOtazka5.cc`
- `Protokol ICMPv6.docx`
- `Kontrolní otázky Laboratorní úloha číslo 3.docx`

A.1.5 uloha4

- Zdrojový súbor `uloha4.cc`
- Zdrojový súbor On / Off aplikácie `onoff-application.cc`
- Hlavičkový súbor On / Off aplikácie `onoff-application.h`
- Súbor `wscript`
- `Wifi s podporou QoS.docx`
- `Kontrolní otázky Laboratorní úloha číslo 4.docx`