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

LABORATORNÍ ÚLOHY V PROSTŘEDÍ NS3 PRO PŘEDMĚT POKROČILÉ KOMUNIKAČNÍ TECHNIKY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

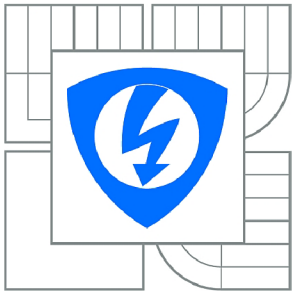
AUTOR PRÁCE
AUTHOR

Bc. GABRIEL VADKERTI

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

LABORATORNÍ ÚLOHY V PROSTŘEDÍ NS3 PRO PŘEDMĚT POKROČILÉ KOMUNIKAČNÍ TECHNIKY

LABORATORY EXERCISES IN NS3 ENVIROMENT FOR ADVANCED COMMUNICATION TECHNOLOGY COURSE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

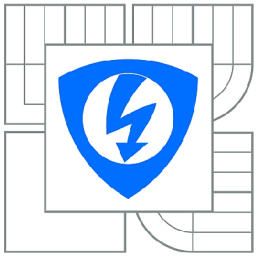
AUTOR PRÁCE — Bc. GABRIEL VADKERTI

AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ LANGHAMMER

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. Gabriel Vadkerti	***ID:***	136594
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Laboratorní úlohy v prostředí NS3 pro předmět Pokročilé komunikační techniky

POKYNY PRO VYPRACOVÁNÍ:

Náplní diplomové práce je návrh čtyř nových laboratorních úloh v simulačním prostředí NS3 do předmětu Pokročilé komunikační techniky. Prostudujte možnosti simulačního prostředí NS3 a na základě možností tohoto prostředí vytvořte kompletní návody vhodné pro studentyvčetně kontrolních otázek či doplňujících úkolů. Při návrhu úloh se zaměřte na okruhy: srovnání IPv4 a IPv6, porovnání unicastu a multicastu, zajištění kvality služeb v síti, či rozbor aplikačního protokolu (jako ICMPv6, DNS, popřípadě další).Časová náročnost každé úlohy musí být přibližně dvě hodiny.

DOPORUČENÁ LITERATURA:

[1] FOROUZAN, Behrouz A. TCP/IP protocolsuite. 4th ed. Boston: McGraw-HillHigherEducation, 2010, xxxv, 979 s. ISBN 978-0-07-337604-2.
[2] NS3 Consortium: Dokumentace k Network Simulator 3 (NS3), 2014. Dostupné online <http://www.nsnam.org/documentation/>, citováno 17.9.2014.
[3] 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. Lukáš Langhammer
Konzultanti diplomové práce:

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

UPOZORNĚNÍ:

ABSTRAKT

Táto diplomová práca sa zaoberá s vytvorením laboratórnych úloh v prostredí NS-3, kde sú predstavené protokoly IPv4 a IPv6, unicastový a multicastový prenos dát, kvality služieb v bezdrôtových sieťach, ďalej protokoly TCP a UDP. Najprv je popísaný simulátor NS-3, vývojové prostredie Eclipse IDE a ich spojenie. V ďalšej časti sú predstavené vlastnosti protokolov IPv4 a IPv6, je podrobne popísaná štruktúra záhlaví a technika fragmentácie. Podľa získaných vedomostí je vytvorený postup laboratórnej úlohy, ktorá sa zaoberá s porovnaním protokolov IPv4 a IPv6. V ďalšej časti sú popísané typy prenosov, ďalej je podrobnejšie vysvetlený multicastový prenos dát. V druhej praktickej časti je vytvorený postup pre laboratórnu úlohu, ktorá sa zaoberá s porovnaním unicastového a multicastového prenosu. V ďalšej teoretickej časti je rozobraná problematika kvality služieb a metódy na jej zaistenie. Podrobnejšie je popísaná prístupová metóda EDCA, ktorá sa používa v bezdrôtových sieťach. Táto prístupová metóda je implementovaná v nasledujúcej simulácii, v ktorej je skúmaný vplyv prístupovej kategórie na rôzne parametre prenosu. V ďalšej teoretickej časti sú predstavené protokoly UDP a TCP, ďalej štruktúra záhlaví týchto protokolov. Posledná časť práce sa zaoberá s porovnaním vlastností týchto protokolov.

KĽÚČOVÉ SLOVÁ

NS-3, Eclipse, IPv4, IPv6, Gnuplot, unicast, multicast, NetAnim, QoS, EDCA, NS-3-Generator UDP, TCP

ABSTRACT

This master thesis deals with creating laboratory exercises for students in simulator NS-3, where the IPv4 and IPv6 protocols, unicast and multicast transmission types, quality of services in wireless networks, furthermore protocols TCP and UDP are analyzed. The first section describes the NS-3 network simulator, the Eclipse development environment and the method how to connect them. In the next section protocols IPv4 and IPv6 are presented, the structure of the headers and the technique of fragmentation are described in detail. Based on this a laboratory exercise is created which deals with comparison of protocols IPv4 and IPv6. In the next section the types of transmissions are described, furthermore the multicast transmission type is explained more in detail. In the next practical section the second laboratory exercise is created, which deals with comparison of unicast and multicast transmission types. The subject of the following section is ensuring quality of services in data networks, the EDCA access method is discussed more in detail. In the next section the EDCA access method is implemented in a simulation as a laboratory exercise, in which the impact of the access category of EDCA to different transmission parameters is investigated. The next section deals with protocols TCP and UDP, furthermore with the structure of their headers. In the last section a laboratory exercise is created, in which the properties of the protocols TCP and UDP are compared.

KEYWORDS

NS-3, Eclipse, IPv4, IPv6, Gnuplot, unicast, multicast, NetAnim, QoS, EDCA, NS-3-Generator UDP, TCP,

Bibliografická citace:

VADKERTI, G. *Laboratorní úlohy v prostředí NS3 pro předmět Pokročilé komunikační techniky.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 118 s. Vedoucí diplomové práce Ing. Lukáš Langhammer.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Laboratorní úlohy v prostředí NS3 pro předmět Pokročilé komunikační techniky jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brne dne . . . .
podpis autora

POĎAKOVANIE

Ďakujem vedúcemu diplomovej práce, pánovi Ing. Lukášovi Langhammerovi, za účinnú metodickú, pedagogickú a odbornú pomoc a ďalšie cenné rady pri spracovaní mojej práce.

V Brne dne . . . .
podpis autora

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

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

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

Úvod

V dnešnej dobe Internet a jeho používanie je pre každého samozrejmosťou. V posledných rokoch sa ale veľkosť sietí a počet sieťových zariadení veľmi rýchlo rástol a predpokladá sa, že nárast v budúcnosti bude ešte rýchlejší.

Jeden z kľúčových protokolov sieťových technológií je IPv4, ktorý s týmto trendom sa už nevyrovná. Protokol IPv4 zabezpečuje globálne adresovanie na Internete, ale kvôli určitým nedostatkom sa predpokladá, že v budúcnosti bude úplne nahradená protokolom IPv6. Ďalšou veľmi rozšírenou technikou Internetu je streamovanie audia alebo videa, ktorá by s unicastovým (bežným) prenosom nemohol fungovať. Pre tento účel sa používa multicastový prenos, ktorý umožní, aby zdroj dát posielal správu vždy len raz, ktorá sa s postupným duplikovaním dostane ku každému príjemcovi, ktorí chcú daný obsah prijímať. V poslednej dobe sa prenos multimediálnych dát v reálnom čase na Internete stal samozrejmosťou, ale tieto nové služby majú iné požiadavky na prenos. Preto bolo treba stávajúce systémy k novým požiadavkám prispôsobiť. Pojem zaistenie kvality služieb označuje techniku odlišného zachádzania s dátovými jednotkami, pri čom sú zohľadnené dostupné sieťové zdroje a požiadavky rôznych služieb. S touto problematikou súvisí aj voľba transportného protokolu, medzi nich patria TCP a UDP. Protokol TCP zaistí spojovo orientovaný a garantovaný prenos, na druhej strane UDP prenáša dáta nespojovo, ale s menším oneskorením.

Táto diplomová práca je určená študentom prvého ročníka magisterského stupňa univerzity VUT (Vysoké učení technické v Brně) a má za účel objasniť fungovanie vyššie zmienených technológií. Hlavným cieľom bol vytvoriť štyri laboratórne úlohy na predmet MPKT (Pokročilé komunikační techniky), v ktorých budú porovnané protokoly IPv4 a IPv6, unicastový a multicastový prenos dát, protokoly TCP a UDP, ďalej objasniť techniku zaistení kvality služieb. K tomu je použitý program NS-3, ktorý je nástroj pre simuláciu sietí založený na programovacom jazyku C++.

V prvej kapitole je predstavený simulátor NS-3 a jeho štruktúra. NS-3 nemá grafické užívateľské rozhranie, preto pre písanie zdrojového kódu je použitý program Eclipse IDE. V ďalšej časti je uvedený postup spojenia NS-3 s Eclipsom, aby sme mohli využívať všetky jeho funkcie, ako napr. automatické dopĺňanie kódu.

V ďalšej kapitole sú predstavené vlastnosti protokolov IPv4 a IPv6, je podrobne popísaná štruktúra záhlaví a technika fragmentácie. Sú ďalej vystihnuté rozdiely medzi zmienenými protokolmi, výhody a nevýhody protokolu IPv6.

V prvej praktickej časti je uvedený postup pre prvú laboratórnu úlohu, ktorá sa zaoberá s protokolmi IPv4 a IPv6. Ich fungovanie je predstavené pomocou jednoduchej siete, kde je definovaný súčasný beh týchto protokolov. Na konci úlohy sú výsledky analyzované programom Wireshark. Samostatná úloha sa bližšie zaoberá s fragmentáciou paketov v prípade protokolu IPv4 a IPv6.

Ďalšia časť práce sa zaoberá s typmi prenosu a je podrobnejšie popísaný multicastový prenos. Sú uvedené jeho výhody oproti unicastového prenosu a podmienky, kedy sa tieto výhody

uplatňujú. Je ďalej vysvetlená technika vytvorenia multicastovej MAC adresy z multicastovej IP adresy, čo sú to distribučné stromy a aké typy poznáme. Sú predstavené aj protokoly, ktoré sa používajú v prípade multicastového prenosu medzi smerovačmi a medzi smerovačom a koncovou stanicou.

V druhej praktickej časti je uvedený postup pre vytvorenie simulácie, v ktorom sa porovnáva unicastový a multicastový prenos. Výsledky s unicastovým prenosom sú porovnané s výsledkami simulácie, v ktorej sa používa multicastový prenos. Topológia a prenos paketov sú znázornené aj graficky pomocou programu NetAnim. Samostatná úloha sa zaoberá s unicastovým prenosom v prípade, že prenosová rýchlosť linky je menšia než bitová rýchlosť generovaných dát.

V nasledujúcej kapitole je predstavená problematika zaistení kvality služieb, ďalej sú popísané jednotlivé techniky, ktoré sa používajú pri jej fungovaní. Hlbšie je rozobraná prístupová metóda EDCA, ktorá sa používa na zaistenie kvality služieb na linkovej vrstve v bezdrôtových sieťach.

V ďalšej praktickej časti je vytvorený postup pre simuláciu jednoduchej bezdrôtovej siete, kde je implementovaná prístupová metóda EDCA. Výstupom simulácie sú trasovacie súbory, pomocou ktorých sú preštudované položky záhlavia linkovej vrstvy, ktoré majú za úlohu zaistiť kvalitu služieb. Ďalším výstupom sú výpisy nameraných parametrov prenosov do konzolového okna, ako stratovosť, oneskorenie a kolísanie oneskorenia. V rámci samostatnej úlohy je skúmaný vzájomný vplyv prenosov rôznych služieb.

Ďalšia teoretická časť sa zaoberá s protokolmi transportnej vrstvy. Podrobnejšie sú predstavené protokoly TCP a UDP, ďalej sú uvedené a vysvetlené položky ich záhlaví. Na konci tejto časti je uvedená tabuľka s porovnaním transportných protokolov, ktorá obsahuje okrem TCP a UDP aj iné, nie až tak rozšírené protokoly.

V poslednej kapitole práce je uvedený postup vytvorenia simulácie pre porovnanie protokolov TCP a UDP. Výsledky simulácie sú analyzované pomocou programu Wireshark, je predstavený princíp vytvorenia a zrušenia spojenia v prípade protokolu TCP. Simulácia je zobrazená aj graficky pomocou programu NetAnim. Doplňujúce úlohy sa zaoberajú s chovaním protokolov TCP a UDP na krátkodobý výpadok a na zahltenie siete.

1 NETWORK SIMULATOR 3

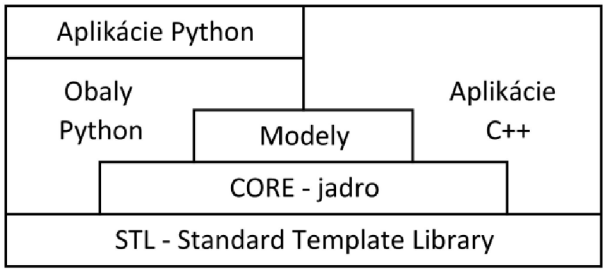
NS-3 (Network Simulator 3) je simulátor s diskrétnymi udalosťami (discrete event simulator) vyvinutý primárne na výskumné a vzdelávacie účely. Je licencovaný projektom GNU GPLv2 a je voľne dostupný k stiahnutiu a použitiu. Na prvej verzii sa začalo pracovať v roku 1995 v LBNL (Lawrence Berkeley National Laboratory), základom bol simulátor REAL. Vydanie prvej verzie NS-3 sa uskutočnilo v roku 2008. Nové stabilné verzie vychádzajú pravidelne, v tejto práci bola použitá verzia 3.21 [1][2].

1.1 Štruktúra NS-3

Softwarová infraštruktúra NS-3 podporuje vývoj simulačných modelov, ktoré sú dosť realistické, aby bolo možné ich použiť ako emulátor dátových sietí v reálnom čase. Umožňuje teda implementáciu veľa protokolov, ktoré existujú aj v reálnom svete a podporuje aj simulácie sietí, ktoré nie sú založené na protokole IP (Internet Protocol) [1][2].

NS-3 je v podstate C++ knižnica, ktorá poskytuje sadu modelov pre sieťové simulácie, realizované ako C++ objekty zabalené prostredníctvom skriptovacieho jazyka Python (obr. 1-1). Užívatelia používajú túto knižnicu, aby vytvorili aplikáciu v jazyku C++ alebo Python, ktorá po spustení vytvorí simulačné modely, nastaví scenár, vstúpi do hlavnej funkcie *main* a vystúpi po ukončení simulácie [1][2]. Výhody NS-3 oproti ostatným simulátorom:

- používa programovacie jazyky C++ a Python, čo umožňuje využiť všetky možnosti týchto jazykov,
- udalosti sú volaním funkcií a môžeme presne naplánovať, kedy sa majú vykonať,
- je možné detailne konfigurovať komunikáciu na fyzickej a spojovej vrstve,
- NS-3 pakety sú uložené vo vnútornom zásobníku podobne, ako reálne pakety v operačnom systéme, čo umožňuje komunikáciu s reálnym sieťovým rozhraním,
- používa tzv. *helepry*, ktoré poskytujú ľahšie používateľné funkcie a rozumné predvolené správanie objektov [1][2][3] – sú popísané v dokumentácii Doxygen [4].



Obr. 1-1 Štruktúra simulátora NS-3 [1]

1.2 Vývojové prostredie Eclipse IDE

NS-3 je v súčasnosti vyvíjaný pre platformy založené na OS Linux, inštalácia simulátora je podrobne popísaná na oficiálnej internetovej stránke [5]. Po inštalácii však nie je k dispozícii

žiadne vývojové prostredie, ako zvyčajne u iných programovacích jazykoch. Pre písanie zdrojového kódu máme možnosť vybrať z rôznych voľne šíriteľných textových editorov, ktoré však neponúkajú žiadnu výhodnú funkciu pre programátora. Preto v tejto práci bolo použité vývojové prostredie Eclipse (obr. 1-2), ktoré nie je natívne podporované, ale v posledných rokoch dostalo väčšiu pozornosť medzi vývojármi NS-3. Výhody vývojového prostredia Eclipse:

- podpora väčšiny programovacích jazykov, ako napr. Java, C++, C# a Python,
- spustenie simulácie priamo vo vývojovom prostredí, použitie príkazového riadku je potrebné len vo výnimočných prípadoch,
- obsahuje textovú oblasť, kde dostaneme informácie o priebehu a výsledku simulácie, o chybových správach a varovaniach,
- obsahuje nápovedu, ktorá uľahčí písanie kódu a umožňuje zobraziť napr. dostupné objekty danej triedy [3].

V ďalšej podkapitole budú predstavené najčastejšie používané funkcie prostredia Eclipse.

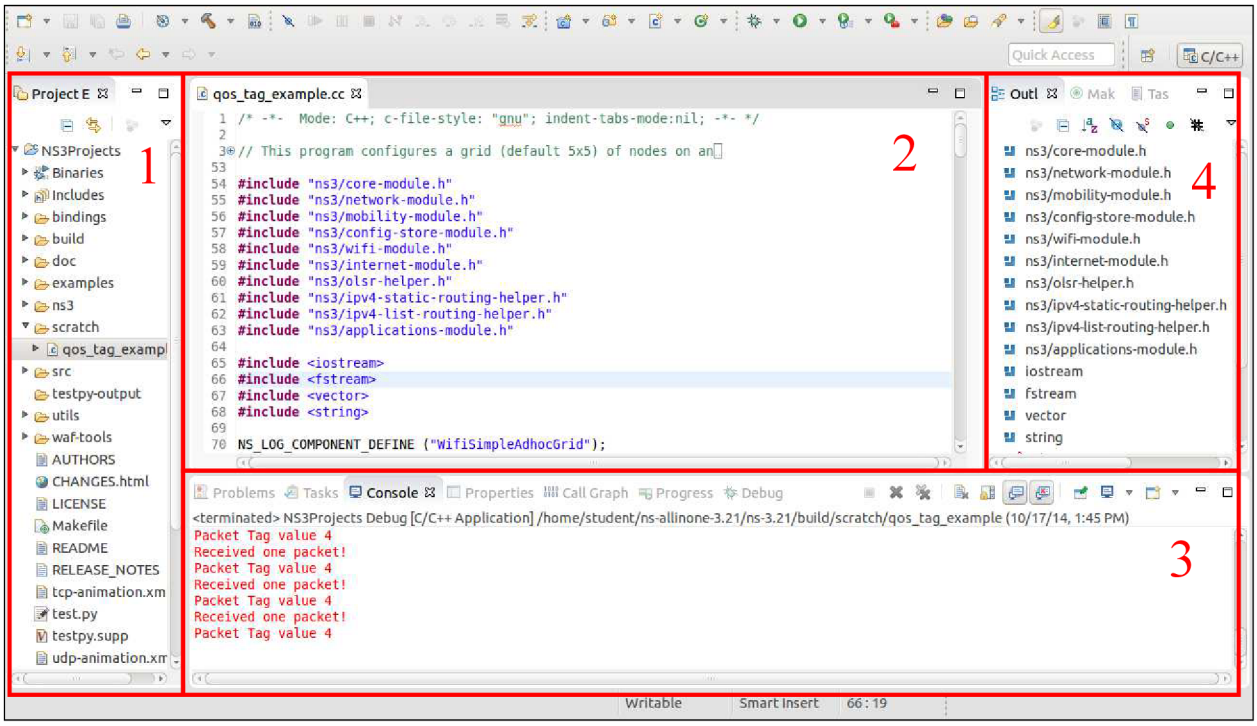
1.3 Popis grafického užívateľského prostredia Eclipse

Grafické rozhranie prostredia Eclipse je rozdelené na menšie okná nasledovne (obr. 1-2):

1. Na ľavej strane hlavného okna nájdeme časť, ktorá sa nazýva *Project Explorer*. Je tu zobrazená stromová štruktúra súborov aktuálneho projektu. V tomto prípade je to nastavené na zložku, kde sa nachádza simulátor NS-3.
2. Stredná časť obrazovky je rozdelená na dve časti. Prvá z nich je textový editor, kde vytvárame zdrojový kód. Na obr. 1-2 je práve otvorený súbor *qos_tag_example.cc*. Veľkou výhodou je, že môžeme otvoriť viac súborov naraz v oddelených oknách, ktoré môžeme umiestniť podľa potreby (napr. vedľa seba).
3. Pod editorom je umiestnená textová oblasť (konzola), kde dostávame chybové správy, varovania, ďalej informácie o stave kompilácie a o výsledku simulácie vo forme jednoduchých výpisov.
4. Na pravej strane obrazovky sa nachádza časť *Outline*, kde môžeme jednoduchým spôsobom kontrolovať, aké moduly a menný priestor (namespace) sme použili v zdrojovom kóde a aké funkcie máme vytvorené. Kliknutím na niektorú položku sa dostaneme na príslušnú časť zdrojového kódu.

Tieto okná je možné podľa potreby premiestniť, zväčšiť, zmenšiť, skryť a môžeme pridať aj ďalšie. Tento popis bol vytvorený na základe základných nastavení.

Väčšina nastavení Eclipsu je sprístupnená z hlavného menu (obr. 1-3). V nasledujúcej podkapitole bude popísané nastavenie projektu a spojenie Eclipsu s NS-3, k tomu budeme potrebovať položky *Project* a *Run* [3]. V podmenu *Run* budú zaujímavé nastavenia spúšťača (Run Configurations) a debuggera (Debug Configurations), ale môžeme odtiaľto aj napr. spustiť simuláciu. V podmenu *Project* sa nachádza položka *Properties*, kde je možné nastaviť náš projekt.



Obr. 1-2 Grafické rozhranie vývojového prostredia Eclipse IDE

Obr. 1-3 Hlavné menu

Pod hlavným menu sa nachádzajú tlačidlá rýchleho prístupu, ktoré je vidno na obr. 1-4. Ich funkcie z ľava sú nasledovné:

- vytvorenie nového projektu,
- vytvorenie novej zdrojovej zložky,
- vytvorenie nového hlavičkového súboru,
- vytvorenie novej triedy,
- spustenie prekladu projektu (simulácie) v ladiacom móde,
- spustenie projektu [3].

Obr. 1-4 Tlačidlá rýchleho prístupu

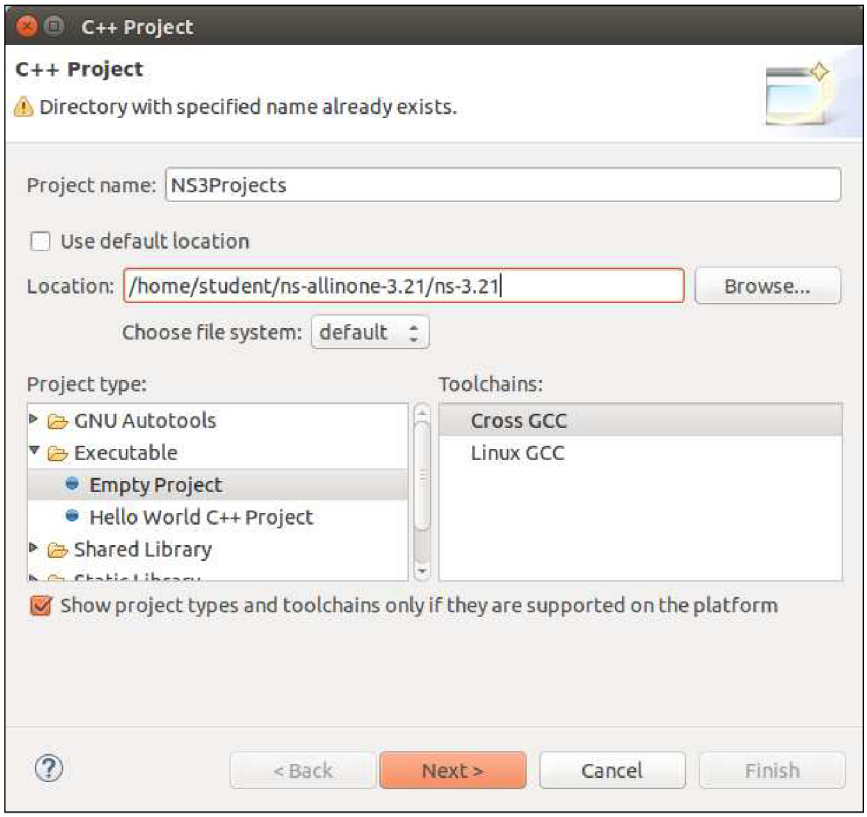
1.4 Eclipse a NS-3

Aby sme mohli využívať vyššie spomínané výhody Eclipsu pri písaní zdrojového kódu, musíme to určitým spôsobom spojiť so simulátorom NS-3. V nasledujúcej časti budú uvedené kroky tejto operácie.

1.4.1 Waf Builder

Prvým krokom je vytvorenie nového projektu a nastavenie w*af buildera*.

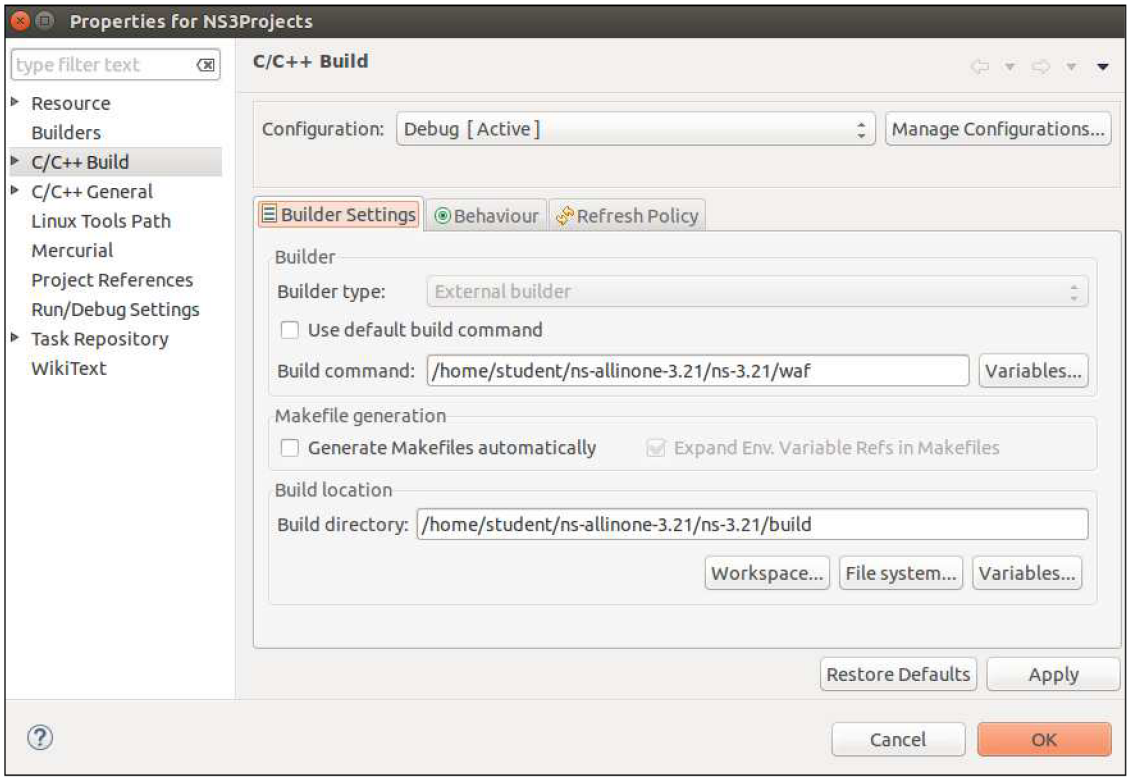
- Z hlavného menu vyberieme *File* → *New* → *C++ Project* a nastavíme meno projektu, napr. *NS3Projects* (obr. 1-5).
- Ďalej vyberieme typ projektu *Empty Project* a z okna *Toolchains* vyberieme položku *Cross GCC*. Vypneme možnosť *Use Default Locations* a nastavíme cestu k projektu. V tomto prípade to je */home/student/ns-allinone-3.21/ns-3.21*, ale môže to byť odlišné podľa toho, kde sa NS-3 nachádza na disku a akú verziu používame. Treba nastaviť cestu tam, kde sa nachádza zložka *Scratch*.
- Tým máme nastavený projekt, tlačítkom *Next* postupujeme ďalej, kým nemôžeme potvrdiť tlačítkom *Finish*. Po ukončení dostaneme varovnú správu, že po pokračovaní sa aktuálne nastavenia stratia, čo potvrdíme.



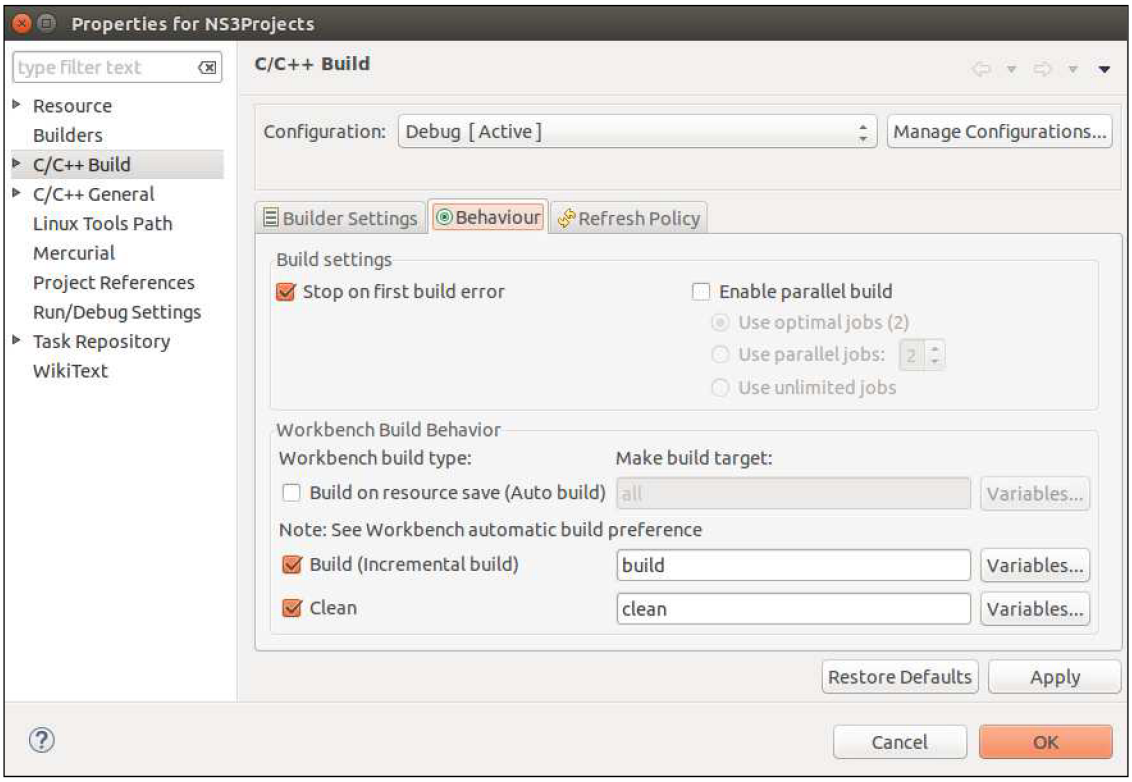
Obr. 1-5 Vytvorenie nového projektu

- Vytvorený projekt sa objavil na ľavej strane obrazovky v časti *Projekt Explorer*. Pravým tlačítkom klikneme na meno projektu, z menu vyberieme položku *Properties* a v otvorenom okne zvolíme *C/C++ Build*. Prvé, čo budeme modifikovať, je *Builder Settings*.
- Vypneme možnosti *Use default build command* a *Generate Makefiles automatically*.
- Do textového poľa *Build command* napíšeme cestu k *waf builderu*: */home/student/ns-allinone-3.21/ns-3.21/waf*, ďalej nastavíme cestu k zložke *build*: */home/student/ns-allinone-3.21/ns-3.21/build* (obr. 1-6).

- V otvorenom okne prepneme na nastavenia *Behaviour*, obsah textového poľa *Build (Incremental build)* vymažeme a namiesto toho napíšeme *build* (obr. 1-7) [6][7].



Obr. 1-6 Builder Settings

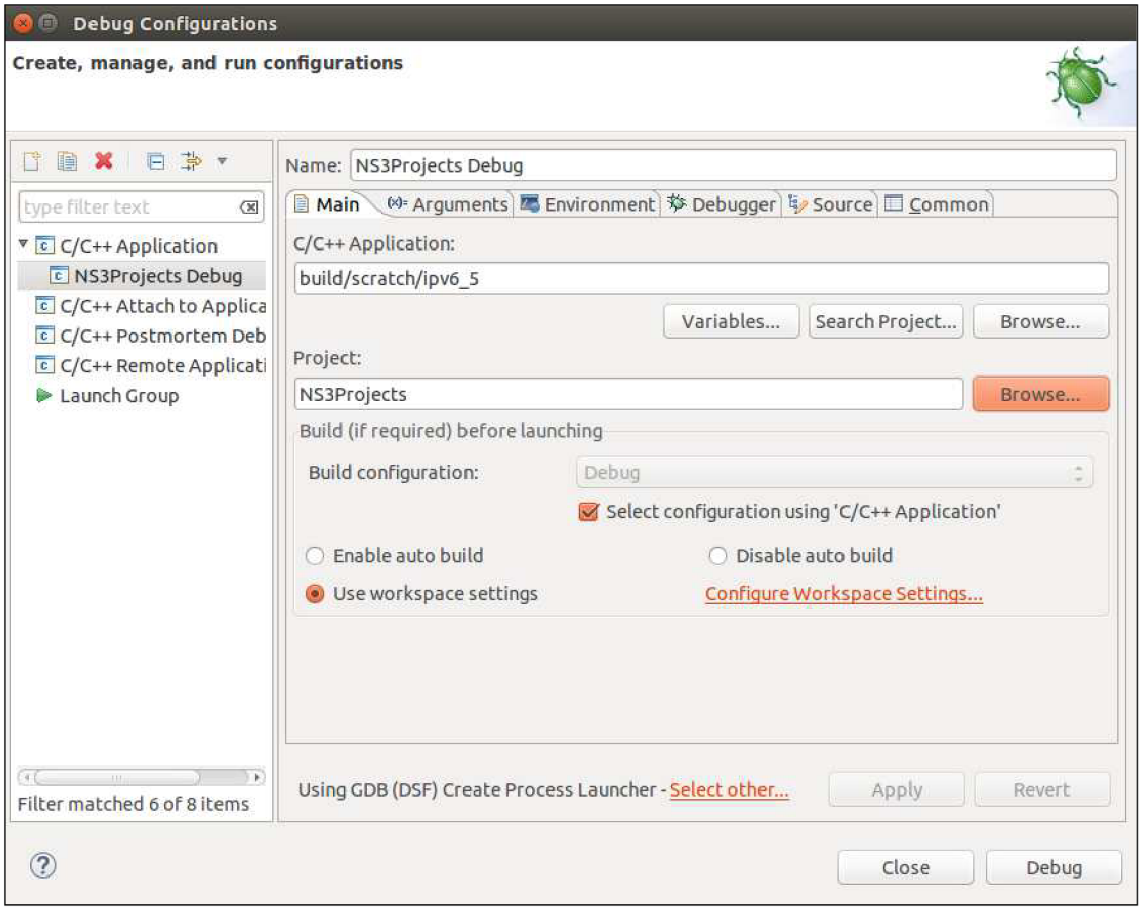


Obr. 1-7 Behaviour Settings

1.4.2 Debugger

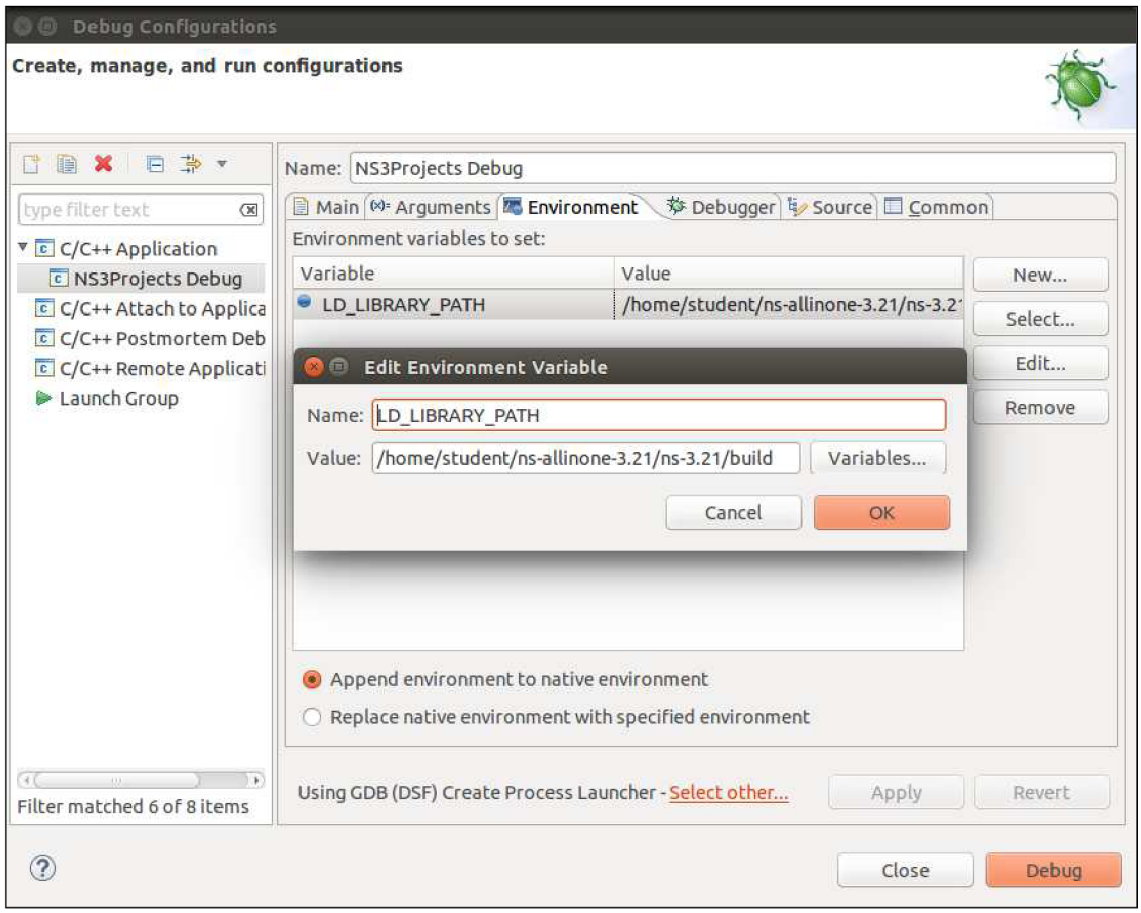
Teraz nasleduje nastavenie *debuggera* a pripojenie C++ súboru k projektu.

- V hlavnom menu vyberieme položku *Run* → *Debug Configurations*. V otvorenom okne vytvoríme novú konfiguráciu a pomenujeme to (obr. 1-8).
- Pomocou tlačítka *Search Project* vyberieme už preložený zdrojový kód, ktorý sa nazýva *scratch-simulator* – tento súbor po spustení vypíše jednoduchú správu do konzoly. Môžeme ale vybrať ľubovoľný súbor, ktorý už bol preložený kompilátorom.
- Pomocou tlačítka *Browse* v sekcii *Project* vyberieme náš projekt.



Obr. 1-8 Debug Configuration

- V otvorenom okne prepneme na nastavenia *Environment* a pridáme novú premennú. Do textového poľa *Name* napíšeme *LD_LIBRARY_PATH* a do poľa *Value* napíšeme cestu do zložky *build*: */home/student/ns-allinone-3.21/ns-3.21/build* (obr. 1-9).
- Treba dať pozor, aby možnosť *Append environment to native environment* bola označená.
- V tomto momente môžeme spustiť exemplár tlačítkom, ktorý sme nastavili (*scratch-simulator*) pomocou ikonu *Run*. Mali by sme vidieť výpis do konzoly: „Scratch Simulator“.
- V prípade, že už máme hotový súbor s funkčným zdrojovým kódom a chceme to použiť v projekte, vložíme tento súbor do zložky *Scratch*.



Obr. 1-9 Environment Configuration

- V opačnom prípade je treba vytvoriť C++ súbor v zložke *Scratch*, do ktorého budeme písať zdrojový kód. Tento súbor nemôže byť prázdny, preto tam vložíme nasledujúce riadky a uložíme to:

```
#include "ns3/core-module.h"
NS_LOG_COMPONENT_DEFINE ("Debug");
using namespace ns3;
int main (int argc, char *argv[])
{
      NS_LOG_UNCOND ("Nastavenie debuggera");
}
```

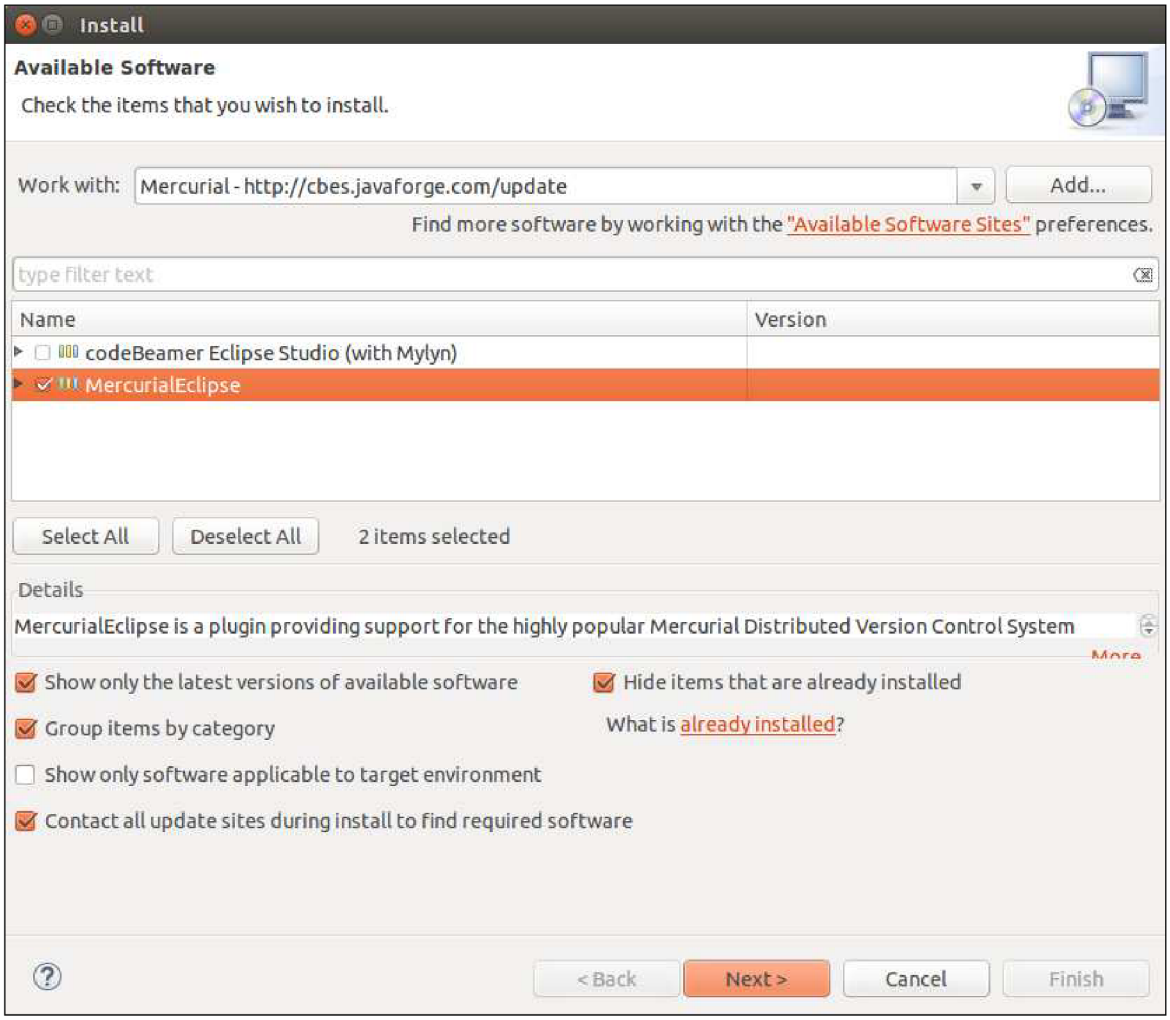
- Ešte raz spustíme exemplár *scratch-simulator* a tým sa skompiluje aj náš súbor v zložke *Scratch*.
- Z hlavného menu vyberieme položku *Run* → *Run Configurations* a pomocou tlačítka *Search Project* vyberieme náš súbor. Zmeny uložíme a spustíme simuláciu. V konzole by sme mali vidieť výstup nášho zdrojového kódu [3][6][7].

1.4.3 Mercurial

Inštalácia systému Mercurial nie je nevyhnutná, má to význam najmä pre správu NS-3 priamo z vývojového prostredia Eclipse. Po nastavení budeme môcť aktualizovať NS-3, kontrolovať

históriu a pod. z grafického užívateľského rozhrania. Pre nastavenie musíme previesť nasledujúce kroky:

- Z hlavného menu vyberieme *Help → Install New Software* a pridáme nasledujúcu adresu: http://cbes.javaforge.com/update. Pomenujeme to napríklad *Mercurial* (obr. 1-10).
- Zo zoznamu vyberieme položku *mercurialeclipse*, začneme inštalovať a držíme sa inštrukciám na obrazovke.
- Keď dostaneme chybové hlásenie, že autentičnosť a platnosť softwaru nie je zabezpečená, pokračujeme ďalej.
- Nakoniec klikneme pravým tlačítkom na náš projekt na ľavej strane obrazovky a z menu vyberieme položku *Team → Share Project*. V otvorenom okne vyberieme *Mercurial*, klikneme na tlačítko *Next* a tlačítkom *Finish* potvrdíme nastavenie [6][7].



Obr. 1-10 Inštalácia systému Mercurial

Poznámka: pri písaní tejto práce bolo zistené, že spustenie simulácií priamo z grafického užívateľského rozhrania Eclipse je časovo a výpočtovo oveľa náročnejší, než spustenie z príkazového riadku. Preto pri práci na menej výkonnom počítači je výhodnejší len písať zdrojový kód v tomto programe a spustiť z príkazového riadku.

1.5 Základné pojmy v NS-3

Táto podkapitola má za účel predstaviť základné pojmy v simulátore NS-3, oboznámenie sa s nimi je dôležité z hľadiska pochopenia štruktúry a fungovania simulátora. Tieto pojmy sú obecne používané v oblasti počítačových sietí, v tomto prípade však majú špecifický význam.

Koncové zariadenie v „Internetovom" žargóne sa nazýva *end system* alebo *host* – tie sú však asociované protokolmi Internetu, preto používame oveľa obecnejší výraz – uzly (node), ktoré sú zastúpené v programovom kóde triedou `Node` [2][8][9]. Nie sú rozlíšené zariadenia ako počítač alebo smerovač, v niektorej časti programového kódu im musíme prideliť potrebnú funkciu.

V reálnom svete potrebujeme mať na koncovom zariadení systémový software – operačný systém, aby sme mohli spúšťať aplikácie. Účelom tohto softwaru je vhodné rozdelenie zdrojov, medzi užívateľskými aplikáciami. Pretože operačný systém nevykonáva žiadnu úlohu, z ktorej by užívateľ mal priamo prínos, v NS-3 ostala len koncepcia užívateľskej aplikácie. Tieto aplikácie slúžia na generovanie prevádzky v sieti alebo majú testovací účel. V zdrojovom kóde sú reprezentované triedou `Application` a môžeme ich inštalovať na objekty `Node` [2][8][9].

Keď chceme komunikovať ďalšími uzlami v sieti, musíme ich spojiť vhodným komunikačným kanálom, ktorý je v NS-3 zastúpený triedou `Channel` [2][8][9]. Tento kanál má rôzne vlastnosti, ktoré môžeme nastaviť podľa vlastných potrieb, napr.: prenosová rýchlosť, oneskorenie alebo v prípade bezdrôtového spojenia simulácia 3D priestoru aj s prekážkami. Špeciálne prípady triedy `Channel` podľa typu spojenia:

- `CsmaChannel`,
- `PointToPointChannel`,
- `WifiChannel`.

Aby sme sa mohli pripojiť k sieti v skutočnom svete, je treba mať v počítači vhodnú periférium (sieťovú kartu), ďalej potrebujeme ešte software, ktorý riadi komunikáciu (ovládač). Funguje to podobne aj v NS-3, tie sú však zastúpené jedným objektom, ktorý sa nazýva `NetDevice` [2][8][9]. Podobne ako trieda `Channel`, aj trieda `NetDevice` má svoje špeciálne prípady podľa spojenia, ktoré sú nasledovné:

- `CsmaNetDevice`,
- `PointToPointNetDevice`,
- `WifiNetDevice`.

Vytvorenie, konfigurácia a spojenie objektov `Node`, `NetDevice` a `Channel` sú bežné a časté úlohy v NS-3, preto existujú tzv. *containery* a *helpery*, ktoré majú za účel zjednodušiť tieto operácie a uľahčiť prácu programátora [2][8][9]. Nasleduje zopár základných príkladov, ktoré budú použité aj v tejto práci:

- `NodeContainer`,
- `NetDeviceContainer`,

- `Ipv4AddressHelper`,
- `PoinToPointHelper`,
- `CsmaHelper`.

1.6 Postup pri vytvorení simulácie

- Definícia topológie – vytvorenie určitého počtu uzlov a ich umiestnenie vo virtuálnom priestore.
- Pridanie modelov – k simulácii sú pridané modely, napr. point-to-point a CSMA (Carrier Sense Multiple Access) zariadenia a linky, UDP, IPv4, IPv6 a aplikácie.
- Nastavenie uzlov a liniek – nastavenie parametrov modelov, napr. veľkosť paketu alebo veľkosť MTU (Maximum Transmission Unit).
- Prevedenie simulácie – po spustení simulácie požadované dáta a informácie sú zaznamenávané.
- Analýza výsledkov – po ukončení simulácie dáta sú zaznamenávané ako diskrétne udalosti v danom časovom okamihu, ktoré môžeme spracovať rôznymi metódami.
- Grafická vizualizácia – zaznamenané dáta môžeme vizualizovať rôznymi spôsobmi, napr. pomocou programu Gnuplot alebo NetAnim [2].

V ďalšej kapitole budú popísané protokoly IPv4 a IPv6, ktoré fungujú na sieťovej vrstve modelu ISO/OSI. Budú popísané ich hlavičky, výhody a vylepšenia IPv6 oproti IPv4, ďalej bude hlbšie popísaná princíp fragmentácie.

2 PROTOKOLY IPV4 A IPV6

V dnešnej dobe väčšina sieťových zariadení je zostavená na základe modelu ISO/OSI, ktorý sa skladá z rôznych vrstiev (obr. 2-1). V tejto kapitole sa budeme zaoberať so sieťovou vrstvou a v rámci toho protokolmi IPv4 a IPv6.

Obr. 2-1 Model ISO/OSI

Sieťová vrstva zabezpečuje predovšetkým smerovanie medzi rôznymi sieťami, tým vytvorí zdanlivú homogénnu sieť. Rôzne siete však môžu byť založené na rôznych technológiách na spojovej vrstve, preto sa musí vyrovnať s nasledujúcimi odlišnosťami:

- formáty adries (na spojovej vrstve),
- formát a maximálna veľkosť paketov,
- charakter služieb (spojové alebo nespojové) [9].

2.1 Protokol IPv4

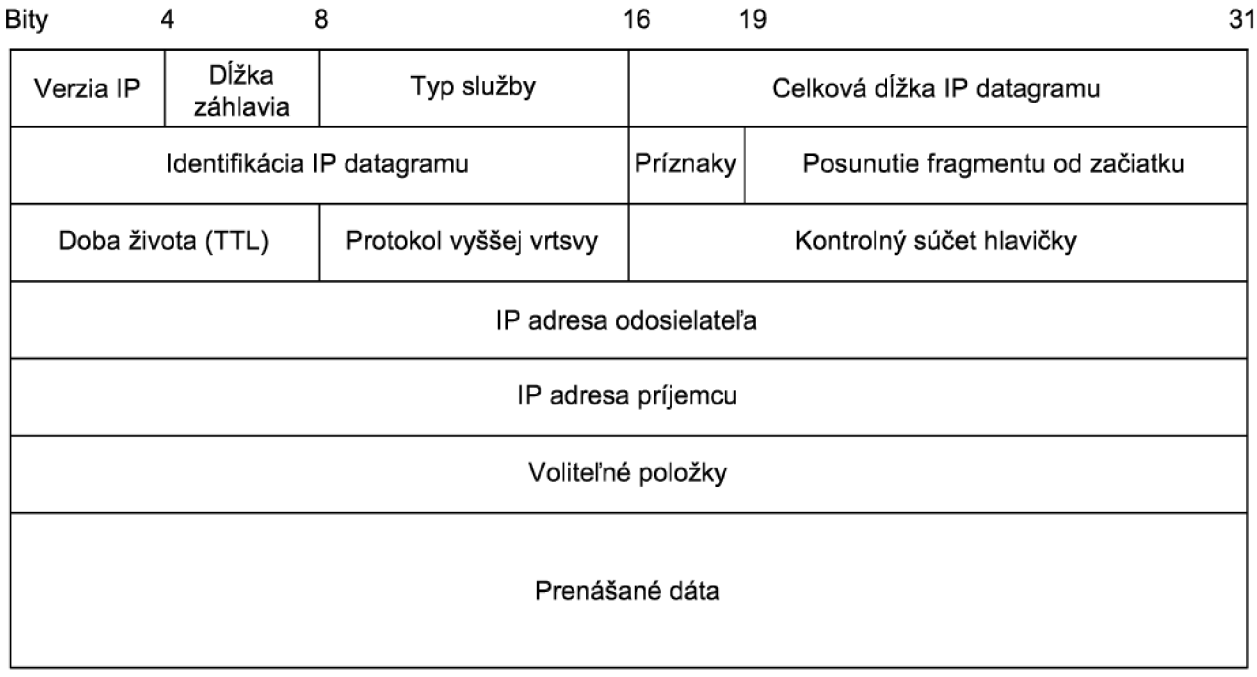
Ako to už bolo zmienené, protokol IPv4 funguje na sieťovej vrstve modelu ISO/OSI, má za úlohu nájsť cestu a doručiť pakety od odosielateľa k príjemcovi. Dáta, ktoré dostáva od vyšších vrstiev rozdelí na menšie časti a všetkým pridelí vlastnú IPv4 hlavičku, ktorá definuje, ako budú s paketom zaobchádzať ostatné zariadenia (smerovače) na ceste k príjemcovi.

2.1.1 Záhlavie IPv4

Záhlavie IPv4 je vidno na obr. 2-2 a obsahuje nasledujúce polia:

- Verzia – verzia protokolu IP, v prípade IPv4 má hodnotu 4.
- Dĺžka záhlavia – v základnom prípade je to 20 B, ale záhlavie obsahuje aj časť pre voliteľné položky, preto sa dĺžka môže meniť, maximálne na 60 B.

- Typ služby (ToS – Type of Service) – zabezpečuje kvalitu prenosu (QoS – Quality of Service), teda smerovače na ceste musia brať do úvahy hodnotu tohto poľa a podľa toho vybrať cestu k príjemcovi, aby boli dodržané určité parametre prenosu (napr. prenosová rýchlosť, oneskorenie, kolísanie oneskorenia).
- Celková dĺžka IP datagramu – je to 16 bitové pole, teda maximálna veľkosť paketu je 65536 B.
- Identifikácia IP datagramov – identifikácia spolu súvisiacich fragmentov.
- Príznaky (Flags) – je to 3 bitové pole, ale používajú sa len 2 bity:
  - DF (don‘t fragment) – určuje, či paket môže byť fragmentovaný,
  - MF (more fragments) – indikuje, že aktuálny fragment nie je posledný.
- Posunutie fragmentu od začiatku – poradové číslo aktuálneho fragmentu voči prvému.
- Doba života – maximálny počet skokov.
- Protokol vyššej vrstvy – napr. TCP alebo UDP.
- Kontrolný súčet záhlavia – každý smerovač vypočíta kontrolný súčet rovnakým algoritmom, v prípade, že sa výsledky nezhodujú, došlo k chybe behom prenosu.
- IP adresa – 32 bitová logická adresa.
- Voliteľné položky – maximálne do 40 B, používajú sa zriedkavo.
- Prenášané dáta (payload) [9].



Obr. 2-2 Záhlavie IPv4 datagramu

2.1.2 Fragmentácia

Ako to už bolo zmienené na začiatku kapitoly, sieťová vrstva a protokol IP dokáže pracovať s rôznymi technológiami implementované na spojovej vrstve. Každá technológia používa svoj vlastný formát rámce (dáta na úrovni spojovej vrstvy), ktoré majú obmedzenú veľkosť. Preto je veľkosť paketov (dáta na úrovni sieťovej vrstvy) tiež obmedzená, tento parameter je označený

ako MTU a jeho hodnoty pre vybrané technológie sú zobrazené v tab. 2-1. Môže sa stať, že MTU niektorého zariadenia na ceste medzi koncovými zariadeniami je menší, než MTU ostatných, v tomto prípade sú dve možnosti:

- v záhlaví IPv4 nie je nastavený bit DF, teda fragmentácia je povolená – paket pokračuje ďalej v menších častiach,
- v záhlaví je nastavený bit DF, teda fragmentácia je zakázaná – smerovač informuje odosielateľa a paket zahodí [9].

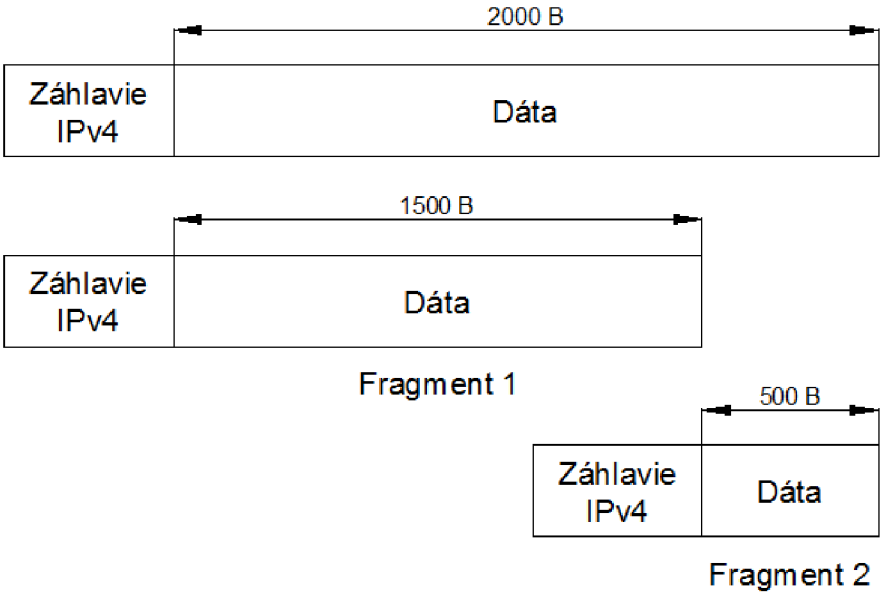
Tab. 2-1 Záhlavie IPv4 datagramu

Technológia na vrstve sieťového rozhrania	MTU [B]
FDDI	4478
Frame Relay	1600
Ethernet II	1500
Ethernet 802.3 SNAP	1492
ATM	48

V prípade, že fragmentácia je povolená, ďalší príznakový bit MF potom indikuje, že aktuálny fragment nie je posledný, nasleduje ešte ďalší. Pri fragmentácii sú nastavené aj polia *Posunutie fragmentu* a *Celková dĺžka datagramu* na príslušné hodnoty. Spolu súvisiace fragmenty sa identifikujú pomocou poľa *Identifikácia IP datagramu* [9]. Ostatné časti záhlavia ostávajú bez zmeny a každému fragmentu je pripojené nové záhlavie, čím vzniknú nové pakety (obr. 2-3).

V prípade, že fragmentácia je zakázaná, odosielateľ dostane od smerovača ICMP (Internet Control Message Protocol) správu „fragmentácia zakázaná“. Táto situácia sa rieši preskúmaním cesty (MTU Path Discovery) tak, že odosielateľ znižuje veľkosť paketov a sleduje, či dostane naspäť ICMP správu o zlyhaní komunikácie [11].

Je treba poznamenať, že fragmentácia sa môže uskutočniť v zdrojovom koncovom zariadení alebo v smerovačoch, ale ich poskladanie sa vykonáva len v koncových staniciach [11]. Hlavným dôvodom toho je, že jednotlivé pakety k zhodnému cieľu môžu byť posielané rôznymi cestami. Mohlo by sa stať, že smerovač na ceste nedostane všetky fragmenty pôvodného paketu. Ďalej je treba zmieniť, že k fragmentácii môže dôjsť aj viackrát.



Obr. 2-3 Fragmentácia paketu

2.1.3 Nedostatky protokolu IPv4

Protokol IPv4 bol definovaný a popísaný v dokumente RFC 791 [12], ktorý bol publikovaný v roku 1981 organizáciou IETF (Internet Engineering Task Force). Ide teda o pomerne zastaraný protokol, sieťové technológie sa pri tom zmenili výrazne. V posledných rokoch sa veľkosť sietí a počet sieťových zariadení veľmi rýchlo rástol a predpokladá sa, že nárast v budúcnosti bude ešte rýchlejší. Tým sa protokol IPv4 už nevyrovná kvôli určitým obmedzeniam:

- 32 bitové IP adresy umožňujú pripojenie 2^{32} rozhraní na sieť, čo je v dnešnej dobe málo.
- S predchádzajúcim bodom súvisí, že neexistuje skutočná *end-to-end* komunikácia, pretože kvôli nedostatku IP adries bol zavedený preklad adries (NAT – Network Address Translation).
- Rozsah smerovacích tabuliek na najvyššej úrovni Internetu je veľmi veľký kvôli nehierarchickému rozdeleniu rozsahu adries, čo kladie veľké nároky na hardware a software týchto zariadení [9].

2.2 Protokol IPv6

Následníkom protokolu IPv4 je protokol IPv6, ktorý má za účel odstrániť vyššie spomínané nedostatky dnešných sietí a prináša ďalšie výhody, ktoré budú popísané v tejto podkapitole. Nie je to len nová verzia protokolu IP, ale je to celá protokolová sada, ktorá v sebe zahrňuje nasledovné:

- adresovanie,
- automatická konfigurácia,
- ICMPv6,
- DHCPv6,
- premyslená mobilita staníc,
- správa multicastu [9].

2.2.1 Rozdiely medzi IPv6 a IPv4

Protokol IPv6 zavedie nasledujúce vylepšenia oproti IPv4:

- Možnosť autentizácie a kryptografického zabezpečenia,
- predpripravené mechanizmy pre zaistenie kvality služieb,
- menej povinných položiek v záhlaví,
- menšie náklady na smerovače pomocou hierarchického smerovania,
- neprepočítava sa kontrolný súčet (CRC – Cyclic Redundancy Check) a v normálnom prípade nedochádza k fragmentácii, čím sa zníži čas spracovania,
- premyslená mobilita staníc,
- nové podporné protokoly ICMPv6 (nevyhnutné pre fungovanie IPv6) a DHCPv6,
- jednotná adresná schéma, zabezpečenie spojenia *end-to-end*,
- neexistuje všesmerová (broadcast) adresa, je zastúpená multicastovými adresami,
- lepšia podpora multicastu,
- rozšírenie adresného priestoru [9][10].

Protokol IPv6 taktiež prináša zopár nevýhod, ktoré sú nasledovné:

- kvôli obrovskému adresnému priestoru nie je možné v rozumnom čase a jednoducho zistiť, ktoré adresy sa nachádzajú v sieti,
- zariadenia majú viac IP adries zároveň, ktoré sa môžu v čase aj meniť, čo komplikuje využívanie služby DNS (Domain Name System),
- kladie väčšie nároky na zariadenia a taktiež na ľudské zdroje, pretože v súčasnej dobe protokoly IPv4 a IPv6 musia fungovať súčasne a to tak, aby z pohľadu bežného užívateľa bolo irelevantné, ktorý práve používa,
- existuje veľa softvérov a hardvérov, ktoré boli vytvorené pre daný účel s čo najmenšími nákladmi, ktoré IPv6 nepodporujú,
- IPv6 v priebehu písaní tejto práce je ešte stále vo vývoji, preto má veľa funkcií, ktoré ešte nefungujú, alebo ich fungovanie nie je presne stanovené [9][10].

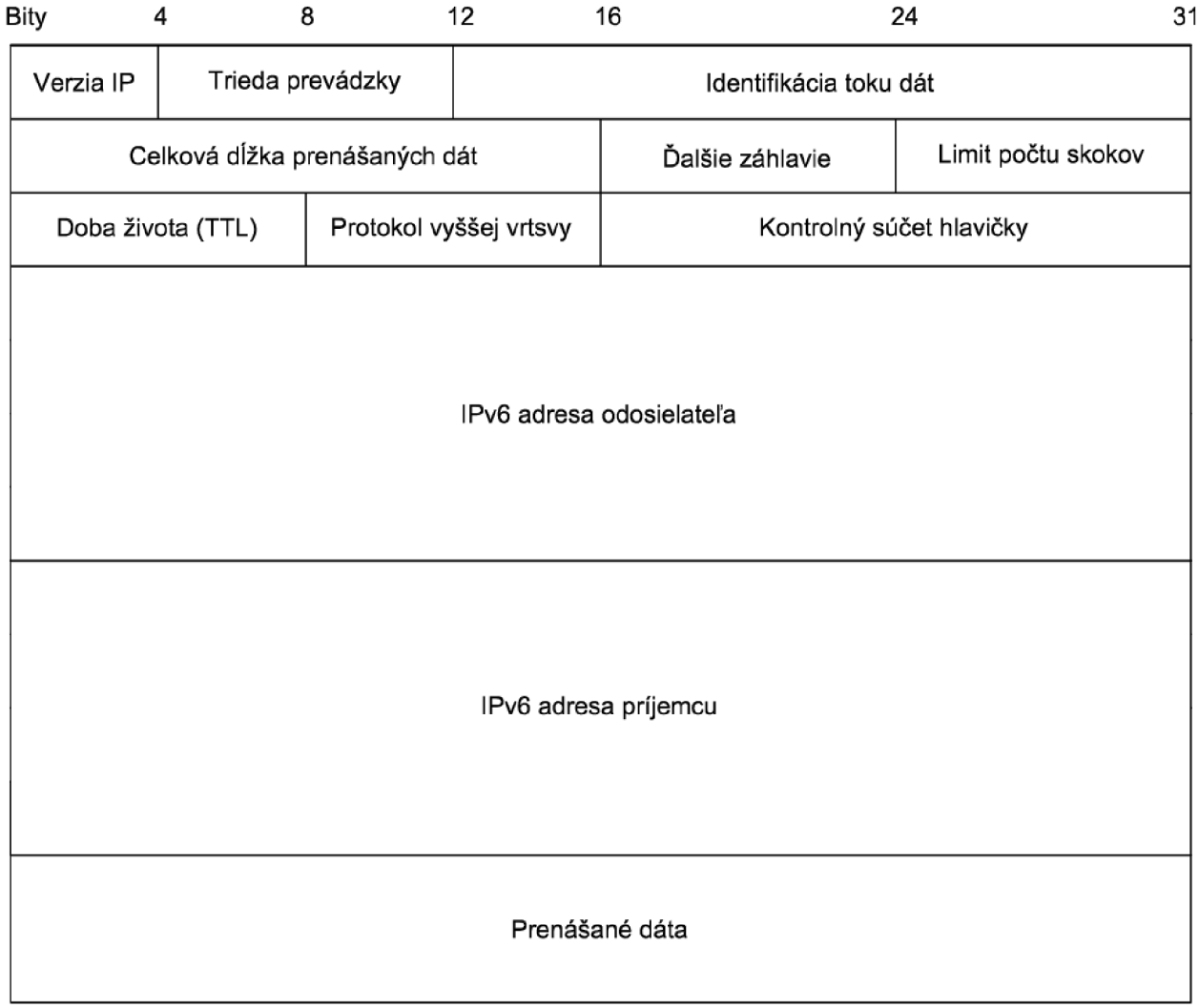
2.2.2 Záhlavie IPv6

V porovnaní s IPv4, záhlavie IPv6 obsahuje menej povinných položiek, tým sa zrýchľuje prenos – posiela sa menej režijných dát a spracovanie v smerovačoch taktiež zaberie menej času. Napriek tomu je veľkosť záhlavia dvojnásobné oproti záhlaviu IPv4 (20 B) kvôli 128 bitovým adresám, čo je ale akceptovateľné vzhľadom na výhody, ktoré prináša.

Záhlavie IPv6 datagramu (obr. 2-4) obsahuje nasledujúce pole:

- Verzia – verzia protokolu, v prípade IPv6 má hodnotu 6.
- Trieda prevádzky – značkovanie poľa typu služby (podobne ako u IPv4 pole ToS).
- Identifikácia toku dát (flow label) – umožňuje jednoduchšie smerovanie tým, že pakety s rovnakým číslom pošle rovnakou cestou, netreba znova hľadať výstupné rozhranie v smerovacej tabuľke.

- Celková dĺžka prenášaných dát (payload length).
- Ďalšie záhlavie (next header) – obsahujú rozširujúce informácie, napr. o fragmentácii alebo o smerovaní, ktoré sú potom zreťazené za sebou. V prípade, že nie je použité ďalšie záhlavie, pole ukazuje na prenášané dáta.
- Limit počtu skokov (hop limit) – maximálny počet skokov, po dosiahnutí je paket zahodený.
- IPv6 adresa odosielateľa a príjemcu (source/destination address) – 128 bitová logická adresa [9][10].



Obr. 2-4 Záhlavie IPv6 datagramu

2.2.3 Fragmentácia

Na rozdiel od IPv4, IPv6 vo vychádzajúcom stave nepovoľuje fragmentáciu, pretože čím menej dát nasleduje za záhlavím, tým je prenos menej efektívny. U protokolu IPv6 sa predpokladá, že každá sieť je schopná preniesť pakety s veľkosťou aspoň 1280 B, na menšie časti sa ani nedá paket rozdeliť. Fragmentácia je možná len v zdrojovej stanici a je považovaná za špeciálny prípad. K povoleniu je treba pripojiť k základnému ešte aj rozširujúce záhlavie [9]. V prípade,

keď je fragmentácia nevyhnutná, bola zavedená technika spätnej väzby, podobne ako u protokolu IPv4. Smerovač pošle odosielateľovi správu pomocou protokolu ICMPv6, že paket bol zahodený kvôli veľkosti (MTU Size Error). Táto správa obsahuje ďalej aj hodnotu MTU danej siete.

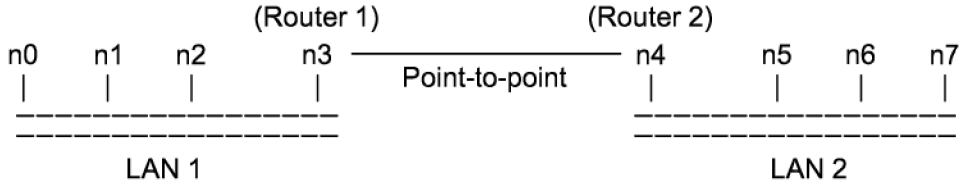
Poznámka: na žiadanie vedúceho práce boli postupy pre jednotlivé úlohy (kapitoly 3, 5, 7 a 9) napísané v českom jazyku.

3 Úloha 1: Srovnání protokolů IPv4 a IPv6

V této laboratorní úloze bude vytvořena síť, která používá protokoly IPv4 a IPv6 současně. Komunikace mezi různými sítěmi bude v případě protokolu IPv4 zajištěna globálním směrováním (`Ipv4GlobalRouting` – na začátku simulace jsou směrovací tabulky předem vytvořené a naplněné příslušnými cestami do každé sítě), v případě protokolu IPv6 statickým směrováním (`Ipv6StaticRouting`). Provoz bude zajištěn aplikacemi `UdpEchoServer` a `UdpEchoClient` a bude probíhat mezi uzly n0 a n7 (obr. 3-1). V závěru budou analyzovány výsledky pomocí programu Wireshark, důraz bude kladen na záhlaví zmíněných IP protokolů a na chování těchto protokolů v případě nutnosti fragmentace.

3.1 Topologie

Jak je to vidět na obr. 3-1, budou vytvořeny dvě lokální sítě LAN1 a LAN2, které budou spojeny technologií point-to-point. Propojení uzlů uvnitř těchto sítí bude realizováno sběrnicí (Ethernet).



Obr. 3-1 Topologie úlohy

3.2 Vytvoření modelu sítě

Vytvořte nový projekt ve složce *Scratch* a pojmenujte ho. Prázdný soubor následně upravte vhodným způsobem a nastavte, aby ho bylo možné spustit z prostředí Eclipse.

3.2.1 Vložení modulů

V této laboratorní úloze budeme potřebovat následující moduly:

```
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/gnuplot.h"
```

Před hlavní funkcí dále povolíme logování a definujeme jmenný prostor, který budeme v úloze používat:

```
NS_LOG_COMPONENT_DEFINE("Lab");

using namespace ns3;
```

3.2.2 Logování

V hlavní funkci `main`, jako první, nastavíme logování vybraných modulů, v tomto případě aplikací, které budeme používat pro generování provozu. Účelem toho je hlavně kontrola, zda zdrojový uzel poslal nastavené množství dat v určeném čase a zda cílový uzel posílány data dostal, případně na ně odpověděl.

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

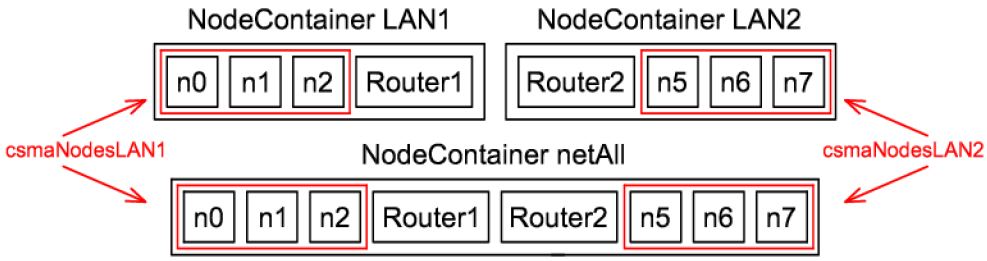
3.2.3 Definice uzlů

Koncové zařízení na lokálních sítích LAN1 (n0-n2) a LAN2 (n5-n7) vytvoříme pomocí třídy `NodeContainer`, směrovače (R1 a R2) však jiným způsobem – definujeme je jako objekty `Node`, které i pojmenujeme. Vytvořená proměnná `router1` ukazuje tedy přímo na příslušný uzel. Tato metoda je vhodná v případě, kdy máme v simulaci málo uzlů nebo chceme k některým z nich zjednodušit přístup podle jména.

```
NodeContainer csmaNodesLAN1;
csmaNodesLAN1.Create(3);

Ptr<Node> router1 = CreateObject<Node>();
Ptr<Node> router2 = CreateObject<Node>();

NodeContainer csmaNodesLAN2;
csmaNodesLAN2.Create(3);
```

Koncová zařízení a směrovače musíme dále v rámci jedné lokální sítě spojit do jedné třídy, abychom jim mohli přidělit společný adresní rozsah. Pro tento účel použijeme opět třídu `NodeContainer`. Jako vstupní parametr můžeme zadat již existující objekt `NodeContainer` nebo ukazatel na objekt `Node`. Pořadí uzlů v objektu `NodeContainer` závisí na pořadí vstupních parametrů, princip je vysvětlen na obr. 3-2.

```
NodeContainer LAN1(csmaNodesLAN1, router1);
NodeContainer p2p(router1, router2);
NodeContainer LAN2(router2, csmaNodesLAN2);
NodeContainer netAll(csmaNodesLAN1, router1, router2, csmaNodesLAN2);
```



Obr. 3-2 Pořadí uzlů v objektu NodeContainer

Nakonec „nainstalujeme“ protokolovou sadu TCP/IP na každý uzel. Proto byl vytvořen objekt `netAll`, který obsahuje všechny uzly v síti právě jednou. Kdybychom to dělali postupně

pomocí LAN1, LAN2 a p2p, na některé uzly bychom nainstalovali dva krát a po spuštění simulace bychom dostali chybové hlášení.

```
InternetStackHelper internetStack;
internetStack.Install(netAll);
```

3.2.4 Definícia kanálov

Z obr. 3-1 je zřejmé, že v úloze je nutné definovat dva typy spojení: sběrnici a bod-bod. Uděláme to pomocí tříd `CsmaHelper` a `PointToPointHelper`, **jejich parametry nastavíme podle** tab. 3-1. MTU sběrnic (csma) v NS-3 je ve výchozí konfiguraci 1500 B [13], vychází to z technologie Ethernet II. Tato hodnota zůstává nezměněna během úlohy, proto ji není nutné nastavovat. Následně vytvoříme a přidělíme síťová zařízení pomocí třídy `NetDeviceContainer`.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
pointToPoint.SetDeviceAttribute("Mtu", UintegerValue(1500));

CsmaHelper csma;
csma.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csma.SetChannelAttribute("Delay", TimeValue(NanoSeconds(6560)));

NetDeviceContainer netDevicesLAN1 = csma.Install(LAN1);
NetDeviceContainer netDevicesP2P = pointToPoint.Install(p2p);
NetDeviceContainer netDevicesLAN2 = csma.Install(LAN2);
```

Tab. 3-1 Hodnoty pro nastavení kanálů

Typ spojení	Parametr	Hodnota
Bod-bod	Rychlost	5 Mb/s
	Zpoždění	2 ms
	MTU	1500 B
Sběrnice	Rychlost	100 Mb/s
	Zpoždění	6560 ns
	MTU	1500 B

3.2.5 Nastavení IPv4 a IPv6 adres

Abychom mohli porovnat výše zmíněné protokoly, je nutné nastavit IPv4, i IPv6 adresu na uzlech, tím vytvoříme současný běh těchto protokolů. Pro nastavení IPv4 adres použijeme třídu `Ipv4AddressHelper` a adresní rozsahy uvedené v tab. 3-2. Po nastavení příslušného adresního rozsahu vytvoříme skupinu síťových rozhraní a přiřadíme k nim síťová zařízení.

```
Ipv4AddressHelper ipv4;

ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer ipv4InterfacesLAN1;
```

```
ipv4InterfacesLAN1 = ipv4.Assign(netDevicesLAN1);

ipv4.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer ipv4InterfacesP2P;
ipv4InterfacesP2P = ipv4.Assign(netDevicesP2P);

ipv4.SetBase("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer ipv4InterfacesLAN2;
ipv4InterfacesLAN2 = ipv4.Assign(netDevicesLAN2);
```

Tab. 3-2 Adresní rozsahy pro protokoly IPv4 a IPv6

Protokol	Síť	IP adresa sítě	Délka prefixu
IPv4	LAN1	10.1.1.0	/24
	P2P	10.1.2.0	/24
	LAN2	10.1.3.0	/24
IPv6	LAN1	2001:1:0:0::	/64
	P2P	2001:2:0:0::	/64
	LAN2	2001:3:0:0::	/64

Přidělení IPv6 adres se provádí podobně, jako přidělení IPv4 adres, existují ale určité odlišnosti. V případě protokolu IPv4 je NS-3 sestaven tak, že se všechny uzly chovají jako směrovače, které používají směrovací protokol OSPF [13]. Mají tedy v paměti směrovací tabulku, podle které rozhodují o přeposílání paketů v případě, že paket nebyl určen pro ně.

V případě protokolu IPv6 se však uzly ve výchozím nastavení chovají jako koncové stanice, které zahodí příchozí pakety, které pro ně nebyly určeny [13]. Proto musíme povolit směrování pro určité uzly, v našem případě logicky pro směrovače R1 a R2. Nastavuje se to pomocí `SetForwarding`, jenž je funkcí třídy `Ipv6InterfaceContainer`. Z toho vyplývá, že směrování je možné nastavit pro každé rozhraní zvlášť.

Další důležitou funkcí této třídy je `SetDefaultRouteInAllNodes`, kterou nastavíme výchozí rozhraní (cesta nebo brána) pro ostatní uzly. Je důležité si uvědomit, že tuto informaci dostanou i sousední směrovače a objeví se to ve směrovací tabulce jako defaultní cesta. Proto je toto vhodné nastavovat hlavně pro koncové stanice nebo směrovače (stub router/network), které mají jen jednu cestu ven do ostatních sítí.

Níže uvedený zdrojový kód obsahuje nastavení adresních rozsahů, např. pro síť LAN1 byly nastaveny následující hodnoty:

- IPv6 adresní rozsah *2001:1:0:0::/64*,
- směrování pro rozhraní č. 3 v rámci `ipv6InterfacesLAN1`,
- parametr `true`, který povoluje směrování pro dané rozhraní,
- rozhraní č. 3 pro výchozí cestu pro ostatní uzly.

Číslo rozhraní je zděděné od objektu `NodeContainer LAN1` (obr. 3-2) podobně jako u protokolu IPv4. Posledních 64 bitů IPv6 adresy jsou generovány podle MAC adresy stanic, používá se standard IEEE EUI-64.

```
Ipv6AddressHelper ipv6;

ipv6.SetBase(Ipv6Address("2001:1:0:0::"), Ipv6Prefix(64));
Ipv6InterfaceContainer ipv6InterfacesLAN1 = ipv6.Assign(netDevicesLAN1);
ipv6InterfacesLAN1.SetForwarding(3, true);
ipv6InterfacesLAN1.SetDefaultRouteInAllNodes(3);

ipv6.SetBase(Ipv6Address("2001:2:0:0::"), Ipv6Prefix(64));
Ipv6InterfaceContainer ipv6InterfacesP2P = ipv6.Assign(netDevicesP2P);
ipv6InterfacesP2P.SetForwarding(0, true);
ipv6InterfacesP2P.SetDefaultRouteInAllNodes(0);
ipv6InterfacesP2P.SetForwarding(1, true);
ipv6InterfacesP2P.SetDefaultRouteInAllNodes(1);

ipv6.SetBase(Ipv6Address("2001:0:0:3::"), Ipv6Prefix(64));
Ipv6InterfaceContainer ipv6InterfacesLAN2 = ipv6.Assign(netDevicesLAN2);
ipv6InterfacesLAN2.SetForwarding(0, true);
ipv6InterfacesLAN2.SetDefaultRouteInAllNodes(0);
```

3.2.6 Nastavení směrování

V předchozí podkapitole bylo zmíněno, že uzly s protokolem IPv4 se chovají jako směrovače s protokolem OSPF. Nazývá se to globální směrování, není to však defaultně povoleno. Nastavení globálního směrování se provádí následujícím řádkem:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

V průběhu psaní této práce je směrování v NS-3 s protokolem IPv6 stále ve vývoji, proto mnoho možností nemáme. Jediným řešením jsou statické cesty, které však není možné pohodlně konfigurovat, je nutné dát pozor na čísla jednotlivých rozhraní.

Objekt `routingHelper` můžeme chápat jako ukazatel na danou směrovací tabulku a pomocí třídy `Ipv6StaticRoutingHelper` nastavíme tento ukazatel na IPv6 směrovací tabulku směrovače R1. Dalším krokem je přidání samotné cesty do sítě LAN2 pomocí funkce `AddHostRouteTo`.

První řádek představuje adresu cílového rozhraní. Protože uzly s protokolem IPv6 mají více IP adres, musíme je nějakým způsobem rozlišit. První vstupní parametr ve funkci `GetAddress` znamená číslo rozhraní a druhý, pořadové číslo adresy. Hodnota 0 představuje obvykle lokální linkovou adresu, která je platná pouze v rámci lokální sítě, proto není vhodná pro náš případ. Hodnota 1 pak znamená námi přidělenou adresu, v tomto případě globální unikátní adresu [13]. Všimnete si, že je to adresa konkrétního zařízení a ne adresní rozsah. Záznam ve směrovací tabulce bude mít prefix /128, což se v praxi používá pouze ve výjimečných případech.

Druhým řádkem nastavíme adresu rozhraní následujícího skoku (next hop), princip je popsán výše. Poslední řádek slouží k nastavení výstupního rozhraní, vstupním parametrem funkce `GetInterfaceIndex` je číslo rozhraní.

```
Ipv6StaticRoutingHelper routingHelper;

Ptr<Ipv6StaticRouting> routingTable = routingHelper.GetStaticRouting(
                    router1->GetObject<Ipv6>());
routingTable->AddHostRouteTo( ipv6InterfacesLAN2.GetAddress(3, 1),
                              ipv6InterfacesP2P.GetAddress(1, 1),
                              ipv6InterfacesP2P.GetInterfaceIndex(0));
routingTable = routingHelper.GetStaticRouting(
                         router2->GetObject<Ipv6>());
routingTable->AddHostRouteTo(
                         ipv6InterfacesLAN1.GetAddress(0, 1),
                         ipv6InterfacesP2P.GetAddress(0, 1),
                         ipv6InterfacesP2P.GetInterfaceIndex(1));
```

3.2.7 Nastavení aplikací

V této části budou vytvořeny dvě aplikace `UdpEchoClient`, jedna pro přenos s protokolem IPv4, druhá pro přenos s protokolem IPv6. Dosáhneme toho tak, že v prvním případě použijeme IPv4 cílové rozhraní, v druhém případě IPv6 cílové rozhraní. `UdpEchoServer` je nezávislý na zmíněných protokolech, proto stačí vytvořit jednu aplikaci, která bude odpovídat na požadavky s protokolem IPv4, i s protokolem IPv6.

Na začátku vytvoříme aplikaci serveru (`echoServer`) s otevřeným portem č. 9 a nainstalujeme ji na uzel n7. Dále nainstalujeme aplikaci klienta (`echoClientIPv4`) s IPv4 cílovým rozhraním na uzel n0 s parametry uvedenými v 0. Nastavení velikosti paketu v NS-3 může být poněkud matoucí, protože atribut `PacketSize` označuje pouze datovou část bez záhlaví IP protokolu.

```
UdpEchoServerHelper echoServer(9);
ApplicationContainer serverApps = echoServer.Install(LAN2.Get(3));

UdpEchoClientHelper echoClientIPv4(ipv4InterfacesLAN2.GetAddress(3), 9);
echoClientIPv4.SetAttribute("MaxPackets", UintegerValue(4));
echoClientIPv4.SetAttribute("Interval", TimeValue(Seconds(1.0)));
echoClientIPv4.SetAttribute("PacketSize", UintegerValue(1024));
ApplicationContainer clientAppsIPv4 =
echoClientIPv4.Install(csmaNodesLAN1.Get(0));
```

Následně nastavíme další aplikaci `UdpEchoClient` na uzlu n0, ale v tomto případě s rozhraním IPv6. Při vytvoření objektu `echoClientIPv6` musíme uvést rozhraní cílového zařízení (ne rozhraní následujícího zařízení na cestě) a port, funkce `GetAddress` byla popsána v předchozí podkapitole. Pomocí funkce `SetAttribute` nastavíme požadované parametry. Nakonec aplikaci nainstalujeme na zdrojový uzel n0 a nastavíme čas spuštění a ukončení aplikace. Je důležité, aby se běh dvou klientů nepřekrýval, může to způsobit zpoždění pro jednotlivé provozy.

```
UdpEchoClientHelper echoClientIPv6(ipv6InterfacesLAN2.GetAddress(3,1), 9);
echoClientIPv6.SetAttribute("MaxPackets", UintegerValue(4));
echoClientIPv6.SetAttribute("Interval", TimeValue(Seconds(1.0)));
echoClientIPv6.SetAttribute("PacketSize", UintegerValue(1024));

ApplicationContainer clientAppsIPv6 =
echoClientIPv6.Install(csmaNodesLAN1.Get(0));
clientAppsIPv6.Start(Seconds(8.0));
clientAppsIPv6.Stop(Seconds(16.0));
```

Tab. 3-3 Hodnoty pro nastavení aplikací

Atributy	**Hodnoty**
Čas spuštění klienta a serveru	2 s
Čas ukončení klienta	8 s
Čas ukončení serveru	16s
Počet poslaných paketů	4
Interval mezi posíláním paketů	1 s
Velikost datové části bez záhlaví	1024 B

3.2.8 Definice výstupů

Na začátku úlohy byly povoleny logové zprávy pro použité aplikace. Kromě toho nastavíme výpis IPv6 směrovacích tabulek do konzole následujícím zdrojovým kódem:

```
Ptr<OutputStreamWrapper> routingStream =
Create<OutputStreamWrapper>(&std::cout);
routingHelper.PrintRoutingTableAt(Seconds(16.0), router1, routingStream);
routingHelper.PrintRoutingTableAt(Seconds(16.0), router2, routingStream);
```

Nejprve vytvoříme ukazatel na vstup příkazového řádku pomocí třídy `OutputStreamWrapper`, pak k výpisu použijeme objekt `routingHelper`, který jsme vytvořili v kapitole 3.2.6. Funkce `PrintRoutingTableAt` má 3 vstupní parametry:

- čas v simulaci, kdy nás obsah směrovací tabulky zajímá (logicky může se měnit v průběhu simulace),
- směrovač, jehož směrovací tabulku chceme vypisovat,
- výstup, kam údaje mají být poslány, v našem případě už dopředu definovaný vstup příkazového řádku (konzole).

Dále nastavíme generování trasovacího souboru pro uzel n0, abychom mohli hlouběji analyzovat provoz na síti. Nakonec definujeme spuštění a ukončení simulátoru.

```
csma.EnablePcap("ipv4-6", netDevicesLAN1.Get(0));

Simulator::Run();
Simulator::Destroy();
return 0;
```

3.2.9 Měření hodnoty RTT (round-trip time)

V této podkapitole bude uvedena a popsána část zdrojového kódu pro měření hodnoty RTT, která udává zpoždění mezi odesláním paketu a přijetím odpovědi na tento paket. Výsledky budou prezentovány jako závislost RTT v čase.

Pro měření byl použit systém trasovacích zdrojů (trace sources), na který je možné se připojit pomocí metody `Config::Connect`. Jako vstupní parametr je nutné uvést cestu k požadovanému zdroji, v našem případě jsou to zdroje `Ipv4L3Protocol` a `Ipv6L3Protocol` na uzlu n0, které indikují odchod a příchod (`Tx` a `Rx`) IPv4 a IPv6 paketů. Druhým parametrem je funkce, kterou chceme volat v případě, kdy nastane příslušná událost. **Níže uvedenou část zdrojového kódu umístěte před řádek, který spustí simulátor.**

```
Config::Connect("/NodeList/0/$ns3::Ipv4L3Protocol/Tx",
MakeCallback(&ipv4_node0_tx));
Config::Connect("/NodeList/0/$ns3::Ipv4L3Protocol/Rx",
MakeCallback(&ipv4_node0_rx));
Config::Connect("/NodeList/0/$ns3::Ipv6L3Protocol/Tx",
MakeCallback(&ipv6_node0_tx));
Config::Connect("/NodeList/0/$ns3::Ipv6L3Protocol/Rx",
MakeCallback(&ipv6_node0_rx));
```

Funkce, které se mají volat při posílání paketů, je nutné vytvořit před hlavní funkcí `main`. Proměnné, které budeme v těchto funkcích používat, musíme deklarovat jako globální, **tedy následující řádky umístěte pod řádek, kde je definován jmenný prostor**.

```
Gnuplot2dDataset datasetIPv4;
Gnuplot2dDataset datasetIPv6;
double rtt;
double simulationTime;
int ipv4PacketCount = 0;
int ipv6PacketCount = 0;
```

Proměnné `datasetIPv4` a `datasetIPv6` budou sloužit jako databáze naměřených hodnot, význam ostatních proměnných bude vysvětlen později.

V dalších částech budeme potřebovat proměnnou typu `string`, která však není součástí jmenného prostoru `ns3`. Proto je nutné definovat další jmenný prostor `std`, který je standardní knihovnou jazyka C. **Následující řádek umístěte pod řádek, kde je definován jmenný prostor** `ns3`**.**

```
using namespace std;
```

Níže je uveden zdrojový kód pro funkce, které se mají volat při posílání paketů. Funkce `ipv4_node0_tx` se má volat, když uzel n0 pošle IPv4 paket. Pomocí `ipv4PacketCount` počítáme počet odchozích paketů, do proměnných `simulationTime` a `rtt` se uloží aktuální čas v simulaci. Podmínka na začátku slouží k odfiltrování provozu, který se generuje v síti mezi uzly automaticky (např. kvůli protokolu ARP).

Funkce `ipv4_node0_rx` je volána, když uzel n0 dostane IPv4 paket. V předchozí fázi se do `rtt` byl uložen čas posílání, v tomto kroku tuto hodnotu odečteme od aktuálního času, tím

dostaneme hodnotu RTT. Dále čas posílání a hodnotu RTT uložíme do databáze pomocí funkce `Add`. Podmínka před tímto řádkem nepovoluje záznam prvního příchozího paketu. Je to kvůli tomu, že se směrování musí „rozjet“ a první paket bude mnohem více zpožděný, tím by zkresloval výsledky.

Dále jsou uvedeny i funkce pro měření hodnoty RTT v případě protokolu IPv6, které fungují podobně jako v případě protokolu IPv4, proto nebudou detailně popsány. Uvedený zdrojový kód umístěte před hlavní funkci `main`.

```
void ipv4_node0_tx(string path, Ptr<Packet const> packet, Ptr<Ipv4> ipv4,
unsigned int parameter) {
      if (packet->GetSize() >= 100) {
            ipv4PacketCount++;
            simulationTime = Simulator::Now().GetSeconds();
            rtt = Simulator::Now().GetSeconds();
      }}

void ipv4_node0_rx(string path, Ptr<Packet const> packet, Ptr<Ipv4> ipv4,
unsigned int parameter) {
      if (packet->GetSize() > 100) {
            rtt = (Simulator::Now().GetSeconds() - rtt) * 1000;
            if(ipv4PacketCount > 1)
                  datasetIPv4.Add(simulationTime, rtt);
      }}
void ipv6_node0_tx(string path, Ptr<Packet const> packet, Ptr<Ipv6> ipv6,
unsigned int parameter) {
      if (packet->GetSize() >= 100) {
            ipv6PacketCount++;
            simulationTime = Simulator::Now().GetSeconds();
            rtt = Simulator::Now().GetSeconds();
      }}
void ipv6_node0_rx(string path, Ptr<Packet const> packet, Ptr<Ipv6> ipv6,
unsigned int aaa) {
      if (packet->GetSize() > 100) {
            rtt = (Simulator::Now().GetSeconds() - rtt) * 1000;
            if(ipv6PacketCount > 1)
                  datasetIPv6.Add(simulationTime, rtt);
      }}
```

Abychom naměřené hodnoty mohli zobrazit graficky, musíme je z databáze uložit do souboru *.plt*. Slouží k tomu třída `Gnuplot` a **na konec zdrojového kódu (před řádek `return 0;`) musíme uvést následující řádky**:

```
string fileNameWithNoExtension = "RTT";
string graphicsFileName = fileNameWithNoExtension + ".png";
string plotFileName = fileNameWithNoExtension + ".plt";
string plotTitle = "RTT";

datasetIPv4.SetTitle("IPv4");
datasetIPv4.SetStyle(Gnuplot2dDataset::LINES_POINTS);
datasetIPv6.SetTitle("IPv6");
datasetIPv6.SetStyle(Gnuplot2dDataset::LINES_POINTS);

Gnuplot plot(graphicsFileName);

plot.SetTitle(plotTitle);
plot.SetTerminal("png");
```

```
plot.SetLegend("Time [s]", "RTT [ms]");
plot.AppendExtra ("set yrange [7.7:+7.9]");
plot.AddDataset(datasetIPv4);
plot.AddDataset(datasetIPv6);

ofstream plotFile(plotFileName.c_str());
plot.GenerateOutput(plotFile);
plotFile.close();
```

Aby zdrojový kód byl přehlednější, byla zvolena taková metoda, že jednotlivé pojmenování byly nejprve uloženy do proměnných `string`. Před přidáním dat do výstupního souboru nastavíme jméno jednotlivých sérií dat (`datasetIPv4.SetTitle`) a typ zobrazení (`datasetIPv4.SetStyle`), v našem případě jsou to body, které jsou spojeny čarou.

Při práci s třídou `Gnuplot` je možné nastavit následující parametry (existují i jiné, které však nebyly použity):

- při vytvoření třídy vstupním parametrem je jméno výstupního grafického souboru,
- `SetTitle` – jméno grafu,
- `SetTerminal` – typ výstupního grafického souboru,
- `SetLegend` – popis x-ové a y-ové osy,
- `AppendExtra` – nastavení rozsahu x-ové nebo y-ové osy (je nastaveno pouze z estetických důvodů, můžeme vynechat),
- `AddDataset` – přidání vytvořených databází (sérií dat) – jako je to vidět ve zdrojovém kódu, v našem případě jsme přidali dvě databáze.

Následuje vytvoření samotného datového souboru (ještě ne grafického) s příponou *.plt*, vstupním parametrem proměnné `plotFile` je jméno výstupního datového souboru. Pomocí funkce `GenerateOutput` uložíme naměřené hodnoty do vytvořeného souboru a nakonec soubor zavřeme.

3.3 Výsledky

3.3.1 Výstup z příkazového řádku

Po spuštění simulace dostaneme delší výstup do konzole, proto byl výstup rozdělen na menší části. Prvních pár řádků zobrazují UDP provoz na síti s protokolem IPv4. Scénář byl nastaven tak, aby aplikace `UdpEchoClient` začala posílat pakety s velikostí datové části 1024 B v čase 2 s. Je vidět, že výstup tomu odpovídá, klient s IP adresou 10.1.1.1 poslal čtyři pakety a server s IP adresou 10.1.3.4 na ně odpověděl.

```
At time 2s client sent 1024 bytes to 10.1.3.4 port 9
At time 2.01492s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.01492s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.02484s client received 1024 bytes from 10.1.3.4 port 9
At time 3s client sent 1024 bytes to 10.1.3.4 port 9
At time 3.00387s server received 1024 bytes from 10.1.1.1 port 49153
At time 3.00387s server sent 1024 bytes to 10.1.1.1 port 49153
At time 3.00774s client received 1024 bytes from 10.1.3.4 port 9
```

```
At time 4s client sent 1024 bytes to 10.1.3.4 port 9
At time 4.00387s server received 1024 bytes from 10.1.1.1 port 49153
At time 4.00387s server sent 1024 bytes to 10.1.1.1 port 49153
At time 4.00774s client received 1024 bytes from 10.1.3.4 port 9
At time 5s client sent 1024 bytes to 10.1.3.4 port 9
At time 5.00387s server received 1024 bytes from 10.1.1.1 port 49153
At time 5.00387s server sent 1024 bytes to 10.1.1.1 port 49153
At time 5.00774s client received 1024 bytes from 10.1.3.4 port 9
```

Následující část výstupu zobrazuje UDP provoz s protokolem IPv6. Je to podobný výpis, zdrojové a cílové adresy jsou ale IPv6 adresy. Je vidět, že klient s IP adresou 2001:1:0:0:200:ff:fe00:1 poslal 4 pakety na adresu 2001:3:0:1:200:ff:fe00, server tyto zprávy dostal a odpověděl na ně.

```
At time 8s client sent 1024 bytes to 2001:3::200:ff:fe00:a port 9
At time 8.00696s server received 1024 bytes from 2001:1::200:ff:fe00:1 port
49153
At time 8.00696s server sent 1024 bytes to 2001:1::200:ff:fe00:1 port 49153
At time 8.02293s client received 1024 bytes from 2001:3::200:ff:fe00:a port 9
At time 9s client sent 1024 bytes to 2001:3::200:ff:fe00:a port 9
At time 9.00391s server received 1024 bytes from 2001:1::200:ff:fe00:1 port
49153
At time 9.00391s server sent 1024 bytes to 2001:1::200:ff:fe00:1 port 49153
At time 9.00781s client received 1024 bytes from 2001:3::200:ff:fe00:a port 9
At time 10s client sent 1024 bytes to 2001:3::200:ff:fe00:a port 9
At time 10.0039s server received 1024 bytes from 2001:1::200:ff:fe00:1 port
49153
At time 10.0039s server sent 1024 bytes to 2001:1::200:ff:fe00:1 port 49153
At time 10.0078s client received 1024 bytes from 2001:3::200:ff:fe00:a port 9
At time 11s client sent 1024 bytes to 2001:3::200:ff:fe00:a port 9
At time 11.0039s server received 1024 bytes from 2001:1::200:ff:fe00:1 port
49153
At time 11.0039s server sent 1024 bytes to 2001:1::200:ff:fe00:1 port 49153
At time 11.0078s client received 1024 bytes from 2001:3::200:ff:fe00:a port 9
```

Poslední částí konzolového výstupu jsou IPv6 směrovací tabulky směrovačů R1 a R2. Vidíme, že v obou tabulkách existují námi vytvořené statické cesty s prefixem /128 ke koncovým stanicím, mezi kterými probíhá provoz. Všimněte si, že směrovací tabulky obsahují také defaultní cesty s adresou ::/0, jak to bylo popsáno v kapitole 3.2.5.

```
Node: 3 Time: 16s Ipv6ListRouting table
  Priority: 0 Protocol: ns3::Ipv6StaticRouting
Node: 3 Time: 16s Ipv6StaticRouting table
Destination                    Next Hop                   Flag Met Ref Use If
::1/128                        ::                         UH   0   -   -   0
fe80::/64                      ::                         U    0   -   -   1
2001:1::/64                    ::                         U    0   -   -   1
fe80::/64                      ::                         U    0   -   -   2
2001:2::/64                    ::                         U    0   -   -   2
::/0                           fe80::200:ff:fe00:6        UG   0   -   -   2
2001:3::200:ff:fe00:a/128      2001:2::200:ff:fe00:6      UH   0   -   -   2

Node: 4 Time: 16s Ipv6ListRouting table
  Priority: 0 Protocol: ns3::Ipv6StaticRouting
Node: 4 Time: 16s Ipv6StaticRouting table
Destination                    Next Hop                   Flag Met Ref Use If
::1/128                        ::                         UH   0   -   -   0
fe80::/64                      ::                         U    0   -   -   1
```

```
2001:2::/64                 ::                  U     0      -      -      1
::/0                        fe80::200:ff:fe00:5 UG    0      -      -      1
fe80::/64                   ::                  U     0      -      -      2
2001:3::/64                 ::                  U     0      -      -      2
2001:1::200:ff:fe00:1/128   2001:2::200:ff:fe00:5 UH    0      -      -      1
```

3.3.2 Trasovací súbor

Vytvořený trasovací soubor můžeme otevřít v příkazovém řádku pomocí příkazu `tcpdump -nn -tt -r ipv4-6-0-1.pcap`. Je ale mnohem pohodlnější použít software Wireshark, který má grafické uživatelské rozhraní a je volně dostupný ke stažení. Otevřete tedy zmíněný soubor v programu Wireshark.

Na obr. 3-3 a obr. 3-4 jsou zachyceny záhlaví paketů s pořadovými čísly 22 a 34 (ve Wiresharku). Zobrazte si tyto pakety (můžete si vybrat i jiné) a samostatně si prostudujte význam jednotlivých položek.

```
▼Internet Protocol Version 4, Src: 10.1.1.1 (10.1.1.1), Dst: 10.1.3.4 (10.1.3.4)
  Version: 4
  Header length: 20 bytes
 ▶Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 1052
  Identification: 0x0003 (3)
 ▼Flags: 0x00
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 64
  Protocol: UDP (17)
 ▶Header checksum: 0x0000 [validation disabled]
  Source: 10.1.1.1 (10.1.1.1)
  Destination: 10.1.3.4 (10.1.3.4)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
```

Obr. 3-3 Výstup z programu Wireshark – záhlaví protokolu IPv4 (č. 22)

```
▼Internet Protocol Version 6, Src: 2001:1::200:ff:fe00:1 (2001:1::200:ff:fe00:1), Dst: 2001:3::200:ff:fe00:a (2001:3::200:ff:fe00:a)
 ▶0110 .... = Version: 6
 ▶.... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
   .... .... .... 0000 0000 0000 0000 0001 = Flowlabel: 0x00000001
  Payload length: 1032
  Next header: UDP (17)
  Hop limit: 64
  Source: 2001:1::200:ff:fe00:1 (2001:1::200:ff:fe00:1)
  [Source SA MAC: 00:00:00_00:00:01 (00:00:00:00:00:01)]
  Destination: 2001:3::200:ff:fe00:a (2001:3::200:ff:fe00:a)
  [Destination SA MAC: 00:00:00_00:00:0a (00:00:00:00:00:0a)]
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
```

Obr. 3-4 Výstup z programu Wireshark – záhlaví IPv6 (č. 34)

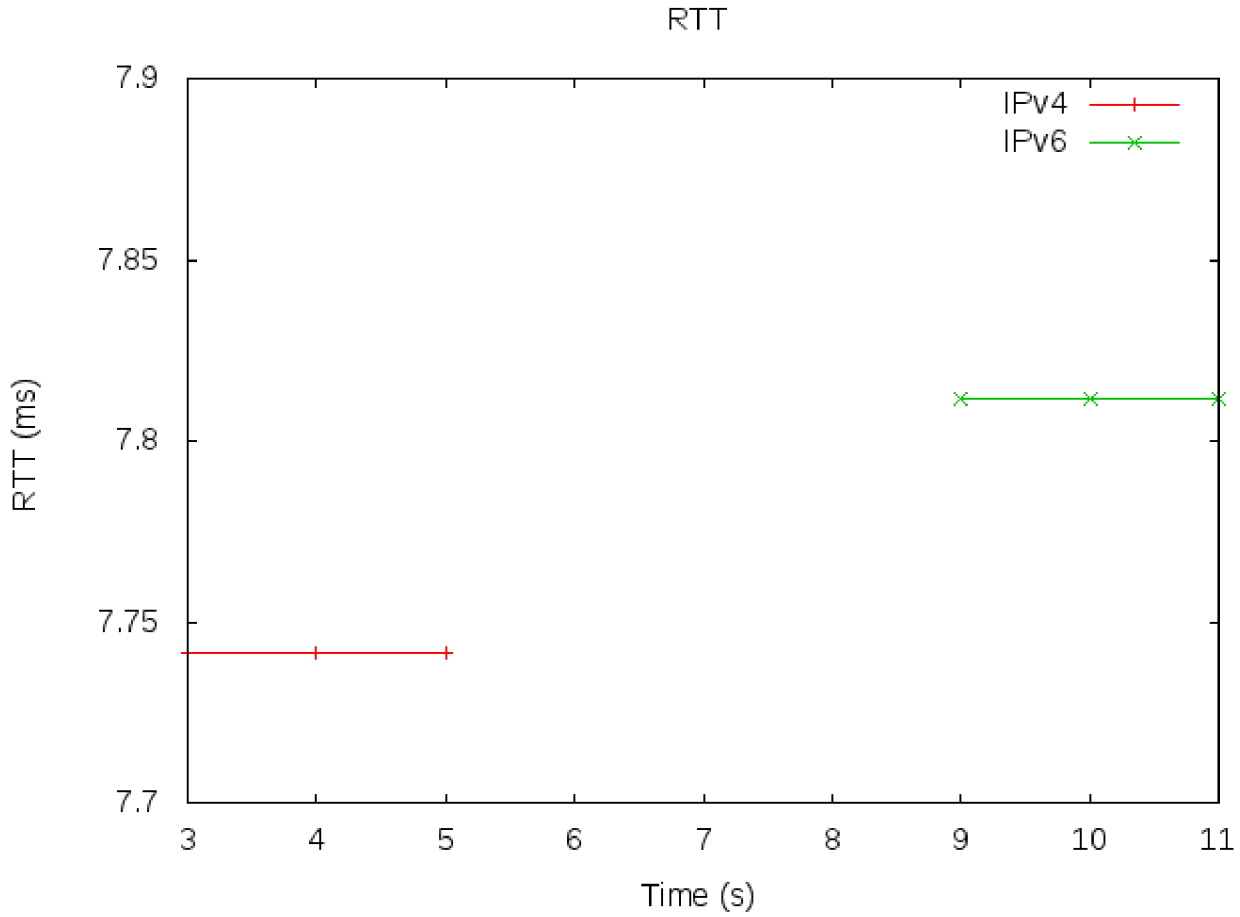
3.3.3 RTT

V kořenové složce se po spuštění simulace vytvořil soubor *RTT.plt*. Otevřete tuto složku v příkazovém řádku a napište příkaz `gnuplot RTT.plt` (obr. 3-5). Tím se ve stejné složce vytvoří soubor *RTT.png* (obr. 3-6).

```
student@fekt: ~/ns-allinone-3.21/ns-3.21
student@fekt:~/ns-allinone-3.21/ns-3.21$ gnuplot RTT.plt
```

Obr. 3-5 Příkaz k vytvoření grafického souboru

Červená čára označuje hodnotu RTT IPv4 paketů, zelená hodnotu RTT IPv6 paketů. Vytvořená síť je dost malá, přesto je vidět rozdíl mezi protokoly.



Obr. 3-6 Velikost hodnoty RTT u protokolů IPv4 a IPv6

3.3.4 Kontrolní otázky

- V simulaci byla nastavena velikost dat na 1024 B. Program Wireshark vypisuje, že celková velikost poslaných dat s protokolem IPv4 byla 1070B a s protokolem IPv6 1090 B. Vysvětlete rozdílné hodnoty. Do těchto hodnot se započítávají i velikosti záhlaví na jednotlivých vrstvách. Uveďte velikost těchto záhlaví (velikost záhlaví a kontrolní součet na spojové vrstvě mají velikost 18 B).
- Jaký standard byl použit na generování IPv6 adres? Vysvětlete princip tohoto standardu.
- Z obr. 3-3 a obr. 3-4 je vidět, že záhlaví IPv4 obsahuje položku *Header length* (velikost záhlaví), záhlaví IPv6 však ne. Proč v případě protokolu IPv6 je tato informace nepodstatná? Vysvětlete princip rozšíření záhlaví v případě protokolu IPv4 a IPv6.

- Na co slouží položky *Protocol* (IPv4) a *Next header* (IPv6)? Vysvětlete funkci položky *Next header* z pohledu rozšíření záhlaví.
- Proč v záhlaví IPv6 neexistuje pole kontrolního součtu (Header checksum)? Jak je zajištěna kontrola bezchybného příjmu u protokolu IPv6?
- K čemu slouží pole *Flow label* v záhlaví protokolu IPv6?
- Vysvětlete rozdíl v hodnotách RTT u protokolu IPv4 a IPv6.

3.4 Samostatné úkoly

Tato část úlohy se blíže zabývá s fragmentací paketů v případě protokolů IPv4 a IPv6:

- Vytvořenou simulaci modifikujte tak, aby došlo k fragmentaci paketů na lince bod-bod – velikost datových částí paketů zvyšte na 1450 B a velikost MTU linky bod-bod snižte na 1350 B. Analyzujte provoz zachycený na uzlu n0 pomocí Wireshark, po vhodné filtrací byste měli dostat podobný výstup, jako na obr. 3-7 a obr. 3-8.
- Vysvětlete princip fragmentace v případě obou protokolů. K čemu slouží příznakových bity (Flags) v záhlaví IPv4 a proč podobné pole neexistuje v záhlaví IPv6?

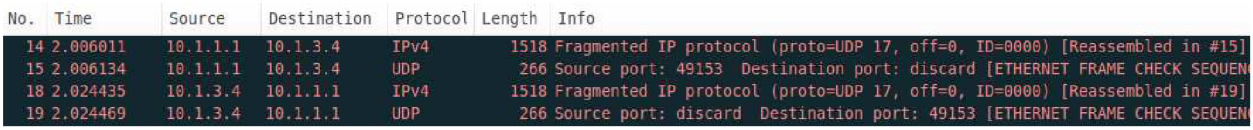
No.	Time	Source	Destination	Protocol	Length	Info
14	2.006011	10.1.1.1	10.1.3.4	UDP	1496	Source port: 49153 Destination port: discard [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
17	2.023912	10.1.3.4	10.1.1.1	IPv4	1366	Fragmented IP protocol (proto=UDP 17, off=0, ID=0000) [Reassembled in #18] [ETHERNET FR
18	2.023939	10.1.3.4	10.1.1.1	UDP	168	Source port: discard Destination port: 49153 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
19	2.997987	10.1.1.1	10.1.3.4	UDP	1496	Source port: 49153 Destination port: discard [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
20	3.006938	10.1.3.4	10.1.1.1	IPv4	1366	Fragmented IP protocol (proto=UDP 17, off=0, ID=0001) [Reassembled in #21] [ETHERNET FR
21	3.007085	10.1.3.4	10.1.1.1	UDP	168	Source port: discard Destination port: 49153 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]

Obr. 3-7 Výstup z Wiresharku po nastavení fragmentace - protokol IPv4 (data = 1450 B, MTU = 1350 B)

No.	Time	Source	Destination	Protocol	Length	Info
30	7.998015	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	UDP	1516	Source port: 49153 Destination port: discard [ILLEGAL CH
33	8.004284	2001:1::200:ff:fe00:4	2001:1::200:ff:fe00:1	ICMPv6	1298	Packet Too Big [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
34	8.997987	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	IPv6	1362	IPv6 fragment (nxt=UDP (17) off=0 id=0xa68f4557) [ETHERNE
35	8.998096	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	UDP	228	Source port: 49153 Destination port: discard [ILLEGAL CH
36	9.997987	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	IPv6	1362	IPv6 fragment (nxt=UDP (17) off=0 id=0x8c7cd8e0) [ETHERNE
37	9.998096	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	UDP	228	Source port: 49153 Destination port: discard [ILLEGAL CH
38	10.007004	2001:3::200:ff:fe00:a	2001:1::200:ff:fe00:1	IPv6	1362	IPv6 fragment (nxt=UDP (17) off=0 id=0x18485103) [ETHERNE
39	10.007253	2001:3::200:ff:fe00:a	2001:1::200:ff:fe00:1	UDP	228	Source port: discard Destination port: 49153 [ILLEGAL CH
40	10.997987	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	IPv6	1362	IPv6 fragment (nxt=UDP (17) off=0 id=0x8ed0be55) [ETHERNE
41	10.998096	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	UDP	228	Source port: 49153 Destination port: discard [ILLEGAL CH

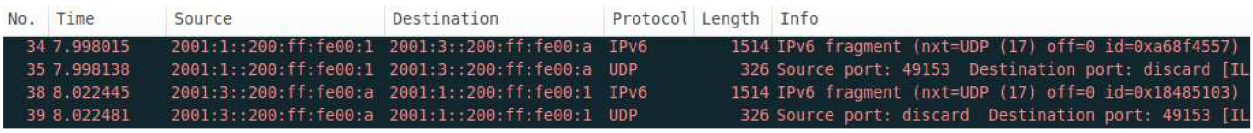
Obr. 3-8 Výstup z Wiresharku po nastavení fragmentace – protokol IPv6 (data = 1450 B, MTU = 1350 B)

- Změňte velikost datové části paketů na 1700 B (více než 1500 B s IP záhlavím) a velikost MTU na lince bod-bod na 1500 B. Opět analyzujte provoz, měli byste dostat odlišné výsledky. Jak je to vidět na obr. 3-9, k fragmentaci došlo již ve zdrojovém uzlu n0 i v případě protokolu IPv4 a ani protokol IPv6 (obr. 3-10) nepotřeboval zprávu *Packet Too Big*, fragmentace se začala automaticky. Vysvětlete důvod rozdílných výsledků oproti předchozímu případu.
- Kde dochází k poskládání fragmentovaného paketu a proč?
- Můžete si dále podívat na změny v záhlaví obou protokolů (Next header, Offset, More fragments).



No.	Time	Source	Destination	Protocol	Length	Info
14	2.006011	10.1.1.1	10.1.3.4	IPv4	1518	Fragmented IP protocol (proto=UDP 17, off=0, ID=0000) [Reassembled in #15]
15	2.006134	10.1.1.1	10.1.3.4	UDP	266	Source port: 49153 Destination port: discard [ETHERNET FRAME CHECK SEQUEN
18	2.024435	10.1.3.4	10.1.1.1	IPv4	1518	Fragmented IP protocol (proto=UDP 17, off=0, ID=0000) [Reassembled in #19]
19	2.024469	10.1.3.4	10.1.1.1	UDP	266	Source port: discard Destination port: 49153 [ETHERNET FRAME CHECK SEQUEN

Obr. 3-9 Výstup z Wiresharku po nastavení fragmentace – protokol IPv4 (dáta = 1700 B, MTU = 1500 B)



No.	Time	Source	Destination	Protocol	Length	Info
34	7.998015	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	IPv6	1514	IPv6 fragment (nxt=UDP (17) off=0 id=0xa68f4557)
35	7.998138	2001:1::200:ff:fe00:1	2001:3::200:ff:fe00:a	UDP	326	Source port: 49153 Destination port: discard [IL
38	8.022445	2001:3::200:ff:fe00:a	2001:1::200:ff:fe00:1	IPv6	1514	IPv6 fragment (nxt=UDP (17) off=0 id=0x18485103)
39	8.022481	2001:3::200:ff:fe00:a	2001:1::200:ff:fe00:1	UDP	326	Source port: discard Destination port: 49153 [IL

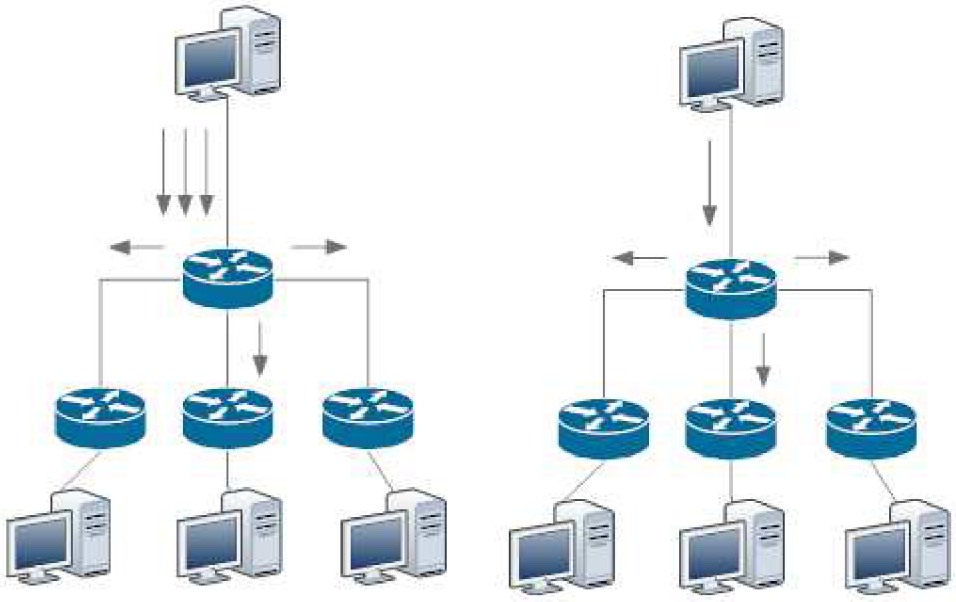
Obr. 3-10 Výstup z Wiresharku po nastavení fragmentace – protokol IPv6 (paket = 1700 B, MTU = 1500 B)

4 MULTICASTOVÝ PRENOS

V tejto kapitole budú predstavené rôzne typy prenosov a bude detailnejšie popísaný multicastový prenos – jeho vlastnosti, výhody a nevýhody, ďalej protokoly, na ktorých je založený.

4.1 Typy prenosov

- Unicast – najpoužívanejší typ prenosu, zdrojom a taktiež cieľom dát je jednoznačne určená stanica (one-to-one).
- Broadcast – je to prenos z jedného bodu k všetkým ostatným na sieti v rámci broadcastovej domény (one-to-all), ďalej sa tieto správy nešíria – broadcastové domény rozdeľujú smerovače. Všetky zariadenia na sieti dostanú tento typ správy, preto sa musia zaoberať s jeho spracovaním. Výhodou je šetrenie s prenosovou šírkou pásma, na druhej strane sú zaťažené aj zariadenia, ktorých správa sa netýka.
- Multicast – je to prenos v rámci jednej skupiny, kde môže byť jeden alebo viac zdrojov (one-to-many/many-to-many). Funguje to tak, že správa od zdroja je na ceste postupným vetvením duplikovaná. Pre správne fungovanie tohto typu prenosu je treba, aby všetky zariadenia (smerovače a prepínače) na sieti podporovali multicast. Ušetrená šírka pásma je potom veľká najmä v prípade, keď existuje veľa prijímacích staníc (obr. 4-1).



Obr. 4-1 Unicastový (ľavý) a multicastový (pravý) prenos

- Anycast – existuje na sieti viac staníc s rovnakou adresou (one-to-nearest), ktoré majú rovnakú úlohu. Komunikácia sa deje vždy s najbližšou alebo najlepšie dostupnou stanicou [9].

4.2 Výhody a nevýhody multicastového prenosu

Výhody multicastového prenosu:

- Je to oveľa efektívnejší typ prenosu z pohľadu šírky pásma, než unicast, a to hlavne v prípade, že existuje veľký počet prijímacích staníc.
- Pretože zdroj vysiela pakety vždy len raz a tieto pakety sú duplikované v príslušných miestach, multicastový prenos kladie oveľa menšie nároky na spracovanie paketov a tým šetrí s výkonom zariadení.
- Distribuované aplikácie by často neboli realizovateľné bez multicastového prenosu [9].

Dve najviac využívané protokoly na transportnej vrstve sú TCP a UDP. Pretože pri použití protokolu TCP je treba naviazať komunikáciu medzi dvoma komunikujúcimi stranami, nie je vhodný na prenos multicastových dát. K multicastovému prenosu sa teda využíva protokol UDP, z toho vyplýva väčšina jeho nevýhod:

- Neexistuje možnosť riadenia prenosu, taktiež sa nestará o tom, či svojou prevádzkou blokuje celú kapacitu linky alebo či nespôsobuje zahltenie siete.
- Pri zmene multicastovej topológie v priebehu prenosu môžu vzniknúť duplicitné pakety, alebo sa môže zmeniť poradie paketov. Aplikácie s týmito prípadmi musia počítať.
- Prenos nie je spoľahlivý, v prípade, že potrebujeme určitú mieru spoľahlivosti, je treba to zaistiť na aplikačnej vrstve.
- Pripojenie k multicastovej skupine je jednoduché, preto komunikácia je ľahko odpočúvatelná [9].

4.3 Adresovanie multicastu

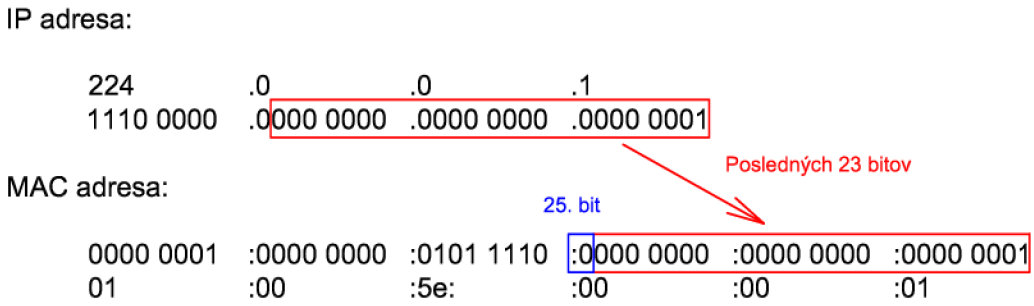
Pre multicastový prenos bol vyhradený väčší IPv4 adresný priestor, ktorý je označený ako skupina D s rozsahom 224.0.0.0 ÷ 239.255.255.255. Tento rozsah je ďalej rozdelený na menšie skupiny, niektoré z nich je vidno v tab. 4-1 [9]. Multicastové adresy používajú aj niektoré smerovacie protokoly pre výmenu smerovacích informácií, napr. EIGRP používa adresu 224.0.0.10.

Tab. 4-1 Skupiny multicastových adries [9]

Adresy	Popis
224.0.0.0 – 224.0.0.255	Určené pre použitie v rámci lokálnych sietí
224.0.1.0 – 238.255.255.255	Určené pre použitie v rámci Internetu
239.0.0.0 – 239.255.255.255	Určené pre privátne použitie vnútry domény
224.0.0.1	Všetky stanice na lokálnej sieti
224.0.0.2	Všetky multicastové smerovače

Keď chceme doručiť multicastovú správu staniciam na lokálnej sieti, musíme poznať ich MAC adresu, to však v prípade multicastu nie je jednoznačné. Najjednoduchším spôsobom je broadcast na adresu ff:ff:ff:ff:ff:ff, túto správu by však dostala každá stanica. Preto bola vyvinutá technika mapovania IP multicastovej adresy na MAC multicastovú adresu [9][14]. MAC adresa

sa skladá z dvoch častí, prvá je kód výrobcu (24 bitov), druhá je kód zariadenia (24 bitov). V prípade multicastu, kód výrobcu je vymenený na špeciálny kód 01:00:5e, ďalej platí, že 25. bit má vždy hodnotu 0. Pretože všetky multicastové IP adresy binárne sa začínajú číslicami 1110, je treba namapovať len ostávajúcich 28 bitov IP adresy na 23 bitov MAC adresy. Vykonáva sa to tak, že najnižších 23 bitov IP adresy sú skopírované do MAC adresy bez zmeny. Z toho vyplýva, že zariadenie na linkovej vrstve nevie jednoznačne zistiť, či on je adresátom, musí to potvrdiť aj vyššia vrstva (obr. 4-2).



Obr. 4-2 Princíp vytvorenia multicastovej MAC adresy

4.4 Multicastové protokoly

Multicastové protokoly môžeme rozdeliť na dve hlavné skupiny:

- protokoly, ktoré sa používajú medzi koncovými užívateľmi a smerovačom (napr. IGMP – Internet Group Management Protocol),
- protokoly, ktoré sa používajú na komunikáciu medzi smerovačmi (napr. PIM – Protocol Independent Multicast) [9].

Multicastový prenos môžeme ďalej rozdeliť podľa toho, ako je zdroj v skupine definovaný:

- ASM (any source multicast) – nie je v skupine pevne dané, kto bude zdroj a kto príjemca, spravidla sa tieto úlohy striedajú. Takto vytvorený systém je pomerne zložitý, má aj určité nevýhody: prijímač nemôže stanoviť, od ktorých zdrojov chce a od ktorých nechce prijímať dáta, do skupiny môže vysielať ktorýkoľvek člen skupiny.
- SSM (specific source multicast) – zdroj dát je presne špecifikovaný unicastovou IP adresou, ale v jednej skupine sa môže vyskytovať viac zdrojov. Príjemca v tomto prípade sa môže rozhodnúť od ktorého chce prijímať dáta [9].

4.4.1 IGMP (Internet Group Management Protocol)

Základnou úlohou protokolu IGMP je umožniť prístup koncovým staniciam do multicastovej skupiny [9]. Komunikácia prebieha medzi koncovými uzlami a smerovačom, na ktorý sú pripojené. Správy sa na internet nedostanú kvôli nastavenej hodnoty TTL=1.

IGMPv1

Prvá verzia protokolu IGMP obsahuje dva typy správ:

- *Membership query*,
- *Membership report*.

Smerovač pravidelne posiela správy *Membership query*, stanice potom odpovedajú správou *Membership report* pre každú skupinu zvlášť, do ktorej sa chcú pripojiť. V prípade, že stanica sa chce odpojiť od skupiny, nereaguje na správy *Membership query*, neoznamuje to priamo smerovaču [9][14].

Častým prípadom je, že existuje viac prijímačov na jednej koncovej sieti, vtedy by stanice museli posielať odpovede na *Membership query* hromadne, tým by mohli zahltiť linku [9]. Smerovača však nezaujíma počet staníc, ktoré sa chcú pripojiť do určitej skupiny, ale či existuje aspoň jedna stanica, teda či má do koncovej siete posielať multicastové dáta. Preto stanice neodpovedajú ihneď po obdržaní správy *Membership query*, ale nastavia svoj časovač na náhodnú hodnotu (obmedzený interval). Komu skoršie vyprší časovač, ten odpovie. Ostatné stanice tiež dostanú túto odpoveď a svoj časovač zase nastavia na náhodnú hodnotu atď. Tým spôsobom sú odpovede *Membership report* čo najviac rozložené medzi dvoma výzvami *Membership query*.

IGMPv2

Je to najviac rozšírená verzia protokolu v dnešnej dobe, obsahuje nasledujúce správy a rozšírenia:

- *Membership query* – dotaz na všetky prijímané skupiny, bol rozšírený s dotazom len na konkrétnu skupinu,
- *Version 2 Membership report* – je to nová verzia správy, umožňuje prihlásenie do skupiny aj bez výzvy od smerovača,
- *Leave group* – umožňuje okamžité odpojenie od skupiny,
- *Version 1 Membership report* – je to správa z predchádzajúcej verzie, zabezpečuje kompatibilitu.

Hlavným rozšírením je možnosť okamžitého odpojenia od skupiny, ďalej správa *Version 2 Membership report* umožňuje pripojenie do skupín aj bez výzvy od smerovača [9][14]. V prípade, že na koncovej sieti existuje viac staníc, ktoré sa chcú prihlásiť do určitej skupiny, podobne ako u protokolu IGMPv1, nastavia svoj časovač na náhodnú hodnotu. Odpoveď ale pošle len stanica, ktorej časovač vyprší skôr, ostatné už duplicitné správy neposielajú.

IGMPv3

Protokol IGMPv3 obsahuje nasledujúce správy:

- *Membership query,*
- *Version 3 Membership report,*
- *Version 2 Membership report,*
- *Version 1 Membership report,*
- *Version 2 Leave group,*

Hlavným prínosom je rozšírená správa *Version 3 Membership report*, ktorá umožňuje filtrovanie zdrojov v rámci jednej skupiny [9][14]. V prípade teda, že existuje viac zdrojov v skupine, stanice si môžu zvoliť, od koho budú a od koho nebudú prijímať dáta v rámci skupiny. Správa *Version 3 Membership report* je posielaná na multicastovú adresu, na ktorej poslúchajú len smerovače, ktoré podporujú protokol IGMPv3. Ostatné stanice teda odpoveď nedostanú, preto musia na správu *Membership query* vždy reagovať.

4.4.2 IGMP snooping

V predchádzajúcej časti bolo predstavené, ako fungujú smerovače v rámci lokálnej siete z pohľadu multicastového prenosu. Keď chceme zabezpečiť optimálny prenos, musíme rozšíriť podporu multicastu do určitej miery aj na prepínače. Dnešné zariadenia sú schopné analyzovať rámce, konkrétne záhlavie IGMP a podľa toho sa rozhodovať o výstupnom portu. Táto technika sa nazýva *IGMP snooping* [9]. Bez toho by prepínač musel kopírovať rámce na všetky rozhrania okrem toho, na ktorom aktuálny rámec prijal. Bolo by to neefektívne a v určitých prípadoch by mohol viesť aj k zablokovaniu linky. Prepínače sú schopné ďalej rozpoznať správy *Membership report*, ktoré posielajú len k smerovaču.

4.4.3 Distribučné stromy

Doteraz sme sa zaoberali s multicastovým prenosom z pohľadu koncových sietí, dáta sa však musia dostať aj na krajné smerovače. Prenos medzi smerovačmi sa deje na základe multicastových smerovacích protokolov, ktoré vytvárajú tzv. distribučné stromy, ktoré môžeme rozdeliť na stromy najkratších ciest (obr. 4-3) a na zdieľané stromy (obr. 4-4) [9][14].

Shortest Path Tree (SPT) – Strom najkratších ciest

Koreňom stromu je vždy zdroj dát, respektíve smerovač, na ktorý je zdroj pripojený a cesta od zdroja k príjemcom je vždy najkratšia [9]. Z toho vyplýva, že pre rôzne zdroje musia existovať stromy zvlášť.

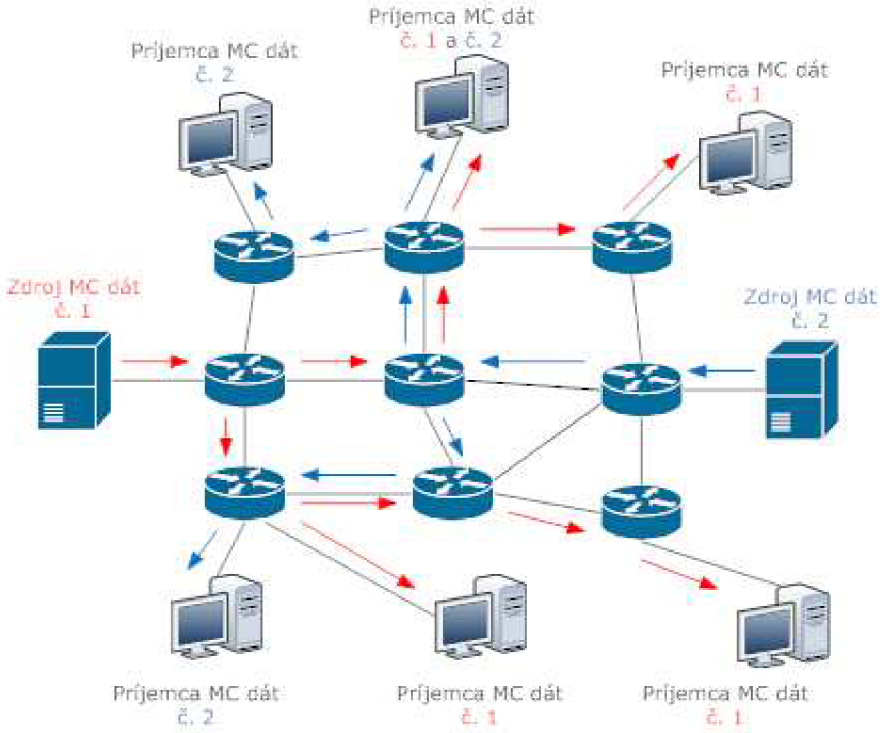
Shared Tree – Zdieľaný strom

Koreňom stromu je tzv. *Rendezvous point* (RP), ktorý je smerovač s veľkým výkonom. Jednotlivé zdroje posielajú svoje dáta tomuto uzlu, ktorý sa stará o doručení. Systém potrebuje len jeden strom aj v prípade viac zdrojov, doručenie sa však nevykonáva vždy najoptimálnejšou cestou [9]. Umiestnenie RP je teda kritické z pohľadu správneho fungovania. V prípade, že krajný smerovač má prijímateľa na koncovej sieti, oznamuje to smerovači RP unicastovou správou, do ktorej sa postupne uloží vykonaná cesta. Tým spôsobom vzniká strom relatívne ľahko na základe unicastových smerovacích protokolov.

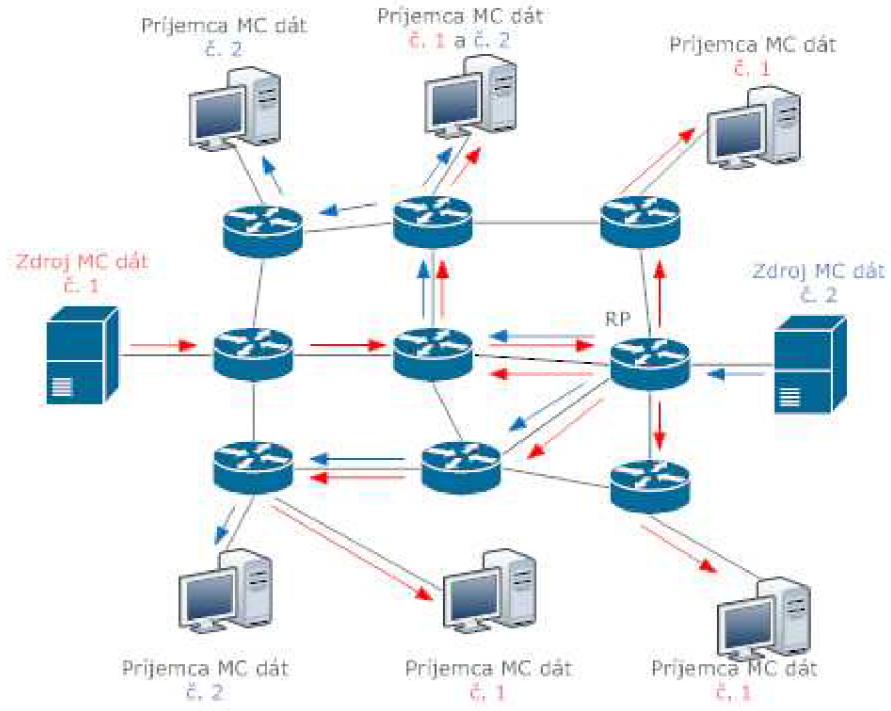
Reverse Path Forwarding

Je to mechanizmus, ktorý zabraňuje vzniku slučiek v priebehu multicastového prenosu. Smerovače kontrolujú smer (rozhranie), odkiaľ prichádzajú pakety a porovnávajú to s informáciami v smerovacej tabuľke [9]. V prípade, že paket bol prijatý cez rozhranie, ktoré nevedie naspäť k zdroju, smerovač paket zahodí. Zdrojom môže byť v prípade zdieľaného

stromu smerovač RP, alebo v prípade stromu najkratších ciest smerovač, na ktorý je zdrojová stanica pripojená.



Obr. 4-3 Strom najkratších ciest s dvoma zdrojmi dát a dvoma skupinami príjemcov [9]



Obr. 4-4 Zdieľaný strom s dvoma zdrojmi dát a dvoma skupinami príjemcov [9]

4.4.4 PIM (Protocol Independent Multicast)

Je to najpoužívanejší multicastový smerovací protokol, jeho hlavnou úlohou je vytvorenie čo najlepšieho distribučného stromu. Na zisťovanie topológie nepoužíva žiadne mechanizmy, ale informácie od unicastových smerovacích protokolov [9]. Tento protokol je funkčný len v rámci jedného autonómneho systému (AS), na komunikáciu medzi AS bol vyvinutý protokol MSDP (Multicast Source Discovery Protocol). Protokol PIM môže fungovať v móde DM (Dense Mode) alebo SM (Sparse Mode).

PIM-DM (Dense Mode)

Tento režim predpokladá, že na sieti existuje veľa prijímačov, preto na začiatku je celá sieť zaplavená multicastovými správami [9]. V prípade, že sa na niektorej koncovej sieti nenachádza príjemca, príslušný smerovač pošle koreňu správu *prune* (je treba posielať periodicky), tým sa odpojí od distribučného stromu. S týmto módom je možné vytvoriť len strom najkratších ciest a oplatí sa používať, keď prijímače sú v topológii blízko k sebe. V praxi sa v dnešnej dobe veľmi nepoužíva.

PIM-SM (Sparse Mode)

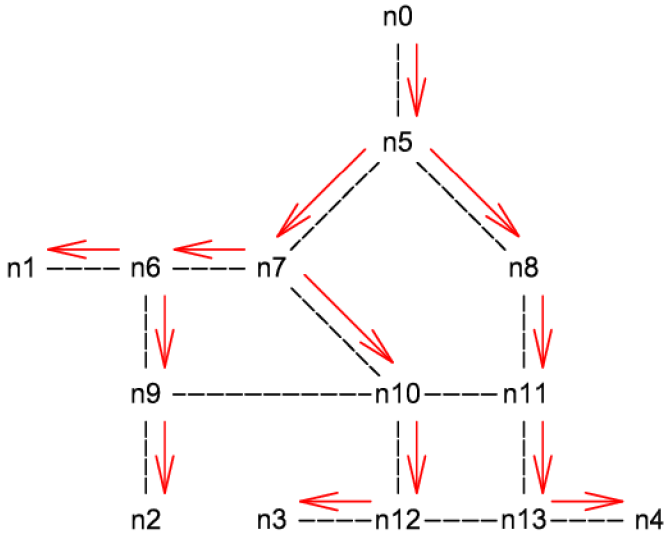
Tento režim funguje opačne, než PIM-DM. Na začiatku každý smerovač, ktorý má na koncovej sieti prijímača, pošle koreňu správu *join*, tým sa pripojí k distribučnému stromu [9]. Túto správu je treba tiež periodicky opakovať, v prípade, že sa tak nestane, smerovač je odpojený od stromu – je to možné vykonať aj priamo so správou *prune*. Tento mód slúži primárne na vytvorenie zdieľaného stromu, ale umožňuje aj vytvorenie stromu najkratších ciest. V dnešnej dobe je najpoužívanejším multicastovým smerovacím protokolom.

5 Úloha 2: Srovnání unicastového a multicastového přenosu

V této laboratorní úloze bude vytvořena síť s 14 uzly, které budou propojeny linkou bod-bod. Pro generování provozu bude použita aplikace `OnOffApplication`, pro příjem dat `PacketSink`. Uzel n0 bude posílat pakety uzlům n1, n2, n3 a n4 (obr. 5-1) nejprve unicastom, pak multicastom. Komunikace bude zajištěna v případě unicastu globálním směrováním (`Ipv4GlobalRoutingHelper`), v případě multicastu statickým směrováním (`Ipv4StaticRoutingHelper`). Na závěr bude provedena analýza zatížení linky mezi uzly n0 a n5 v případě obou typů přenosu pomocí trasovacího systému a programu NetAnim.

5.1 Topologie

Topologie úlohy je vidět obr. 5-1, uzly n0 až n4 představují koncové stanice, uzly n5 až n13 směrovače, jednotlivá zařízení jsou propojeny pomocí linek typu bod-bod. Červené šipky ukazují směr posílání paketů v případě multicastu.



Obr. 5-1 Zapojenie úlohy

5.2 Vytvoření modelu sítě

Připravený soubor *multicast.cc* **zkopírujte do složky** *Scratch* **a nastavte, aby bylo možné spustit z prostředí Eclipse (na začátku není nutné nic doplňovat).** V tomto souboru jsou připraveny moduly, jmenné prostory, globální a lokální proměnné, které budeme používat. Logování je nastaveno pro aplikaci `OnOffApplication` a `PacketSink`. Dále jsou vytvořeny uzly, spojení mezi nimi a jsou přiděleny IP adresní rozsahy. Síťová zařízení (net devices) a rozhraní (interfaces) byly pojmenovány podle toho, které uzly spojují pomocí linky bod-bod (můžete si to prostudovat v předpřipraveném souboru). Na konci souboru je nastaveno

spuštění a ukončení simulátoru. Soubor obsahuje také komentář, kde máte různé části kódu doplnit.

5.2.1 Nastavení pohybového modelu

Uzly se v této úloze pohybovat nebudou, přesto je nutné nastavit pohybový model kvůli výstupu do programu NetAnim. Použijeme tedy model pro stacionární uzly, `ConstantPositionMobilityModel`. **Zdrojový kód zkopírujte na začátek hlavní funkce, kde je to označeno (definice pohybového modelu).**

```
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.InstallAll();
```

5.2.2 Nastavení směrování

Jak to již bylo zmíněno v předchozích úlohách, uzly s protokolem IPv4 při vycházejícím nastavení fungují jako směrovače se směrovacím protokolem OSPF, to se nazývá globální směrování. Zdrojový kód na jeho povolení je následující:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

V průběhu psaní této práce nejsou klasické multicastové směrovací protokoly dosud podporovány, proto budeme muset nastavit statické cesty pro multicastový provoz. Na začátku uložíme adresu zdroje (n0) a adresu multicastové skupiny (225.1.2.4) do proměnných `multicastSource` a `multicastGroup`, neboť je budeme používat vícekrát. Dále vytvoříme objekt `multicastRouting`, pomocí kterého nastavíme statické cesty v směrovacích tabulkách uzlů podobně jako statické IPv6 cesty v kapitole 3.2.6.

První cestu definujeme jako defaultní pro všechny multicastové adresy na uzlu n0 (zdroj dat) pomocí funkce `SetDefaultMulticastRoute`. Vstupním parametrem je ukazatel na uzel, na kterém chceme cestu nastavit a výstupní síťové zařízení tohoto uzlu (net device). Jak to již bylo zmíněno na začátku úkoly, síťová zařízení a rozhraní byly pojmenovány podle určité logiky. V našem případě pomocí zápisu `netDevices0_5.Get(0)` dostaneme síťové zařízení uzlu n0. Kdyby vstupní hodnota byla 1, dostali bychom síťové zařízení uzlu n5.

V případě, že chceme pakety při průchodu určitými uzly duplikovat a poslat dále přes několik rozhraní, musíme příslušná síťové zařízení spojit do jedné třídy. K tomu slouží např. objekt `outputDevices_n5`, do které byly uloženy síťové zařízení, přes které jsou uzly n7 a n8 z uzlu n5 dostupné. Stejným způsobem byly vytvořeny objekty i pro uzly n7, n6 a n12.

Zdrojové kódy uvedené v této podkapitole postupně zkopírujte na místo, které je označeno jako definice unicastového a multicastového směrování.

```
Ipv4Address multicastSource("10.1.1.1");
Ipv4Address multicastGroup("225.1.2.4");

Ipv4StaticRoutingHelper multicastRouting;
multicastRouting.SetDefaultMulticastRoute(nodes.Get(0),netDevices0_5.Get(0));
```

```
NetDeviceContainer outputDevices_n5;
outputDevices_n5.Add(netDevices5_7.Get(0));
outputDevices_n5.Add(netDevices5_8.Get(0));

NetDeviceContainer outputDevices_n7;
outputDevices_n7.Add(netDevices7_6.Get(0));
outputDevices_n7.Add(netDevices10_7.Get(1));

NetDeviceContainer outputDevices_n6;
outputDevices_n6.Add(netDevices6_1.Get(0));
outputDevices_n6.Add(netDevices6_9.Get(0));

NetDeviceContainer outputDevices_n12;
outputDevices_n12.Add(netDevices12_3.Get(0));
outputDevices_n12.Add(netDevices13_12.Get(1));
```

Dalším krokem je nastavení multicastových cest na směrovačích (n5 až n13). Použijeme funkci `AddMulticastRoute`, jejíž vstupní parametry jsou následující:

- ukazatel na uzel, na kterém chceme nastavit statickou cestu,
- IP adresa zdroje multicastových dat,
- IP adresa multicastové skupiny,
- vstupní síťové zařízení uzlu, který nastavujeme,
- výstupní síťové zařízení uzlu, který nastavujeme (případně skupina síťových zařízení).

Např. na uzlu n5 bylo nastaveno vstupní síťové zařízení (`netDevices0_5.Get(1)`) přes které je zdroj dat (n0) dostupný. Výstupním zařízením je skupina síťových zařízení `outputDevices_n5`, která byla popsána výše. Podobným způsobem byly definovány statické cesty pro všechny směrovače.

```
//n5
multicastRouting.AddMulticastRoute(nodes.Get(5), multicastSource,
      multicastGroup, netDevices0_5.Get(1), outputDevices_n5);
//n6
multicastRouting.AddMulticastRoute(nodes.Get(6), multicastSource,
      multicastGroup, netDevices7_6.Get(1), outputDevices_n6);
//n7
multicastRouting.AddMulticastRoute(nodes.Get(7), multicastSource,
      multicastGroup, netDevices5_7.Get(1), outputDevices_n7);
//n8
multicastRouting.AddMulticastRoute(nodes.Get(8), multicastSource,
      multicastGroup, netDevices5_8.Get(1), netDevices8_11.Get(0));
//n9
multicastRouting.AddMulticastRoute(nodes.Get(9), multicastSource,
      multicastGroup, netDevices6_9.Get(1), netDevices9_2.Get(0));
//n10
multicastRouting.AddMulticastRoute(nodes.Get(10), multicastSource,
      multicastGroup, netDevices10_7.Get(0), netDevices12_10.Get(1));
//n11
multicastRouting.AddMulticastRoute(nodes.Get(11), multicastSource,
      multicastGroup, netDevices8_11.Get(1), netDevices11_13.Get(0));
//n12
multicastRouting.AddMulticastRoute(nodes.Get(12), multicastSource,
      multicastGroup, netDevices12_10.Get(0), netDevices12_3.Get(0));
```

```
//n13
multicastRouting.AddMulticastRoute(nodes.Get(13), multicastSource,
      multicastGroup, netDevices11_13.Get(1), netDevices13_4.Get(0));
```

5.2.3 Nastavení aplikací

V této části nastavíme nejprve unicastový provoz z uzlu n0 k uzlům n1 až n4. Pro generování provozu bude použita `OnOff` aplikace, z toho budeme muset vytvořit čtyři instance, pro každého příjemce jednu. Před tím ještě vytvoříme jednu aplikaci `PacketSink` pro úspěšný příjem posílaných dat, kterou budeme moci nainstalovat na všechny přijímací uzly.

```
PacketSinkHelper sink("ns3::UdpSocketFactory",
      InetSocketAddress(Ipv4Address::GetAny (), 9));
```

Následuje nastavení aplikace pomocí `OnOffHelper`, ako príklad bude popísaná len prvá aplikácia. Objekt `onOffUnicast1` představuje aplikaci, která posílá data uzlu n1 unicastovým přenosem. Nejprve vybereme požadovaný protokol na transportní vrstvě, v našem případě je to UDP, pak zadáme cílové rozhraní a cílový port. Zápis `interfaces6_1.GetAddress(1)` představuje rozhraní uzlu n1, logika je podobná jako u síťových zařízení, která byla vysvětlena v předchozí podkapitole. V případě vstupní hodnoty 0 bychom dostali síťové rozhraní uzlu n6, přes které je n1 dostupný. Dále nastavíme velikost datové části paketů na 1024 B a přenosovou rychlost na 1 Mb/s (proměnná `appBitRate` a `packetSize` je deklarována na začátku hlavní funkce).

Hlavní vlastnost aplikace `OnOff`, která plyne i z jejího pojmenování je, že pakety v daném okamžiku buď posílá, nebo ne. Tyto dva stavy (ON-zapnuto a OFF-vypnuto) při vycházejícím nastavení střídají pravidelně, což není vhodné pro náš případ. Proto atribut `OffTime` nastavíme na hodnotu 0, tím dosáhneme, aby aplikace posílala data neustále. Následně vytvoříme objekt `unicastAppCont1` a pomocí něj aplikaci nainstalujeme na uzel n0. Na začátku zdrojového kódu můžete najít předem připravené proměnné `startTime` a `endTime`, které udávají, kdy se aplikace mají odstartovat a skončit (jsou nastaveny na hodnotu 2 s a 10 s). Nakonec nainstalujeme ještě aplikaci `PacketSink` na uzel n1, aby data byly úspěšně přijaty. Zdrojový kód pro vytvoření aplikací je následující:

```
OnOffHelper onOffUnicast1("ns3::UdpSocketFactory",
      InetSocketAddress(interfaces6_1.GetAddress(1), 9));
onOffUnicast1.SetAttribute("PacketSize", UintegerValue(1024));
onOffUnicast1.SetAttribute("DataRate", StringValue (appBitRate));
onOffUnicast1.SetAttribute("OffTime", StringValue
      ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer unicastAppCont1 = onOffUnicast1.Install(nodes.Get(0));
unicastAppCont1.Start(Seconds(startTime));
unicastAppCont1.Stop(Seconds(endTime));
unicastAppCont1 = sink.Install(nodes.Get(1));

OnOffHelper onOffUnicast2("ns3::UdpSocketFactory",
      InetSocketAddress(interfaces9_2.GetAddress(1), 9));
onOffUnicast2.SetAttribute("PacketSize", UintegerValue(1024));
onOffUnicast2.SetAttribute("DataRate", StringValue(appBitRate));
```

```
onOffUnicast2.SetAttribute("OffTime", StringValue
      ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer unicastAppCont2 = onOffUnicast2.Install(nodes.Get(0));
unicastAppCont2.Start(Seconds(startTime));
unicastAppCont2.Stop(Seconds(endTime));
unicastAppCont2 = sink.Install(nodes.Get(2));

OnOffHelper onOffUnicast3("ns3::UdpSocketFactory",
      InetSocketAddress(interfaces12_3.GetAddress(1), 9));
onOffUnicast3.SetAttribute("PacketSize", UintegerValue(1024));
onOffUnicast3.SetAttribute("DataRate", StringValue(appBitRate));
onOffUnicast3.SetAttribute("OffTime", StringValue
      ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer unicastAppCont3 = onOffUnicast3.Install(nodes.Get(0));
unicastAppCont3.Start(Seconds(startTime));
unicastAppCont3.Stop(Seconds(endTime));
unicastAppCont3 = sink.Install(nodes.Get(3));

OnOffHelper onOffUnicast4("ns3::UdpSocketFactory",
      InetSocketAddress(interfaces13_4.GetAddress(1), 9));
onOffUnicast4.SetAttribute("PacketSize", UintegerValue(1024));
onOffUnicast4.SetAttribute("DataRate", StringValue(appBitRate));
onOffUnicast4.SetAttribute("OffTime", StringValue
      ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer unicastAppCont4 = onOffUnicast4.Install(nodes.Get(0));
unicastAppCont4.Start(Seconds(startTime));
unicastAppCont4.Stop(Seconds(endTime));
unicastAppCont4 = sink.Install(nodes.Get(4));
```

Po definování unicastového provozu následuje multicastový, použijeme k tomu také aplikaci `OnOff`. Nastavení aplikace probíhá stejně, namísto adresy rozhraní uzlu však nastavíme adresu multicastové skupiny. Níže uvedený zdrojový kód můžete do souboru zkopírovat, ale v tomto případě tyto řádky ještě zakomentujte, budeme je potřebovat později.

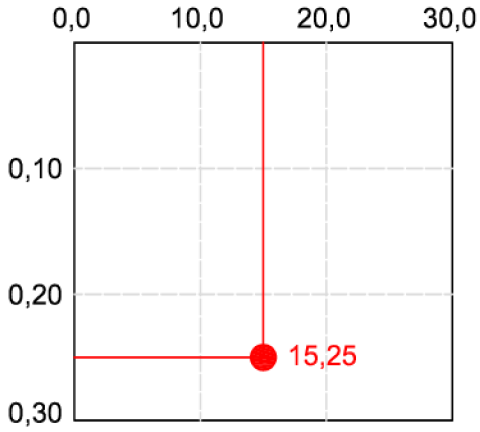
```
OnOffHelper onOffMulticast("ns3::UdpSocketFactory",
      Address(InetSocketAddress(multicastGroup, 9)));
onOffMulticast.SetAttribute("PacketSize", UintegerValue(1024));
onOffMulticast.SetAttribute("DataRate", StringValue (appBitRate));
onOffMulticast.SetAttribute("OffTime", StringValue
      ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer multicastAppCont = onOffMulticast.Install(nodes.Get(0));
multicastAppCont.Start(Seconds(startTime));
multicastAppCont.Stop(Seconds(endTime));
multicastAppCont = sink.Install(nodes.Get(1));
multicastAppCont = sink.Install(nodes.Get(2));
multicastAppCont = sink.Install(nodes.Get(3));
multicastAppCont = sink.Install(nodes.Get(4));
```

5.2.4 Nastavení grafického výstupu do programu NetAnim

Na grafické zobrazení simulace se v NS-3 používá program NetAnim, který není součástí NS-3, je nutné to nainstalovat zvlášť, v této práci byla použita verze 3.105. Pracuje se soubory *.xml*, podle kterého umožňuje znázornit např.:

- pakety procházející přes drátovou nebo bezdrátovou linku,
- časovou linii (timeline) paketů,
- statistiku o pohybu uzlů,
- směrovací tabulky v určitém čase.

V předem vytvořeném souboru je již deklarována proměnná `anim` jako pointer, který slouží k vytvoření výstupního souboru pro NetAnim. Na základě toho si vytvoříme nový objekt, jako vstupní parametr uvedeme jméno *.xml* souboru bez přípony. Dalším řádkem povolíme logování informací o paketech (metadata) do *.xml* souboru. Pomocí funkce `SetConstantPosition` můžeme nastavit jednotlivě pozici uzlů ve virtuálním prostředí. Vstupním parametrem je uzel, jehož pozici chceme nastavit dále souřadnice x a y (obr. 5-2). V předpřipraveném souboru jsou již pozice uzlů nastavené, **proto příslušný řádek v níže uvedeném zdrojovém kódu slouží pouze k vysvětlení, tento řádek do souboru nezkopírujte**.



Obr. 5-2 Souřadnicový systém v NetAnim – bod na souřadnicích x=15 a y=25

Barvu uzlů můžeme změnit funkcí `UpdateNodeColor`, vstupními parametry jsou ukazatel na požadovaný uzel a hodnota jeho barvy v modelu RGB (parametry je nutné zadávat v pořadí RBG). Další funkce `UpdateNodeDescription` slouží ke změně popisu uzlů, vstupními parametry jsou zase ukazatel na požadovaný uzel a jeho popis. Poslední funkce, která v této úloze bude představena je `UpdateNodeSize`, která změní velikost ikonky při zobrazení. První vstupní parametr je „globální“ pořadové číslo uzlu, není to tedy ukazatel na uzel, jako v předchozích případech. V našem případě jsme vytvořili všechny uzly pomocí jedné třídy `NodeContainer`, tedy globální pořadové číslo uzlů se rovná pořadovému číslu v rámci objektu `nodes`. Další parametry představují zvětšení ve směru osy x a y.

V níže uvedeném zdrojovém kódu je vidět nastavení uzlu n0 (zdroj dat). Ten je umístěn na souřadnicích x=30 a y=0, má modrou barvu a byla přidána popiska „Server“. Jeho velikost je dvojnásobná ve směru osy x a y. **Podle uvedeného příkladu a tab. 5-1 si samostatně nastavte barvu, popis a velikost ostatních uzlů, případně si je můžete nastavit podle vlastních představ (souřadnice uzlů ve virtuálním prostředí již byly předem nastaveny). Uvedený zdrojový kód patří na místo, které je označeno jako definice výstupu pro program NetAnim.**

```
anim = new AnimationInterface("unicast");

anim->EnablePacketMetadata (true);
anim->SetConstantPosition (nodes.Get(0), 30, 0);
anim->UpdateNodeColor(nodes.Get(0), 0, 255, 0);
anim->UpdateNodeDescription(nodes.Get(0), "Server");
anim->UpdateNodeSize(0, 2, 2);
```

Tab. 5-1 Hodnoty pro nastavení vzhledu uzlů

Uzel	Parametr	Hodnota
n1 až n4	Barva	Červená
	Popis	Klient 1 až 4
	Velikost	Dvojnásobná ve směru osy x a y
n5 až n13	Barva	Černá
	Velikost	Dvojnásobná ve směru osy x a y

5.2.5 Měření zatížení linky

V této části bude představen a popsán zdrojový kód pro měření zatížení linky mezi uzly n0 a n5. Bude k tomu použitý trasovací systém, výsledky budou zobrazeny pomocí grafu jako závislost bitové rychlosti v čase.

Níže uvedeným zdrojovým kódem se připojíme na trasovací systém, budeme pozorovat následující události na uzlu n0:

- `Ipv4L3Protocol/Tx` – indikuje posílání paketů ze síťové vrstvy na nižší vrstvu, v NS-3 konkrétně síťovému zařízení (net device) – vytvořená data,
- `PointToPointNetDevice/PhyTxEnd` – indikuje, že paket byl úspěšné přenesen přes přenosový kanál – úspěšně přenesená data,
- `PointToPointNetDevice/MacTxDrop` – indikuje, že paket byl zahozen před posíláním – zahozené data.

Uvedený zdrojový kód si zkopírujte před řádek, který spustí simulátor. Při kopírování si dejte pozor, cesta k trasovacímu zdroji (označená modrým písmem) musí být uvedena v jednom řádku bez zalomení.

```
Config::Connect("/NodeList/0/$ns3::Ipv4L3Protocol/Tx",
      MakeCallback(&ipv4_node0_tx));
Config::Connect("/NodeList/0/DeviceList/*/$ns3::PointToPointNetDevice/
PhyTxEnd",MakeCallback(&ipv4_node0_phyTx));
Config::Connect("/NodeList/0/DeviceList/*/$ns3::PointToPointNetDevice/
      MacTxDrop", MakeCallback(&ipv4_node0_macTxDrop));
```

V případě, že se nastane příslušná událost, je volána některá z funkcí `ipv4_node0_tx`, `ipv4_node0_phyTx` a `ipv4_node0_macTxDrop`, **které mají být definovány před hlavní funkcí** `main`. Níže jsou uvedeny tyto funkce, podmínky na začátku každé funkce slouží k filtrování nechtěných paketů (např. provoz generovanou směrovacími protokoly).

Proměnné `txAllDataSize`, `txTransferedDataSize`, `txDroppedDataSize` byly definovány jako globální a uloží se do nich množství vytvořených, přenesených a zahozených dat.

```
void ipv4_node0_tx(string path, Ptr<Packet const> packet, Ptr<Ipv4> ipv4,
unsigned int parameter) {
      if (packet->GetSize() >= 100) {
            txAllDataSize = txAllDataSize + packet->GetSize();
      }}

void ipv4_node0_phyTx (string path, Ptr<Packet const> packet) {
      if (packet->GetSize() >= 100) {
            txTransferedDataSize = txTransferedDataSize + packet->GetSize();
      }}

void ipv4_node0_macTxDrop (string path, Ptr<Packet const> packet) {
      if (packet->GetSize() >= 100) {
            txDroppedDataSize = txDroppedDataSize + packet->GetSize();
      }}
```

Abychom z těchto údajů mohli vytvořit graf, musíme tyto hodnoty v určitých časových intervalech zaznamenávat. Následující řádky pravidelně volají funkci `saveData` od začátku až po konec simulace, která slouží k uložení hodnot. Proměnnou `section`, která je definována na začátku zdrojového kódu jako globální, je možné nastavit, jak často (jemně) chceme údaje „vzorkovat“. **Následující kód vložte před řádek, který spustí simulátor.**

```
for(double time=0; time<endTime; time=time+section)
      Simulator::Schedule (Seconds(time), &saveData, time);
```

Na začátku zdrojového kódu jsou dále definovány tři objekty typu `Gnuplot2dDataset`, které slouží jako databáze a budeme do nich zaznamenávat příslušné hodnoty. Níže je uveden zdrojový kód funkce `saveData`, který pravidelně ukládá aktuální čas a přenosovou rychlost (protože množství dat je vydělen proměnnou `section`) dat do příslušné databáze. Na konci jsou proměnné, které obsahují množství dat, vynulováno. **Zdrojový kód umístěte před hlavní funkci.** Takto uložené hodnoty budou v jednotkách B/s, **upravte zdrojový kód tak, aby hodnoty byly uloženy v jednotkách Mb/s** (jde o jednoduché násobení a dělení).

```
void saveData(double time) {
      txAllDataset.Add(time,(double)txAllDataSize/section);
      txTransferredDataset.Add(time,(double)txTransferedDataSize/section);
      txDroppedDataset.Add(time,(double)txDroppedDataSize/section);

      txAllDataSize = 0;
      txTransferedDataSize = 0;
      txDroppedDataSize = 0;
}
```

Nakonec vytvoříme soubor *.plt* podobně jako v kapitole 3.2.9, v tomto případě však do něj uložíme hodnoty pro tři různé grafy. Pomocí funkce `SetTitle` nastavíme jméno pro každou sérii dat a pomocí funkce `SetStyle` styl vykreslení. Dále již známým způsobem nastavíme typ výstupního grafického souboru a popis os. Z estetických důvodů nastavíme rozsah

osy y funkcí `AppendExtra` (můžete si to v dalších částech úlohy zakomentovat nebo změnit dle potřeby). **Níže uvedené řádky vložte na konec zdrojového kódu před řádek** `return 0`.

```
string fileNameWithNoExtension = "unicast";
string graphicsFileName = fileNameWithNoExtension + ".png";
string plotFileName = fileNameWithNoExtension + ".plt";
string plotTitle = "Bitrate-unicast";

txAllDataset.SetTitle("Sent");
txAllDataset.SetStyle(Gnuplot2dDataset::LINES_POINTS);
txTransferredDataset.SetTitle("Transferred");
txTransferredDataset.SetStyle(Gnuplot2dDataset::LINES_POINTS);
txDroppedDataset.SetTitle("Dropped");
txDroppedDataset.SetStyle(Gnuplot2dDataset::LINES_POINTS);

Gnuplot plot(graphicsFileName);

plot.SetTitle(plotTitle);
plot.SetTerminal("png");
plot.SetLegend("Time [s]", "Bitrate [Mb/s]");
plot.AppendExtra ("set yrange [0:+1]");
plot.AddDataset(txAllDataset);
plot.AddDataset(txTransferredDataset);
plot.AddDataset(txDroppedDataset);
ofstream plotFile(plotFileName.c_str());
plot.GenerateOutput(plotFile);
plotFile.close();
```

5.3 Výsledky simulace s unicastovým přenosem

Před spuštěním simulace se přesvědčte, **že část kódu, která generuje multicastový provoz je zakomentována**, v této části se budeme zabývat s unicastovým přenosem. Také zkontrolujte, že následující proměnné (bitová rychlost aplikací, maximální přenosová rychlost linek, délka front) jsou nastaveny na správnou hodnotu:

```
appBitRate = "0.2Mbps";
linkBitRate = "1Mbps";
queueLength = "10";
```

Poznámka: délka fronty se v tomto případě udává pomocí počtu paketů.

5.3.1 Výstup z příkazového řádku

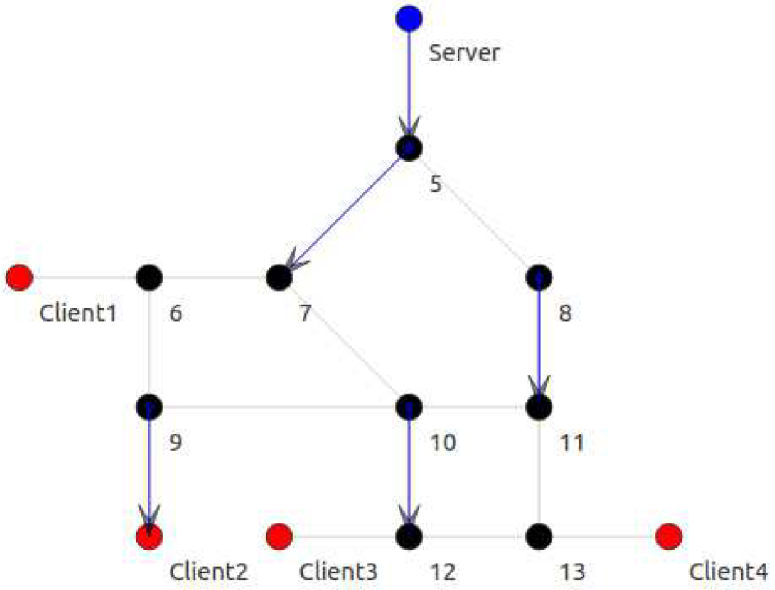
Níže je zobrazena část výstupu z příkazového řádku, jsou to logové zprávy aplikace `OnOffApplication` a `PacketSink`. Je vidět, že zdrojový uzel n0 poslal čtyři různé pakety klientům na port č. 9. Klienti paket dostali, je možné je rozlišit podle čísla zdrojového portu.

```
At time 9.98722s on-off application sent 1024 bytes to 10.1.11.2 port 9 total
Tx 199680 bytes
At time 9.98722s on-off application sent 1024 bytes to 10.1.14.2 port 9 total
Tx 199680 bytes
At time 9.98722s on-off application sent 1024 bytes to 10.1.7.2 port 9 total
Tx 199680 bytes
At time 9.98722s on-off application sent 1024 bytes to 10.1.5.2 port 9 total
Tx 199680 bytes
```

```
At time 9.98799s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 198656 bytes
At time 10.0069s packet sink received 1024 bytes from 10.1.1.1 port 49154
total Rx 198656 bytes
At time 10.0153s packet sink received 1024 bytes from 10.1.1.1 port 49155
total Rx 198656 bytes
At time 10.0237s packet sink received 1024 bytes from 10.1.1.1 port 49156
total Rx 198656 bytes
```

5.3.2 Grafický výstup programu NetAnim

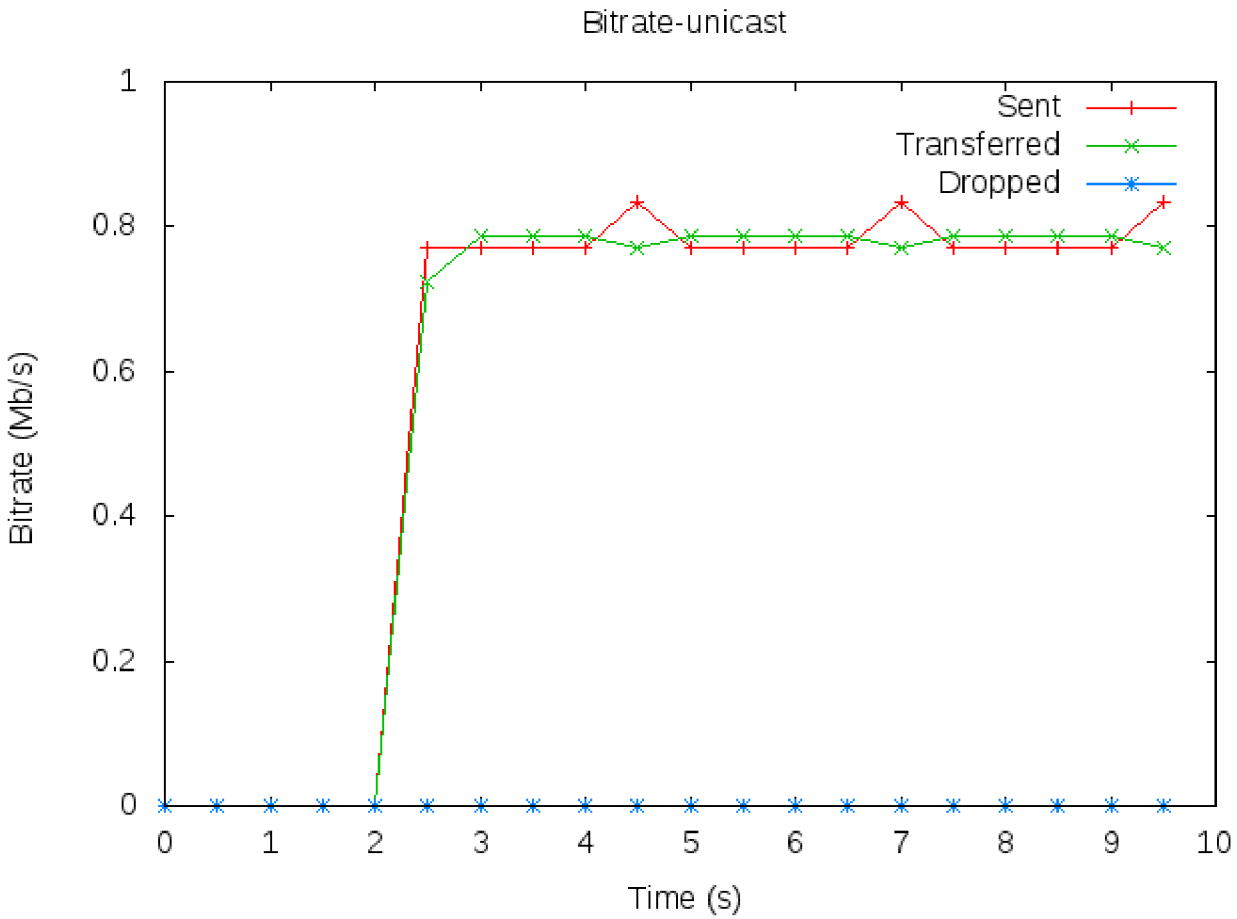
Po spuštění simulace se v kořenové složce NS-3 vytvoří soubor *unicast.xml*, otevřete ho v programu NetAnim. V případě, že jste parametry nastavili správně, měli byste vidět shodnou topologii, jako na obr. 5-3. Po spuštění animace je vidět, že uzel n0, označený jako Server, posílá data klientům 1 až 4. Jak již bylo zmíněno v kapitole 3.2.7, současný běh více UDP aplikací má za následek, že jeden druhého zpozdí. Nastalo se to i v tomto případě, tedy např. Client 4 dostane paket mnohem později, než Client 2 přesto, že byly generovány ve stejném čase a zmíněné uzly jsou stejně daleko od zdroje (počet skoků).



Obr. 5-3 Unicastový přenos v programu NetAnim

5.3.3 Grafický výstup programu Gnuplot

Po spuštění simulace se v kořenové složce také vytvořil soubor *unicast.plt*. Po vytvoření grafického souboru příkazem `gnuplot unicast.plt` byste měli dostat shodný graf jako na obr. 5-4. Červená čára představuje množství generovaných, zelená množství přenesených a modrá množství zahozených dat za sekundu. Jelikož přenosový kanál zvládne přenos s rychlostí maximálně 1 Mb/s a čtyři aplikace posílají data rychlostí 0,2 Mb/s (celkem 0,8 Mb/s), je zřejmé, že se k zahazování paketů v tomto případě nedochází.



Obr. 5-4 Rychlost posílání dat v případě unicastového přenosu na lince mezi uzly n0 a n5

5.4 Výsledky simulace s multicastovým přenosem

Spusťte simulaci ještě jednou, v tomto případě ale zakomentujte část zdrojového kódu, kde jsou definovány aplikace pro unicastový přenos a do zdrojového kódu dejte zpět řádky, které definují aplikaci pro multicastový přenos. Dále přejmenujte výstupní soubory pro NetAnim a Gnuplot (tři soubory s příponami *.plt*, *.png* a *.xml*). Můžete si také zdrojový kód uložit pod jiným názvem, neboť budeme ještě potřebovat i simulaci s unicastovým přenosem.

5.4.1 Výstup z příkazového řádku

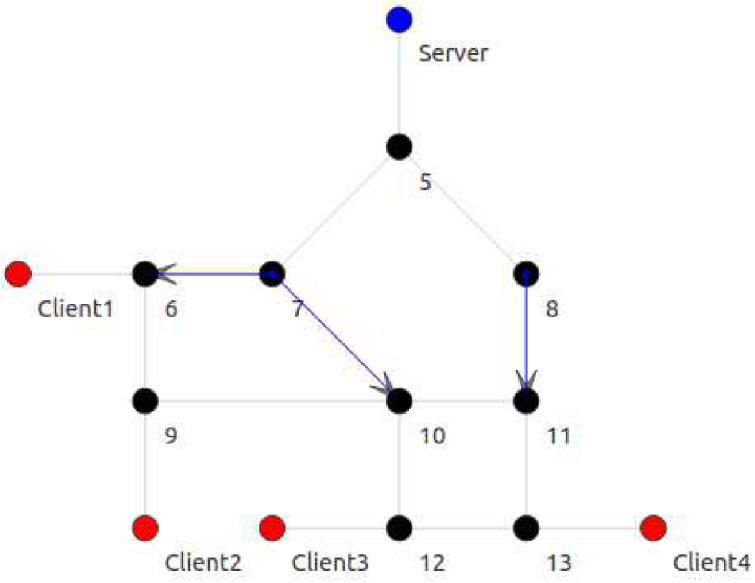
Níže je zobrazena část výstupu z příkazového řádku, jsou to podobné logové zprávy, jako v případě unicastového přenosu. Je vidět, že cílová adresa, na kterou uzel n0 posílá data je námi nastavena multicastová adresa. Je důležité si uvědomit, že uzel n0 posílá vždy jen jeden paket, jeho duplikát pak dostane každý klient. Klienty v tomto případě není možné rozlišit, správnost konfigurace je vidět z toho, že na každý posílaný paket odpovídá aplikace `PacketSink` čtyři krát (čtyři různé aplikace na klientech).

```
At time 9.9053s on-off application sent 1024 bytes to 225.1.2.4 port 9 total
Tx 197632 bytes
At time 9.90607s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 196608 bytes
```

```
At time 9.9165s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 196608 bytes
At time 9.9165s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 196608 bytes
At time 9.9165s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 196608 bytes
At time 9.94626s on-off application sent 1024 bytes to 225.1.2.4 port 9 total
Tx 198656 bytes
At time 9.94703s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 197632 bytes
At time 9.95746s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 197632 bytes
At time 9.95746s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 197632 bytes
At time 9.95746s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 197632 bytes
```

5.4.2 Grafický výstup programu NetAnim

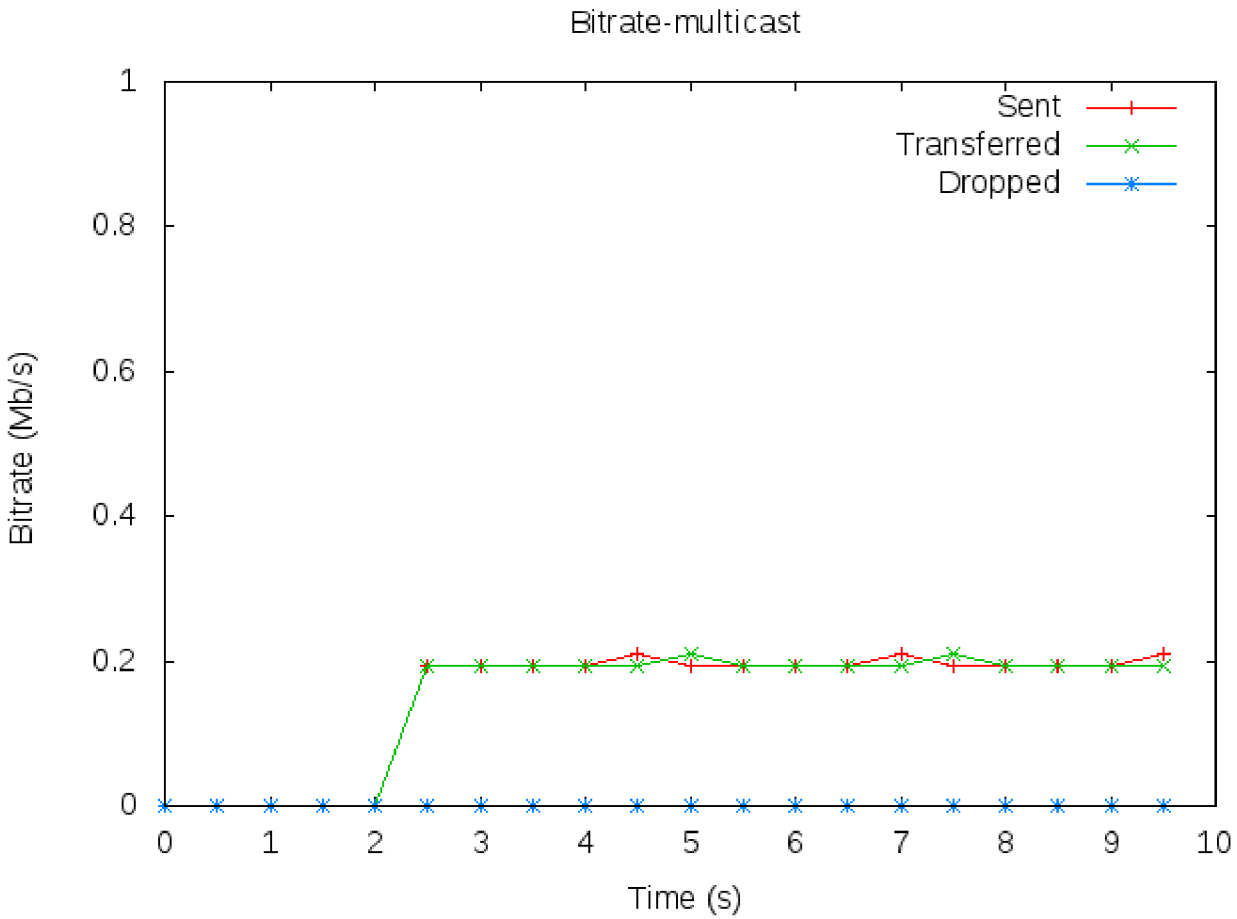
Otevřete generovaný *.xml* soubor, ve kterém je zachycen multicastový přenos v programu NetAnim. Měli byste vidět shodný výstup, jako na obr. 5-5. Průběh obou animací si samostatně porovnejte a rozdíly si poznamenejte.



Obr. 5-5 Multicastový přenos v programu NetAnim

5.4.3 Grafický výstup programu Gnuplot

Na obr. 5-6 je zobrazena množství přenesených dat za sekundu s multicastovým přenosem. Je vidět, že v porovnání s unicastovým provozem (obr. 5-4) mnohem méně zatěžoval linku. Je to kvůli tomu, že pakety jsou na cestě k příjemcům postupně duplikovány a zdroj posílá data vždy jen jednou nezávisle na počtu příjemců.



Obr. 5-6 Rychlost posílání dat v případě multicastového přenosu na lince mezi uzly n0 a n5

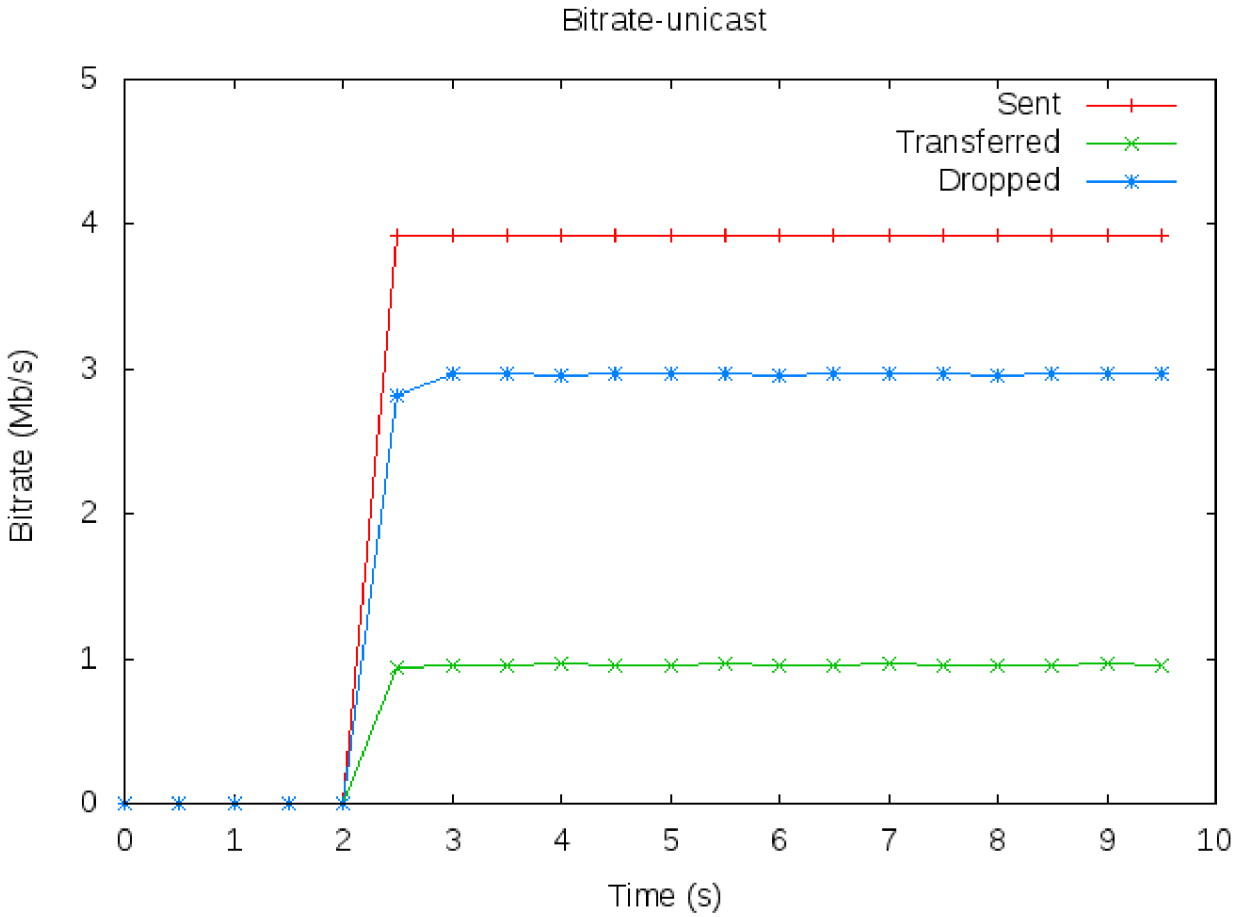
5.5 Kontrolní otázky

- Vysvětlete princip unicastového a multicastového přenosu. Porovnejte animace v programu NetAnim pro unicastový a multicastový přenos, měli byste spatřit dva zásadní rozdíly. Ve kterém případě je multicastový přenos mnohem efektivnější, než unicastový.
- Který protokol je využíván na transportní vrstvě při multicastového přenosu a proč? Jaké nevýhody tento protokol přináší?
- Vysvětlete princip vytvoření multicastové MAC adresy z multicastové IP adresy. Dá se na linkové vrstvě jednoznačně zjistit, zda stanice je adresátem nebo potřebujeme k tomu i vyšší vrstvy?
- Co jsou to distribuční stromy v případě multicastového přenosu a jaké typy distribučních stromů znáte? Vysvětlete jejich princip, výhody a nevýhody. Pomocí programu NetAnim zjistěte, jaký distribuční strom byl vytvořen manuálně v této úloze.
- Na co se používají multicastové protokoly IGMP a PIM, jaký je mezi nimi rozdíl?

5.6 Samostatné úkoly

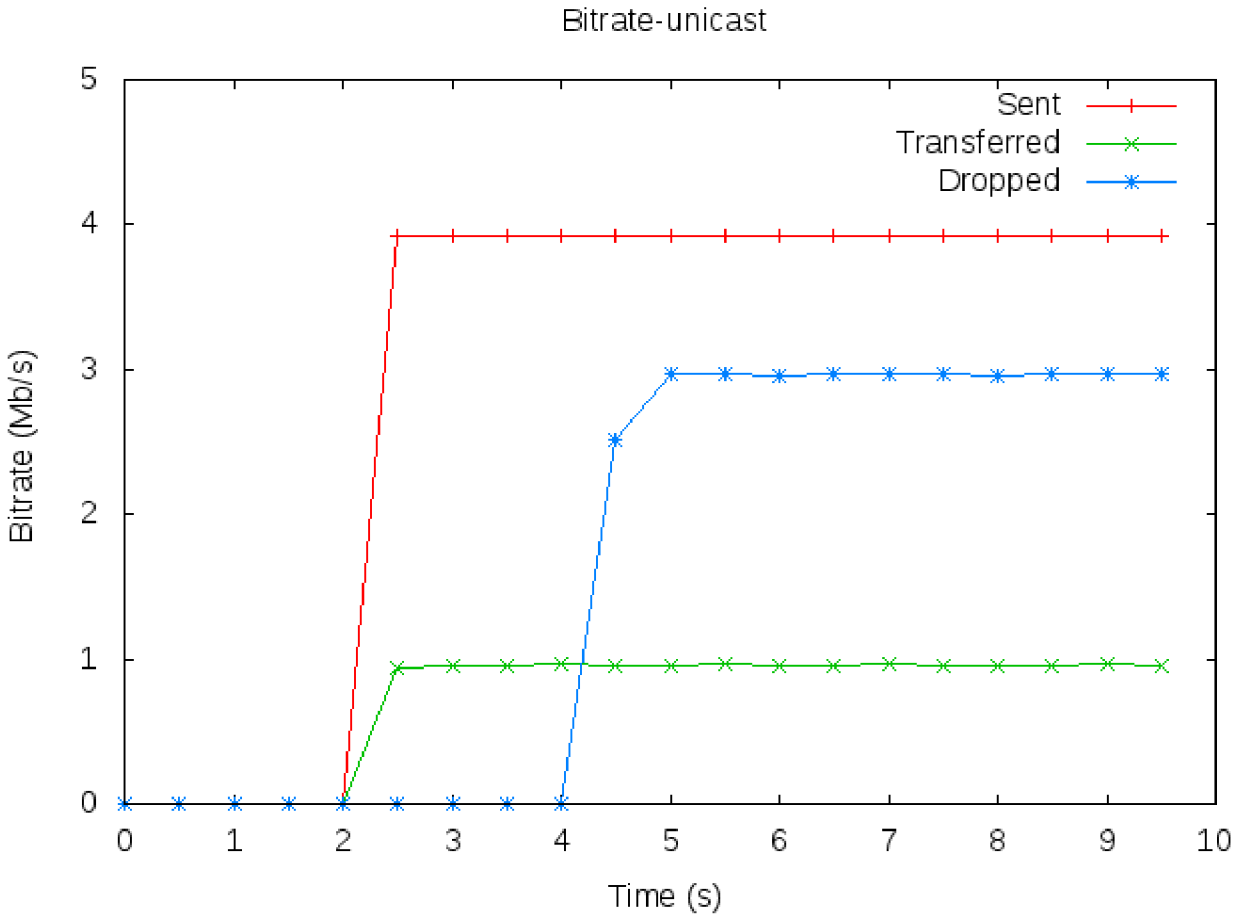
V této části úlohy se budeme zabývat případem, kdy je bitová rychlost dat generována aplikacemi větší, než přenosová rychlost linky.

- Nastavte simulaci pro posílání unicastových paketů dále nastavte přenosovou rychlost aplikací (`appBitRate`) na hodnotu 1 Mb/s (větší než 0,25 Mb/s). Dále zakomentujte řádek pro nastavení rozsahu osy y.
- Po spuštění simulace byste měli dostat shodný graf jako na obr. 5-7. Je vidět, že dochází k zahazování paketů ihned, protože délka fronty (`queueLength`) pro posílání je nastavena na malou hodnotu. Jaká je souvislost mezi bitovým rychlostmi poslaných, úspěšně přenesených a zahozených dat?



Obr. 5-7 Rychlost posílání dat v případě unicastového přenosu – přetížení linky

- Vypočtěte počet posílaných paketů (předpokládejte velikost 1024 B) za sekundu a počet paketů, který je schopen kanál přenést za sekundu. Podle těchto údajů nastavte délku fronty (`queueLength`) tak, aby k zahazování došlo až v době 4 s (2 s po odstartování aplikací). Poznámka: do proměnné `queueLength` je nutné uložit počet paketů, nikoliv velikost v bajtech. Po spuštění simulace byste měli dostat shodný graf jako na obr. 5-8.



Obr. 5-8 Rychlost posílání dat v případě unicastového přenosu – přetížená linka a zvýšená délka fronty

- Na konec si můžete vyzkoušet tyto nastavení i s multicastovým přenosem, přesvědčte se, že k zahazování paketů nedochází (ani v případě, že velikost fronty nastavíte na původní velikost).

6 ZAISTENIE KVALITY SLUŽIEB (QOS)

V nedávnej minulosti ešte stále veľa sieťových technológií, ktoré sú založené na prepínaní paketov a rámcov, spracoval všetky dátové jednotky rovnakým spôsobom. Tento prístup sa nazýva best-effort a na prvý pohľad sa môže zdať, že je to spravodlivá metóda. V skutočnosti to tak nie je, dva najväčšie nedostatky tohto prístupu sú nasledovné:

- Nie je možné rozoznať jednotlivé dátové toky a všetky dátové jednotky sa spracovávajú nezávisle od ostatných. Preto nie je možné na sieťach s náhodným prístupom rôzne dátové triedy uprednostniť alebo naopak, zatlačiť do pozadia.
- Jeden zdroj s agresívnym správaním je schopný zaplaviť sieť a tým vyčerpať dostupné sieťové zdroje (napr. obsadiť veľkú časť prenosovej kapacity, alebo zaplniť fronty aktívnych sieťových prvkov – smerovače) [15].

Z vyššie uvedených je jasné, že nedostatok sieťových zdrojov má rovnaký dopad na všetky dátové toky, má však veľmi odlišný dopad na služby, ktoré tieto dátové toky vytvorili [15]. Napr. pri prenose veľkého súboru aj najmenšia strata dát môže znehodnotiť celý súbor, oneskorenie alebo kolísanie oneskorenia (jitter) však nemá z hľadiska užívateľa rozoznateľný vplyv na prenos. Pri prenose audia alebo videa v reálnom čase sú parametre ako oneskorenie a jitter kritické, ale služba toleruje určitú mieru stratovosti. V riadiacich systémoch zase nie je možné dovoliť ani veľké oneskorenie, ani straty dát.

Kvôli efektívnemu fungovaniu služieb je potrebné, aby sieť bola schopná rozoznať dátové jednotky vytvorené rôznymi službami. V dnešnej dobe existujú dve metódy, ktoré odlišné zachádzanie v pevných dátových sieťach zaistia:

- Integrované služby (Integrated Services – IntServ) – systém je schopný rozoznať dátový tok rôznych služieb podľa identifikátora (IP adresa, ďalšie polia v IP záhlaví). Kvalitu prenosu zabezpečí tak, že potrebnú šírku pásma a ďalšie zdroje rezervuje na ten čas, kým prenos trvá. Nevýhodou je neefektívne využitie zdrojov, pretože rezervovanú časť nemôže používať iná služba ani v prípade, že je nevyužitá.
- Diferencované služby (Differentiated Services – DiffServ) – sieťovú prevádzku delí na menšie triedy a zabezpečuje rôzne spracovanie tried podľa nastavených pravidiel. Technika DiffServ nezaručuje, že dohodnuté parametre budú dodržané, ale oveľa menej zaťažuje procesor a je škálovateľnejšia, než IntServ [15].

6.1 Mechanizmy pre zaistenie kvality služieb

Každá technológia, ktorá zaručuje kvalitu služieb, musí splniť nasledujúce funkcie:

- značkovanie,
- klasifikácia,
- dohľad nad prevádzkou,
- aktívna správa front,
- plánované odosielanie paketov,

- tvarovanie prevádzky.

V ďalších častiach budú vyššie uvedené funkcie krátko popísané.

6.1.1 Značkovanie paketov

Prvým krokom pri zaistení kvality služieb je označenie jednotlivých dátových tokov. To sa najčastejšie vykonáva nastavením určitého poľa v IP záhlaví na príslušnú hodnotu, ale značka môže byť aj pripojená k IP záhlaviu. Paket prichádzajúci do smerovača môže byť už označený, v tomto prípade môže dôjsť k preznačeniu. Nastane to v prípade, že paket došiel zo siete, kde sú nastavené odlišné pravidlá značkovania, alebo paket nespĺňa vopred stanovené kritériá. V prípade techniky DiffServ sa pre značkovanie používa prvých šesť bitov 8-bitového poľa ToS v IP záhlaví. V rámci DiffServ sa toto pole nazýva DSCP (Differentiated Services Code Point) [15].

6.1.2 Klasifikácia paketov

Klasifikácia je proces, počas ktorého sú pakety podľa dopredu stanovených pravidiel rozdelené do skupín. Rozdiel medzi značkovaním a klasifikáciou je v tom, že pridelenú značku je možné použiť na klasifikáciu. Klasifikácia sa vykonáva na základe informácií v záhlaví protokolov, dve najviac používané metódy sú nasledovné:

- podľa jediného identifikátora (BA – Behaviour Aggregate) – používa sa v prípade, že paket bol už označený a klasifikovanie je vykonané podľa jediného identifikátora (ToS, DSCP),
- viacpoložková klasifikácia (MF – Multi-Field Classification) – klasifikácia sa vykonáva podľa jedného alebo viacerých položiek záhlavia IP alebo TCP/UDP (cieľová a zdrojová IP adresa a port, ďalej ich kombinácia) [15].

6.1.3 Dohľad nad sieťovou prevádzkou

Dohľad nad prevádzkou má zabezpečiť, aby parametre dátového toku sa pohybovali v dopredu stanovenom rozmedzí, ktoré boli dohodnuté medzi zákazníkom a poskytovateľom QoS. Základom tohto kroku je meranie prevádzky siete. Výsledky merania potom môžu viesť k ponechaniu značky, preznačkovaniu alebo k zahodeniu paketu. Značkovanie teda odráža výsledky merania. Najčastejšie merané parametre sú nasledovné:

- Maximálna okamžitá rýchlosť (PIR – Peak Information Rate) – vyjadruje maximálnu rýchlosť prenosu, meria sa v B/s a fyzicky je obmedzená šírkou pásma linky.
- Garantovaná priemerná prenosová rýchlosť (CIR – Committed Information Rate) – udáva dlhodobú priemernú prenosovú rýchlosť, tiež sa meria v jednotkách B/s.
- Veľkosť zhluku (Burst Size) – k meraniu prevádzky používajú ďalšie tri parametre: veľkosť garantovaného zhluku (CBS – Committed Burst Size), veľkosť nadmerného zhluku (EBS – Excess Burst Size) a veľkosť maximálneho zhluku (PBS – Peak Burst Size). Parameter CBS udáva množstvo dát, ktoré môžeme poslať rýchlosťou PIR bez prerušenia. Z predchádzajúcich je zrejmé, že krátkodobo môžeme posielať dáta

s rýchlosťou PIR do vyčerpania CBS a pri tom dlhodobo neprekročíme rýchlosť CIR. V prípade, že dátový tok neprekročí dohodnuté parametre, paketom je zaistená dohodnutá priorita. V opačnom prípade ďalším paketom do hodnoty EBS je pridelená nižšia priorita. Pakety, ktoré prekročia EBS majú ešte nižšiu prioritu, sú spravidla zahodené. Podobné využitie má aj parameter PBS, ale je definovaný pre rýchlosť PIR [15].

Meranie prevádzky sa najčastejšie vykonáva mechanizmom Token Bucket (TB). Je možné to predstaviť ako nádobu, ktorá obsahuje určité množstvo tokenov v určitom okamihu. Každý token dovoľuje posielanie určitého množstva dát (spravidla 1 B). Po príchode paketu sa skontroluje, či je v nádobe potrebné množstvo tokenov, v prípade že áno, pokračuje sa ďalej štandardným spôsobom. V opačnom prípade pakety môžu byť preznačkované, zahodené, alebo uložené do vyrovnávacej pamäte [15].

6.1.4 Riadenie odosielania paketov

Z pohľadu odlišného zachádzania sú fronty a riadenie odosielania dátových jednotiek z týchto front kľúčovými blokmi celého procesu. Základné požiadavky na tento riadiaci systém sú nasledovné:

- rozdelenie dostupnej šírky pásma podľa nastaveného prioritného systému,
- rozdelenie dostupnej šírky pásma rovnomerne medzi dátovými tokmi s rovnakou prioritou [15].

Riadenie odosielania sa vykonáva pre každý port zvlášť, preto pred zaradením prichádzajúcich paketov do rôznych front pakety sú smerované na príslušný port. Potom sa riadiaci systém rozhodne, ktorý paket bude poslaný na danom porte. Najznámejšie fronty sú nasledovné:

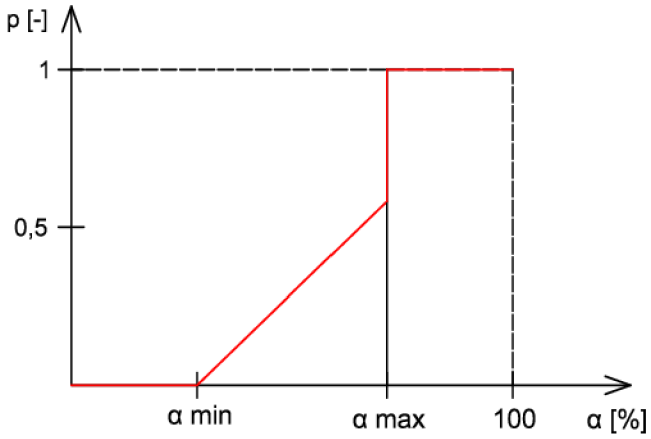
- FIFO (First In First Out) – táto fronta posiela pakety v poradí, v akom prišli. Výhodou je jednoduchá implementácia, pretože neobsahuje prakticky žiadne riadenie. Je vhodná pre siete typu best-effort.
- PQ (Priority Queuing) – u tohto systému existuje M front s prioritami 1 až M. Poradie posielania paketov závisí od priority pri čom platí, že nie je možné poslať paket z fronty s danou prioritou, keď existuje paket vo fronte s vyššou prioritou. Výhodou je jednoduchosť, ale v prípade, že fronty s vyššou prioritou sú neustále naplňované, fronty s nižšou prioritou nedostanú príležitosť na posielanie. Je vhodný pre siete, kde prevádzka s vyššou prioritou predstavuje len zlomok celej prevádzky.
- FQ (Fair Queuing) – pakety sú rozdelené do M front, ktoré sú cyklicky obsluhované a počas každého cyklu je poslaný jeden paket z každej fronty bez ohľadu na ich veľkosť. Algoritmus zase nie je zložitý, ale fronta nie je príliš efektívna, keď rôzne dátové toky majú odlišné požiadavky na prenosovú rýchlosť.
- WRR (Weighted Round Robbin) – používa dvojstupňové triedenie, pakety sú najprv rozdelené do M tried a v rámci triedy do N front. Prenosová rýchlosť tried je priamo úmerná ich priorite, ďalej fronty každej triedy sú obsluhované metódou FQ.

- WFQ (Weighted Fair Queuing) – princíp je podobný, ako u WRR, ale pri obsluhovaní front v jednotlivých triedach je zohľadnená aj veľkosť paketov. Praktická realizácia nie je príliš efektívna, preto sa nepoužíva.
- CB WFQ (Class-Based Weighted Fair Queuing) – principiálne je tiež podobná ako WRR, ale pre obsluhovanie front v jednotlivých triedach sa používa metóda WFQ [15].

6.1.5 Aktívna správa front

Úlohou aktívnej správy front je ochrana vyrovnávacej pamäte proti úplnému naplneniu, čo môže spôsobiť výpadok celej siete. Pasívna správa front nie je dostačujúca hlavne kvôli TCP spojení, môže nastať nasledujúca situácia: naplní sa vyrovnávacia pamäť sieťového prvku, kedy sa začne zahadzovanie ďalších prichádzajúcich paketov. Nie sú tak doručené ani TCP segmenty, teda zdrojové stanice ich vyhodnotia ako stratené a každá stanica zopakuje vysielanie v tom istom čase s nižšou rýchlosťou. Toto zníženie rýchlosti ale nie je dostačujúce a sieťový prvok bude zase zahltený. Môže sa stať, že sieťový prvok tak bude oscilovať medzi zahltením a minimálnou prevádzkou [15].

Základným riešením hore popísaného problému je metóda RED (Random Early Detection). Jej hlavnou úlohou je časová analýza dátových jednotiek, ktoré sú uložené vo vyrovnávacej pamäti, ďalej pozoruje dĺžku fronty, stanový mieru zahltenia a vypočíta percentuálne využitie pamäte α. K tomuto parametru je priradená ľubovoľná krivka (profil), podľa ktorého sa určí pravdepodobnosť zahodenia prichádzajúcich paketov (obr. 6-1). Parameter α_{MIN} stanový minimálne využitie pamäte, pri ktorom sa začne zahadzovanie, α_{MAX} udáva hranicu, po ktorej je zahodený každý prichádzajúci paket. Náhodné zahadzovanie paketov zruší a spomalí naraz len niekoľko TCP spojení, tak sa zníži intenzita prevádzky. Na rýchlosť posielania s protokolom UDP nemajú vplyv zahodené pakety, preto táto metóda nie je vhodná pre riadenie UDP dátového toku. Vylepšením RED je metóda WRED (Weighted Random Early Detection), ktorá umožňuje použitie rôznych profilov pre jednotlivé fronty [15].



Obr. 6-1 Príklad profilu zahadzovania paketov metódou RED [15]

Ďalším riešením tejto problematiky je metóda ECN (Explicit Congestion Notification), ktorá umožňuje signalizáciu blížiaceho sa, alebo úplneho zahltenie vyrovnávacej pamäte ostatným uzlom siete. Tento mechanizmus môže fungovať len ako prídavný, komunikujúce uzly

sa musia dopredu dohodnúť o používaní tohto systému. Výpočet zaťaženia funguje rovnako ako u metódy RED a signalizácia zahltenia sa vykonáva pomocou posledných dvoch bitov poľa DSCP v IP pakete [15].

6.2 Riadenie kvality služieb v bezdrôtových sieťach

V poslednom čase sa bezdrôtové siete vyvíjali veľmi rýchlo aj pre domáce, aj pre priemyselné využitie kvôli dobrému pomeru cena/výkon. Najrozšírenejšou bezdrôtovou technológiou je v dnešnej dobe štandard 802.11, kvôli nízkej cene však až do nedávna chýbali z neho mechanizmy pre zaistenie kvality služieb. V tejto podkapitole budú predstavené riešenia tejto problematiky.

6.2.1 Architektúra bezdrôtových sietí

Základom bezdrôtových sietí je prístupový bod (AP – Access Point), ktorý je určitým spôsobom (kábel, optika, rádiové spojenie) spojený ďalšími časťami siete a z pohľadu funkcie sa veľmi podobá na prepínače v pevných sieťach. Stanice, ktoré sú vybavené vhodnou perifériou (bezdrôtová sieťová karta), sa môžu pripojiť na sieť cez AP. Pre komunikáciu medzi stanicami a AP sa používa kódovanie, ktoré je založené na rozprestretí spektra: FHSS (Frequency Hopping Spread Spectrum), DSSS (Direct Sequence Spread Spectrum) alebo OFDM (Orthogonal Frequency Division Multiplexing). Prenos je vždy poloduplexný a pri návrhu je treba zohľadniť oblasť pokrytia a vzájomnú vzdialenosť prvkov. Pri zvýšení vzdialenosti sa prenosová rýchlosť logicky zníži [15].

Bezdrôtové siete môžeme podľa vzájomného usporiadania prvkov a podľa sady služieb rozdeliť na nasledujúce skupiny:

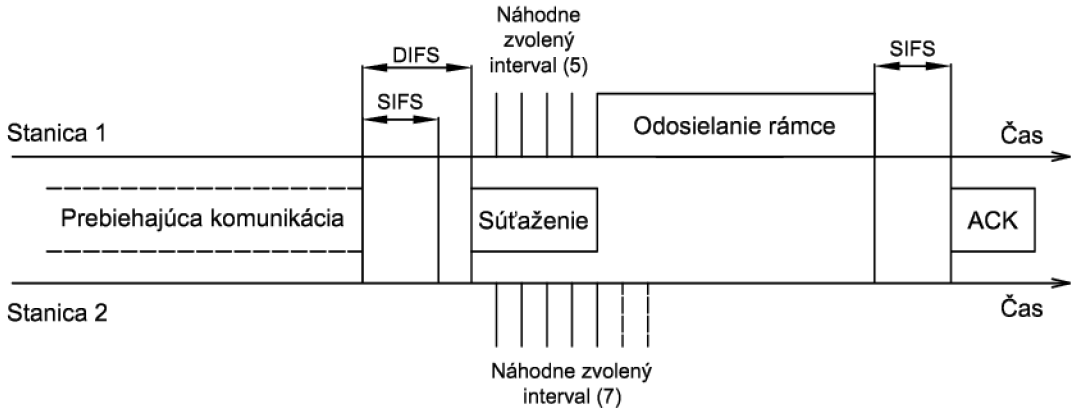
- Základná sada služieb (BSS – Basic Service Set) – obsahuje jeden AP a niekoľko staníc, stanice môžu komunikovať len nepriamo, cez AP.
- Rozšírená sada služieb (ESS – Extended Service Set) – je zostavená z dvoch alebo viacerých AP, ktoré sú spojené určitým spôsobom do distribučnej siete. Komunikácia prebieha tiež nepriamo, cez jednotlivé AP.
- Nezávislá sada služieb (IBSS – Independent Basic Service Set) – bezdrôtová sieť neobsahuje AP, existuje len skupina staníc, ktoré komunikujú buď priamo, alebo cez ďalšie stanice podľa toho, či komunikujúce zariadenia sa vzájomne nachádzajú v oblasti pokrytia [15].

6.2.2 Základné mechanizmy riadenia prístupu v bezdrôtových sieťach

Prístupové metódy k médiu štandardu 802.11 sa nazývajú koordinačné funkcie a špecifikácie 802.11a//b/g/n definujú dve takéto funkcie:

- Distribuovaná koordinačná funkcia (DCF – Distributed Coordination Function) – v prípade náhodného prístupu klienti súťažia o prístup k zdieľanému médiu. Je možné použiť každou, vyššie vymenovanou sadou služby.
- Centralizovaná koordinačná funkcia (PCF – Point Coordination Function) – je to metóda bez súťaženia, pred použitím je treba registrácia klienta u AP. Z toho vyplýva, že môže fungovať len kombinovane s metódou DCF. AP potom pravidelne dotazuje registrované stanice, či majú niečo na poslanie. V praxi sa príliš nepoužíva [15].

V komunikácii cez WLAN (Wireless LAN) hrá veľkú rolu medzirámcová medzera, dĺžka tejto doby totiž ovplyvňuje pravdepodobnosť, že stanica dostane prístup k zdieľanému médiu, preto je vhodná pre prioritné riadenie prístupu (obr. 6-2). Najmenší takýto interval je krátka medzirámcová medzera (SIFS – Short Interframe Space), ktorá ponúka najväčšiu šancu úspešného prístupu, preto sa používa pre rámce s najvyššou prioritou (napr. rámec ACK – Acknowledgement). Medzirámcová medzera DIFS (Distributed Coordination Function Interframe Space) je základom koordinačnej funkcie DCF, po uvoľnení sa média, každá stanica musí čakať túto dobu pred začatím súťaženia. Pred začatím súťaženia je ešte nastavený časovač každej stanice na náhodnú hodnotu z intervalu <0, CW>. Hodnota okna súťaženia (CW – Contention Window) je v základnom režime konštantná, ale v prípade kolízie sa navýši (pri mnohonásobnej kolízii aj viackrát), aby pravdepodobnosť ďalšej kolízie bola minimálna. Pretože kolízie alebo iné rušiace faktory môžu znehodnotiť správu, druhá strana musí vždy poslať potvrdenie ACK (musí čakať len SIFS) [15].



Obr. 6-2 Medzirámcové medzery a ich využitie [15]

Existuje aj mechanizmus, ktorý umožňuje rezerváciu médiu na určitý čas, používajú sa k tomu nasledujúce správy:

- RTS (Ready to Send) – zdrojová stanica oznamuje AP a cieľovej stanici, že je pripravená na posielanie,
- CTS (Clear to Send) – cieľová stanica oznamuje AP a zdrojovej stanici, že je pripravená na príjem.

Vyššie popísané správy dostanú aj ostatné stanice v dosahu, čím je zaručený, že počas posielania nepokúsia obsadiť médium. V praxi sa používa len v špecifických prípadoch [15].

6.2.3 Mechanizmy pre zaistenie QoS v bezdrôtových sieťach

Nedostačujúca podpora QoS u seiťach WLAN značne obmedzil využitie tejto technológie u služieb v reálnom čase. Preto bol vyvinutý štandard 802.11e, ktorý rozšíril stávajúce prístupové metódy s mechanizmami pre zaistenie kvality služieb. Tieto rozšírenia sú kompatibilné aj so zariadeniami založené na predchádzajúcich štandardoch. Štandard 802.11e definuje novú sadu služieb, ktorá je označená ako QBSS (QoS Supporting BSS). Siete QBSS obsahujú tzv. hybridný koordinátor (HC – Hybrid Coordinator), ktorý je spravidla AP. Stanice s podporou nového štandardu sú označené ako QSTA (QoS Station) [15].

6.2.4 Rozšírený distribuovaný prístup ku kanálu (EDCA)

Kvalita služieb u prístupovej metódy EDCA (Enhanced Distributed Channel Access) je zabezpečená na základe kategórie prístupu (AC – Access Category) – každý klient triedi prevádzku do štyroch kategórií a do ôsmych prioritných úrovní (tab. 6-1). Tieto prioritné úrovne sú rovnaké ako definované v štandarde 802.1P, preto je zabezpečená spolupráca s pevnými lokálnymi sieťami. V dnešnej dobe väčšina prevádzky je označená ako best-effort, teda má pridelenú prioritnú úroveň 0. Z tab. 6-1 je vidno, že táto úroveň má vyššiu prioritu, než prenos na pozadí. Exellent-effort sa tiež používa na prevádzku s charakterom best-effort, ale označuje skupinu užívateľov, ktorá je zvýhodnená oproti ostatným (napr. riaditeľstvo vo firme). Riadená záťaž je taká prevádzka, ktorá spĺňa nejaké dopredu stanovené podmienky (napr. klasifikovaná alebo tvarovaním upravená prevádzka) [15]. Poznámka: v tomto bode je treba si uvedomiť, že rovnaké zachádzanie s dátovými jednotkami označenými ako prístup best-effort nemá nič spoločného s prístupovou kategóriou best-effort. Logicky, keď sú pakety značkované a rozdelené do rôznych tried, dátové jednotky nemôžu byť spracované metódou best-effort.

Tab. 6-1 Kategórie prístupu a prioritné úrovne v EDCA [15]

Prioritná úroveň (802.1P)	Predpokladané využitie (802.1P)	Kategórie prístupu (AC)	Predpokladané využitie (802.11e)
1 (najnižšia)	prenos na pozadí	AC_BK	prenos na pozadí
0	best-effort (vychádzajúce)	AC_BE	best-effort
2	excellent-effort	AC_BE	best-effort
3	riadený záťaž	AC_VI	video
4	video (oneskorenie do 100ms)	AC_VI	video
5	hlas (oneskorenie do 10ms	AC_VO	hlas
6	management siete	AC_VO	hlas
7 (najvyššia)	správa siete	AC_VO	hlas

Rámce po príchode na aktívny prvok sú triedené do týchto kategórií a sú uložené do príslušnej fronty. Každá fronta sa chová ako virtuálna stanica, ktoré súťažia o prístup k médiu podobne, ako bezdrôtové stanice u prístupovej metódy DCF. Fronta, ktorej časovač vyprší

najskôr, dostane tzv. príležitosť prenosu (TXOP – Transmission Opportunity). Je to dopredu stanovená doba, počas ktorej je možné poslať rámec. Pevnou dĺžkou TXOP je vyriešená aj problematika synchronizácie [15].

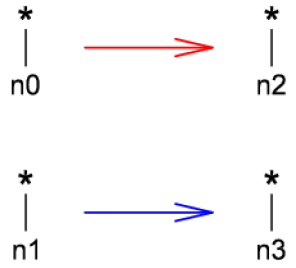
Aby spomínaná metóda fungovala, bola zavedená nová medzirámcová medzera, ktorá sa nazýva medzirámcová medzera výberu (AIFS – Arbitration Interframe Space). Každá kategória AC má inú hodnotu AIFS, ktorá je nastaviteľná podľa potreby. Musí ďalej platiť pravidlo, že AIFS ≥ DIFS. Veľkosť okna súťaženia CW je tiež nastaviteľná, logicky pre prevádzku s vyššou prioritou je treba nastaviť menšie okno. Môže nastať situácia, že v rámci jedného zariadenia dostane prístup viac front naraz, vtedy hovoríme o virtuálnej kolízii. V tomto prípade TXOP bude pridelená fronte s vyššou prioritou [15].

7 ÚLOHA 3: PRINCIP QOS V BEZDRÁTOVÝCH SÍTÍCH

V této laboratorní úloze bude vytvořena bezdrátová síť s technologií 802.11b, která bude obsahovat 4 uzly (obr. 7-1). Pro generování provozu a příjem dat budou použity již dobře známé aplikace `OnOffApplication` a `PacketSink`. V simulaci budou vytvořeny dva datové toky, jeden od uzlu n0 k n2, druhý od n1 k n3. Datovému toku n0 ↔ n2 bude nastavena vyšší priorita, než druhému a budou pozorovány parametry jako ztrátovost, zpoždění a jitter u obou přenosů. Z toho vyplývá, že uzly budou nastaveny do módu „ad-hoc“, což znamená, že nebude nastaven centrální prvek pro řízení komunikace, uzly budou komunikovat přímo. Z tohoto důvodu bude komunikace zajištěna speciálním MANET (Mobile Ad-Hoc Network) směrovacím protokolem OLSR (Optimized Link State Routing). V závěru budou analyzovány záhlaví přenesených rámců a budou vyhodnoceny změřeny parametry.

7.1 Topologie úlohy

Topologie úlohy je zobrazena na obr. 7-1, červená šipka představuje zvýhodněný provoz (provoz označeny jako živé video), modrá šipka provoz typu best-effort. Vzdálenost stanic bude nastavena tak, aby všechny uzly byly vzájemně v dosahu a aby měly společný přenosový kanál.



Obr. 7-1 Topológie úlohy

7.2 Úvod do zajištění kvality služeb

V nedávné minulosti stále mnoho síťových technologií, které jsou založeny na přepínání paketů a rámců, zpracoval všechny datové jednotky stejným způsobem. Tento přístup se nazývá best-effort a na první pohled se může zdát, že je to spravedlivá metoda. Ve skutečnosti to tak není, dva největší nedostatky tohoto přístupu jsou následující:

- Není možno rozeznat jednotlivé datové toky a všechny datové jednotky se zpracovávají nezávisle na ostatních. Proto není možno na sítích s náhodným přístupem různé datové třídy upřednostnit nebo naopak, zatlačit do pozadí.
- Jeden zdroj s agresivním chováním je schopen zaplavit síť a tím vyčerpat dostupné síťové zdroje (např. obsadit velkou část přenosové kapacity, nebo zaplnit fronty aktivních síťových prvků – směrovače) [15].

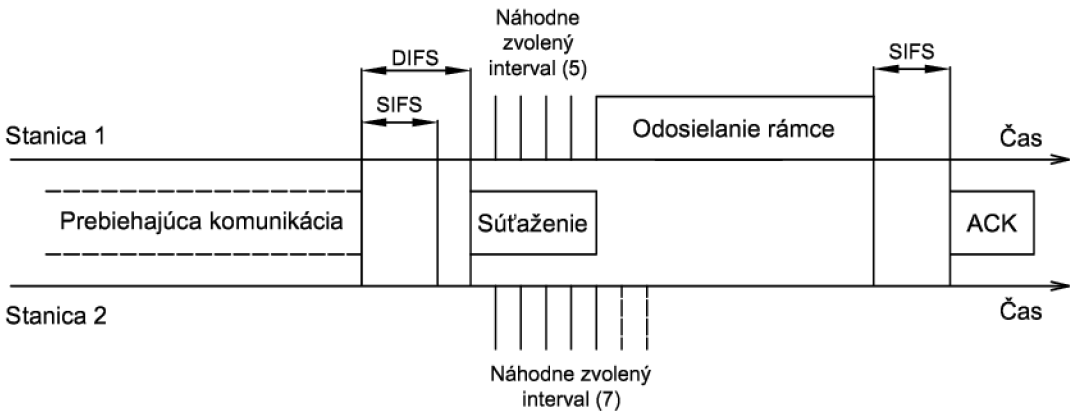
Z výše uvedeného je jasné, že nedostatek síťových zdrojů má stejný dopad na všechny datové toky, má však velmi odlišný dopad na služby, které tyto datové toky vytvořili [15]. Např. při přenosu velkého souboru i nejmenší ztráta dat může znehodnotit celý soubor, zpoždění nebo kolísání zpoždění (jitter) však nemá z hlediska uživatele rozeznatelný vliv na přenos. Při přenosu audia nebo videa v reálném čase jsou parametry jako zpoždění a jitter kritické, ale služba toleruje určitou míru ztrátovosti. V řídicích systémech zase nelze dovolit ani velké zpoždění, ani ztráty dat.

7.2.1 Zajištění kvality služeb v bezdrátových sítích

Rádiovou komunikaci si můžeme představit jako přenos přes sdílené médium, je tedy důležité zajistit bezkolizní přístup jednotlivých stanic k tomuto médiu. Nejpoužívanější metodou je tzv. distribuována koordinační funkce (DCF – Distributed Coordination Function), která je založena na principu soutěžení: stanice vygenerují náhodné číslo v určitém rozsahu (okno soutěžení) a první stanice, jejíž časovač vyprší, může posílat data [15].

U komunikace přes WLAN (Wireless LAN) hraje dále velkou roli medzirámcová mezera, délka této doby totiž ovlivňuje pravděpodobnost, že stanice dostane přístup ke sdílenému médiu, proto je vhodná pro prioritní řízení přístupu (obr. 7-2). Nejčastěji používané medzirámcové mezery:

- Medzirámcová mezera pro DCF (DIFS – Distributed Coordination Function Interframe Space) – po uvolnění média musí každá stanice čekat tuto dobu před zahájením soutěžení.
- Krátká medzirámcová mezera (SIFS - Short Interframe Space) – nabízí větší šanci úspěšného přístupu, než DIFS, proto se používá pro speciální rámce s vyšší prioritou (např. rámec pro potvrzení přenosu ACK) [15].



Obr. 7-2 Mezirámcové mezery a jejich využití [15]

7.2.2 Rozšířený distribuovaný přístup ke kanálu (EDCA)

V této úloze bude použita přístupová metoda EDCA (Enhanced Distributed Channel Access), která je schopna zajistit i kvalitu služeb na druhé vrstvě OSI modelu. Provádí to na základě přístupových kategorií (AC – Access Category) – každý klient třídí provoz do čtyř kategorií

a do osmi prioritních úrovní (tab. 7-1). Tyto prioritní úrovně jsou stejné jako jsou definovány ve standardu 802.1p, proto je zajištěna spolupráce s pevnými lokálními sítěmi [15]. Poznámka: v tomto bodě je třeba si uvědomit, že stejné zacházení s datovými jednotkami označenými jako přístup best-effort nemá nic společného s přístupovou kategorií best-effort. Logicky, když jsou pakety označkovány a rozděleny do různých tříd, datové jednotky nemohou být zpracovány metodou best-effort.

Tab. 7-1 Kategorie přístupu a prioritní úrovně v EDCA [15]

Prioritní úroveň (802.1P)	Předpokládané využití (802.1P)	Kategorie přístupu (AC)	Předpokládané využití (802.11e)
1 (nejnižší)	přenos na pozadí	AC_BK	přenos na pozadí
0	best-effort (vycházející)	AC_BE	best-effort
2	excellent-effort	AC_BE	best-effort
3	řízený zátěž	AC_VI	video
4	video (zpoždění do 100ms)	AC_VI	video
5	hlas (zpoždění do 10ms	AC_VO	hlas
6	management sítě	AC_VO	hlas
7 (nejvyšší)	správa sítě	AC_VO	hlas

Rámce jsou po příchodu na aktivní prvek tříděny do těchto kategorií a jsou uloženy do příslušné fronty. Každá fronta se chová jako virtuální stanice, které soutěží o přístup k médiu podobně, jako bezdrátové stanice u přístupové metody DCF. Fronta, jejíž časovač vyprší nejdříve, dostane tzv. příležitost přenosu (TXOP – Transmission Opportunity). Je to dopředu stanovená doba, po kterou je možné poslat rámec. Pevnou délkou TXOP je vyřešena i problematika synchronizace [15].

Aby zmíněná metoda fungovala efektivně, byla zavedena nová medzirámcová mezera, která se nazývá mezirámcová mezera výběru (AIFS – Arbitration Interframe Space). Každá kategorie AC má jinou hodnotu AIFS, která je nastavitelná podle potřeby. Musí dále platit pravidlo AIFS ≥ DIFS. Velikost okna soutěžení je také nastavitelná, logicky pro provoz s vyšší prioritou je třeba nastavit menší okno. Může nastat situace, že v rámci jednoho zařízení dostane přístup více front najednou, tehdy mluvíme o virtuální kolizi. V tomto případě TXOP bude přidělena frontě s vyšší prioritou.

7.3 Vytvoření modelu sítě

Předem připravený soubor `qos.cc` zkopírujte do složky *Scratch* a nastavte, aby jej bylo možné spustit z prostředí Eclipse (na začátku není třeba nic doplňovat). Po spuštění by se zdrojový kód měl úspěšně zkompilovat a proběhnout, výsledkem jsou zatím „nesmyslné“ hodnoty pozorovaných parametrů (ztrátovost, zpoždění a jitter). Na začátku hlavní funkce main je nastaveno logování aplikací `OnOffApplication` a `PacketSink`, toto bude sloužit

k ověření přenosu mezi danými uzly. Dále jsou zde předem deklarované proměnné typu `string`, které budou sloužit k nastavení kódování přenosu na fyzické vrstvě (`phyMode`) a na nastavení bitové rychlosti aplikací (`appBitRate`). Jak to již bylo zmíněno, v úloze budeme potřebovat 4 uzly, řádky na jejich vytvoření jsou již také přítomny ve zdrojovém kódu. Na konci hlavní funkce je nastaveno spuštění a zastavení simulace a poslední řádky provádějí výpočet a výpis parametrů přenosu do konzole.

7.3.1 Definice bezdrátového spojení

Prvním krokem tohoto úkolu bude vytvoření bezdrátového spojení mezi uzly. Pomocí níže uvedeného zdrojového kódu definujeme standard 802.11b a přenosovou rychlost na fyzické vrstvě na 1 Mb/s s linkovým kódováním DSSS. Dále objekty `wifiPhy` a `wifiChannel` nastavíme ztrátový model a model zpoždění při přenosu. Třída `QosWifiMacHelper` slouží k definici kvality služeb – nastavíme ji na výchozí hodnotu. Pomocí funkce `SetType` nastavíme vytvořenou síť na linkové vrstvě na Ad-Hoc. Na konec vytvoříme síťová zařízení a spolu s nastaveními je nainstalujeme na vytvořené uzly. **Níže uvedený zdrojový kód zkopírujte na místo označené jako** *Definice bezdrátového spojení***.**

Všimnete si, že pro přenosovou technologii jsme nepoužili nejnovější standard 802.11n, ale v dnešní době poměrně zastaralý 802.11b. Je to kvůli tomu, že třída `QosWifiMacHelper` nejnovější standard nepodporuje, pro 802.11n je nutná tzv. podpora vysoké propustnosti (high throughput). Namísto `QosWifiMacHelper` bychom měli použít třídu `HtWifiMacHelper` a další nastavení, které by dále komplikovaly zdrojový kód. Také byla nastavena přenosová rychlost na velmi nízkou hodnotu (802.11b teoreticky zvládne i 11 Mb/s) – nastavená rychlost však pro účely této simulace úplně stačí a sníží se i výpočetní náročnost simulace.

```
WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                              "DataMode",StringValue (phyMode),
                              "ControlMode",StringValue (phyMode));
YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
wifiPhy.SetChannel (wifiChannel.Create ());

QosWifiMacHelper wifiMac = QosWifiMacHelper::Default ();
wifiMac.SetType ("ns3::AdhocWifiMac");
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, nodes);
```

7.3.2 Nastavení pohybového modelu

Podobně jako v předchozích úlohách, se uzly nebudou pohybovat ani v tomto případě, ale kvůli použití rádiového spojení je třeba specifikovat jejich umístění ve virtuálním prostředí. Použijeme tedy model pro stacionární uzly `ConstantPositionMobilityModel`. **Níže uvedený zdrojový kód umístěte na místo označené jako** *Nastavení pohybového modelu***.**

```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
```

7.3.3 Nastavení směrovacího protokolu

V této úloze použijeme na směrování protokol OLSR, jenž je protokol pro MANET sítě. Všechny uzly se tedy budou chovat jako směrovače, v simulaci s pohybujícími se uzly by se tento systém byl schopen adaptovat i na změny v topologii. Při definování ve zdrojovém kódu musíme vytvořit jeden objekt pro směrovací protokol OLSR (`olsr`) a také jeden pro statické směrování (`staticRouting`). Dalšími příkazy pak nastavíme prioritu statického směrování na hodnotu 0 a prioritu OLSR směrování na hodnotu 10 (nejvyšší priorita). Nakonec vytvoříme protokolovou sadu `internet`, přiřadíme k ní seznam směrovacích protokolů a jejich prioritu (`list`), a nainstalujeme na vytvořené uzly. **Níže uvedený zdrojový kód umístěte na místo označené jako** *Nastavení směrování***.**

```
OlsrHelper olsr;
Ipv4StaticRoutingHelper staticRouting;

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

InternetStackHelper internet;
internet.SetRoutingHelper (list);
internet.Install (nodes);
```

7.3.4 Nastavení IP adres

Následujícím krokem je nastavení IP adres, použijeme k tomu známý způsob pomocí třídy `Ipv4AddressHelper`. V této úloze budou všechny počítače v jednom IP rozsahu: 10.1.1.0/24. Vybraný rozsah pak přiřadíme k síťovým zařízením. **Níže uvedený zdrojový kód umístěte na místo označené jako** *Nastavení IP adres***.**

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

7.3.5 Definice aplikací

Jak již to bylo zmíněno na začátku, pro generování provozu bude použita aplikace `OnOffApplication`. Budeme z nich potřebovat dvě, jednu s vyšší, druhou s nižší prioritou. Postup samotné definice v tomto případě bude ale trošku odlišný, než v předchozích úlohách kvůli zpřehlednění zdrojového kódu.

Na začátku vytvoříme funkci `setApp` před hlavní funkcí `main`, ve které definujeme jednu aplikaci `OnOffApplication` a `PacketSink` standardním způsobem. Tuto funkci pak můžeme volat s vhodnými vstupními parametry, tím vytvoříme aplikace mnohem jednodušší. Vstupními parametry funkce `setApp` jsou:

- `interfaces` – skupina vytvořených rozhraní,
- `nodes` – skupina vytvořených uzlů,
- `port` – cílový port pro odesílání,
- `source` a `destination` – číslo zdrojového a cílového uzlu,
- `stratTime` a `stopTime` – čas spuštění a zastavení aplikace,
- `dataRate` – bitová rychlost aplikace.

Uvedený zdrojový kód umístěte na místo označené jako *Funkce setApp*.

```
void setApp(Ipv4InterfaceContainer interfaces, NodeContainer nodes, int port,
int source, int destination,  double startTime, double stopTime, string
dataRate) {

	OnOffHelper onOff("ns3::UdpSocketFactory",
		InetSocketAddress(interfaces.GetAddress(destination), port));
	onOff.SetAttribute("OffTime",
		StringValue("ns3::ConstantRandomVariable[Constant=0]"));
	onOff.SetAttribute("DataRate",StringValue (dataRate));
	onOff.SetAttribute("PacketSize", UintegerValue(1024));

	ApplicationContainer appCont = onOff.Install(nodes.Get(source));
	appCont.Start(Seconds(startTime));
	appCont.Stop(Seconds(stopTime));

	PacketSinkHelper sink("ns3::UdpSocketFactory",
		InetSocketAddress(Ipv4Address::GetAny(), port));
	appCont = sink.Install(nodes.Get(destination));
}
```

Samotné vytvoření aplikací se provádí voláním této funkce v hlavní funkci `main`, využijeme k tomu níže uvedený zdrojový kód. Ze vstupních hodnot vyplývá chování aplikací:

- První aplikace bude nainstalována na uzel n0 a bude posílat data na uzel n2 s rychlostí 0,5 Mb/s (`appBitRate`).
- Druhá aplikace bude nainstalována na uzel n1 a bude posílat data na uzel n3 také s rychlostí, kterou určuje proměnná `appBitRate`.

Všimnete si, že obě dvě aplikace budou spuštěny v čase 20 s, ne hned na začátku simulace. Je to kvůli směrovacímu protokolu OLSR, který je tzv. preaktívní směrovací protokol, tedy cesty k jiným uzlům se hledají hned po sestavení sítě, proto je zpoždění obecně malé. Po odstartování sítě však úplná konvergence chvilku trvá. Experimentálně bylo zjištěno, že pro naši topologii je to přibližně 15 s. Aplikace pak budou zastaveny v čase 25 s. Nastavení priorit se provádí pomocí trasovacího systému, toto bude představeno v následujících částech návodu (kap. 7.3.7). Zdrojový kód na volání funkce, který nastaví aplikace, je uveden níže, **zkopírujte ho na místo označené jako *Vytvoření aplikací*.**

```
uint16_t port = 9;
setApp(interfaces, nodes, port, 0, 2, 20, 25, appBitRate);
setApp(interfaces, nodes, port, 1, 3, 20, 25, appBitRate);
```

7.3.6 Databáze pro měření parametrů sítě

Hlavními výstupy tohoto úkolu budou naměřené parametry datových toků, jejichž priorita bude nastavena na různé hodnoty. Kvůli velkému počtu proměnných, které jsou nezbytné pro měření jednoho datového toku, byla navržena třída `Database`, která byla předem umístěna do zdrojového kódu. Ve zdrojovém kódu byly také předem vytvořeny dva objekty tohoto typu: `preferred` a `bestEffort`. V následující části budou důležité proměnné a funkce této třídy krátce popsané, aby další části kódu byly pochopitelné.

Vybrané proměnné třídy `Database`:

- `double dataSent` – množství poslaných dat,
- `double dataReceived` – množství přijatých dat,
- `double averageLatency` – průměrná hodnota zpoždění,
- `double averageJitter` – průměrná hodnota kolísání zpoždění.

Vybrané funkce třídy `Database`:

- `void` **`saveSentTime`**`(Ptr<Packet const> packet)`
  uloží číslo a čas odesílání paketu, vstupním parametrem je ukazatel na aktuální paket,
- `void` **`saveReceivedTime`**`(Ptr<Packet const> packet)`
  uloží čas přijetí paketu podle jeho čísla, vstupním parametrem je ukazatel na aktuální paket,
- `void` **`calculateLatencyAndJitter`**`()`
  vypočte průměrné zpoždění a kolísání zpoždění z uložených hodnot.

7.3.7 Nastavení priorit paketů a měření parametrů přenosu

Jak již to bylo zmíněno, nastavení priorit se provádí pomocí trasovacího systému. Níže uvedeným zdrojovým kódem se na něj připojíme a budeme pozorovat, kdy aplikace `OnOffApplication` pošle paket z uzlu n0 nebo z n1. V případě, že tyto události nastanou, je volána funkce `setQoSTag`, která bude podrobněji představena v následující části návodu. V metodě `Connect` namísto funkce `MakeCallback` byla v tomto případě použita funkce `MakeBoundCallback`, která umožňuje, aby funkce `setQoSTag` měla kromě povinných vstupních parametrů ještě jeden další, námi definovaný vstupní parametr. V našem případě to bude proměnná typu `string` a ve funkci `setQoSTag` je pojmenována jako `packetType`. Tato proměnná definuje, zda paket byl poslán uzlem n0 nebo n1, tedy zda jde o upřednostňovaný přenos nebo o přenos typu best-effort (v níže uvedeném zdrojovém kódu jsou tyto proměnné „preferred“ a „best-effort“). **Zdrojový kód umístěte na místo označené jako** *Zachycování posílání***.**

```
Config::Connect("/NodeList/0/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&setQoSTag, "preferred"));
Config::Connect("/NodeList/1/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&setQoSTag, "best-effort"));
```

Pro nastavení priorit bude použita funkce `setQoSTaq`, která je uvedena níže. V této funkci je nejprve vytvořen objekt `qosTag`, pak se podle zmíněné vstupní proměnné

`packetType` rozhodne, zda má být přenos preferovaný nebo ne. Po podmínce se uloží čas posílání do příslušné databáze a k proměnné `dataSent` se přičítá velikost aktuálního paketu. Na začátku zdrojového kódu můžete najít proměnné `tidPreferred` a `tidBestEffort`, které použijeme k nastavení priority datových toků – mají hodnotu 4 a 0. Pomocí funkce `SetTid` nastavíme hodnotu TID (Traffic ID – je to další pojmenování na kategorii přístupu) objektu `qosTag` a na konec přiřadíme k aktuálnímu paketu funkcí `AddPacketTag`. **Uvedený zdrojový kód umístěte na místo označené jako** *Funkce setQoSTag*.

```
void setQoSTag (string packetType, string path, Ptr<Packet const> packet)
{
      QosTag qosTag;

      if (packetType == "preferred") {
            preferred.dataSent = preferred.dataSent + packet->GetSize();
            preferred.saveSentTime(packet);
            qosTag.SetTid(tidPreferred);
      }
      else if (packetType == "best-effort") {
            bestEffort.dataSent = bestEffort.dataSent + packet->GetSize();
            bestEffort.saveSentTime(packet);
            qosTag.SetTid(tidBestEffort);
      }

      packet->AddPacketTag (qosTag);
}
```

Dále pro měření zpoždění a kolísání zpoždění je třeba pozorovat události, kdy aplikace `PacketSink` přijme pakety na uzlech n2 popř. n3. Pro toto také použijeme trasovací systém a funkci `MakeBoundCallback`. **Uvedený zdrojový kód zkopírujte na místo označené jako** *Zachycování příjmu*.

```
Config::Connect("/NodeList/2/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback (&receivePacket, "preferred"));
Config::Connect("/NodeList/3/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback (&receivePacket, "best-effort"));
```

Když nastane příslušná událost, je volána funkce `receivePacket`, která má také námi volený vstupní parametr `packetType`. Podle toho se rozhodne, do které databáze budou uloženy časy přijetí paketů a ke které proměnné se přičítá velikost přijatého paketu. **Zdrojový kód umístěte na místo označené jako** *Funkce receivePacket*.

```
void receivePacket(string packetType, string path, Ptr<Packet const> packet,
Address const& address) {

      if (packetType == "preferred") {
            preferred.dataReceived =
                  preferred.dataReceived + packet->GetSize();
            preferred.saveReceivedTime(packet);
      }
      else if (packetType == "best-effort") {
            bestEffort.dataReceived =
                  bestEffort.dataReceived + packet->GetSize();
            bestEffort.saveReceivedTime(packet);
      }}
```

Na konec nastavte vytvoření trasovacích souborů, budeme je potřebovat v dalších částech úlohy (stačí povolit příslušný zakomentovaný řádek).

7.4 Výsledky simulace

7.4.1 Výstup z příkazové řádky

Prvními výstupy simulace jsou logové zprávy aplikací `OnOffApplication` a `PacketSink`, které slouží zejména ke kontrole, zda simulace byla sestavena podle představ.

Z níže uvedené části logů zpráv je vidět, že obě aplikace fungují a posílají pakety na IP adresu 10.1.1.3 resp. 10.1.1.4 (uzly n2 a n3) na cílový port 9. Aplikace `PacketSink` pak tyto pakety přijímají. Po chvilce hledání můžeme najít i případ, kdy některý z paketů nedorazil do cílového uzlu (v uvedeném výstupu označené červeně). Je to kvůli maximálnímu využití kanálu – obě aplikace posílají data rychlostí 0,5 Mb/s (bez záhlaví nižších vrstev), při čemž maximální přenosová rychlost rádiového spojení na linkové vrstvě je 1 Mb/s. Je tedy vidět, že přenosový kanál je mírně přetížen, musíme dále brát do úvahy i vliv nastaveného ztrátového modelu. **Pokud jste dostali podobné výsledky, můžete logování aplikací zakomentovat, aby vám výpisy nepřekážely při další práci.**

```
At time 23.9322s on-off application sent 1024 bytes to 10.1.1.3 port 9 total
Tx 245760 bytes
At time 23.9322s on-off application sent 1024 bytes to 10.1.1.4 port 9 total
Tx 245760 bytes
At time 23.9385s packet sink received 1024 bytes from 10.1.1.2 port 49153
total Rx 179200 bytes
At time 23.9479s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 245760 bytes
At time 23.9485s on-off application sent 1024 bytes to 10.1.1.3 port 9 total
Tx 246784 bytes
At time 23.9485s on-off application sent 1024 bytes to 10.1.1.4 port 9 total
Tx 246784 bytes
At time 23.9575s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 246784 bytes
At time 23.9649s on-off application sent 1024 bytes to 10.1.1.3 port 9 total
Tx 247808 bytes
At time 23.9649s on-off application sent 1024 bytes to 10.1.1.4 port 9 total
Tx 247808 bytes
At time 23.9669s packet sink received 1024 bytes from 10.1.1.2 port 49153
total Rx 180224 bytes
At time 23.9764s packet sink received 1024 bytes from 10.1.1.1 port 49153
total Rx 247808 bytes
```

Hlavními výstupy simulace jsou změřeny parametré preferovaného provozu a provozu typu best-effort. Z výstupu je vidět, že obě aplikace („preferred“ a „best-effort“) poslali stejné množství dat (312 320 B), níže je uvedeno, kolik z tohoto množství dat bylo přijato. Vidíme, že upřednostňovaný přenos měl mnohem menší ztrátovost, jak jsme to i očekávali. Z pohledu ztrátovosti je důležité ještě dodat, že simulace je ukončena v témže okamžiku, kdy jsou ukončeny i aplikace (v čase 25 s). Tedy pakety uložené ve vyrovnávací paměti po zastavení aplikací nejsou doručeny.

Dále jsou uvedeny průměrné hodnoty zpoždění a kolísání zpoždění celého přenosu, které také potvrdí znalosti z teorie. Zpoždění pro upřednostňovaný přenos v našem případě bylo přibližně 16,9 ms a kolísání zpoždění se pohybovalo kolem 5,6 ms, které splňují i požadavky standardu 802.1p – zpoždění má být menší než 100 ms, viz tab. 7-1. Přenos typu best-effort při tom měl mnohem větší zpoždění, které je ale např. v případě přenosu souboru s protokolem TCP zcela v pořádku.

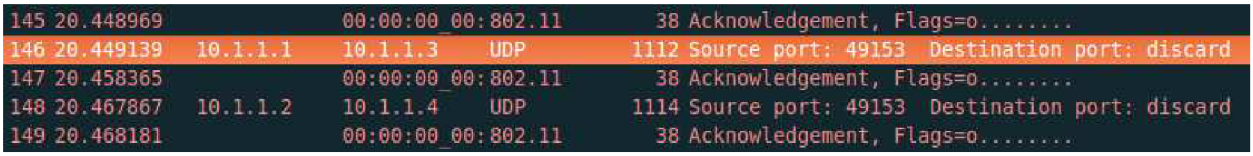
```
Preferred data sent: 312320 B
Best-effort data sent: 312320 B
Preferred data received: 311296 B, error rate = 0.327869 %
Best-effort data received: 227328 B, error rate = 27.2131 %
Average latency for preferred transmission: 16.918 ms
Average jitter for preferred transmission: 5.56048 ms
Average latency for best-effort transmission: 688.854 ms
Average jitter for best-effort transmission: 8.43052 ms
```

7.4.2 Trasovací súbor

V této části návodu budou analyzovány záhlaví rámců pomocí programu Wireshark, které byly zařazeny do některé přístupové kategorie. Otevřete tedy trasovací soubor uzlu n0 nebo n1 a vyberte jeden paket (dávejte pozor, abyste neotevřeli pakety směrovacího protokolu OLSR nebo potvrzovací zprávy ACK) se zdrojovou IP adresou 10.1.1.1 (obr. 7-3). Po vybrání položky *Radiotep Header* byste měli vidět podobný výstup, jako na obr. 7-4. Můžete odtud odečíst rychlost na linkové vrstvě (1 Mb/s) a typ přenosového kanálu (802.11b).

Další zajímavou položkou z pohledu bezdrátového přenosu je *IEEE 802.11 QoS Data* (obr. 7-5). Hned na začátku výpisu je specifikován typ rámce jako *QoS Data*, dále je vidět, že hodnota TID rámce je nastavena na 4 a že se tedy jedná o přenos videa. Otevřete další paket se zdrojovou IP adresou 10.1.1.2 a porovnejte informace v záhlaví o QoS, měli byste vidět TID nastavenou na hodnotu 0 a typ přenosu na best-effort. Je třeba zmínit, že položku TID jsme ve zdrojovém kódu virtuálně nastavovali pro pakety, ve skutečnosti se to děje na linkové vrstvě.

Na konec otevřete ještě potvrzovací zprávu *Acknowledgement* a všimnete si, že typ rámce je nastaven na *Acknowledgement* a že tento rámec neobsahuje záhlaví vyšších vrstev, je to zpráva na linkové vrstvě. Poznámka: zprávy *Acknowledgement* nepleťte s ACK zprávami protokolu TCP, v našem případě jde o potvrzování rámců na druhé vrstvě – tato funkce slouží k detekci kolize při posílání na sdíleném médiu.



Obr. 7-3 Zachycené pakety č. 146 a 148 na uzlu n0

```
Radiotap Header v0, Length 22
  Header revision: 0
  Header pad: 0
  Header length: 22
▶ Present flags
  MAC timestamp: 20479551
▶ Flags: 0x10
  Data Rate: 1,0 Mb/s
  Channel frequency: 2412 [BG 1]
▶ Channel type: 802.11b (0x00a0)
```

Obr. 7-4 Položka *Radiotep Header* paketu č. 146

```
▼ IEEE 802.11 QoS Data, Flags: o........
  Type/Subtype: QoS Data (0x28)
▶ Frame Control Field: 0x8880
  .000 0001 0011 1010 = Duration: 314 microseconds
  Receiver address: 00:00:00_00:00:03 (00:00:00:00:00:03)
  Destination address: 00:00:00_00:00:03 (00:00:00:00:00:03)
  Transmitter address: 00:00:00_00:00:01 (00:00:00:00:00:01)
  Source address: 00:00:00_00:00:01 (00:00:00:00:00:01)
  BSS Id: 00:00:00_00:00:01 (00:00:00:00:00:01)
  Fragment number: 0
  Sequence number: 28
▶ Frame check sequence: 0x00000000 [incorrect, should be 0x769f1a5d]
▼ Qos Control: 0x0004
    .... .... .... 0100 = TID: 4
    [.... .... .... .100 = Priority: Controlled Load (Video) (4)]
    .... .... ...0 .... = QoS bit 4: Bits 8-15 of QoS Control field are TXOP Duration Requested
    .... .... .00. .... = Ack Policy: Normal Ack (0x0000)
    .... .... 0... .... = Payload Type: MSDU
    0000 0000 .... .... = TXOP Duration Requested: 0 (no TXOP requested)
```

Obr. 7-5 Položka *IEEE 802.11 QoS Data* paketu č. 146

7.4.3 Kontrolní otázky

- Uveďte dva největší nedostatky přístupu best-effort.
- Jako zaručuje přístupový systém EDCA kvalitu služeb (AIFS, okno soutěžení), co jsou to přístupové kategorie?
- Co je to virtuální kolize a jak se to řeší v přístupovém systému EDCA?
- Na které vrstvě se nastavují kategorie přístupu?

7.5 Samostatné úkoly

V této části úkolu budou vyzkoušeny další simulace s různě nastavenými datovými toky z pohledu QoS a bude pozorován jejich vliv na parametry přenosu.

- Nastavte TID obou přenosů (`tidPreferred` a `tidBestEffort`) na hodnotu 0, tedy na přenos typu best-effort. Měli byste dostat podobné hodnoty, jako jsou zobrazeny

v následujícím výstupu. Je vidět, že ani jeden přenos nesplňuje požadavky na přenos živého videa.

```
Preferred data sent: 312320 B
Best-effort data sent: 312320 B
Preferred data received: 275456 B, error rate = 11.8033 %
Best-effort data received: 261120 B, error rate = 16.3934 %
Average latency for preferred transmission: 324.898 ms
Average jitter for preferred transmission: 7.21375 ms
Average latency for best-effort transmission: 390.271 ms
Average jitter for best-effort transmission: 7.63308 ms
```

- Nastavte TID prvního přenosu na hodnotu 5 (hlas) a TID druhého přenosu na hodnotu 4 (video). Měli byste dostat podobné výsledky, jako jsou níže uvedené řádky. Vysvětlete, proč má přenos hlasu vyšší prioritu, než přenos videa.

```
Preferred data sent: 312320 B
Best-effort data sent: 312320 B
Preferred data received: 278528 B Error rate 10.8197%
Best-effort data received: 264192 B Error rate 15.4098%
Average latency for preferred: 237.763 ms
Average jitter for preferred: 7.57244 ms
Average latency for best-effort: 433.645 ms
Average jitter for best-effort: 8.05707 ms
```

- Nastavte TID obou přenosů na původní hodnoty, dále nastavte, aby simulace byla ukončena v době 35s. Vysvětlete, proč je počet ztracených paketů u obou přenosů nulový a proč jsou hodnoty zpoždění větší (pomůcka: algoritmus pro měření zpoždění nezohledňuje ztracené pakety).

```
Preferred data sent: 312320 B
Best-effort data sent: 312320 B
Preferred data received: 312320 B, error rate = 0 %
Best-effort data received: 312320 B, error rate = 0 %
Average latency for preferred transmission: 16.9246 ms
Average jitter for preferred transmission: 5.54997 ms
Average latency for best-effort transmission: 796.303 ms
Average jitter for best-effort transmission: 8.01678 ms
```

- Nastavené hodnoty nechte nezměněny a zvyšte přenosovou rychlost (`appBitRate`) aplikací na 0,7 Mb/s, pak na 0,8 Mb/s a pozorujte zase zpoždění přenosů. Měli byste vidět, že mezi těmito hodnotami je hranice přetížení, kde systém EDCA již není schopen zaručit kvalitu služeb. Také je vidět, že u přenosu best-effort již docházelo k zahazování paketů. U této části je třeba zmínit, že průměrné zpoždění v reálném světě by nedosáhlo hodnoty 4 s, síťová zařízení by začala zahazovat pakety mnohem dříve kvůli značně omezené kapacitě vyrovnávací paměti.

```
Preferred data sent: 437248 B
Best-effort data sent: 437248 B
Preferred data received: 437248 B, error rate = 0 %
Best-effort data received: 437248 B, error rate = 0 %
Average latency for preferred transmission: 17.7998 ms
Average jitter for preferred transmission: 3.51693 ms
Average latency for best-effort transmission: 3140.79 ms
Average jitter for best-effort transmission: 10.5018 ms
```

```
...
Preferred data sent: 499712 B
Best-effort data sent: 499712 B
Preferred data received: 499712 B, error rate = 0 %
Best-effort data received: 488448 B, error rate = 2.2541 %
Average latency for preferred transmission: 236.278 ms
Average jitter for preferred transmission: 2.21671 ms
Average latency for best-effort transmission: 4022.61 ms
Average jitter for best-effort transmission: 9.81411 ms
```

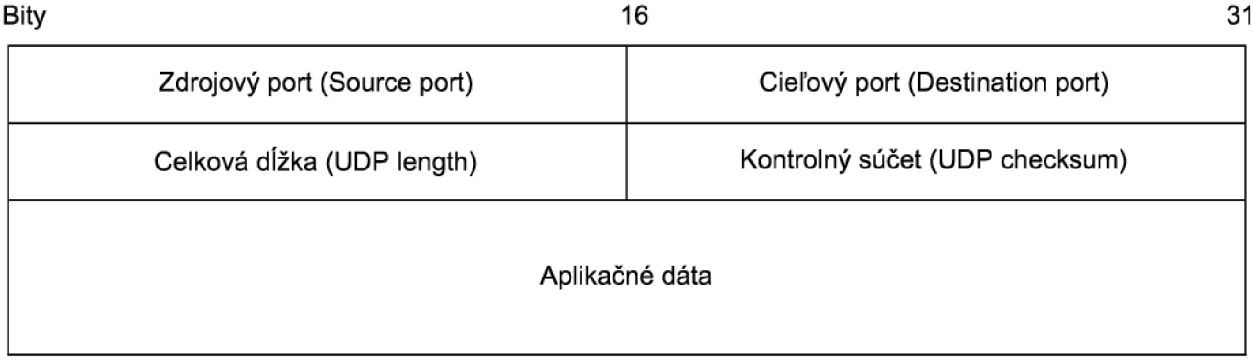
8 PROTOKOLY TRANSPORTNEJ VRSTVY

V tejto kapitole budú predstavené protokoly transportnej vrstvy: UDP a TCP. Budú popísané ďalej záhlavia týchto protokolov a význam jednotlivých polí. Nakoniec budú porovnané vlastnosti zmienených a niektorých ďalších protokolov.

8.1 UDP (User Datagram Protocol)

Protokol UDP umožňuje nespojový a nespoľahlivý charakter prenosu, teda poslané a doručené správy prijímač nepotvrdzuje. V prípade, že potvrdzovanie je nevyhnutné, je treba to zabezpečiť na aplikačnej vrstve. Dátové jednotky vytvorené týmto protokolom sa nazývajú datagramy [9]. Typicky sa tento protokol používa pri prenose hlasu (VoIP – Voice over IP) alebo videa v reálnom čase, kde nie je potrebné, aby bol doručený každý datagram, je dovolená určitá miera stratovosti – malé množstvo stratených dát neovplyvňuje výrazne kvalitu zvuku a obrazu na prijímacej strane.

Záhlavie protokolu UDP je kvôli maximálnemu zrýchleniu prenosu jednoduché, účel polí nevyžaduje ďalšie vysvetlenie (obr. 8-1).



Obr. 8-1 Záhlavie protokolu UDP [9]

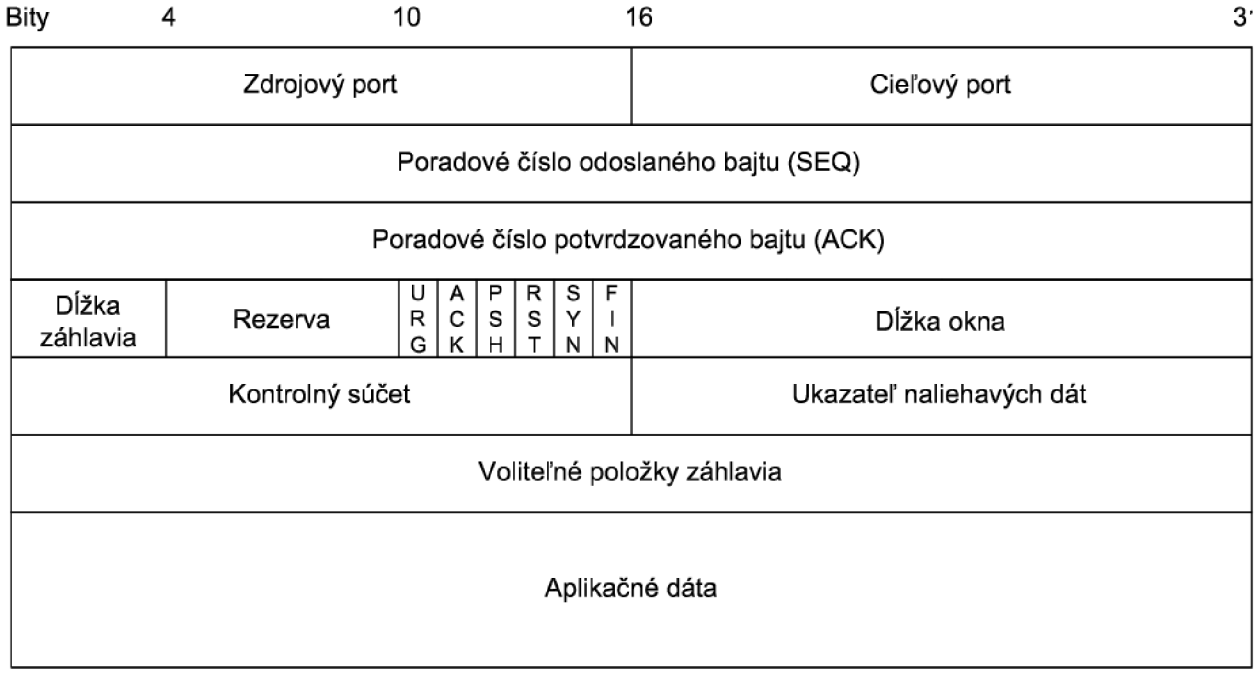
8.2 TCP (Transmission Control Protocol)

Protokol TCP je oproti protokolu UDP spojovo orientovaný a umožňuje spoľahlivý charakter prenosu, teda prenesené dátové jednotky (segmenty) sú potvrdzované. Ďalej pred samotným posielaním dát sa najprv vytvorí spojenie medzi komunikujúcimi stranami [9].

Záhlavie TCP (obr. 8-2) je oveľa rozsiahlejšie ako záhlavie UDP a obsahuje nasledujúce položky:

- Zdrojový port (Source port) – port odosielateľa.
- Cieľový port (Destination port) – port príjemcu.
- Poradové číslo odoslaného bajtu (Sequence number – SEQ) – odoslané bajty sa číslujú, toto pole obsahuje poradové číslo prvého bajtu z odoslaných v segmentu.
- Poradové číslo potvrdzovaného bajtu (Acknowledgement number – ACK) – slúži na potvrdenie dát prijímačom, toto pole obsahuje číslo nasledujúceho očakávaného bajtu.

- Dĺžka záhlavia (Header length).
- Príznakové bity (Flags) – je to 6-bitové pole, každý bit môže byť nastavený na hodnotu 1 (true) alebo 0 (false). V prípade hodnoty 1 majú nasledujúci význam:
  - URG (Urgent) – naliehavé dáta,
  - ACK (Acknowledgement) – indikuje, že číslo potvrdzovacieho bajtu je platné, teda segment potvrdzuje dáta (môže pri tom aj posielať dáta),
  - PSH (Push function) – dáta majú byť predané aplikácii a nemá sa čakať na prijatie ďalších segmentov,
  - RST (Reset the connection) – žiadosť o resetovanie spojenia,
  - SYN (Synchronize sequence number) – odosielateľ začína novú sekvenciu číslovania bajtov, používa sa pri naviazaní spojenia,
  - FIN (No more data from server) – používa sa pre ukončenie spojenia.
- Dĺžka okna (Window size) – maximálny počet bajtov, ktoré je možné poslať bez čakania na potvrdenie od príjemcu.
- Kontrolný súčet (TCP checksum).
- Ukazateľ naliehavých dát (Urgent pointer) – je vyplnený len v prípade, že príznak URG je nastavený na hodnotu 1.
- Voliteľné položky záhlavia (Options) – nepovinné pole, obsahuje voliteľné položky [9].



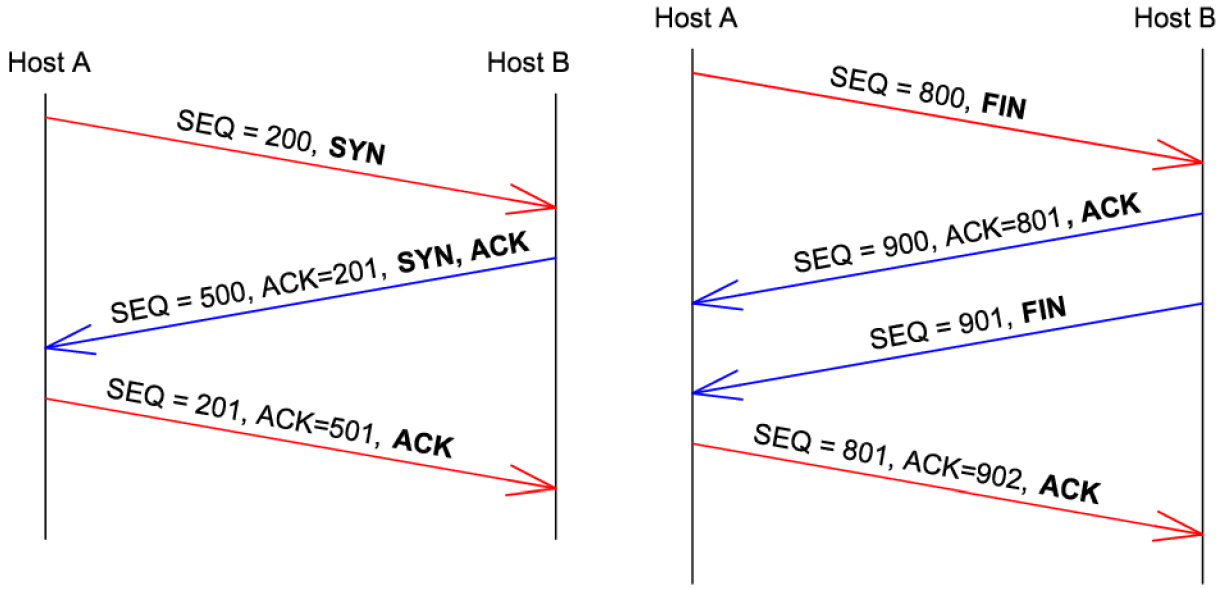
Obr. 8-2 Záhlavie protokolu TCP [9]

8.2.2 Priebeh naviazania a ukončenia spojenia

Ako to už bolo zmienené, protokol TCP pred samotným posielaním dát naviaže spojenie s druhou stranou. Priebeh naviazania sa nazýva *three-way-handshake* (trojcestné podanie rukou) a jednotlivé kroky sú nasledovné (obr. 8-3):

- Strana A, ktorá spojenie inicializuje, pošle segment s nastaveným príznakovým bitom SYN a nastaví sekvenčné číslo SEQ prvého bajtu na náhodnú hodnotu (200).
- Druhá strana odpovie nastaveným príznakom ACK a hodnotu poľa ACK nastaví na hodnotu ďalšieho očakávaného bajtu (201). Ďalej tiež nastaví bit SYN a náhodne zvolí sekvenčné číslo SEQ prvého bajtu (500).
- Strana A tiež pošle potvrdzovací segment s nastaveným príznakovým bitom ACK a pole ACK nastaví na hodnotu nasledujúceho očakávaného bajtu (501). Sekvenčné číslo je samozrejme zvýšené (201).
- Je treba zmieniť, že keď prenos dát je obojsmerný, čo je vo väčšine prípadov pravda, spojenie sa vytvára pre oba smery prenosu zvlášť [9].

Ukončenie TCP spojenia sa nazýva *four-way-handshake* (štvorcestné podanie rukou) a používa príznaky FIN a ACK, princíp je podobný ako u zostavení spojenia (obr. 8-3) [9].



Obr. 8-3 Nadväzovanie (naľavo) a ukončenie (napravo) spojenia protokolom TCP [9]

8.3 Porovnanie protokolov transportnej vrstvy

V tejto podkapitole budú porovnané vlastnosti protokolov transportnej vrstvy pomocou tabuľky. Pre porovnanie boli vybrané okrem TCP a UDP aj novšie, zatiaľ nie až tak používané protokoly, ktoré vznikli pridaním nových funkcií alebo zmiešaním funkcií UDP a TCP:

- SCTP (Stream Control Transmission Protocol),
- DCCP (Datagram Congestion Control Protocol),
- Multipath TCP.

Pomerne novou technikou na transportnej vrstve, ktorá je súčasťou porovnania, je multistreaming a multihoming, ich význam je nasledovný:

- Multihoming – umožňuje priradiť koncovému bodu pri naviazaní spojenia viac IP adries, jedna adresa bude primárna, druhá potom záložná. To je napr. u protokolu TCP nemožné, lebo spojenie je viazané ku konkrétnemu socketu (IP adresa a port).
- Multistreaming – umožňuje v rámci jedného transportného spojenia vytvoriť nezávislé prúdy dát. Dáta v rámci jednotlivých prúdov sú doručované zoradene, kým dáta v rôznych prúdoch nie sú nijak zoradené alebo zviazané. Podobnú funkciu je možné dosiahnuť aj s protokolom TCP, ale je treba k tomu vytvoriť viac samostatných spojení [9].

Tab. 8-1 Porovnanie vlastností transportných protokolov [9][16]

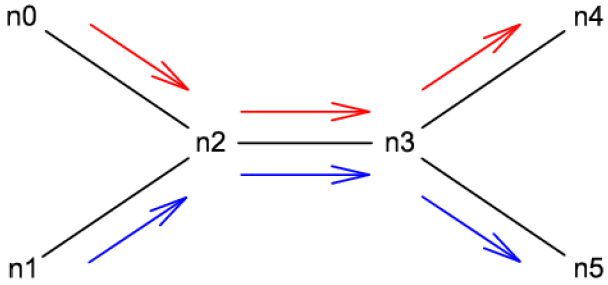
Vlastnosť	TCP	UDP	SCTP	DCCP	Multipath TCP
Spojovo orientovaný	áno	nie	áno	áno	áno
Spoľahlivý prenos	áno	nie	áno	nie	áno
Nespoľahlivý prenos	nie	áno	áno	áno	nie
Doručovanie častí dát v správnom poradí	áno	nie	áno	nie	áno
Doručovanie dát bez organizácie správneho poradia	nie	áno	áno	áno	nie
Riadenie toku dát	áno	nie	áno	nie	áno
Indikácia a riadenie a stavu zahltenia	áno	nie	áno	áno	áno
Zachovanie hraníc správ aplikačnej úrovne	nie	áno	áno	áno	nie
Multistreaming a multihoming	nie	nie	áno	nie	áno

9 Úloha 4: Porovnání transportních protokolů TCP a UDP

V této laboratorní úloze bude vytvořena jednoduchá síť s point-to-point linkami, která bude obsahovat 6 uzlů (obr. 9-1). Na rozdíl od předchozích úloh, v tomto případě bude největší část zdrojového kódu vytvořena a nastavena pomocí grafického rozhraní NS-3-Generator. Použitím aplikací `OnOffApplication` a `PacketSink` budou vytvořeny dva datové toky, jeden s protokolem TCP, druhý s protokolem UDP. Komunikace mezi zařízeními v tomto případě bude zajištěna jednoduše globálním směrováním – na začátku simulace jsou směrovací tabulky předem vytvořené a naplněné příslušnými cestami do každé sítě. V závěru budou analyzovány změřené parametry jako ztrátovost a jitter, dále budou porovnány záhlaví datových jednotek (TCP a UDP) a simulace bude zobrazena i graficky.

9.1 Topológia

Topologie úlohy je zobrazena na obr. 9-1, červené šipky představují provoz s protokolem TCP (n0 ↔ n4), modré šipky provoz s protokolem UDP (n1 ↔ n5).



Obr. 9-1 Topológie úlohy

9.2 NS-3-Generator

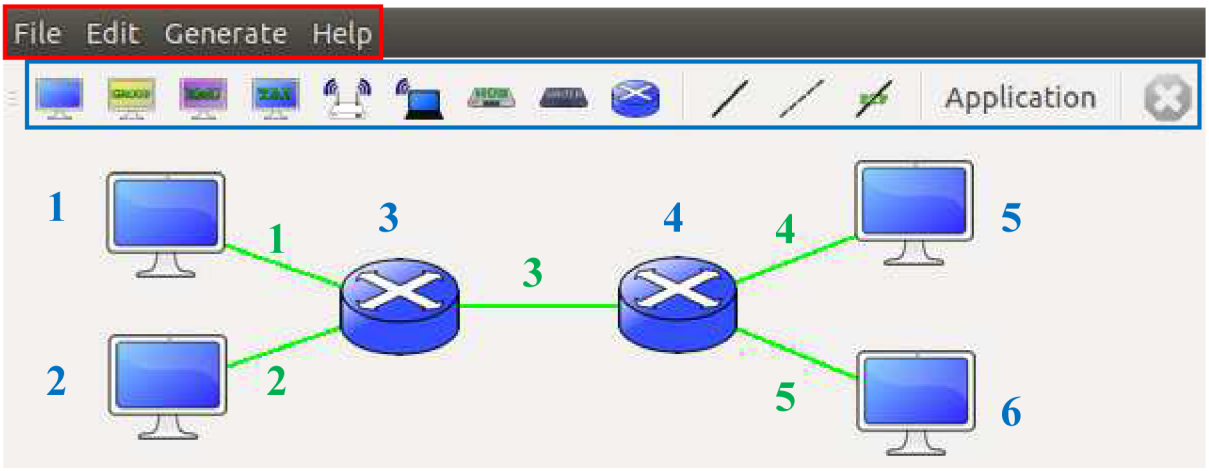
NS-3-Generator je jednoduchý program, který slouží k vytvoření a nastavení simulací pomocí grafického uživatelského prostředí (obr. 9-2). Hlavní výhodou je, že program nevytvoří pouze konfigurační soubor, který je v NS-3 dost náročné načíst, ale celý zdrojový kód se základními nastaveními od začátku do konce. Na druhé straně, NS-3-Generator byl vytvořen pro starší verzi NS-3, proto se jména některých tříd a hlavičkových souborů v generovaném souboru neshodují se jmény v aktuální verzi. Tyto malé chyby se ale dají pomocí debuggru snadno a rychle odstranit.

Základní ikony jsou na obr. 9-2 vyznačeny modrým obdélníkem a jejich význam z leva doprava je následující:

- vytvoření uzlu,
- vytvoření skupiny uzlů,
- vytvoření uzlu s reálným rozhraním,

- vytvoření uzlu s virtuálním rozhraním,
- vytvoření WiFi přístupového bodu,
- vytvoření stanice WiFI,
- vytvoření opakovače (hub),
- vytvoření přepínače (switch),
- vytvoření směrovače (router),
- vytvoření spojení CSMA,
- vytvoření bezdrátového spojení,
- vytvoření P2P spojení (bod-bod),
- vytvoření aplikace,
- smazání označeného objektu.

Hlavní menu programu NS-3-Generátor je také zobrazeno na obr. 9-2 a je označené červenou barvou. Pomocí položek *File → Sava as XML* a *File → Load as XML* je možné vytvořenou topologii uložit do XML souboru a později načíst. Generování C++ souboru se provádí pomocí položky *Generate → C++*, tento soubor ale není ještě spustitelný kvůli neplné kompatibilitě, jak to bylo vysvětleno výše.



Obr. 9-2 Grafické uživatelské rozhraní programu NS-3-Generator

9.3 Vytvoření modelu sítě

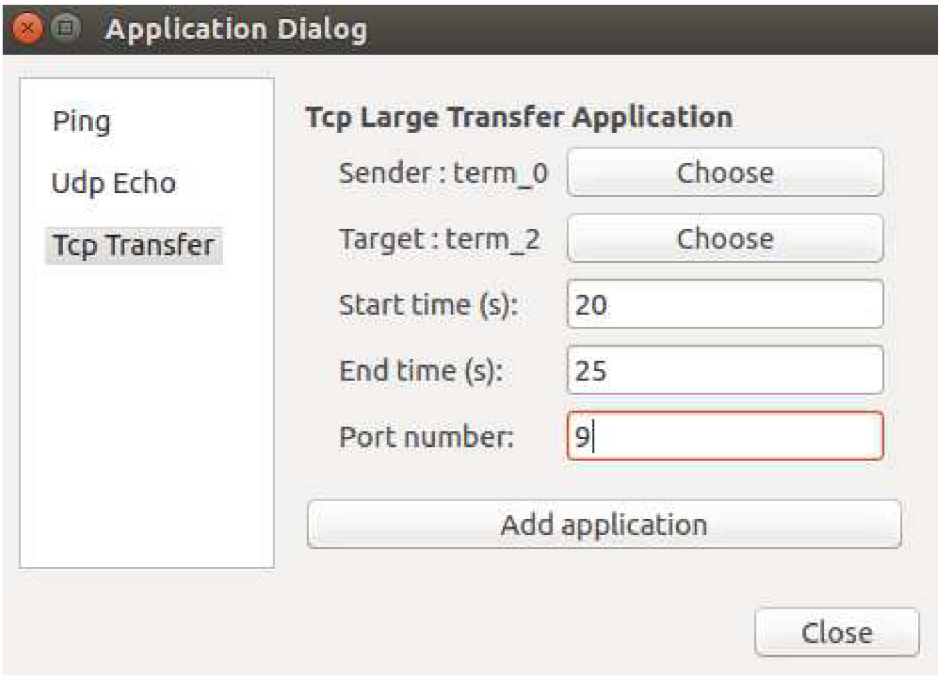
Otevřete program NS-3-Generator a vytvořte topologii podle obr. 9-2. Použijte k tomu ikonky jednoduchého uzlu (terminálu) a směrovače, dále jednotlivá zařízení propojte spojem bod-bod. **Dodržte pořadí vytvoření uzlů (modrá čísla) a spojů (zelená čísla) podle obr. 9-2, jinak hrozí možnost, že vám simulace nebude fungovat podle představ. Simulaci je nejlepší vytvořit bez mazání prvků, v opačném případě bude číslování zařízení odlišné od číslování uvedené v tomto návodu.**

V úloze budeme potřebovat dvě aplikace, které budou mít následující parametry:

- první aplikace bude posílat pakety z uzlu n0 na uzel n4,
- druhá aplikace bude posílat pakety z uzlu n1 na uzel n5,

- obě aplikace budou spuštěny v čase 20 s a zastaveny v čase 25 s, cílový port bude 9 (*well-known* echo port).

Pro nastavení aplikací použijeme ikonu *Application*, po vybrání této položky se objeví konfigurační okno (obr. 9-3). V otevřeném okně je třeba vybrat aplikaci *Tcp Transfer* (`OnOffApplication` s protokolem TCP) a pomocí tlačítka *Choose* vybrat zdrojové a cílové uzly (obr. 9-1), dále nastavit ostatní parametry. **V případě, že se vybrané uzly neuložily do pole** *Sender* **a** *Target*, **zavřete konfigurační okno tlačítkem** *Close* **(ne tlačítkem „X“) a zkuste to ještě jednou.** Obě aplikace je třeba nastavit s protokolem TCP, jedna z nich bude později přenastavena, aby používala protokol UDP. **Po ukončení konfigurace vygenerujte C++ soubor (při zadání jména nezapomeňte na příponu** .cc**), který takto přemístěte do adresáře Scratch.**



Obr. 9-3 Menu na vytvoření aplikace v programu NS-3-Generator

9.4 Oprava chyb v generovaném souboru

9.4.1 Hlavičkové soubory

Jak to již bylo zmíněno, generovaný zdrojový kód obsahuje drobné chyby, které v této části návodu opravíme. Kdybychom nyní spustili simulaci, dostali bychom chybu, že některé připojené hlavičkové soubory neexistují. **Namísto starších hlavičkových souborů připojte tedy novější, které jsou uvedeny níže.** Ve zdrojovém kódu budeme používat i proměnné typu `string`, proto je vhodné nastavit kromě jmenného prostoru `ns3` i standardní jmenný prostor `std`.

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

```
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/netanim-module.h"
#include "ns3/gnuplot.h"
#include "ns3/ipv4-global-routing-helper.h"

using namespace std;
```

9.4.2 Spojení bod-bod

Další chybou je nevhodná konfigurace spojení bod-bod, protože zpoždění je nastaveno na 10 s. V našem případě by pakety cestovali 30 s od zdroje k cíli (3 spoje mezi nimi), což znamená, že simulace by byla ukončena dříve, než pakety dorazí na cílový uzel. Je to dále velmi extrémní a nereálná hodnota. **Nastavte tedy zpoždění každého spojení bod-bod na hodnotu 2 ms, zdrojový kód po úpravě je zobrazen níže.** Přenosová rychlost je ve výchozím stavu nastavena na 100 Mb/s, která je v našem případě vyhovující hodnota.

```
PointToPointHelper p2p_p2p_0;
p2p_p2p_0.SetDeviceAttribute ("DataRate", DataRateValue (100000000));
p2p_p2p_0.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
PointToPointHelper p2p_p2p_1;
p2p_p2p_1.SetDeviceAttribute ("DataRate", DataRateValue (100000000));
p2p_p2p_1.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
PointToPointHelper p2p_p2p_2;
p2p_p2p_2.SetDeviceAttribute ("DataRate", DataRateValue (100000000));
p2p_p2p_2.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
PointToPointHelper p2p_p2p_3;
p2p_p2p_3.SetDeviceAttribute ("DataRate", DataRateValue (100000000));
p2p_p2p_3.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
PointToPointHelper p2p_p2p_4;
p2p_p2p_4.SetDeviceAttribute ("DataRate", DataRateValue (100000000));
p2p_p2p_4.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
```

9.4.3 Aplikace

Nakonec je nutné ještě opravit definici aplikací `OnOff` na více místech:

- Aplikace `OnOff` v generovaném souboru jsou defaultně nastaveny, aby stále posílali pakety, tedy atribut `OffTime` je nastaven na hodnotu 0. Tato vstupní hodnota je ale typu `RandomVariableValue`, což není správně, **proto tuto část kódu nahraďte u obou aplikací**:

```
"OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]")
```

- Dalším nesprávně nastaveným atributem je `OnTime`, ale vzhledem k tomu, že jsme už atribut `OffTime` nastavili, tento řádek by byl zbytečný. **Vymažte proto nastavení atributu `OnTime` u obou aplikací.**
- Na začátku úlohy bylo zmíněno, že jedna z vytvořených aplikací bude používat protokol TCP, druhá protokol UDP. V kap. 9.3 však byly nastaveny dvě aplikace s TCP protokolem (*Tcp Transfer*). Proto v tomto kroku přenastavíme druhou aplikaci tak, aby používala protokol UDP. **Docílíme toho tak, že vstupní parametr**

`ns3::TcpSocketFactory` **u vytvořených aplikací** `OnOff` a `PacketSink` **vyměníme za parametr** `ns3::UdpSocketFactory`**.**

Zdrojový kód pro vytvoření aplikací by po úpravách měl vypadat následovně:

```
//Prenos s TCP
uint16_t port_tcp_0 = 9;
Address sinkLocalAddress_tcp_0 (InetSocketAddress (Ipv4Address::GetAny (),
      port_tcp_0));
PacketSinkHelper sinkHelper_tcp_0 ("ns3::TcpSocketFactory",
      sinkLocalAddress_tcp_0);
ApplicationContainer sinkApp_tcp_0 = sinkHelper_tcp_0.Install (term_2);
sinkApp_tcp_0.Start (Seconds (20.0));
sinkApp_tcp_0.Stop (Seconds (25.0));
OnOffHelper clientHelper_tcp_0 ("ns3::TcpSocketFactory", Address ());
clientHelper_tcp_0.SetAttribute ("OffTime",
      StringValue("ns3::ConstantRandomVariable[Constant=0]"));
ApplicationContainer clientApps_tcp_0;
AddressValue remoteAddress_tcp_0 (InetSocketAddress
      (iface_ndc_p2p_3.GetAddress (1), port_tcp_0));
clientHelper_tcp_0.SetAttribute ("Remote", remoteAddress_tcp_0);
clientApps_tcp_0.Add (clientHelper_tcp_0.Install (term_0));
clientApps_tcp_0.Start (Seconds (20.0));
clientApps_tcp_0.Stop (Seconds (25.0));

//Prenos s UDP
uint16_t port_tcp_1 = 9;
Address sinkLocalAddress_tcp_1 (InetSocketAddress (Ipv4Address::GetAny (),
      port_tcp_1));
PacketSinkHelper sinkHelper_tcp_1 ("ns3::UdpSocketFactory",
      sinkLocalAddress_tcp_1);
ApplicationContainer sinkApp_tcp_1 = sinkHelper_tcp_1.Install (term_3);
sinkApp_tcp_1.Start (Seconds (20.0));
sinkApp_tcp_1.Stop (Seconds (25.0));
OnOffHelper clientHelper_tcp_1 ("ns3::UdpSocketFactory", Address ());
clientHelper_tcp_1.SetAttribute ("OffTime",
      StringValue("ns3::ConstantRandomVariable[Constant=0]"));
ApplicationContainer clientApps_tcp_1;
AddressValue remoteAddress_tcp_1 (InetSocketAddress
      (iface_ndc_p2p_4.GetAddress (1), port_tcp_1));
clientHelper_tcp_1.SetAttribute ("Remote", remoteAddress_tcp_1);
clientApps_tcp_1.Add (clientHelper_tcp_1.Install (term_1));
clientApps_tcp_1.Start (Seconds (20.0));
clientApps_tcp_1.Stop (Seconds (25.0));
```

9.4.4 Kontrola bezchybnosti

V tomto bodě byste měli být schopni spustit simulaci bez chyb, **k tomu abyste dostali i nějaký výstup, je nutné povolit logování aplikací** `OnOff` **a** `PacketSink` **hned na začátku hlavní funkce main**:

```
LogComponentEnable("OnOffApplication", LOG_LEVEL_INFO);
LogComponentEnable("PacketSink", LOG_LEVEL_INFO);
```

Níže je uvedena část výpisu logů, z něhož plyne, že obě nastavené aplikace fungují: jedna posílá pakety s protokolem TCP (označené červeně) a se zdrojovou IP adresou 10.0.0.2 na uzel s IP adresou 10.0.0.3.2, druhá posílá pakety s protokolem UDP (označené modrou) a se

zdrojovou adresou 10.0.1.2 na uzel s adresou 10.0.4.2. Všimnete si, že zprávy s protokolem TCP a UDP byly poslány ve stejnou dobu, ale zpráva s UDP došla o něco dříve (je to vidět z pořadí výpisu).

```
At time 24.9889s on-off application sent 512 bytes to 10.0.3.2 port 9 total
Tx 311808 bytes
At time 24.9889s on-off application sent 512 bytes to 10.0.4.2 port 9 total
Tx 311808 bytes
At time 24.9951s packet sink received 512 bytes from 10.0.1.2 port 49153
total Rx 311808 bytes
At time 24.9951s packet sink received 512 bytes from 10.0.0.2 port 49153
total Rx 311808 bytes
```

9.5 Rozšíření generovaného souboru

V této části návodu bude zdrojový kód rozšířen pohybovým modelem a řádky, které generují výstup pro program NetAnim. Dále budou přidány dvě funkce, které sledují posílání a příjem paketů. V rámci těchto funkcí bude zaznamenán čas těchto událostí a množství odeslaných a přijatých dat.

9.5.1 Definování pohybového modelu

Stanice se v této úloze pohybovat nebudou, ale na konci úlohy budeme simulaci zobrazovat graficky, proto pro všechny uzly nastavíme pohybový model pro stacionární uzly. **Níže uvedený zdrojový kód umístěte před řádek označený jako** *Build link*.

```
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.InstallAll();
```

9.5.2 Měření parametrů přenosu

Pro měření parametrů přenosu bude použita třída `Database`, která byla představena v kap. 7.3.6. Význam proměnných a účel funkcí se nezměnil, ale jednotlivé části byly přizpůsobeny k aktuální úloze. Definice této třídy se nachází v souboru *Database.txt*, **jeho obsah zkopírujte před hlavní funkci** `main`. **Dále vytvořte dva objekty tohoto typu hned po definici, které budou sloužit jako databáze pro naměřené hodnoty:**

```
Database tcp;
Database udp;
```

Funkce třídy `Database` budou volané při odeslání a příjmu paketů – tyto události monitoruje trasovací systém, na který se připojíme metodou `Config::Connect`. **Následující zdrojový kód umístěte před řádek označený jako** *Simulation*.

```
Config::Connect("/NodeList/0/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "tcp"));
Config::Connect("/NodeList/1/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "udp"));
Config::Connect("/NodeList/4/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback(&receivePacket, "tcp"));
Config::Connect("/NodeList/5/ApplicationList/*/$ns3::PacketSink/Rx",
```

```
MakeBoundCallback(&receivePacket, "udp"));
```

Když nastane příslušná událost, budou volané funkce `sendPacket` nebo `receivePacket` se vstupním parametrem typu `string` („udp“ nebo „tcp“), ve kterých se počítá celkové množství odeslaných a přijatých dat (`dataSent` a `dataReceived`), dále jsou uloženy časové hodnoty odeslání a příjmu paketů (`saveSentTime` a `saveReceivedTime`). **Níže uvedený zdrojový kód umístěte před hlavní funkci** `main`**.**

```
void sendPacket(string protocol, string path, Ptr<Packet const> packet) {
      if (protocol == "tcp") {
            tcp.dataSent = tcp.dataSent + packet->GetSize();
            tcp.saveSentTime(packet); }
      else if (protocol == "udp") {
            udp.dataSent = udp.dataSent + packet->GetSize();
            udp.saveSentTime(packet); }}

void receivePacket(string protocol, string path, Ptr<Packet const> packet,
Address const& address) {
      if (protocol == "tcp") {
            tcp.dataReceived = tcp.dataReceived + packet->GetSize();
            tcp.saveReceivedTime(packet, protocol); }
      else if (protocol == "udp") {
            udp.dataReceived = udp.dataReceived + packet->GetSize();
            udp.saveReceivedTime(packet); }}
```

9.5.3 Nastavení výstupu simulace

Prvním výstupem simulace budou naměřené parametry (množství odeslaných a přijatých dat, ztrátovost, zpoždění a jitter), které vypíšeme do konzolového okna. Na začátku je ale třeba volat funkci `calculateLatencyAndJitter` u obou objektů `tcp` a `udp`, která na základě uložených časových hodnot vypočítá zpoždění a jitter. **Následující řádky umístěte na konec zdrojového kódu:**

```
tcp.calculateLatencyAndJitter();
udp.calculateLatencyAndJitter();

NS_LOG_UNCOND("TCP data sent: " << tcp.dataSent << " B");
NS_LOG_UNCOND("UDP data sent: " << udp.dataSent << " B");
NS_LOG_UNCOND("TCP data received: " << tcp.dataReceived  << " B"
  << ", error rate: " << (1-tcp.dataReceived/tcp.dataSent)*100 << " %");
NS_LOG_UNCOND("UDP data received: " << udp.dataReceived  << " B"
  << ", error rate: " << (1-udp.dataReceived/udp.dataSent)*100 << " %");
NS_LOG_UNCOND("Average latency for TCP: " << tcp.averageLatency << " ms");
NS_LOG_UNCOND("Maximum latency for TCP: " << tcp.maxLatency << " ms");
NS_LOG_UNCOND("Average jitter for TCP: " << tcp.averageJitter << " ms");
NS_LOG_UNCOND("Average latency for UDP: " << udp.averageLatency << " ms");
NS_LOG_UNCOND("Maximum latency for UDP: " << udp.maxLatency << " ms");
NS_LOG_UNCOND("Average jitter for UDP: " << udp.averageJitter << " ms");
```

Následuje vytvoření XML souboru pro NetAnim a nastavení pozice uzlů. Docílíme toho známou třídou `AnimationInterface`, dále povolíme i zaznamenávání metadat paketů. Na konec definujeme vytvoření trasovacích souborů, **níže uvedený zdrojový kód umístěte před řádek označený jako** *Simulation***.**

```
AnimationInterface anim("tcp-udp.xml");
anim.EnablePacketMetadata(true);
anim.SetConstantPosition(term_0.Get(0), 0, 0);
anim.SetConstantPosition(term_1.Get(0), 0, 50);
anim.SetConstantPosition(router_0.Get(0), 50, 25);
anim.SetConstantPosition(router_1.Get(0), 100, 25);
anim.SetConstantPosition(term_2.Get(0), 150, 0);
anim.SetConstantPosition(term_3.Get(0), 150, 50);

p2p_p2p_3.EnablePcapAll("tcp-udp");
```

9.6 Výsledky simulace

Na začátku se pomocí logových zpráv přesvědčte, že obě aplikace `OnOff` fungují a že aplikace `PacketSink` data přijímají. V dalších krocích logové zprávy už potřebovat nebudeme, proto je můžete vypnout (zakomentovat).

9.6.1 Výstup z příkazového řádku

Prvním výstupem simulace je výpis naměřených parametrů přenosu do příkazového řádku. Z níže zobrazených výsledků je vidět, že obě aplikace poslali 312 320 B, přenos s protokolem TCP byl bezchybný a přenos s UDP proběhl se ztrátovostí přibližně 0,16%. Zajímavější jsou parametry zpoždění a jitter, z nichž je zřejmé, že přenos s protokolem TCP měl o něco vyšší zpoždění i přesto, že jsme v simulaci použili velmi malou a jednoduchou síť. Maximální zpoždění je však mnohem větší u protokolu TCP, více než dvojnásobek průměrné hodnoty. Je důležité si uvědomit, že v tomto případě relativně velkou část zpoždění přidávají spojení bod-bod – přenos přes tři spojení, pro každé jsme nastavili zpoždění 2 ms. Posledním měřeným parametrem byl jitter, který byl mnohem nižší v případě přenosu s protokolem UDP, jak jsme to i očekávali.

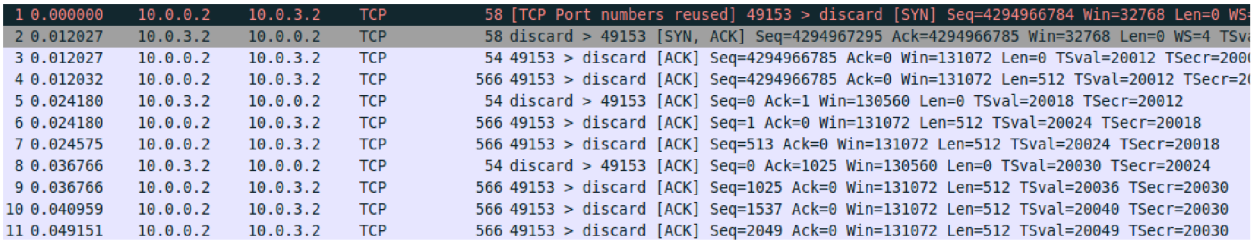
Poznámka: v dalších částech úlohy bude simulované přetížení a výpadek sítě, proto vyrovnávací paměť protokolu TCP bude mít velký vliv na zpoždění. Z tohoto důvodu se měření zpoždění provádí na aplikační vrstvě, kde máme ale určitá omezení. Jediným identifikátorem zprávy na této vrstvě je tzv. globální identifikátor (UID – Unique Identifier), který je číslo přidělené v rámci NS-3, nejedná se o pole žádného záhlaví. Na měření zpoždění u protokolu UDP byla implementována metoda na základě tohoto identifikátoru, v případě protokolu TCP se však čísla UID zpráv mění v průběhu přenosu. Proto zpoždění u protokolu TCP je měřeno od odeslání k následujícímu příjmu. Tato skutečnost zkreslí naměřené hodnoty, které nejsou z tohoto důvodu přesné, ale jen směrodatné.

```
TCP data sent: 312320 B
UDP data sent: 312320 B
TCP data received: 312320 B, error rate: 0 %
UDP data received: 311808 B, error rate: 0.163934 %
Average latency for TCP: 6.20271 ms
Maximum latency for TCP: 13.9328 ms
Average jitter for TCP: 0.0322264 ms
Average latency for UDP: 6.13008 ms
Maximum latency for UDP: 6.13008 ms
Average jitter for UDP: 2.00424e-12 ms
```

9.6.2 Trasovací soubory

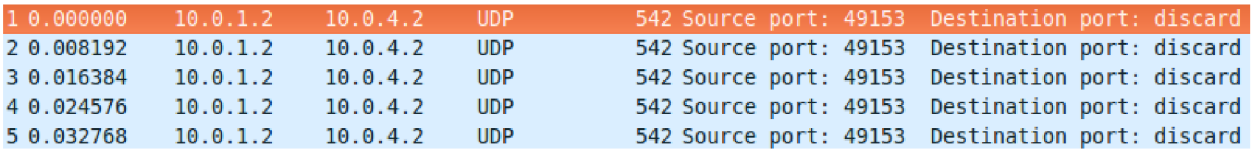
Dalším výstupem simulace jsou trasovací soubory, pro nás budou zajímavá data z uzlu 0 a 1 (pokud jste dodrželi pořadí vytvoření uzlů a spojů, tak se jedná o soubory *tcp-udp-0-0.pcap* a *tcp-udp-1-0.pcap*). Postupně si tedy otevřete oba dva soubory a prostudujte si průběh komunikace a záhlaví transportních protokolů.

U protokolu TCP je vidět, že hned na začátku bylo sestaveno spojení segmenty s příznakovými bity [SYN] – [SYN, ACK] – [ACK] (obr. 9-4). Když otevřete libovolný paket, můžete snadno zjistit, že nastavená aplikace posílá data s velikostí 512 B. Z dalších záznamů v trasovacom souboru je zřejmé, že se po ustálení situace jsou potvrzovací zprávy posílané mnohem častěji, než bychom to předpokládali z hodnoty velikosti okna. Tím pádem zdrojový uzel může posílat data bez přerušení a poslané segmenty jsou průběžně potvrzovány. Na konci přenosu je pak sestavené spojení ukončeno segmenty s nastavenými příznaky [FIN] – [ACK].



Obr. 9-4 Sestavení spojení a přenos dat s protokolem TCP

V případě protokolu UDP však podobný mechanismus jako sestavení a ukončení spojení nebo potvrzování přijatých paketů neexistuje. Je vidět, že komunikace je jednostranná (obr. 9-5), v případě že se na cestě ztratí paket, protokol UDP na to nijak nereaguje.



Obr. 9-5 Přenos dat s protokolem UDP

Na obr. 9-6 a obr. 9-7 jsou zachyceny záhlaví TCP segmentu a UDP datagramu s pořadovým číslem 16. Hned na první pohled je zřejmé, že záhlaví UDP datagramu je mnohem kratší, obsahuje pouze čísla zdrojového a cílového portu, celkovou délku datagramu a kontrolní součet. V záhlaví TCP segmentu existují navíc pole na zajištění opakovaného přenosu v případě chyby (ACK, SEQ, Window size). Dále pole Flags (příznakové bity) přiřadí speciální funkci TCP segmentu, které nebudou v tomto návodu blíže popsány.

```
Transmission Control Protocol, Src Port: 49153 (49153), Dst Port: discard (9), Seq: 3585, Ack: 0, Len: 512
  Source port: 49153 (49153)
  Destination port: discard (9)
  [Stream index: 0]
  Sequence number: 3585    (relative sequence number)
  [Next sequence number: 4097    (relative sequence number)]
  Acknowledgment number: 0    (relative ack number)
  Header length: 32 bytes
▶ Flags: 0x010 (ACK)
  Window size value: 32768
  [Calculated window size: 131072]
  [Window size scaling factor: 4]
▶ Checksum: 0x0000 [validation disabled]
▶ Options: (12 bytes), Timestamps, End of Option List (EOL)
▶ [SEQ/ACK analysis]
```

Obr. 9-6 Záhlaví TCP segmentu č. 16

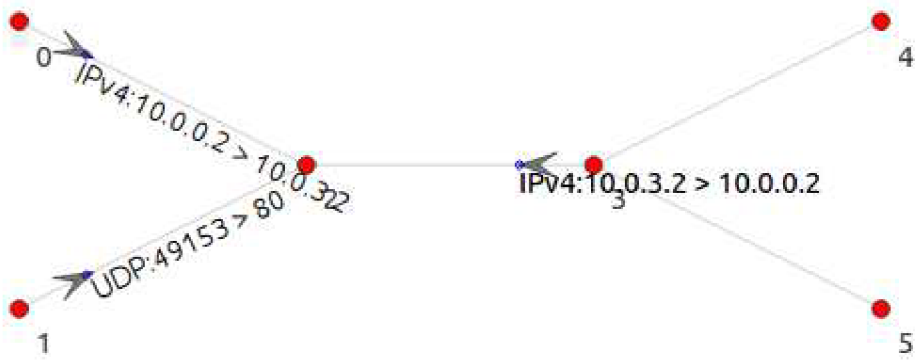
```
User Datagram Protocol, Src Port: 49153 (49153), Dst Port: discard (9)
  Source port: 49153 (49153)
  Destination port: discard (9)
  Length: 520
▶ Checksum: 0x0000 (none)
```

Obr. 9-7 Záhlaví UDP datagramu č. 16

9.6.3 Výstup z programu NetAnim

Posledním výstupem simulace je XML soubor pro NetAnim, otevřete tedy generovaný soubor *tcp-udp.xml* a spusťte animaci. Grafické zobrazení simulace potvrdí dosavadní předpoklady:

- uzel n0 posílá data uzlu n4 s protokolem TCP,
- uzel n1 posílá data uzlu n5 s protokolem UDP,
- přenášené TCP segmenty jsou průběžně potvrzovány, zatímco UDP datagramy ne.



Obr. 9-8 Simulace zobrazena v programu NetAnim

9.6.4 Kontrolní otázky

- Popište průběh a ukončení spojení s protokolem TCP. Jakou mají úlohu pole Sequence number, Acknowledgement number a Window size?

- Pro jaké aplikace je vhodný protokol TCP a pro jaké protokol UDP? Odpověď vysvětlete, uveďte pár příkladů.
- Co se stane v případě poruchy na síti u protokolů TCP a UDP?
- I ze simulace je vidět, že TCP segment při přenosu má o něco větší zpoždění, než UDP datagram, proč je tomu tak?

9.7 Doplňující úkoly

9.7.1 Simulace poruchy linky

V této části laboratorní úlohy bude zkoumáno, jak reagují protokoly TCP a UDP na krátkodobou poruchu na přenosové cestě – bude vypnuto rozhraní mezi uzly n2 a n3 (obr. 9-1). Pozorované bude zpoždění a ztrátovost obou přenosů.

Pro vypnutí rozhraní použijeme ukazatel na IPv4 protokolovou sadu uzlu n2 (ve zdrojovém kódu označen jako `router_0`) a pomocí metody `Schedule` naplánujeme vypnutí rozhraní č. 3 (rozhraní mezi uzly n2 a n3) v čase 21 s. Toto rozhraní pak opět zapneme v čase 21,5 s, výpadek bude tedy trvat půl sekundy. **Následující zdrojový kód umístěte před řádek označený jako** *Simulation***.**

```
Ptr<Ipv4> IPv4 = router_0.Get(0)->GetObject<Ipv4> ();
Simulator::Schedule(Seconds(21), &Ipv4::SetDown, IPv4, 3);
Simulator::Schedule(Seconds(21.5), &Ipv4::SetUp, IPv4, 3);
```

Analýza výsledků

- Spusťte simulaci a výsledky vysvětlete. Proč měl přenos s protokolem TCP nulovou ztrátovost, ale vyšší zpoždění a přenos s protokolem nenulovou ztrátovost, ale stejné zpoždění jako v předchozím případě? Všimnete si i to, že v tomto případě již maximálně zpoždění s protokolem TCP byl mnohonásobně více, než průměrná hodnota.

```
TCP data sent: 312320 B
UDP data sent: 312320 B
TCP data received: 312320 B, error rate: 0 %
UDP data received: 280576 B, error rate: 10.1639 %
Average latency for TCP: 153.411 ms
Maximum latency for TCP: 1005.91 ms
Average jitter for TCP: 3.36971 ms
Average latency for UDP: 6.13017 ms
Maximum latency for UDP: 6.16242 ms
Average jitter for UDP: 0.000189799 ms
```

- Znovu otevřete soubor *tcp-udp-0-0.pcap* a prostudujte průběh komunikace s protokolem TCP, zaměřte se hlavně na část, kde byla komunikace přerušena. Co se stalo v momentě, kdy protokol TCP zjistil, že došlo k chybě při přenosu? Jak se zjistí u protokolu TCP, že přenos nebyl úspěšný?
- Otevřete soubor *tcp-udp-0-1.pcap* a prostudujte průběh komunikace i s protokolem UDP. Jak reagoval tento protokol na poruchu na přenosové trase?

9.7.2 Simulace přetížení sítě

V poslední části úlohy bude simulováno, jak reagují protokoly TCP a UDP na přetížení sítě. Dosáhneme toho snížením velikosti vyrovnávací paměti uzlu n2 a snížením přenosové rychlosti spoje bod-bod mezi uzly n2-n3. Během simulace bude pozorované zatížení linek mezi uzly n0-n2 (provoz TCP) a n1-n2 (provoz UDP), výsledky budou zobrazeny ve formě grafu. Na konec bude ještě přidán zdrojový kód na měření počtu opětovně přenesených paketů.

Na začátku odstraňte řádky na vypnutí linky mezi uzly n2 a n3, které byly vloženy v předchozí části. Dále nastavte nové parametry aplikací podle následujících bodů:

- aplikace `OnOff` s protokolem TCP (`clientApp`) bude posílat data v čase 20÷40 s,
- aplikace `PacketSink` (`sinkApp`) pro přenos s protokolem TCP bude spuštěn v čase 20÷45 s,
- aplikace `OnOff` s protokolem UDP bude posílat data v čase 30÷40 s,
- aplikace `PacketSink` pro přenos s protokolem UDP bude spuštěn v čase 30÷45 s.

Přetížení sítě dosáhneme snížením velikosti vyrovnávací paměti na posílání na uzlu n2 na hodnotu 40 paketů a snížením přenosové rychlosti spoje bod-bod mezi uzly n2-n3 na 0,95 Mb/s (datové rychlosti obou aplikací je nastavena na 0,5 Mb/s). Slouží k tomu následující zdrojový kód, **se kterým nahraďte aktuální nastavení spoje označeného jako** `p2p_p2p_2`:

```
PointToPointHelper p2p_p2p_2;
p2p_p2p_2.SetDeviceAttribute ("DataRate", DataRateValue (950000));
p2p_p2p_2.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
p2p_p2p_2.SetQueue ("ns3::DropTailQueue", "MaxPackets", StringValue ("40"));
```

Jak to již bylo zmíněno, bude pozorované zatížení linek mezi uzly n0-n2 a n1-n2, k tomu potřebujeme opět trasovací systém. Vstupní hodnotou je cesta k požadovanému zdroji, v tomto případě to je `Ipv4L3Protocol/Tx`, který indikuje odchod IPv4 paketů. Dalším vstupním parametrem je funkce, kterou chceme volat (`trasmitPacket`), když nastane zmíněná událost, posledním parametrem je typ protokolu. **Uvedený zdrojový kód umístěte před řádek označený jako** *Simulation***.**

```
Config::Connect("/NodeList/0/$ns3::Ipv4L3Protocol/Tx",
      MakeBoundCallback(&trasmitPacket, "tcp"));
Config::Connect("/NodeList/1/$ns3::Ipv4L3Protocol/Tx",
      MakeBoundCallback(&trasmitPacket, "udp"));
```

Volána funkce `trasmitPacket` jednoduše počítá množství přenesených dat a postupně přičítá k proměnné `throughput`. První podmínka zjišťuje, zda je paket větší než 512 B (zda byl poslán námi vytvořenými aplikacemi), druhá pak zda jde o protokol TCP nebo UDP. **Uvedený zdrojový kód umístěte před hlavní funkci** *main***.**

```
void trasmitPacket(string protocol, string path, Ptr<Packet const> packet,
Ptr<Ipv4> ipv4, unsigned int cs) {
      if(packet->GetSize() > 512) {
            if(protocol == "tcp")
                  tcp.throughput = tcp.throughput + packet->GetSize();
            else if(protocol == "udp")
                  udp.throughput = udp.throughput + packet->GetSize(); }}
```

Zatížení linek budeme chtít zobrazit i graficky, použijeme na to proměnné typu `Gnuplot2dDataset`, které je již vytvořeno v třídě `Database` a která bude sloužit jako databáze naměřených hodnot při vytvoření grafu. Uložení hodnot do databáze bude provádět další funkce `saveThroughput`. Na začátku této funkce je nastaven interval „vzorkování“ na 0,5 s (`step`). Po uložení (funkce `Add`) naměřených hodnot do databáze jsou proměnné `throughput` nastaveny na nulu (příprava na další měření) a nakonec je funkce `saveThroughput` volána ještě jednou, ale o 0,5 s později – takto je volána funkce pravidelně každou půl sekundy. **Uvedený zdrojový kód umístěte před hlavní funkci** `main`**.**

```
void saveThroughput() {
      double step = 0.5;
      tcp.throughputDatabase.Add(Simulator::Now().GetSeconds(),
            tcp.throughput/1024/1024*8/step);
      udp.throughputDatabase.Add(Simulator::Now().GetSeconds(),
            udp.throughput/1024/1024*8/step);

      tcp.throughput = 0;
      udp.throughput = 0;

      Simulator::Schedule(Seconds(step), &saveThroughput);
}
```

První volání výše uvedené funkce by mělo být provedeno v čase 20 s (spuštění aplikací), proto následující řádek **vložte do zdrojového kódu před řádek označený jako** *Simulation*.

```
Simulator::Schedule(Seconds(20), &saveThroughput);
```

Pro vytvoření grafů použijeme opět program Gnuplot, pomocí funkce `AddDataset` mu předáme obe dvě databáze naměřených hodnot. Poslední dva řádky jsou příkazy pro operační systém, které budou provedeny již mimo simulace. První příkaz vytvoří grafický soubor (*.png*) z datového souboru (*.plt*), druhý příkaz otevře grafický soubor, není třeba to dělat manuálně přes příkazový řádek. **Uvedený řádky umístěte na konec zdrojového kódu.**

```
string fileNameWithNoExtension = "throughput";
string graphicsFileName = fileNameWithNoExtension + ".png";
string plotFileName = fileNameWithNoExtension + ".plt";
string plotTitle = "Throughput";

tcp.throughputDatabase.SetTitle("TCP");
tcp.throughputDatabase.SetStyle(Gnuplot2dDataset::LINES_POINTS);
udp.throughputDatabase.SetTitle("UDP");
udp.throughputDatabase.SetStyle(Gnuplot2dDataset::LINES_POINTS);

Gnuplot plot(graphicsFileName);
plot.SetTitle(plotTitle);
plot.SetTerminal("png");
plot.SetLegend("Time (s)", "Throughput (Mb/s)");
plot.AddDataset(tcp.throughputDatabase);
plot.AddDataset(udp.throughputDatabase);

ofstream plotFile(plotFileName.c_str());

plot.GenerateOutput(plotFile);
plotFile.close();
system("gnuplot throughput.plt");
```

```
system("gnome-open throughput.png");
```

Na konec ještě vytvoříme algoritmus pro měření počtu opětovně přenesených paketů. Použijeme na to nepřímou metodu: bude měřený počet zahozených paketů s protokolem TCP na uzlu n2, které bude muset TCP posílat ještě jednou. Pro indikaci zahození paketů použijeme trasovací systém, v tomto případě se připojíme na `PointToPointNetDevice` uzlu n2 – slovo „MacTxDrop“ označuje zahození rámců, tedy i zapouzdřených paketů. **Uvedený zdrojový kód umístěte před řádek označený jako** *Simulation***.**

```
Config::Connect
("/NodeList/2/DeviceList/*/$ns3::PointToPointNetDevice/MacTxDrop",
MakeCallback(&dropPacket));
```

Když nastane zahození rámce na uzlu n2, bude volána funkce `dropPacket`, která jednoduše počítá množství zahozených paketů pomocí proměnné `retransmittedPackets`. Podmínka zajistí, aby zahozené pakety s protokolem UDP nebyly započítány. **Uvedenou funkci umístěte před hlavní funkci** `main`**.**

```
void dropPacket(string path, Ptr<Packet const> packet) {
      if(packet->GetSize() >= 552)
            retransmittedPackets++;
}
```

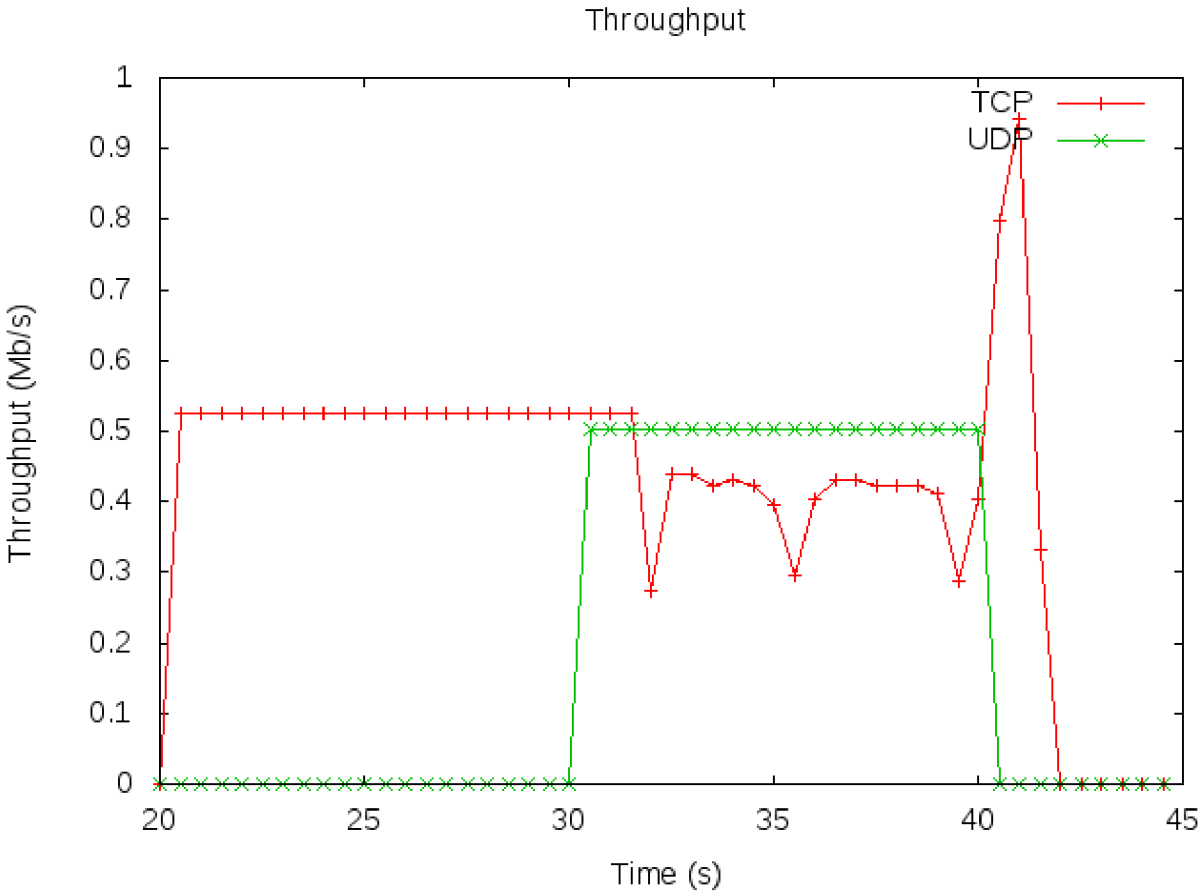
Výpis počtu opakovaně posílaných paketů provedeme podobně, jako v předchozích částech. **Uvedený řádek zkopírujte na místo, kde se vypisují ostatní měřené parametry.**

```
NS_LOG_UNCOND("Number of retransmitted packets: " << retransmittedPackets);
```

Analýza výsledků

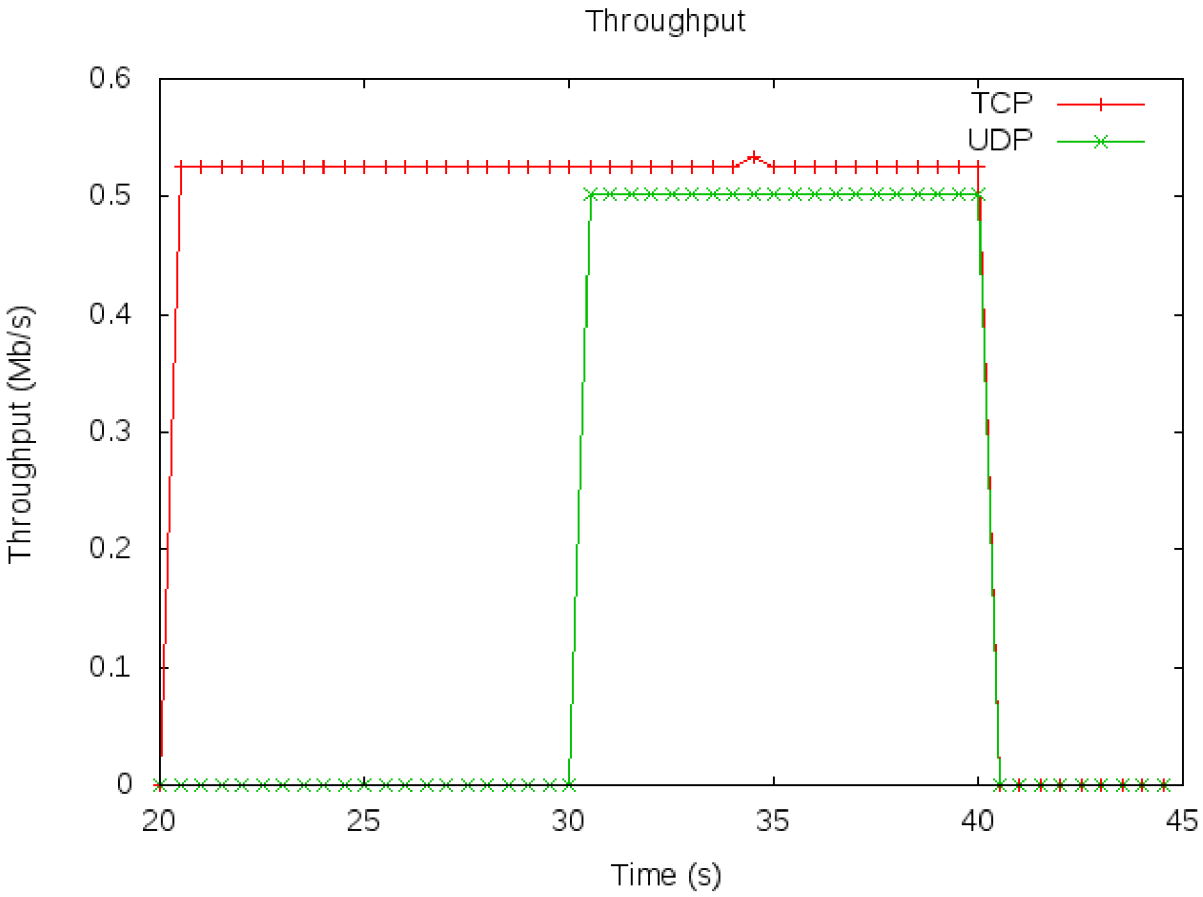
- Po spuštění simulace byste měli dostat podobný graf, jako na obr. 9-9. Proč má přenos s protokolem TCP od okamžiku 20 s nižší přenosovou rychlost? Jaká technika se používá k řízení toku u protokolu TCP, vysvětlete její princip. Všimnete si, že počet znovu přenesených paketů je 10, což není až tak velká hodnota. Při větším zatížení sítě by se to měla zvětšovat.

Poznámka: protokol TCP používá mechanismy potvrzování a opětovného přenosu, aby posílány data se určitě dostali k přijímací straně. Z konzolového výstupu ale můžeme odečíst, že přenos s protokolem TCP měl určitou ztrátovost, tedy výsledky se liší od předpokladů. Příčina odlišného chování protokolu TCP nebyla odhalena při psaní této práce, bude to pravděpodobně chyba v NS-3.



Obr. 9-9 Zatížení linek mezi uzly n0-n2 (TCP) a n1-n2 (UDP) při přetížení sítě

- Zkuste si zvýšit přenosovou rychlost spoje `p2p_p2p_2` na více než 1 Mb/s, výsledky opět komentujte. Proč má přenos s protokolem TCP o něco vyšší přenosovou rychlost, když aplikace jsou nastaveny na stejnou rychlost 0,5 Mb/s



Obr. 9-10 Zatížení linek mezi uzly n0-n2 (TCP) a n1-n2 (UDP) bez přetížení sítě

10 ZÁVER

Cieľom diplomovej práce bol preštudovať protokoly sieťovej vrstvy IPv4 a IPv6, protokoly transportnej vrstvy UDP a TCP, fungovanie unicastového a multicastového prenosu, ďalej problematiku zaistení kvality služieb. Na základe získaných poznatkov potom vytvoriť štyri laboratórne úlohy v NS-3, ktoré sa zaoberajú s týmito tematikami.

Prvá teoretická časť práce sa zaoberá s programom NS-3, ktorý je simulátor sieťových technológií založený na jazyku C++, neobsahuje však žiadne grafické rozhranie na vytvorenie simulácie. Preto v nasledujúcej časti bolo popísané vývojové prostredie Eclipse IDE, ktorý slúži na písanie zdrojového kódu a prináša veľa výhod pre programátora oproti jednoduchým textovým editorom. Bol uvedený podrobný postup pre spojenie týchto nástrojov, čo je nevyhnutné z hľadiska efektivity práce.

V ďalšej kapitole bol rozobratý protokol IPv4 a jeho následník, IPv6. Boli predstavené vlastnosti oboch protokolov, položky záhlaví a spôsob fragmentácie. Boli vystihnuté najmä rozdiely, výhody a nevýhody protokolu IPv6 oproti IPv4.

Prvá praktická časť sa zaoberá s protokolmi IPv4 a IPv6 a ich porovnaním. Bol uvedený postup prvej laboratórnej úlohy, kde bol definovaný súčasný beh týchto protokolov. V rámci úlohy boli vytvorené dve aplikácie, jeden používal protokol IPv4, druhý IPv6. Zdrojový kód obsahuje aj algoritmus pre meranie hodnoty RTT medzi vybranými koncovými zariadeniami v prípade oboch protokolov, bol k tomu použitý trasovací systém. Na konci úlohy bol vytvorený trasovací súbor pre vybraný uzol, pomocou ktorého boli porovnané záhlavia protokolov IPv4 a IPv6. V rámci samostatnej úlohy bola zmenená veľkosť paketov a MTU linky bod-bod a bolo pozorované, ako sa jednotlivé protokoly chovajú v rôznych prípadoch.

Druhá teoretická časť popisuje typy prenosov v dátových sieťach a zaoberá sa podrobnejšie s multicastovým prenosom. V tejto kapitole boli vysvetlené výhody a nevýhody multicastového prenosu oproti unicastovému, ďalej adresovanie multicastových dát na sieťovej a spojovej vrstve. Bol predstavený najpoužívanejší multicastový protokol pre zaistenie prevádzky medzi smerovačmi (PIM) a tiež protokol pre komunikáciu medzi smerovačom a koncovou stanicou (IGMP).

V ďalšej časti práce bola vytvorená druhá laboratórna úloha, ktorá sa zaoberá s porovnaním unicastového a multicastového prenosu. K tejto úlohe bol vytvorený začiatočný súbor, pretože definícia uzlov, spojenia medzi nimi a pridelenie adresných priestorov je zdĺhavý. Topológia bola vytvorená tak, že jeden zdroj posiela dáta štyrom klientom najprv unicastovým, potom multicastovým prenosom. Bol vytvorený ďalej algoritmus, ktorý meria množstvo vytvorených, prenesených a zahodených dát za sekundu. Na konci sú porovnané výsledky simulácie s unicastovým a multicastovým prenosom. V rámci samostatnej úlohy bola zvýšená bitová rýchlosť generovaných dát na vyššiu hodnotu, než prenosová rýchlosť linky, cez ktorú je koncová stanica pripojená k sieti.

V nasledujúcej kapitole bola rozobratá problematika odlišného zaobchádzania sa s dátovými jednotkami pri prenose a boli krátko popísané mechanizmy, ktoré sa využívajú pri

zaistení kvality služieb. Na konci bola predstavená prístupová metóda EDCA, ktorá sa používa v bezdrôtových sieťach na zaistenie kvality služieb.

V nasledujúcej časti práce bola vytvorená simulácia so štyrmi uzlami a zdieľaným bezdrôtovým kanálom. Súčasne boli vytvorené dva dátové toky, pri čom ich priorita bola nastavená na „best-effort“ a „video“. Prenosová rýchlosť bezdrôtového kanálu bola nastavená, aby bola mierne preťažená. Pre analýzu komunikácie na linkovej vrstve bol použitý program Wireshark – boli predstavené položky záhlaví pre zaistenie kvality služieb na linkovej vrstve v bezdrôtových sieťach. Ďalším výstupom bol výpis nameraných parametrov do príkazového riadku pomocou ktorých boli porovnané stratovosť a oneskorenie prenosov. V samostatnej úlohe bola priorita prenosov rôzne nastavovaná a opäť boli pozorované parametre siete.

Nasledujúca teoretická kapitola sa zaoberá s transportnými protokolmi TCP a UDP. Bola uvedená štruktúra ich záhlaví s popisom jednotlivých polí. U protokolu TCP bol ďalej vysvetlený princíp zostavenia a ukončenia spojenia. Na konci boli protokoly TCP a UDP porovnané s ďalšími, nie až tak známymi protokolmi transportnej vrstvy, ako SCTP, DCCP a Multipath TCP.

V poslednej časti práce bol vytvorený postup laboratórnej úlohy pre porovnanie protokolov TCP a UDP. V tejto úlohe bol použitý program NS-3-Generator, ktorý umožňuje vytvoriť topológiu simulácie pomocou grafického užívateľského rozhrania. Po vytvorení topológie boli nastavené dva dátové toky, jeden s protokolom TCP, druhý s protokolom UDP. Vygenerovaný C++ súbor bol ďalej rozšírený s meraním parametrov siete, ako stratovosť, oneskorenie a jitter, ktoré boli použité na porovnanie zmienených protokolov. Ďalej pomocou programu Wireshark bolo predstavené zostavenie a ukončenie spojenia u protokolu TCP. V doplňujúcej úlohe bola krátkodobo prerušená linka u jedného z centrálnych prvkov a bol pozorovaný vplyv tejto udalosti na fungovanie protokolov TCP a UDP. Ďalej bolo simulované mierne preťaženie siete a opäť bola pozorovaná reakcia transportných protokolov.

POUŽITÁ LITERATÚRA

[1] Ns-3. *Ns-3* [online]. 2014 [cit. 2014-10-20]. Dostupné z: http://www.nsnam.org/

[2] VADKERTI, G. Vytvoření experimentální MANET sítě v simulátoru NS-3. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 57 s. Vedoucí semestrální práce Ing. Milan Bartl.

[3] HOŠEK, Jiří. Pokročilé komunikační techniky – laboratorní cvičení. Brno, 2013.

[4] *Ns-3: ns-3 Documentation* [online]. 2014 [cit. 2014-10-20]. Dostupné z: http://www.nsnam.org/doxygen/

[5] Installation - Nsnam. *Nsnam* [online]. 2014 [cit. 2014-10-20]. Dostupné z: http://www.nsnam.org/wiki/Installation

[6] HOWTO configure Eclipse with ns-3 - Nsnam. *Nsnam* [online]. 2012 [cit. 2014-10-20]. Dostupné z: http://www.nsnam.org/wiki/Installation

[7] Install and configure Eclipse IDE on ns-3. *Computer Based Training for all* [online]. 2014 [cit. 2014-10-20]. Dostupné z: http://www.cbt4all.com/2014/04/install-and-configure-eclipse-ide-on-ns.html

[8] ns-3 Tutorial. *Ns-3* [online]. 2014 [cit. 2014-10-20]. Dostupné z: http://www.nsnam.org/docs/release/3.21/tutorial/html/index.html

[9] JEŘÁBEK, J. Pokročilé komunikační techniky. Skriptum FEKT Vysoké učení technické v Brně, 2015. s. 1-193.

[10] SATRAPA, P.: IPv6: Internet Protocol verze 6, třetí vydání. Neocortex, Praha 2011

[11] The TCP/IP Guide - Internet Protocol (IP/IPv4, IPng/IPv6) and IP-Related Protocols (IP NAT, IPSec, Mobile IP). *The TCP/IP Guide* [online]. 2012 [cit. 2014-10-30]. Dostupné z: http://www.tcpipguide.com/free/t_InternetProtocolIPIPv4IPngIPv6andIPRelatedProtocol.htm

[12] RFC: 791. *Internet Engineering Task Force (IETF)* [online]. 1981 [cit. 2014-10-27]. Dostupné z: https://www.ietf.org/rfc/rfc791.txt

[13] Ns-3 Model Library. *Ns-3* [online]. 2014 [cit. 2014-11-10]. Dostupné z: http://www.nsnam.org/docs/models/html/index.html

[14] MUFER, T. A.: Deploying IP multicast in the enterprise. Prentice-Hall, Upper Saddle River, 1992.

[15] KOTON, Jaroslav. *Moderní síťové technologie*. Brno, 2013, 191 s.

[16] ANTONOVA, Olga. *Introduction and Comparison of SCTP, TCP-MH, DCCP protocols* [online]. Sjökulla, 2004 [cit. 2015-05-01]. Dostupné z: http://www.tml.tkk.fi/Studies/T-110.551/2004/papers/Antonova.pdf

[17] Ns-3 Manual, Release ns-3.21. *Ns-3* [online]. 2014 [cit. 2015-2-28]. Dostupné z: https://www.nsnam.org/docs/release/3.21/manual/html/index.html

ZOZNAM POUŽITÝCH SKRATIEK

AC	Access Category
ACK	Acknowledgement / Acknowledgement number
AIFS	Arbitration Interframe Space
AP	Access Point
ARP	Address Resolution Protocol
AS	Autonomous System
ASM	Any-source multicast
ATM	Asynchronous Transfer Mode
BA	Behaviour Aggregate
BSS	Basic Service Set
CB WFQ	Class-Based Weighted Fair Queuing
CBS	Committed Burst Size
CIR	Committed Information Rate
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CTS	Clear to Send
CW	Contention Window
DCCP	Datagram Congestion Control Protocol
DCF	Distributed Coordination Function
DF	Dont't fragment
DHCP	Dynamic Host Configuration Protocol
DiffServ	Differentiated Services
DIFS	Distributed Coordination Function Interframe Space
DM	Dense Mode
DNS	Domain Name System
DSCP	Differentiated Services Code Point
DSSS	Direct Sequence Spread Spectrum
EBS	Excess Burst Size
ECN	Explicit Congestion Notification

EDCA	Enhanced Distributed Channel Access
EIGRP	Enhanced Interior Gateway Routing Protocol
ESS	Extended Service Set
FDDI	Fiber Distributed Data Interface
FHSS	Frequency Hopping Spread Spectrum
FIFO	First In First Out
FQ	Fair Queuing
HC	Hybrid Coordinator
IBSS	Independent Basic Service Set
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IntServ	Integrated Services
IP	Internet Protcol
ISO	International Organization for Standardization
LAN	Loacal Area Network
MAC	Media Access Control
MANET	Mobile Ad-Hoc Network
MF	More fragments
MF	Multi-Field Classification
MSDP	Multicast Source Discovery Protocol
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NetAnim	Network Animator
NS-3	Network Simulator 3
OFDM	Orthogonal Frequency Division Multiplexing
OLSR	Optimized Link State Routing
OS	Operating System
OSI	Open Systems Interconnection

OSPF	Open Shortest Path First
P2P	Point-to-Point
PBS	Peak Burst Size
PCF	Point Coordination Function
PIM	Protocol Independent Multicast
PIR	Peak Information Rate
PQ	Priority Queuing
QBSS	QoS Supporting BSS
QoS	Quality of Services
QSTA	QoS Station
RED	Random Early Detection
RFC	Request for Comments
RP	Rendezvous point
RTS	Ready to Send
RTT	Round-trip time
Rx	Receive
SCTP	Stream Control Transmission Protocol
SEQ	Sequence number
SIFS	Short Interframe Space
SM	Sparse Mode
SNAP	SubNetwork Access Protocol
SPT	Shortest-path tree
SSM	Specific-source multicast
TB	Token Bucket
TCP	Transmission Control Protocol
TID	Traffic ID
ToS	Type of Service
TTL	Time To Live
Tx	Transfer
TXOP	Transmission Opportunity
UDP	User Datagram Protocol

UID	Unique Identifier
VoIP	Voice over IP
WFQ	Weighted Fair Queuing
WiFi	Wireless Fidelity
WLAN	Wireless LAN
WRED	Weighted Random Early Detection
WRR	Weighted Round Robbin

OBSAH PRILOŽENÉHO CD

- Diplomová práca v elektronickej forme
- Začiatočné zdrojové súbory návodov:
  - `multicast.cc`
  - `qos.cc`
  - `Database.txt`