VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

# DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Eva Rybníčková

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS


# OPTIMALIZACE SÍŤOVÉHO PROVOZU POMOCÍ OMNET++
OPTIMIZATION OF NETWORK ACTIVITY BY OMNET++

DIPLOMOVÁ PRÁCE
MASTER'S THESIS


AUTOR PRÁCE
AUTHOR

Bc. Eva Rybníčková

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Vladislav Škorpil, CSc.


BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

***Studentka:*** Bc. Eva Rybníčková ***ID:*** 117346
***Ročník:*** 2 ***Akademický rok:*** 2015/16

**NÁZEV TÉMATU:**

## Optimalizace síťového provozu pomocí OMNeT++

**POKYNY PRO VYPRACOVÁNÍ:**

Prostudujte přenos dat v moderních sítích a simulační nástroj OMNeT++. Popište možnosti a hlavní funkce simulačního nástroje a to jak existující, tak vlastní. Optimalizujte návrh sítě, uvažujte kvalitu služby QoS. Uveďte výhody a nevýhody navržených alternativ. V rámci diplomové práce navrhněte tři počítačová cvičení, která budou využívat OMNET++.

**DOPORUČENÁ LITERATURA:**

[1] OMNET++, akademická licence, stránky http://www.omnetpp.org

[2] CHAO, H.J., GUO, X. Quality of Service Control in High-Speed Networks. John Wiley, New York 2002

[3] PUŽMANOVÁ, R. Moderní komunikační sítě A-Z. Computer Press, Brno 2007

***Termín zadání:*** 1.2.2016 ***Termín odevzdání:*** 25.5.2016

***Vedoucí práce:*** doc. Ing. Vladislav Škorpil, CSc.
***Konzultant diplomové práce:***

**doc. Ing. Jiří Mišurec, CSc.**, *předseda oborové rady*

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

## ABSTRAKT

Tato diplomová práce pojednává o programu OMNeT++ jakožto simulačnímu nástroji pro simulaci síťového provozu. Je zde popsána jeho instalace, základní funkce a jeho použití dohromady s nadstavbou INET framework. Dále je ukázáno sestavení jednoduché simulace, její části a analýza výstupu zpráv simulace. Je zde stručně vysvětlen pojem kvalita služby, který je dále v OMNeT++ prakticky implementován v názorných simulacích. Práce dále obsahuje dvě laboratorní cvičení a vzorové vypracované protokoly.

## KLÍČOVÁ SLOVA

OMNeT++, simulace, QoS, kvalita služby, INET framework

## ABSTRACT

This diploma thesis deals with OMNeT++, a simulation tool for network traffic simulations. It describes its installation, basic functions and its use together with an extension INET framework. Further it shows construction of a simple simulation, its parts and analysis of the traffic output. Briefly it explains Quality of Service, which is further practically implemented in visual simulations. The paper further contains two laboratory exercises with model protocols.

## KEYWORDS

OMNeT++, simulation, QoS, Quality of Service, INET framework

RYBNÍČKOVÁ, Eva *Optimalizace síťového provozu pomocí OMNeT++*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 69 s. Vedoucí práce byl doc. Ing. Vladislav Škorpil, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Optimalizace síťového provozu pomocí OMNeT++“ jsem vypracovala 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 diplomové práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení S 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno . . . . . . . . . . . . . . . .                                   . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

(podpis autora)

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu diplomové práce doc. Ing. Vladislavu Škorpilovi, CSc. a Ing. Václavu Oujezskému za metodickou pomoc při zpracovávaní diplomové práce.

Brno . . . . . .

. . . . . .

(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

Seznam obrázků ..... 10
Seznam tabulek ..... 11
1. Úvod ..... 12
2. OMNeT++ ..... 13
2.1. Instalace a příprava ..... 13
2.2. Hlavní části OMNeT++ ..... 14
2.3. Hlavní části OMNeT++ projektu ..... 18
2.4. Moduly ..... 19
2.5. Vytvoření prázdného projektu ..... 20
2.6. Použití INET modulů ..... 20
3. Ukázka simulace ..... 23
3.1 Nejčastěji používané INET podmoduly ..... 24
3.1.1 StandardHost ..... 24
3.1.2 Router ..... 25
3.1.3 EtherSwitch ..... 26
3.1.4 IPv4NetworkConfigurator ..... 27
4. Quality of Service ..... 31
4.1 QoS modely a výhody a nevýhody navržených alternativ ..... 31
4.1.1 Best effort ..... 31
4.1.2 Integrated Services ..... 31
4.1.3 Differentiated Services ..... 32
4.1.4 Porovnání všech tří QoS modelů ..... 33
5. Optimalizace návrhu sítě aplikací QoS v OMNeT++ ..... 34
5.1 Video vs. UDP zprávy ..... 34
5.2 Ping vs. TCP zprávy ..... 37
6. Laboratorní cvičení 1 – Sestavení topologie a spuštění jednoduché simulace ..... 41
6.1 Zadání ..... 41
6.2 Teoretický úvod ..... 41
6.3 Pracovní postup ..... 42
6.4 Vzorový protokol ..... 45
Zadání ..... 45
Vypracování ..... 46
Závěr ..... 46
7. Laboratorní cvičení 2 - Nastavení aplikací ..... 47
7.1 Zadání ..... 47

7.2 Teoretický úvod ..... 47
7.3 Pracovní postup ..... 48
7.4 Vzorový protokol ..... 50
Zadání ..... 50
Vypracování ..... 50
Závěr ..... 55
8. Návod pro vyučující ..... 56
8.1 Obecné informace ..... 56
8.1.1 Instalace OMNeT++ ..... 56
8.1.2 Integrace INET Framework ..... 56
8.1.3 Vytvoření projektu ..... 56
8.1.4 Sestrojení topologie ..... 56
8.1.5 Nastavení parametrů simulace ..... 56
8.1.6 Spuštění simulace ..... 56
8.2 Komentáře k laboratorním úlohám ..... 57
8.2.1 Laboratorní cvičení 1 – Sestavení topologie a spuštění jednoduché simulace ..... 57
8.2.2 Laboratorní cvičení 2 - Nastavení aplikací ..... 60
9. Závěr ..... 61
Seznam literatury ..... 62
Seznam zkratek ..... 63
Seznam příloh ..... 64

# Seznam obrázků

Obrázek 1 Instalace INET Framework ..... 13
Obrázek 2  Průvodce obsahem složky C:\omnetpp-5.0\samples ..... 14
Obrázek 3 Grafická část pracovního prostředí ..... 15
Obrázek 4 Paleta podmodulů a spojů ..... 16
Obrázek 5 Konfigurační část pracovního prostředí ..... 17
Obrázek 6 Spuštění simulace ..... 18
Obrázek 7 Složený modul a jeho instance ..... 19
Obrázek 8 Příklad vnitřní struktury složeného modulu ..... 19
Obrázek 9 Import INET do projektu ..... 21
Obrázek 10 Paleta INET podmodulů ..... 21
Obrázek 11 Grafická podoba topologie sítě TCP_example ..... 23
Obrázek 12 Topologie Video vs. UDP ..... 34
Obrázek 13 Traffic Conditioner ..... 35
Obrázek 14 Topologie Ping vs. TCP ..... 38
Obrázek 15 Topologie pro laboratorní úlohu 1 ..... 41
Obrázek 16 Topologie pro laboratorní úlohu 2 ..... 47

# Seznam tabulek

Tabulka 1 Parametry podmodulu StandardHost ..... 25
Tabulka 2 Parametry podmodulu StandardHost (pokračování) ..... 25
Tabulka 3 Parametry podmodulu Router ..... 26
Tabulka 4 Parametry podmodulu EtherSwitch ..... 26
Tabulka 5 Typy zpráv ve výstupu simulace ..... 29
Tabulka 6 Typy zpráv ve výstupu simulace (pokračování) ..... 29
Tabulka 7 Bity pole ToS ..... 32
Tabulka 8 Třídy assured forwarding ..... 33
Tabulka 9 Porovnání QoS modelů [3] ..... 33
Tabulka 10 Porovnání s QoS a bez QoS ..... 37
Tabulka 11 Ukázka simulace s různými hodnotami efMeter.cir ..... 39
Tabulka 12 Výstup zpráv simulace s aplikací PingApp – laboratorní úloha 1 ..... 46
Tabulka 13 Výstup simulace s ICMP errory ..... 51
Tabulka 14 Výstup simulace s ICMP errory ..... 52
Tabulka 15 Výstup simulace bez ICMP errorů ..... 53
Tabulka 16 Výstup simulace s VoIP službou ..... 54
Tabulka 17 Výstup zpráv simulace s aplikací PingApp – návod pro vyučující ..... 59
Tabulka 18 Zdrojový kód pro laboratorní úlohu - NED soubor ..... 65
Tabulka 19 Zdrojový kód pro laboratorní úlohu - INI soubor ..... 66
Tabulka 20 Výstup zpráv simulace projektu TCP_example ..... 67

# 1. Úvod

Pod pojmem simulace rozumíme něco na způsob virtuálního sestavení nebo realizace takové technologie, kterou nám není umožněno doopravdy uskutečnit nebo ji potřebujeme například otestovat, než ji uvedeme do provozu. Možnost simulovat nějakou technologii přijde vhod jak v komerčním, tak akademickém nebo domácím prostředí. Příkladem může být například simulace prostředí mimo naši planetu za účelem výzkumu vesmíru nebo simulace pilotování letadla jakožto počítačové hry. Pro testování síťových modelů nebo pro seznámení se s fungováním různých protokolů nebo nastavením sítě je velmi vhodnou a efektivní pomůckou právě aplikace OMNeT++. OMNeT++ umožňuje sestavení a simulaci sítí od jednoduchých LAN až po rozsáhlé WAN sítě.

OMNeT++ je zkratkou pro *Objective Modular Network Testbed in C++* a je to prostředek založený na platformě *Eclipse*, který slouží k simulování síťových modelů využíváním komponent navržených v jazyce C++. OMNeT++ byl vynalezen zakladatelem společnosti OpenSim Ltd. Andrásem Vargou. Je jakožto akademická licence zdarma, jeho komerční verzí je potom tzv. OMNEST vlastněný společností Omnest Global, Inc. OMNeT++ umožňuje simulaci všech možných síťových prvků či protokolů a je stále rozvíjen a obohacován o nové plug-iny, funkce a simulační moduly. [1]

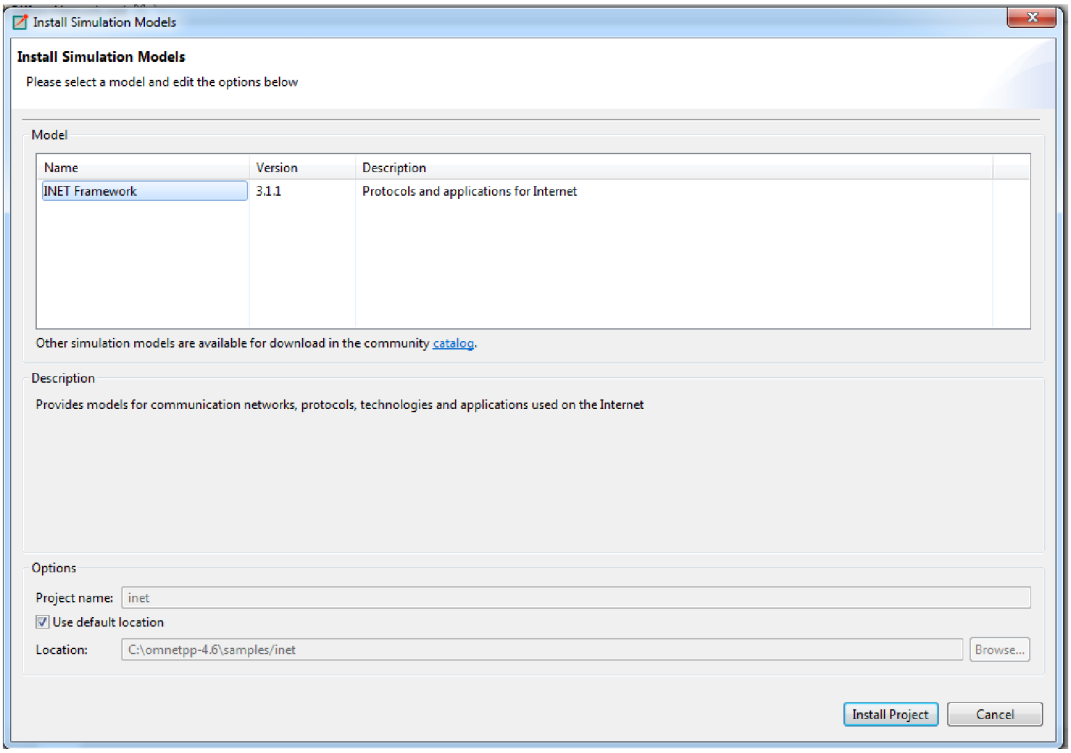
# 2. OMNeT++

## 2.1. Instalace a příprava

OMNeT++ není k dispozici jako spustitelný program, ale jako zdrojový kód, který je potřeba nejdříve zkompilovat. Neinstaluje se klasicky jako většina programů instalačním průvodcem, nýbrž přes příkazový řádek, kde si ale v případě Windows vystačíme se dvěma jednoduchými příkazy.

OMNeT++ najdeme ke stažení na oficiálních stránkách https://omnetpp.org/omnetpp, kde se nám nabízí nejaktuálnější verze pro Linux i pro Windows, a tou je OMNeT++ 5.0. Pro tuto práci bude použit OMNeT++ pouze pro Windows, takže se instalací Linuxové verze nebudeme zabývat.

Postup instalace je následovný: nejdříve stáhneme a rozbalíme soubor *omnetpp-5.0-src-windows.zip,* a to nejlépe do kořenového adresáře, respektive do takové složky, která nemá v názvu diakritiku a mezery. Potom spustíme soubor *mingwenv.cmd* a zobrazí se příkazový řádek. Vepíšeme do něj příkaz `./configure` a po proběhnutí operace příkaz `make`. Po odhadem 20 minutách můžeme spustit OMNeT++ příkazem `omnetpp`. Zobrazí se pracovní plocha *(workspace)*, kde vlevo vidíme již předinstalované simulační modely, jejichž moduly můžeme použít pro své vlastní projekty. Nejdůležitější, respektive nejobsáhlejší sbírkou simulačních modelů pro drátové i bezdrátové síťové protokoly je aplikace INET, která není původně součástí a instaluje se zvlášť. Nalezneme ji taktéž na oficiálních stránkách, nicméně ji můžeme nainstalovat i přes OMNeT++ samotný, a to následujícím způsobem. V záložce *Help* zvolíme možnost *Install Simulation Models* a zobrazí se nám možnost nainstalovat „INET Framework“. Dáme jej instalovat a INET se sám stáhne a zkompiluje. Vlevo mezi ostatními simulačními modely se zobrazí nová složka „inet“. INET zpravidla obsahuje mnoho proměnných, které jsou nastavené, ale nejsou použité, proto vidíme na jeho ikoně žlutou výstražnou značku. Tu můžeme ignorovat, občas se ale stane, že se objeví značka červená, která již může bránit některým funkcím v průběhu. Zde pomůže celou aplikaci restartovat nebo INET přeinstalovat. Použití INETu bude nicméně vysvětleno dále. [1]

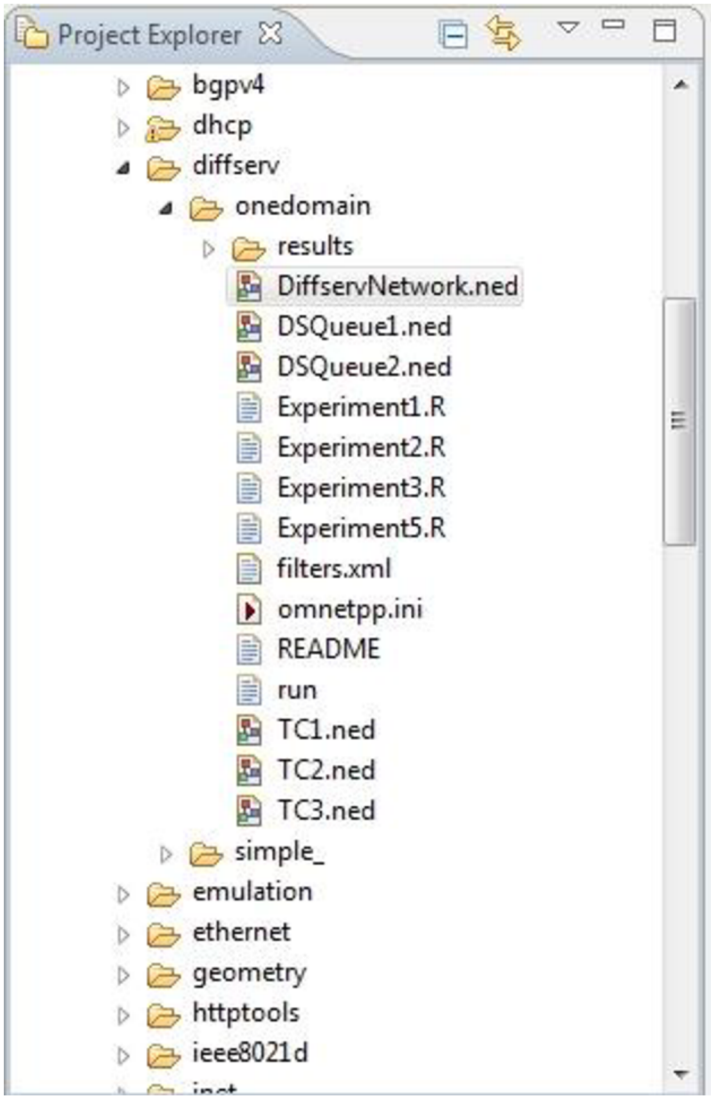



*Obrázek 1 Instalace INET Framework*

## 2.2. Hlavní části OMNeT++

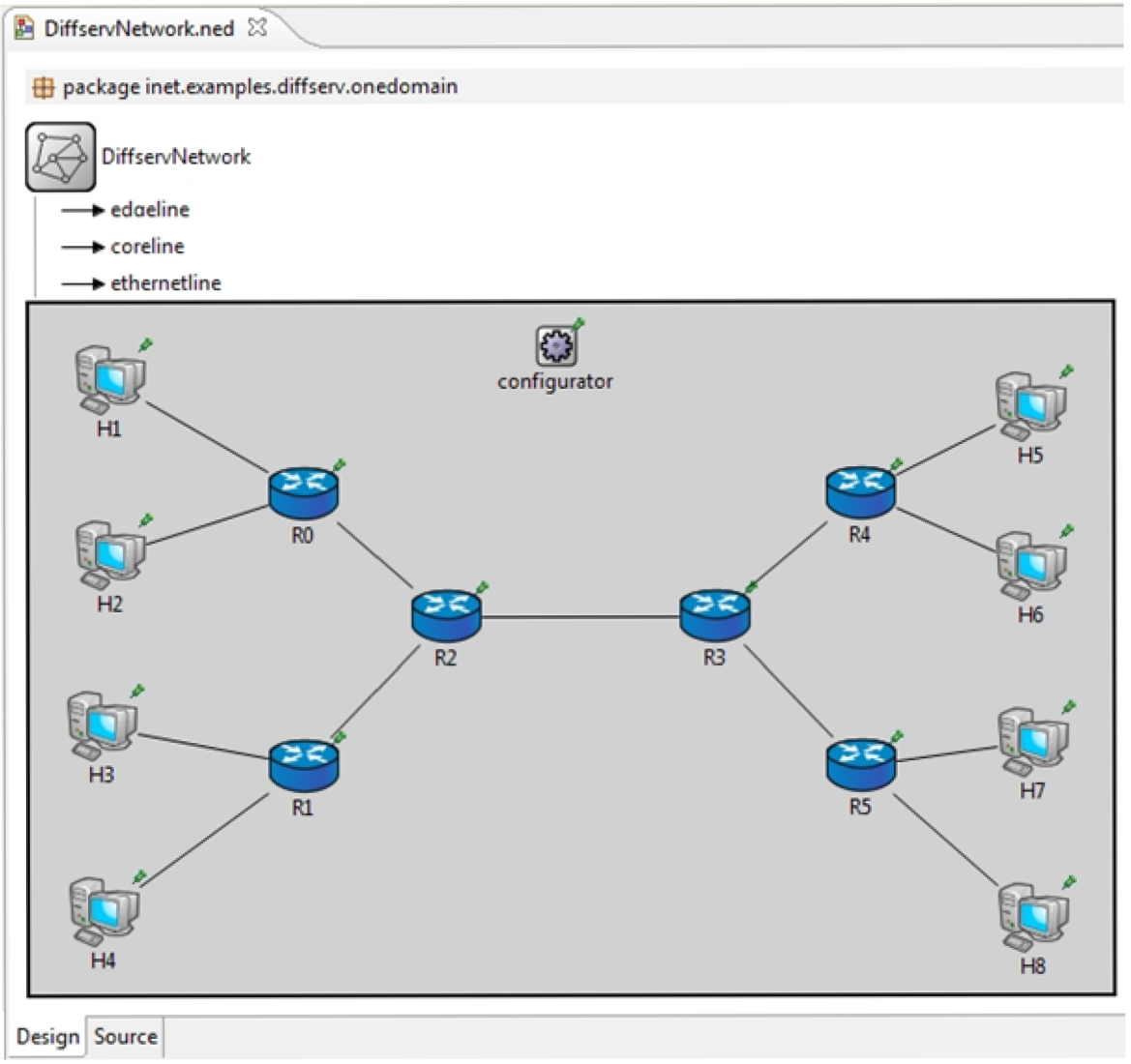
Správně nainstalovanou aplikaci tvoří tyto části:

1. **Průvodce obsahem složky C:\omnetpp-5.0\samples**. Zde nalezneme jak již hotové simulační modely, tak si zde ukládáme své vlastní projekty



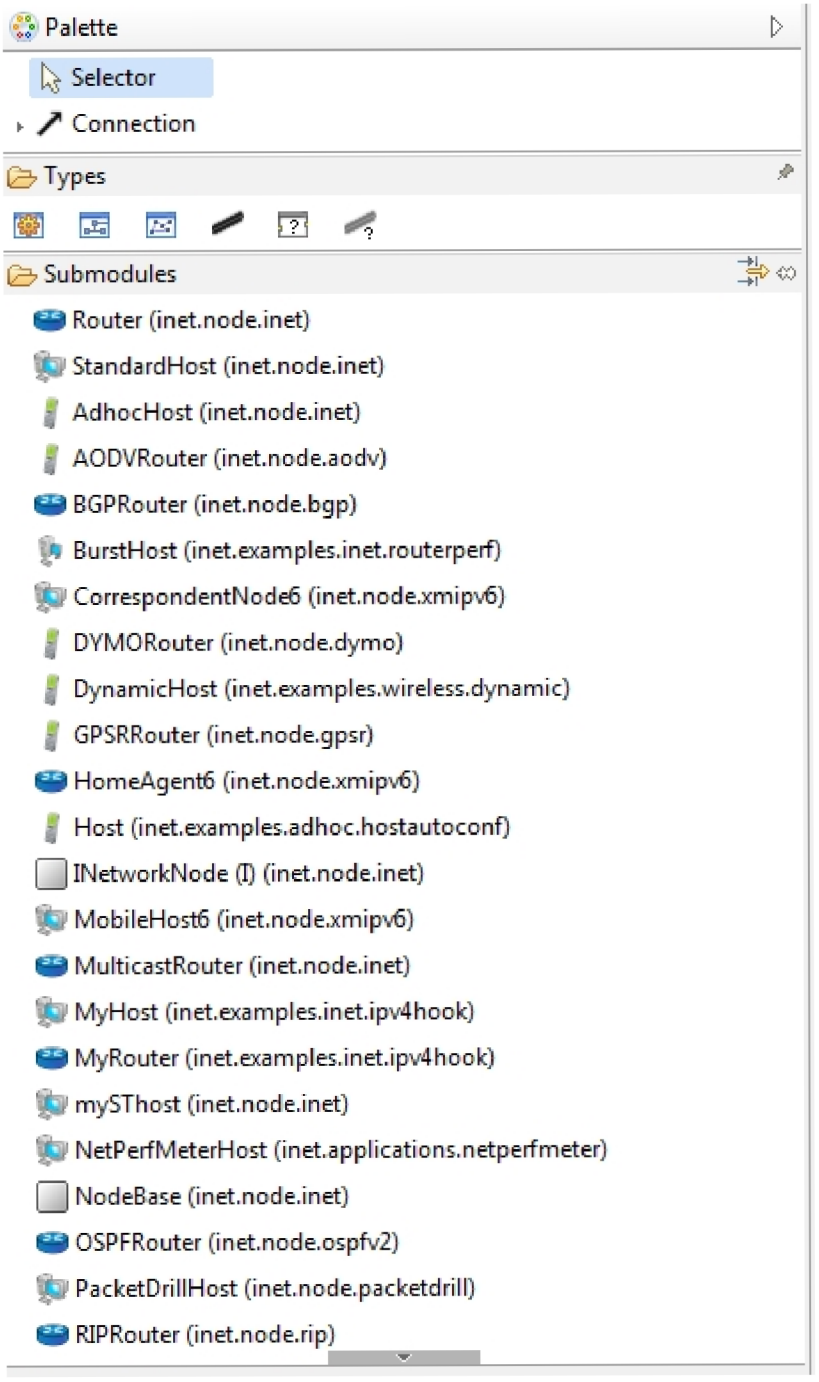

*Obrázek 2 Průvodce obsahem složky C:\omnetpp-5.0\samples*

## 2. Grafická část pracovního prostředí




*Obrázek 3 Grafická část pracovního prostředí*

3. **Paleta podmodulů a spojů** – získáme ji importováním simulačních modelů, např. INETu.




*Obrázek 4 Paleta podmodulů a spojů*

## 4. Konfigurační část pracovního prostředí

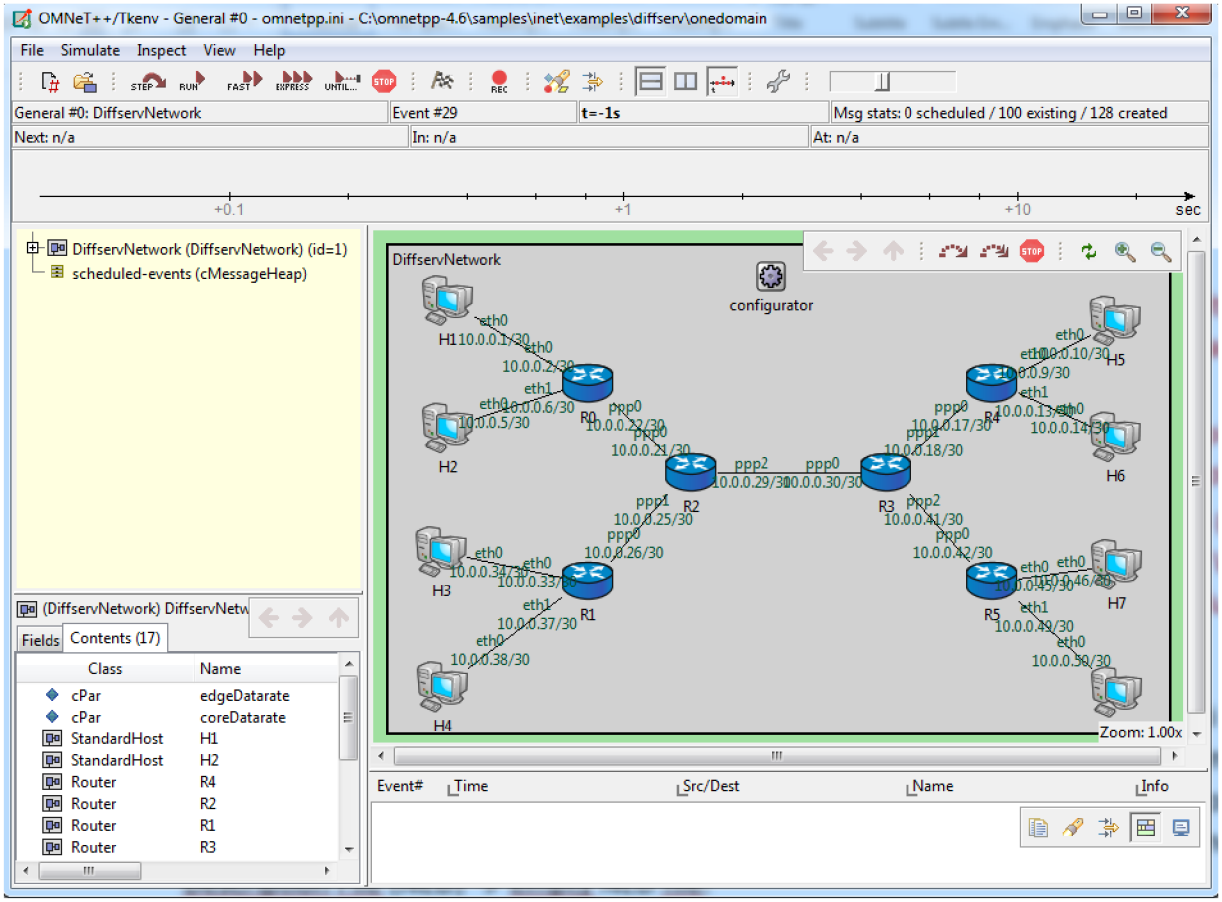
DiffservNetwork.ned

```
package inet.examples.diffserv.onedomain;

import inet.common.misc.ThruputMeteringChannel;
import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.inet.Router;
import inet.node.inet.StandardHost;


network DiffservNetwork
{
    parameters:
        double edgeDatarate @unit(bps);
        double coreDatarate @unit(bps);
    types:
        channel edgeline extends ThruputMeteringChannel
        {
            delay = 2ms;
            datarate = edgeDatarate;
            thruputDisplayFormat = "b B U";
        }
        channel coreline extends ThruputMeteringChannel
        {
            delay = 2ms;
            datarate = coreDatarate;
            thruputDisplayFormat = "b B U";
        }
        channel ethernetline extends ThruputMeteringChannel
        {
            delay = 0.1us;
            datarate = 100Mbps;
            thruputDisplayFormat = "b B U";
        }
    submodules:
        H1: StandardHost {
            @display("p=46,43");
        }
        H2: StandardHost {
            @display("p=46,143");
        }
        R4: Router {
            @display("p=473,108");
        }
        R2: Router {
```

Design | Source

*Obrázek 5 Konfigurační část pracovního prostředí*

5. **Simulace připravená ke spuštění** vypadá například takto:

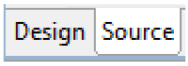



*Obrázek 6 Spuštění simulace*

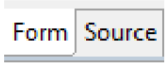
## 2.3. Hlavní části OMNeT++ projektu

Aby simulace fungovala tak, jak má, jsou zapotřebí tyto prvky:

- ***NED* soubor** – soubor s příponou *.ned*, ve kterém definujeme strukturu neboli topologii sítě. Můžeme ji vytvořit ručně pomocí nabídky podmodulů a spojů v pravé části záložky *Design*, anebo pomocí příkazů (čili psaním kódu) v záložce *Source*.




- **Konfigurační soubor** – soubor pojmenovaný zpravidla jako *omnetpp.ini*. V tomto souboru jsou nakonfigurována všechna nastavení simulace i nastavení vlastností modulů, spojů, atd. Lze jej používat taktéž dvěma způsoby, a to buď v GUI v záložce *Form*, nebo pomocí příkazů v záložce *Source*.

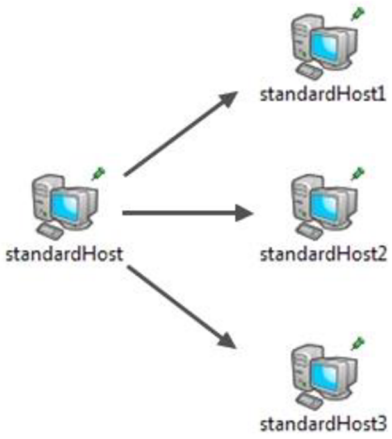



- **Zpráva (*message*)** – soubor s příponou *.msg*. Můžeme definovat více typů zpráv a přidávat do nich různá datová pole. OMNeT++ potom překládá definice zpráv do plnohodnotných C++ tříd.
- **Zdroje jednoduchých modulů (*simple modules sources*)** – C++ soubory s příponou *.h* nebo *.cc*. Tyto soubory se tvoří samy, pokud pracujeme pouze s již hotovými moduly. Pokud tvoříme své vlastní moduly, můžeme si psát C++ zdroje sami nebo stávající různě modifikovat.

V této práci budou použity pouze předpřipravené INET moduly, takže budeme pracovat aktivně pouze s *.ned* souborem a *.ini* souborem. [1]

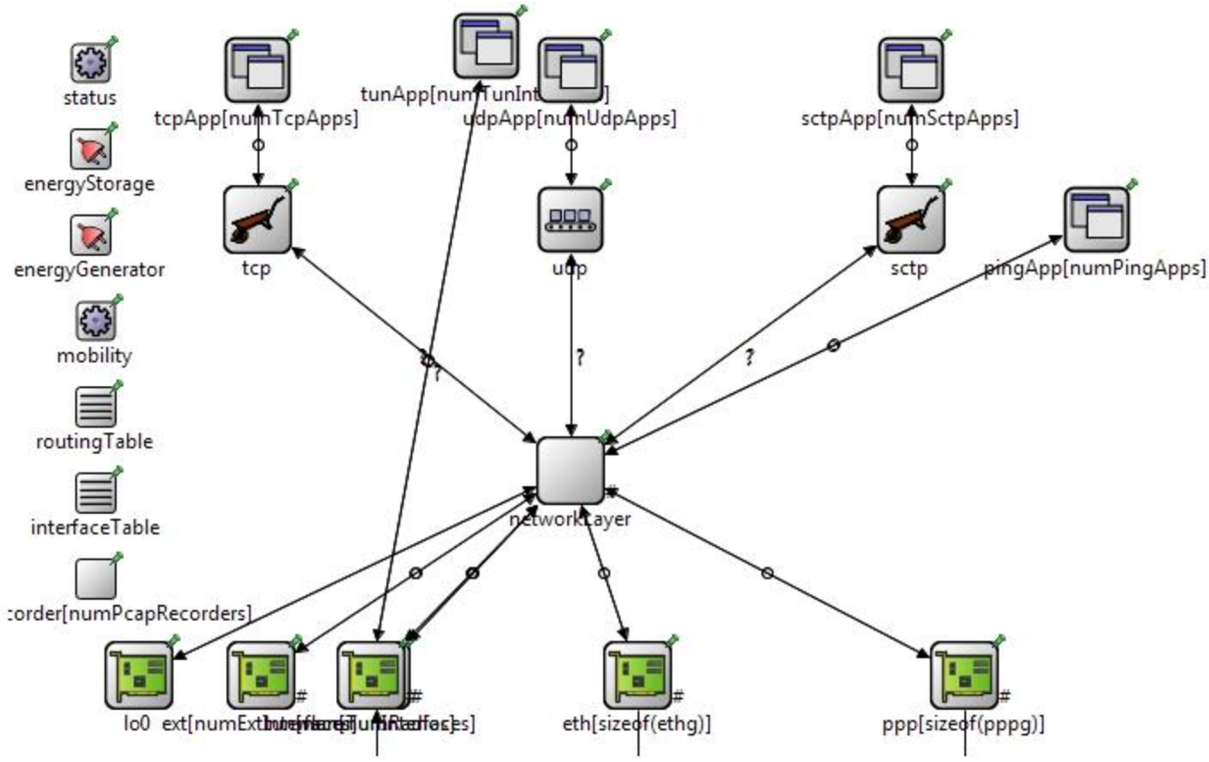
## 2.4. Moduly

Práce v OMNeT++ spočívá ve tvoření simulačním modelů za použití a aplikace modulových typů, jejichž instancemi jsou hierarchicky uspořádané moduly. Nejnižší v hierarchii se nazývají jednoduché moduly *(simple modules)*, naopak nejvyšší je systémový modul. Podmoduly *(submodules)* takto tvoří tzv. složený modul *(compound module)*. Zde je pro lepší pochopení názorná ukázka:




*Obrázek 7 Složený modul a jeho instance*

Složeným modulem je zde modul *StandardHost*, ze kterého můžeme vytvořit libovolné množství instancí, které můžeme pojmenovat očíslováním původního modulu (*StandardHost1*, *StandardHost2* …) nebo libovolně podle potřeby (*PC1*, *WebServer* atd.). Po rozkliknutí ikony *StandardHost* potom vidíme, z jakých podmodulů je tento modul tvořen:

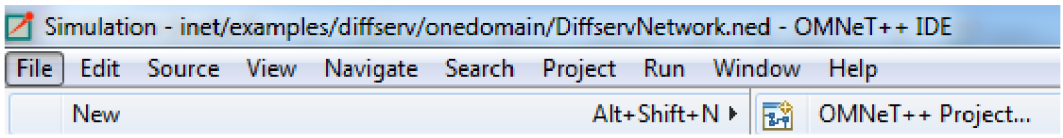



*Obrázek 8 Příklad vnitřní struktury složeného modulu*

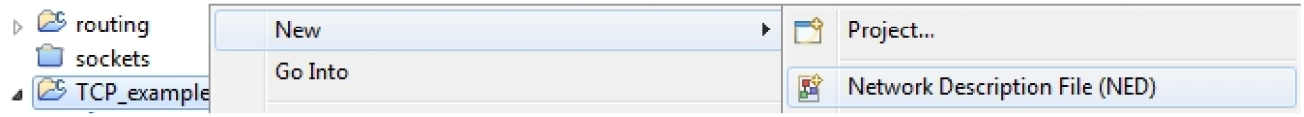
Moduly spolu komunikují pomocí zpráv *(messages)*, jako například rámce nebo pakety. Zprávy se mezi moduly přenášejí prostřednictvím bran (*gates*), které jsou propojené pomocí spojů *(connections)* o různých parametrech. Běžně se vlastnosti spojů mohou modifikovat pomocí zpoždění *(propagation delay)*, chybovosti *(bit error rate)* a rychlosti *(data rate)*. [1]

## 2.5. Vytvoření prázdného projektu

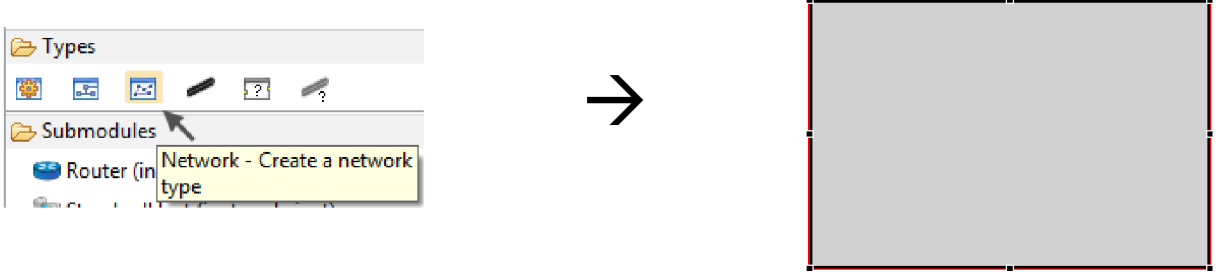
Než si představíme konkrétní simulaci, začneme úplně od začátku vytvořením tzv. *Projektu*. V záložce *File* tlačítkem *New* vytvoříme nový *„OMNeT++ Project“*, který nazveme libovolně, v tomto ukázkovém případě *TCP_example*.



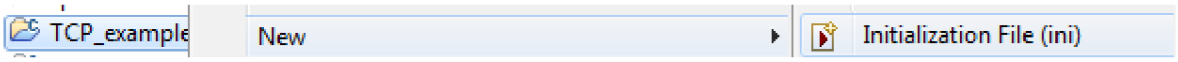

V poli *Project Explorer* se objeví nová složka. Nyní vytvoříme grafické prostředí pro sestavení simulace – *NED* soubor. Pravým kliknutím na složku *TCP_example* přes možnost *New* vybereme *„Network Description File“*.




Jak již bylo řečeno, s *NED* souborem lze pracovat dvěma způsoby – pomocí ikon podmodulů nebo pomocí kódu. My zvolíme prozatím grafickou možnost. V záložce *Design* vytvoříme prázdnou síť a v ní poté budeme moci sestavit danou topologii.



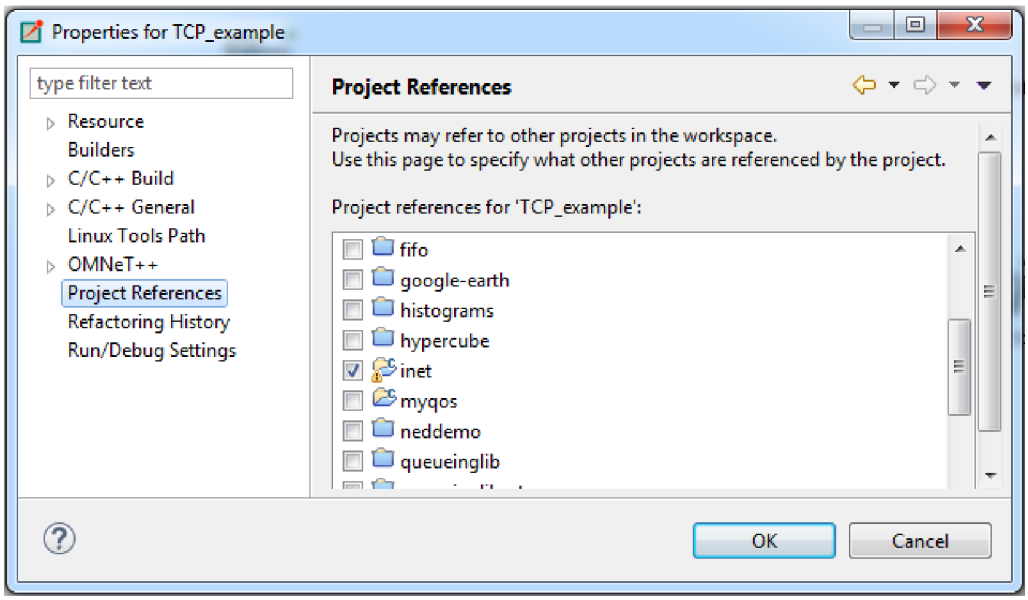

Další, co budeme potřebovat je *.ini* soubor, který vytvoříme podobně:



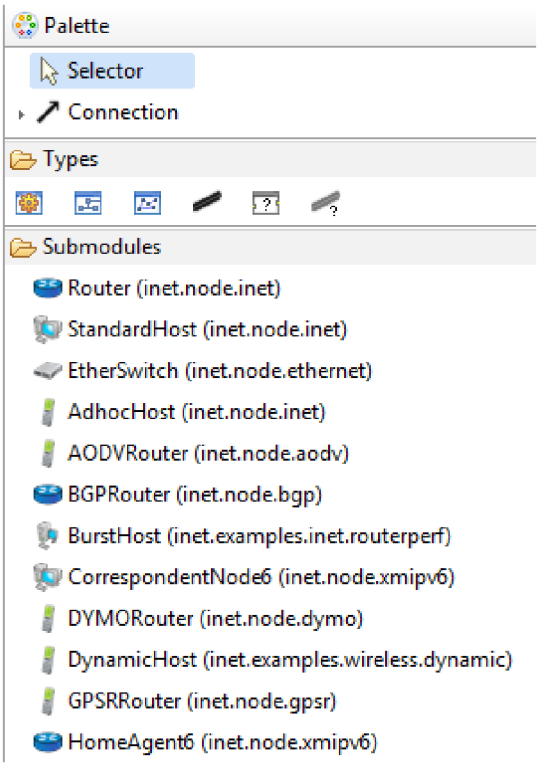

Konkrétní funkce *NED* a *INI* prostředí budou popsány později.

## 2.6. Použití INET modulů

Než začneme tvořit topologii, je nutné importovat do projektu INET podmoduly. Jak již bylo zmíněno, je možné si psát každý modul zvlášť, ale to je v některých případech zbytečně komplikované a sada INET nám dává k dispozici celou řadu materiálu, se kterým lze pracovat i při tvorbě složitějších sítí. Jako názorná ukázka nám poslouží dva klienti spojení ethernetovým kabelem. INET moduly do své sítě importujeme přes *Properties* pravým kliknutím na náš projekt a v *Project References* zaškrtneme „inet“. Následně se vpravo objeví nabídka modulů, kde vybereme například podmodul St*andardHost*, který v praxi běžně reprezentuje například stolní PC, notebook, server nebo třeba IP telefon.

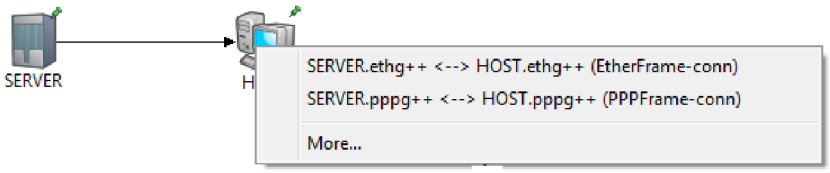



*Obrázek 9 Import INET do projektu*




*Obrázek 10 Paleta INET podmodulů*

*StandardHost* vložíme dvakrát do předem vytvořené sítě (kliknutím na ikonu a poté kliknutím do sítě) a spojíme je spojem, který si vybereme z nabídky *connection*. Modul si můžeme přejmenovat nebo mu změnit ikonu v nabídce *Properties* pravým kliknutím na obrázek.



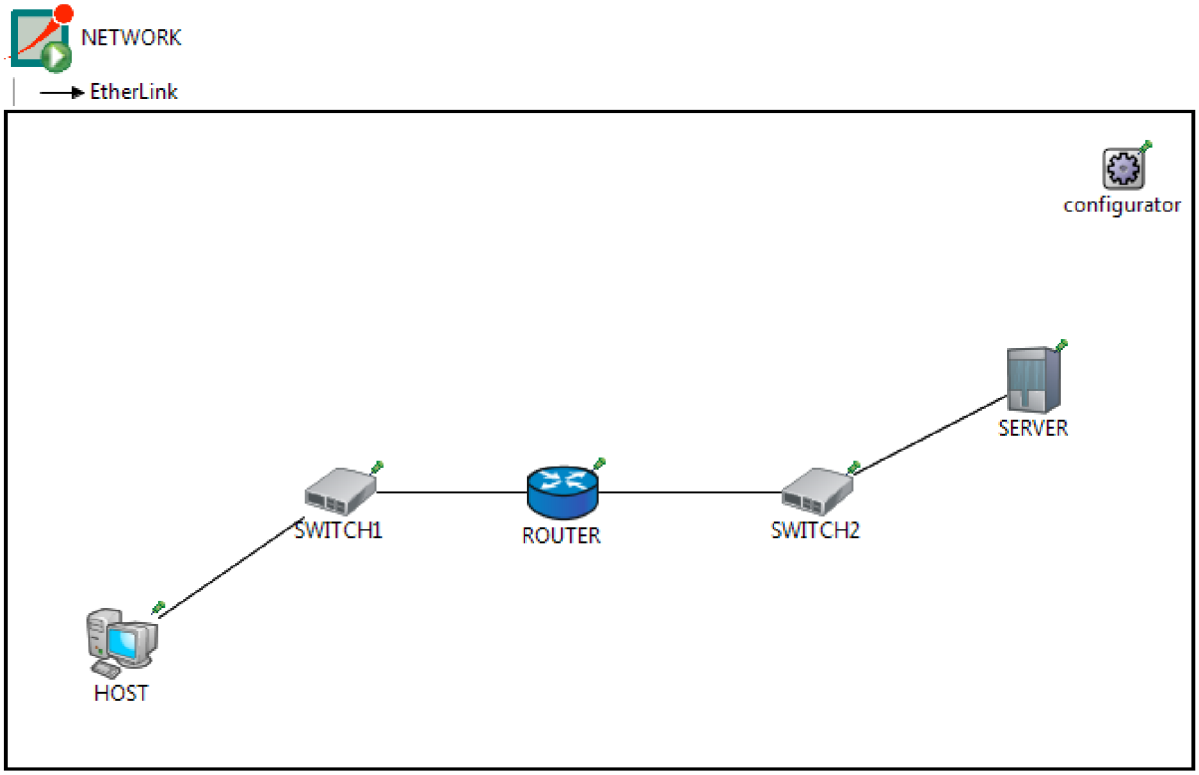
Když se potom podíváme do záložky *Source*, uvidíme, že se vytvořil tento kód:

```
package sp.simulations;

import inet.node.ethernet.Eth10G; // Spoj Eth10G byl importován.
Import inet.node.inet.StandardHost; // Submodul StandardHost byl
importován.

@license(LGPL); // Lesser General Public License - znamená, že kód je možné
kopírovat a distribuovat, ale nesmí se měnit. Tento řádek může být klidně
vynechán.

network Network // Jméno sítě "Network" bylo vytvořeno automaticky.
{
    @display("bgb=1036,474,white"); // Rozměry a barva prostředí sítě v
simulaci.
    submodules:
        SERVER: StandardHost {
            @display("p=347,219;i=device/server"); // SERVER je instancí
modulu StandardHost. Je nastavena jeho pozice a obrázek ikony.
        }
        HOST: StandardHost {
            @display("p=531,219"); // HOST je další instancí modulu
StandardHost. Je nastavena jeho pozice a obrázek ikony je stejný jako
výchozí, proto není specifikován.

                    }
    connections:
        SERVER.ethg++ <--> Eth10G <--> HOST.ethg++; // Propojení serveru a
hosta.
}
```

Z ukázky vidíme, že nastavení pomocí ikon v GUI je podstatně jednodušší. Takto vytvořená topologie nicméně nemůže v praxi fungovat, protože je potřeba ji nejdříve nakonfigurovat a to bude ukázáno dále.

## 3. Ukázka simulace

Jako názornou ukázku běhu provozu v simulátoru bude vysvětlena jednoduchá simulace sítě s TCP přenosem, která bude tvořena jedním hostem a jedním serverem (dvakrát *StandardHost*), jedním směrovačem (*Router*) a dvěma přepínači (dvakrát *EtherSwitch*). Jednotlivé prvky propojíme ethernetovým kabelem (*EtherLink*). Nakonec přidáme IPv4 konfigurátor (*IPv4NetworkConfigurator*). Grafická podoba bude vypadat tedy zhruba takto:




*Obrázek 11 Grafická podoba topologie sítě TCP_example*

A výstup kódu bude vypadat takto:

```
package tcp_example.simulations;

import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.ethernet.EtherSwitch;
import inet.node.inet.Router;
import inet.node.inet.StandardHost;
import inet.node.ethernet.EtherLink;
import ned.DatarateChannel;

network NETWORK

{
    @display("bgb=661,365,white;i=logo/logo128m;is=vl");
    types:
        channel EtherLink extends DatarateChannel
        {
            datarate = 100Mbps; // rychlost linky
            delay = 0.1us; // zpoždění
        }
    submodules:
        SWITCH1: EtherSwitch {
            parameters:
```

```
            @display("p=186,211");
        }
        ROUTER: Router {
            parameters:
                @display("p=310,212");
        }
        SWITCH2: EtherSwitch {
            parameters:
                @display("p=452,211");
        }
        SERVER: StandardHost {
            parameters:
                @display("p=573,149;i=device/server");
                forwarding = true;
        }
        HOST: StandardHost {
            parameters:
                @display("p=64,296");
                forwarding = true;
        }
        configurator:
inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator {
            parameters:
                @display("p=623,31");

        }
    connections:
        SWITCH1.ethg++ <--> EtherLink <--> ROUTER.ethg++;
        ROUTER.ethg++ <--> EtherLink <--> SWITCH1.ethg++;
        ROUTER.ethg++ <--> EtherLink <--> SWITCH2.ethg++;
        SWITCH2.ethg++ <--> EtherLink <--> ROUTER.ethg++;
        SWITCH2.ethg++ <--> EtherLink <--> SERVER.ethg++;
        SERVER.ethg++ <--> EtherLink <--> SWITCH2.ethg++;
        SWITCH1.ethg++ <--> EtherLink <--> HOST.ethg++;
        HOST.ethg++ <--> EtherLink <--> SWITCH1.ethg++;
}
```

Nabídka podmodulů nebo aplikací je poměrně rozsáhlá, proto je vhodné čerpat informace z INET dokumentace, kterou nalezneme zde https://omnetpp.org/doc/inet/api-current/neddoc/index.html . Je zde popsána každá položka INETu včetně seznamu parametrů, které je nutné jí nastavit. [2]

## 3.1 Nejčastěji používané INET podmoduly

### 3.1.1 StandardHost

StandardHost je IPv4 host s SCTP, TCP, UDP vrstvami a aplikacemi. Může být přes ethernetové rozhraní připojen k ostatním uzlům přes ethernetovou bránu (*ethg gate*). Je zde podporován pouze fullduplex (přes křížený kabel). Změna na halfduplex se provede nastavením `**.eth[*].typename ="EthernetInterface"` pro full/halfduplex CSMA/CD implementaci (přes koaxiální kabel). StandardHost původně nemá kartu pro bezdrátové připojení, lze ji ale přidat parametrem *numRadios*. Typ karty pro bezdrátové připojení se konfiguruje nastavením `**.wlan[*].typename parameter`.

StandardHost má všechny parametry defaultně nastavené na nějakou hodnotu, jediné, co musíme udělat je nastavit počet aplikací, které bude používat a specifikovat jejich názvy.

Na StandardHost můžeme aplikovat tyto parametry [2]:

*Tabulka 1 Parametry podmodulu StandardHost*

| Name | Type | Default value |
|---|---|---|
| hasStatus | bool | false |
| numExtInterfaces | int | 0 |
| numRadios | int | 0 |
| numPcapRecorders | int | 0 |
| mobilityType | string | numRadios > 0 ? "StationaryMobility" : "" |
| routingFile | string | "" |
| IPForward | bool | false |
| forwardMulticast | bool | false |
| batteryType | string | "" |
| numTcpApps | int | 0 |
| numUdpApps | int | 0 |
| numSctpApps | int | 0 |
| numPingApps | int | 0 |
| hasTcp | bool | numTcpApps>0 |
| hasUdp | bool | numUdpApps>0 |
| hasSctp | bool | numSctpApps>0 |
| tcpType | string | firstAvailable("TCP", "TCP_lwIP", "TCP_NSC", "TCP_None") |
| udpType | string | firstAvailable("UDP","UDP_None") |
| sctpType | string | firstAvailable("SCTP","SCTP_None") |

*Tabulka 2 Parametry podmodulu StandardHost (pokračování)*

| Name | description |
|---|---|
| numRadios | the number of radios in the router. by default no wireless |
| numPcapRecorders | no of PcapRecorders. |
| numTcpApps | no of TCP apps. Specify the app types in INI file with tcpApp[0..1].typename="TCPEchoApp" syntax |
| numUdpApps | no of UDP apps. Specify the app types in INI file with udpApp[0..1].typename="UDPVideoStreamCli" syntax |
| numSctpApps | no of SCTP apps. Specify the app types in INI file with sctpApp[0..1].typename="SCTPServer" syntax |
| numPingApps | no of PING apps. Specify the app types in INI file with pingApp[0..1].typename="PingApp" syntax |
| tcpType | tcp implementation (e.g. TCP, TCP_lwIP, TCP_NSC) or TCPSpoof |

### 3.1.2 Router

IPv4 směrovač podporuje bezdrátové, Ethernet, PPP a externí rozhraní. Může být připojen přes ethernetové rozhraní k ostatním uzlům přes ethernet bránu (*ethg gate*). Je zde podporován pouze full duplex (přes křížený kabel). Změna na halfduplex se provede nastavením `**.eth.typename ="EthernetInterface"` pro full/halfduplex CSMA/CD implementaci (přes koaxiální kabel). *Router* původně nemá kartu pro bezdrátové připojení, lze ji ale přidat parametrem numRadios. Typ karty pro bezdrátové připojení se konfiguruje nastavením `**.wlan.typename parameter`.

Dynamické směrování není defaultně nastavené. Podmoduly *Routeru* mohou být navíc *BGPRouter*, *OSPFRouter* nebo *RIPRouter*, které podporují tyto dané směrovací protokoly. Defaultně je modul *Router* nepodporuje, ale mohou se přidat parametrem *hasOSPF*, *hasRIP* nebo *hasBGP*.

Na *Router* můžeme aplikovat tyto parametry [2]:

*Tabulka 3 Parametry podmodulu Router*

| Name | Type | Default value | Description |
|---|---|---|---|
| hasStatus | bool | false | |
| numExtInterfaces | int | 0 | |
| numRadios | int | 0 | the number of radios in the router. by default no wireless |
| numPcapRecorders | int | 0 | no of PcapRecorders. |
| mobilityType | string | numRadios > 0 ? "StationaryMobility" : "" | |
| routingFile | string | “” | |
| IPForward | bool | true | |
| forwardMulticast | bool | false | |
| batteryType | string | “” | |
| hasOSPF | bool | false | |
| hasRIP | bool | false | |
| hasBGP | bool | false | |
| tcpType | string | firstAvailable("TCP", "TCP_lwIP", "TCP_NSC", "TCP_None") | tcp implementation (e.g. TCP, TCP_lwIP, TCP_NSC) or TCPSpoof |
| udpType | string | UDP | |

### 3.1.3 EtherSwitch

*EtherSwitch* plní funkci klasického přepínače a můžeme na něj aplikovat tyto parametry [2]:

*Tabulka 4 Parametry podmodulu EtherSwitch*

| Name | Type | Default value | Description |
|---|---|---|---|
| hasStatus | bool | false | |
| csmacdSupport | bool | true | by default use CSMA/CD |
| macType | string | csmacdSupport ? "EtherMAC" : "EtherMACFullDuplex" | EtherMAC or EtherMACFullDuplex |
| spanningTreeProtocol | string | | |
| relayUnitType | string | firstAvailable("Ieee8021dRelay","MACRelayUnit") | type of the IMACRelayUnit; |
| macTableType | string | MACAddressTable | type of the IMACAddressTable |

### 3.1.4 IPv4NetworkConfigurator

Konfigurátor IPv4 sítě. Tento modul přiřazuje IP adresy a zavádí statické směrování pro IPv4 síť. Děje se tak automaticky, takže není potřeba nic nastavovat nebo upravovat, pokud si ale například přejeme použít jiný rozsah IP adres než výchozí (10.x.x.x 255.x.x.x), můžeme tak učinit přidáním tohoto příkazu do INI souboru a přizpůsobit si jeho obsah.

```
**.configurator.config = xml("<config> … … … </config>")
```

Nejjednodušší a zároveň výchozí nastavení potom vypadá v principu takto:

```
<config>
  <interface hosts='**' names='**' address='10.x.x.x' netmask='255.x.x.x'/>
</config>
```

kde

- `interface` značí, že bude nastaveno konkrétní (jedno nebo více) rozhraní
- `hosts` definuje zařízení v síti, v našem případě ROUTER, SERVER nebo HOST (přepínač pochopitelně IP adresu nemá, pokud nebereme v potaz virtuální rozhraní, což zde není relevantní)
- `names` definuje konkrétní rozhraní, například *eth0*, *ppp0*, *ppp1* atd.
- `address` zároveň s `netmask` definuje rozsah IP adres

Kromě parametru `<interface>`, lze dále například přizpůsobit nastavení parametrů `<wireless>`, `<route>` nebo `<autoroute>`. Veškeré podrobnosti jsou potom uvedeny v INET dokumentaci. Názorný příklad manuální konfigurace IP adres zle nalézt zde v kapitole 5.1

Tímto byla sestavena grafická podoba sítě, ale aby se v ní mohly přenášet zprávy, je potřeba každému hostu přiřadit aplikaci, která určí, jak se bude v síti chovat a pomocí které se nám otevřou adekvátní parametry pro nastavení sítě. Tyto aplikace nalezneme v adresáři *inet/src/inet/Applications* a nastavujeme je příkazem `tcpApp[*].typename`, `udpApp[*].typename`, `pingApp[*].typename` a podobně. V ukázkové síti bude použita aplikace *TCPBasicClientApp* pro počítač a *TCPEchoApp* pro server. Vytvoříme tedy soubor *.ini* a první, co je potřeba udělat je specifikovat síť, která se bude konfigurovat. V tomto případě jednoduše příkazem `network = NETWORK`. Dále je třeba příkazem `numTcpApps` určit počet aplikací, které hosté a server budou používat. Pokud tak neučiníme, resp. nastavíme počet aplikací na 0, budou jednoduše ignorovány. Naopak, pokud zadáme větší počet aplikací, než jsme nastavili, obdržíme při spouštění simulace chybovou hlášku a bude vyžadováno aplikaci doplnit.

*TCPBasicClientApp* aplikace umožňuje „žádost-odpověď“ komunikaci přes TCP protokol. Klient komunikuje se serverem v relacích. Během jedné relace klient zahájí TCP spojení se serverem, pošle mu žádost o spojení, počká na kompletní odpověď od serveru a poté spojení ukončí. Hlavní parametry, které musíme aplikaci nastavit, jsou lokální a cílový port, kdy cílový port je defaultně nastaven na hodnotu 1000 a lokální port je možné nastavit libovolně, případně jej nechat prázdný. Podobně se nastaví lokální a cílová adresa. Lokální adresa bude buď jméno klienta v uvozovkách, nebo zůstane prázdná („“) a cílová adresa bude jméno serveru v uvozovkách.

Na server byla aplikována aplikace *TCPEchoApp*. Aplikace přijme TCP spojení a příchozí zprávu odešle nazpět klientovi. Délka zprávy je násobena tzv. echo-faktorem (*echoFactor*), takže pokud je tento nastaven defaultně, čili na hodnotu 1, vrací se zpráva nezměněna. Dále je možné specifikovat, jaké zprávy se budou sítí přenášet, a to parametrem *dataTransferMode,* který nabízí 3 možnosti:

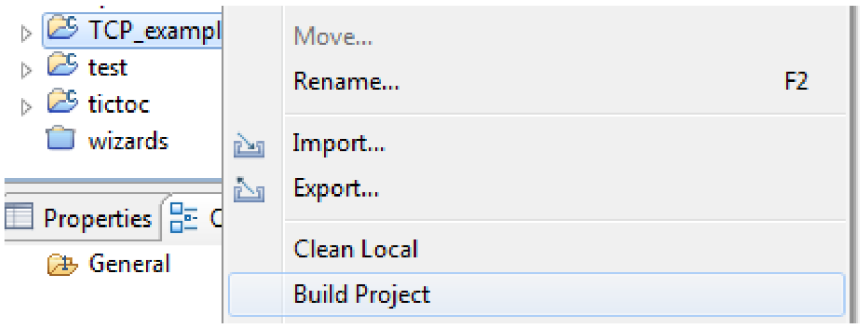
- **bytecount** – použity jsou „virtuální bajty“, nikoliv opravdová data
- **object** – přenáší se objekty *cMessage*
- **bytestream** – přenáší se opravdové bajty

Jinou alternativou by bylo použití aplikace *TCPGenericSrvApp*. Klient a server si poté mezi sebou posílají zprávy definované typem zprávy *GenericAppMsg*, kdy klient posílá serveru informaci o tom, kolik bajtů mu má server poslat jako odpověď. [2]

Konfigurace bude tedy vypadat takto:

```
[General]
network = NETWORK

*.SERVER.numTcpApps = 1
*.HOST.numTcpApps = 1
*.HOST.tcpApp[0].typename="TCPBasicClientApp"
*.HOST.tcpApp[0].connectAddress = "SERVER"
*.HOST.tcpApp[0].connectPort = 80

*.SERVER.tcpApp[0].typename="TCPEchoApp"
*.SERVER.tcpApp[0].localPort = 80
```

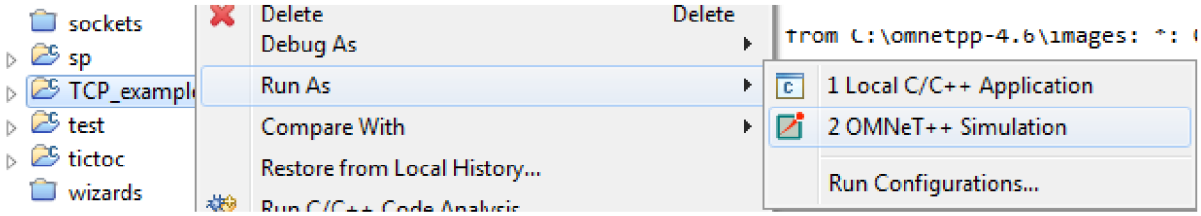
Nyní projekt zkompilujeme funkcí *Build Project*:




První kompilování po každém spuštění OMNeT++ trvá déle, další kompilace zabere pouze pár vteřin.

Simulaci spustíme funkcí *Run As OMNeT++ Simulation* a poté tlačítkem

.




Pokud bylo vše nastaveno správně, uvidíme, jak se v simulačním prostředí rozběhne přenos zpráv. V dolní části okna simulace v záložce „*Show message/packet traffic*“ můžeme sledovat podrobnosti každé zprávy. Nachází se zde tyto typy zpráv:

*Tabulka 5 Typy zpráv ve výstupu simulace*

| Event# | Time | Source/Destination | Name |
|---|---|---|---|
| #15 | 1.00000 | HOST --> SWITCH1 | arpREQ |
| #44 | 1.00001172 | ROUTER --> SWITCH1 | arpREPLY |
| #69 | 1.00002344 | HOST --> SWITCH1 | SYN |
| #195 | 1.00008204 | SERVER --> SWITCH2 | SYN+ACK |
| #227 | 1.00010548 | HOST --> SWITCH1 | ACK |
| #237 | 1.000112199999 | HOST --> SWITCH1 | tcpseg(l=200) |
| #362 | 1.000296679997 | HOST --> SWITCH1 | FIN |

*Tabulka 6 Typy zpráv ve výstupu simulace (pokračování)*

| Event# | Info |
|---|---|
| #15 | ARP req: 10.0.0.1=? (s=10.0.0.17(0A-AA-00-00-00-0F)) ETH: 0A-AA-00-00-00-0F > FF-FF-FF-FF-FF-FF (72 bytes) |
| #44 | ARP reply: 10.0.0.1=0A-AA-00-00-00-05 (d=10.0.0.17(0A-AA-00-00-00-0F)) ETH: 0A-AA-00-00-00-05 > 0A-AA-00-00-00-0F (72 bytes) |
| #69 | TCP: 10.0.0.17.1025 > 10.0.0.11.80: S 250000:250000(0) win 7504 IPv4: 10.0.0.17 > 10.0.0.11 ETH: 0A-AA-00-00-00-0F > 0A-AA-00-00-00-05 (72 bytes) |
| #195 | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A S 250020:250020(0) ack 250001 win 7504 IPv4: 10.0.0.11 > 10.0.0.17 ETH: 0A-AA-00-00-00-0D > 0A-AA-00-00-00-07 (72 bytes) |
| #227 | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250021 win 7504 IPv4: 10.0.0.17 > 10.0.0.11 ETH: 0A-AA-00-00-00-0F > 0A-AA-00-00-00-06 (72 bytes) |
| #237 | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A 250001:250201(200) ack 250021 win 7504 IPv4: 10.0.0.17 > 10.0.0.11 ETH: 0A-AA-00-00-00-0F > 0A-AA-00-00-00-06 (266 bytes) |
| #362 | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A F ack 250221 win 7504 IPv4: 10.0.0.17 > 10.0.0.11 ETH: 0A-AA-00-00-00-0F > 0A-AA-00-00-00-06 (72 bytes) |

1. Klient rozešle **ARP** požadavek jako všesměrové vysílání *(broadcast)* a následně dostává ARP odpověď s MAC adresou. Takto se všechna zařízení v síti informují o svých MAC adresách.
2. Jedno zařízení navazuje s druhým zařízením spojení zprávou **SYN** (synchronizace)
3. Opačná strana nabízené spojení akceptuje a zasílá zprávu **SYN + ACK** (synchronizace + potvrzení)
4. První strana posílá potvrzující zprávu **ACK** (potvrzení) a tímto dochází k navázání spojení
5. **Tcpseg** = TCP segment
6. Zpráva **FIN** (konec) znamená ukončení spojení

Kompletní výstup zpráv nalezneme níže jako přílohu. Velmi podrobné logy probíhající simulace se zobrazují v záložce *„Show module log“* a detailně popisují každou akci v simulaci. Zde je například log zprávy SYN+ACK na HOSTu.

| |
|---|
| ** **Event #219** t=1.00010548 NETWORK.HOST.eth[0].mac (EtherMACFullDuplex, id=95), on `SYN+ACK' (inet::EthernetIIFrame, id=357)<br>*Reception of (inet::EthernetIIFrame)SYN+ACK started.*<br>*Reception of (inet::EthernetIIFrame)SYN+ACK successfully completed.*<br>*Sending (inet::EthernetIIFrame)SYN+ACK to upper layer.* |
| ** **Event #220** t=1.00010548 NETWORK.HOST.eth[0].encap (EtherEncap, id=96), on `SYN+ACK' (inet::EthernetIIFrame, id=357)<br>*Received (inet::EthernetIIFrame)SYN+ACK from lower layer.*<br>*DETAIL: Decapsulating frame `SYN+ACK', passing up contained packet `SYN+ACK' to higher layer* |

| |
|---|
| *Sending (inet::IPv4Datagram)SYN+ACK to upper layer.* |
| ** **Event #221** t=1.00010548 NETWORK.HOST.networkLayer.ip (IPv4, id=89), on `SYN+ACK' (inet::IPv4Datagram, id=362)<br>*Received (inet::IPv4Datagram)SYN+ACK from network.*<br>*DETAIL: Received datagram `SYN+ACK' with dest=10.0.0.17*<br>*Delivering (inet::IPv4Datagram)SYN+ACK locally.* |
| ** **Event #222** t=1.00010548 NETWORK.HOST.tcp (TCP, id=87), on `SYN+ACK' (inet::tcp::TCPSegment, id=364)<br>*DETAIL: Connection <none>:1025 to 10.0.0.11:80 on app[0], connId=3 in SYN_SENT*<br>*Seg arrived: .80 > .1025: SYN+ACK [250020..250020) (l=0) ack 250001 win 7504 options MSS*<br>*DETAIL: TCB: snd_una=250000 snd_nxt=250001 snd_max=250001 snd_wnd=0 rcv_nxt=0 rcv_wnd=7504 snd_cwnd=0 rto=3 ssthresh=4294967295*<br>*DETAIL: Processing segment in SYN_SENT*<br>*DETAIL: ACK bit set, AckNo acceptable*<br>*Updating send window from segment: new wnd=7504*<br>*SYN+ACK bits set, connection established.*<br>*TCP Header Option(s) received:*<br>*DETAIL: Option type 2 (MSS), length 4*<br>*TCP Header Option MSS(=536) received, SMSS is set to 536*<br>*Completing connection setup by sending ACK (possibly piggybacked on data)*<br>*Sending: .1025 > .80: ack 250021 win 7504*<br>*Notifying app: ESTABLISHED*<br>*Transition: SYN_SENT --> ESTABLISHED (event was: RCV_SYN_ACK)*<br>*[testing] DEBUG: tcp: SYN_SENT --> ESTABLISHED (on RCV_SYN_ACK)* |

Kromě toho, že je možné pozorovat průběh zpráv v topologii tak, jak byla sestrojena, je možné rovněž sledovat děje na jednotlivých prvcích sítě a to dvojklikem na kterýkoliv z nich.

# 4. Quality of Service

Z hlediska kvality přenosu se dají data putující sítí rozdělit podle dopadu latence doručení zprávy na dva typy.

- Zprávy, kde nezáleží na rychlosti doručení – zpravidla data přenášející textovou nebo obrazovou zprávu.
- Zprávy citlivé na rychlost doručení – zpravidla zvuk a video.

Pokud jednou šířkou pásma prochází data různé povahy, resp. různé citlivosti na zpoždění, je to poznat na výstupu zprávy, pokud k takovému zpoždění opravdu dojde. Pokud přenášená data slouží k přenosu obrazu nebo textu, je nám prakticky jedno, jestli se zpozdí. Nezáleží až tak na tom, jestli email dorazí během dvou sekund nebo dvou minut, naopak důležité je, aby zpráva dorazila bez chyb. Naopak u zvuku nebo videa nemusí být signál úplně čistý, nicméně je důležité, aby byl přenos co nejvíce kontinuální. O to, aby jistý druh provozu měl přednost před ostatním se stará služba QoS.

Služba QoS se skládá ze tří fází, a to klasifikace, značkování a frontování:

- **Klasifikace** – identifikování dat na základě rozdílných vlastností a shlukování takových dat dohromady. Příkladem může být například hlasová VLAN.
- **Značkování** – značkování určitých paketů tak, aby je zbytek sítě mohl identifikovat a dal jim větší prioritu. IP telefony například značkují hlasové pakety pomocí *Class of Service* (CoS - „třída provozu“) a *Differentiated Services Code Point* (DSCP). CoS je pole umístěné v záhlaví ethernetového rámce a provoz dělí na 8 tříd od 0 po 7. Čím vyšší je hodnota CoS, tím je větší priorita. Defaultně je u hlasu třída 5, a pokud třída není nastavena, je považována za 0. DSCP je prakticky to samé co CoS, ale funguje na třetí vrstvě, čili na směrovačích a L3 přepínačích.
- **Frontování** – znamená seřazování paketů do fronty. Fronty jsou logická úložiště používaná pro odchozí provoz.

Podle doporučení ITU-T G.114 by zpoždění nemělo překročit 150 ms pro hlas. Podle Cisco doporučení by proměnlivé zpoždění *(jitter)* mělo zůstat pod 30 ms a ztráta paketů *(packetloss)* by neměl být vetší než 1 %. [3]

## 4.1 QoS modely a výhody a nevýhody navržených alternativ

Existují tři QoS modely:

- *Best-effort*
- *Integrated Services (IntServ)*
- *Differentiated Services (DiffServ) [3]*

### 4.1.1 Best effort

Tento model se zpravidla nijak nesnaží o zvýšení kvality služeb vůči různým stranám. Se všemi druhy provozu je zacházeno stejně. Takto funguje síť nebo část sítě, která nemá implementováno QoS. [3]

### 4.1.2 Integrated Services

Slouží k vyčleňování určitého kusu šířky pásma specifickému provozu. Takto vyčleněné pásmo není používané ostatním provozem, takže ani když se tento kus pásma zrovna nepoužívá, data z jiných pásmových šířek mohou být vystavena přetížení a ztrátě paketů. Tato služba používá *Resource Reservation Protocol* (RSVP). Častěji je nicméně používán model DiffServ. [3]

### 4.1.3 Differentiated Services

DiffServ používá značkování provozu založené buď na *Type of Service* (ToS) bajtech nebo *Differentiated Services* (DS) bajtech uložených v každém IP záhlaví. ToS se od DS nijak významně neliší, jde jen o starší pojmenování, nicméně v IP paketu zaujímají to samé pole.

ToS je osmice bitů, které definují požadavky odesílatele paketu na zacházení s paketem v průběhu jeho přenosu sítí.

Bity pole ToS jsou následující [4]:

*Tabulka 7 Bity pole ToS*

| Pořadí bitu | Význam |
|---|---|
| 1 | Priorita paketu – 0 je nejnižší, 7 je nejvyšší |
| 2 | |
| 3 | |
| 4 | Zpoždění – 0 = normální, 1 = nízké |
| 5 | Propustnost – 0 = normální, 1 = vysoká |
| 6 | Spolehlivost – 0 = normální, 1 = vysoká |
| 7 | Nevyužitá rezerva |
| 8 | |

Nyní se do pole pro ToS zapisují spíše hodnoty pro DiffServ, které vyjadřují „třídu provozu“ – *Class of Service* (CoS).

Jelikož máme k dispozici 6 bitů, máme tedy až 64 ($2^6$) možných označení priorit. IETF vytvořilo 4 strukturované DSCP typy nazvané *„per-hop behaviors“* (PHB), a to:

- Default PHB
- Expedited Forwarding (EF) PHB
- Assured Forwarding (AF) PHB
- Class Selector (CS) PHB [3]

#### *Default PHB*

Default PHB odpovídá modelu Best-effort. Používá se například pro protokol FTP nebo HTTP a další služby, které nejsou citlivé na zpoždění. V DCSP poli odpovídá hodnotě 0, čili 000000 binárně.

#### *Expedited Forwarding (EF) PHB*

Expedited Forwarding PHB je definováno v RFC 3246 a je používáno tam, kde je vyžadována nízká latence, ztráta paketů *(packetloss)* a proměnlivé zpoždění *(jitter)*. Používá se zejména u videa a hlasových služeb a jeho hodnota je 46, čili 101110 v binární formě.

#### *Assured Forwarding (AF) PHB*

Assured Forwarding PHB je definováno v RFC 2597 a 3260 a ve své skupině má 12 různých priorit, které jsou rozděleny do 4 dalších skupin, kde každá určuje tři pravděpodobnosti zahození paketu, a to následovně:

*Tabulka 8 Třídy assured forwarding*

| Pravděpodobnost zahození paketu | Třída 1 | Třída 2 | Třída 3 | Třída 4 |
|---|---|---|---|---|
| Nízká | AF11 (DCSP 10) | AF21 (DCSP 18) | AF31 (DCSP 26) | AF41 (DCSP 34) |
| Střední | AF12 (DCSP 12) | AF22 (DCSP 20) | AF32 (DCSP 28) | AF42 (DCSP 36) |
| Vysoká | AF13 (DCSP 14) | AF23 (DCSP 22) | AF33 (DCSP 30) | AF43 (DCSP 38) |

Čím vyšší je AF číslo třídy, tím vyšší prioritu paket má.

#### *Class Selector PHB*

Class Selector PHB je definován v RFC 2474. Technicky používá pouze 3 bity úplně vlevo a 3 bity úplně vpravo zůstávají 000. Pokud je tedy hodnota CS 110000 (decimálně 40), zařízení, která jsou kompatibilní pouze s ToS přečtou pouze 100, čili IP priorita 5, což odpovídá videu nebo hlasové službě. [3]

### 4.1.4 Porovnání všech tří QoS modelů

*Tabulka 9 Porovnání QoS modelů [3]*

| Model | Pro | Proti |
|---|---|---|
| **Best-effort** | Vysoce škálovatelný<br>Není potřeba změna konfigurace | Žádné rozdělení provozu<br>Služba není garantována |
| **IntServ** | Absolutní garance služby<br>Kompletní kontrola nad šířkou pásma | Neškálovatelnost kvůli komplexnosti konfigurace<br>Musí být nakonfigurované na každém směrovači v síti.<br>Plýtvání šířkou pásma<br>Složitá signalizace |
| **DiffServ** | Vysoce škálovatelný | Ne absolutní garance služby |

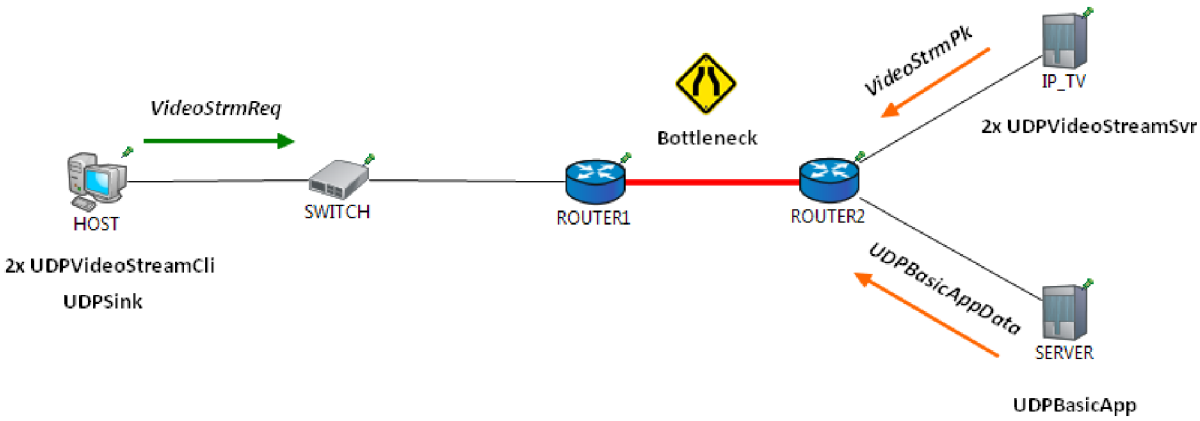
Mezi největší nevýhody IntServ služeb patří vyšší nároky na hardware, kde je nutná podpora protokolu RSVP na všech zařízeních, směrovače s RSVP udržují velký počet toků současně, obecně je nutná vyšší režie a je zde špatná škálovatelnost. Služba IntServ lze nejlépe použít pouze v malých sítích a na okrajích sítě. Další nevýhodou je orientace RSVP protokolu na přijímač, který musí být hlavním iniciátorem rezervace zdrojů. RSVP protokol selže, pokud vysílač neustanoví datový tok s definovanou QoS.

Nevýhodou služby Diffserv potom může být to, že PHB nemůže garantovat QoS po celé délce trasy mezi koncovými uzly. Model Diffserv je orientován pouze na vysílač. DiffServ poskytuje QoS služby pouze skupinám datových toků. [5,6]

# 5. Optimalizace návrhu sítě aplikací QoS v OMNeT++

## 5.1 Video vs. UDP zprávy

Jako praktickou realizaci QoS v simulátoru bude předvedena služba *DiffServ*. Bude použita síť tvořena jedním klientem, jedním přepínačem, dvěma směrovači a dvěma servery.




*Obrázek 12 Topologie Video vs. UDP*

Popis funkce této topologie je následující:

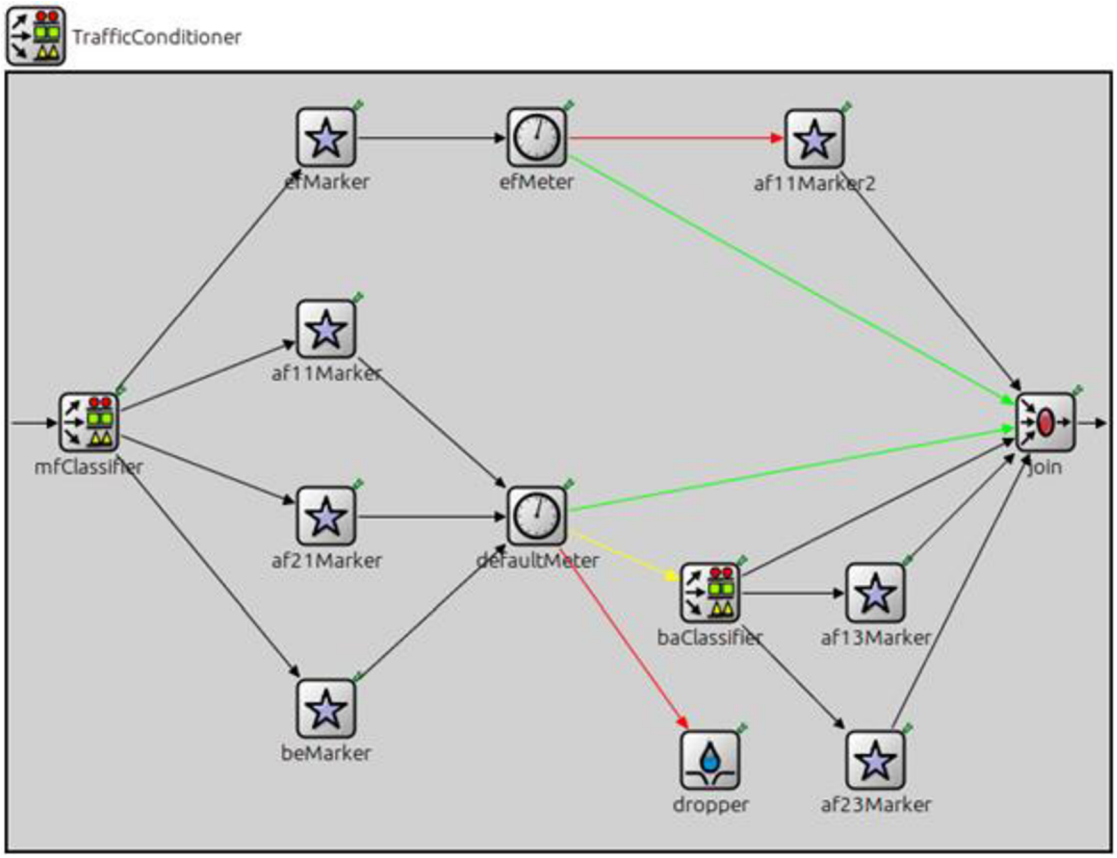
1. Klient používá celkem tři UDP aplikace. Dvě z nich slouží k přijímání video souborů (*VideoStrmPk*) od serveru nazvaného „IP_TV“ a třetí k přijímání UDP zpráv od serveru nazvaného jednoduše „SERVER“.
2. Všechny spoje v síti jsou nastaveny na rychlost 100 Mbps kromě spoje mezi směrovači, kde se nachází tzv. „*bottleneck*“ o 128 kbps a kde má právě docházet k upřednostňování zpráv s aplikovaným QoS.
3. Server posílající UDP zprávy bude posílat každou jednu sekundu zprávy o větší velikosti, než je propustnost spoje mezi směrovači a zprávy se budou tudíž dělit na fragmenty (*UDPBasicAppData-n-frag* a *UDPBasicAppData-n-frag-frag*).
4. Na směrovačích jsou nastaveny dvě konfigurace. Jedna, ve které je aplikováno QoS a druhá, která je prázdná, tudíž bez QoS.

Nastavení na klientovi a na serverech vypadá následovně:

```
[General]

*.IP_TV.numUdpApps = 2
*.SERVER.numUdpApps = 1
*.HOST.numUdpApps = 3

**.configurator.config = xml("<config><interface hosts='HOST' names='eth0'
address='192.168.24.x' netmask='255.255.255.x'/><interface hosts='ROUTER1'
names='eth0' address='192.168.24.x' netmask='255.255.255.x'/><interface
hosts='ROUTER1' names='ppp0' address='192.168.24.x'
netmask='255.255.255.x'/><interface hosts='ROUTER2' names='ppp0'
address='192.168.24.x' netmask='255.255.255.x'/><interface hosts='ROUTER2'
names='ppp1' address='192.168.24.x' netmask='255.255.255.x'/><interface
hosts='ROUTER2' names='ppp2' address='192.168.24.x'
netmask='255.255.255.x'/><interface hosts='IP_TV' names='ppp0'
address='192.168.24.x' netmask='255.255.255.x'/><interface hosts='SERVER'
names='ppp0' address='192.168.24.x' netmask='255.255.255.x'/></config>")

*.HOST.udpApp[0].typename="UDPVideoStreamCli"
```

```
*.HOST.udpApp[0].serverAddress = "IP_TV"
*.HOST.udpApp[1].typename="UDPVideoStreamCli"
*.HOST.udpApp[1].serverAddress = "IP_TV"
*.HOST.udpApp[0].localPort = 90
*.HOST.udpApp[1].localPort = 91
*.HOST.udpApp[0].serverPort = 100
*.HOST.udpApp[1].serverPort = 101

*.HOST.udpApp[2].typename="UDPSink"
*.HOST.udpApp[2].localPort = 92

*.IP_TV.udpApp[*].typename="UDPVideoStreamSvr"
*.IP_TV.udpApp[0].localPort = 100
*.IP_TV.udpApp[1].localPort = 101
*.IP_TV.udpApp[*].packetLen = 1000B
*.IP_TV.udpApp[0].videoSize = 10000B
*.IP_TV.udpApp[*].sendInterval = 1s
*.IP_TV.udpApp[1].videoSize = 1000000B

*.SERVER.udpApp[0].typename="UDPBasicApp"
*.SERVER.udpApp[0].destAddresses = "HOST"
*.SERVER.udpApp[0].destPort = 92
*.SERVER.udpApp[0].localPort = 102
*.SERVER.udpApp[0].messageLength = 20000B
*.SERVER.udpApp[0].sendInterval = 1s
```

K zavedení QoS do simulace je v tomto případě zapotřebí upravit parametry na směrovačích a nastavit filtr, který bude určovat, který provoz bude využívat QoS a který bude fungovat jako *Best Effort*. Na směrovačích byl nastaven tzv. *TrafficConditioner*, který značkuje jednotlivé pakety určitou QoS prioritou. Jeho struktura vypadá následovně:




*Obrázek 13 Traffic Conditioner*

Je zde tedy vidět, že se provoz po průchodu modulem „*multifield classifier*“ rozdělí podle priority, kterou byl označkován na provoz EF, AF11, AF21 a BE (best effort) a dále pokračuje do modulu *efMeter* a *defaultMeter*, který zprávu buď dále značkuje, nebo ji zahodí, nebo pošle dál.

*TrafficConditioner* je zaveden pomocí příkazu:

```
**.ROUTER1.ppp[*].egressTCType = "TrafficConditioner"
**.ROUTER1.ppp[*].ingressTCType = "TrafficConditioner"
```

*Egress* značí odchozí provoz a *ingress* příchozí. Směrování paketů z *TrafficConditioneru* do *efMeteru*, nebo *defaultMeteru* se nastaví jako poměr důležitosti těchto dvou cest příkazem

```
**.efMeter.cir = "XY%"
```

a

```
**.defaultMeter.cir = "XY%"
```

Místo procentuálních hodnot je možné také použít požadovanou šířku pásma. Čím větší je zastoupení EF značkování, tím větší je priorita paketu. Čím větší je zastoupení defaultního značkování, tím větší je místo pro provoz s Best Effort značkováním.

Nastavení QoS na směrovačích potom vypadá takto:

```
[Config Without_QoS]
description = "Simulace bez QoS"

[Config With_QoS]
description = "DiffServ vyhodnocování provozu je zapnuté na všech ppp[*] rozhraních na obou směrovačích"
**.ROUTER1.ppp[*].egressTCType = "TrafficConditioner"
**.ROUTER1.ppp[*].ingressTCType = "TrafficConditioner"
**.ROUTER2.ppp[*].egressTCType = "TrafficConditioner"
**.ROUTER2.ppp[*].ingressTCType = "TrafficConditioner"

**.efMeter.cir = "70%" # cir = committed information rate
**.efMeter.cbs = 50KiB
**.defaultMeter.cir = "30%"
**.defaultMeter.cbs = 2KiB
**.defaultMeter.ebs = 4KiB
```

Aby *TrafficConditioner* věděl, na který druh provozu má aplikovat QoS, je nutné vytvořit filtr. Filtr je textový soubor formátu xml, který udává směr provozu, který má být upřednostněn. Je uložen ve složce *Simulations* a v tomto případě je nastaven tak, aby filtroval zprávy s video pakety:

```
<filters>
  <filter destAddress="IP_TV" protocol="udp" destPort="100" gate="0"/>
  <filter destAddress="IP_TV" protocol="udp" destPort="101" gate="1"/>
  <filter destAddress="HOST" protocol="udp" destPort="90" gate="0"/>
  <filter destAddress="HOST" protocol="udp" destPort="91" gate="1"/>
</filters>
```

Zde je porovnání simulace s a bez aplikace QoS pro prvních 10 sekund simulace:

*Tabulka 10 Porovnání s QoS a bez QoS*

| | S QoS | Bez QoS |
|---|---|---|
| **Počet video paketů přijatých klientem** | 20 | 14 |
| **Počet UDP zpráv (fragmentů) přijatých klientem** | 20 | 126 |
| **Počet video paketů přijatých na ROUTER2** | 20 | 20 |
| **Počet UDP zpráv (fragmentů) přijatých na ROUTER2** | 50 | 50 |

Je tedy vidno, že v simulaci bez QoS bylo přeneseno více UDP zpráv na úkor video paketů.

V podrobném výstupu zpráv simulace můžeme potom pozorovat tato hlášení:

```
** Event #85  t=1.00003516  myqos.ROUTER1.ppp[0].egressTC.mfClassifier
(MultiFieldClassifier, id=348) on VideoStrmReq (inet::IPv4Datagram,
id=9967)
** Event #86  t=1.00003516  myqos.ROUTER1.ppp[0].egressTC.efMarker
(DSCPMarker, id=351) on VideoStrmReq (inet::IPv4Datagram, id=9967)
DETAIL (DSCPMarker)myqos.ROUTER1.ppp[0].egressTC.efMarker: Marking packet
with dscp=EF
```

```
** Event #102  t=1.00004188  myqos.ROUTER1.ppp[0].egressTC.mfClassifier
(MultiFieldClassifier, id=348) on VideoStrmReq (inet::IPv4Datagram,
id=9989)
** Event #103  t=1.00004188  myqos.ROUTER1.ppp[0].egressTC.af11Marker
(DSCPMarker, id=352) on VideoStrmReq (inet::IPv4Datagram, id=9989)
DETAIL (DSCPMarker)myqos.ROUTER1.ppp[0].egressTC.af11Marker: Marking
packet with dscp=AF11
```

```
** Event #111  t=1.0003581  myqos.ROUTER2.ppp[2].ingressTC.mfClassifier
(MultiFieldClassifier, id=225) on UDPBasicAppData-0-frag
(inet::IPv4Datagram, id=9895)
** Event #112  t=1.0003581  myqos.ROUTER2.ppp[2].ingressTC.beMarker
(DSCPMarker, id=234) on UDPBasicAppData-0-frag (inet::IPv4Datagram,
id=9895)
DETAIL (DSCPMarker)myqos.ROUTER2.ppp[2].ingressTC.beMarker: Marking
packet with dscp=BE
```

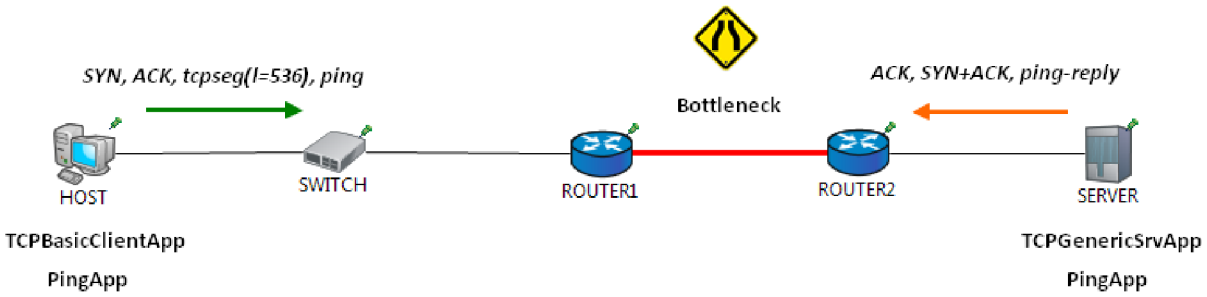
```
** Event #176  t=1.004475657691
myqos.ROUTER2.ppp[1].ingressTC.mfClassifier (MultiFieldClassifier,
id=181) on VideoStrmPk (inet::IPv4Datagram, id=10095)
** Event #177  t=1.004475657691  myqos.ROUTER2.ppp[1].ingressTC.efMarker
(DSCPMarker, id=184) on VideoStrmPk (inet::IPv4Datagram, id=10095)
DETAIL (DSCPMarker)myqos.ROUTER2.ppp[1].ingressTC.efMarker: Marking
packet with dscp=EF
```

Vidíme tedy, že QoS funguje tak, jak má. Žádosti o video jsou značkované jako EF nebo AF11, UDP fragmenty jako Best Effort a video pakety jako EF.

## 5.2 Ping vs. TCP zprávy

Jako další příklad bude aplikováno QoS na TCP zprávy, které budou rovněž procházet místem s menší šířkou pásma a budou o něj soupeřit s ICMP pakety. Cílem této simulace je předvést funkčnost QoS a především funkci parametru *efMeter* (viz níže).

Topologie bude následující:




*Obrázek 14 Topologie Ping vs. TCP*

Klient používá jednu aplikaci pro TCP komunikaci a 6 totožných aplikací pro ping. Server používá jednu aplikaci pro TCP komunikaci a jednu aplikaci pro ping. Nastavení potom vypadá takto:

```
[General]

*.SERVER.numTcpApps = 1
*.SERVER.numPingApps = 1
*.HOST.numTcpApps = 1
*.HOST.numPingApps = 6

*.HOST.tcpApp[*].typename="TCPBasicClientApp"
*.HOST.tcpApp[*].connectAddress = "SERVER"
*.HOST.tcpApp[*].localAddress = "HOST"
*.HOST.tcpApp[*].localPort = 90
*.HOST.tcpApp[*].connectPort = 100
*.HOST.tcpApp[*].startTime = 0s
*.HOST.tcpApp[*].thinkTime = 0.2s
*.HOST.tcpApp[*].idleInterval = 0.2s
*.HOST.tcpApp[*].requestLength = 300000B
*.HOST.tcpApp[*].replyLength = 300000B

*.HOST.pingApp[*].typename="PingApp"
*.HOST.pingApp[*].destAddr="SERVER"
*.HOST.pingApp[*].count=30
*.HOST.pingApp[*].sendInterval=0.2s
*.HOST.pingApp[*].packetSize=1000B

*.SERVER.tcpApp[*].typename="TCPGenericSrvApp"
*.SERVER.tcpApp[*].localPort = 100

*.SERVER.pingApp[*].typename="PingApp"
*.SERVER.pingApp[*].srcAddr="HOST"
```

Klient tedy posílá serveru každých 0,2 sekund TCP zprávy a žádosti o odpověď na ping. Těch je 30 z 6 aplikací, čili dohromady 180. V praxi je to relativně málo, ale při simulaci to postačuje k zahlcení linky ICMP pakety.

Na směrovačích je aplikováno toto nastavení, které je stejné jako v předchozím případě:

```
[Config Without_QoS]

description = "Simulace bez QoS"

[Config With_QoS]
```

```
description = "DiffServ vyhodnocování provozu je zapnuté na všech ppp[*] rozhraních na obou směrovačích"
**.ROUTER1.ppp[*].egressTCType = "TrafficConditioner"
**.ROUTER1.ppp[*].ingressTCType = "TrafficConditioner"
**.ROUTER2.ppp[*].egressTCType = "TrafficConditioner"
**.ROUTER2.ppp[*].ingressTCType = "TrafficConditioner"

**.efMeter.cir = "70%"
**.efMeter.cbs = 50KiB
**.defaultMeter.cir = "30%"
**.defaultMeter.cbs = 2KiB
**.defaultMeter.ebs = 4KiB
```

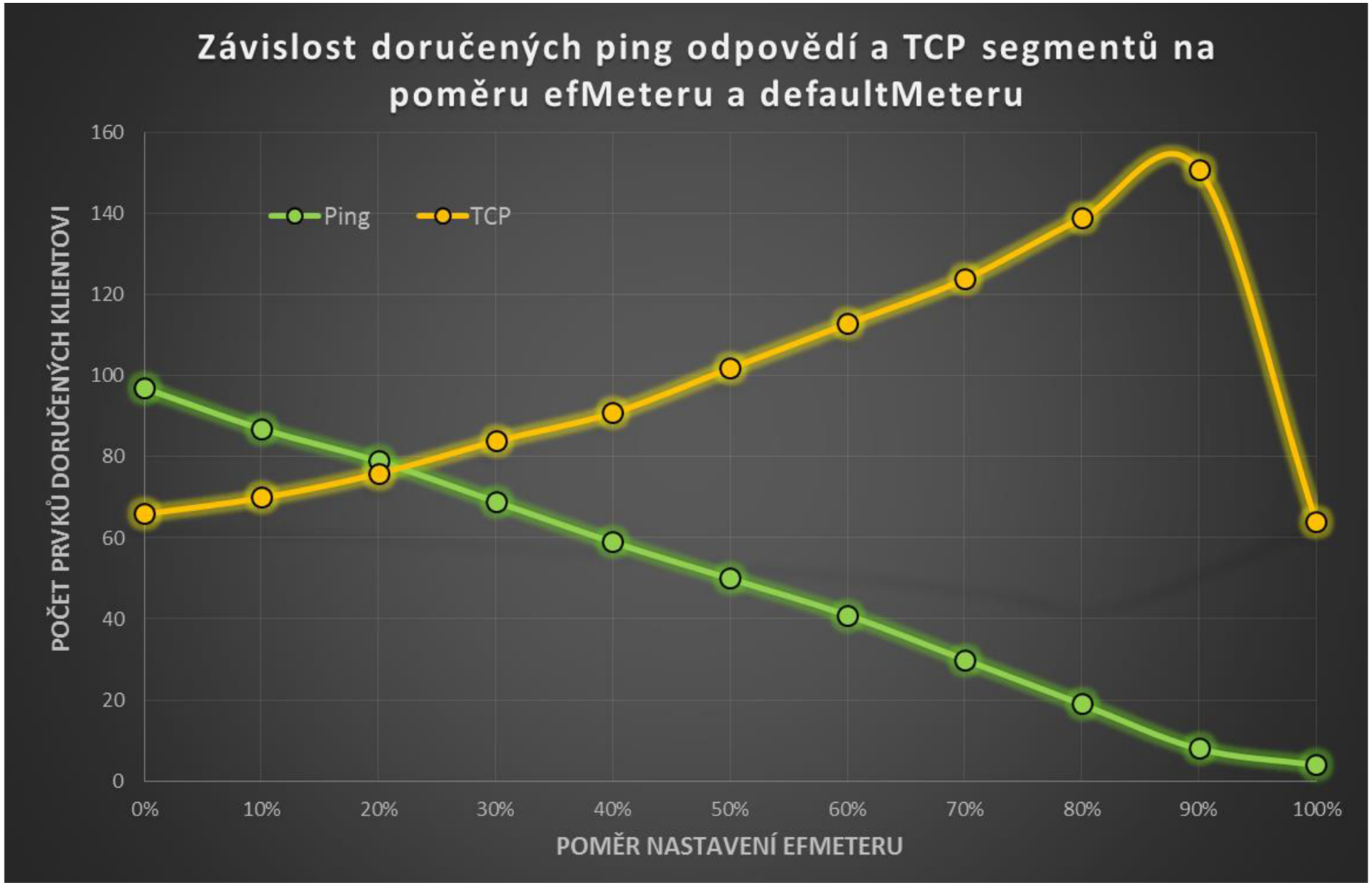
Filtr byl nastaven tak, aby bylo QoS aplikováno pouze na TCP zprávy:

```
<filters>
  <filter destAddress="SERVER" protocol="tcp" destPort="100" gate="0"/>
  <filter destAddress="HOST" protocol="tcp" destPort="90" gate="0"/>
</filters>
```

Pro představu o fungovaní efMeteru, bylo provedeno několik simulací s různým nastavením parametru *efMeter.cir* a *defaultMeter.cir*. Simulace byla ve všech případech zastavena po poslední odpovědi na ping a byl sledován úbytek přijatých odpovědí na ping při zvyšování zastoupení efMeteru a zároveň nárůst počtu včas doručených TCP zpráv. Jelikož se jedná o TCP, nedochází zde k zahazování TCP segmentů, ale k jejich znovuodeslání, proto je vhodné sledovat simulaci pouze do ukončení aplikace PingApp. Výsledky jsou uvedeny zde v tabulce a dále v grafu. Při hodnotě *efMeter.cir* = 100 % se rapidně zmenšil počet doručených TCP zpráv, což bylo způsobeno nepřesnými nebo chybějícími daty ve výstupu simulace, jelikož bylo možné pozorovat, že byla řada zpráv ve výstupu vynechána.

*Tabulka 11 Ukázka simulace s různými hodnotami efMeter.cir*

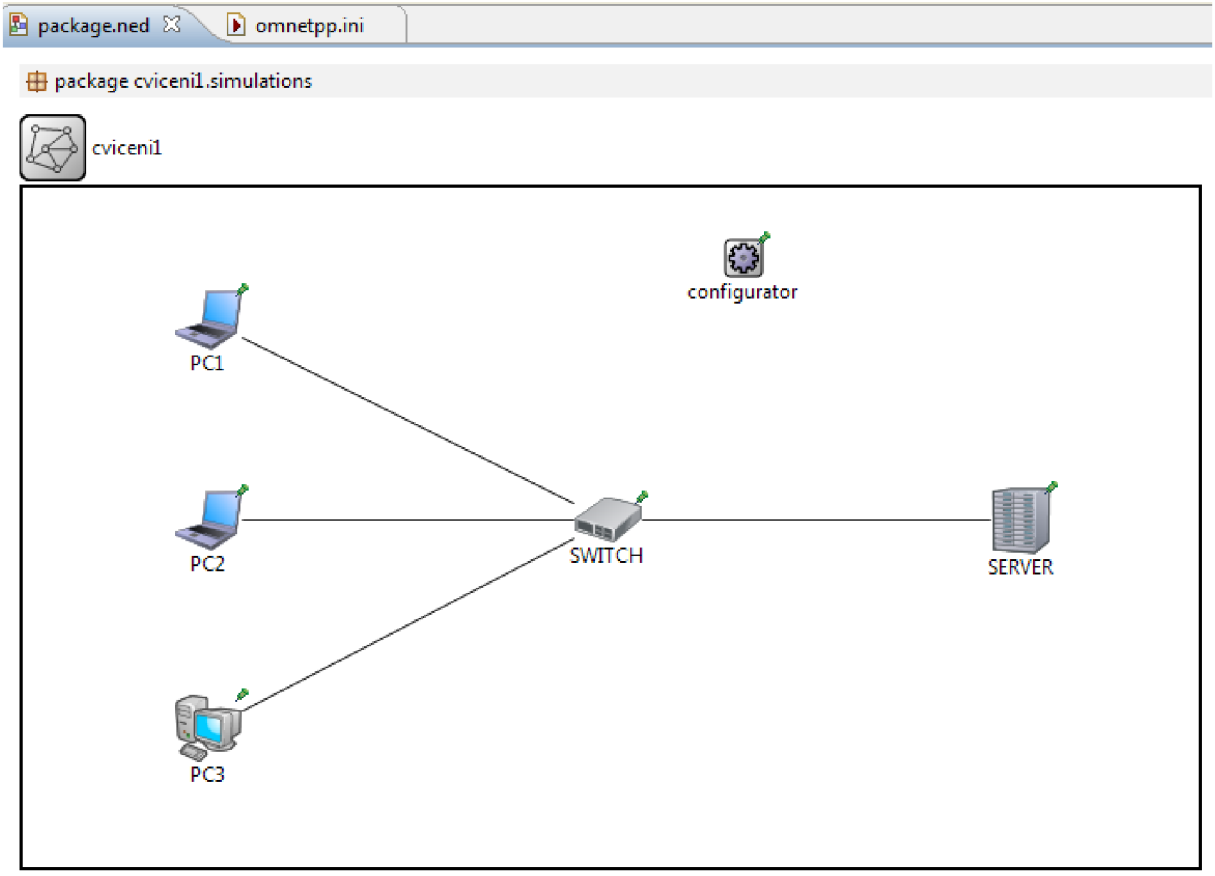
| Nastavení parametru efMeter.cir | Nastavení parametru defaultMeter.cir | Počet doručených odpovědí na ping | Počet doručených TCP zpráv |
|---|---|---|---|
| **bez QoS** | bez QoS | 162 | 44 |
| **0%** | 100% | 97 | 66 |
| **10%** | 90% | 87 | 70 |
| **20%** | 80% | 79 | 76 |
| **30%** | 70% | 69 | 84 |
| **40%** | 60% | 59 | 91 |
| **50%** | 50% | 50 | 102 |
| **60%** | 40% | 41 | 113 |
| **70%** | 30% | 30 | 124 |
| **80%** | 20% | 19 | 139 |
| **90%** | 10% | 8 | 151 |
| **100%** | 0% | 4 | 64 |


Závislost doručených ping odpovědí a TCP segmentů na poměru efMeteru a defaultMeteru
POČET PRVKŮ DORUČENÝCH KLIENTOVI
160
140
120
100
80
60
40
20
0
Ping
TCP
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
POMĚR NASTAVENÍ EFMETERU

# 6. Laboratorní cvičení 1 – Sestavení topologie a spuštění jednoduché simulace

## 6.1 Zadání

1. Vytvořte v OMNeT++ nový prázdný projekt.
2. Importujte do projektu simulační model INET.
3. Vytvořte síť, která se bude skládat ze tří klientů, jednoho přepínače a jednoho serveru a prostředí upravte tak, aby síť vypadala takto:




*Obrázek 15 Topologie pro laboratorní úlohu 1*

4. Zařízení propojte ethernetovým kabelem o 100 Mbps
5. V konfiguračním prostředí *omnetpp.ini* síť adekvátně nakonfigurujte.
6. Každému zařízení přiřaďte aplikaci *PingApp* a nastavte počet odeslaných žádostí o odpověď na ping, interval jejich odesílání a velikost ICMP paketu.
7. Pozorováním simulace a jejího výstupu se přesvědčte, že všichni klienti dostávají od serveru odpověď na ping.

## 6.2 Teoretický úvod

OMNeT++ je nástroj pro tvorbu simulací síťového provozu, jehož části fungují na bázi jazyka C++. V OMNeT++ je možné si síťové prvky buď tvořit nové, nebo používat již přednastavené moduly nadstavby INET, případně tyto modifikovat dle libosti. Cílem této laboratorní úlohy je sestavit jednoduchou síť a seznámit se s fungováním simulace a s možnostmi aplikace INET. Úkolem této úlohy je sestavit předem určenou topologii sítě a zprovoznit na ní komunikaci pomocí služby ping. Hlavním základem OMNeT++ simulace je soubor *NED (package.ned)*, který popisuje topologii sítě a soubor *INI (omnetpp.ini)*, kterým síť konfigurujeme. Oba tyto soubory umožňují práci s kódem i práci v GUI.

V souboru NED bude použit modul *StandardHost*, *IPv4NetworkConfigurator*, *EtherSwitch* a *EtherLink*, které jsou součástí balíčku simulačních modelů INET. *StandardHost* představuje hosta v sítí, například PC, notebook, server, IP telefon. *EtherLink* je ethernetový spoj, jehož podtřídami se dá nastavit rychlost spoje (*Eth10M, Eth100M*, *Eth1G, Eth10G*, *Eth40G, Eth100G*). *IPv4NetworkConfigurator* přiděluje zařízením IP adresy a nastavuje statické směrování.

Provoz ping zpráv umožňuje aplikace nazvaná *PingApp*. Klient zde posílá druhé straně zprávy *„ping#"* a jako odpověď dostává zprávy *„ping#-reply"*, kde # značí pořadové číslo jednotlivých žádostí o ping odpověď.

Aby tato aplikace fungovala, jak má, je potřeba nastavit tyto parametry:

- **destAddr** – cílová adresa v uvozovkách. Pokud není nastavena, čili je zde pouze "", ping je vypnutý. "*" značí všechna rozhraní v dané simulaci. Jako adresu uvádíme jméno cílového zařízení, nikoliv jeho IP adresu (například "SERVER")
- **srcAddr** – zdrojová adresa
- **packetSize** – velikost ICMP paketu v bajtech
- **sendInterval** – interval v sekundách, ve kterém jsou odesílané jednotlivé ping zprávy
- **hopLimit** – TTL
- **count** – počet poslaných ICMP paketů, -1 značí nekonečno
- **startTime** – čas prvního ICMP paketu v sekundách
- **stopTime** – čas posledního ICMP paketu v sekundách, -1 značí nekonečno

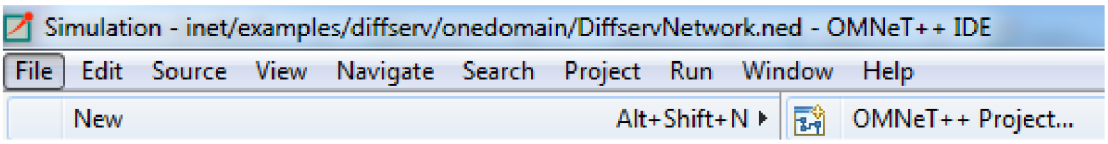
Dále je nutné nastavit počet aplikací na daném zařízení. Pokud je tento příkaz vynechán, aplikace je ignorována.

Parametry se nastavují tímto způsobem: *nazev_projektu.nazev_modulu.nazev_aplikace[*].parametr = hodnota parametru*, kde * je identifikační číslo aplikace, pokud se na jednom zařízení/modulu nachází více stejných aplikací (začíná se nulou). Hvězdička značí, že daný parametr platí pro všechno tyto aplikace.

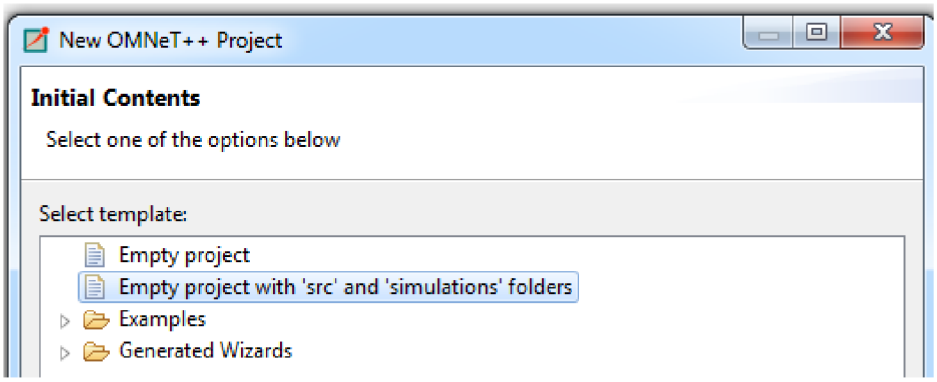
Příklad: `cviceni1.PC1.pingApp[1].srcAddr = "PC1"`. Název projektu je logicky vždy stejný a proto může být nahrazen *.

## 6.3 Pracovní postup

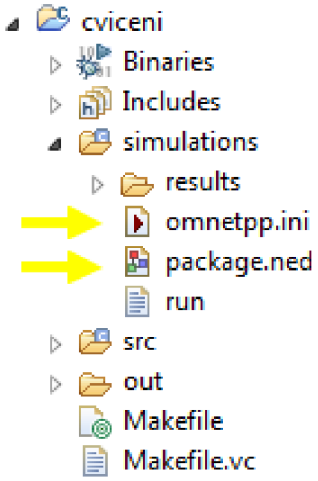
1) Spusťte OMNeT++ zadáním příkazu `omnetpp` do příkazové řádky souboru *mingwenv.cmd*, který naleznete na Ploše.
2) Vytvořte prázdný projekt následujícím způsobem:
   a. Přes záložku *File* vytvořte *„OMNeT++ Project…"* a ten libovolně pojmenujte (bez mezer a diakritiky)



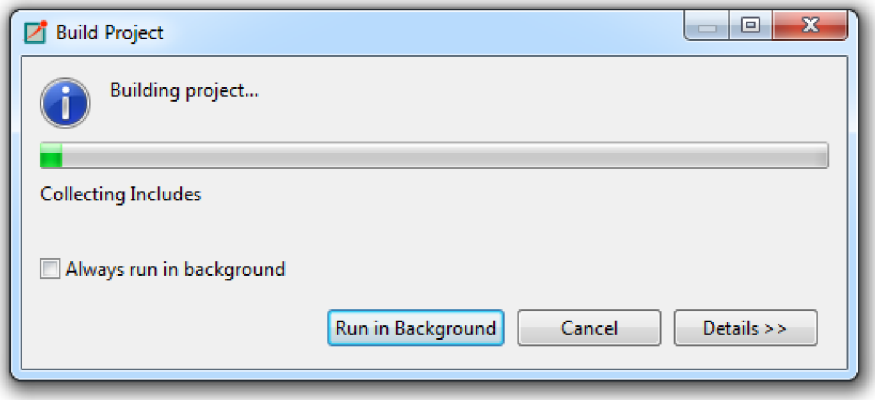

   b. V dalším kroku zvolte možnost *„Empty project with ‚src' and ‚simulations' folders"*



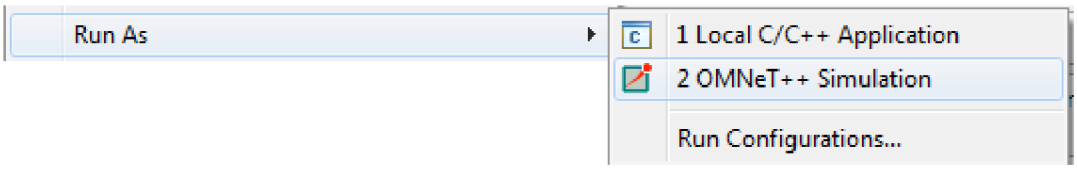

3) Vlevo v *Project Explorer* se objevila nová složka s vaším projektem. Ze složky *Simulations* otevřete soubory *package.ned* a *omnetpp.ini*.




4) Pravou myší klikněte na složku se soubory a v *Properties* pod *Project References* naimportuje do projektu *INET framework*. Následně se vpravo zobrazí nabídka modulů a podmodulů.
5) Pomocí těchto prvků vytvořte nejprve prázdnou síť a do ní vkládejte zařízení podle topologie v zadání. Nezapomeňte přidat *IPv4NetworkConfigurator*, který slouží k přiřazení IP adres v síti a zajištění směrování. Pojmenujte jej jako „*configurator*“.
6) Přes *Properties* nastavte ikony a barvu pozadí.
7) V souboru *omnetpp.ini* proveďte nastavení aplikace *PingApp* pro server a každý z klientů. Klientům nastavte počet odeslaných žádostí o odpověď na ping, interval jejich odesílání a velikost ICMP paketu. Parametry, které je možné nastavit, najdete v záložce *Form*.
8) Pravou myší klikněte na vámi vytvořený soubor a poté na „*Build Project*“, čímž zahájíte kompilaci. Po prvním spuštění OMNeT++ trvá kompilace déle, další kompilace zaberou zhruba 3 vteřiny.

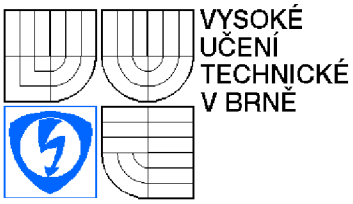


9) Znovu klikněte pravou myší na projekt a přes *„Run As OMNeT++ Simulation“* se dostanete do simulačního prostředí.



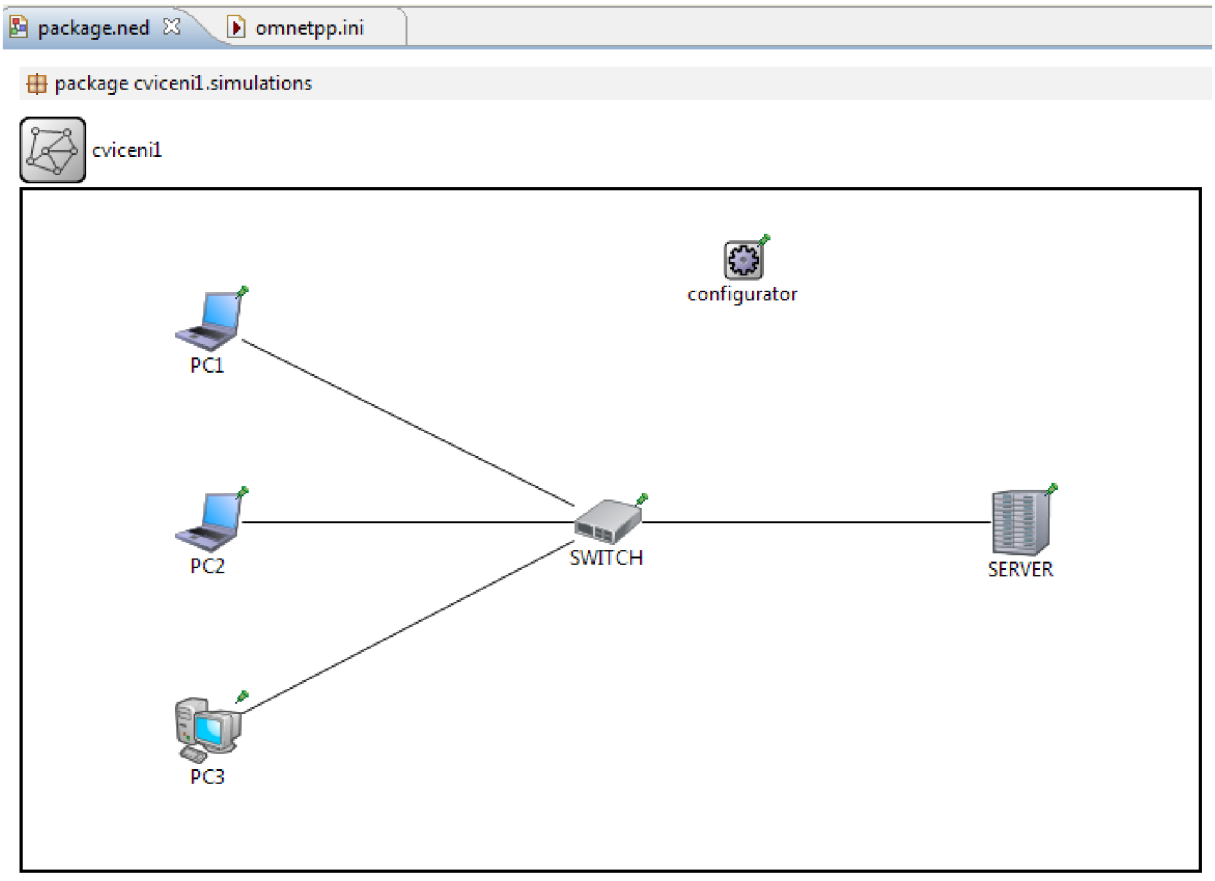

10) Simulaci zapněte tlačítkem a sledujte výstup zpráv v dolní části simulace. Pokud není hlášena žádná chyba a server odpoví každému z klientů, znamená to, že je projekt nakonfigurován správně.

## 6.4 Vzorový protokol

<table>
<tr><td rowspan="4"> 
</td><td colspan="2">Předmět</td></tr>
<tr><td colspan="2">Jméno</td></tr>
<tr><td>Ročník</td><td>Studijní skupina</td></tr>
<tr><td>Spolupracoval</td><td>Datum</td></tr>
<tr><td colspan="2">Kontroloval</td><td>Hodnocení</td><td>Datum</td></tr>
<tr><td>Číslo úlohy<br>X</td><td colspan="3">Název úlohy<br>Optimalizace síťového provozu pomocí OMNeT++ - Sestavení topologie a spuštění jednoduché simulace</td></tr>
</table>

### Zadání

1. Vytvořte v OMNeT++ nový prázdný projekt.
2. Importujte do projektu simulační model INET.
3. Vytvořte síť, která se bude skládat ze tří klientů, jednoho přepínače a jednoho serveru a prostředí upravte tak, aby síť vypadala takto:




4. Zařízení propojte ethernetovým kabelem o 100 Mbps
5. V konfiguračním prostředí *omnetpp.ini* síť adekvátně nakonfigurujte.

6. Každému zařízení přiřaďte aplikaci *PingApp* a nastavte počet odeslaných žádostí o odpověď na ping, interval jejich odesílání a velikost ICMP paketu.
7. Pozorováním simulace a jejího výstupu se přesvědčte, že všichni klienti dostávají od serveru odpověď na ping.

## Vypracování

- Byl vytvořen nový projekt jako „Prázdný projekt se ‚src' a ‚simulations' složkami" a byl pojmenován jako „cviceni1".
- V souboru *package.ned* byla sestavena topologie sítě podle zadání.
- V souboru *omnetpp.ini* byla nastavena aplikace *PingApp* na všech nařízeních a to tak, aby byly vyslány 4 žádosti o odpověď na ping o velikosti 10 B v intervalu 0,2 s.
- Podle výstupu simulace můžeme vidět, že všichni klienti obdrželi celkem 4 odpovědi na ping. Výstup zpráv je v rámci úspory místa vyfiltrován pouze na zprávy, které dorazily od přepínače ke klientovi.

*Tabulka 12 Výstup zpráv simulace s aplikací PingApp – laboratorní úloha 1*

| ***Event#*** | **Time** | **Src/Dest** | **Name** |
|---|---|---|---|
| *#80* | 0.109803370461 | SWITCH --> PC1 | ping0-reply |
| *#161* | 0.120593344076 | SWITCH --> PC2 | ping0-reply |
| *#242* | 0.17162979397 | SWITCH --> PC3 | ping0-reply |
| *#277* | 0.309780130461 | SWITCH --> PC1 | ping1-reply |
| *#312* | 0.320570104076 | SWITCH --> PC2 | ping1-reply |
| *#347* | 0.37160655397 | SWITCH --> PC3 | ping1-reply |
| *#382* | 0.509780130461 | SWITCH --> PC1 | ping2-reply |
| *#417* | 0.520570104076 | SWITCH --> PC2 | ping2-reply |
| *#452* | 0.57160655397 | SWITCH --> PC3 | ping2-reply |
| *#487* | 0.709780130461 | SWITCH --> PC1 | ping3-reply |
| *#522* | 0.720570104076 | SWITCH --> PC2 | ping3-reply |
| *#557* | 0.77160655397 | SWITCH --> PC3 | ping3-reply |

## Závěr

Podle zadání byla vytvořena topologie sítě a byla nastavena pro provoz ICMP zpráv pomocí aplikace *PingApp*. Z výstupu zpráv simulace byla ověřena správnost nastavení projektu, každý z klientů dostal příslušný počet odpovědí na ping.

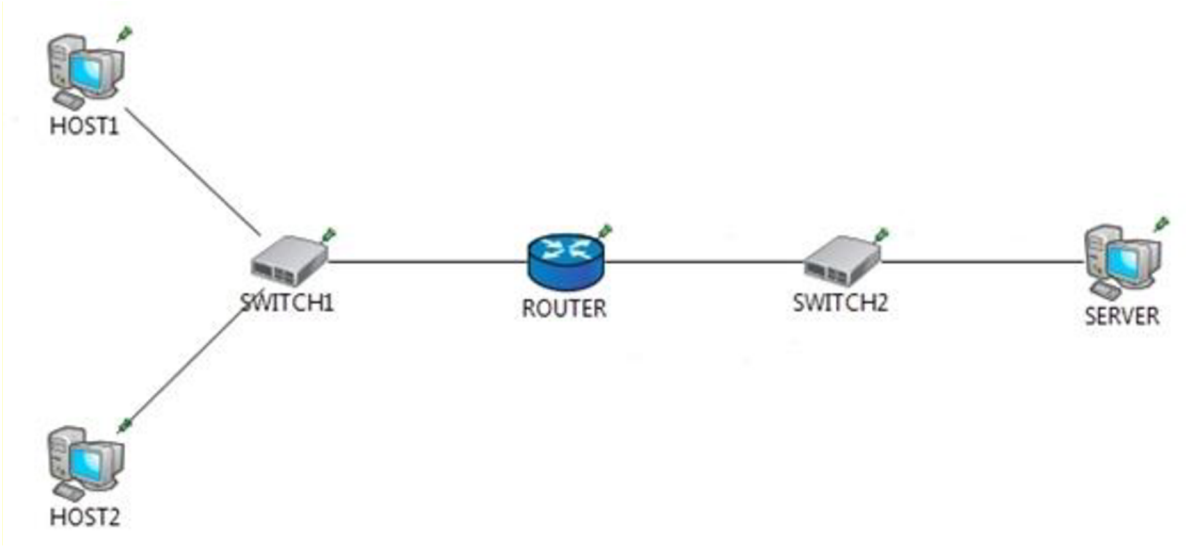
# 7. Laboratorní cvičení 2 - Nastavení aplikací

## 7.1 Zadání

1) Vytvořte v OMNeT++ nový projekt. Pro sestavení topologie a nastavení simulace použijte konfiguraci přiloženou níže (viz příloha).
2) Projekt zkompilujte a spusťte
3) Proveďte tyto úkoly:
    a. *Sledujte probíhající zprávy. Po chvíli se objeví chybová hláška – vysvětlete ji a upravte konfiguraci tak, aby provoz běžel bezchybně (je třeba upravit/přidat dva řádky)*
    b. *UDP aplikaci na jednom z hostů nahraďte aplikací SimpleVoIPSender a na serveru aplikací SimpleVoIPReceiver. Sledujte výstup simulace. Nastavte interval odesílání zpráv tak, aby byl počet zpráv obou aplikací stejného množství.*

## 7.2 Teoretický úvod

OMNeT++ je nástrojem pro tvorbu simulací síťového provozu, jehož části fungují na bázi jazyka C++. V OMNeT++ je možné si síťové prvky buď tvořit nové, nebo používat již přednastavené moduly nadstavby INET, případně tyto modifikovat dle libosti. Cílem této laboratorní úlohy je sestavit jednoduchou síť a seznámit se s fungováním simulace a s možnostmi aplikace INET. Student má za úkol zprovoznit běh simulace z již připraveného kódu a nalézt v něm chybu a tu opravit. Druhým úkolem je pozměnit konfiguraci za zachování funkčnosti simulace. Student bude pracovat s touto topologií:




*Obrázek 16 Topologie pro laboratorní úlohu 2*

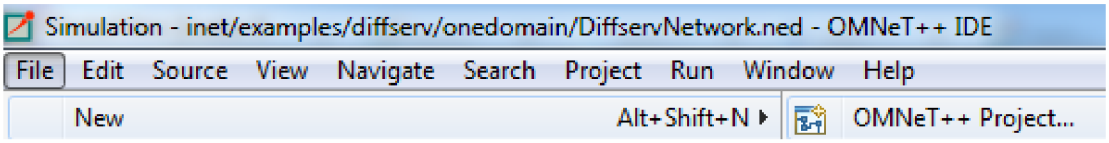
Na stolních PC a na serveru jsou implementované UDP aplikace, takže simulace znázorňuje přenos UDP paketů. Hlavním základem OMNeT++ simulace je soubor *NED*, který popisuje topologii sítě a soubor *INI*, kterým síť konfigurujeme. Oba tyto soubory umožňují práci s kódem i práci v GUI. V souboru NED je použit modul *StandardHost*, *Router*, *IPv4NetworkConfigurator*, *EtherSwitch* a *EtherLink*, které jsou součástí balíčku simulačních modelů INET. *StandardHost* představuje hosta v sítí, například PC, notebook, server, IP telefon. *EtherLink* je ethernetový spoj, jehož podtřídami se dá nastavit rychlost spoje (*Eth10M, Eth100M*, *Eth1G, Eth10G*, *Eth40G, Eth100G*). *IPv4NetworkConfigurator* přiděluje zařízením IP adresy a nastavuje statické směrování. V souboru INI je použita aplikace *UDPBasicApp*, která posílá druhé straně UDP pakety v určitém námi zvoleném intervalu. [1]

## 7.3 Pracovní postup

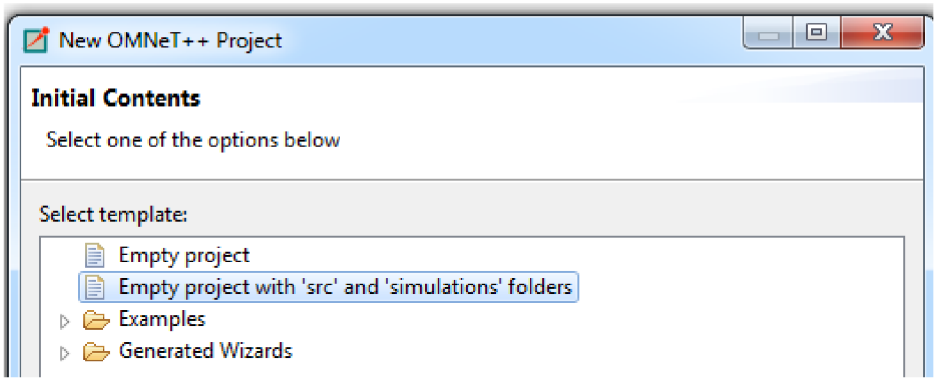
1 Spusťte OMNeT++ zadáním příkazu `omnetpp` do příkazové řádky souboru *mingwenv.cmd*, který naleznete na Ploše.

2 Vytvořte prázdný projekt následujícím způsobem:

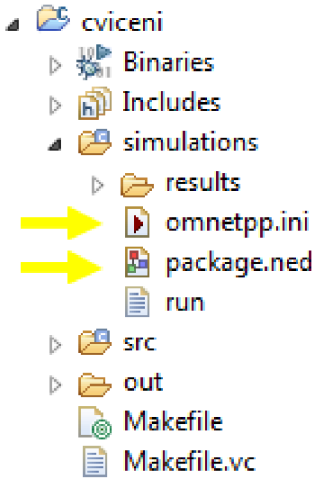
a) Přes záložku File vytvořte „OMNeT++ Project…“ a ten libovolně pojmenujte (bez mezer a diakritiky)



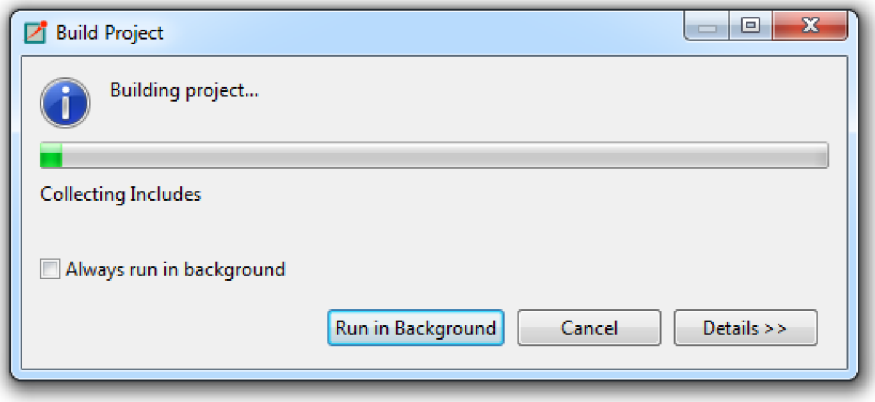

b) V dalším kroku zvolte možnost *„Empty project with ‚src‘ and ‚simulations‘ folders“*



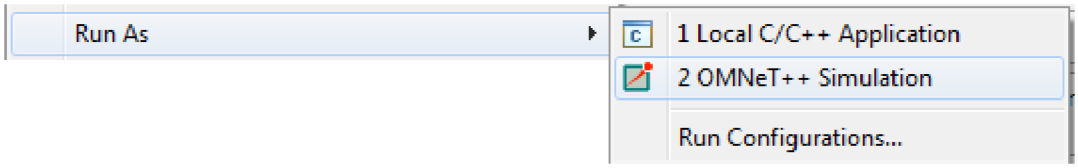

3 Vlevo v *Project Exploreru* se objevila nová složka s vaším projektem. Ze složky *Simulations* otevřete soubory *package.ned* a *omnetpp.ini* a zkopírujte do nich přiložený kód.




4 Pravou myší klikněte na vámi vytvořený soubor a poté na *„Build Project“*, čímž zahájíte kompilaci. Po prvním spuštění OMNeT++ trvá kompilace déle, další kompilace zaberou zhruba 3 vteřiny.




5 Znovu klikněte pravou myší na projekt a přes *„Run As OMNeT++ Simulation“* se dostanete do simulačního prostředí.




6 Simulaci spusťte tlačítkem a sledujte výstup zpráv v dolní části simulace

7 Ve výstupu se objeví chybová zpráva. Zjistěte její význam, prostudujte si kód konfigurace simulace a chybu opravte.

8 Znovu projekt zkompilujte a spusťte.

9 Ve výstupu se objeví další chybová zpráva. Zjistěte její význam, prostudujte si kód konfigurace simulace a chybu opravte.

10 Ve druhé části úlohy nahraďte aplikaci na jednom z hostů a na serveru VoIP aplikacemi *SimpleVoIPSender* a *SimpleVoIPReceiver*.

11 Přizpůsobte parametry VoIP aplikace tak, aby zmizela varovná hlášení a nastavte interval odesílání zpráv tak, aby byl počet zpráv obou aplikací stejného množství (přehled všech parametrů naleznete v tabu *Form* pod *Parameters*).

## 7.4 Vzorový protokol

<table>
<tr><td colspan="2" rowspan="4">VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ</td><td colspan="2">Předmět</td></tr>
<tr><td colspan="2">Jméno</td></tr>
<tr><td>Ročník</td><td>Studijní skupina</td></tr>
<tr><td>Spolupracoval</td><td>Datum</td></tr>
<tr><td colspan="2">Kontroloval</td><td>Hodnocení</td><td>Datum</td></tr>
<tr><td>Číslo úlohy<br>X</td><td colspan="3">Název úlohy<br>Optimalizace síťového provozu pomocí OMNeT++ - Nastavení aplikací</td></tr>
</table>

### Zadání

1) Vytvořte v OMNeT++ nový projekt. Pro sestavení topologie a nastavení simulace použijte konfiguraci přiloženou níže (viz příloha).
2) Projekt zkompilujte a spusťte
3) Proveďte tyto úkoly:
    a. *Sledujte probíhající zprávy. Po chvíli se objeví chybová hláška – vysvětlete ji a upravte konfiguraci tak, aby provoz běžel bezchybně (je třeba upravit/přidat dva řádky)*
    b. *UDP aplikaci na jednom z hostů nahraďte aplikací SimpleVoIPSender a na serveru aplikací SimpleVoIPReceiver. Sledujte výstup simulace. Nastavte interval odesílání zpráv tak, aby byl počet zpráv obou aplikací stejného množství.*

### Vypracování

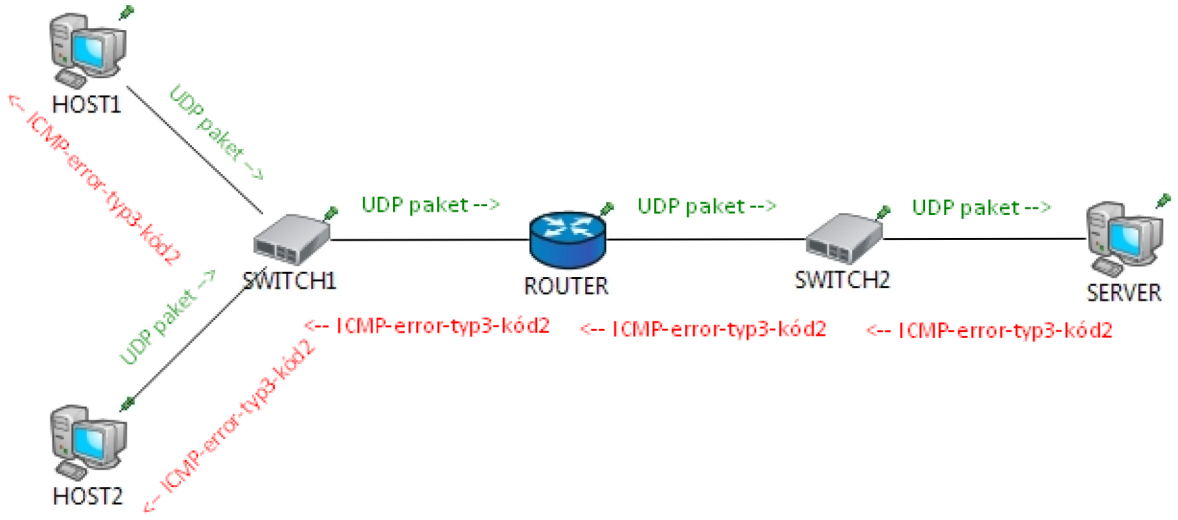
- Byl vytvořen nový projekt jako „Prázdný projekt se ‚src' a ‚simulations' složkami" a byl pojmenován jako „cviceni2".
- Do souborů *package.ned* a *omnetpp.ini* byla vložena konfigurace ze zadání.
- Projekt byl zkompilován. Kompilace proběhla v pořádku, nejsou hlášeny žádné chyby.
- Spustili jsme simulaci

*Úkol a)*

- Sledujeme probíhající provoz a po chvíli pozorujeme chybovou hlášku *ICMP-error*, konkrétně typ 3 kód 2, což znamená „Cíl nedosažitelný" + „Cílový protokol nedosažitelný".
- Když si detailněji prohlédneme výstup zpráv, zjistíme, že UDP pakety jsou odesílány směrem od hostů k serveru v pořádku, nicméně v opačném směru od serveru k hostům se posílají již jen chybové hlášky, viz následující výstup:

*Tabulka 13 Výstup simulace s ICMP errory*

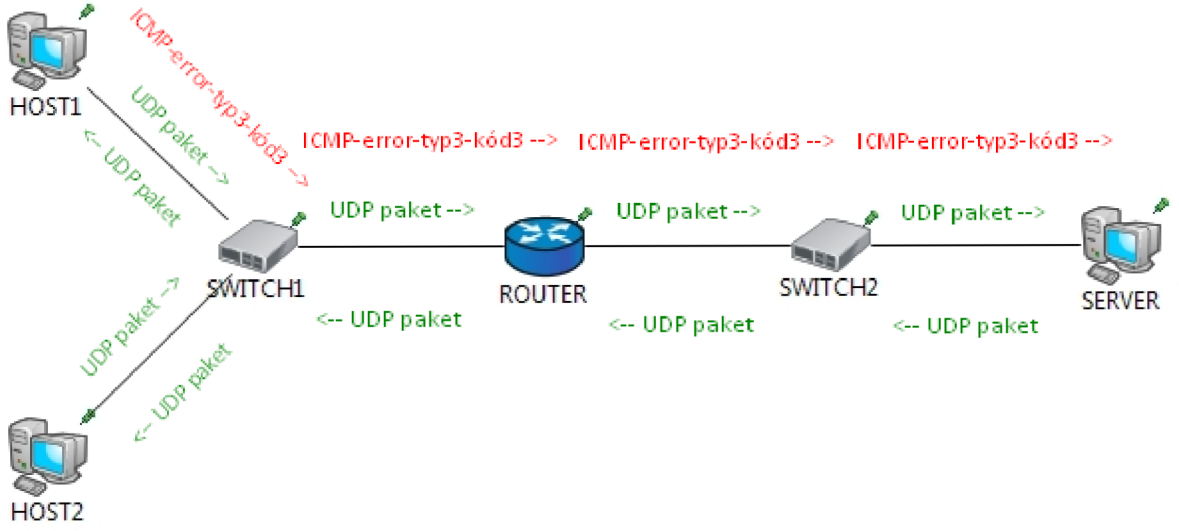
| Event# | Time | Src/Dest | Name |
|---|---|---|---|
| #84 | 1.00002344 | HOST1 --> SWITCH1 | UDPBasicAppData-0 |
| #99 | 1.000030159999 | HOST2 --> SWITCH1 | UDPBasicAppData-0 |
| #105 | 1.00003186 | SWITCH1 --> ROUTER | UDPBasicAppData-0 |
| #122 | 1.000041139999 | SWITCH1 --> ROUTER | UDPBasicAppData-0 |
| #157 | 1.00006372 | ROUTER --> SWITCH2 | UDPBasicAppData-0 |
| #164 | 1.00007214 | SWITCH2 --> SERVER | UDPBasicAppData-0 |
| #165 | 1.000072999999 | ROUTER --> SWITCH2 | UDPBasicAppData-0 |
| #174 | 1.00008056 | SERVER --> SWITCH2 | ICMP-error-#1-type3-code2 |
| #180 | 1.000081419999 | SWITCH2 --> SERVER | UDPBasicAppData-0 |
| #186 | 1.00008722 | SWITCH2 --> ROUTER | ICMP-error-#1-type3-code2 |
| #194 | 1.000089839999 | SERVER --> SWITCH2 | ICMP-error-#2-type3-code2 |
| #202 | 1.00009388 | ROUTER --> SWITCH1 | ICMP-error-#1-type3-code2 |
| #208 | 1.000096499999 | SWITCH2 --> ROUTER | ICMP-error-#2-type3-code2 |
| #214 | 1.00010054 | SWITCH1 --> HOST1 | ICMP-error-#1-type3-code2 |
| #222 | 1.000103159999 | ROUTER --> SWITCH1 | ICMP-error-#2-type3-code2 |
| #235 | 1.000109819999 | SWITCH1 --> HOST2 | ICMP-error-#2-type3-code2 |




- Je tedy očividné, že chyba bude na straně serveru, takže zkontrolujeme jeho nastavení a vidíme, že ačkoliv je na serveru specifikována aplikace, není zapnutá nastavením počtu aplikací.
- Doplníme tedy příkaz `**.SERVER.numUdpApps = 2` a znovu projekt zkompilujeme a spustíme.
- Pozorujeme takový výstup (ARP zprávy byly vyfiltrovány pro lepší přehlednost):

*Tabulka 14 Výstup simulace s ICMP errory*

| Event# | Time | Src/Dest | Name |
|---|---|---|---|
| #126 | 1.00002344 | HOST1 --> SWITCH1 | UDPBasicAppData-0 |
| #127 | 1.00002344 | SERVER --> SWITCH2 | UDPBasicAppData-0 |
| #145 | 1.000030159999 | HOST2 --> SWITCH1 | UDPBasicAppData-0 |
| #154 | 1.00003186 | SWITCH1 --> ROUTER | UDPBasicAppData-0 |
| #155 | 1.00003186 | SWITCH2 --> ROUTER | UDPBasicAppData-0 |
| #157 | 1.000032719999 | SERVER --> SWITCH2 | UDPBasicAppData-0 |
| #177 | 1.00004028 | ROUTER --> SWITCH2 | UDPBasicAppData-0 |
| #178 | 1.00004028 | ROUTER --> SWITCH1 | UDPBasicAppData-0 |
| #183 | 1.000041139999 | SWITCH1 --> ROUTER | UDPBasicAppData-0 |
| #186 | 1.000041139999 | SWITCH2 --> ROUTER | UDPBasicAppData-0 |
| #195 | 1.0000487 | SWITCH2 --> SERVER | UDPBasicAppData-0 |
| #196 | 1.0000487 | SWITCH1 --> HOST1 | UDPBasicAppData-0 |
| #209 | 1.000049559999 | ROUTER --> SWITCH2 | UDPBasicAppData-0 |
| #210 | 1.000049559999 | ROUTER --> SWITCH1 | UDPBasicAppData-0 |
| #228 | 1.00005712 | HOST1 --> SWITCH1 | ICMP-error-#1-type3-code3 |
| #238 | 1.000057979999 | SWITCH2 --> SERVER | UDPBasicAppData-0 |
| #239 | 1.000057979999 | SWITCH1 --> HOST2 | UDPBasicAppData-0 |
| #247 | 1.00006378 | SWITCH1 --> ROUTER | ICMP-error-#1-type3-code3 |
| #269 | 1.00007044 | ROUTER --> SWITCH2 | ICMP-error-#1-type3-code3 |
| #275 | 1.0000771 | SWITCH2 --> SERVER | ICMP-error-#1-type3-code3 |
| #301 | 2 | HOST1 --> SWITCH1 | UDPBasicAppData-1 |
| #302 | 2 | HOST2 --> SWITCH1 | UDPBasicAppData-1 |
| #303 | 2 | SERVER --> SWITCH2 | UDPBasicAppData-1 |
| #317 | 2.00000842 | SWITCH1 --> ROUTER | UDPBasicAppData-1 |
| #319 | 2.00000842 | SWITCH2 --> ROUTER | UDPBasicAppData-1 |
| #322 | 2.000009279999 | SERVER --> SWITCH2 | UDPBasicAppData-1 |
| #337 | 2.00001684 | ROUTER --> SWITCH2 | UDPBasicAppData-1 |
| #338 | 2.00001684 | ROUTER --> SWITCH1 | UDPBasicAppData-1 |



- Vidíme tedy, že provoz probíhá v pořádku, ale v jednom momentě opět vidíme ICMP error, tentokrát typ 3 kód 3, což znamená „Cíl nedosažitelný“ + „Cílový port nedosažitelný“, a to konkrétně ve směru od Hosta1 k Serveru.
- Zkontrolujeme tedy nastavení portů na HOST1 a SERVER a vidíme, že na serveru mají aplikace [0] i aplikace [1] nastaven *destination port* (cílový port) na stejnou hodnotu, respektive pouze na hodnotu portu druhého hosta:

```
*.SERVER.udpApp[0].destPort = 1002
*.SERVER.udpApp[1].destPort = 1002
```

- Podle *Destination address* vidíme, že aplikace [0] komunikuje s Hostem1 a aplikace [1] s Hostem2:

```
*.SERVER.udpApp[0].destAddresses = "HOST1"
*.SERVER.udpApp[1].destAddresses = "HOST2"
```

- Opravíme tedy Destination port aplikace [0] následovně:

```
*.SERVER.udpApp[0].destPort = 1001
*.SERVER.udpApp[1].destPort = 1002
```

  a projekt znovu zkompilujeme a spustíme.

- Nyní vidíme, že provoz běží v pořádku všemi určenými směry:

*Tabulka 15 Výstup simulace bez ICMP errorů*

| Event# | Time | Src/Dest | Name | Info |
|---|---|---|---|---|
| #271 | 2.0000000 | HOST1 --> SWITCH1 | UDPBasicAppData-1 | cPacket:50 bytes |
| #272 | 2.0000000 | HOST2 --> SWITCH1 | UDPBasicAppData-1 | cPacket:50 bytes |
| #273 | 2.0000000 | SERVER --> SWITCH2 | UDPBasicAppData-1 | cPacket:50 bytes |
| #287 | 2.00000842 | SWITCH1 --> ROUTER | UDPBasicAppData-1 | cPacket:50 bytes |
| #289 | 2.00000842 | SWITCH2 --> ROUTER | UDPBasicAppData-1 | cPacket:50 bytes |
| #292 | 2.0000092799 | SERVER --> SWITCH2 | UDPBasicAppData-1 | cPacket:50 bytes |
| #307 | 2.00001684 | ROUTER --> SWITCH2 | UDPBasicAppData-1 | cPacket:50 bytes |
| #308 | 2.00001684 | ROUTER --> SWITCH1 | UDPBasicAppData-1 | cPacket:50 bytes |
| #313 | 2.0000176999 | SWITCH1 --> ROUTER | UDPBasicAppData-1 | cPacket:50 bytes |
| #316 | 2.0000176999 | SWITCH2 --> ROUTER | UDPBasicAppData-1 | cPacket:50 bytes |
| #325 | 2.00002526 | SWITCH2 --> SERVER | UDPBasicAppData-1 | cPacket:50 bytes |
| #326 | 2.00002526 | SWITCH1 --> HOST1 | UDPBasicAppData-1 | cPacket:50 bytes |
| #339 | 2.0000261199 | ROUTER --> SWITCH2 | UDPBasicAppData-1 | cPacket:50 bytes |
| #340 | 2.0000261199 | ROUTER --> SWITCH1 | UDPBasicAppData-1 | cPacket:50 bytes |
| #365 | 2.0000345399 | SWITCH2 --> SERVER | UDPBasicAppData-1 | cPacket:50 bytes |
| #366 | 2.0000345399 | SWITCH1 --> HOST2 | UDPBasicAppData-1 | cPacket:50 bytes |

*Úkol b)*

Za úkol je nahradit na jednom z hostů UDP aplikaci VoIP aplikací. Vybrali jsme si aplikaci *SimpleVoIPSender* pro HOST2 a pro SERVER tím pádem připadne aplikace *SimpleVoIPReceiver*. Na serveru budou tedy aplikace dvě. [0] bude sloužit pro UDP zprávy a [1] pro VoIP. Konfigurace bude vypadat následovně:

```
[General]
network = Network
**.HOST1.numUdpApps = 1
**.HOST2.numUdpApps = 1
**.SERVER.numUdpApps = 2

*.HOST1.udpApp[0].typename="UDPBasicApp"
*.HOST1.udpApp[*].destPort = 1003
*.HOST1.udpApp[*].messageLength = 50B
*.HOST1.udpApp[*].sendInterval = 0.08s
*.HOST1.udpApp[*].localPort = 1001
*.HOST1.udpApp[*].destAddresses = "SERVER"
*.HOST1.udpApp[*].stopTime = -1s
*.HOST1.udpApp[*].timeToLive = -1
*.HOST1.udpApp[*].typeOfService = -1

*.HOST2.udpApp[0].typename="SimpleVoIPSender"
*.HOST2.udpApp[*].destPort = 1004
*.HOST2.udpApp[*].localPort = 1002
*.HOST2.udpApp[*].destAddress = "SERVER"
*.HOST2.udpApp[*].talkPacketSize = 50B
*.HOST2.udpApp[*].packetizationInterval = 0.04s
*.HOST2.udpApp[*].startTime = 0s
*.HOST2.udpApp[*].stopTime = -1s

*.SERVER.udpApp[0].typename="UDPBasicApp"
*.SERVER.udpApp[1].typename="SimpleVoIPReceiver"
*.SERVER.udpApp[0].destPort = 1001
*.SERVER.udpApp[1].destPort = 1002
*.SERVER.udpApp[*].messageLength = 50B
*.SERVER.udpApp[*].sendInterval = 0.08s
*.SERVER.udpApp[0].localPort = 1003
*.SERVER.udpApp[1].localPort = 1004
*.SERVER.udpApp[0].destAddresses = "HOST1"
*.SERVER.udpApp[1].destAddresses = "HOST2"
*.SERVER.udpApp[*].stopTime = -1s
*.SERVER.udpApp[*].timeToLive = -1
*.SERVER.udpApp[*].typeOfService = -1
```

A probíhající provoz bude vypadat takto:

*Tabulka 16 Výstup simulace s VoIP službou*

| Event# | Time | Src/Dest | Name | Info |
|---|---|---|---|---|
| #185 | 0.08001843 | ROUTER --> SWITCH1 | UDPBasicAppData-0 | cPacket:50 bytes |
| #192 | 0.08002099 | ROUTER --> SWITCH2 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #201 | 0.08002344 | HOST1 --> SWITCH1 | UDPBasicAppData-0 | cPacket:50 bytes |
| #207 | 0.08002685 | SWITCH1 --> HOST1 | UDPBasicAppData-0 | cPacket:50 bytes |
| #213 | 0.08002941 | SWITCH2 --> SERVER | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #219 | 0.08003186 | SWITCH1 --> ROUTER | UDPBasicAppData-0 | cPacket:50 bytes |
| #240 | 0.08004028 | ROUTER --> SWITCH2 | UDPBasicAppData-0 | cPacket:50 bytes |
| #246 | 0.0800487 | SWITCH2 --> SERVER | UDPBasicAppData-0 | cPacket:50 bytes |
| #260 | 0.12 | HOST2 --> SWITCH1 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #265 | 0.12000842 | SWITCH1 --> ROUTER | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #273 | 0.12001684 | ROUTER --> SWITCH2 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #279 | 0.12002526 | SWITCH2 --> SERVER | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #301 | 0.16 | HOST1 --> SWITCH1 | UDPBasicAppData-1 | cPacket:50 bytes |

| #302 | 0.16 | SERVER --> SWITCH2 | UDPBasicAppData-1 | cPacket:50 bytes |
|---|---|---|---|---|
| #303 | 0.16 | HOST2 --> SWITCH1 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #316 | 0.16000842 | SWITCH1 --> ROUTER | UDPBasicAppData-1 | cPacket:50 bytes |
| #317 | 0.16000842 | SWITCH2 --> ROUTER | UDPBasicAppData-1 | cPacket:50 bytes |
| #335 | 0.16001684 | ROUTER --> SWITCH2 | UDPBasicAppData-1 | cPacket:50 bytes |
| #336 | 0.16001684 | ROUTER --> SWITCH1 | UDPBasicAppData-1 | cPacket:50 bytes |
| #339 | 0.16001769 | SWITCH1 --> ROUTER | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #347 | 0.16002526 | SWITCH2 --> SERVER | UDPBasicAppData-1 | cPacket:50 bytes |
| #348 | 0.16002526 | SWITCH1 --> HOST1 | UDPBasicAppData-1 | cPacket:50 bytes |
| #356 | 0.16002611 | ROUTER --> SWITCH2 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #377 | 0.16003453 | SWITCH2 --> SERVER | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #391 | 0.2 | HOST2 --> SWITCH1 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #396 | 0.20000842 | SWITCH1 --> ROUTER | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #404 | 0.20001684 | ROUTER --> SWITCH2 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #410 | 0.20002526 | SWITCH2 --> SERVER | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #432 | 0.24 | HOST1 --> SWITCH1 | UDPBasicAppData-2 | cPacket:50 bytes |
| #433 | 0.24 | SERVER --> SWITCH2 | UDPBasicAppData-2 | cPacket:50 bytes |
| #434 | 0.24 | HOST2 --> SWITCH1 | VoIP | inet::SimpleVoIPPacket:50 bytes |
| #447 | 0.24000842 | SWITCH1 --> ROUTER | UDPBasicAppData-2 | cPacket:50 bytes |
| #448 | 0.24000842 | SWITCH2 --> ROUTER | UDPBasicAppData-2 | cPacket:50 bytes |
| #466 | 0.24001684 | ROUTER --> SWITCH2 | UDPBasicAppData-2 | cPacket:50 bytes |
| #467 | 0.24001684 | ROUTER --> SWITCH1 | UDPBasicAppData-2 | cPacket:50 bytes |
| #470 | 0.24001769 | SWITCH1 --> ROUTER | VoIP | inet::SimpleVoIPPacket:50 bytes |

Provoz byl nastaven tak, aby probíhalo stejné množství zpráv VoIP jako zpráv UDP (lze pozorovat u většího vzorku dat). Intervaly v odesílání zpráv byly tedy nastaveny takto:

```
*.HOST1.udpApp[*].sendInterval = 0.08s
*.HOST2.udpApp[*].packetizationInterval = 0.04s
*.SERVER.udpApp[*].sendInterval = 0.08s
```

UDP pakety mají záměrně jedenkrát tolik delší interval, protože sítí běží dvakrát – od hosta k serveru a od serveru k hostu, na rozdíl od VoIP, jehož pakety jdou jen jedním směrem. Bylo potřeba upravit parametry aplikace, a to zaměnit parametr `destAddresses` za `destAddress` a parametr `sendInterval` za `packetizationInterval`.

## Závěr

Podle návodu byl sestaven projekt. V první úloze bylo za úkol najít chyby v kódu. Chyby jsme objevili díky chybovému hlášení *ICMP-error-#1-type3-code2* a *ICMP-error-#1-type3-code3*, které jsme opravili definováním počtu aplikací na serveru a nastavením správného cílového portu na serveru.

V druhé úloze jsme zaměnili aplikaci *UDPBasicApp* za aplikace *SimpleVoIPSender* na hostu2 a *SimpleVoIPReceiver* na serveru. Upravili jsme parametry *sendInterval* a *packetizationInterval* tak, aby pakety obou aplikací byly přenášeny ve stejném množství. Pozorovali jsme VoIP pakety přenášet se od hosta 2 k serveru bez chybových hlášení.

# 8. Návod pro vyučující

## 8.1 Obecné informace

### 8.1.1 Instalace OMNeT++

OMNeT++ je pro akademické a nekomerční účely zdarma ke stažení zde: https://omnetpp.org/omnetpp. Stáhněte nejnovější verzi pro daný operační systém a rozbalte ji na disk C. Pro instalaci spusťte soubor *mingwenv* a vepište příkaz `./configure` a poté příkaz `make` (nyní bude instalace trvat zhruba 20 minut). Jakmile instalace skončí, program spustíte příkazem `omnetpp`.

### 8.1.2 Integrace INET Framework

Aby bylo možné používat předpřipravené moduly a submoduly, je potřeba doinstalovat některé simulační modely. Nejdůležitější a nejobsáhlejší z nich je INET framework. Ten nainstalujete přes *Help → Install Simulation Models*. Dále jej musíte integrovat do samotného projektu, a to kliknutím pravou myší na Váš projekt → *Properties* → *Project References* a tam vyberete „*inet*“.

### 8.1.3 Vytvoření projektu

Projekt vytvoříte pře *File* → *New* → *OMNeT++ Project…* Libovolně jej pojmenujte a uložte jej do předem nabízeného adresáře „*samples*“. Dále zvolte *Empty project with ‚src‘ and ‚simulations‘ folders* a pokračujte až do konce.

### 8.1.4 Sestrojení topologie

Grafická podoba topologie se tvoří v souboru *package.ned*. Pokud jste přidali do projektu odkaz na inet soubory, vidíte vpravo nabídku spojů a podmodulů, kterými sestavíte topologii. Do prostoru v souboru *package.ned* nejdříve vložte prázdnou síť, kterou najdete jako třetí ikonu zleva (*Network – create a network type*) pod záložkou *Types*.

### 8.1.5 Nastavení parametrů simulace

Parametry simulace se nastavují v souboru *omnetpp.ini*. Ten je tvořen takzvanými „sekcemi“, kde sekce *General* je výchozí a je použita vždy. K ní můžeme přidávat své vlastní sekce, pokud chceme například vytvořit více variant nastavení v jedné simulaci. Pro vytvoření každé simulace jsou potřeba následující kroky:

- Deklarovat síť, se kterou budeme pracovat, např.:

```
[General]
network = cviceni1
```

- Nastavit počet aplikací koncových zařízení, např.:

```
*.PC1.numPingApps = 1
```

- Nastavit koncovým zařízením příslušné aplikace, např.:

```
*.HOST.udpApp[0].typename="UDPVideoStreamCli"
```

- Nastavit parametry aplikace, které nejsou nastavené na nějakou výchozí hodnotu

Podrobný popis všech podmodulů, aplikací, spojů atd. se nachází v INET dokumentaci na stránce https://omnetpp.org/doc/inet/api-current/neddoc/index.html.

### 8.1.6 Spuštění simulace

Projekt je potřeba nejdříve zkompilovat (*„Build Project“*) a poté jej můžeme spustit (*„Run As → OMNeT++ Simulation”)*.

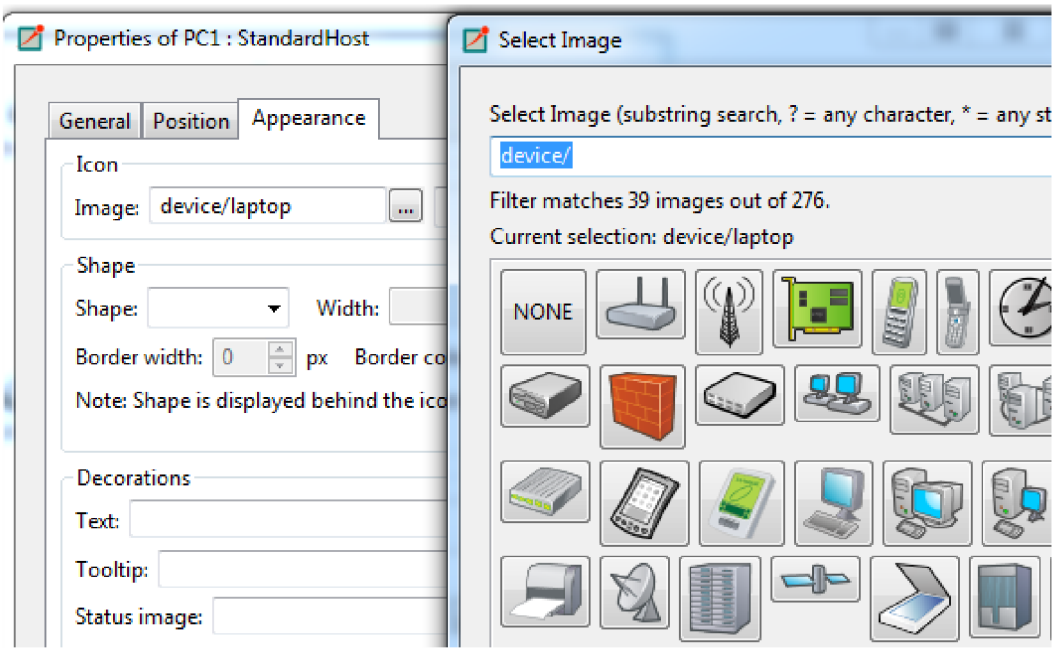
## 8.2 Komentáře k laboratorním úlohám

### 8.2.1 Laboratorní cvičení 1 – Sestavení topologie a spuštění jednoduché simulace

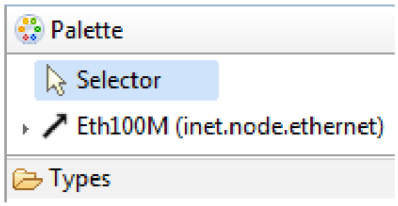
Student má za úkol sestavit topologii, kde budou tři klienti posílat serveru dotaz na jeho dostupnost pomocí ping aplikace *„PingApp“*. Topologie je sestrojena takto:

Do prázdné sítě jsou naimportovány podmoduly INET (kliknutím na ikonu a poté kliknutím do sítě):

Přes *Properties* student přiřadí každému zařízení patřičnou ikonu:




A následně zařízení propojí 100 Mbps ethernetovým kabelem:




Kód topologie by měl vypadat následně takto:

```
package cviceni1.simulations;

import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.ethernet.Eth100M;
import inet.node.ethernet.EtherSwitch;
import inet.node.inet.StandardHost;
import inet.node.ethernet.EtherLink;

network cviceni1
{
    @display("bgb=703,406,white");
    submodules:
```

```
        PC1: StandardHost {
            @display("p=111,79;i=device/laptop");
        }
        SERVER: StandardHost {
            @display("p=597,199;i=device/router");
        }
        PC2: StandardHost {
            @display("p=111,199;i=device/laptop");
        }
        PC3: StandardHost {
            @display("p=111,323");
        }
        SWITCH: EtherSwitch {
            @display("p=350,198");
        }
        configurator:
inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator {
            @display("p=262,41");
        }
    connections:
        PC1.ethg++ <--> Eth100M <--> SWITCH.ethg++;
        SWITCH.ethg++ <--> Eth100M <--> SERVER.ethg++;
        PC2.ethg++ <--> Eth100M <--> SWITCH.ethg++;
        PC3.ethg++ <--> Eth100M <--> SWITCH.ethg++;
}
```

V souboru omnetpp.ini student deklaruje síť jako `network = cviceni1`

Nastaví počet aplikací:

```
*.PC1.numPingApps = 1
*.PC2.numPingApps = 1
*.PC3.numPingApps = 1
*.SERVER.numPingApps = 3
```

Server prakticky nedělá nic jiné, než že odpovídá na ping, nicméně musí obsahovat tolik takových aplikací, s kolika hosty komunikuje, proto má v tomto případě aplikace tři.

Dále se nastaví každému zařízení aplikace a její parametry:

```
*.PC1.pingApp[*].typename="PingApp"
*.PC1.pingApp[*].destAddr="SERVER"
*.PC1.pingApp[*].count=4
*.PC1.pingApp[*].sendInterval=0.2s
*.PC1.pingApp[*].packetSize=10B

*.PC2.pingApp[*].typename="PingApp"
*.PC2.pingApp[*].destAddr="SERVER"
*.PC2.pingApp[*].count=4
*.PC2.pingApp[*].sendInterval=0.2s
*.PC2.pingApp[*].packetSize=10B

*.PC3.pingApp[*].typename="PingApp"
*.PC3.pingApp[*].destAddr="SERVER"
*.PC3.pingApp[*].count=4
*.PC3.pingApp[*].sendInterval=0.2s
*.PC3.pingApp[*].packetSize=10B

*.SERVER.pingApp[*].typename="PingApp"
*.SERVER.pingApp[0].srcAddr="PC1"
*.SERVER.pingApp[1].srcAddr="PC2"
*.SERVER.pingApp[2].srcAddr="PC3"
```

Každý parametr musí být očíslovaný, aby se vědělo, ke které aplikaci patří. Parametry PC1, PC2 a PC3 mohou být očíslované jako [0], nebo jako [*], což znamená, že daný parametr platí pro všechny aplikace u daného zařízení. Server obsahuje třikrát stejný parametr `srcAddr`, proto má každý z nich jiné identifikační číslo (vždy se začíná nulou).

Výstup simulace by měl vypadat takto:

*Tabulka 17 Výstup zpráv simulace s aplikací PingApp – návod pro vyučující*

| Event# | Time | Src/Dest | Name |
|---|---|---|---|
| **#14** | 0.109762700461 | PC1 --> SWITCH | arpREQ |
| **#19** | 0.109768510461 | SWITCH --> SERVER | arpREQ |
| **#20** | 0.109768510461 | SWITCH --> PC2 | arpREQ |
| **#21** | 0.109768510461 | SWITCH --> PC3 | arpREQ |
| **#41** | 0.109774320461 | SERVER --> SWITCH | arpREPLY |
| **#49** | 0.109780130461 | SWITCH --> PC1 | arpREPLY |
| **#58** | 0.109785940461 | PC1 --> SWITCH | ping0 |
| **#64** | 0.109791750461 | SWITCH --> SERVER | ping0 |
| **#74** | 0.109797560461 | SERVER --> SWITCH | ping0-reply |
| **#80** | 0.109803370461 | SWITCH --> PC1 | ping0-reply |
| **#95** | 0.120552674076 | PC2 --> SWITCH | arpREQ |
| **#100** | 0.120558484076 | SWITCH --> PC1 | arpREQ |
| **#101** | 0.120558484076 | SWITCH --> SERVER | arpREQ |
| **#102** | 0.120558484076 | SWITCH --> PC3 | arpREQ |
| **#122** | 0.120564294076 | SERVER --> SWITCH | arpREPLY |
| **#130** | 0.120570104076 | SWITCH --> PC2 | arpREPLY |
| **#139** | 0.120575914076 | PC2 --> SWITCH | ping0 |
| **#145** | 0.120581724076 | SWITCH --> SERVER | ping0 |
| **#155** | 0.120587534076 | SERVER --> SWITCH | ping0-reply |
| **#161** | 0.120593344076 | SWITCH --> PC2 | ping0-reply |
| **#176** | 0.17158912397 | PC3 --> SWITCH | arpREQ |
| **#181** | 0.17159493397 | SWITCH --> PC1 | arpREQ |
| **#182** | 0.17159493397 | SWITCH --> SERVER | arpREQ |
| **#183** | 0.17159493397 | SWITCH --> PC2 | arpREQ |
| **#203** | 0.17160074397 | SERVER --> SWITCH | arpREPLY |
| **#211** | 0.17160655397 | SWITCH --> PC3 | arpREPLY |
| **#220** | 0.17161236397 | PC3 --> SWITCH | ping0 |
| **#226** | 0.17161817397 | SWITCH --> SERVER | ping0 |
| **#236** | 0.17162398397 | SERVER --> SWITCH | ping0-reply |
| **#242** | 0.17162979397 | SWITCH --> PC3 | ping0-reply |
| ***<<<<<<<<<<<<<<<<<<<<<<*** | ***PING 1 a 2 vynechán*** | ***>>>>>>>>>>>>>>>>>>*** | |
| **#466** | 0.709762700461 | PC1 --> SWITCH | ping3 |
| **#471** | 0.709768510461 | SWITCH --> SERVER | ping3 |
| **#481** | 0.709774320461 | SERVER --> SWITCH | ping3-reply |
| **#487** | 0.709780130461 | SWITCH --> PC1 | ping3-reply |
| **#501** | 0.720552674076 | PC2 --> SWITCH | ping3 |
| **#506** | 0.720558484076 | SWITCH --> SERVER | ping3 |
| **#516** | 0.720564294076 | SERVER --> SWITCH | ping3-reply |

| | | | |
|---|---|---|---|
| **#522** | 0.720570104076 | SWITCH --> PC2 | ping3-reply |
| **#536** | 0.77158912397 | PC3 --> SWITCH | ping3 |
| **#541** | 0.77159493397 | SWITCH --> SERVER | ping3 |
| **#551** | 0.77160074397 | SERVER --> SWITCH | ping3-reply |
| **#557** | 0.77160655397 | SWITCH --> PC3 | ping3-reply |

### 8.2.2 Laboratorní cvičení 2 - Nastavení aplikací

Student má za úkol vytvořit projekt pomocí poskytnutého kódu. V prvním úkolu má ve výstupu simulace nalézt chyby a ty opravit. Druhým úkolem je nahradit aplikaci *UDPBasicApp* VoIP aplikací. První chyba je způsobena chybějícím příkazem `**.SERVER.numUdpApps = 2` a projevuje se chybovou hláškou `ICMP-error-#n-type3-code2`, kterou posílá server. Tato zpráva má znamenat, že cílový protokol není dosažitelný a objevuje se proto, jelikož není nastaven počet aplikací na serveru a ten je tudíž ignoruje. Stačí tedy přidat příkaz informující o počtu UDP aplikací.

Druhou chybou je `ICMP-error-#n-type3-code3` a znamená, že cílový port není dostupný. Zde je příčinou to, že je na serveru nastaven jako cílový port dvakrát port HOST2. Student tedy musí změnit hodnotu aplikace [0] z 1002 na 1001.

```
*.SERVER.udpApp[0].destPort = 1002
```

```
*.SERVER.udpApp[1].destPort = 1002
```

Ve druhé části úlohy student nahradí jednu *UDPBasicApp* aplikaci aplikací *SimpleVoIPSender* na hostu a *SimpleVoIPReceiver* na serveru (je to možné i naopak). Příkaz
`*.SERVER.udpApp[*].typename="UDPBasicApp"`
se tedy nahradí dvěma příkazy
`*.SERVER.udpApp[0].typename="UDPBasicApp"`
`*.SERVER.udpApp[1].typename="SimpleVoIPReceiver".`

Jelikož byla jedna aplikace smazána, objeví se varovné hlášení 🔒 u těchto parametrů a je potřeba je pozměnit.

```
*.HOST2.udpApp[*].destAddresses = "SERVER"
*.HOST2.udpApp[*].messageLength = 50B
*.HOST2.udpApp[*].sendInterval = 1s
```

Přehled všech parametrů pro použité aplikace je v tabu *Form* pod *Parameters* a tam musí student zjistit, jak se tyto parametry jmenují u VoIP aplikace a přejmenovat je. Je tedy třeba pouze přepsat „*destAddresses*“ na „*destAddress*“ a „*sendInterval*“ na „*packetizationInterval*“. Parametr určující délku zprávy může být smazán.
Aby probíhalo sítí stejné množství zpráv VoIP jako zpráv UDP, intervaly v odesílání zpráv musí být nastaveny takto:

```
*.HOST1.udpApp[*].sendInterval = 0.08s
```

```
*.HOST2.udpApp[*].packetizationInterval = 0.04s
```

```
*.SERVER.udpApp[*].sendInterval = 0.08s
```

UDP pakety mají záměrně jedenkrát tolik delší interval, protože sítí putují dvakrát – od hosta k serveru a od serveru k hostu, na rozdíl od VoIP, jehož pakety jdou jen jedním směrem.

# 9. Závěr

Cílem diplomové práce bylo optimalizovat návrh sítě v aplikaci OMNeT++ a představit její praktické využití. OMNeT++ s použitím nadstavby INET byl shledán jako poměrně jednoduché a intuitivní řešení, jak simulovat síťový provoz. Nejdříve byl představen postup instalace a zprovoznění programu a byly ukázány jeho hlavní části. Dále byla předvedena ukázková simulace s přenosem TCP paketů a byly rozebrány použité moduly a výstup zpráv simulace. Cílem diplomové práce bylo také představit službu QoS a implementovat ji do OMNeT++ simulace. Pro QoS byl použit model DiffServ, model Intserv byl popsán pouze teoreticky, jelikož není v OMNeT++ zcela úplně k dispozici a je již méně používanou variantou QoS. Byly navrženy alternativy aplikování QoS, a to pomocí dvou projektů, kde v prvním bylo QoS aplikováno na video zprávy, které měly být upřednostňovány před UDP zprávami, a v druhém bylo QoS aplikováno na TCP zprávy, které měly díky QoS odolávat zahlcení ICMP pakety. V druhém případě bylo porovnáváno nastavení s různými procentuálními hodnotami značkování EF, přičemž při vyšším zastoupení EF značkování docházelo ke ztrátám ICMP paketů a naopak při nižším zastoupení EF značkování bylo pozorováno zpoždění TCP zpráv.

Dále byly navrženy dvě laboratorní úlohy. V první úloze bylo cílem pro studenta sestavit a pochopit jednoduchý simulovaný provoz pomocí aplikování ping zpráv na zařízení v síti a ověření jejich vzájemné dostupnosti. Ve druhé úloze student pracuje s již připraveným kódem a má za úkol z něj vytvořit simulační projekt a dále jej analyzovat a pozměnit za zachování funkčnosti simulace. Přiloženy jsou také ukázkové protokoly a podrobnější návod pro vyučujícího. Laboratorní úlohy jsou časově nastaveny na zhruba dvě vyučovací hodiny.

Během vypracovávání diplomové práce průběžně vycházely nové verze OMNeT++, v praxi tedy byly použity celkem tři, a to verze OMNeT++ 4.6 (listopad 2014), OMNeT++ 5.0 beta 3 (prosinec 2015) a OMNeT++ 5.0 rc1 (březen 2016). Aktuální verze vyšla 15. dubna a jedná se o stabilní verzi 5.

V zadání diplomové práce bylo za úkol navrhnout tři laboratorní cvičení, nicméně po dohodě s vedoucím práce bylo naznáno, že postačují jen dvě s přihlédnutím k tomu, že druhá laboratorní úloha obsahuje dva úkoly.

# Seznam literatury


[1] THOMAS CHAMBERLAIN. *Learning OMNeT++: make realistic and insightful network simulations with OMNeT++*. New Edition. Birmingham: Packt Publ, 2013. ISBN 9781849697149.

[2] FROEHLICH, Andrew. *CVOICE 8.0: Implementing Cisco Unified Communications Voice over IP and QoS v8.0 (Exam 642-437)*. 1st. Indianapolis, Indiana: John Wiley & Sons, Inc., 2012. ISBN 978-8126533909.

[3] INET Framework for OMNeT++/OMNEST. *OMNeT++: Discrete Event Simulator* [online]. OpenSim Ltd., 2015, 2015 [cit. 2015-12-14]. Dostupné z: https://omnetpp.org/doc/inet/api-current/neddoc/index.html

[4] BURDA, CSC., Doc. Ing. Karel. *Návrh, správa a bezpečnost počítačových sítí*. Brno, 2014, 2014 [cit. 2015-12-14]. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[5] ČÍKA, P. *Multimediální služby*. Brno: FEKT VUT v Brně, 2007. 107 s.

[6] SZIGETI, Tim, HATTINGH, Christina. *End-to-End QoS Network Design : Quality of Service in LANs, WANs and VPNs* : Cisco Press, 2004. 734 s. ISBN 9781587051760

# Seznam zkratek

| | |
|---|---|
| AF | Assured Forwarding |
| ARP | Address Resolution Protocol |
| BGP | Border Gateway Protocol |
| CIR | Committed Information Rate |
| COS | Class of Service |
| CS | Class Selector |
| CSMA/CD | Carrier Sense Multiple Access with Collision Detection |
| DS | Differentiated Services |
| DSCP | Differentiated Services Code Point |
| EF | Expedited Forwarding |
| FTP | File Transfer Protocol |
| GUI | Graphic User Interface |
| HTTP | HyperText Transfer Protocol |
| ICMP | Internet Control Message Protocol |
| IETF | Internet Engineering Task Force |
| IP | Internet Protocol |
| IPv4 | Internet Protocol version 4 |
| LAN | Local Area Network |
| LGPL | Lesser General Public License |
| MAC | Media Access Control |
| OMNeT++ | Objective Modular Network Testbed in C++ |
| OSPF | Open Short Path First |
| PHB | Per-Hop Behaviors |
| PPP | Point-to-point Protocol |
| QoS | Quality of Service |
| RFC | Request for Comments |
| RIP | Routing Information Protocol |
| RSVP | Resource Reservation Protocol |
| SCTP | Stream Control Transmission Protocol |
| TCP | Transmission Control Protocol |
| ToS | Type of Service |
| TTL | Time to Live |
| UDP | User Datagram Protocol |
| VLAN | Virtual Local Area Network |
| VoIP | Voice over IP |
| WAN | Wide Area Network |
| WLAN | Wireless Local Area Network |

# Seznam příloh

Zdrojový kód pro laboratorní úlohu - NED soubor

Zdrojový kód pro laboratorní úlohu - INI soubor

Výstup zpráv simulace projektu TCP_example

Obsah CD

*Tabulka 18 Zdrojový kód pro laboratorní úlohu - NED soubor*

**NED soubor**

```
package cviceni2.simulations;

import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;// byl
importován konfigurátor IPv4 sítě
import inet.node.ethernet.EtherSwitch;// byl importován podmodul
EtherSwitch - přepínač
import inet.node.inet.Router;// byl importován podmodul Router - směrovač
import inet.node.inet.StandardHost;// byl importován podmodul
StandardHost - host
import inet.node.ethernet.EtherLink;// byl importován podmodul spoje
EtherLink - kabel
import ned.DatarateChannel;// byl importován podmodul DatarateChannel

network Network
{
    @display("bgb=731,428,white,#400040");
    types:
        channel EtherLink extends DatarateChannel
        {
            datarate = 100Mbps;// rychlost linky
            delay = 0.1us;// zpoždění
        }
    submodules:
        HOST1: StandardHost { // vytvoření instance podmodulu
StandardHost
            @display("p=103,123");// poloha v síti
        }
        HOST2: StandardHost {
            @display("p=103,327");
        }
        SERVER: StandardHost {
            @display("p=643,222");
        }
        SWITCH1: EtherSwitch {
            @display("p=209,221");
        }
        SWITCH2: EtherSwitch {
            @display("p=497,221");
        }
        ROUTER: Router {
            @display("p=353,222");
        }
        configurator: IPv4NetworkConfigurator {
            @display("p=681,29");

        }
    connections:
        HOST1.ethg++ <--> EtherLink <--> SWITCH1.ethg++;
        SWITCH1.ethg++ <--> EtherLink <--> HOST2.ethg++;
        SWITCH1.ethg++ <--> EtherLink <--> ROUTER.ethg++;
        ROUTER.ethg++ <--> EtherLink <--> SWITCH2.ethg++;
        SWITCH2.ethg++ <--> EtherLink <--> SERVER.ethg++;
}
```

*Tabulka 19 Zdrojový kód pro laboratorní úlohu - INI soubor*

**INI soubor**

```
[General]
network = Network
**.HOST1.numUdpApps = 1
*.HOST1.udpApp[0].typename="UDPBasicApp" # určení aplikace
*.HOST1.udpApp[*].destPort = 1003 # cílový port
*.HOST1.udpApp[*].messageLength = 50B # délka zprávy
*.HOST1.udpApp[*].sendInterval = 1s # interval posílání zpráv
*.HOST1.udpApp[*].localPort = 1001 # zdrojový port
*.HOST1.udpApp[*].destAddresses = "SERVER" # cílová adresa

**.HOST2.numUdpApps = 1
*.HOST2.udpApp[0].typename="UDPBasicApp"
*.HOST2.udpApp[*].destPort = 1004
*.HOST2.udpApp[*].messageLength = 50B
*.HOST2.udpApp[*].sendInterval = 1s
*.HOST2.udpApp[*].localPort = 1002
*.HOST2.udpApp[*].destAddresses = "SERVER"

*.SERVER.udpApp[*].typename="UDPBasicApp"
*.SERVER.udpApp[0].destPort = 1002
*.SERVER.udpApp[1].destPort = 1002
*.SERVER.udpApp[*].messageLength = 50B
*.SERVER.udpApp[*].sendInterval = 1s
*.SERVER.udpApp[0].localPort = 1003
*.SERVER.udpApp[1].localPort = 1004
*.SERVER.udpApp[0].destAddresses = "HOST1"
*.SERVER.udpApp[1].destAddresses = "HOST2"
```

*Tabulka 20 Výstup zpráv simulace projektu TCP_example*

| Event# | Time | Src/Dest | Name | Info |
|---|---|---|---|---|
| #15 | 1 | HOST --> SWITCH1 | arpREQ | ARP req: 10.0.0.1=? (s=10.0.0.17(0A-AA-00-00-00-0F)) |
| #20 | 1.00000586 | SWITCH1 --> ROUTER | arpREQ | ARP req: 10.0.0.1=? (s=10.0.0.17(0A-AA-00-00-00-0F)) |
| #21 | 1.00000586 | SWITCH1 --> ROUTER | arpREQ | ARP req: 10.0.0.1=? (s=10.0.0.17(0A-AA-00-00-00-0F)) |
| #22 | 1.00000586 | SWITCH1 --> HOST | arpREQ | ARP req: 10.0.0.1=? (s=10.0.0.17(0A-AA-00-00-00-0F)) |
| #44 | 1.00001172 | ROUTER --> SWITCH1 | arpREPLY | ARP reply: 10.0.0.1=0A-AA-00-00-00-05 (d=10.0.0.17(0A-AA-00-00-00-0F)) |
| #45 | 1.00001172 | ROUTER --> SWITCH1 | arpREPLY | ARP reply: 10.0.0.1=0A-AA-00-00-00-06 (d=10.0.0.17(0A-AA-00-00-00-0F)) |
| #57 | 1.00001758 | SWITCH1 --> HOST | arpREPLY | ARP reply: 10.0.0.1=0A-AA-00-00-00-05 (d=10.0.0.17(0A-AA-00-00-00-0F)) |
| #69 | 1.00002344 | HOST --> SWITCH1 | SYN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: S 250000:250000(0) win 7504 |
| #71 | 1.000024299999 | SWITCH1 --> HOST | arpREPLY | ARP reply: 10.0.0.1=0A-AA-00-00-00-06 (d=10.0.0.17(0A-AA-00-00-00-0F)) |
| #75 | 1.0000293 | SWITCH1 --> ROUTER | SYN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: S 250000:250000(0) win 7504 |
| #89 | 1.00003516 | ROUTER --> SWITCH2 | arpREQ | ARP req: 10.0.0.11=? (s=10.0.0.9(0A-AA-00-00-00-07)) |
| #95 | 1.00004102 | SWITCH2 --> ROUTER | arpREQ | ARP req: 10.0.0.11=? (s=10.0.0.9(0A-AA-00-00-00-07)) |
| #96 | 1.00004102 | SWITCH2 --> SERVER | arpREQ | ARP req: 10.0.0.11=? (s=10.0.0.9(0A-AA-00-00-00-07)) |
| #97 | 1.00004102 | SWITCH2 --> SERVER | arpREQ | ARP req: 10.0.0.11=? (s=10.0.0.9(0A-AA-00-00-00-07)) |
| #121 | 1.00004688 | ROUTER --> SWITCH2 | arpREPLY | ARP reply: 10.0.0.11=0A-AA-00-00-00-08 (d=10.0.0.9(0A-AA-00-00-00-07)) |
| #122 | 1.00004688 | SERVER --> SWITCH2 | arpREPLY | ARP reply: 10.0.0.11=0A-AA-00-00-00-0D (d=10.0.0.9(0A-AA-00-00-00-07)) |
| #123 | 1.00004688 | SERVER --> SWITCH2 | arpREPLY | ARP reply: 10.0.0.11=0A-AA-00-00-00-0E (d=10.0.0.9(0A-AA-00-00-00-07)) |
| #139 | 1.00005274 | SWITCH2 --> ROUTER | arpREPLY | ARP reply: 10.0.0.11=0A-AA-00-00-00-08 (d=10.0.0.9(0A-AA-00-00-00-07)) |
| #154 | 1.0000586 | ROUTER --> SWITCH2 | SYN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: S 250000:250000(0) win 7504 |
| #156 | 1.000059459999 | SWITCH2 --> ROUTER | arpREPLY | ARP reply: 10.0.0.11=0A-AA-00-00-00-0D (d=10.0.0.9(0A-AA-00-00-00-07)) |
| #160 | 1.00006446 | SWITCH2 --> ROUTER | SYN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: S 250000:250000(0) win 7504 |
| #167 | 1.000066179998 | SWITCH2 --> ROUTER | arpREPLY | ARP reply: 10.0.0.11=0A-AA-00-00-00-0E (d=10.0.0.9(0A-AA-00-00-00-07)) |
| #173 | 1.00007032 | ROUTER --> SWITCH2 | SYN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: S 250000:250000(0) win 7504 |
| #185 | 1.00007618 | SWITCH2 --> SERVER | SYN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: S 250000:250000(0) win 7504 |
| #195 | 1.00008204 | SERVER --> SWITCH2 | SYN+ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A S 250020:250020(0) ack 250001 win 7504 |
| #201 | 1.0000879 | SWITCH2 --> ROUTER | SYN+ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A S 250020:250020(0) ack 250001 win 7504 |

| | | | | |
|---|---|---|---|---|
| #209 | 1.00009376 | ROUTER --> SWITCH1 | SYN+ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A S 250020:250020(0) ack 250001 win 7504 |
| #215 | 1.00009962 | SWITCH1 --> HOST | SYN+ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A S 250020:250020(0) ack 250001 win 7504 |
| #227 | 1.00010548 | HOST --> SWITCH1 | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250021 win 7504 |
| #236 | 1.00011134 | SWITCH1 --> ROUTER | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250021 win 7504 |
| #237 | 1.000112199999 | HOST --> SWITCH1 | tcpseg(l=200) | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A 250001:250201(200) ack 250021 win 7504 |
| #245 | 1.0001172 | ROUTER --> SWITCH2 | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250021 win 7504 |
| #251 | 1.00012306 | SWITCH2 --> SERVER | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250021 win 7504 |
| #264 | 1.000133579999 | SWITCH1 --> ROUTER | tcpseg(l=200) | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A 250001:250201(200) ack 250021 win 7504 |
| #272 | 1.000154959999 | ROUTER --> SWITCH2 | tcpseg(l=200) | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A 250001:250201(200) ack 250021 win 7504 |
| #278 | 1.000176339999 | SWITCH2 --> SERVER | tcpseg(l=200) | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A 250001:250201(200) ack 250021 win 7504 |
| #291 | 1.000197719999 | SERVER --> SWITCH2 | ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A ack 250201 win 7504 |
| #299 | 1.000203579999 | SWITCH2 --> ROUTER | ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A ack 250201 win 7504 |
| #300 | 1.000204439998 | SERVER --> SWITCH2 | tcpseg(l=200) | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A 250021:250221(200) ack 250201 win 7504 |
| #308 | 1.000209439999 | ROUTER --> SWITCH1 | ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A ack 250201 win 7504 |
| #314 | 1.000215299999 | SWITCH1 --> HOST | ACK | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A ack 250201 win 7504 |
| #326 | 1.000225819998 | SWITCH2 --> ROUTER | tcpseg(l=200) | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A 250021:250221(200) ack 250201 win 7504 |
| #334 | 1.000247199998 | ROUTER --> SWITCH1 | tcpseg(l=200) | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A 250021:250221(200) ack 250201 win 7504 |
| #340 | 1.000268579998 | SWITCH1 --> HOST | tcpseg(l=200) | TCP: 10.0.0.11.80 > 10.0.0.17.1025: A 250021:250221(200) ack 250201 win 7504 |
| #353 | 1.000289959998 | HOST --> SWITCH1 | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250221 win 7504 |
| #361 | 1.000295819998 | SWITCH1 --> ROUTER | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250221 win 7504 |
| #362 | 1.000296679997 | HOST --> SWITCH1 | FIN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A F ack 250221 win 7504 |
| #370 | 1.000301679998 | ROUTER --> SWITCH2 | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250221 win 7504 |
| #376 | 1.000302539997 | SWITCH1 --> ROUTER | FIN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A F ack 250221 win 7504 |
| #382 | 1.000307539998 | SWITCH2 --> SERVER | ACK | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A ack 250221 win 7504 |
| #389 | 1.000308399997 | ROUTER --> SWITCH2 | FIN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A F ack 250221 win 7504 |
| #402 | 1.000314259997 | SWITCH2 --> SERVER | FIN | TCP: 10.0.0.17.1025 > 10.0.0.11.80: A F ack 250221 win 7504 |

OBSAH CD

- lab_uloha_1.txt
- lab_uloha_2.txt
- qos_tcp_vs_ping.txt
- qos_video_vs_udp.txt
- tcp_example.txt