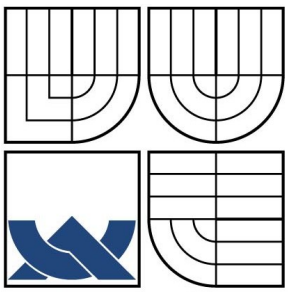
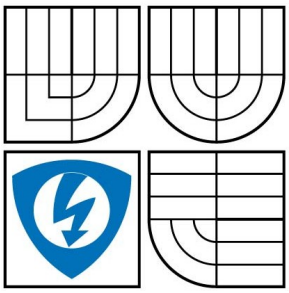


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

QOS V IP SÍTI ZA POMOCI SIMULAČNÍHO NÁSTROJE OMNET++

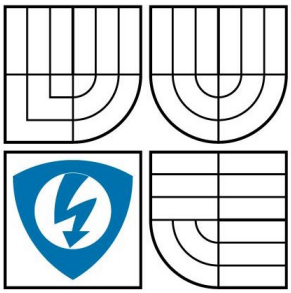
QOS IN IP NETWORK WITH THE HELP OF OMNET++ SIMULATION TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ HUDEC
AUTHOR

VEDOUCÍ PRÁCE — ING. LUKÁŠ RŮČKA
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Lukáš Hudec	***ID:***	98253
Ročník:	3	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

QoS v IP síti za pomoci simulačního nástroje OMNeT++

POKYNY PRO VYPRACOVÁNÍ:
Seznamte se se simulačním nástrojem OMNeT++ a jeho nadstavbou INET Framework. Seznamte se s možnostmi zajištění QoS v IP sítích. Prozkoumejte možnosti simulace IP sítí implementujících QoS za pomoci simulačního nástroje OMNeT++ a jeho nadstavby. Na základe teoretických poznatků realizujte dvě ukázkové laboratorní úlohy. Jedna z úloh se bude zabývat technologií DiffServ, kterou realizujte za pomocí implementace upraveného chování front. Druhá úloha bude demonstrovat možnosti a vlastnosti technologie MPLS. Výsledky simulací přehledne prezentujte a okomentujte.

DOPORUČENÁ LITERATURA:

[1] Wang, Zhendi. Internet QoS: Architectures and Mechanisms for Quality of Service. San Francisco: Morgan Kaufmann, 2001. 256 s. ISBN 1-55860-608-4.

[2] Varga, András. OMNeT++ User Manual [online]. Budapest: 2005. Dostupný z WWW: <http://www.omnetpp.org/doc/manual/usman.html>.

Termín zadání: 9.2.2009 ***Termín odevzdání:*** 2.6.2009

Vedoucí práce: Ing. Lukáš Růčka

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

UPOZORNĚNÍ:

ABSTRAKT

Obsahem práce je seznámení se se simulačním nástrojem OMNeT++ a s jeho nadstavbou INET Framework. Na toto téma je v práci realizovaná ukázková simulace IP sítě za pomoci sady modelů INET Framework.

Dále se práce zabývá rozborem možností zajištění kvality služeb v IP sítích a následnou implementací kvality služeb v prostředí simulačního nástroje OMNeT++ a jeho nadstavby INET Framework. Realizace je provedena dvěma způsoby.

První je implementace diferencovaných služeb, tzv. DiffServ, za pomoci úpravy chování výstupních front. Tato úprava spočívá v tom, že datové pakety jsou řazeny do různých front na základě jejich priority.

Druhý způsob ukazuje principy technologie MPLS a také funkci směrovacího protokolu OSPF. Technologie MPLS je realizovaná pomocí speciálních směrovačů s implementovaným modulem rezervačního protokolu.

KLÍČOVÁ SLOVA

OMNeT++, INET Framework, simulační nástroj, QoS, kvalita služeb, TCP/IP, IntServ, DiffServ, MPLS

ABSTRACT

The content of this thesis is giving information about the network simulator OMNeT++ and its superstructure INET Framework. In this thesis a sample simulation of IP network with the help of simulation package INET Framework is realized on this theme.

The thesis also deals with analysis of possibilities, how to assure quality service in IP networks and subsequently implementation quality of service in surrounding of network simulator OMNeT++ and its superstructure INET Framework. The implementation is accomplished by two methods.

The first method is implementation of differential service, in other words DiffServ, with the help of change the behavior of output. This change rest in sorting of data packets to different queues by their device priority.

The second method shows principles of the technology MPLS and also function code protocol OSPF. The MPLS technology is realized by special routers with implemented moduls reservation protocol.

KEYWORDS

OMNeT++, INET Framework, Simulation tool QoS, Quality of Service, TCP/IP, IntServ, DiffServ, MPLS

HUDEC, L. *QoS v IP síti za pomoci simulačního nástroje OMNeT++.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 66 s. Vedoucí bakalářské práce Ing. Lukáš Růčka.

PROHLÁŠENÍ

Prohlašuji, že svůj semestrální projekt na téma QoS v IP síti za pomoci simulačního nástroje OMNeT++ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Postupem času s neustálým rozvojem komunikačních sítí nastal požadavek, aby bylo možné technologie používané v telekomunikačních sítích napřed vyzkoušet a otestovat pomocí výpočetní techniky a až potom uvést do provozu. Za tímto účelem začaly vznikat nejrůznější simulační programy a to jak komerční, tak i s volnou licencí.

V komerční oblasti je to například OPNET Modeler nebo OMNEST. V oblasti volných licencí pro akademické a nekomerční používání jsou to například Network Simulator nebo OMNeT++, kterým se bude tato práce zabývat dále podrobněji.

Úkolem práce je prozkoumat možnosti simulačního nástroje OMNeT++. Jde především o možnosti simulace kvality služeb QoS (Quality of Service). Před vlastní implementací QoS do tohoto simulačního nástroje je nutné seznámit se s tímto programem, zjistit způsoby ovládání a jeho možnosti v oblasti QoS. Pro simulace IP sítí je k dispozici rozšíření INET Framework. Toto rozšíření bude popsáno v úvodu práce společně se základní architekturou a postupem instalace programu OMNeT++. Součástí tohoto tématu bude i seznámení se s ovládáním programu na jednoduchém příkladu.

V následujícím textu budou rozebrány technologie zaručující kvalitu služeb v IP sítích, jejich vlastnosti a principy, ale také možnosti implementace těchto technologií do nástroje OMNeT++.

Závěrem práce budou realizovány možnosti vlastní implementace kvality služeb do simulačního nástroje OMNeT++ s rozšířením INET Framework.

1 OMNeT++

1.1 Úvod

OMNeT++ je objektově orientovaný diskrétní simulační systém s otevřenou architekturou pro modelování komunikačních sítí. Jak již bylo řečeno v úvodu OMNeT++ je volně k dispozici pro akademické a nekomerční použití. Tomuto programu jsou věnovány internetové stránky www.omnetpp.org, kde je podrobně popsán a dá se zde také stáhnout. Odkaz pro stažení je uveden v lit. [1]. Pro komerční používání je k dispozici varianta OMNEST.

Výhodou OMNeT++ je, že jde o multiplatformní systém, což znamená, že je možné ho používat jak na platformě Linux, tak i na operačním systému Windows.

1.1.1 Vývoj

Autorem programu je András Varga, který začal s vývojem v roce 1992. První verze programu byla dokončena v létě roku 1995. Tohoto roku byl program rozšířen o možnost simulace paralelních procesů. Architektura byla nazvána SSM (Statistical Synchronization Mode).

Dalším důležitým vývojářem je Gábor Lencse, jehož zásluhou byl implementován model FDDI (Fiber Distributed Data Interface) a přidáno rozšíření jazyka NED (Network Design) pro SSM.

1.1.2 Dostupné verze programu

Práce je realizována na verzi programu OMNeT++ 3.3 kterou lze ji zdarma stáhnout na internetových stránkách [2]. Tato verze byla vydána 26.10.2006. Jde o vyzkoušenou a stabilní verzi. Dále je zde také dostupná verze OMNeT++ 4.0, která vyšla v průběhu sestavování této práce a z důvodu že již některé projekty byly realizovány pro verzi 3.3, bude i zbytek práce dokončen na stejné verzi.

1.1.3 Použití simulačního nástroje

Základní princip

Jak již bylo řečeno v úvodu, jde o diskrétní simulační systém. To znamená, že jednotlivé změny stavu celého systému jsou plánovány jako události do seznamu událostí na určitý okamžik v nespojitém simulačním čase. Mezi dvěmi naplánovanými událostmi se nepředpokládá žádná změna systému, což je rozdíl oproti simulacím v reálném spojitém čase. Během simulace jsou pak v tzv. smyčce událostí jednotlivé události ze seznamu vybírány a zpracovávány, přičemž se předpokládá, že doba zpracování události je nulová. Jako důsledek jejich zpracování se mění stav systému a generují se nové události.

Oblast použití

Simulátor OMNeT++ se používá především pro:

- modelování provozu telekomunikačních sítí
- modelování protokolů
- modelování síťových front
- modelování multiprocesorových systémů
- testování a návrh hardware
- zkoumání síťových architektur
- k hodnocení výkonových aspektů pro kompletní software systémy

Hlavním uplatněním toho simulačního programu je simulace a následný rozbor počítačových sítí.

1.1.4 Rozšíření pro OMNeT++

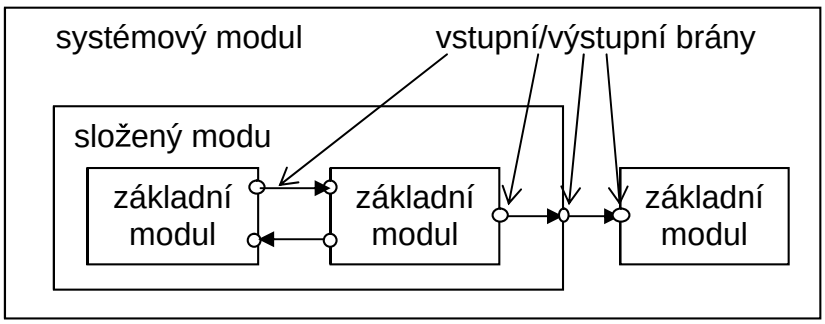
V současné době existuje několik nadstaveb (Framework), které rozšiřují použití tohoto simulačního programu. Jedná se například o sadu modulů Mobilty Framework, která rozšiřuje použití do oblasti mobilních, senzorových a bezdrátových sítí, ale hlavně o rozšíření INET Framework. Ten otevírá prostor pro simulace drátových, bezdrátových a ad-hoc sítí. Podporuje mnoho známých protokolů, jako jsou například IP (Internet Protocol), UDP (User Datagram Protocol) a TCP (Transmission Control Protocol). Obsahuje modely pro simulaci standardu 802.11, Ethernet, PPP (Point to Point Protocol) dokonce i pro IPv6 (Internet Protocol verze 6). Dále zahrnuje modely směrovacích protokolů OSPF (Open Shortest Path First) a RIP (Routing Information Protocol). Pro signalizaci je popsán protokol RSVP (ReSerVation Protocol). Obsahuje i mnoho dalších, ale těmito se již nebudeme zabývat.

1.2 Složení simulačních modulů - architektura

- Základní simulační elementy, ze kterých se skládá simulační model jsou popsány programovacím jazykem C++.
- Jednotlivé moduly se skládají z těchto základních elementů.
- Rozmístění prvků jako jsou směrovače, jejich vzájemné propojení, barva a vzhled jsou ve výsledné simulaci popsány jazykem NED.
- Grafické rozhraní je tvořeno pomocí jazyka Tcl/Tk.

Výhodou simulačního nástroje OMNeT++ je tzv. vnořování modulů. Modul na vrcholu hierarchicky sestaveného modulového stromu se nazývá systémový modul (System module). Ten se skládá z podmodulů (Submodule), které mohou být dále tvořeny dalšími podmoduly. Instancí systémového modulu je simulační model (Network), kde je v postatě popsáno výsledné složení a vzhled simulované sítě.

Moduly které jsou složeny z jednoho nebo více podmodulů se označují jako složené moduly (Compound Module). Složené moduly se skládají ze základních modulů (Simple Module). Pro lepší představu hierarchického vnořování je uveden Obr. 1.1.



Obr. 1.1: Hierarchické vnořování modulů

Moduly mezi sebou komunikují přes brány (Gates) pomocí rozesílání zpráv, které reprezentují události. Zprávy mohou mít různý formát, tedy i takový, jaký si definuje sám uživatel. Může jít tedy i o pakety, rámce a nebo o prostý tok bitů.

Moduly mohou mít různý počet vstupních a výstupních bran. Propojení (Connections) mezi jednotlivými moduly je definováno pomocí jazyka NED v definici modulů. Spoje jsou vždy definované jako jednosměrné. Zprávy odeslané z výstupní brány prvního modulu jsou přijímány vstupní branou druhého modulu. Přenosový kanál (Connections) může být ovlivněn různými vlastnostmi přenosových cest. Může zde být

definováno zpoždění, chybovost nebo přenosová rychlost. Podrobnější informace o architektuře celého systému se dají najít v literatuře [13].

1.3 Jazyk NED

Jazykem NED (Network Design) jsou definovány vstupní a výstupní brány modulů, nastaveny vlastnosti modulů a jejich složení z podmodulů nižší úrovně. Jazyk NED rozlišuje velká a malá písmena (case sensitive). Vlastnosti modulů popsané pomocí jazyka NED se ukládají do souborů s příponou .ned.

Dále si uvedeme příklady definic modulů. Kompletní popis jazyka NED je uveden v literatuře [13], konkrétně v 3. kapitole.

a) Jednoduchý modul (Simple module)

Definice se umisťuje mezi klíčová slova *simple* a *endsimple.* Příklad:

```
simple JednoduchyModul              //název modulu
   parameters:
      par1,par2,...,parN;           //definovány parametry modulu
   gates:                           //definice bran
      in:in1,in2,...,inN;           //vstupní brány
      out:out1,out2,...,outN;       //výstupní brány
endsimple                           //konec definice jednoduchého modulu
```

b) Složený modul (Compound Module)

Složený modul se zpravidla skládá z více jednoduchých modulů nebo z jiných složených modulů. Připojení jiného .ned souboru se provádí pomocí příkazu *import*. Přípona .ned se již za jménem importovaného souboru neuvádí.

Definice se uzavírá mezi klíčová slova *module* a *endmodule.* Příklad:

```
import
  „JednoduchyModul“;                //připojení modulu s názvem...
                                    //... JednoduchyModul
module SlozenyModul
   parameters:
      par1,par2,...,parN;           //definovány parametry modulu
   gates:                           //definice bran
      in:in;                        //vstupní brána
      out:out;                      //výstupní brána
   submodules
      jm:JednoduchyModul            //vytvoření instance s názvem jm
        parameters:
           par1=1                   //definice parametru par1
        connections:                //propojení bran
            jm.in1-->in
            jm.out1-->out
endmodule
```

c) Simulační model (Network)

Simulační model tvoří vrchol hierarchie simulačního stromu. Klíčová slova pro deklaraci jsou *network* a *endnetwork*. Toto označení je poněkud matoucí, ale důvodem tohoto názvu je fakt, že OMNeT++ je simulační nástroj především pro simulaci sítí. Proto zde byl zvolen název Network.

Simulační model se vytváří jako instance existujícího modelu. Simulační model nesmí obsahovat žádné vstupní ani výstupní brány a to z toho důvodu, že jde již o simulaci (např. počítačové sítě) a nedá se zde definovat jakákoliv komunikace s vnějším okolím. Ukázka definice:

```
network Simulace
   display:"p=x,y;i=image";        //nastavení vlastností zobrazení
   parameters:
      par1,par2,...,parN;
endnatwork
```

Příkazem *display* se nastavují vlastnosti zobrazení. Parametr *p* určuje pozici zobrazení objektu pomocí souřadnic x,y. Parametr *i* určuje použitý obrázek. Dá se zde určit barva a další různé vlastnosti týkající se vzhledu prvku.

d) Komunikační kanál (Channel)

Vlastnosti přenosových cest mezi dvěmi branami se přiřazují buď přímo konkrétnímu kanálu a nebo se nadefinují jednou a pak už se jen opakovaně používají.

Zde mohou být například ovlivněny parametry zpoždění, chybovost a přenosová rychlost.

Definování vlastností u konkrétního spoje:

```
...
   connection:
      out-->delay 100ms-->in       //kanál se zpožděním 100ms
...
```

Druhá možnost je definovat vlastnosti kanálu „globálně“ a potom tento kanál opakovaně používat. Příklad:

```
channel Kanal
   delay t;                  //zpoždění kanálu v sekundách
   error c;                  //chybovost kanálu z intervalu <0;1>
   datarate dr;              //datová rychlost v bitech za sekundu
endchannel
```

Přiřazení k určitému kanálu se provede následujícím způsobem:

```
...
   connection:
      out-->Kanal-->in;
...
```

1.4 Základní popis souborového složení simulace

V souborech s příponou .cc jsou popsány základní jednoduché moduly. Zprávy jsou definované v souborech s příponou .msg. Složené moduly a také celé simulační schéma je popsané v souborech s příponou .ned.

Při skládání simulace z jednotlivých modulů může být NED soubor zkompilován a přilinkován k simulátoru. Tím se urychlí celkový průběh simulace. Druhou možností je, že soubory .ned budou načítány dynamicky až v průběhu simulace. Tím dojde sice k prodloužení doby simulace, ale při změně textové podoby souboru .ned není nutné znovu kompilovat celou simulační architekturu.

Postup při překladu je následující: nejprve se pomocí skriptu *opp_msg* přeloží soubor zpráv .msg. Dále, pokud není nastaveno dynamické načítání .ned souborů, se přeloží do C++ i .ned soubory pomocí překladače *nedtool*. Následuje samotný překlad C++ programu. Soubory se tedy přeloží do objektových souborů s příponou .obj a pomocí linkování se spojí s potřebnými knihovnami a vytvoří se spustitelný program. K tomu je potřeba simulační knihovna *sim_std* (knihovna simulačního jádra), *envir*

(knihovna prostředí simulace), *cmdevn* (uživatelské textové rozhraní) a také *tkenv* (uživatelské grafické rozhraní).

cmdevn – je textové rozhraní příkazového řádku pro překlad, kompilování a spouštění simulací. Výhodou tohoto rozhraní je jeho malá náročnost na výkon počítače a rychlost simulace.

tkenv – je grafické rozhraní, které využívá skriptovací jazyk Tcl. Jazyk Tcl používá grafickou knihovnu Tk. Tento jazyk podporuje jednoduché spouštění simulace v jednoduchém grafickém prostředí. Grafické rozhraní je velice názorné a intuitivní. Je proto zvlášť vhodné pro navrhování, testování a také pochopení funkcí jednotlivých komponent.

Výstupem kompilace je jednak spustitelný soubor s příponou .exe, kterým se spouští simulace. Dalším výstupem může být soubor s naměřenými hodnotami (přípona .sca) a nebo výpis vektorů s příponou .vcc, který může sloužit k dalšímu zpracování výsledků.

Pro zobrazení těchto dvou výstupů jsou v OMNeT++ dostupné dva nástroje. Jde v podstatě o grafické zobrazení naměřených hodnot. Tyto dva nástroje se nazývají:

- **scalars** – tento nástroj je určen především pro tvorbu sloupcových a x – y grafů. Výstup se standardně provádí ze souboru s příponou .sca.
- **plove** – nástroj sloužící pro zobrazení souborů s příponou .vcc. Může zobrazovat jeden nebo více výstupů v jednom a nebo více zobrazeních. U plove výstupu můžeme specifikovat požadovaný styl zobrazení (tloušťka čáry, vlastnosti bodů, atd.). Dají se zde také nastavovat vlastnosti a popis os.
  V grafech lze výstupní výsledky upravovat průměrováním, upravováním zkreslení, atd. Na výstupy je možné aplikovat různé filtry a to nejen předdefinované ale i vlastní.

Oba tyto výstupy je možné převést do souborů s příponou .gif nebo .esp a tím umožnit zobrazení jako klasické obrázky.

1.5 Instalace OMNeT++

V této kapitole bude rozebrán postup jak správně nainstalovat simulační program OMNeT++.

Instalace bude popsána pro operační systém Windows XP Professional. Postup instalace je též popsán na internetových stránkách [3].

Ještě před vlastní instalací OMNeTu je potřeba stáhnout a nainstalovat překladač jazyka C++. Pro systémy Windows nabízí Microsoft program s názvem Visual C++. On line instalační balíček je dostupný na adrese [4]. Další pomocný program, který bude potřeba je Platform SDK. Jde o rozšíření programu Visual C++, které obsahuje balíček hlavičkových souborů. Tento program je dostupný ke stažení na adrese [5]. Jako poslední program bude potřeba stáhnout a nainstalovat program ze stránek [6] s názvem nmake. Tento program bude využíván při kompilaci. Nakonec stáhneme instalátor programu OMNeT++ ze stránek [1].

Při instalaci je lepší vyhnout se různým kritickým místům, kde by mohly nastat potíže se správným fungováním programu. Je lepší nainstalovat programu do složek v jejichž názvu nejsou mezery a nebo nestandardní znaky. Proto je vhodné zvolit umístění C:\OMNeT++.

1.5.1 Nastavení kompilačního prostředí

Pro snadnější a rychlejší práci s programem OMNeT++ je vhodné vytvořit zástupce pro spuštění příkazového řádku. Tím zajistíme, že se nám při každém spuštění tohoto zástupce načtou pomocné programy, které by jsme jinak museli složitě

spouštět při každém novém spuštění příkazového řádku. Postup vytvoření toho zástupce je následující:

V kořenovém adresáři (v mém případě C:\ ) vytvoříme soubor s názvem omnetevn.bat. Tento soubor budeme editovat například spuštěním v textovém editoru. Do souboru uložíme následující:

```
call "C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"

 SET PATH=%PATH%;C:\Program Files\Microsoft Platform SDK\Bin
 SET INCLUDE=%INCLUDE%;C:\Program Files\Microsoft Platform SDK\Include
 SET LIB=%LIB%;C:\Program Files\Microsoft Platform SDK\Lib
```

Pro ještě snadnější spuštění vytvoříme zástupce na pracovní ploše následujícím způsobem. Kliknutím na pracovní plochu pravým tlačítkem myši vyvoláme místní nabídku. Z té vybereme položku *Nový* a dále *Zástupce*. Do umístění zadáme:

```
„%comspect%/k C:\omnetevn.bat“
```

Spuštěním tohoto zástupce zavoláme skript *vcvarsall.bat*, nastavíme požadované cesty a zahrneme potřebné knihovny. Získáme tedy příkazový řádek s veškerým nastavením potřebným pro vytváření a kompilaci projektů. Tento postup je uveden v literatuře [10].

1.6 Sada modelů INET Framework

INET Framework obsahuje simulační modely specializované především pro simulaci protokolů TCP/IP. Jde o simulace drátových, bezdrátových a ad-hoc sítí, které využívají známé protokoly jako jsou například IPv4, IPv6, TCP, UDP, standardy 802.11, Ethernet, PPP, směrovací protokoly OSPF, RIP a protokoly MPLS, RSVP a další.

1.6.1 Architektura a vzájemná komunikace mezi moduly INET Framework

Filozofie simulování provozu v síti je stejná jako u OMNeT++. Moduly reprezentují síťová zařízení, jejich části si mezi sebou zasílají zprávy, které mohou reprezentovat rámce, pakety, datagramy apod., v závislosti na simulovaném protokolu. Protokoly jsou reprezentovány jednoduchými moduly, jejichž vnější rozhraní je definováno v jazyku NED a funkční část je implementována v C++ třídě stejného jména. Tyto stavební bloky lze dále libovolně kombinovat do větších celků reprezentujících síťová zařízení (*Router*, *Host*).

Síťová rozhraní (pro Ethernet, 802.11 atd.) jsou obvykle tvořené frontou (*Queue*), vrstvou MAC (Media Access Control), případně dalšími. Hlavičky protokolů reprezentují zprávy (v souborech .msg), ze kterých jsou utilitou *opp_msgc* z OMNeT++ automaticky generovány odpovídající C++ třídy (jako potomci třídy *cMessage*). Pokud protokol vyšší vrstvy chce odeslat data vygeneruje odpovídající zprávu kterou zašle modulu reprezentujícímu protokol nižší vrstvy. Tam dojde k zapouzdření obsahu zprávy do jiné, nově vytvářené, která reprezentuje datovou jednotku protokolu této vrstvy a ta je odeslána opět buď nižší vrstvě nebo již přímo partnerskému modulu. Po přijetí zprávy je proces opačný.

Ne všechny moduly ale implementují protokoly. Jsou zde moduly, které pouze obsahují statická data (*RoutingTable*), ulehčují komunikaci modulů (*NotificationBoard*), provádějí automatickou konfiguraci sítě (*FlatNetworkConfigurator*), nebo např. pohybují bezdrátovými prvky.

1.6.2 Vlastnosti INET Framework a obsah distribuce

INET obsahuje řadu simulačních ukázkových příkladů, které jsou připraveny k simulacím a jsou již překompilovány. Slouží především k obeznámení se s danou technologií a pochopení způsobu komunikace. Jako příklad se dá uvést : BulkTransfer, MulticastNetwork, ARPTest,MixedLan, atd.

V INET Framework je také mnoho předdefinovaných modulů týkajících se simulací IP sítí. Z těchto modulů se dá jednoduše sestavit potřebná simulovaná síť. Jde především o moduly směrovačů (model Router), přepínačů (model EthernetSwitch, EthernetHub) a nebo hostů (StandardHost). Pro přístup k médiu a vzájemnou komunikaci jsou popsány nižší vrstvy neboli síťová rozhraní (EthernetInterface, PPPInterface). Konkrétně pro síťovou vrstvu je zde popsán IP protokol (modul IP), ARP (Address Resolution Protocol) protokol a také třeba ICMP (Internet Control Message Protocol) protokol.

INET Framework podporuje i verzi 6 IP protokolu. Pro sestavení topologie IPv6 se využívají moduly IPv6, IPv6NeighbourDiscovery, ICMPv6 a třeba RoutingTablev6.

Pro transportní vrstvu jsou zde moduly TCP pro simulování potvrzovaného spojově orientovaného spojení a pro nepotvrzované nespojově orientované spojení je zde popsán protokol UDP.

Na výstupech zařízení jsou použity moduly front (DropTailQueue, REDQueue a DropTailQoSQueue). Posledně jmenovaný typ fronty bude využit při implementaci QoS do prostředí simulačního nástroje. Tento typ fronty upřednostňuje pakety od různých typů služeb.

V neposlední ředě jsou zde také popsány druhy zpráv. Tyto moduly jsou přizpůsobeny tak, aby svým obsahem odpovídaly například IP paketům (modul IPDatagram), TCP segmentům (TCPSegment) a nebo datovým jednotkám UDP (UDPPacket).

Díky velkému množství předdefinovaných modulů a jejich variabilitě lze snadno sestavit téměř jakoukoliv síť. Ostatní moduly, které zde nejsou uvedeny jsou popsány na internetových stránkách [9].

1.6.3 Obsah distribuce INET Frameworku

V této podkapitole jsou uvedeny jen základní složky, které získáme po nainstalování INET Frameworku.

Aplication/	Složka obsahuje různé aplikace, které pracují na aplikační vrstvě např.: PingApp, TCPApp, UDPApp, atd. V případě TCP a UDP by se mohlo zdát, že jde o protokoly transportní vrstvy. V podstatě jde ovšem o zdroje vysílání TCP a UDP dat, proto jsou zařazeny do této složky.
Base/	Jde o základní moduly, ze kterých se dále skládají složené moduly.
bin/	Systémová složka.
Documantation/	Dokumentace k modulům.
Examples/	Obsahuje připravené příklady. Např.: ethernet, IPv4, atd.
Network/	Zde jsou prvky týkajících se především datových sítí.
NetworkInterface/	Rozhraní prvků (převážně fyzická vrstva).
Nodes/	Obsahuje uzly, které se vyskytují v různých typech sítí.

1.6.4 Instalace INET Framework

Před instalací je dobré prostudovat instalační textový dokument s názvem *install*, který je součástí instalačního balíku.

Postup instalace INET Frameworku je následující:

1) Z internetových stránek [12] si stáhneme instalační archiv požadované verze INET Framework. Ten rozbalíme do zvoleného adresáře. Pro přehlednost je dobré zvolit umístění do složky s již nainstalovaným Omnetem. Cesta je tedy následující: c:\OMNeT++\INET.
2) V dalším kroku je potřeba rozšíření INET Framework zkompilovat. Zde by mohla nastat chyba. Někdo by mohl chtít rovnou spustit kompilaci příkazem *opp_nmakemake –f.* To by však nebyl správný postup. Nejdřív je potřeba zadat příkaz *makemake.cmd.* To provedeme tak, že se pomocí příkazového řádku se přepneme do složky, kde máme rozbalen archiv INET Framework. V mém případě do složky c:\OMNeT++\INET. Následně zadáme příkaz *makemake.cmd*. Tím se vytvoří *makefiles*. Následným příkazem *nmake –f makefile.vc depend* se určí závislosti cest mezi složkami. Tím se zajistí propojení mezi jednotlivými modely. Toto propojení je důležité, protože se často jednotlivé moduly skládají ze základních a ty jsou v odlišných složkách.
3) Pro vytvoření spustitelných souborů zadáme příkaz *nmake –f Makefile.vc*.
4) Funkčnost si můžeme vyzkoušet spuštěním *rundemo.bat*. Ten se nachází ve složce INET\Examples\.

Při tomto postupu můžeme narazit na problém. Ukázkové příklady mohou jít sice spustit, ale při nabíhání grafického rozhraní se může objevit zpráva o tom, že program přestal pracovat a bude ukončen. Tuto chybu lze odstranit nahrazením spustitelného souboru *inet.exe* v adresáři \inet\bin souborem se stejným názvem, který je převzat z nainstalovaného dema INET Frameworku verze 10.21.06. Po takovémto nahrazení bude program pracovat bez jakýchkoli problémů.

1.6.5 Postup práce v simulačním rozšíření INET Framework

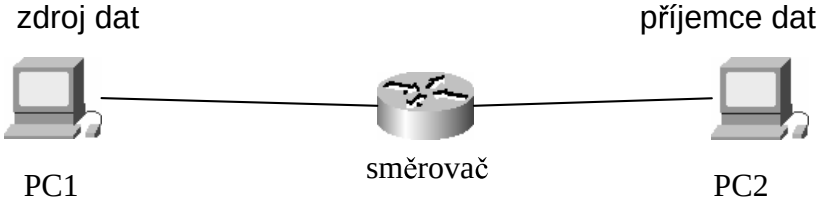
V kapitole bude rozebrán podrobný postup jak sestavit vlastní síť. Tento postup bude využit při sestavování modelů sítí s podporou QoS.

Sestavení vlastní sítě

Pro jednoduchost bude síť sestavena z následujících prvků:

- 2x uživatelský počítač PC (StandardHost) umístění: \inet\Nodes\Inet
- 1x směrovač (Router) umístění: \inet\Nodes\Inet
- 1x prvek automatického nastavení sítě (FlatNetworkConfigurator) umístění: \inet\Network\AutoRouting

V simulaci budou tedy dva počítače PC1 a PC2 (StandardHost), které budou propojeny přes směrovač (Router). Konfigurace sítě bude zajištěna pomocí modulu pro automatickou konfiguraci (FlatNetworkConfigurator). PC1 bude zdrojem vysílání pomocí transportního protokolu TCP. Komponenta PC2 bude tento datový tok přijímat a potvrzovat komponentě PC1. Schéma sítě je zobrazeno na Obr. 1.2.



Obr. 1.2: Schéma simulované sítě

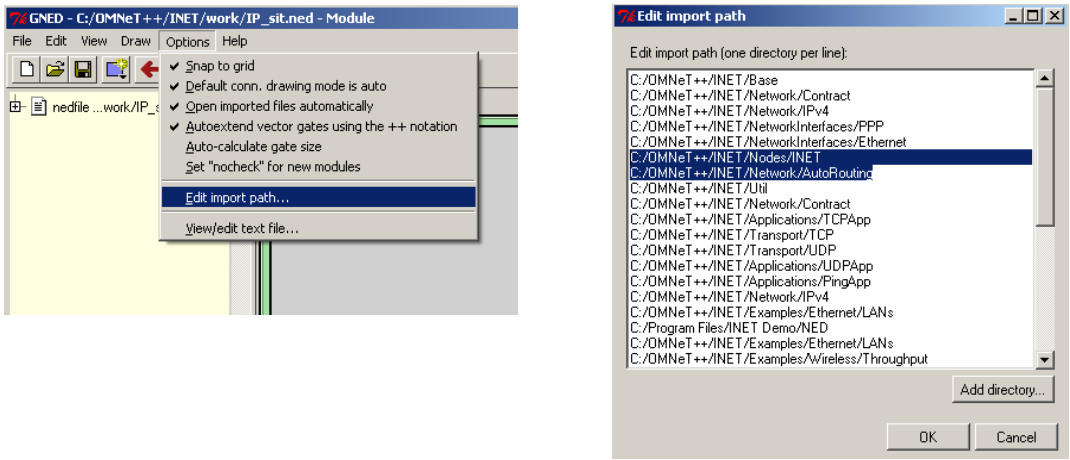
Postup sestavení simulované sítě:

Vytvoříme složku s určitým názvem, která bude reprezentovat pracovní adresář.

Spustíme grafické rozhraní pro návrh simulované sítě. Před prací je vhodné si uložit naši simulaci do vytvořené složky. Uložením vytvoříme soubor s příponou .ned například s názvem IP_sit.ned

Ještě před sestavováním je nutné nastavit cesty k simulačním modelům. Pokud si tedy přejeme simulovat síť s výše uvedenými prvky, je nutné zadat cesty kde se tyto modely nachází. Nastavení cest se provádí v menu **Options->Edit import path.** přidáme cesty, které budou potřeba pro přidání modulů. Konkrétně tedy půjde o cesty (viz Obr. 1.3):

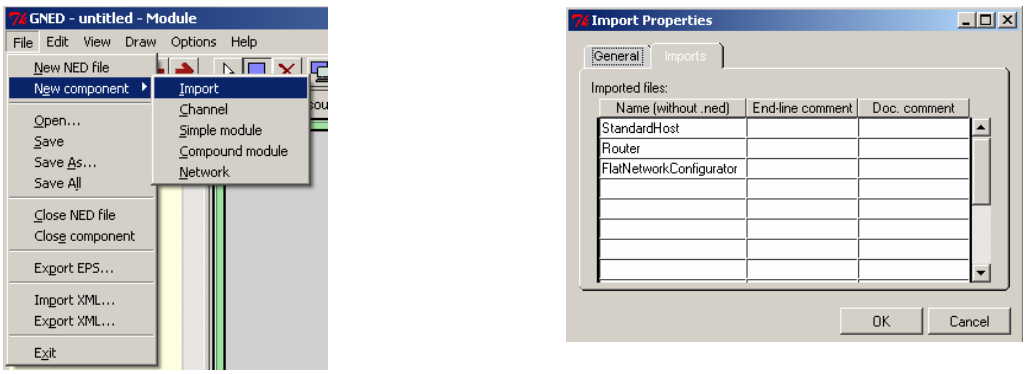
```
C:/OMNeT++/INET/Nodes/INET
C:/OMNeT++/INET/Network/AutoRouting
```



a) b)

Obr. 1.3: Přidání cest: a) přístup do Edit import path, b) zvýrazněné konkrétní cesty

V dalším kroku je důležité naimportovat moduly, které budeme používat v naší síti. Půjde o modul počítače (StandardHost), směrovače (Router) a o modul automatického nastavení sítě (FlatNetworkConfigurator). Tyto prvky se nachází ve složkách, která jsou určeny nadefinovanými cestami. Importování provedeme v menu **File->New component->Import**. Do dialogového okna Import Properties, které se nám zobrazí zadáme názvy modulů, které budeme používat. Do jména se již nezadává přípona .ned. Zadané cesty potvrdíme tlačítkem OK. Pro ilustraci je postup na Obr. 1.4.



a) b)

Obr. 1.4: Přidání modulů: a) přístup do Import Properties, b) zadání názvů importovaných modulů

Nyní je nutné celý program restartovat. Tedy vypnout a znovu zapnout, protože bez tohoto postupu se nenačtou importované moduly. Po znovu zapnutí se nám již v levé části grafického prostředí zobrazí načtené moduly (Router, StandardHost, FlatNetworkConfigurator) a také jejich podmoduly (např.: IP, TCP, UDP, EtherMAC, atd).

Nyní vytvoříme topologii, kterou chceme simulovat. V grafickém prostředí si zhotovíme podmoduly (submodule). Kliknutím na ikonu **Draw submodule and connections** (viz. Obr. 1.5) v panelu nástrojů vytvoříme požadovanou topologii. Stejnou ikonou provedeme i propojení jednotlivých podmudulů.

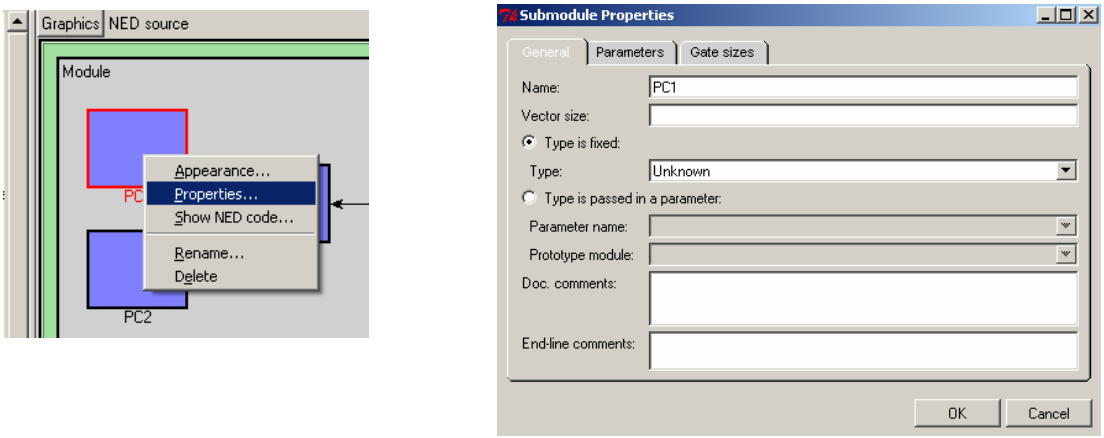
Obr. 1.5: Panel nástrojů a zvýrazněná ikona Draw submodule and connections

Přejmenování podmodulů se provádí položkou **Rename** z nabídky po stisknutí pravého tlačítka myši na příslušném prvku.

Při grafickém navrhování můžeme sledovat, jak se mění nedfile naší simulované sítě. Přepneme se do něho pomocí záložky **NED source** nad sestavenou sítí. Pro úspěšnou simulaci je nutné do **NED source** nadefinovat parametry týkající se auto konfiguračního modulu a parametry přenosových linek mezi moduly. U propojovacích linek je nutné nastavit přenosovou rychlost (data rate), protože linky jsou připojeny k fyzickým rozhraním a to vyžaduje, aby přenosová rychlost byla nastavena. Do zdrojového souboru **NED source** připíšeme následující definice:

```
        configurator: FlatNetworkConfigurator;
            parameters:
                moduleTypes = "Router StandardHost", //moduly, které bude
                                                     //konfigurátor obsluhovat
                                                     //(oddělovačem je mezera!)
                nonIPModuleTypes = "",               //moduly bez IP adresy - žádné
                networkAddress = "172.16.0.0",       //použitá IP adresa
                netmask = "255.255.0.0";             //použitá maska
    connections:
        Router.out++ --> datarate 10000 --> PC2.in++;      //nastavení
        PC2.out++ --> datarate 10000 --> Router.in++;      //přenos. rychl.
        PC1.out++ --> datarate 8000000 --> Router.in++;    //pro jednotlivé
        Router.out++ --> datarate 8000000 --> PC1.in++;    //linky
```

Nyní u jednotlivých podmodulů nadefinujeme, zda jde o směrovač, uživatelský počítač a nebo o auto konfigurační modul. Na určitém podmodulu vyvoláme lokální nabídku pravým tlačítkem myši a vybereme položku **Properties** (Obr. 1.6 a). Zobrazí se okno Submodule Properties (Obr. 1.6 b).

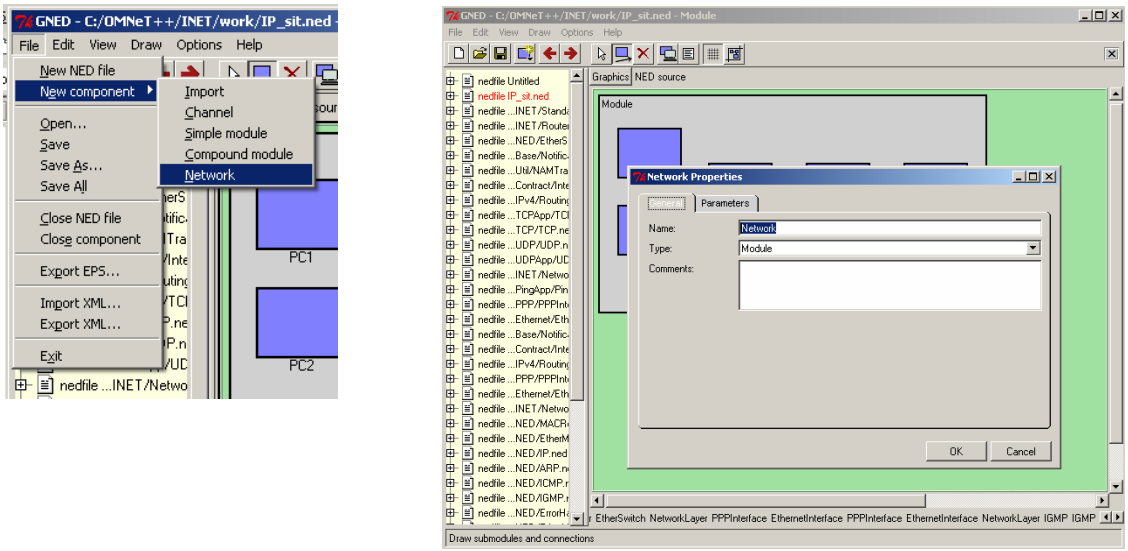


a) b)

Obr. 1.6: Nastavení vlastností podmodulu: a) přístup do Submodule Properties, b) nastavení typu

V tomto okně musíme v poli **Type** vybrat naimportovaný modul. Pro PC1 a PC2 musíme vybrat modul StandardHost, pro směrovač modul Router a pro auto konfigurační prvek modul FlatNetworkConfigurator.

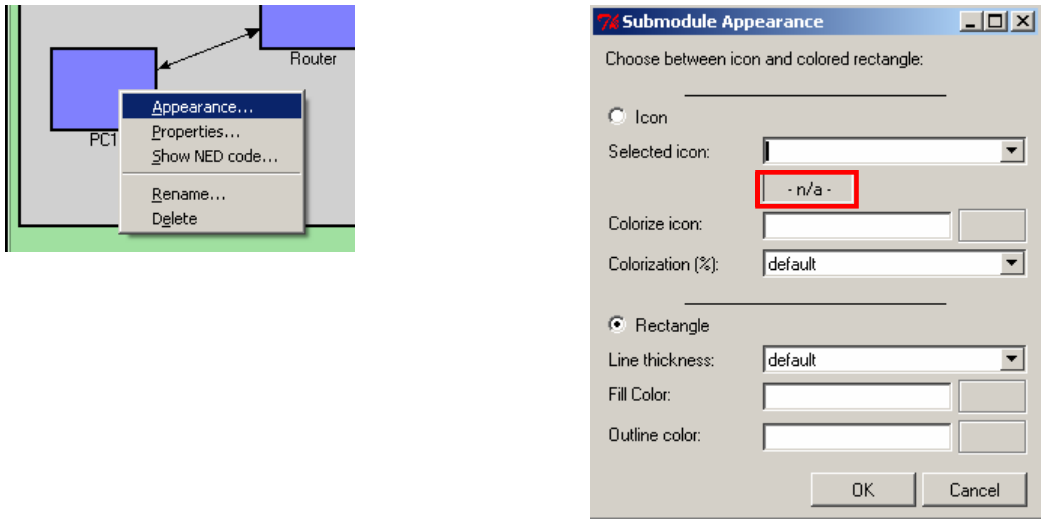
V posledním kroku je nutné překladači určit, kterou síť má simulovat. V grafickém rozhraní se to dělá tak, že přidáme novou komponentu pomocí menu **File->New component->Network** (Obr. 1.7 a). Na Obr. 1.7 b) je zobrazeno okno Network Properties. Do pole **Name** zadáme název simulované sítě a v poli **Type** vybereme modul ze kterého chceme simulovanou síť vytvořit. Protože jsme náš modul nepřejmenovali, má stále název Module.



a) b)

Obr. 1.7: Nastavení simulované sítě: a) přístup do Network Propesties, b) nastavení simulované sítě

Nakonec je ještě možné nastavit grafické ikony jednotlivým prvkům. Toto nastavení nemá samozřejmě vliv na funkčnost sítě, nicméně pro lepší přehled a grafickou podobu je to vhodné. To provedeme výběrem položky zobrazení **Appearance** v místní nabídce u určitého prvku (viz. Obr. 1.8 a). V nabídce Submodule Appearance v sekce Icon klikneme na tlačítko **-n/a-** (zvýrazněno na Obr. 1.8 b). Ve spuštěném okně vybereme vhodnou ikonu a potvrdíme tlačítkem OK. V grafickém prostředí se nám zobrazí požadované ikona místo modrého obdélníčku.



a) b)

Obr. 1.8: Přiřazení ikony: a) přístup do položky Appearance, b) nabídka Submodule Appearance

Pro kompilaci je nutné vytvořit soubor s názvem omnetpp.ini. Tímto souborem se řídí kompilace i výsledná simulace. Jsou zde nastaveny parametry pro prvky simulované sítě a také události, ke kterým při simulaci dojde.

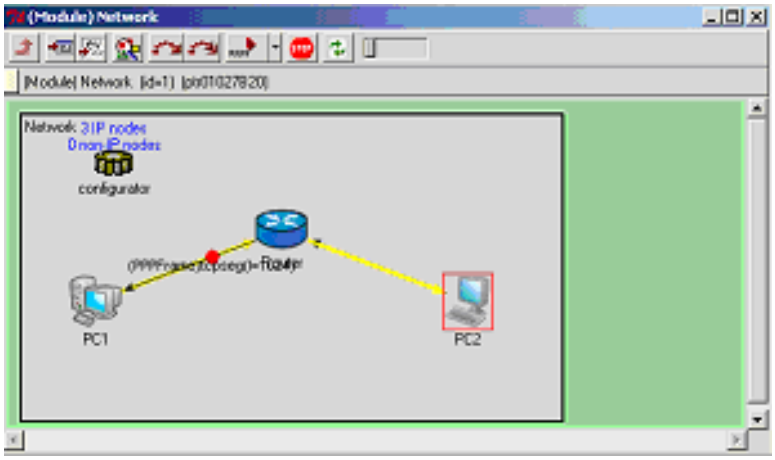
V našem pracovním adresáři si tedy vytvoříme soubor s názvem *omnetpp.ini*. Do něho uložíme zdrojový kód uvedený v příloze A. Část zdrojového kódu je převzata z lit. [22].

Nyní již můžeme náš projet zkompilovat a následně spustit. Pro spuštění bude nutné do naší složky zkopírovat spustitelný soubor s názvem **inet.exe** ze složky c:\OMNeT++\INET\bin.

Pro kompilaci musíme použít příkaz pro dynamické načítání .ned souborů. To provedeme tak, že spustíme příkazový řádek, který jsme si připravili v kapitole 1.5.1, přepneme se do naší složky a tam zadáme příkaz uvedený níže. Tím vytvoříme Makefile.vc

```
opp_nmakemake -f -N -b c:\omnet++\inet -c c:\omnet++\inet\inetconfig -I –n
```

Zadáním **inet** do příkazového řádku spustíme simulaci. Spustí se i grafické rozhraní. Tlačítkem **Run** zahájíme simulaci. Výsledek je zobrazen na Obr. 1.9.



Obr. 1.9: Průběh simulace

Výstup simulace

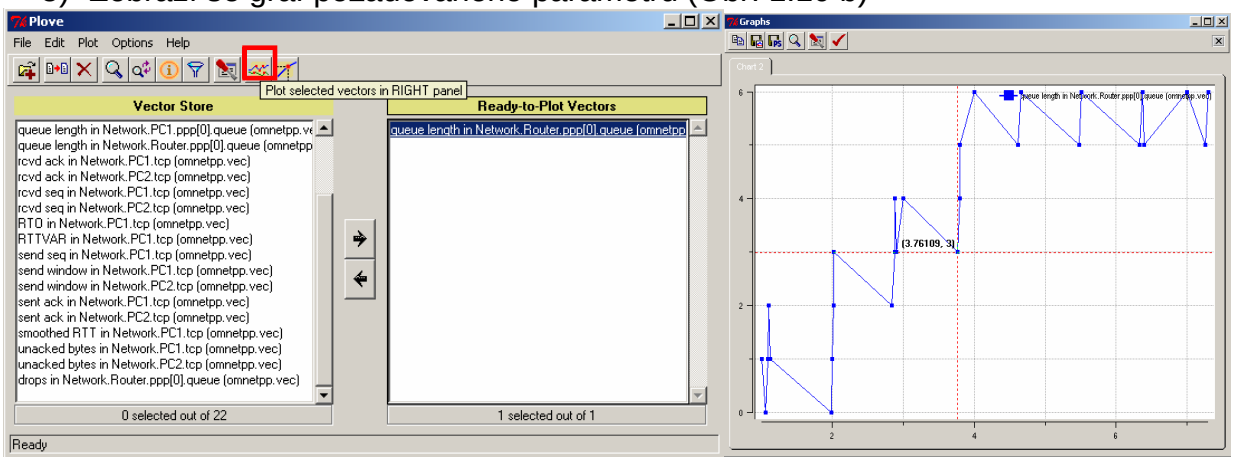
Zde jsou uvedeny možnosti zhodnocení výsledků simulace. Druhy výstupních měření jsou uvedeny v závěru kapitoly 1.4, kde jsou popsány pojmy **plove** a **scalars**.

Výstupem naší simulace jsou mimo jiné také dva soubory s příponou .vec a .sca. V těchto souborech jsou uloženy výsledky simulace. V souboru s příponou .vec lze najít výsledky týkající se zahozených paketů ve směrovači, výsledek příkazu ping, využití front jak na komponentách PC tak ve směrovači a nebo zde jde zjistit množství nedoručených potvrzení. Všechny tyto parametry lze jednoduše zobrazit pomocí grafu.

Nástroj Plove

Postup zobrazení požadovaného výstupu je následující:

1) spustíme soubor s příponou .vec. Zobrazí se nám okno Plove.
2) Vybereme který výstup chceme zobrazit do grafu.
3) Vybraný výstup přidáme do pravého pole **Ready-to-Plot Vector**.
4) Stiskneme ikonu **Plot selected vectors in RIGHT panel** (zvýrazněna na Obr. 1.10 a).
5) Zobrazí se graf požadovaného parametru (Obr. 1.10 b)



a) b)

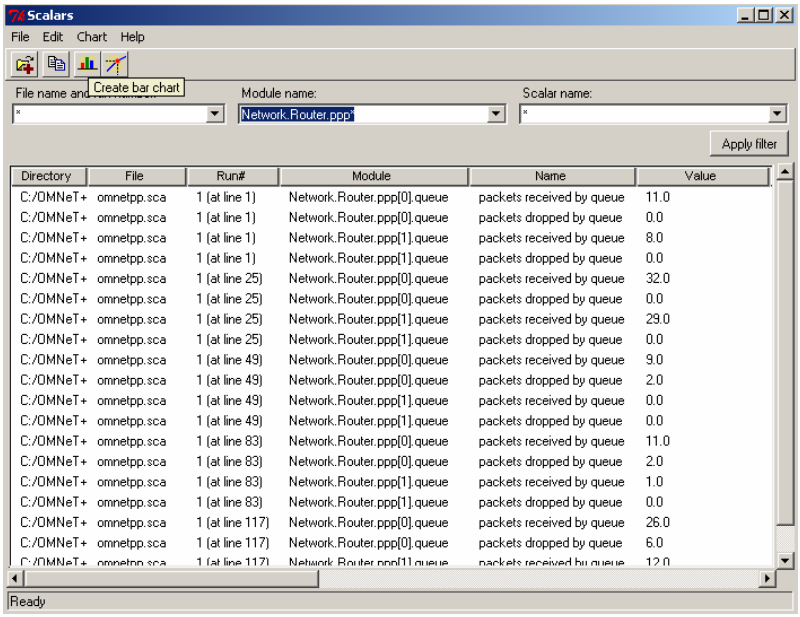
Obr. 1.10: a) Výběr proměnných pro zobrazení., b) Grafické zobrazení využití fronty ve směrovači.

V grafu je patrna závislost vytížení fronty směrovače (osa Y) na času probíhající simulace (osa X). Z průběhu je patrné, že výstupní linka ze směrovače směrem k PC2 má menší šířku pásma než linka mezi PC1 a směrovačem. Data jsou přijímána do směrovače rychleji než odesílána. Tím se ve směrovači tvoří fronta, která roste po celou dobu simulace až do doby 7,4 sekund, kdy tato fronta již nestačí a dojde k zastavení

simulace. Aby k tomuto nedocházelo, bylo by potřeba buď zvětšit frontu na směrovači (v souboru omnetpp.ini je nastavena fronta na 6 paketů) a nebo by bylo nutné zvýšit propustnost linky mezi PC2 a směrovačem (nastaveno 10 000b/s).

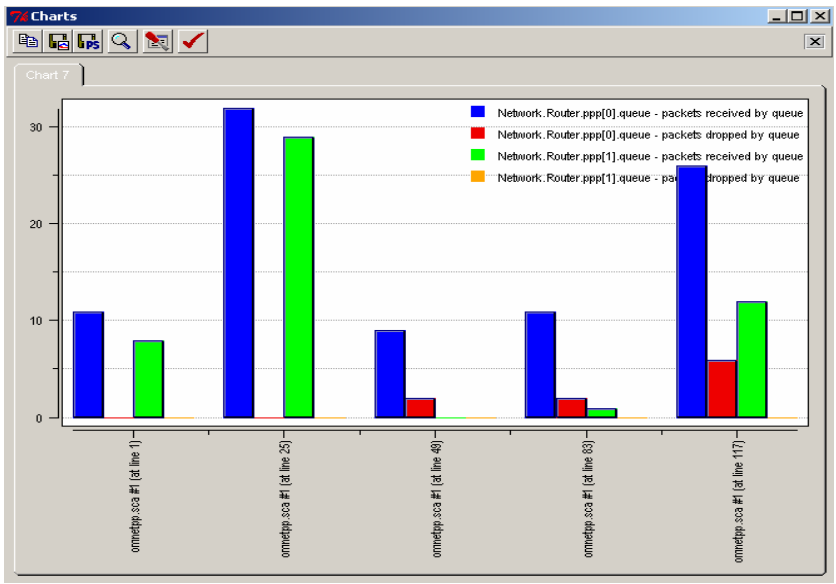
Nástroj Scalars

Dalším nástrojem na zobrazení výsledků simulace je nástroj Scalars, který slouží především pro tvorbu sloupcových grafů. Volba zobrazení požadovaných parametrů se provádí pomocí filtrů. Můžeme zde zvolit filtr podle jména souboru a čísla simulace, podle jména modulu a nebo podle názvu. Při volbě filtru se používá hvězdičkové pravidlo, kdy za znak který nechceme přímo udávat použijeme hvězdičku. Nastavení filtru je patrné z Obr. 1.11.



Obr. 1.11: Nastavení filtru podle jména modulu

Na Obr. 1.12 jsou zobrazeny fronty na směrovači (Router). Jsou zde zobrazeny fronty výstupních paketů a fronty ztracených paketů pro propojení [0] (mezi PC1 a směrovačem) a [1] (mezi směrovačem a PC2).



Obr. 1.12: Zobrazení front na směrovači

2 Kvalita služeb QoS (Quality of Service)

2.1 Úvod

Pojem kvalita služby QoS (Quality of Service) vyjadřuje trend vývoje technologií a služeb počítačových sítí poskytovat uživatelům služby s požadovanou kvalitou. Technický vývoj dává uživatelům k dispozici stále rychlejší a spolehlivější přenos dat počítačovou sítí. Přidělování kapacity sítě však zatím není řešeno systematicky. Počítačová síť se snaží vyhovět stejně všem požadavkům. Pokud to není možné, protože požadavky na přenos dat jsou v určitém okamžiku větší než kapacita počítačové sítě, jsou pakety přenášející data jednotlivých aplikací zahazována nebo pozdržována. To se projeví zpomalením přenosu, výpadkem spojení a dalšími problémy. Tento způsob se nazývá služba s maximálním úsilím (Best Effort). V současné době stále většina komunikace v Internetu probíhá tímto způsobem.

Aby síť efektivně využívala síťové prostředky z hlediska aplikací pracujících v reálném čase (Real Time), musí datům z těchto aplikací poskytovat určitou kvalitu služeb. Pojem kvalita služeb (Quality of Services) v postatě znamená, že daná komunikační síť může rozlišovat jednotlivé typy datového provozu a zacházet s nimi tak, aby splnila jejich požadavky na zpoždění, ztrátovost, chybovost a proměnlivost zpoždění. V současné době je termín kvalita služeb především skloňován v souvislosti s dnešním Internetem. Internet jako celosvětová síť je založen na protokolové sadě TCP/IP a ta sama o sobě není schopna poskytovat kvalitu služeb. To znamená, že není schopna rozlišovat jednotlivé druhy služeb.

2.2 Parametry QoS

Celkem rozlišujeme čtyři veličiny, na kterých je závislá výsledná kvalita služeb. Těmito parametry jsou ovlivňovány pakety, které putují sítí.

- **zpoždění** (Delay) – je doba potřebná k přenosu informací od odesílatele k adresátovi. Čím větší je tato doba, tím horší jsou vlastnosti výsledného spojení. Zpoždění je velice nežádoucí především při realizování telefonních hovorů. Pokud by zpoždění během hovoru vzrostlo nad mez 200ms, volající by si často skákali do řeči. U přenosu videa (Streaming) není hodnota zpoždění nejdůležitější. Důležitější je proměnlivost zpoždění. Zpoždění můžeme dále rozdělit do tří skupin:
  - **propagační zpoždění** – čas potřebný k cestě z jednoho konce sítě na druhý. Je způsobeno rychlostí šíření signálu po přenosovém médiu.
  - **paketizační zpoždění** – je čas potřebný k převodu analogového signálu na digitální a naplnění přenášeného paketu.
  - **zpoždění ukládáním do vyrovnávací paměti (buffer)** – pro zajištění konstantního zpoždění si přijímací strana ukládá přijímané datové jednotky do vyrovnávací paměti.

- **kolísání zpoždění** (Jitter) – zachycuje změnu zpoždění během přenosu. Zdroj vysílání vysílá pakety s konstantním zpožděním. Pakety putují sítí a výsledné zpoždění u jednotlivých paketů je různé, protože pakety jsou řazeny do různých front (prioritní data), zpracovávají se různou dobu (zatížení výpočetní jednotky), atd. K přijímači tedy dorazí s různým zpožděním. Změna zpoždění hraje výraznou roli především v oblasti přenosu videa.

- **ztrátovost paketů** (Loss) – v různých sítích dochází během přenosu vlivem přenosových cest ke ztrátám paketů. Tyto ztráty výrazně neovlivní přenosy hovorů

ani videa. Malé ztráty jsou na přijímací straně opraveny samoopravnými kódy, popřípadě jsou při přehrávání vynechány. Ztrátovost paketů se naopak velice projevuje při přenosu datových souborů, protože data je potřeba přenést bez jakýchkoli chyb.

- **propustnost** (Throughput) – udává objem přenesených dat za jednotku času. Video a datové přenosy vyžadují velkou propustnost, kdy požadujeme přenos velkého množství informací za co nejkratší dobu. Telefonní hovor zatěžuje síť v malé míře.

2.3 Teorie technologie QoS

V současnosti existují dva hlavní přístupy k implementaci QoS do počítačové sítě definované organizací IETF (Internet Engineering Task Force). Více informací lze nalézt v literatuře [14].

Integrované služby (Integrated Services) zkráceně IntServ definované v RFC 2210, 2211, 2212, 2215.

Rozlišované služby (Differentiated Services) zkráceně DiffServ definované v RFC 2474, 2475, 2597, 2598.

Dříve než se objevily zmíněné modely IntServ a DiffServ pro zajištění QoS na 3. vrstvě modelu OSI (Open Systems Interconnection) existovala podpora pro QoS na 2. vrstvě modelu OSI. Technologie jako ATM (Asynchronous Transfer Mode) nebo FrameRelay disponují bohatou podporou pro zajištění QoS. Podmínkami pro dosažení opravdové end-to-end podpory QoS jsou nezávislost implementace na médiu resp. na technologii 2. vrstvy OSI a vzájemné mapování (interoperabilita) mezi QoS na 2. a 3. vrstvě modelu OSI. Modely IntServ a DiffServ lze implementovat nad technologiemi ATM a FrameRelay a s výhodou přitom využít QoS podporu, která je v těchto technologiích k dispozici. U modelu DiffServ lze např. IP pakety na základě hodnoty v poli ToS (Type of Service) vyslat přes různé ATM spoje.

2.3.1 Služba s maximálním úsilím (Best Effort)

Před objasněním pojmů IntServ a DiffServ, kterými se budeme zabývat podrobněji, bude objasněn pojem Best Effort, který bude v následujících kapitolách vhodný pro názornost.

V tomto modelu posílají aplikace data kdy se jim zachce, kolik chtějí a bez vyžádání si jakéhokoliv povolení. Síťové komponenty se pokoušejí přenést data co nejlépe bez ohledu na zpoždění. Data odesílají i tehdy, pokud je nemohou doručit třeba z důvodu přetížení sítě. Příkladem takovéto služby je doručování v IP sítích.

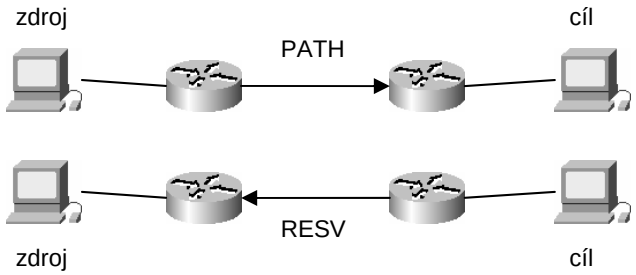
2.3.2 Integrované služby

Model Integrovaných služeb (IntServ) byl navržen v roce 1994 a je definován v dokumentu RCF1633. Je navržen pro zajištění kvality služeb (QoS) v počítačových sítích. Implementační rámec IntServ obsahuje tyto čtyři části: **plánovač paketů** (packet scheduler), **kontrolu přístupu** (admission control), **klasifikátor** (classifier) a **rezervační protokol RSVP** (ReSerVation Protocol). Tyto implementační rámce musí být definovány ve směrovačích v celé síti přes kterou chceme provozovat QoS a také v koncových zařízeních (IP telefon, video telefon, PC, atd.).

Protokol RSVP

Jde o signalizační protokol pro rezervaci síťových prostředků.

Tok dat v RSVP je sekvence zpráv, které mají tentýž zdroj a cíl. V RSVP jsou zdroje rezervovány pro data v jednom směru od vysílače do přijímače. Vysílaná data jsou označována jako upstream, přijímaná jako downstream. Hostitelský systém, který chce vysílat data s požadavkem na zajištění QoS posílá speciální paket nazývaný *PATH* k potenciálním příjemců. Tento paket nese informace o charakteristice přenosu a vytváří tzv. stav cesty podél směru přenosu. Zpráva *PATH* je přenášena z hostitelského systému do sousedního směrovače, který pošle *PATH* do následujícího směrovače v cestě a ten zase následujícímu. Cesta nebude vytvořena v případě, že se v opačném směru bude šířit zpráva *PATH Error*. Zpráva *PATH Error* se vyšle v případě, že některá část sítě není schopna zajistit požadované parametry na přenos s požadovanou kvalitou služeb. Poté co přijímač obdrží zprávu *PATH* vyšle zprávu *RESV* zpět k vysílači s typem požadované rezervace. Postup zasílání zpráv *PATH* a *RESV* je zachycen na Obr. 2.1.



Obr. 2.1: Průběh rezervace síťových prostředků RSVP protokolu

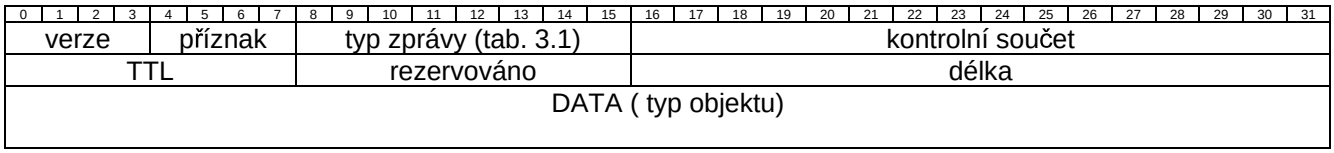
Existují čtyři typy rezervací:

- **Odlišná rezervace (Distinct Reservation):** Přijímač si přeje rezervovat část pásma pro každý vysílač.
- **Sdílená rezervace (Shared Reservation):** V tomto případě požaduje přijímač rezervovat část pásma pro všechny zdroje s danou skupinovou adresou.
- **Wildcard Filter Type:** V tomto případě požaduje přijímač rezervaci přenosové kapacity pro všechny zdroje v multicast doručovacím stromu.
- **Shared Explicit Reservation:** Přijímač požaduje rezervaci přenosové kapacity pro všechny zdroje, přičemž určí pevný počet vysílačů.

Nyní mají všechna zařízení podél cesty informace o požadavcích na přenos QoS a jsou si vědoma charakteristik přenosu potenciálního toku dat. *RESV* obsahuje aktuální QoS charakteristiky očekávané přijímačem. Různé přijímače mohou specifikovat různé požadavky na QoS pro tentýž skupinový tok dat. *RESV* se přenáší v opačném směru, než ze kterého přichází *PATH* zprávy. Tak každé zařízení podél cesty zná aktuální charakteristiky QoS pro tok dat, požadovaný přijímačem a každý usoudí samostatně, který z požadavků potvrdit a který odmítnout. Pokud je požadavek odmítnut, je poslána zpráva *RESV Error* do přijímače, který požadavek generoval. Poté co jsou *RESV* zprávy přijaty vysílačem a nebyla přijata žádná zpráva *RESV Error*, pošle zdroj zprávu *RESV Conf* do všech uzlů, které to požadují. Okamžitě poté začne vysílač posílat datové zprávy. Mezilehlá síťová směrovací zařízení posílají data s využitím rezervovaných zdrojů. Na konci přenosu pošle vysílač *PATH TEAR* zprávu, na kterou přijímač odpoví *RESV TEAR* a rezervace se zruší. Zprávy typu *TEAR* mohou být vysílány i směrovačem z důvodu vypršení časovače nebo koncovým systémem při přerušení přenosu. Přehled zpráv je v Tab. 2.1. Na Obr. 2.2 je uveden formát RSVP zprávy. Podrobnější popis viz. lit. [15].

Tab. 2.1: Typy zpráv RSVP protokolu

číslo zprávy	typ zprávy	význam
1	Path zpráva	navazování spojení
2	Resv zpráva	potvrzení a rezervace síťových prostředků
3	PathErr zpráva	chyba v PATH zprávě naposledy odeslané
4	ResvErr zpráva	chyba v poslední RESV zprávě
5	PathTear zpráva	odstranění PATH stavu ve všech směrovačích
6	ResvTear zpráva	odstranění rezervací ve všech směrovačích
7	ResvConfirm zpráva	informace o sestaveném spojení



Obr. 2.2: Formát RSVP zprávy

Jednotlivá pole naznačená na obr. 3.1 mají následující význam:
kontrolní součet – kontrola bezchybného přenosu
TTL (Time To Live) – doba platnosti dané zprávy
délka – celková délka RSVP zprávy včetně hlavičky a objektů
data – toto pole má proměnnou délku a skládá se z jednoho nebo více objektů. Přehled základních typů objektů je uveden v Tab. 2.2.

Tab. 2.2: Základní typy objektů RSVP protokolu

Objekt	Popis
NULL	je ignorován cílem
SESSION	sezení (cíl cesty datového toku)
RSVP_HOP	adresa předchozího nebo následujícího skoku
FLOWSPEC	požadovaná QoS
SENDER_TEMPLATE	identifikátor odesilatele (adresa transportní úrovně)
ADSPEC	schopnost podpory QoS na přenosové cestě
SCOPE	pro zabránění smyčkám
TIME_VALUES	obnovovací doba PATH nebo RESV zpráv

Služba s řízenou zátěží

Služba s řízenou zátěží je určená pro adaptivní aplikace v reálném čase (real-time), které jsou velmi citlivé k přetížení v síti. Tato služba umožní doručení dat ve stejných hranicích jako v případě nezatížené sítě. Služba s řízenou zátěží neakceptuje ani neužívá parametry specifikované QoS jako ztrátovost paketů, zpoždění či proměnlivost zpoždění.

Zaručená služba

Zaručená služba je služba, která poskytuje záruky šířky pásma a hranice zpoždění a je určená pro nepřizpůsobivé aplikace v reálném čase s přísnými požadavky na QoS. Zaručená služba kontroluje jen maximální zpoždění. Nehlídá minimální zpoždění a nekontroluje nebo neminimalizuje proměnlivost zpoždění.

Nevýhody Integrovaných služeb

Hlavní nevýhodou u integrovaných služeb je, že všechna zařízení přes která prochází IP pakety datových toků včetně koncových zařízení musí podporovat protokol RSVP. Každý směrovač musí udržovat stavovou informaci pro každou relaci (každý datový tok s požadavkem na zajištění určité kvality služeb), což značně zvyšuje požadavky na samotné směrovače.

Omezení u IntServ také plyne z důvodu jeho spojového charakteru. Toto omezení se týká právě rozsáhlých sítí, kde množství datových toků roste do statisíců. Právě při tomto značném nárůstu datových toků bychom narazili především na vysoké paměťové nároky směrovačů. Proto tento model nalézá především uplatnění v menších (podnikových) sítích, kde je potřeba pomocí QoS přenášet například telefonní hovory (IP telefonie). Pro odstranění těchto nevýhod byl navržen model DiffServ.

2.3.3 Diferencované služby

Jde o model diferencovaných služeb (DiffServ). Byl standardizován v doporučení RFC2475 v druhé polovině devadesátých let.

Hlavní výhodou rozlišovaných služeb je, že neoznamuje předem počítačové síti své požadavky na QoS. Použití rezervačních protokolů je možné, ale není nutné a obvykle se v tomto případě nepoužívá. Jednotlivé směrovače neudržují žádnou stavovou informaci o jednotlivých spojeních. Implementace QoS je řešena tak, že každý paket vstupující do počítačové sítě je označen značkou, která říká, jak má být s paketem zacházeno neboli určuje třídu přenosu poskytnutou paketu. Toto označení paketů probíhá pouze na vstupu do počítačové sítě. Během přenosu paketů počítačovou sítí další směrovače pouze přečtou značku každého paketu a dle této značky se řídí při zpracování paketu. Počet různých značek je relativně malý. Obvykle jednotky, maximálně desítky.

Zatímco u integrovaných služeb udržuje každý směrovač informaci o prostředcích přidělených každému jednotlivému spojení, u rozlišovaných služeb směrovače pouze přidělí určité prostředky každé třídě přenosu a zajistí určitý vztah mezi jednotlivými třídami. Například může být stanoveno, že pakety s určitou značkou mohou být poslány dále jen pokud nejsou ve frontě čekající pakety s jinou značkou.

Způsoby realizace DiffServ služeb

Při poskytování určité kvality služeb jsou mezi sítí a účastníkem dohodnuty požadované parametry. Poskytovatelé musí brát ohled především na své schopnosti realizovat smluvenou kvalitu služeb. Podrobnější informace o řazení paketů do front a jejich třídění je uvedeno v literatuře [16]. Funkční celky se dají rozdělit takto:

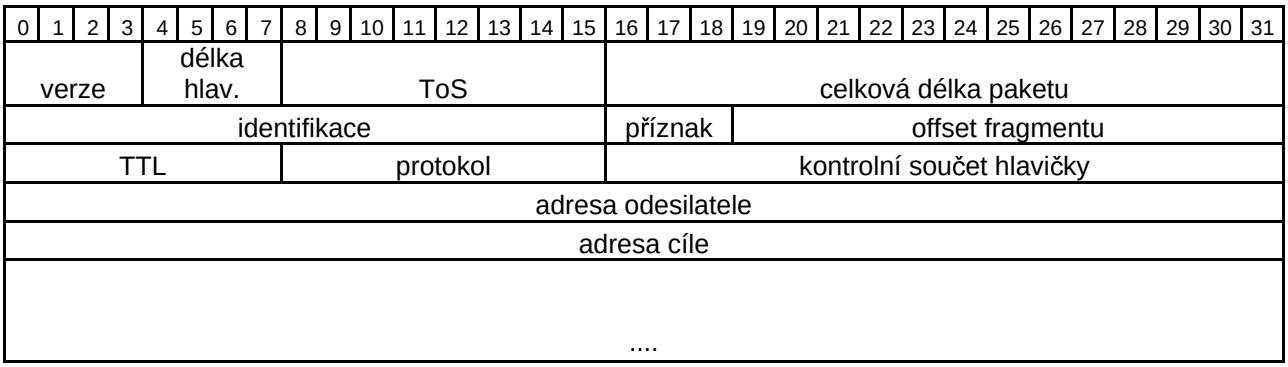
A) Označení druhu paketů v hlavičce IP paketu

Pakety se při vstupu do sítě označují značkami. Značení probíhá jak při vstupu do sítě, tak při přechodu paketu do jiné sítě s odlišnou DiffServ doménou. Značka je paketu přidělena na základě protokolu, IP adresy, čísla portu a dalších kritérií. Při přechodu do jiné DiffServ domény může být značka zachována a nebo změněna podle toho, jestli cílová DiffServ doména podporuje stejné číslování paketů.

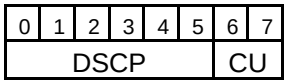
V některých případech může první značku přidělit již aplikace ze které jsou pakety posílány. Změnu značky pak může provést první směrovač v cestě.

Způsob značení paketů závisí na použitém protokolu. Nejčastěji je značka specifikující typ priority umístěna v poli typ služby ToS (Type of Service) v IP paketu. Tato značka je tedy přidávána na síťové vrstvě. V případě jiného protokolu, kde není vhodné pole pro upřesnění typu služby se typ služby připojí vně paketu.

V případě paketu IPv4 (zobrazen na Obr. 2.3) se umístí do pole ToS označení rozlišovaných služeb DS (Differentiated Services) viz Obr. 2.4, které udává, že jde o paket zahrnut do diferencovaných služeb a zadá se zde i hodnota priority.



Obr. 2.3: Hlavička IP paketu verze 4



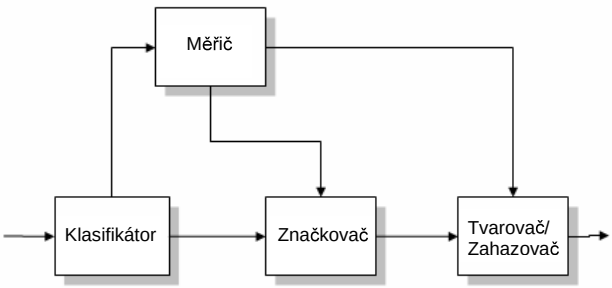
Obr. 2.4: Pole DS

B) Úprava provozu

Prvek který upravuje provoz je součást směrovače nutná k implementaci DiffServ. Úprava provozu v podstatě znamená zacházení s dopravou tak, aby data vstupující do DiffServ domény byla přizpůsobena specifikovaným pravidlům. V DiffServ je úprava provozu vykonávána v mezních uzlech (směrovače). Uzly musí mít implementovánu podporu zasílání založenou na hodnotách DSCP (Differentiated Services Code Point).

Implementované prvky musejí mít speciální vlastnosti, které zajistí třídění přicházejících dat. Data která jsou určena pro rychlejší přenos musí být zařazena před data, která nejsou náchylná na delší dobu zpoždění. Tím se zajistí jednak podpora QoS a docílí se také lepšího využití síťových prvků.

Ze služebního hlediska jsou prvky úpravy provozu jediné součásti potřebné pro funkci služby. Jak jsou spravovány a realizovány je pouze otázka správy a řízení sítě. Úprava provozu a jeho funkční části jsou zobrazené na Obr. 2.5.



Obr. 2.5: Prvek pro úpravu provozu

Klasifikátor paketů vybírá pakety z datového toku podle obsahu hlavičky paketu.
Měřiče paketů změří dočasné vlastnosti proudu paketů vybraných klasifikátorem. Měřič pošle informaci o stavu datového proudu do značkovače a tvarovače.
Značkovače paketů nastaví pole DS paketu tak, aby zařadil označený paket k určitému DiffServ režimu agregace. Značkovač může být konfigurovaný pro označení všech paketů, nebo může přeznačit pouze určité pakety podle aktuálního stavu měřiče.

Tvarovače zpožďují určité nebo všechny pakety v datovém proudu za účelem uvést tento datový proud do souladu s dopravním profilem.
Zahazovač vyřadí nějaké nebo všechny pakety v datovém proudu, aby tento proud uvedl do souladu s dopravním profilem. Tento proces je nazýván jako „hlídání“ (policing) proudu.

C) Prvky úpravy provozu

Prvky pro úpravu provozu jsou obvykle umístěné uvnitř DiffServ vstupních a výstupních mezních uzlech, ale mohou také být umístěný v uzlech uvnitř DiffServ domény nebo domény bez podpory DiffServ.

D) Dopravní profily

Dopravní profil specifikuje dočasné vlastnosti dopravního proudu vybraného klasifikátorem. To poskytuje pravidla pro určení, zda jednotlivý paket patří do profilu nebo zda-li je mimo profil.

Paketům, které jsou v profilu, může být povoleno vstoupit do DiffServ domény bez dalších úprav nebo případně může být změněno jejich DiffServ označení. Druhý případ nastane, když hodnota pole DS je nastavena na neimplicitní hodnotu, nebo když pakety vstupují do jiné DiffServ domény, která využívá rozdílnou skupinu nebo mapování.

Pakety mimo profil mohou zůstat ve frontě, dokud nejsou zformovány, vyřazeny, označeny, nebo poslány dále.

Zpracování paketů

Zpracování paketů pomocí popisové architektury DiffServ lze rozdělit na dvě úrovně abstrakce, přičemž druhá úroveň je rozdělena na dvě podúrovně.

- **Nejnižší úroveň:** Služba je zde definovaná souborem parametrů popisující vazbu mezi účastníkem a sítí. Tato vazba se označuje jako SLA (Service Level Agreemet). Příkladem této služby je pronajatý virtuální okruh.
- **Střední úroveň:** Na této úrovni je definováno, jak se s paketem zachází v konkrétním směrovači, ale bez ohledu na ostatní směrovače v cestě. Tento způsob se označuje PBH (Per Hop Behaiour). Každý směrovač se stará pouze o to, aby jeho vlastní zpracování paketu odpovídalo značkám v paketu. V současné době jsou standardizované dvě PBH:
    - **expedited forwarding (EF)** zajišťuje, že každý paket zařazen do EF PBH je odesílán rychlostí větší nebo rovnou průměrné rychlosti. Průměrná rychlost je výsledkem měření v určitých časových intervalech. EF je vhodné implementovat do virtuálního pronajatého okruhu. Zajišťuje minimální odchozí rychlost, hlídá šířku pásma.
    - **assured forwarding (AF)** umožňuje řadit pakety do jedné ze čtyř tříd. Každé třídě je ve směrovačích vyhrazen určitý objem prostředků (vyrovnávací paměť, kapacita výstupní linky, atd.) Pro případ zahlcení je zde možnost nastavení priorit. AF je vhodné použít tam, kde je vyžádána volitelná úroveň kvality přenosu. Jsou-li volné zdroje dokáže rozšířit šířku pásma.

Nevýhody diferencovaných služeb

Hlavní nevýhodou je, že podpora QoS je rozdělena pouze do omezeného množství skupin, kterým se dá přiřadit určitá priorita.

Poskytování kvality služeb je zajištěno pouze mezi dvěma sousedními směrovači (per hop) a tím nelze zajistit garance stálé kvality služeb na celé trase mezi koncovými uzly.

Další nevýhodou by mohlo v budoucnu být, že model DiffServ je orientován pouze na zdroj vysílání a nikoli na celou přenosovou síť.

2.3.4 Technologie MPLS

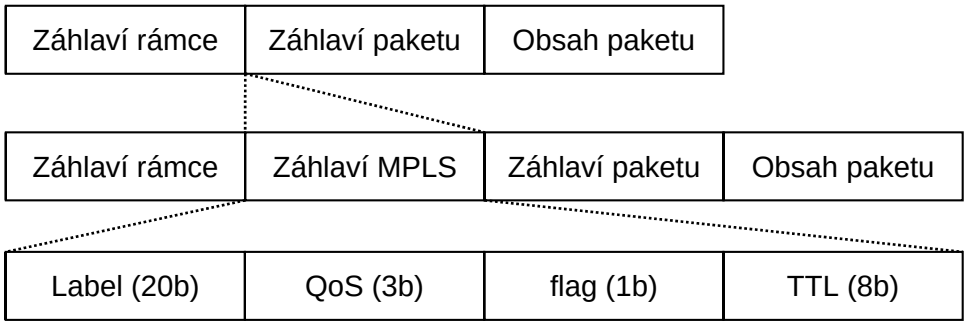
Technologie MPLS (MultiProtocol Label Switching) se využívá pro přepojování paketů. Touto technologií lze zajistit podporu QoS nejen v sítích pracujících s protokolem IP. MPLS je díky své architektuře vhodná pro vytváření virtuálních sítí VPN (Virtual Private Network).

Technologie se vzhledem k OSI modelu umisťuje mezi linkovou a síťovou vrstvu, proto se často označuje jako protokol 2,5-té vrstvy.

Princip MPLS

MPLS pracuje tak, že vkládá vlastní hlavičku před pakety. Pracuje mezi síťovou a linkovou vrstvou OSI modelu a proto svou hlavičku vkládán mezi hlavičky přidávané na těchto dvou vrstvách viz. Obr. 2.6. Toto rozdělení je popsáno v literatuře [17]. Hlavička MPLS se skládá z jednoho nebo více návěstí (Labesl). Pokud je návěstí více, řadí se za sebe formou zásobníku. Návěstí se skládá ze čtyř položek:

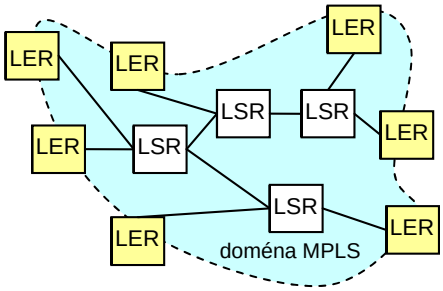
- hodnota návěstí (Label) (20b)
- pole QoS (3b)
- pole vlajky (Flag) (1b)
- životnost TTL (Time To Live) (8b)



Obr. 2.6: Vkládání hlavičky MPLS

Způsob směrování MPLS

Pakety vstupující do MPLS domény jsou na vstupu pomocí prvku LER (Label Edge Router) označeny MPLS návěstím. Prvek LER na vstupu přidává a na výstupu odebírá záhlaví MPLS. Podle tohoto záhlaví se řídí směrovače v MPLS doméně. Tím se urychluje směrování. Směrovače, které se označují LSR (Label Switch Route) se řídí pouze záhlavím MPLS. Směrování na úrovni síťové vrstvy se zde již neprovádí. Princip funkce LER a LSR je zobrazen na Obr. 2.7. Podrobnější informace lze najít v literatuře [18].



Obr. 2.7: Princip MPLS v rámci jedné domény

Paketům přicházejícím do sítě jsou přiřazovány FEC (Forward Equivalence Class). Každá FEC má své návěstí. Typ FEC závisí na IP adrese, typu dat, atd. V některých případech již může být vstupující paket opatřen značkou. Pokud tomu tak je, hraniční směrovač vloží druhé návěstí.

Technologie MPLS podporuje i ochranu před zahlcením prvku v síti. Tato technika spočívá v tom, že pokud je určitý směrovač přetížen, může požádat nejbližší předchozí LSR, aby odmítal pakety s návěstím které směřuje za něj. Tím dojde k odlehčení zátěže.

Distribuce směrovacích tabulek

Návěstí jsou mezi LER a LSR distribuovány pomocí LDP (Label Distribution Protokol) protokolu. LSR si v MPLS síti pravidelně vyměňují seznamy návěstí a dostupnost cílových stanic. Toto je prováděno pomocí standardizovaných procedur. Díky tomu si každý LSR může vytvořit představu o podobě MPLS sítě a může tak na základě těchto směrovacích informací vytvořit v MPLS síti virtuální cesty LSP (Label Switch Path). Tyto LSP jsou obdobou virtuálních okruhů u ATM. Nejsou však závislé na technologii druhé vrstvy modelu ISO/OSI.

2.3.5 Správa přenosové kapacity podsítí SBM

SBM (Subnet Bandwidth Management) je technologie, která se zaměřuje na řízení toku dat a zajištění QoS v prostředí sítí LAN. Technologie je orientována na protokoly linkové vrstvy (2. vrstva) ISO/OSI modelu. Využívá se tedy pro podporu QoS mezi směrovači. Pro jednoduchost jsou definovány pouze dva typy front. První typ fronty je pro data, která nejsou náchylná na zpoždění a druhý typ pro data, která vyžadují lepší parametry. V literatuře [19] je uveden detailnější popis technologie SBM.

Rámce na vstupu se označí podle požadavků na kvalitu služeb. V protokolu SBM jsou definovány způsoby jakými se označují vstupující data a také způsob, který umožňuje komunikaci a koordinaci mezi síťovými uzly a přepínači.

Přepínače, které podporují protokol SBM obsahují následující prvky:

- **přidělovač přenosové kapacity** BA (Bandwidth Allocator) ten udržuje přehled o rezervovaných prostředcích sítě, provádí kontrolu dostupnosti prostředků při žádosti o novou rezervaci a zná topologii sítě.

- **vyžadující modul** RM (Requestor Module) je umístěn v každé koncové stanici. Podle požadavků prvek RM překládá požadavky QoS třetí vrstvy na úrovně priorit.

Kompletní popis protokolu SBM lze nalézt v doporučení RFC2814. Tento zdroj je uveden v literatuře [20].

3 Základy směrování

V simulační úloze, která se bude týkat MPLS bude využito i dynamické směrování. Konkrétně bude využit dynamický směrovací protokol OSPF a proto je zde ve zkratce uveden princip jeho funkce.

3.1 Obecný popis směrování

Každý směrovač v síti má tzv. směrovací tabulku. Ta slouží k nalezení cíle. Paket přijde na vstup směrovače, směrovač musí určit kam paket směřuje a podle toho ho umístit na příslušné rozhraní. K tomu aby určil na které rozhraní musí daný paket umístit použije směrovací tabulku. Směrovací tabulka může být vytvářena staticky nebo dynamicky. Více viz lit. [23].

3.1.1 Statické směrování

Při statickém směrování zůstává obsah směrovací tabulky stále stejný. Tato směrovací tabulka je nastavena pevně a případné změny musí být provedeny ručně. Nevýhody jsou jasné z popisu: při každé změně v síti je nutné ručně upravit směrovací tabulku.

3.1.2 Dynamické směrování

Dynamické směrování je označované také jako adaptivní, průběžně reaguje na změny v topologii sítě a těmto změnám přizpůsobuje směrovací tabulku. Protokoly pro dynamické směrovaní jsou uvedeny v tab. 3.1 a rozdělují se ze dvou hledisek:

1) **Podle určování metriky**
   distance vector – metrika se určí nejčastěji jako počet přeskoků
   link state – protokoly berou v úvahu stav linky (propustnost, zatížení, cenu, atd.). Znají podobu celé sítě.

2) **Podle způsobu práce s maskou**
   classfull – protokoly nepracují s maskou a proto je možná pouze práce s classfull adresami.
   classless – protokoly rozlišují i masky sítí a tím pádem je možné i beztřídní adresování.

Tab. 3.1: Přehled směrovacích protokolů

Druh směrovacího protokolu	Distance Vector	Link State
Classfull	RIP, IGRP	
Classless	RIPv2, EIGRP	OSPFv2, IS-IS
IPv6	RIPng, EIGRP pro IPv6	OSPFv3, IS-IS pro IPv6

3.2 Protokol OSPF (Open Shortest Path First)

OSPF je dynamický směrovací protokol využívaný v IP sítích. Jeho vývoj byl započat v roce 1988, kdy už nedostačoval doposud používaný směrovací protokol RIP (Routing Information Protocol). Ten nedokázal pracovat v rozsáhlejších sítích které obsahovali řádově desítky směrovačů. To je dáno tím, že směrovací protokol RIP využívá pro měření nejkratší cesty (Metrika) počet přeskoků. Jeden přeskok odpovídá průchodu paketu jedním směrovačem.

OSPF jako metriku využívá cenu linky a proto se řadí do skupiny směrovacích protokolů Link State Routing Protocol. Cena linky je číslo v rozmezí 1 až 65 535, které

je přiřazeno každému rozhraní. Čím je hodnota nižší, tím je cesta více upřednostňována. Standardně je cena odvozena z šířky pásma přenosové linky, která je připojena na dané rozhraní. V případě že chceme preferovat některou z linek, můžeme cenu nastavit na jakékoli číslo z daného rozsahu.

3.2.1 Postup práce OSPF protokolu

1) V prvním kroku směrovač vysílá Hello pakety. Úkolem těchto paketů je ujednání sousedství. Směrovače se pomocí nich domluví na společných parametrech a komunikačních vlastnostech.

2) Následně si směrovače mezi sebou vysílají pakety LSA (Link State Advertisement). Struktura je na Obr. 3.1. V těchto paketech jsou obsaženy informace o rozhraních daného směrovače a také seznam směrovačů připojených k dané síti.

3) LSA pakety si směrovače ukládají do vlastní databáze a zároveň je přeposílají dál na ostatní přilehlé směrovače. Pakety se postupně rozšíří mezi celou síť. Výsledkem bude stejná topologická databáze pro všechny směrovače.

4) Následuje nalezení nejkratší cesty do každé známé sítě. Proto každý směrovač provede výpočet SPF algoritmu.

5) Nalezené nejkratší cesty se uloží do směrovací tabulky. Struktura je podrobněji popsaná v literatuře [24].

Pokud se změní topologie sítě musí směrovač, na kterém ke změně, došlo odeslat znovu paket LSA. Zpráva se bude postupně šířit celou sítí a každý směrovač znovu provede výpočet SPF.

Z důvodu, že výpočet SPF algoritmu představuje pro směrovač značnou zátěž je důležité, aby neprobíhal často. Zdrojem takového stavu by mohla být nestabilní linka. Z toho důvodu se definuje minimální časový interval mezi dvěma výpočty SPF.

1	1	2	4	4	2	2	8	pro-měnné
Verze	Typ	Délka	Označení směrovače	Označení oblasti	Kontrolní součet	Typ ověření	Ověření	Data

Obr. 3.1: Struktura OSPF paketu

Význam polí paketu:

Verze – určuje verzi použitého OSPF

Typ – identifikuje typ paketu

- Hello – ujednává sousedství
- Database description – zpráva je vyměňována při inicializaci směrovače. Doplňuje databázi topologie.
- Link-state request – směrovač si vyžádá záznamy ze sousedního směrovače. Zpráva je zasílána při objevení sousedního směrovače.
- Link-state update – paket se zasílá jako odpověď na paket Links-state request. Požívá se pro šíření LSA zpráv. Může obsahovat i více než jednu LSA zprávu.
- Link-state acknowledgement – potvrzení paketu link-state update

Délka – dálka paketu včetně hlavičky v bajtech,

Označení směrovače – identifikuje směrovač který je zdrojem paketu.

Označení oblasti – označuje oblast, pro kterou je paket určen.

Kontrolní součet – pro zjištění chyb během přenosu.
Typ ověření – specifikuje typ ověření pro každou oblast.
Ověření – obsahuje ověřovací informace.
Data – obsahuje zapouzdřené informace.

3.2.2 Ověření (Authentication)

Vyžaduje se pro zamezení neoprávněného zásahu do OSPF paketů. Případný útočník by mohl způsobit změny ve směrovacích tabulkách směrovačů a omezit tak provoz. Mohl by i zajistit přeposílání paketů uživatelů sítě na vlastní počítač.

OSPF umožňuje autentizaci pomocí hesla. Metoda je jednoduchá, ale není příliš bezpečná.

Druhou možností je kryptografická autentizace. Na každý směrovač se zadá klíč a identifikátor. Z obsahu paketu, klíče a identifikátoru se vypočítá **hash** pomocí algoritmu md5 a ten se připojí k paketu. Díky tomu, že se klíč nepřenáší přes síť je metoda bezpečnější.

4 Implementace kvality služeb v prostředí OMNeT++ s nadstavbou INET Framework

V kapitole budou rozebrány konkrétní způsoby implementace kvality služeb (QoS) v simulačním nástroji OMNeT++ a INET Framework.

Implementace na základě diferencovaných služeb bude rozebrána v kapitole 4.1. Půjde o implementaci za pomoci modulu DropTailQoSQueue, který je součástí sady modulů INET Framework.

V následující kapitole 4.2 bude objasněna technologie MPLS. V modelu bude implementován i směrovací protokol OSPF.

Další možnosti simulování již nejsou v OMNeT++ ani v nadstavbě INET Framework popsány. Pro podporu QoS na principu technologie IntServ by však šla použít sada modulů, která podporuje rezervační protokol RSVP. Sestavená síť z těchto modulů pak dokáže zajišťovat kvalitu služeb s odlišnou prioritou založenou na technologii IntServ.

Posledním způsobem podpory kvality služeb by mohla být úprava existujícího modelu. Protože jde o otevřený simulační program, ve kterém se dá vytvořit vlastní zařízení pro podporu QoS. Takovým řešením by byla například úprava modulu přepínače (EtherSwitch) tak, aby podporoval technologii SBM (technologie založená na druhé vrstvě OSI/ISO modelu).

4.1 Simulace QoS za pomoci DiffServ

Pro simulování QoS v sítích je vhodné vytvořit simulaci na základě diferencovaných služeb (DiffServ) pomocí nadstavby INET Framework. INET Framework obsahuje základní strukturu, která může být s výhodou použita pro simulaci sítí se zaručením požadované kvality a to především způsobem diferencovaných služeb.

Teoretický popis simulace:

Pro způsob zajištění kvality sužeb pomocí DiffServ se využije klasický modul směrovače (Router). S tímto modulem jsme pracovali v kapitole 1.6.5, kde jsme sestavili vlastní síť. Pro simulování podpory QoS bude zapotřebí upravit rozhraní směrovače. Půjde především o úpravu výstupních front, protože model DiffServ je založen na třídění paketů do různých tříd, kterým odpovídají výstupní fronty. Podrobnější vysvětlení architektury DiffServ je uvedeno v kapitole 2.3.3. Budeme se tedy především zabývat frontami s názvem PPPInterface, EthernetInterface, atd.

Síťová rozhraní se skládají z front a implementují se do linkové vrstvy L2 (PPP, Ethernet, atd.). Typ front je definován parametrem **queueType** síťového rozhraní. Tomuto typu můžeme přiřadit např. DropTailQueue, DropTailQoSQueue nebo třeba REDQueue. Vždy však můžeme přiřadit pouze jeden typ fronty. Pro simulaci kvality služeb samozřejmě využijeme frontu **DropTailQoSQueue**, která realizuje řazení podle priority.

```
virtual int numQueues();                        //počet front
virtual int classifyPacket(cMessage *msg);      //index podfront
```

První metoda je určena pro nastavení počtu dílčích front. Druhá metoda je volána pro každý paket a vrací index podfronty (subqueue).

Nový klasifikátor třídy musí být zaregistrován v C++ kódu, v opačném případě nebude modul DropTailQoSQueue nalezen.

```
  Register_Class(classname);                    //registrace klasifikátoru
```

Výsledné konfigurace se uloží do souboru *omnetpp.ini*. Jde o konfiguraci PPP rozhraní. Do souboru omnetpp.ini tedy zapíšeme následující:

```
**.ppp[*].queueType = "DropTailQoSQueue"
**.ppp[*].queue.classifierClass = "BasicDSCPClassifier"
**.ppp[*].queue.frameCapacity = 20
```

Samozřejmě lze upravovat zdrojové kódy popsané v jazyce C++ nebo vytvořit vlastní pro svou potřebu. K základnímu seznámení se simulací QoS v simulačním nástroji OMNeT++ postačí výše uvedený postup. Postup je uveden v literatuře [21].

4.1.1 Popis modulu DropTailQoSQueue

Modul je důležitý pro implementaci QoS do simulované sítě. Proto je zde uveden jeho základní popis. Detailnější informace se dají najít v dokumentaci modulů v lit. [9].

Jde o modul fronty s požadovanou kvalitou služeb. Používá se v síťových rozhraních. Tento typ fronty řadí datové jednotky do front s určitou prioritou. Parametry modulu jsou uvedeny v tab. 4.1. Přehled vstupních a výstupních bran v tab. 4.2.

Tab. 4.1: Parametry modulu DropTailQoSQueue

Jméno	Typ	Popis
frameCapacity	číselná konstanta	kapacita jednotlivých podfront
classifierClass	řetězec znaků	třída

Tab. 4.2: Vstupní a výstupní brány modulu DropTailQoSQueue

Jméno	Směr	Popis
in	vstup	vstupní brána
out	výstup	výstupní brána

4.2 Model MPLS

V tomto modelu se předpokládá využití modulu **RSVP_LSR**, který je umístěn ve složce INET/Nodes/MPLS. Modul vkládá mezi fyzickou a linkovou vrstu jakousi mezivrstvu, která směruje pakety na základě MPLS návěstí. Pracuje v kombinaci s modulem TED (jeho umístění INET/Network/TED), který zjišťuje nejlepší cestu od zdroje k cíli. Dále je použit modul **RSVP**, což je popis signalizačního systému u technologie MPLS.

Pro model počítače je využit modul **StandardHost**. Jde o stejný modul, který je využívám pro simulování klasických sítí jako například v kapitole 1.6.5. Celé simulační schéma využívá LDP protokol.

V simulované síti (Obr. 5.1) bude také implementován směrovací protokol OSPF. Ten po zahájení simulace zajistí naplnění směrovacích tabulek síťových prvků. Proto jsou definované **link-state** zprávy, které mají za úkol šířit informace ostatním směrovačům. Podrobnější teoretický popis funkce směrovacího protokolu je uveden v kapitole 3. Pro správnou funkci směrovacího protokolu bude zapotřebí definovat u každého prvku nastavení OSPF protokolu. V tomto nastavení bude mimo jiné uvedeno, které sítě má směrovač přímo připojené, nastavení vlastních rozhraní, atd.

Teoretický popis simulace

Simulace by měla probíhat následovně: PC1 a PC2 budou vysílat UDP data. Ta se budou šířit ke směrovači LSR1, kde budou směrovány k cíli. Na začátku simulace pravděpodobně nebudou naplněny směrovací tabulky a tak bude paket odmítnut.

Až budou všechny směrovací tabulky kompletní tak, aby mohl být paket doručen k cíly, bude přenášený paket směrován (zatím klasickým způsobem) do cílového počítače server1. Žádné potvrzení o duručení se posílat nebude, protože jde o nespolehlivé spojení.

Propojení pomocí technologie MPLS nastane až v čase simulace 1s. V tuto dobu vygeneruje modul RSVP zprávu **Path**, která nese informace o požadavku na vytvoření spojení a rezervování určitých přenosových vlastností linky. Zpráva se bude šířit přes ostatní směrovače až k cílovému. Pokud jsou požadované parametry z hlediska sítě splnitelné v každém směrovači se zpráva předá dál a nevygeneruje se zpráva Error.

Zpráva **Path** dorazí až do koncového směrovače. Ten vytvoří požadované rezervace a odešle zpět k původci zprávy Path zprávu **Resv**. Zpráva se tedy bude šířit všemi směrovači, kterými prošla zpráva Path a v každém směrovači vytvoří požadované rezervace. Až zpráva Resv dorazí na první směrovač, dojde k vytvoření kompletní cesty a pakety, které byly dříve směrovány klasicky pomocí směrovací tabulky, jsou nyní na prvním prvku opatřeny návěstím a jsou již přeposílány vrstvou MPLS. Tím dojde k urychlení přenosu.

Další událost, která bude následovat se vyskytne v simulačním čase 2s. V tuto dobu bude využit speciální soubor s názvem *scenario.xml*. V souboru je zadefinováno vytvoření nového spojení. Vytvoří se tedy nová MPLS cesta, která bude inicializována opět prvkem LSR1. Tato cesta povede přes prvky LSR2, LSR4 a LSR5 a bude požadovat rezervaci šířky pásma 400 kbit/s tak, jak je definováno v souboru *scenario.xml*.

Z důvodu, že na prvku LSR4 je již rezervovaná šířka pásma 400 kbit/s a celková dostupná šířka pásma je 600 kbit/s, nemůže již dojít k další takovéto rezervaci. Tím pádem směrovač LSR4 odmítne požadavek a vyšle zprávu **PathErr**. Totéž provedou i ostatní prvky v cestě. V posledním směrovači LSR5 však pořád ještě zůstala požadovaná rezervace. Proto až směrovač LSR1 obdrží zprávu PathErr, odešle ještě zprávu **Path_TEAR**, která zajistí zrušení veškerých rezervací po celé cestě.

V simulačním čase 2.4 s vygeneruje modul RSVP na základě souboru *scenario.xml* zprávu **Path_TEAR** a zruší vytvořenou cestu přes LSR1, LSR3. LSR7 a LSR5, takže v síti zůstane navázána pouze jedna cesta přes LSR1, LSR2, LSR4 a LSR5. Ostatní pakety putující sítí se budou směrovat klasickým způsoben pomocí IP adres na síťové vrstvě.

Po ukončení simulace se nám samozřejmě vytvoří i soubory pro zobrazení grafů (Plove, Scalars). Zde si můžeme zobrazit využití kapacit u jednotlivých prvků. Změnou parametrů v *omnetpp.ini* můžeme změnit velikost front a nebo můžeme v souboru *RSVPTE4.ned* změnit například kapacity linek mezi spoji.

Detailnější popis modulů, ze kterých se skládá simulace lze najít v literatuře [9].

5 Praktická realizace

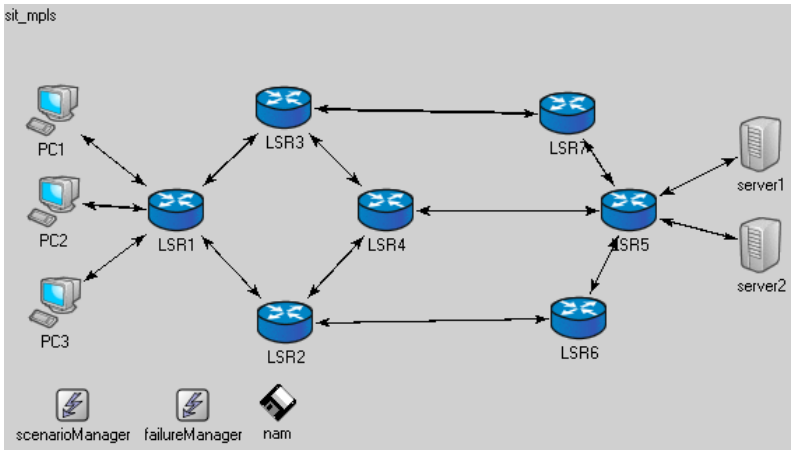
Jako první je zde uvedena realizace pomocí technologie MPLS, a to z důvodu, že technologie je názornější a jsou v ní patrny principy simulačního nástroje OMNeT++. V druhé kapitole je potom uveden základní simulační model technologie DiffServ.

5.1 Technologie MPLS

Úloha bude zaměřena na seznámení se se simulačním nástrojem OMNeT++ a jeho nadstavbou INET Framework. Dále budou zachyceny principy směrovacího protokolu OSPF. Hlavním cílem úlohy bude podrobné seznámení s technologií MPLS.

5.1.1 Postup vytváření simulované sítě

Výsledná síť by měla mít podobu zachycenou na Obr. 5.1.



Obr. 5.1: Výsledná simulovaná síť MPLS

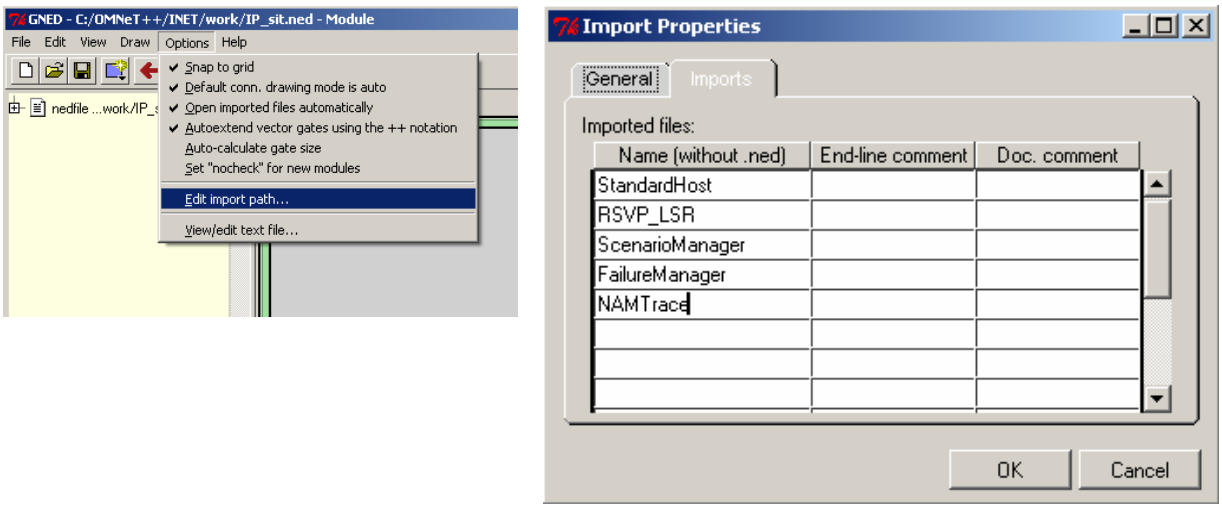
V prvním kroku si vytvoříme grafickou podobu simulované sítě. V pracovním adresáři (umístěný nejlépe do složky s nainstalovaným INET Framework) si vytvoříme soubor s příponou **.ned**. Název není závazný, například *sit_mpls.ned*. Spuštěním souboru otevřeme nástroj pro grafický návrh simulované sítě.

Po spuštění nejprve nastavíme cesty k jednotlivým simulačním modelům, které budeme využívat. To provedeme pomocí menu **Options -> Edit import path**. Do cest musíme zaznamenat následující:

```
C:/OMNeT++/INET/Nodes/INET
C:/OMNeT++/INET/Nodes/MPLS
C:/OMNeT++/INET/Network/Extras
C:/OMNeT++/INET/World
C:/OMNeT++/INET/Base
C:/OMNeT++/INET/Util
C:/OMNeT++/INET/Network/Contract
C:/OMNeT++/INET/Network/IPv4
C:/OMNeT++/INET/Applications/TCPApp
C:/OMNeT++/INET/Transport/TCP
C:/OMNeT++/INET/Transport/UDP
C:/OMNeT++/INET/Applications/UDPApp
C:/OMNeT++/INET/Applications/PingApp
C:/OMNeT++/INET/NetworkInterfaces/PPP
C:/OMNeT++/INET/NetworkInterfaces/Ethernet
C:/OMNeT++/INET/Network/MPLS
C:/OMNeT++/INET/Network/RSVP_TE
C:/OMNeT++/INET/Network/TED
C:/OMNeT++/INET/Network/ARP
C:/OMNeT++/INET/Network/Queue
```

Poznámka: Cesty lze přidat přímo zkopírováním výše uvedeného textu a nebo je možné cesty k jednotlivým modulům dohledat pomocí funkce vyhledávání v systému Windows.

Nyní provedeme **import** modulů, které budeme používat v simulaci. Půjde o moduly počítače (StandardHost), směrovače s podporou MPLS (RSVP_LSR) a dále o modul pro nahrání simulačního skriptu a chybového manageru (ScenarioManager, FailureManager) a v poslední řadě o modul pro generování výstupního souboru (NAMTrace). Moduly, které budeme importovat, se nachází ve složkách, které jsme určili nastavením cest. Ostatní podmoduly, ze kterých se skládají hlavní (např. StandardHost), se již importují automaticky společně s nadřazeným modulem. Do importačního dialogu se dostaneme přes menu **File -> New component -> Import** následně se přepneme na záložku **Properties**. Zadáme názvy požadovaných modulů bez přípony .ned a potvrdíme tlačítkem OK. Postup je patrný z Obr. 5.2

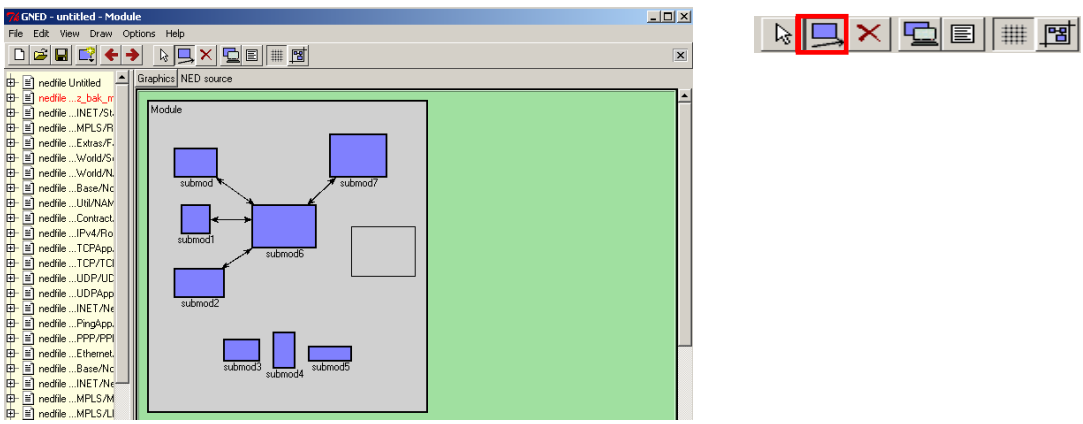


a) b)

Obr. 5.2: Import modulů: a) přístup do Import Properties, b) zadání názvů importovaných modulů

V tuto chvíli musíme program **restartovat**, aby se načetly importované moduly. Po znovu zapnutí programu již budou v levé části grafického prostření načteny požadované moduly (StandardHost, RSVP_LSR, atd.), ale i jejich podmoduly (např.: TCP, UDP nebo MPLS).

V dalším kroku vytvoříme v grafickém návrhovém prostředí topologii sítě (Obr. 5.3 a), kterou chceme simulovat. To provedeme pomocí ikony **Draw submodule and connections** (viz. Obr. 5.3 b)



a) b)

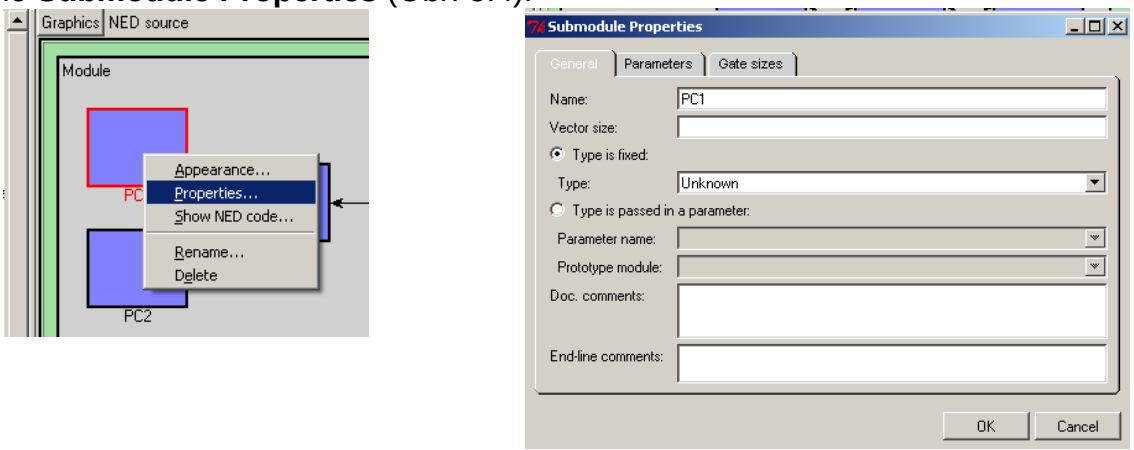
Obr. 5.3: Práce v návrhovém prostředí a) kreslení podmodulů, b) panel pro grafický návrh

Přejmenování podmodulů se provádí položkou **Rename** z nabídky po stisknutí pravého tlačítka myši na příslušném prvku.

Při grafickém navrhování můžeme sledovat, jak se mění nedfile naší simulované sítě. Přepneme se do něho pomocí záložky **NED source** nad sestavenou sítí. Pro úspěšnou simulaci je nutné do **NED source** nadefinovat parametry, týkající se přenosových linek mezi moduly. U propojovacích linek je nutné nastavit přenosovou rychlost (data rate), protože linky jsou připojeny k fyzickým rozhraním a to vyžaduje, aby přenosová rychlost byla nastavena. Do zdrojového souboru **NED source** ke každému spojení připíšeme následující definice:

```
PC1.out++ --> delay 10ms datarate 600000 --> LSR1.in++;
```

Nyní u jednotlivých prvků nadefinujeme, zda jde o směrovač, uživatelský počítač a nebo o jiný prvek. To provedeme tak, že na určitém prvku vyvoláme lokální nabídku pomocí pravého tlačítka myši a vybereme položku **Properties** (Obr. 5.4). Zobrazí se okno **Submodule Properties** (Obr. 5.4).

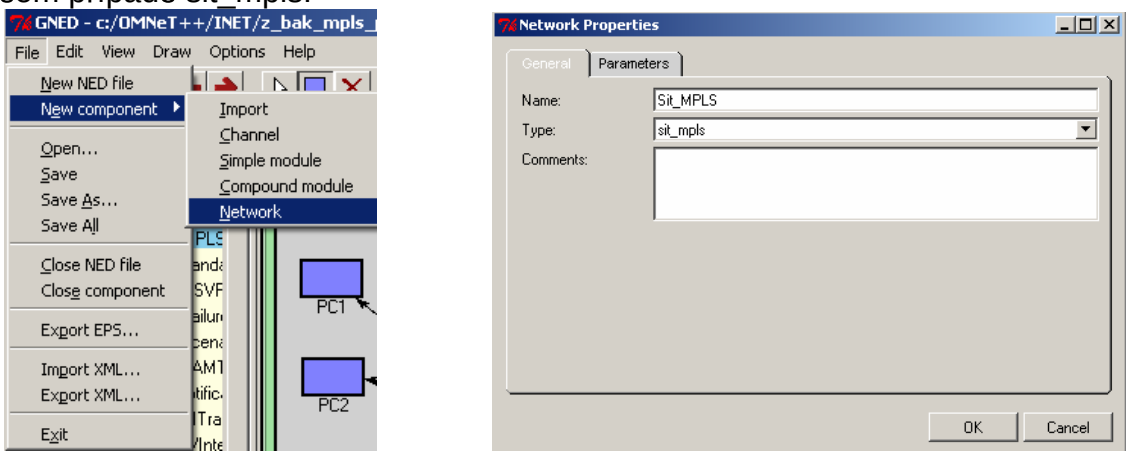


a) b)

Obr. 5.4: Nastavení vlastností podmodulu: a) přístup do Submodule Properties, b) nastavení typu

V tomto okně musíme v poli **Type** vybrat naimportovaný modul. Pro uživatelské počítače a servery vybereme StandardHost, pro směrovač modul RSVP_LSR, atd.

V posledním kroku musíme překladači určit, kterou síť má simulovat. V grafickém rozhraní se to dělá tak, že přidáme novou komponentu pomocí menu **File->New component->Network** (Obr. 5.5 a). Do pole **Name** zadáme název simulované sítě a v poli **Type** vybereme modul, ze kterého chceme simulovanou síť vytvořit (Obr. 5.5 b). V našem případě sit_mpls.



a) b)

Obr. 5.5: a) Přístup do Network Properties, b) volba názvu sítě a simulovaného modelu

Nakonec je ještě možné nastavit grafické ikony jednotlivým prvkům. Nastavení se provádí v položce **Appearance** v místní nabídce na daném prvku po kliknutí pravým tlačítkem v sekci **Icon**.

Po uvedeném postupu by jsme měli získat zdrojový kód v souboru *sit_mpls.ned*, který se podobá zdrojovému kódu v příloze B.1.

Dalším souborem, který budeme potřebovat bude již známý soubor *omnetpp.ini*, v němž jsou definovány nastavení jednotlivých prvků. Jeho obsah je uveden v příloze B.2. Podle zvoleného umístění pracovního adresáře je opět potřeba upravit cestu k seznamu souborů s příponou ned (druhý řádek v souboru *omnetpp.ini*).

Zavedení směrovacího protokolu OSPF

Protokol OSPF vyžaduje u každého prvku definovat přímo připojené sítě. To se provádí pomocí souborů s příponou **.rt**. V těchto souborech jsou nastaveny i IP adresy jednotlivých rozhraní. Každému prvku, který máme v simulačním modelu, vytvoříme speciální soubor *PC1.rt* (například pomocí textového editoru). Obsah pro PC1 bude tedy následující:

```
ifconfig:
name: ppp0  inet_addr: 10.0.1.1     MTU: 1500   Metric: 1
ifconfigend.

route:
10.1.1.1    *           255.255.255.255   H     0     ppp0
default:    10.1.1.1    0.0.0.0           G     0     ppp0
routeend.
```

V první části je definované nastavení rozhraní (název, IP adresa, Metrika) a v druhé části je definována směrovací cesta. Podrobnější popis je v kapitole 3, která se zabývá směrováním v IP sítích. Definice souborů pro ostatní prvky sítě jsou uvedeny v příloze B.3.

Nastavení MPLS směrovačů

V souboru *omnetpp.ini* jsou nastavení provedena pomocí externích konfiguračních souborů. Tyto soubory (s příponou .xml) umístíme do pracovního adresáře a slouží tedy k externímu konfigurování námi vytvořené sítě. Soubor s názvem *classifier.xml* slouží pro nastavení klasifikátoru paketů. Z důvodu, že nepožadujeme speciální nastavení tohoto modulu, napíšeme do souboru pouze následující text. Bez tohoto nadefinování by simulace neproběhla.

```
<?xml version="1.0"?>
<fectable>
</fectable>
```

Stejným způsobem vytvoříme i soubory *traffic.xml* a *libtable.xml*. Oba budou obsahovat pouze prázdné tagy. Jejich obsah je uveden v příloze B.4. Nastavení je globální pro všechny prvky LSR. Pro prvek LSR1 bude nastavení odlišné. Opět vytvoříme dva soubory s názvem *classifier_LSR1.xml* a *traffic_LSR1.xml*. Ty se budou aplikovat pouze na prvek LSR1. Jejich obsah je opět v příloze B.4.

Příklad vytváření těchto souborů je převzat z literatury [9], kde se po zadání názvu nadřazených modulů (např. RSVP) dozvíme, jakou mají mít takovéto soubory strukturu a k čemu slouží.

Ve stejné literatuře je definována i struktura posledního souboru, který bude pro simulaci nutný. Jde o soubor s názvem *scenario.xml*, který v simulačním čase 2s zahájí

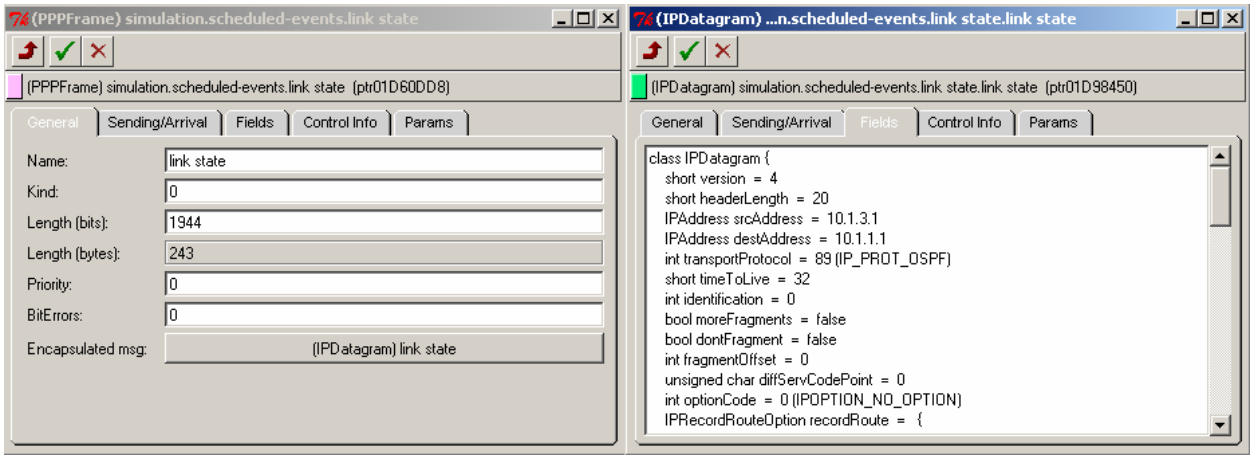
vytváření nového spojení (nové cesty MPLS). Spojení však nepůjde vytvořit z důvodu nepostačující kapacity sítě. (Více viz. Teoretický úvod k simulaci.) Obsah tohoto souboru je opět uveden v příloze B.4.

Nyní již stačí pouze do pracovního adresáře zkopírovat spustitelný soubor *inet.exe* ze složky c:\OMNeT++\INET\bin. Tímto souborem spustíme simulaci.

Simulace je založena na pozorování simulačního průběhu. Jsou zde samozřejmě také výstupy typu Plove a Scalars, ale princip technologie MPLS je patrná pouze z průběhu simulace.

Průběh simulace a pozorování jednotlivých parametrů:

Po spuštění simulace se začnou sítí šířit **link-state** zprávy, které souvisí se směrovacím protokolem OSPF a mají za úkol naplnit směrovací tabulky. Obsah těchto zpráv můžeme zobrazit dvojklikem na symbol zprávy. Poté se otevře okno PPP rámce. Pokud klikneme na tlačítko **Encapsulated msg** získáme IP paket. Zde se přepneme do záložky **Fields** a uvidíme obsah IP paketu, jako je například zdrojová a cílová IP adresa, typ protokolu, TimeToLive, atd. Pokud opět v záložce **General** klikneme na tlačítko **Encapsulated msg**, dostaneme se do struktury zprávy **link-state**. Zde opět v záložce **Fields** jsou uvedeny informace týkající se zprávy Link state. Postup je patrný z Obr. 5.6.



Obr. 5.6: Struktura a přístup do jednotlivých částí přenášené zprávy

Po průchodu několika **Link state** paketů (simulační čas 10ms), začnou počítače PC1 a PC2 vysílat UDP pakety. Tyto pakety lze sledovat stejným způsobem jako Link state pakety v předchozím odstavci. Paket bude putovat do prvního směrovače a tam do síťové vrstvy (zatím totiž není sestavena MPLS cesta). V síťové vrstvě bude položen dotaz na směrovací tabulku, zda je v ní požadovaný záznam. Záznam v tabulce nebude nalezen (zatím není síť konvergována) a tím pádem modul ICMP (součást síťové vrstvy) vygeneruje zprávu ICMP Error a vrátí ji odesilateli UDP paketu. To bude probíhat do té doby, než bude tabulka naplněna potřebnými záznamy o sítích.

Směrovací tabulku si můžeme zobrazit. Klikneme na LSR prvek, dále klikneme na podmodul **routingTable** a v okně zvolíme záložku **Contens**. Zde najdeme objekt se jménem **routes,** na který klikneme a tím získáme záznamy ve směrovací tabulce. Můžeme zde ověřit, že síť, do které směřuje UDP paket, v tabulce není. Ve stejné záložce najdeme i záznam **routerID**, který nám reprezentuje označení směrovače (jde o IP adresu).

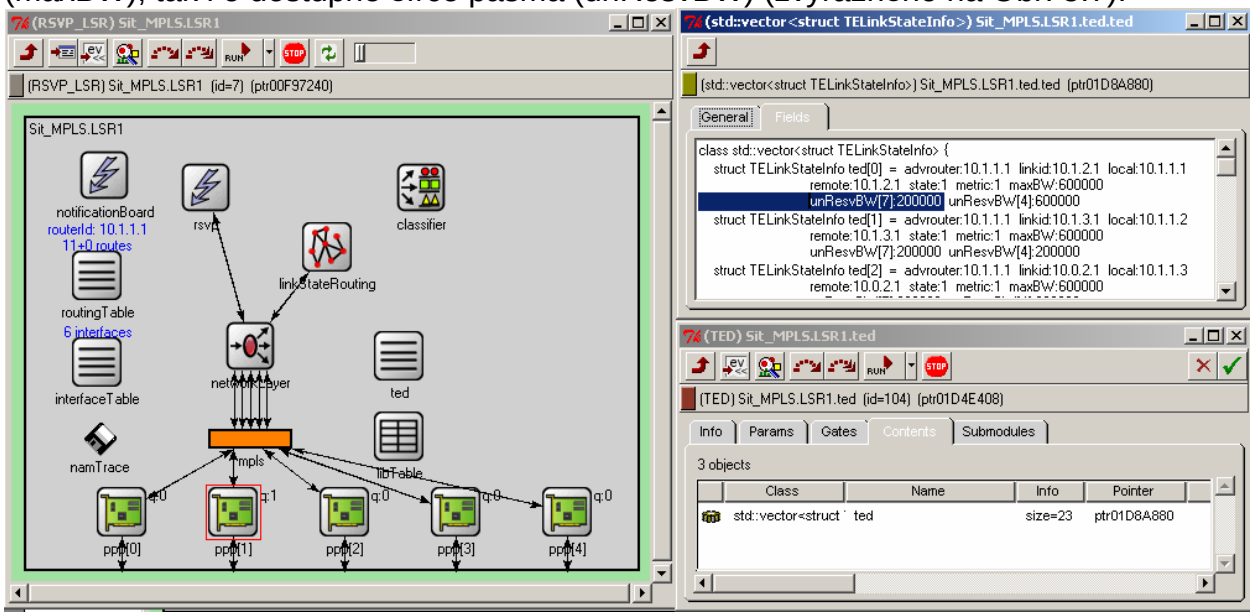
V čase simulace **t=60 ms** budou již směrovací tabulky kompletní a tím pádem již pakety putují sítí. Zatím jsou ovšem směrovány klasickým způsobem na 3. vrstvě pomocí IP adres. Pro každý paket se tedy prohledává směrovací tabulka.

Nyní je možné se přesunout do času simulace **1 s.** V tuto dobu bude vygenerován modulem **RSVP** ve směrovači LSR1 paket **Path**. Důležitý je jeho obsah a proto se do něho opět podíváme. Jaký má tento paket význam je uvedeno v kapitole 2.3.4. Path nese informace o potřebných rezervacích. Jsou v něm uvedeny potřebné adresy pro spojení (NextHop, tunnelID, atd.). Dále je uvedena priorita rezervací atd.

Až tato zpráva dorazí k cílovému směrovači (LSR5), vygeneruje se v něm zpráva **Resv**, která bude putovat stejnou cestou opět k LSR1. Během toho probíhá rezervace přenosové kapacity v jednotlivých prvcích.

Přehled rezervovaných prostředků je možný sledovat v každém prvku LSR v podmodulu **libTable**, na který si dvakrát klikneme a zvolíme záložku **Contents** a poklepeme na položku **lib**.

Neposlední, co ještě můžeme sledovat, je dostupná šířka pásma pro jednotlivá spojení. Opět v prvku LSR poklepeme tentokrát na podmodul **ted** a opět zvolíme **Contents** a položku **ted**. Zde jsou uloženy jak informace o celkové šířce pásma (maxBW), tak i o dostupné šířce pásma (unResvBW) (zvýrazněno na Obr. 5.7).



Obr. 5.7: Přehled rezervovaných a celkem dostupných šířek pásem pro jednotlivá rozhraní

V průběhu simulace (**čas 2s**) se vygeneruje nový požadavek (podle souboru *scenario.xml*) na vytvoření cesty. Ten je ovšem zamítnut prvkem LSR4, protože už nedokáže zaručit požadovanou šířku pásma. Tím se vygeneruje zpráva **Path_Err**, která indikuje zamítnutí požadavku na spojení. Zpráva **Path_Thear**, která ruší sestavenou cestu se vyskytne v čase **t=2,4s** a zruší již dříve navázané spojení. Více viz. teoretický úvod k simulaci.

Přesto, že princip technologie MPLS je nejvíce zřejmý z průběhu simulace, je nutné se ještě zmínit o souboru *vystup.nam*. Tento soubor je generován modulem **NAMTrace** a jsou do něho ukládány časové posloupnosti simulace.

Jeho nastavení je dáno v souboru *omnetpp.ini* položkami začínajícími **nam**. V souboru nejsou prvky zastupovány vlastními názvy, ale jsou označeny číslem, které je definované v *omnetpp.ini* souboru položkou **namid**.

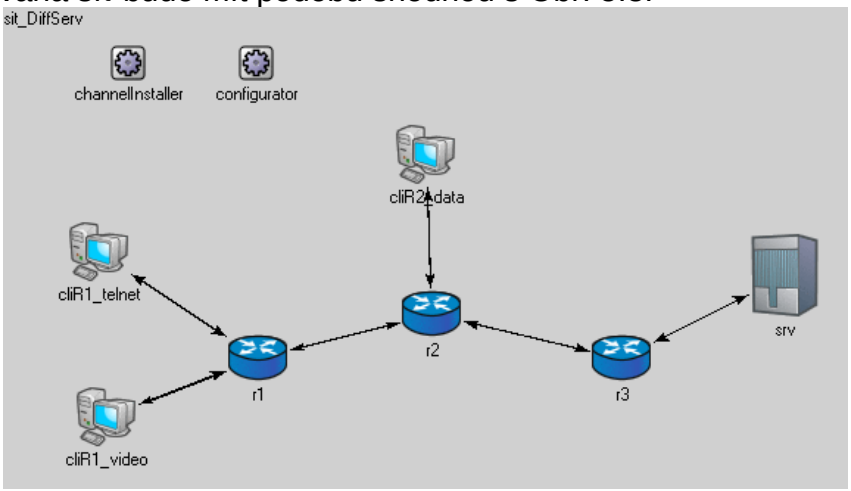
Zpočátku jsou v něm definována označení a dále pak spojení (šířka pásma jednotlivých linek, zpoždění). Na dalších řádcích již potom následuje údaj o simulačním čase, ve kterém událost nastala, údaj o zdrojovém prvku a cílovém prvku (zastupuje ho název definovaný v položce **namid)**.

Dalšími údaji získanými při simulaci jsou již známé soubory s příponou **.sca** a **.vec**. Práce s těmito soubory je rozebrána v kapitole 1.6.5.

5.2 Technologie DiffServ

Možnosti simulování technologie DiffServ v simulačním nástroji OMNeT++ se zdají být podle teoretických informací uvedených na stránkách výrobce značně jednoduché. Při praktické realizaci však narazíme na nejednu chybu, která brání v hladkém průběhu simulování. Simulaci lze tedy provést následujícím (i když ne příliš ideálním) způsobem.

Simulovaná síť bude mít podobu shodnou s Obr. 5.8.



Obr. 5.8: Výsledná simulovaná síť DiffServ

Začátek simulace (vytváření souborů, grafický návrh sítě) je shodný s postupem uvedeným v kapitolách 1.6.5, 5.1.1 a proto je v příloze C.1 uveden pouze zdrojový kód výsledného souboru s příponou .ned. Novinkou v souboru je definování propojení pomocí **channel**. V souboru se tímto definují globální vlastnosti jednotlivých propojení označených **ethernetline**. Pro takto nadefinované spojení je důležitý modul **ChannelInstaller**, který obsluhuje linky nadefinované pomocí direktivity **channel**. U jednoho ze spojení je definována nižší přenosová rychlost než u ostatních pomocí **datarate**. To je zde z toho důvodu, aby se na rozhraní tvořila fronta a mohly se uplatnit vlastnosti QoS.

Další soubor je uveden v příloze C.2, s názvem *omnetpp.ini*. Jeho obsah nastavuje jednotlivá spojení. Za povšimnutí patří hlavně část s konfigurací fyzického rozhraní u jednotlivých modulů:

```
**.ppp[*].queueType = "DropTailQoSQueue"
**.ppp[*].queue.classifierClass = "BasicDSCPClassifier"
**.ppp[*].queue.frameCapacity = 20
```

Zde je místo modulu **DropTailQueue** použit modul **DropTailQoSQueue**. Modul má upravené fronty jak je popsáno v kapitole 4.1. Modul **DropTailQoSQueue** vyžaduje, aby byl nastaven klasifikátor, který se použije pro třídění paketů do front. V rozšíření INET Framework je definován základní klasifikátor s názvem **BasicDSCPClassifier**.

Klasifikátor má za úkol jednak pracovat s polem DSCP (ToS) v IP paketu a také by měl rozlišit data od různých zdrojů. Modul pracuje společně se základním prvkem **IQoSClassifier**. Ten slouží jako virtuální rozhraní pro klasifikátor **BasicDCSPClassifier**. Pro lepší orientaci jsou v příloze C.2 uvedeny komentáře.

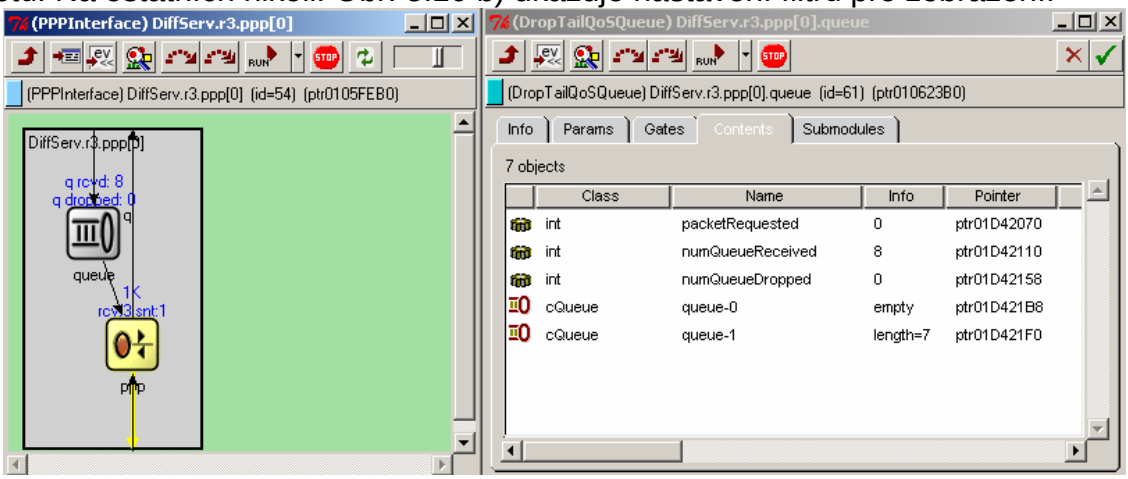
Následně ještě přikopírujeme soubor *inet.exe* pro spuštění simulace a povedeme kompilaci projektu postupem uvedeným v kapitole 1.6.5 za použití příkazu:

```
opp_nmakemake -f -N -b c:\omnet++\inet -c c:\omnet++\inet\inetconfig -I –n
```

Po spuštění simulace funguje správně, ale nefunguje třídění paketů do front. To je způsobeno nefunkční implementací QoS do modulů **DropTailQoSQueue**, který obsahuje **BasicDSCPClassifier** a **IQoSClassifier**. Moduly nedokáží správně označovat pakety, které pochází od různých zdrojů.

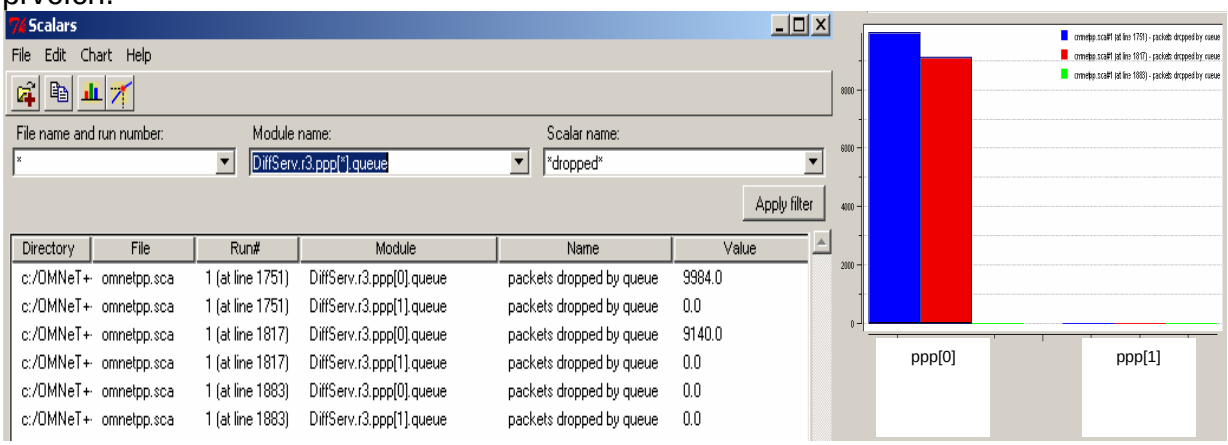
Nejjednodušší nápravou této chyby je vytvoření nového souboru *IPDatagram.msg*, který obsahuje strukturu IP paketu. V něm je položka **diffServCodePoint**, která zastupuje pole ToS v paketu. Pokud přímo modifikujeme její hodnotu a určíme, že v simulaci bude u zpráv týkajících se přenosu videa použit náš upravený soubor *IPDatagram.msg*, bude tímto souborem rovnou u zdroje nastaveno pole ToS. Tímto polem se budou řídit ostatní směrovače a tak v nich bude docházet k upřednostňování paketů, které jsou takto označeny.

Pakety můžeme potom sledovat dvojklikem na směrovač, dále dvojklikem na položku fyzického rozhraní. Ve fyzickém rozhraní klikneme na položku queue a v ní zvolíme záložku **Contents** (Obr. 5.9). Výběrem položky se jménem **queue-0**, popřípadě **1** otevřeme okno, kde budou uvedeny pakety ve frontě. Pakety, které jsou ovšem vytvářeny pomocí námi modifikovaného souboru *IPDatagram.msg* jsou řazeny přednostně. Tím je právě patrný princip QoS za pomoci technologie DiffServ. Na Obr. 5.10 b) jsou uvedeny počty zahozených paketů na směrovači r3 pro obě jeho rozhraní. Na rozhraní, kde je nastavena linka s nižší přenosovou rychlostí dochází k zahazování paketů. Na ostatních nikoli. Obr. 5.10 b) ukazuje nastavení filtrů pro zobrazení.



Obr. 5.9: Přístup do přehledu front na fyzickém rozhraní

Toto řešení není zrovna ideální, nicméně jako názorná ukázka postačuje. Ve výstupních souborech lze opět sledovat především zatížení front na jednotlivých prvcích.



Obr. 5.10: Výsledky simulace DiffServ a) nastavení filtrů, b) počty zahozených paketů

6 Závěr

Úkolem této práce bylo seznámit se se simulačním nástrojem OMNeT++ a jeho nadstavbou INET Framework. Zjistit možnosti simulace QoS v tomto simulačním nástroji a vyzkoušet jeho funkčnost.

Samotný nástroj OMNeT++ je z mého hlediska velice vhodný nejen pro simulaci počítačových a komunikačních sítí, ale lze ho s výhodou použít i pro simulování nejrůznějších typů front, spojení a různorodých situací i těch, které se přímo komunikačních sítí netýkají. Velkou výhodou je sada rozšíření INET Framework, která specializuje použití nástroje právě do oblasti komunikačních sítí a to jak drátových tak i bezdrátových. V programu je možné simulovat datový provoz na různých vrstvách modelu OSI/ISO, což není běžné pro jiné simulační nástroje. Další podstatnou výhodou je jeho otevřenost a samozřejmě licenční politika. Díky otevřenosti systému se dají vyzkoušet nové technologie téměř v jakémkoliv rozsahu. V programu se dají definovat a implementovat nové funkce a metody, které jsou využity v nových technologiích. Tím je tento simulační program vhodný pro vývoj a zkoumání moderních technologií.

Jasnou nevýhodou je z mého hlediska složitost postupu vytváření simulovaných sítí. Nejde jen o grafický návrh, ale především o složitost kompilace, která se provádí přes příkazový řádek. Nastavování všech parametrů simulace je realizováno v textových souborech a definice základních modulů je provedena v jazyce C++. Značnou nepříjemností je při vytváření modelu sítě nutnost pečlivého zadávání importovaných modulů, ze kterých se skládá výsledná síť. To vše klade na uživatele programu značné nároky a pro začínající a nezkušené programátory je systém velice náročný.

V simulačním nástroji není příliš propracovaná technologie pro podporu kvality služeb (QoS) v komunikačních sítích. Pro implementaci kvality služeb do simulovaných sítí existují v podstatě dva způsoby.

První způsob implementace QoS je založen na principu využití rezervačního protokolu RSVP. Protokol RSVP je základem pro model integrovaných služeb a také pro technologii MPLS. Propracování technologie MPLS je v rozšíření INET Framework dotaženo do výsledné formy, která je srovnatelná s funkcí opravdové fyzické sítě. Pomocí modulů podporujících MPLS (především modul MPLS_LSR) lze snadno sestavit požadovanou síť a z průběhu simulace je možné vysledovat princip technologie MPLS, popřípadě lze modifikací jistých parametrů ovlivnit chování této technologie a ověřit tak její vlastnosti i za nestandardních podmínek.

Druhým způsobem je implementace pomocí diferencovaných služeb. Technologie předpokládá schopnost rozeznat pakety od různých zdrojů, označovat je a řadit do front na fyzickém rozhraní modulu. Místo klasické fronty se použije fronta, která dovoluje třídit pakety na základě priorit. Modul této fronty je již definován v rozšíření INET Framework, ale jeho struktura vykazuje chybu. Modul totiž nedokáže značit pakety podle jejich druhu a tím pádem nemůže technologie pracovat správně.

Právě díky těmto malým chybám, které se nachází nejen v rozšíření INET Framework, kde jsou způsobeny tím, že jednotlivé moduly vytváří samotní uživatelé, ale (a to je podstatné) občas se vyskytne chyba i přímo v hlavním programu OMNeT++. Ta potom způsobí pád celého programu, což se mně během realizace této práce stávalo téměř pravidelně. Příčinou těchto chyb je zajisté to, že jde o program s volnou licencí a tudíž nemůžeme předpokládat jeho robustnost a stabilitu. I přes tyto nepříjemné vlastnosti je program velice názorný a užitečný díky jeho modularitě a otevřenosti.

LITERATURA

[1] *OMNeT++ Community Site : File Management* [online]. c2008 [cit. 2008-11-26]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/filemgmt/singlefile.php?lid=123>.

[2] *OMNeT++ Community Site : File Listing - Category View* [online]. c2008 [cit. 2008-10-22]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/filemgmt/viewcat.php?cid=2>.

[3] HAWTIN, Matthew. *OMNeT++ Wiki : WinXP* [online]. 2006 , September 29, 2006, at 03:54 AM [cit. 2008-11-29]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/pmwiki/index.php?n=Main.WinXP>.

[4] *Microsoft : Visual Studio Download* [online]. c2008 [cit. 2008-11-22]. Instalační soubor ke stažení. Dostupný z WWW: <http://www.microsoft.com/express/download/>.

[5]*Microsoft : download center* [online]. c2008 [cit. 2008-11-23]. Platform SDK ke stažení. Dostupný z WWW: <http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en >.

[6]*Microsoft : download center* [online]. [2008] [cit. 2008-11-23]. Instalační soubor nmake.exe ke stažení. Dostupný z WWW: <http://download.microsoft.com/download/vc15/patch/1.52/w95/en-us/nmake15.exe>.

[7] *OMNeT++ : TicToc Tutorial for OMNeT++* [online]. 2005 , Generated on Wed Oct 19 09:59:33 2005 [cit. 2008-11-22]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/doc/tictoc-tutorial>.

[8] *OMNeT++ Tictoc Tutorial : Getting started* [online]. 2005 [cit. 2008-11-23]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/doc/tictoc-tutorial/part1.html>. Oct 19 09:59:33 2005.

[9] *OMNeT++ : Model documentation* [online]. [2006] [cit. 2008-11-29]. Popis modelů. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/doc/INET/neddoc/index.html>.

[10] *Bakalarka* [online]. [2008] , Last modified 21-May-2008 21:56 [cit. 2008-11-26]. Text v angličtině. Dostupný z WWW: <http://www.bakalarkaomnetpp.ic.cz/bakalarka/ >.

[11] *OMNeT++ Community Site : File Management* [online]. c2008 [cit. 2008-12-01]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/filemgmt/singlefile.php?lid=121 >.

[12] *OMNeT++ Community Site : File Management* [online]. c2008 [cit. 2008-12-01]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/filemgmt/singlefile.php?lid=120 >.

[13] RUSSELL, W. Quong. *OMNeT++ : User manual* [online]. [2007] [cit. 2008-12-01]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/doc/manual/usman.html>.

[14] KACÁLEK, Jan. *Modely pro zajištění kvality služeb* [online]. c2006 [cit. 2008-10-20]. Dostupný z WWW: <http://amarok.cesketelekomunikace.cz/xkacal00/?actions_rel >.

[15] ČÍKA, Petr. *Multimediální služby*. Brno : [s.n.], 2007. 106 s.

[16] UBIK, Sven. *CESNET : Technická zpráva 6/2006* [online]. c1996-2008 , 29. 9. 2000 [cit. 2008-10-12]. Dostupný z WWW: <http://www.cesnet.cz/doc/techzpravy/2000-6/>.

[17] *Technologie MPLS* [online]. [2007] [cit. 2008-10-14]. Formát pdf. Dostupný z WWW: <https://dsn.felk.cvut.cz/wiki/_media/vyuka/cviceni/x36mti/prezentace2007/petrim2-doc.pdf?id...2007&cache=cache >.

[18] NOVOTNÝ, Vít Ing, PhD. *Účastnická koncová zařízení*. Brno : [s.n.], 2002. 122 s.

[19] HERMAN, Ivo Ing, CSc. *Komunikační technologie : Kapitola č. 6: Přenos dat v reálném čase* [online]. [2007] , poslední změna 26.12.2007 [cit. 2008-11-21]. Dostupný z WWW: <http://www.utko.feec.vutbr.cz/~herman/bkom.html>.

[20] *SBM (Subnet Bandwidth Manager)* [online]. 2000 , May 2000 [cit. 2008-10-22]. Text v angličtině. RFC. Dostupný z WWW: <http://www.rfc-editor.org/rfc/rfc2814.txt>.

[21] *OMNeT++ : Wiki* [online]. 2006 , Page last modified on August 11, 2006, at 12:58 PM [cit. 2008-12-02]. Text v angličtině. Dostupný z WWW: <http://www.omnetpp.org/pmwiki/index.php?n=Main.INETDiffServ>.

[22] *File Sender/Receiver Over TCP and IPv4* [online]. c1999-2003 , revision r1.8 - 17 Oct 2006 - 03:20 GMT [cit. 2008-12-11]. Text v angličtině. Dostupný z WWW: <http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/FileSenderreceiverOverTCPIPv4>.

[23] *Směrovací protokol OSPF* [online]. [2006] [cit. 2009-05-13]. Dostupný z WWW: <http://www.cs.vsb.cz/grygarek/SPS/lect/OSPF/ospf.html>.

[24] *Internetworking Technology Handbook : Open Shortest Path First (OSPF)* [online]. c1992-2009 [cit. 2009-04-24]. Text v angličtině. Dostupný z WWW: <http://www.cisco.com/en/US/docs/internetworking/technology/handbook/OSPF.html>.

SEZNAM ZKRATEK

ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BA	Bandwidth Allocator
BA	Behavior Aggregate
DiffServ	Differentiated Services
DS	Differentiated Services
DSCP	Differentiated Services Code Point
FDDI	Fiber Distributed Data Interface
FEC	Forward Equivalence Class
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IN	vstup
IntServ	Integrated Services
IP	Internet Protocol
IPv6	Internet Protocol verze 6
LDP	Label Distribution Protokol
LER	Label Edge Router
LSP	Label Switch Path
LSR	Label Switch Route
MF	Multi-Field
MPLS	MultiProtocol Label Switching
MPLS	Multiprotocol Label Switching
NED	Network Design
OSI	Open Systems Interconnection
OSI/ISO	Open Systems Interconection/International Organization for Standardization
OSPF	Open Shortest Path First
OUT	výstup
PBH	Per Hop Behaiour
PPP	Point to Point Protocol
QoS	Quality of Service
RIP	Routing Information Protocol
RM	Requestor Module
RSVP	ReSerVation Protocol
SBM	Subnet Bandwidth Management
SLA	Service Level Agreemet
SSM	Statistical Synchronization
Tcl	Tool Command Language
TCP	Transmission Control Protocol
ToS	Type of Service
TTL	Time To Live
UDP	User Datagram Protocol

SEZNAM PŘÍLOH

A Zdrojové kódy pro simulaci klasické sítě

A.1 Soubor omnetpp.ini

```
[General]
preload-ned-files   =   *.ned   @../../../nedfiles.lst   //cesta   k seznamu
                                                  //nedfiles.lst     (musí     se
                                                  //upravit    v závislosti    na
                                                  // pracovním adresáři)
sim-time-limit = 9s                          //maximální simulační čas

network = Network                            //simulovaná síť s názvem Network

[Cmdenv]
express-mode = no

[Tkenv]
default-run=1                                //zobrazení simulace graf. rozhraní

[Parameters]
**.namid = -1     //parametr pro NAMTraceWriter – udává umístění uvnitř
                  //hosta nebo směrovače
**.numUdpApps=0   //nastavení UDP přenosu (v našem případě nebude UDP využit)
**.udpAppType="UDPBasicApp"
**.PC1.numTcpApps=1                          //budeme využívat jedno TCP spojení
**.PC1.tcpAppType="TCPSessionApp"            //typ TCP přenosu (zdroj)
**.PC1.tcpApp[0].active=true                 //TCP bude aktivováno.
**.PC1.tcpApp[0].address=""                  //adresa
**.PC1.tcpApp[0].port=-1                     //port
**.PC1.tcpApp[0].connectAddress="PC2"        //TCP spojení s PC2
**.PC1.tcpApp[0].connectPort=1000            //port spojení
**.PC1.tcpApp[0].tOpen=exponential(0.1)      //čas založení TCP spoje
**.PC1.tcpApp[0].tSend=0                     //odesílání
**.PC1.tcpApp[0].sendBytes=10000000          //velikost dat k odeslání 10MB
**.PC1.tcpApp[0].sendScript=""               //prázdný parametr
**.PC1.tcpApp[0].tClose=0                    //čas zavření spojení

**.PC2.numTcpApps=1                          //nastavení pro PC2
**.PC2.tcpAppType="TCPSinkApp"               //cíl TCP spojení
**.PC2.tcpApp[0].address=""
**.PC2.tcpApp[0].port=1000

**.tcp.recordStats=1                         //vlastnosti TCP spojení
**.tcp.mss=1024                              //velikost dat. jednotky v Bajtech
**.tcp.advertisedWindow= 65536               //velikost okna

**.tcp.sendQueueClass="TCPVirtualDataSendQueue"      //typ front
**.tcp.receiveQueueClass="TCPVirtualDataRcvQueue"
**.tcp.tcpAlgorithmClass="TCPReno"

# Ping
**.PC1.pingApp.destAddr="PC2"                //parametry pro ping
**.pingApp.destAddr=""
**.pingApp.srcAddr=""
**.pingApp.packetSize=56
**.pingApp.interval=1
**.pingApp.hopLimit=32
**.pingApp.count=0
**.pingApp.startTime=1
**.pingApp.stopTime=0
```

```
**.pingApp.printPing=true

**.routingFile=""            //nastavení IP protokolu
**.ip.procDelay=10us
**.PC1.IPForward=false       //PC1 ani PC2 nejsou určeny k přeposílání
**.PC2.IPForward=false

**.arp.retryTimeout = 1      //nastavení ARP protokolu
**.arp.retryCount = 3
**.arp.cacheTimeout = 100
**.networkLayer.proxyARP = true
                                                //nastavení fyz. rozhraní
**.ppp[*].queueType = "DropTailQueue"           //typ fronty
**.PC1.ppp[*].queue.frameCapacity   = 70        //kapacita fronty PC1
**.PC2.ppp[*].queue.frameCapacity = 70          //kap.fronty PC2 (počet pak.)
**.Router.ppp[*].queue.frameCapacity   = 6      //kapacity u směrovače
```

B Zdrojové kódy pro simulaci MPLS

B.1 Soubor sit_mpls.ned

```
import                              //import  podmodulů  ze  kterých  se  bude
                                    //skládat simulace
    "StandardHost",                 //modul pro PC a server
    "RSVP_LSR",                     //směrovač
    "FailureManager",               //chybový modul
    "ScenarioManager",              //nastavení průběhu simulace
    "NAMTrace";                     //generování výstupních souborů

module sit_mpls                     //definice systémového modulu
    submodules:                     //definice podmodulů
        PC1: StandardHost;                      //PC1 typ StandardHost
            display: "p=47,88;i=device/pc4";    //nastavení pozice a ikony
        PC2: StandardHost;
            display: "p=48,153;i=device/pc4";
        PC3: StandardHost;
            display: "p=49,225;i=device/pc4";
        server1: StandardHost;                  //server1 typ StandardHost
            display: "p=554,112;i=device/server2";
        server2: StandardHost;
            display: "p=554,184;i=device/server2";
        LSR1: RSVP_LSR;                         //směrovač LSR1 typ RSVP_LSR
            gatesizes:              //definice počtu
                in[5],              //vstupů
                out[5];             //a výstupů
            display: "p=136,159;i=abstract/router";
        LSR2: RSVP_LSR;
            gatesizes:
                in[3],
                out[3];
            display: "p=214,239;i=abstract/router";
        LSR3: RSVP_LSR;
            gatesizes:
                in[3],
                out[3];
            display: "p=213,86;i=abstract/router";
        LSR4: RSVP_LSR;
            gatesizes:
                in[3],
                out[3];
            display: "p=286,159;i=abstract/router";
        LSR5: RSVP_LSR;
            gatesizes:
                in[5],
                out[5];
            display: "p=460,159;i=abstract/router";
        LSR6: RSVP_LSR;
            gatesizes:
                in[2],
                out[2];
            display: "p=424,236;i=abstract/router";
        LSR7: RSVP_LSR;
            gatesizes:
                in[2],
                out[2];
            display: "p=416,90;i=abstract/router";

        scenarioManager: ScenarioManager;           //definice podmodulu
```

```
            display: "p=62,298;i=block/control_s";   //pro nastavení scénáře
        failureManager: FailureManager;              //chybový modul
            display: "p=148,298;i=block/control_s";
        nam: NAMTrace;                               //pro generování výstupu
            display: "p=208,296;i=old/floppy1";
    connections nocheck:                             //definice propojení
        LSR1.out[0] --> delay 15ms datarate 600000 --> LSR2.in[0];
        LSR1.in[0] <-- delay 15ms datarate 600000 <-- LSR2.out[0];
                          //je definována přenosová rychlost a zpoždění
        LSR1.out[1] --> delay 5ms datarate 600000 --> LSR3.in[0];
        LSR1.in[1] <-- delay 5ms datarate 600000 <-- LSR3.out[0];

        PC2.in++ <-- delay 10ms datarate 600000 <-- LSR1.out[2];
        PC2.out++ --> delay 10ms datarate 600000 --> LSR1.in[2];

        PC1.in++ <-- delay 10ms datarate 600000 <-- LSR1.out[3];
        PC1.out++ --> delay 10ms datarate 600000 --> LSR1.in[3];

        LSR2.out[1] --> delay 5ms datarate 600000 --> LSR4.in[0];
        LSR2.in[1] <-- delay 5ms datarate 600000 <-- LSR4.out[0];

        LSR3.out[1] --> delay 5ms datarate 600000 --> LSR4.in[2];
        LSR3.in[1] <-- delay 5ms datarate 600000 <-- LSR4.out[2];

        LSR4.out[1] --> delay 5ms datarate 600000 --> LSR5.in[0];
        LSR4.in[1] <-- delay 5ms datarate 600000 <-- LSR5.out[0];

        LSR5.out[1] --> delay 10ms datarate 600000 --> server1.in++;
        LSR5.in[1] <-- delay 10ms datarate 600000 <-- server1.out++;

        LSR5.out[2] --> delay 10ms datarate 600000 --> server2.in++;
        LSR5.in[2] <-- delay 10ms datarate 600000 <-- server2.out++;

        LSR2.out[2] --> delay 10ms datarate 600000 --> LSR6.in[0];
        LSR2.in[2] <-- delay 10ms datarate 600000 <-- LSR6.out[0];

        LSR5.out[3] --> delay 10ms datarate 600000 --> LSR6.in[1];
        LSR5.in[3] <-- delay 10ms datarate 600000 <-- LSR6.out[1];

        LSR3.out[2] --> delay 10ms datarate 600000 --> LSR7.in[0];
        LSR3.in[2] <-- delay 10ms datarate 600000 <-- LSR7.out[0];

        LSR5.out[4] --> delay 10ms datarate 600000 --> LSR7.in[1];
        LSR5.in[4] <-- delay 10ms datarate 600000 <-- LSR7.out[1];

        PC3.in++ <-- delay 10ms datarate 600000 <-- LSR1.out[4];
        PC3.out++ --> delay 10ms datarate 600000 --> LSR1.in[4];
endmodule

network Sit_MPLS : sit_mpls        //definice sítě, kterou budeme simulovat
endnetwork
```

B.2 Soubor omnetpp.ini

```
[General]
preload-ned-files = *.ned @../nedfiles.lst     //cesta k seznamu .ned souborů
network = Sit_MPLS                             //simulovaná síť

sim-time-limit = 5s                            //max. délka simulace

total-stack-kb =  65536                        //velikost mezipaměti

[Cmdenv]                          //definice textového rozhraní
```

```
express-mode = no                   //expresní mód nebude použit

[Tkenv]                             //definice grafického rozhraní
plugin-path=../../Etc/plugins       //cesta k souborům se skriptem
default-run = 1                     //počet opakování simulace 1

[Parameters]                        //nastavení spojení a přenosu dat

# nastaveni IP konfigurace (jsou použity i nastavovací soubory)
**.PC1.IPForward=false              //žádný z koncových prvků ...
**.PC2.IPForward=false              //... nepřeposílá pakety
**.server1.IPForward=false
**.server2.IPForward=false
**.PC3.IPForward=false
**.PC1.routingFile = "PC1.rt"       //nastavení směrovacích tabulek ...
**.PC2.routingFile = "PC2.rt"       //... pomocí směrovacích souborů
**.PC3.routingFile = "PC3.rt"
**.server1.routingFile = "server1.rt"
**.server2.routingFile = "server2.rt"

#nastaveni UPD spojeni u každého prvku
**.PC1.numUdpApps=1                 //definice počtu UDP aplikací na PC1
**.PC1.udpAppType="UDPBasicApp"     //typ UDP aplikace
**.PC1.udpApp[0].local_port = 100   //zdrojový port
**.PC1.udpApp[0].dest_port = 100    //cílový port
**.PC1.udpApp[0].message_length = 1000    //délka zprávy
**.PC1.udpApp[0].message_freq = 0.01s     //opakování vysílání zprávy
**.PC1.udpApp[0].dest_addresses = "10.2.1.1"   //cílová IP adresa

**.PC2.numUdpApps=1
**.PC2.udpAppType="UDPBasicApp"
**.PC2.udpApp[0].local_port = 100
**.PC2.udpApp[0].dest_port = 100
**.PC2.udpApp[0].message_length = 1000
**.PC2.udpApp[0].message_freq = 0.01s
**.PC2.udpApp[0].dest_addresses = "10.2.2.1"   //změna cílové IP adresy

**.server1.numUdpApps=1
**.server1.udpAppType="UDPSink"     //typ pro příjem UDP dat
**.server1.udpApp[0].local_port = 100    //cílový port

**.server2.numUdpApps=1
**.server2.udpAppType="UDPSink"
**.server2.udpApp[0].local_port = 100

**.numUdpApps=0                     //globální nastavení pro ostatní prvky
**.udpAppType="UDPBasicApp"

# nastaveni tcp spojeni musí byt definované i když nebude použito
**.PC*.numTcpApps=0                       //počet TCP aplikací
**.PC*.tcpAppType="TCPGenericSrvApp"      //typ TCP aplikace
**.PC*.tcpApp[0].address=""               //není využito proto není podstatné
**.PC*.tcpApp[0].port=1000
**.PC*.tcpApp[0].replyDelay=0             //čekání před odpovědí

**.server*.numTcpApps=0                   //stejné nastavení pro servery
**.server*.tcpAppType="TCPGenericSrvApp"
**.server*.tcpApp[0].address=""
**.server*.tcpApp[0].port=1000
**.server*.tcpApp[0].replyDelay=0

**.tcp.mss = 1024                         //definice zprávy TCP
**.tcp.advertisedWindow = 14336           //velikost potvrzovacího okna
```

```
**.tcp.sendQueueClass="TCPMsgBasedSendQueue"   //nastavení odesílací ...
**.tcp.receiveQueueClass="TCPMsgBasedRcvQueue" //... a přijímací fronty
**.tcp.tcpAlgorithmClass="TCPReno"             //druh tcp algoritmu
**.tcp.recordStats=true                        //povolení záznamu

# aplikace ping                                //nastavení ping aplikace
**.pingApp.destAddr=""                         //cílová adresa
**.pingApp.srcAddr=""                          //zdrojová
**.pingApp.packetSize=56                       //velikost paketu
**.pingApp.interval=1                          //interval opakování
**.pingApp.hopLimit=32                         //počet přeskoků (TTL)
**.pingApp.count=0                             //počáteční stav čítače
**.pingApp.startTime=1                         //čas spuštění aplikace
**.pingApp.stopTime=0                          //zastavení
**.pingApp.printPing=true                      //zapnutí výpisu

# ARP configuration               //nastavení překladu fyzických adres na IP
**.arp.retryTimeout = 1           //čas opakování
**.arp.retryCount = 3             //počet opakovacích cyklů
**.arp.cacheTimeout = 100         //vypršení časovače
**.networkLayer.proxyARP = true   //zapnutí proxy

# konfigurace fyzického rozhraní
**.ppp[*].queueType = "DropTailQueue"    //typ fronty
**.ppp[*].queue.frameCapacity = 10       //kapacita výstupních front

#nastaveni výstupu do souboru trace.nam
**.nam.logfile = "vystup.nam"            //definice názvu výstupního souboru
**.nam.prolog = "c -t * -i 1 -n Red;c -t * -i 2 -n Blue;c -t * -i 100 -n
Green;c -t * -i 101 -n Magenta;c -t * -i 200 -n Orange;c -t * -i 300 -n
Brown"           //jeho nastavení (patří do jednoho řádku)
**.PC1.namid = 11                        //zastupující název
**.PC2.namid = 12
**.PC3.namid = 13
**.server1.namid = 14
**.server2.namid = 15

# načte vstupní soubor se simulačním skriptem
**.scenarioManager.script = xmldoc("scenario.xml") //práce s extern. souborem
**.ip.procDelay=20us                     //čas potřebný na zprac. události

# konfigurace směrovačů MPLS
**.LSR1.classifier.conf = xmldoc("LSR1_fec.xml")   //nastavení klasifikátoru
**.LSR1.rsvp.traffic = xmldoc("LSR1_rsvp.xml")     //pomocí extern. souborů

**.LSR*.classifier.conf = xmldoc("_fec.xml")
**.LSR*.rsvp.traffic = xmldoc("_traffic.xml")
**.LSR*.rsvp.helloInterval = 0.2               //čas zasílání hello paketu
**.LSR*.rsvp.helloTimeout = 0.5                //vypršení časovače
**.LSR*.libTable.conf = xmldoc("_lib.xml")     //prázdná tabulka s rezervací

**.LSR1.routerId = "10.1.1.1"                  //označení směrovače LSR1
**.LSR1.routingFile = "LSR1.rt"                //načte směrovací soubor
**.LSR1.peers = "ppp0 ppp1"                    //definuje připojené rozhraní
**.LSR1.namid = 1                              //ke směrovačům; označení


**.LSR2.routerId = "10.1.2.1"
**.LSR2.routingFile = "LSR2.rt"
**.LSR2.peers = "ppp0 ppp1 ppp2"
**.LSR2.namid = 2                              //zástupné označení směrovače

**.LSR3.routerId = "10.1.3.1"
```

```
**.LSR3.routingFile = "LSR3.rt"
**.LSR3.peers = "ppp0 ppp1 ppp2"
**.LSR3.namid = 3

**.LSR4.routerId = "10.1.4.1"
**.LSR4.routingFile = "LSR4.rt"
**.LSR4.peers = "ppp0 ppp1 ppp2"
**.LSR4.namid = 4

**.LSR5.routerId = "10.1.5.1"
**.LSR5.routingFile = "LSR5.rt"
**.LSR5.peers = "ppp0 ppp3 ppp4"
**.LSR5.namid = 5

**.LSR6.routerId = "10.1.6.1"
**.LSR6.routingFile = "LSR6.rt"
**.LSR6.peers = "ppp0 ppp1"
**.LSR6.namid = 6

**.LSR7.routerId = "10.1.7.1"
**.LSR7.routingFile = "LSR7.rt"
**.LSR7.peers = "ppp0 ppp1"
**.LSR7.namid = 7
```

B.3 Zdrojové kódy směrovacích souborů

PC1.rt

```
ifconfig:         //definice nastavení rozhraní
      name: ppp0  inet_addr: 10.0.1.1     MTU: 1500   Metric: 1
ifconfigend.
route:            //nastavení směrovací tabulky
      10.1.1.1    *           255.255.255.255   H     0     ppp0
      default:    10.1.1.1    0.0.0.0           G     0     ppp0
routeend.
```

PC2.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.0.2.1     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.1.1    *           255.255.255.255   H     0     ppp0
      default:    10.1.1.1    0.0.0.0           G     0     ppp0
routeend.
```

PC3.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.0.3.1     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.1.1    *           255.255.255.255   H     0     ppp0
      default:    10.1.1.1    0.0.0.0           G     0     ppp0
routeend.
```

server1.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.2.1.1     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.5.1    *           255.255.255.255   H     0     ppp0
```

```
      default:    10.1.5.1    0.0.0.0           G     0     ppp0
routeend.
```

server2.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.2.2.1     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.5.1    *           255.255.255.255   H     0     ppp0
      default:    10.1.5.1    0.0.0.0           G     0     ppp0
routeend.
```

LSR1.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.1.1.1     MTU: 1500   Metric: 1
      name: ppp1  inet_addr: 10.1.1.2     MTU: 1500   Metric: 1
      name: ppp2  inet_addr: 10.1.1.3     MTU: 1500   Metric: 1
      name: ppp3  inet_addr: 10.1.1.4     MTU: 1500   Metric: 1
      name: ppp4  inet_addr: 10.1.1.5     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.2.1    10.1.2.1    255.255.255.255   H     0     ppp0
      10.1.3.1    10.1.3.1    255.255.255.255   H     0     ppp1
      10.0.2.1    10.0.2.1    255.255.255.255   H     0     ppp2
      10.0.1.1    10.0.1.1    255.255.255.255   H     0     ppp3
      10.0.3.1    10.0.3.1    255.255.255.255   H     0     ppp4
routeend.
```

LSR2.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.1.2.1     MTU: 1500   Metric: 1
      name: ppp1  inet_addr: 10.1.2.2     MTU: 1500   Metric: 1
      name: ppp2  inet_addr: 10.1.2.3     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.1.1    10.1.1.1    255.255.255.255   H     0     ppp0
      10.1.4.1    10.1.4.1    255.255.255.255   H     0     ppp1
      10.1.6.1    10.1.6.1    255.255.255.255   H     0     ppp2
routeend.
```

LSR3.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.1.3.1     MTU: 1500   Metric: 1
      name: ppp1  inet_addr: 10.1.3.2     MTU: 1500   Metric: 1
      name: ppp2  inet_addr: 10.1.3.3     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.1.1    10.1.1.2    255.255.255.255   H     0     ppp0
      10.1.4.1    10.1.4.3    255.255.255.255   H     0     ppp1
      10.1.7.1    10.1.7.1    255.255.255.255   H     0     ppp2
routeend.
```

LSR4.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.1.4.1     MTU: 1500   Metric: 1
      name: ppp1  inet_addr: 10.1.4.2     MTU: 1500   Metric: 1
      name: ppp2  inet_addr: 10.1.4.3     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.2.1    10.1.2.2    255.255.255.255   H     0     ppp0
```

```
      10.1.5.1    10.1.5.1    255.255.255.255   H     0     ppp1
      10.1.3.1    10.1.3.2    255.255.255.255   H     0     ppp2
routeend.
```

LSR5.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.1.5.1     MTU: 1500   Metric: 1
      name: ppp1  inet_addr: 10.1.5.2     MTU: 1500   Metric: 1
      name: ppp2  inet_addr: 10.1.5.3     MTU: 1500   Metric: 1
      name: ppp3  inet_addr: 10.1.5.4     MTU: 1500   Metric: 1
      name: ppp4  inet_addr: 10.1.5.5     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.4.1    10.1.4.2    255.255.255.255   H     0     ppp0
      10.2.1.1    10.2.1.1    255.255.255.255   H     0     ppp1
      10.2.2.1    10.2.2.1    255.255.255.255   H     0     ppp2
      10.1.6.1    10.1.6.2    255.255.255.255   H     0     ppp3
      10.1.7.1    10.1.7.2    255.255.255.255   H     0     ppp4
routeend.
```

LSR6.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.1.6.1     MTU: 1500   Metric: 1
      name: ppp1  inet_addr: 10.1.6.2     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.2.1    10.1.2.3    255.255.255.255   H     0     ppp0
      10.1.5.1    10.1.5.4    255.255.255.255   H     0     ppp1
routeend.
```

LSR7.rt

```
ifconfig:
      name: ppp0  inet_addr: 10.1.7.1     MTU: 1500   Metric: 1
      name: ppp1  inet_addr: 10.1.7.2     MTU: 1500   Metric: 1
ifconfigend.
route:
      10.1.3.1    10.1.3.3    255.255.255.255   H     0     ppp0
      10.1.5.1    10.1.5.5    255.255.255.255   H     0     ppp1
routeend.
```

B.4 Zdrojové kódy pro nastavení MPLS

classifie.xml

```
<?xml version="1.0"?>        //defaultní nastavení FEC pro všechny prvky
<fectable>
</fectable>
```

traffic.xml

```
<?xml version="1.0"?>        //defaultní nastavení provozu pro prvky v síti
<sessions>
</sessions>
```

libtable.xml

```
<?xml version="1.0"?>        //naplnění tabulky s rezervacemi šířky pásma
<libtable>
</libtable>
```

classifier_LSR1

```
<?xml version="1.0"?>        //nastavení FEC konkrétně pro prvek LSR1
```

```
<fectable>
      <fecentry>
            <id>1</id>                                  //označení cesty

            <destination>10.2.1.1</destination>       //cíl sestavené cesty

            <tunnel_id>1</tunnel_id>      //označení virtuálního tunelu
            <lspid>100</lspid>            //zástupné označení spoje
      </fecentry>
      <fecentry>
            <id>2</id>                                  //druhé spojení

            <destination>10.2.2.1</destination>       //cíl cesty

            <tunnel_id>2</tunnel_id>                  //označení tunelu
            <lspid>200</lspid>                        //zástupné označení
      </fecentry>
</fectable>
```

traffic_LSR1

```
<?xml version="1.0"?>
<sessions>
      <session>                                 //definuje provoz
            <endpoint>10.2.1.1</endpoint>       //cíl
            <tunnel_id>1</tunnel_id>            //tunel

            <setup_pri>1</setup_pri>            //priorita při vytváření
            <holding_pri>1</holding_pri>

            <paths>
                  <path>                        //definice cesty
                        <lspid>100</lspid>

                        <bandwidth>400000</bandwidth> //rezervovaná šířka ...
                        <route>                       //... pásma
                              <node>10.1.1.1</node>   //definice přes ...
                              <node>10.1.3.1</node>   //... které ...
                              <node>10.1.7.1</node>   //... směrovače ...
                              <node>10.1.5.1</node>   //... povede ...
                        </route>                      //sestavená cesta

                        <permanent>true</permanent>
                        <color>100</color>            //barva spojení
                  </path>
            </paths>
      </session>
      <session>               //druhé spojení (nastavení totožné s předchozím)
            <endpoint>10.2.2.1</endpoint>
            <tunnel_id>2</tunnel_id>

            <paths>
                  <path>
                        <lspid>200</lspid>

                        <bandwidth>400000</bandwidth>
                        <route>
                              <node>10.1.1.1</node>
                              <node>10.1.2.1</node>
                              <node>10.1.4.1</node>
                              <node>10.1.5.1</node>
                        </route>
```

```
                        <permanent>true</permanent>
                        <color>200</color>
                  </path>
            </paths>
      </session>
</sessions>
```

scenario.xml

```
<?xml version="1.0"?>
<scenario>              //definice průběhu simulace
      <at t="2">        //čas, kdy se daná událost aplikuje
            <add-session module="LSR1.rsvp">    //cíl, kam se událost aplikuje
                  <endpoint>10.2.1.1</endpoint> //cílový bod
                  <tunnel_id>1</tunnel_id>      //označení cesty

                  <paths>
                        <path>                  //navázání cesty
                              <lspid>101</lspid>

                              <bandwidth>400000</bandwidth>
                              <route>
                                    <node>10.1.1.1</node>
                                    <node>10.1.2.1</node>
                                    <node>10.1.4.1</node>
                                    <node>10.1.5.1</node>
                              </route>

                              <color>100</color>
                        </path>
                  </paths>
            </add-session>
      </at>
      <at t="2.2">            //další událost v čase 2,2s
            <bind-fec module="LSR1.classifier">
                  <id>1</id>
                  <endpoint>10.2.1.1</endpoint>
                  <tunnel_id>1</tunnel_id>
                  <destination>10.2.1.1</destination>
                  <lspid>101</lspid>
            </bind-fec>
      </at>
      <at t="2.4">            //událost v čase 2,4s
            <del-session module="LSR1.rsvp">          //odstranění spojení
                  <endpoint>10.2.1.1</endpoint>       //označení spojení
                  <tunnel_id>1</tunnel_id>            //označení tunelu

                  <paths>
                        <path>
                              <lspid>100</lspid>
                        </path>
                  </paths>
            </del-session>
      </at>
</scenario>
```

C Zdrojové kódy pro technologii DiffServ

C.1 Soubor sit_DiffServ.ned

```
import                                  //definice podmodulů
    "Router",
    "StandardHost",
    "FlatNetworkConfigurator",
    "ChannelInstaller";

channel ethernetline                    //definice linky ethernetline
    delay 0.1us;                        //zpoždění
    datarate 10*1000000;                //propustnost
endchannel

module sit_DiffServ
    submodules:
        channelInstaller: ChannelInstaller;
            parameters:
                channelClass = "ThruputMeteringChannel",   //nastavení ...
                channelAttrs = "format=u";                 //... modulu ...
            display: "p=98,50;i=block/cogwheel_s";   //... channelInstaler
        configurator: FlatNetworkConfigurator;
            parameters:      //nastavení sítě
                moduleTypes = "Router StandardHost", //kam budou přidány IP
                nonIPModuleTypes = "",               //moduly bez IP adres
                networkAddress = "145.236.0.0",      //síťová adresa
                netmask = "255.255.0.0";             //maska sítě
            display: "p=185,50;i=block/cogwheel_s";
        r1: Router;                                  //r1 typu Router
            display: "p=186,250;i=abstract/router";
        r2: Router;
            display: "p=304,220;i=abstract/router";
        r3: Router;
            display: "p=433,250;i=abstract/router";
        cliR1_telnet: StandardHost;                  //cliR1_telnet ...
            display: "p=78,178;i=device/pc2";        //... typ StandardHost
        cliR1_video: StandardHost;             //nastavení ikony a pozice
            display: "p=83,290;i=device/pc2";
        cliR2_data: StandardHost;
            display: "i=device/pc2;p=300,112";
        srv: StandardHost;
            display: "p=544,194;i=device/server_l";
    connections nocheck:
                 //definice propojení (je použita linka ethernetline)
        cliR1_telnet.out++ --> ethernetline --> r1.in++;
        cliR1_telnet.in++ <-- ethernetline <-- r1.out++;

        cliR1_video.out++ --> ethernetline --> r1.in++;
        cliR1_video.in++ <-- ethernetline <-- r1.out++;

        cliR2_data.out++ --> ethernetline --> r2.in++;
        cliR2_data.in++ <-- ethernetline <-- r2.out++;

        r1.out++ --> ethernetline --> r2.in++;
        r1.in++ <-- ethernetline <-- r2.out++;

        r2.out++ --> ethernetline --> r3.in++;
        r2.in++ <-- datarate 1000 <-- r3.out++;      //přímo nastavená ...
                                        //... propustnost (z důvodu QoS)
        r3.out++ --> ethernetline --> srv.in++;
```

```
        r3.in++ <-- ethernetline <-- srv.out++;
endmodule

network DiffServ : sit_DiffServ          //definice simulované sítě
endnetwork
```

C.2 Soubor omnetpp.ini

```
[General]
preload-ned-files = *.ned @../nedfiles.lst     //cesta k seznamu .ned
;debug-on-errors = true                        //nastavení chybového výstupu

network = DiffServ                             //definice simul. sítě

 [Cmdenv]
express-mode = no

[Tkenv]
plugin-path=../../Etc/plugins
default-run=1

[Parameters]

# UDP aplikace
**.cliR2_data.numUdpApps=0                //UDP provoz na cliR2_data vypnut
**.cliR2_data.udpAppType="UDPBasicApp"    //typ UDP (musí být nastaven ...
**.cliR1_telnet.numUdpApps=0             //... i když není využit)
**.cliR1_telnet.udpAppType="UDPBasicApp"
**.cliR1_video.numTcpApps=0
**.cliR1_video.tcpAppType="TCPBasicApp"

**.cliR1_video.numUdpApps=1                       //počet UDP aplikací
**.cliR1_video.udpAppType="UDPVideoStreamCli"  //typ
**.cliR1_video.udpApp[*].serverAddress = "srv" //server pro UDP provoz
**.cliR1_video.udpApp[*].localPort = 9999      //zdroj. port
**.cliR1_video.udpApp[*].serverPort = 3088     //cílový port
**.cliR1_video.udpApp[*].startTime = 0.99      //začátek vysílání UDP zpráv

**.srv.numUdpApps=1
**.srv.udpAppType = "UDPVideoStreamSvr"
**.srv.udpApp[*].videoSize = 1e7               //velikost přenášeného videa
**.srv.udpApp[*].serverPort = 3088
**.srv.udpApp[*].waitInterval = .01            //čas před odpovědí
**.srv.udpApp[*].packetLen = 1000              //délka paketu

# TCP applikace
**.cliR1_telnet.numTcpApps=1                   //počet TCP na cliR1_telnet
**.cliR1_telnet.tcpAppType="TelnetApp"         //typ provozu
**.cliR1_telnet.tcpApp[0].address=""           //vlastní adresa
**.cliR1_telnet.tcpApp[0].port=-1              //port (automaticky)
**.cliR1_telnet.tcpApp[0].connectAddress="srv" //spojení na server (cíl)
**.cliR1_telnet.tcpApp[0].connectPort=1000     //spojení na port

**.cliR1_telnet.tcpApp[0].startTime=1          //začátek spojení
**.cliR1_telnet.tcpApp[0].numCommands=100      //počet přenesených příkazů
**.cliR1_telnet.tcpApp[0].commandLength=exponential(10)    //proměnlivá ...
**.cliR1_telnet.tcpApp[0].keyPressDelay=exponential(0.1)   //...délka příkazu
**.cliR1_telnet.tcpApp[0].commandOutputLength=exponential(40) //čas zprac.
**.cliR1_telnet.tcpApp[0].thinkTime=truncnormal(2,3)       //čas při ...
**.cliR1_telnet.tcpApp[0].idleInterval=truncnormal(3600,1200) //... spojení
**.cliR1_telnet.tcpApp[0].reconnectInterval=30 //obnovovací interval
**.cliR2_data.numTcpApps=1                //počet TCP na cliR2_data
**.cliR2_data.tcpAppType="TCPSessionApp"  //typ
```

```
**.cliR2_data.tcpApp[0].active=true      //v provozu
**.cliR2_data.tcpApp[0].address=""       //IP adresa zdroje
**.cliR2_data.tcpApp[0].port=-1          //port zdroje
**.cliR2_data.tcpApp[0].connectAddress="srv"   //cíl
**.cliR2_data.tcpApp[0].connectPort=1001       port cílový
**.cliR2_data.tcpApp[0].tOpen=1          //čas otevření spojení
**.cliR2_data.tcpApp[0].tSend=0          //čas odeslání dat
**.cliR2_data.tcpApp[0].sendBytes=10000000     //celková délka dat
**.cliR2_data.tcpApp[0].sendScript=""    //bez skriptu
**.cliR2_data.tcpApp[0].tClose=0         //čas ukončení (není nastaveno)...
                                         //proto až se přenesou celé data
**.tcp.mss = 1024                        //velikost dat
**.tcp.advertisedWindow = 14336          //potvrzovací okno
**.tcp.sendQueueClass="TCPMsgBasedSendQueue"   //třída vysílací
**.tcp.receiveQueueClass="TCPMsgBasedRcvQueue" //přijímací fronty
**.tcp.tcpAlgorithmClass="TCPReno"       //algoritmus TCP
**.tcp.recordStats=true                  //záznam povolen

**.srv.numTcpApps=2                      //server bude odpovídat 2 uživatelům
**.srv.tcpAppType="TCPSinkApp"           //typ aplikace
**.srv.tcpApp[1].address=""              //zdroj. adresa
**.srv.tcpApp[1].port=1001               //port
**.srv.tcpApp[1].replyDelay=0            //doba před odpovědí
**.srv.tcpApp[0].address=""              //definice druhého TCP spojení
**.srv.tcpApp[0].port=1000
**.srv.tcpApp[0].replyDelay=0

# ping aplikace (vypnuto)
**.pingApp.destAddr=""                   //ping vypnut, i přesto je nutné ...
**.pingApp.srcAddr=""                    //definovat následující parametry
**.pingApp.packetSize=56                 //zdroj. a cíl. adresa, velikost ...
**.pingApp.interval=1                    //... paketu
**.pingApp.hopLimit=32                   //interval, limit přeskoků (TTL)
**.pingApp.count=0                       //počítadlo na 0
**.pingApp.startTime=1                   //čas počátku
**.pingApp.stopTime=10                   //a konce
**.pingApp.printPing=true                //výpis

# nastavení síťové vrstvy
**.routingFile=""                        //není použit externí soubor
**.ip.procDelay=10us                     //doba zpracování
**.cliR1_telnet.IPForward=false          //klienti a servery ...
**.cliR1_video.IPForward=false           //... nepřeposílají pakety
**.cliR2_data.IPForward=false
**.srv*.IPForward=false

# nastavení ARP
**.arp.retryTimeout = 1                  //opakovací limit
**.arp.retryCount = 3                    //počet opakování
**.arp.cacheTimeout = 100                //opakovací zásobník
**.networkLayer.proxyARP = true

# konfigurace fyzických rozhraních
**.ppp[*].queueType = "DropTailQoSQueue" //modul pro zajištění QoS
**.ppp[*].queue.classifierClass = "BasicDSCPClassifier"//použitý klasifikátor
**.ppp[*].queue.frameCapacity = 20       //kapacita fronty
**.qosBehaviorClass = "EnqueueWithQoS"   //nastaví jak se chová fronta

# nam trace
**.nam.logfile = "trace1.nam"            //generuje výstupní soubor
**.nam.prolog = ""                       //bez speciálního nastavení
**.namid = -1                            //pojmenování prvků - automaticky
```