VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2017 Jakub Petrůj

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

NOVÉ LABORATORNÍ ÚLOHY V SIMULAČNÍM PROSTŘEDÍ OMNET++

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE
AUTHOR

Jakub Petrůj

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Lukáš Langhammer

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Jakub Petrůj ***ID:*** 173724
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Nové laboratorní úlohy v simulačním prostředí OMNeT++

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku komunikačních protokolů. Zadání práce spočívá v návrhu dvou nových laboratorních úloh v simulačním prostředí OMNeT++ včetně kompletních návodů vhodných pro studenty zahrnující výchozí scénář i doplňující úkoly a kontrolní otázky. Při návrhu úloh se zaměřte na okruhy: porovnání transportních protokolů TCP, UDP, SCTP, srovnání technologií Ethernet a WLAN, či rozbor aplikačních protokolů (FTP, HTTP, SMTP, DNS, popřípadě další). Časová náročnost každé úlohy musí být přibližně dvě hodiny.

DOPORUČENÁ LITERATURA:

[1] NS3 Consortium: Dokumentace k Network Simulator 3 (NS3), 2014. Dostupné online <https://omnetpp.org/documentation>, citováno 13.9.2016.

[2] JEŘÁBEK, J. Komunikační technologie. Skriptum FEKT Vysoké učení technické v Brně, 2016. s. 1-172.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. Lukáš Langhammer, Ph.D.
Konzultant:

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

ABSTRAKT

Náplní bakalářské práce je návrh a vytvoření dvou laboratorních úloh v síťovém simulačním prostředí OMNeT++ s rozšířením INET Framework pro předmět komunikační technologie. Součástí je popis simulačního prostředí včetně jazyka NED a jeho modulů, dále simulace projektu a analýza výsledků. V další části práce jsou vyobrazeny teoretické znalosti potřebné ke zhotovení, simulaci a vyhodnocení výsledků souvisejících s tématy laboratorních úloh. Témata navržených úloh se soustředí na aplikační protokoly, adresaci a podsíťování. První praktická část se zabývá srovnání protokolů HTTP, FTP, Telnet a VOIP ve scénáři souběžného i nesouběžného provozu, za zvýšené chybovosti linky, při jejím výpadku či při přetížení. Druhá praktická část sleduje podrobněji dynamickou konfiguraci sítě pomocí protokolu DHCP, navíc zkoumá problém neefektivního a efektivního podsíťování, věnuje se také statické adresaci sítí.

KLÍČOVÁ SLOVA

OMNeT++, INET Framework, Tkenv, HTTP, FTP, Telnet, VOIP, DHCP, síťový simulátor, podsíťování, IP adresace

ABSTRACT

This Bachelor thesis illustrates a draft and creation of two lab exercises in a network simulation environment of OMNeT++ with INET Framework extension on behalf of Communication Technologie course. Initialy, simulated environment itself is defined and, including NED language and its modules, then the process of simulation and analysis of its results are presented and discused. Further more this work describes theoretical network knowledge required to create the simulation and treat its results. The first methodical part of this work compares HTTP, FTP, Telnet and VOIP protocols in simultaneous and non simultaneous running, with increased occurance of errors on line or its complete outage. The second methodical part of this work observes dynamic network configuration using DHCP protocol more in detail. Futhermore, this part studies the question of both non-efective and efective subnetting and handles issues concerning static adressing of network.

KEYWORDS

OMNeT++, INET Framework, Tkenv, HTTP, FTP, Telnet, VOIP, DHCP, network simulator, subnetting, IP addressing

PETRŮJ, J. *Nové laboratorní úlohy v simulačním prostředí OMNeT++*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. 62 s. Vedoucí bakalářské práce Ing. Lukáš Langhammer, Ph.D..

PROHLÁŠENÍ

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

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Lukáši Langhammerovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ....

(podpis autora)

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

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

PODĚKOVÁNÍ

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

Brno . . . . . .

. . . . . .

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato bakalářská práce si klade za cíl detailně prostudovat síťové simulační prostředí OMNeT++ rozšířené o nástavbu INET Framework. S jeho pomocí pak zhotovit dvě laboratorní úlohy – včetně kompletního návodu – do předmětu komunikační technologie pro studenty druhého ročníku, studijního oboru teleinformatika bakalářského studia. První úloha slouží pro rozšíření znalostí o protokolech aplikační vrstvy, konkrétně protokoly HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), Telnet (Telecommunication Network Protocol) a VoIP (Voice Over Internet Protocol). Druhá úloha pojednává o problematice dynamické konfigurace sítě pomocí protokolu DHCP (Dynamic Host Configuration Protocol) a možných situacích, které mohou při použití tohoto protokolu nastat. Zaobírá se také vytvářením podsítí a nakonec se věnuje statické adresaci několika podsítí v simulačním prostředí OMNeT++. Práce se člení na část teoretickou a praktickou.

V první kapitole teoretické části je popsáno simulační prostředí OMNeT++, popis vývojového prostředí a grafického uživatelského rozhraní. Rozepsán je i jazyk NED (Network Description), jeho vlastnosti a komponenty. Na konci kapitoly se nachází vysvětlení principu analýzy výsledků.

Další kapitola se věnuje části popisu aplikační vrstvy a zejména pak protokolům aplikační vrstvy, tudíž teorie, na které staví jedna z úloh bakalářské práce. V této kapitole je také rozepsán protokol DHCP, který je obsažen v látce druhé úlohy.

Kapitola číslo tři pojednává o IP adresaci a podsíťování, součástí je popis IP adresy, masky sítě a třídní adresace.

Čtvrtá kapitola slouží jako teoretický návrh zadání pro praktickou část bakalářské práce – popisuje cíle úloh, jednotlivé scénáře a topologii sítě.

Poslední dvě kapitoly – praktické části – se zabývají kompletací sítě, simulací scénářů a analýzou výsledku. Jedná se o návody, které krok po kroku vysvětlují principy konkrétních problematik. Součástí každé úlohy je návod s popsaným cílem, teoretickým úvodem a postupem řešení. Na konci každé úlohy jsou otázky, které budou klíčové při kontrole.

1 SÍŤOVÝ SIMULÁTOR OMNET ++

OMNeT++ je modulární, diskrétní simulační nástroj. Je postavený na platformě Eclipse, díky čemuž nabízí mnoho rozšíření. Je naprogramován v jazyce C++. Simulační nástroj nabízí kvalitní objektově orientované prostředí. Jeho licence je volně dostupná jak pro akademické, tak pro nekomerční využití, a proto je vhodný obzvláště pro studenty. Pro komerční využití existuje simulační vývojové prostředí OMNEST. Simulátor OMNeT++ je multiplatformní, a proto není problém využití jak na operačních systémech Windows, tak na systémech s jádrem UNIX.

Vznik celého projektu se datuje od roku 1992, kdy si András Vagra, autor celého prostředí, dal za cíl vytvořit otevřený diskrétní simulační nástroj. Samotná myšlenka byla inspirována již existujícím simulačním nástrojem Network Simulator a komerčním Opnet modelerem. Jeho počin byl dokončen roku 1997. Od začátku vytvoření OMNeT++, se komunita využívající simulačního prostředí stále rozšiřuje. Neustále se také rozšiřují možnosti prostředí.

Vývojové prostředí je modulární, skládá se z jednotlivých modelů, které mezi sebou komunikují. Tyto moduly lze dělit na jednoduché moduly a složené moduly. Každý jednoduchý modul je popsán pomocí knihovny třídy a obsahuje předem naprogramované vstupní a výstupní brány pro komunikaci. Nachází se na nejnižší úrovni hierarchické části. Tomuto prvku lze nastavovat předem definované vlastnosti a určit tak specifikace daného modulu. Složené moduly jsou pak dílčí součástí těchto jednoduchých modelů, počet není omezen [1].

Obrázek 1: Úvodní okno simulačního prostředí OMNeT++ verze 5

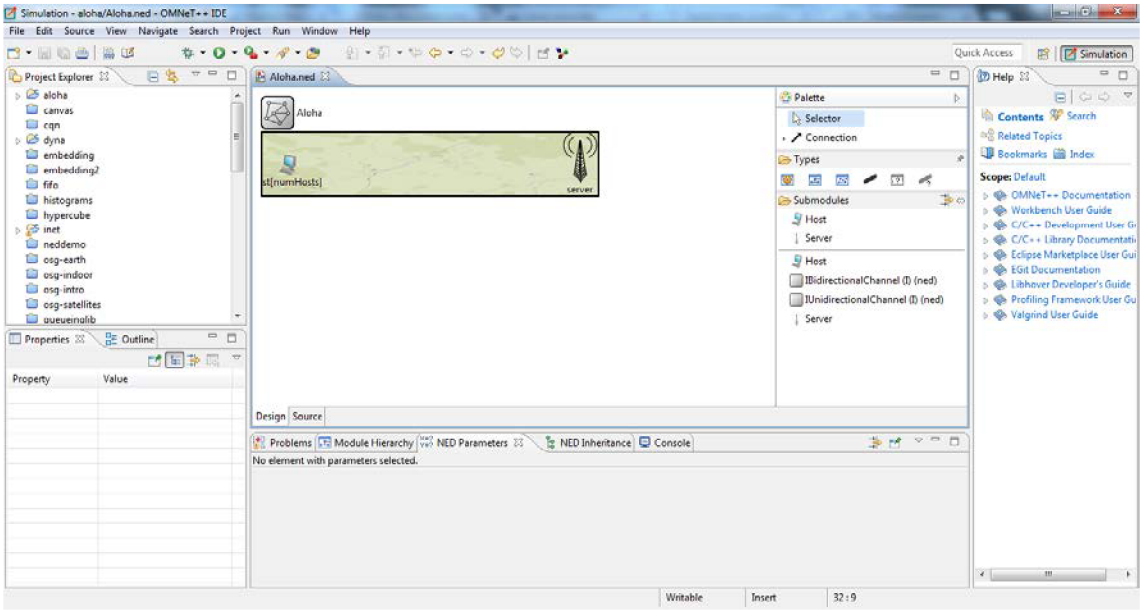
1.1 Diskrétní událostní simulace

Diskrétní systém událostí je systém, v němž ke změně stavu (události) dochází v nespojitých instancích času. OMNeT++ tento systém podporuje. Výhodou je, že si jednotlivé události na své vykonání neberou žádný čas. Předpokládá se, že mezi dvěma po sobě následujícími událostmi nedochází ke změnám systému. Události obsahují časovou značku odpovídající době výskytu v daném systému. Po uplynutí této doby jsou odstraněny [1].

Doba, kdy dojde k událostem, se často nazývá časové razítko. OMNeT++ používá termín čas příjezdu. Doba modelu se pak označuje jako simulační čas, čas modulu, nebo virtuální čas oproti reálnému času, nebo času procesoru. Tyto zmíněné časy se vztahují k tomu, jak dlouho byl simulační program spuštěn, a kolik času procesoru bylo spotřebováno.

1.2 Popis vývojového prostředí

Vývojové prostředí (Obrázek 2) je podobné, jako ostatní IDE (Integrated Development Environment) prostředí typu Java. Je rozděleno na několik částí. Horní část vyplňuje lišta se všemi dostupnými funkcemi a nástroji programu. Pod ní se nachází lišta s ikonkami pro rychlý výběr nejpoužívanějších nástrojů prostředí. V levé části okna lze spatřit Project Explorer, ten slouží pro vytváření a editaci projektů buďto již vytvořených, nebo vlastních. Další okno, nacházející se v levém dolním rohu, slouží pro zobrazení vlastností aktuálně vybraného souboru, případně zobrazuje možnosti nastavení právě vybraného modulu.



Obrázek 2: Prostředí OMNeT++

Centrální okno, nacházející se v centru obrazovky, slouží pro tvorbu simulace. Pro snadnější vytváření topologie sítě je zde možnost návrhového prostředí, kdy lze moduly a podmoduly jednoduše přetáhnout z postranní palety, nebo pak jednotlivou síť popsat skrze zdrojový kód. Vhod může přijít i kombinace obou zmíněných. Nakonec je zde dolní část prostředí, ta slouží pro vypisování chyb při kompilaci, nastavení parametrů modulů, zobrazení hierarchie modulů a záložku console.

1.3 Jazyk NED

Uživatel popisuje strukturu simulačního modelu v jazyce NED (Network Description). Jazyk NED umožňuje deklarovat jednoduché moduly, a pomocí jejich propojení je možno sestavit moduly složené. Přidáním složených modulů a jejich propojením lze snadno vytvořit simulovanou síť. K propojení složených modulů slouží komunikační kanál – další typ komponentů, jejichž instance mohou být použity ve složených modulech.

Jazyk NED obsahuje tyto vlastnosti [1]:

- **Hierarchie** – Běžný způsob pro přístup k jednotlivým modulům je zavedením hierarchického přístupu. V OMNeT++ lze každý modul, který je příliš komplexní, rozdělit na menší moduly.
- **Komponenty** – Jednoduché a složené moduly se dají snadno kopírovat. Vhodné je také použití knihoven komponent (např. INET Framework, MiXim).
- **Rozhraní** – Lze použít jako zástupce modulu či kanálu. Zástupci je možno nastavit parametry při startu. Pro jednoduchou zaměnitelnost modulů se musí implementovat rozhraní.
- **Dědičnost** – Moduly a kanály mohou být odvozeny. Těmto odvozeným prvkům lze jednoduše přidávat nové parametry a rozhraní.
- **Balíčky** – Jazyk NED podporuje balíčkovou strukturu, podobně jako tomu je u jazyku Java. Slouží k redukci stejně pojmenovaných prvků různých modulů.
- **Vnitřní typy** – Vlastní definice lokálních kanálů a modulů pro lepší přehlednost v názvech.
- **Metadata –** Mohou popisovat moduly, parametry, kanály, brány a podmoduly přidáním vlastností. Metadata nejsou využívána simulačním jádrem, ale mohou obsahovat extra informace nebo různé nástroje pro prostředí a jiné moduly. Jedná se například o ikony u výše zmíněných komponent, nebo například měrnou jednotku (miliwatt).

1.3.1 Jednoduché moduly

Jazyk NED umožňuje vytvořit si vlastní aktivní komponenty. Jednou z těchto komponent je jednoduchý modul. Jednoduchý modul se programuje v jazyce C++, využívající třídních knihoven OMNeT++. Pro jeho deklaraci se píše klíčové slovo

`simple`, v těle se nacházejí parametry a brány. Příklad jednoduchého modulu lze vidět v následujícím výpise:

```
simple Queue                          //vytvoření jednoduchého modulu
{
    parameters:                       //blok pro nastavení parametrů
        int capacity;                 //vytvoření celočíselné proměnné
        ("p=167,148");                //určení polohy
    gates:                            //blok pro nastavení bran
        input in;                     //vstupní brána in
        output out;                   //výstupní brána out
}
```

1.3.2 Složené moduly

Složený modul seskupuje ostatní moduly do jedné velké jednotky. Může obsahovat parametry a brány, stejně tak jako jednoduché moduly. Nemůže však obsahovat žádný simulační model.

Modul obsažený ve složených modulech nese název podmodul, v kódu je vždy obsažen v sekci submodules. Tělo navíc obsahuje sekci connections sloužící pro propojení jednotlivých podmodulů v modulu. Složené moduly mohou být odvozeny z jiných složených modulů díky dědičnosti. Zamezuje se tím duplikaci jednotlivých kódů a tím i šetření místa [1].

Příklad složeného modulu s podmoduly lze vidět zde:

```
module WirelessHostBase
{
    gates:
        input radioIn;
    submodules:
        tcp: TCP;
        ip: IP;
        wlan: Ieee80211;
    connections:
        tcp.ipOut --> ip.tcpIn;
        tcp.ipIn <-- ip.tcpOut;
        ip.nicOut++ --> wlan.ipIn;
        ip.nicIn++ <-- wlan.ipOut;
        wlan.radioIn <-- radioIn;
}

module WirelessHost extends WirelessHostBase
{
    submodules:
        webAgent: WebAgent;
    connections:
        webAgent.tcpOut --> tcp.appIn++;
        webAgent.tcpIn <-- tcp.appOut++;
}
```

1.3.3 Parametry a brány

Parametry jsou proměnné, které náleží jednoduchým i složeným modulům. Mohou být přiděleny v NED souboru, nebo v souboru omnetpp.ini. Parametry mohou nabývat datového typu double, int, bool, string a xml. Mohou být deklarovány volatilně, tzn. pouze na určitý čas, nebo podmínkou.

Ukázka parametrů a bran v jazyce NED lze vypozorovat z následujícího výpisu:

```
simple App
{
 parameters:
    string protocol ; // výběr protokolu
    int destAddress ; // cílová adresa
    volatile double sendInterval
 @unit (s) = default ( exponential (1s));
 // čas odesílání paketů
    volatile int packetLength @unit ( byte ) = default (100 B);
    //délka paketu
    volatile int timeToLive = default (32); // maximální počet skoků
 gates:
    input in;
    output out;
}
```

Brány jsou propojovací body modulů. OMNeT++ nabízí tři druhy bran – vstupní, výstupní a vstupně-výstupní. Brána, ať už vstupní či výstupní, může být připojena do jiné brány pomocí propojovacích kanálů. Dokonce je možné propojit výstup modulu s jeho vstupem.

1.3.4 Komunikační kanály

Komunikační kanál zapouzdřuje parametry a prostředí spojené s propojením. Kanály jsou v podstatě jednoduché moduly, které mají přiřazen jazyk C++. Praktický rozdíl v porovnání s modulem je v tom, že je vzácnost si vytvářet vlastní kanál – v NED souboru se nachází předdefinované typy.

Simulační prostředí OMNeT++ nabízí několik druhů kanálu [2]. Prvním z nich je IdealChannel. Tato ideální verze umožňuje zprávě průchod bez jakéhokoliv zpoždění nebo vedlejšího účinku.

Mezi další komunikační média je nutné také zmínit zpožďovací kanál (`DelayChannel`) a přenosový kanál (`DatarateChannel`). U zpožďovacího kanálu lze nastavit parametry zpoždění v jednotkách sekund, nebo nastavit stav disable na pravdu v případě, kdy je potřeba zahazovat veškeré zprávy na médii. Přenosový kanál obsahuje parametry přenosovou rychlost (`datarate`) a bitovou/paketovou chybovost (`ber`/`per`) pro nastavení chybovosti v intervalu <0,1>.

1.4 Simulace v OMNeT++

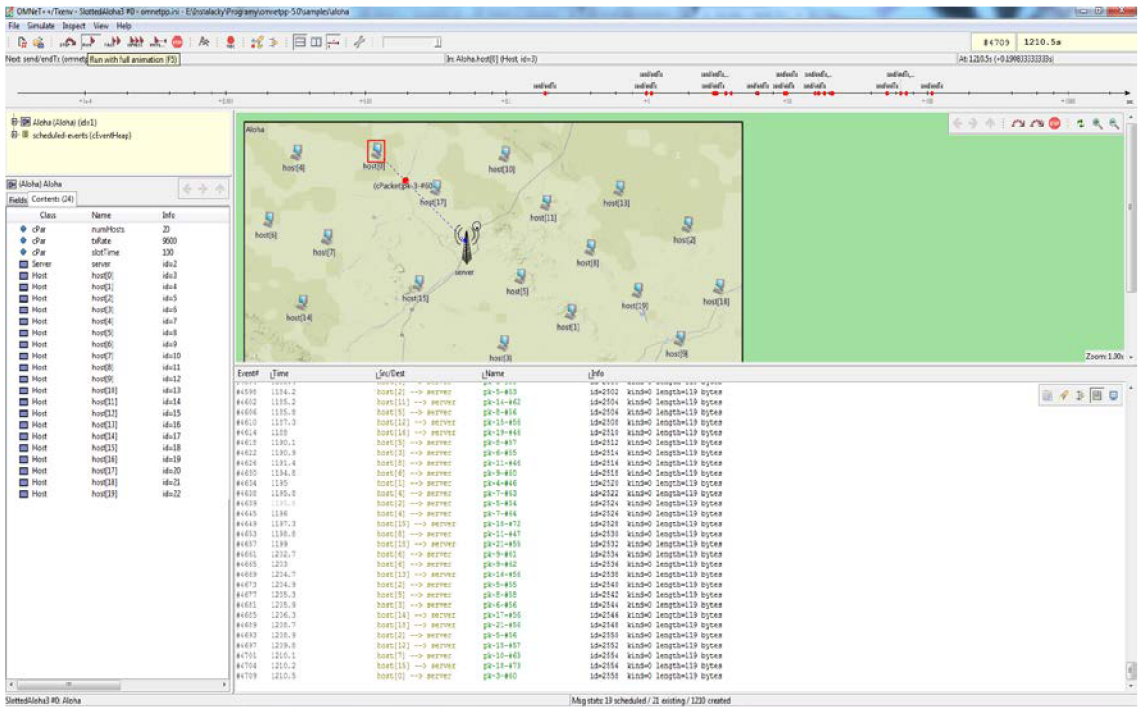
Simulace může být zkompilována v libovolném programu na libovolném stroji tak, že lze vykonat na jiných strojích, aniž by program OMNeT++ musel být přítomen. V takovém případě musí být v daném systému přítomny OMNeT++ sdílené knihovny. Při startu programu jsou nejprve přečteny všechny NED soubory obsahující topologii, poté je přečten konfigurační soubor (obvykle nazván `omnetpp.ini`).

Tento soubor obsahuje nastavení pro vykonání simulace, hodnoty modulů, parametry atd. Soubor `omnetpp.ini` je nepovinný, ulehčuje ale práci s nastavováním. Bez uvedení daného souboru by se sice simulace spustila, bylo by však potřeba nastavovat všechny parametry uvedených modulů v souboru NED ihned po spuštění simulace.

Výstup simulace je zapisován do souboru zvaného result files. Nachází se v něm výstup vektorových a skalárních souborů. OMNET++ nabízí integrované vývojové prostředí (IDE), to poskytuje bohaté prostředí pro analýzu těchto souborů.

1.5 Uživatelské rozhraní

Hlavní účel grafického uživatelského rozhraní je vytvořit vestavbu modelu viditelného pro uživatele, kontrolovat průběh simulace a případně umožnit uživateli zasáhnout do simulace změnou proměnných/objektů uvnitř modulu. Tato část je zejména důležitá pro vývojáře při ladění simulačního projektu. OMNeT++ nabízí grafické uživatelské rozhraní k demonstraci chování jednotlivých modulů.



Obrázek 3: Simulační prostředí OMNeT++ Tkenv

Obrázek 3 zobrazuje rozpoložení simulačního prostředí. Hlavní okno, nacházející se ve středu obrazovky, zobrazuje real-time simulaci NED souboru – přenášené zprávy. Tyto zprávy lze dle potřeby vyfiltrovat vhodně nastaveným filtrem. V levé části okna se nachází hierarchie vytvořené sítě. V levém dolním rohu je po označení jednotlivého prvku možné vypozorovat veškeré informace o objektu. Okno obsahuje informace o směrovacích tabulkách nebo aktivních rozhraních.

Pro sledování veškerého dění simulace slouží textové okno, na Obrázek 3 vpravo dole. Po dokončení simulace projektu uchovává informace o každé předané zprávě mezi dvěma prvky. Prozrazuje čas předání zprávy, zdroj a cíl, typ zprávy a dodatečné informace o zprávě. Poslední, horní část prostředí, slouží k jednoduchému ovládání běhu simulace.

1.5.1 Analýza výsledků

OMNeT++ poskytuje vestavěnou podporu pro zaznamenávání simulačních výsledků, skrze vektorové, nebo skalární hodnoty.

Vektorové hodnoty jsou série dat, zaznamenaných z jednoduchých modulů a kanálů. Výsledků lze využít ke zkoumání například koncového zpoždění, nebo obousměrného zpoždění paketů, zaplnění prioritních front, vytížení linky, zahazování paketů apod.

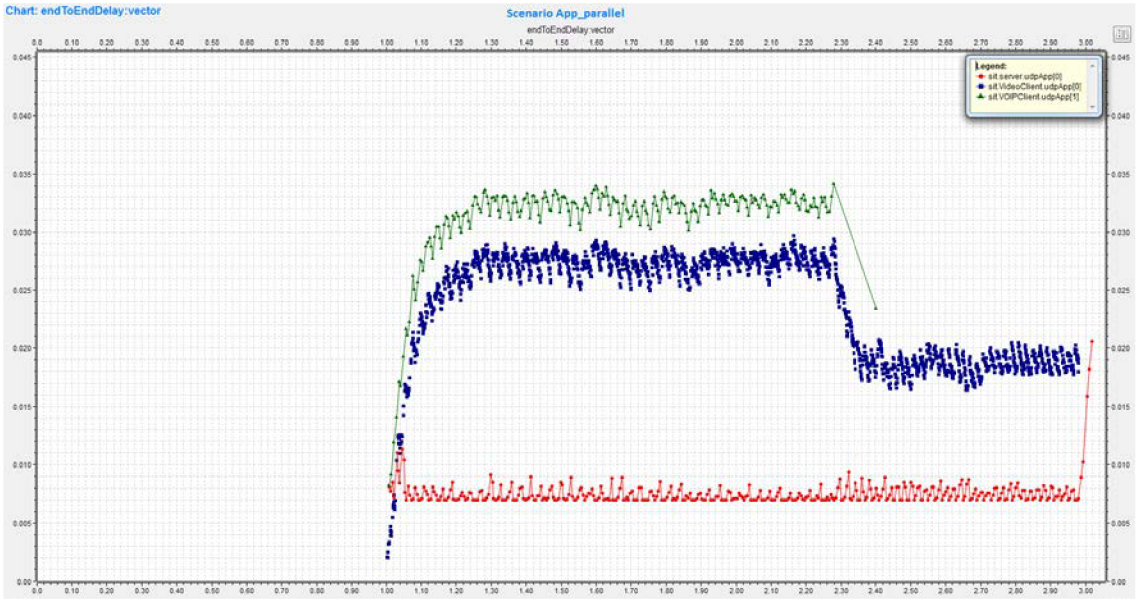
Skalární hodnoty jsou celkové výsledky, vypočteny během simulace a zaznamenány při jejím dokončení. Skalární hodnota je většinou číslo celé či desetinné, nebo se může jednat o statistický přehled složený z několika polí, jako je počet, průměr, směrodatná odchylka, součet, maximum, minimum apod.

Výsledky mohou být shromážděny a zaznamenány dvěma způsoby:

- Založeny na mechanizmu signálu, užitím deklarovaných statistik
- Přímo z C++ kódu, užitím simulačních knihoven

Pro získání výsledků je po simulaci potřeba otevřít `.anf` soubor. V nově otevřeném okně se nacházejí výsledné vektorové a skalární hodnoty, ze kterých lze později vycházet například při zobrazování grafů. Rychlou orientaci ve výsledcích umožňují filtry – module filtr zde slouží ke snadné orientaci mezi moduly, filtr statistik zase může snadno vyfiltrovat potřebné statistiky k analýze. OMNeT++ dovoluje vyexportovat hodnoty do souborového formátu CSV, umožňuje tak další manipulaci s daty.

Za zmínku také stojí možnost vykreslování výsledků do grafu. Při vhodném výběru dat je možné tak jednoduše srovnat hodnoty, které simulace poskytla. Slouží k tomu funkce plot. Na ose X se vždy nachází čas, na ose Y pak sledované hodnoty. Nabízí se také možnost zobrazení legendy. Ukázka skalárního grafu lze vidět na Obrázek 4.



Obrázek 4: Ukázka analýzy zpoždění UDP aplikací

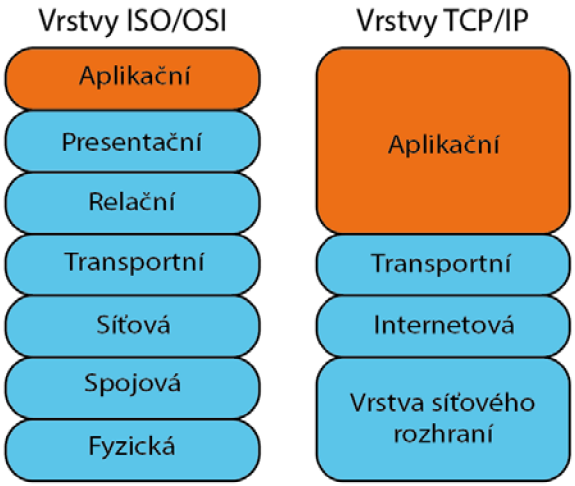
1.6 Inet framework

INET Framework je otevřená modelová sada pro OMNeT++ simulační prostředí. Obsahuje komunikační protokoly (TCP (Transmission Control Protocol), UDP (User Datagram Protocol), IPv4 (Internet Protocol v. 4), IPv6, OSPF (Open Shortest Path First), BGP (Border Gateway Protocol)), technologie linkové vrstvy (Ethernet, PPP (Point-to-Point Protocol), IEEE (Institute of Electrical and Electronics Engineers) 802. 11), podporu pro mobilitu, Diffserver (Differentiated services) a mnohem víc.

Několik jiných simulačních programů pokládá INET Framework jako základ [3]. Toto rozšíření lze buď nainstalovat přímo v prostředí OMNeT++, nebo je možné z oficiálních stránek stáhnout zdrojové soubory a ty pak nakopírovat do složky s pracovním adresářem. Po vytvoření projektu je nezbytné tomuto modelu přiřadit reference v nastavení projektu, jinak by nebylo možné INET Framework použít.

2 APLIKAČNÍ VRSTVA

Aplikační vrstva je 7. vrstvou referenčního modelu ISO/OSI (International Standards Organisation / Open System Interconnection) a 4. vrstvou modelu TCP/IP (Transmission Control Protocol/Internet Protocol). V rámci modelu TCP/IP vrstva zahrnuje vrstvu relační a mnohem důležitější vrstvu prezentační. Zpřístupňuje především informační systém do referenčních modelů [3]. Jedná se o aplikační procesy využívající přenos dat po síti ve spojení se správními funkcemi operačního systému, který tyto procesy podporuje. Vrstva také poskytuje procesům přístup ke komunikačním prostředkům. Poskytovaná služba pro aplikace může být například identifikace dvou či více komunikujících uživatelů, přenos jednotlivých zpráv, určení kvality poskytovaných služeb nebo dohoda o mechanizmu ochrany zpráv. Zařazení aplikační vrstvy v modelu TCP/IP a v modelu ISO/OSI uvádí Obrázek 5.



Obrázek 5: Zařazení aplikační vrstvy v modelu ISO/OSI a TCP/IP

Nejznámější protokoly síťové vrstvy [4]:

- **HTTP** (Hypertext Transfer Protocol) – protokol pro výměnu hypertextových dokumentů ve formátu HTML.
- **FTP** (File Transfer Protocol) – protokol pro přenos souborů v počítačové síti.
- **SMTP** (Simple Mail Transfer Protocol) – protokol pro přenos zpráv elektronické pošty (e-mailů).
- **DNS** (Domain Name System) – protokol pro vzájemný převod doménových jmen a IP adres.
- **DHCP** (Dynamic Host Configuration Protocol) – protokol pro automatickou konfiguraci IP adres na lokální síti.
- **TELET** (Telecommunication network protocol) – protokol pro realizaci vzdáleného přístupu.

- **VOIP** (Voice over Internet Protocol) – protokol pro přenos zvuku a videa

2.1 HTTP

Pro výměnu hypertextových dokumentů se zpravidla používá protokol HTTP. Jedná se o bezestavový protokol. Pro přenos většinou využívá spolehlivost doručení protokolu TCP (může využívat i protokol UDP). Funguje na způsob dotaz – odpověď, kdy klient posílá serveru dotaz a server mu odpovídá podle výsledku zpracovaného dotazu. Server používá standardně port TCP/80. Zabezpečenou verzi protokolu nahrazuje HTTPS (Hypertext Transfer Protocol Secure).

Do verze HTTP 1.0 nebyl protokol schopen uchovat stav komunikace. To znamenalo, že se se nejprve sestavilo TCP spojení, poté následoval přenos dat a nakonec se spojení ukončilo. Při potřebě přenosu více objektů se pokaždé muselo znovu navázat a ukončit spojení. Od verze 1.1 je spojení trvalé – schopnost přenosu více objektů během jednoho spojení [3].

2.2 FTP

Dalším důležitým protokolem aplikační vrstvy TCP/IP modelu je FTP protokol. Hlavním účelem protokolu je přenos souborů v sítích se stejným i rozdílným operačním systémem. Pracuje na principu klient-server a využívá spolehlivého spojení (TCP), stejně jako protokol HTTP. FTP pro přenos využívá dvě oddělené spojení – řídící (port serveru 21) a datové (port serveru 20). Nevýhodou protokolu je jeho kryptografické nezabezpečení – třetí strana může odposlechnout, nebo dokonce změnit posílaná data. Pro zabezpečení posílaných dat se používá varianta tunelování skrze protokol SSH (Secure Shell) nebo FTP s SSL/TLS (Secure Sockets Layer/Transport Layer Security) [3].

Existují dva způsoby přihlašování klienta na server. Prvním z nich je anonymní přihlášení, kdy server nevyžaduje žádnou autentizaci uživatele. Jako přihlašovací jméno je použito klíčové slovo „anonymous“ a heslo může být například emailová adresa. Druhý způsob spočívá v autorizaci uživatele pomocí uživatelského jména a hesla.

Díky řídícímu a datovému kanálu rozeznáváme u FTP dva režimy – aktivní a pasivní režim. U aktivního režimu navazuje spojení pro přenos dat server. Mezitím, co klient otevírá náhodný port (< 1023), posílá serveru příkaz PORT (informace o IP adrese a portu). Server po přijetí tohoto příkazu vytváří datové spojení z vlastního lokálního portu 20 na IP adresu a port klienta [5].

Pasivní režim využívá klient tehdy, chce-li navázat datové i řídící spojení sám. Server si tedy po zahájení spojení alokuje port pro datový kanál. Po přijetí příkazu PORT klientem klient sestavuje datové spojení. Tento režim je častější.

2.3 TELNET

Telnet představuje sadu pravidel, skrze které je možná emulace telekomunikačního terminálu v síti TCP/IP. Lze jej tedy použít pro vzdálené připojení k entitám s přiřazenou IP adresou a se spuštěným telnet serverem. Na straně serveru se využívá port 23 a jedná se o spolehlivé spojení pomocí TCP. Používá se pro přihlášení na entitu za účelem interaktivní práce, administraci či konfiguraci na serveru. Údaje autentizace uživatele při přihlašování jsou po síti přenášeny nešifrovaně – protokol není považován za bezpečný, nahrazuje jej SSH.

Důležitou součástí je podvrstva NVT (Network Virtual Terminal), která představuje síťový virtuální simulátor zabývající se prezentací dat. Využívá 7b kódů pro zaslání standardních znaků z tabulky ASCII (American Standard Code for Information Interchange). Obsahuje také příkazy pro řízení spojení – identifikace je dána nejvýznamnějším bitem, který je nastaven na hodnotu 1. Podvrstva NVT se také v omezené míře používá u protokolů FTP, POP3 (Post Office Protocol 3) či SMTP (Simple Mail Transfer Protocol) [5].

2.4 VoIP

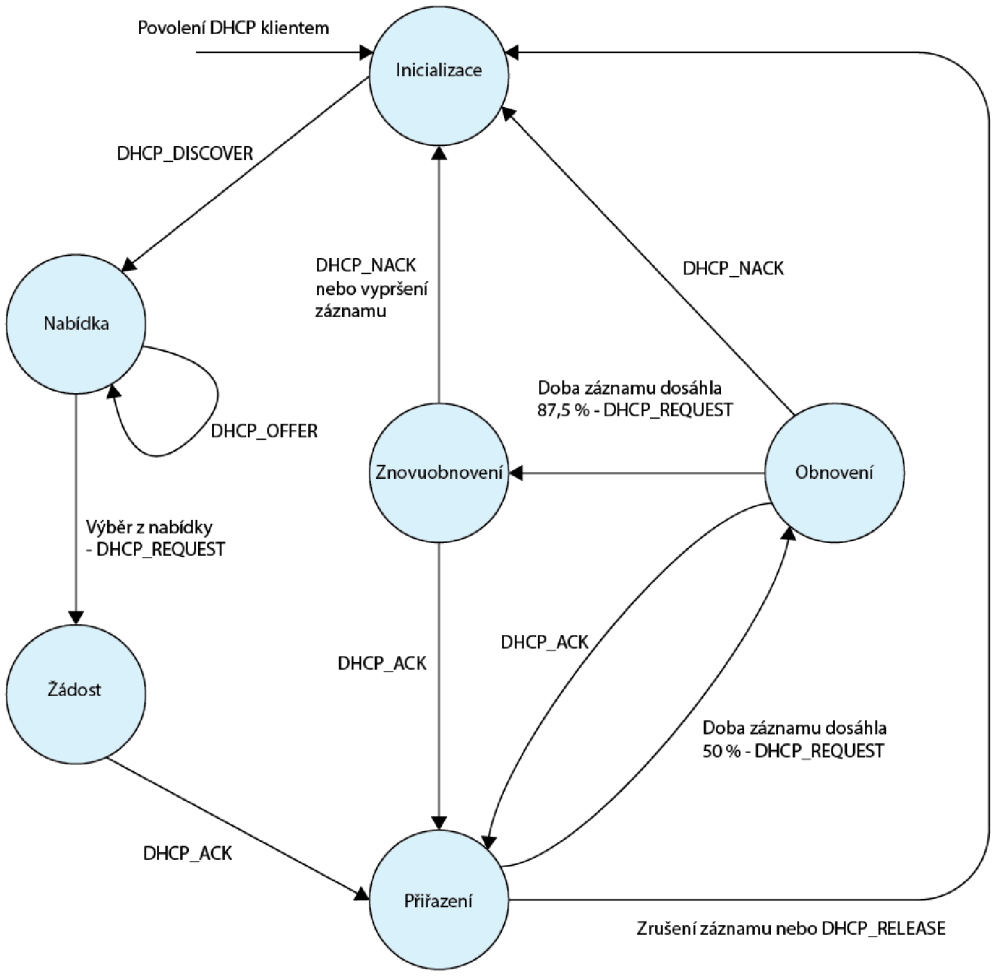
VoIP (Voice over Internet Protocol) je technologie využívána pro přenos digitalizovaného lidského hlasu. Pro přenos hlasu (videa) se využívá IP protokol, nacházející se na 3. vrstvě a UDP protokol, nacházející se na 4. vrstvě modelu ISO/OSI. Lze také aplikovat v sítích VPN (Virtual Private Network). Před odesláním paketu se nejprve musí provést kódování pomocí kodéru/dekodéru – to zajistí digitalizaci analogového signálu. Kromě komprese a dekomprese signálu také zahrnuje samotný přenos paketů. Pro srozumitelnost a spolehlivost VoIP spojení je nutné zajistit kvalitu služeb QoS (Quality of Services) [6].

Nepříznivým jevem technologie VoIP je tzv. proměnlivé zpoždění (jitter). Ten nepříznivě ovlivňuje služby vyžadující zpracování dat. Odstranění jitteru je možné pomocí vyrovnávací paměti, to však vede k navyšování celkového zpoždění dat [5].

Pro správnou funkci přenosu hlasu či videa je využíván protokol RTP (Real-time Transport Protocol). K navázání spojení je zase potřeba využít protokol SIP (Session Initiation Protocol).

2.5 DHCP

DHCP (Dynamic host configuration protocol) je standardní aplikační protokol využívaný v IP sítích. Základem protokolu je DHCP server, který poskytuje klientům dynamické nastavení parametrů sítě (IP adresu, masku sítě, výchozí bránu, DNS server atp.). DHCP protokol využívá transportní protokol pro nespolehlivý přenos (UDP) [5]. Klient komunikuje s DHCP serverem pomocí portu 68, server naslouchá na portu 67. DHCP je stavový protokol, jeho stavový diagram lze vidět na Obrázek 6.



Obrázek 6: Stavový diagram protokolu DHCP

Zmíněný protokol existuje ve dvou variantách – varianta pro IPv4 sítě (DHCPv4) a pro sítě IPv6 (DHCPv6). Kromě adresace se tyto dvě sady pravidel liší například v distribuci jednotlivých zpráv nebo nutnými metodami pro přidělení konfigurace parametrů sítě. Prvotní DHCPv4 však vzniklo ze staršího protokolu pro automatickou konfiguraci – BOOTP (Bootstrap Protocol) [7]. V bezdrátových sítích s častým přístupem velké řady klientů je ovšem BOOTP nevhodný, protože přidělovaný záznam (adresa, maska, brána) je časově neomezen, zatímco u protokolu DHCP je doba záznamu stanovena správcem daného serveru. Po vypršení doby je daný záznam neplatný, přidělené konfigurace jsou smazány jak v tabulce serveru, tak v konfiguraci klienta a klient si musí o údaje zažádat znovu. Dále se popis zabývá pouze protokolem DHCPv4.

Pro úspěšné předání konfiguračních parametrů musí nejprve klient projít skrze určité stavy (viz stavový diagram). Nejprve pomocí všesměrové zprávy posílá koncová stanice metodu DHCP_Discover pro zjištění, zda-li se v síti nachází DHCP server. Pokud je tomu tak, server odpovídá metodou DHCP_Offer s nabízenou možností IP adres (zpráva může být šířena na přímo, nebo všesměrově, záleží na nastaveném příznaku v těle zprávy). Klient si po přijetí zprávy jednu z nabízených adres vybere, zpravidla je to buď první, nebo poslední nabízená adresa, a posílá serveru zprávu DHCP_Request. Poslední zprávou DHCP_ACK server klientu dovoluje používat

konkrétní vybranou adresu [8].

Kromě nabízených metod definuje protokol ještě řadu zpráv: zpráva DHCP_Nack, negativní potvrzení, se odesílá při vypršení klientova záznamu. DHCP_Release slouží k dobrovolnému uvolnění záznamu klientem – například při napsání `ipconfig /release` do příkazové řádky.

3 ADRESACE A PODSÍŤOVÁNÍ V IPV4 SÍTÍCH

3.1 IP adresa

Adresa protokolu IP verze 4 se skládá ze čtyř bajtů (32 bitů). Slouží k jedinečné identifikaci síťového rozhraní konkrétního zařízení v internetu (počítače, rozhraní směrovače, chytrých zařízení atd.). IP adresa se zapisuje notací, kde se jednotlivé bajty mezi sebou oddělují tečkou. Z výběru tří číslicových soustav (dvojková, desítková a šestnáctková) se pro zápis nejvíce uchytila desítková notace. Počítač adresami pracuje v binární reprezentaci – celá kladná čísla v dvojkové soustavě [9].

Adresa se skládá ze dvou částí (složek) – adresa sítě a adresa počítače v síti. Aby bylo znatelné rozlišení těchto dvou složek, zavedlo se tzv. třídní adresování, kdy se adresy dělí do pěti tříd:

- Třída A – Nejvyšší bit prvního oktetu má hodnotu 0, zbylých 7 bitu tvoří adresu sítě a zbytek je určen pro adresu počítače.
- Třída B – Nejvyšší dva bity prvního oktetu mají binární hodnotu 10, zbylých 6 bitů a následující druhý bajt je určen pro adresy sítě. Zbytek je vyhrazen koncovým stanicím.
- Třída C – První tři bity prvního oktetu mají binární hodnotu 110, adresa sítě zasahuje až po konec třetího oktetu a klientskou část tvoří poslední bajt.
- Třída D – První čtyři bity 1110, využití pro skupinové (multicast) adresy.
- Třída E – Prvních pět bitů 11110, pro experimentální účely.

Rozdělení adres do tříd zobrazuje Tabulka 1 [10]:

Tabulka 1: Třídní dělení adresního prostoru IP verze 4

	IPv4 adresa			
Třída	**1. oktet adresy**	**2. oktet adresy**	**3. oktet adresy**	**4. oktet adresy**
A	Adresa sítě (0-127)	Adresa počítače		
B	Adresa sítě (128-191)		Adresa počítače	
C	Adresa sítě (192-223)			Adresa počítače
D	Skupinové adresy (224-239)			
E	Experimentální adresy (240-225)			

V dřívějších dobách se dílčí adresy přidělovaly skutečně podle tříd, nicméně se tento koncept ukázal jako problematický. Problémem bylo neefektivní rozdělení adres spočívajícím především v nedostatku adres koncových stanic ve třídě C a naopak téměř nevyčerpatelnému množství adres uzlů ve třídě A. Postupem času se tedy přešlo na přidělování rozsahů prakticky nezávislých na původních třídách, vzniká pojem podsíťování [3].

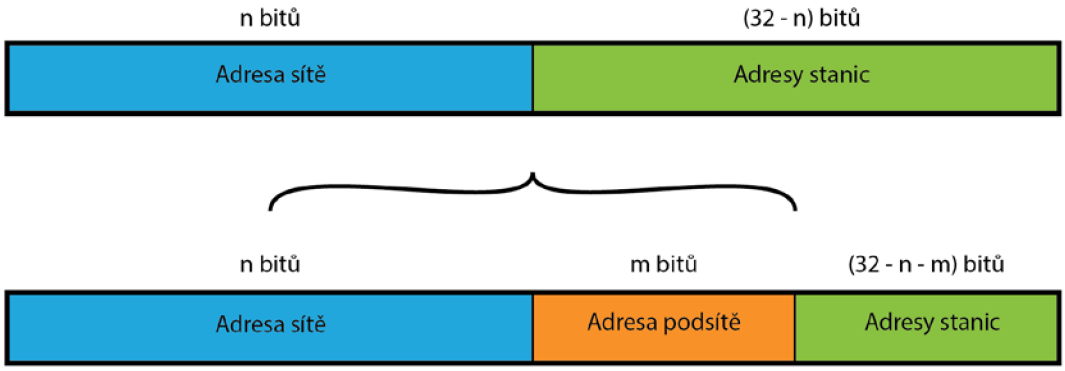
3.2 Maska sítě

Aby bylo možné rozlišit část adresy sítě od adres koncových stanic, vzniká tzv. maska sítě. Jedná se, stejně jako u IP adresy, o čtyřbajtové číslo. Toto číslo vyjádřené v dvojkové soustavě má v bitech určujících adresu sítě samé jedničky a v ostatních bitech samé nuly. Na rozdíl od běžných unicastových adres lze masku zapisovat v prefixovém tvaru – podle počtu nacházejících se jedniček v masce sítě. Například maska sítě 255.255.255.192 má v prvních třech oktetech všechny bity binární „1“, ve čtvrtém oktetu se nachází ještě dvě bitové úrovně „1“ a zbytek masky tvoří binární „0“. Po sečtení všech jedniček vzniká zápis masky v prefixovém tvaru /26.

Podle masky sítě lze také určit adresu sítě, všesměrovou adresa a rozsah adres. Adresa sítě je první možná adresa v konkrétním adresním rozsahu. Tato adresa zastupuje daný adresový prostor, proto nemůže být přiřazena konkrétní stanici. Stejně tak poslední adresa rozsahu – všesměrová adresa – nemůže být přiřazena koncové stanici. Nachází využití při odesílání zpráv (paketů) všem stanicím nacházejícím se v síti. Všechny adresy vyskytující se mezi první a poslední adresou mohou být přiřazeny konkrétnímu síťovému rozhraní koncové stanice. Při výpočtu adres stanic z dříve zmíněné masky zapsané v prefixovém tvaru /26 se dá postupovat tak, že se vezme rozdíl odečtení prefixu od 32 bitů, tento rozdíl utvoří exponent druhé mocniny a od výsledku se nakonec odečtou dvě adresy – adresa sítě a všesměrová adresa. Výsledná rovnice je $2^{(32\text{-prefix})} - 2$ = přidělitelný rozsah adres koncovým stanicím [8].

3.3 Podsíťování

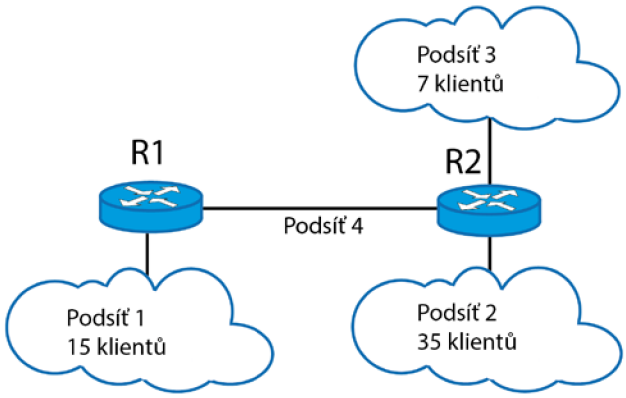
Pojem podsíťování (subnetting) rozšiřuje původně dvousložkovou adresu, sestávající se z adresy sítě a adresy stanice na trojsložkovou adresu, kde ještě přibývá adresa podsítě. Adresa podsítě vzniká rozdělením části původně určené k adresaci stanice na dvě části: adresu podsítě a adresu stanice. Rozšíření zobrazuje Obrázek 7:



Obrázek 7: Trojsložkové rozdělení adresy

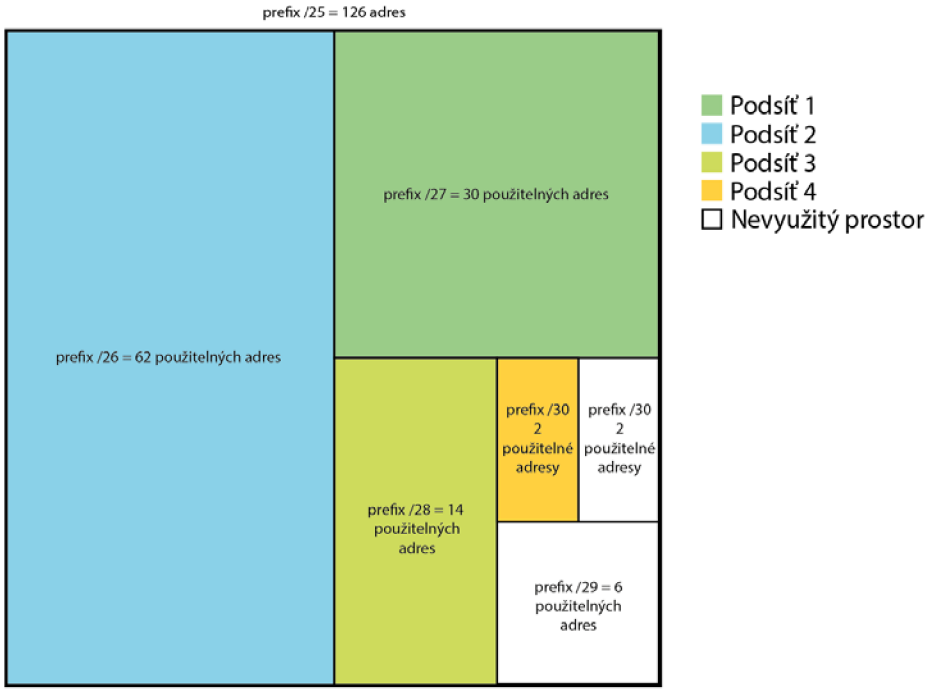
Podsíťováním je možné docílit rozšíření přiděleného adresového prostoru na více dílčích částí s menším počtem koncových stanic. Přináší to výhodu organizace i zabezpečení. Dochází také k efektivnějšímu využití adres [3].

Jako příklad poslouží tři sítě, první o 15 klientech, druhá o 35 klientech a třetí o 7 klientech. Přidělená adresa pro tyto 3 sítě je 180.30.5.0/25. Zapojení sítě se nachází na následujícím Obrázek 8:



Obrázek 8: Příklad podsítí

Jako podsíť je také nezbytné uvažovat trasu mezi směrovači R1 a R2, pro obě síťová rozhraní je nutno přidělit celkem dvě adresy. Pomocí masky se l rozhodnout, kolik podsítí celkem vytvořit. Maska /25 říká, že je možné vytvořit jednu síť se 126 koncovými stanicemi, 2 sítě se 62 koncovými stanicemi, 4 sítě se 30 koncovými stanicemi atd. Pro podsíť 2 je vyhovující maska /26, pro podsíť 1 /27, podsíť 3 /28 a pro podsíť 4 pak maska /30. Rozdělení podsítí znázorňuje Obrázek 9:



Obrázek 9: Přidělení adresního prostoru podsítím

Z hlediska efektivního rozdělení adresního rozsahu se použije podsíťování s proměnnou maskou. Tím se při rozdělení masky na dvě /26 docílí, že druhá nevyužitá část /26 se ještě rozdělí na menší podsítě. Výsledkem pak je rozdělení konkrétního adresního rozsahu přesně podle potřeb.

4 TEORETICKÝ NÁVRH ZADÁNÍ

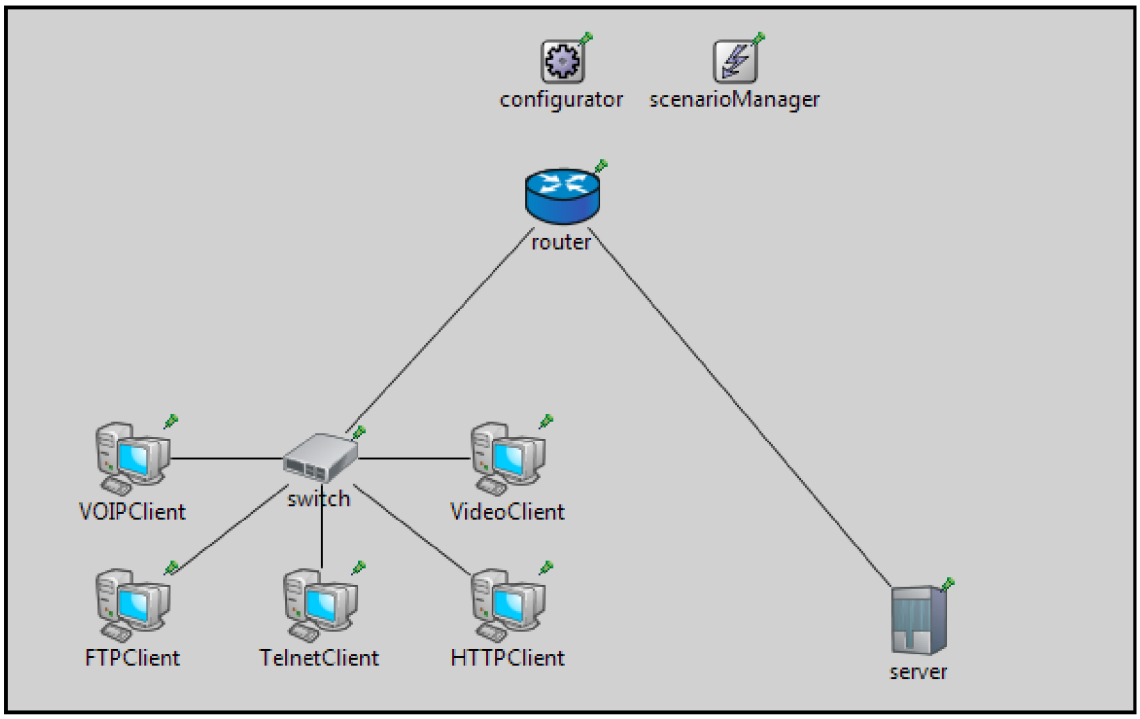
Tato kapitola si klade za cíl stručně shrnout cíle, kterých má být při vypracovávání navržených laboratorních úloh dosaženo.

4.1 Laboratorní úloha číslo 1 – Aplikační protokoly

Cílem této laboratorní úlohy je srovnání aplikačních protokolů HTTP, FTP, Telnet a VoIP. K simulaci poslouží jednoduchá topologie se dvěma směrovači, pěti klienty a jedním serverem, který v případě VOIP vystupuje jako klient. Zapojení topologie lze vidět na Obrázek 10.

V první části úlohy dojde k vytvoření sítě pomocí souboru NED a obecnému nastavení konfiguračního souboru. Dále konfigurační soubor zahrnuje 5 scénářů. První scénář popisuje chod jednotlivých aplikací za sebou. V druhém scénáři se sleduje chod všech aplikací v jednom čase. Nastavení paketové chybovosti sleduje další scénář. Čtvrtá část obsahuje simulaci výpadku linky – sledování chování aplikací během výpadku. Poslední část se zabývá přetížením linky.

Pro simulaci výše zmíněného nastavení se použije simulační nástroj Tkenv. Při analýze výsledků lze pak vysledovat rozdílné hodnoty v obousměrném zpoždění RTT (Round-Trip Time), koncové zpoždění (End To End Delay), využití linek (Utilization) a odeslané a přijaté pakety všech aplikací.



Obrázek 10: Topologie sítě pro simulaci aplikačních protokolů

4.2 Laboratorní úloha číslo 2 - Dynamické a statické adresování

Tato úloha si klade za cíl přiblížit problematiku adresování síťové topologie nejdříve pomocí protokolu pro dynamickou adresaci DHCP, následně pomocí statické adresace. Součástí úlohy je také počítání adres a vhodné přidělení adresního rozsahu pro předem stanovené podsítě. K simulaci poslouží celkem tři topologie.

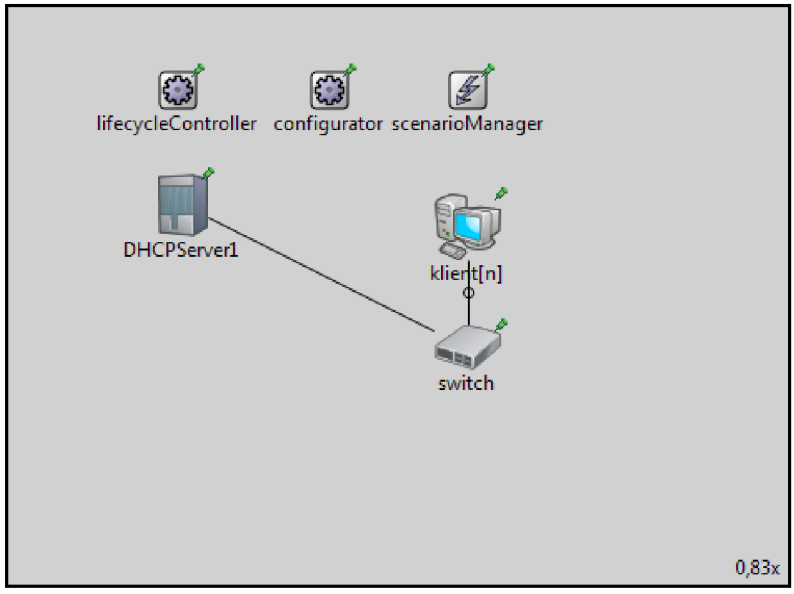
Počáteční úlohou je vytvoření nového projektu v simulačním prostředí OMNeT++. Dále se jedná o vytvoření a nastavení souboru NED pro definici zapojení konkrétních topologií. O nastavení simulace se postará konfigurační soubor .ini, ve kterém se bude nacházet celkem 5 scénářů. Do úlohy spadá také několik .xml skriptovacích souborů.

První část se skládá celkem ze čtyř scénářů. Scénář 1 až 3 (topologie na Obrázek 11) zobrazuje výměnu zpráv mezi klienty a DHCP serverem, poukazuje na vypršení doby záznamu a sleduje situaci při ztrátě záznamu na straně serveru. Čtvrtý scénář poukazuje na problém smyček a možný vznik všesměrových bouří (topologie na Obrázek 12).

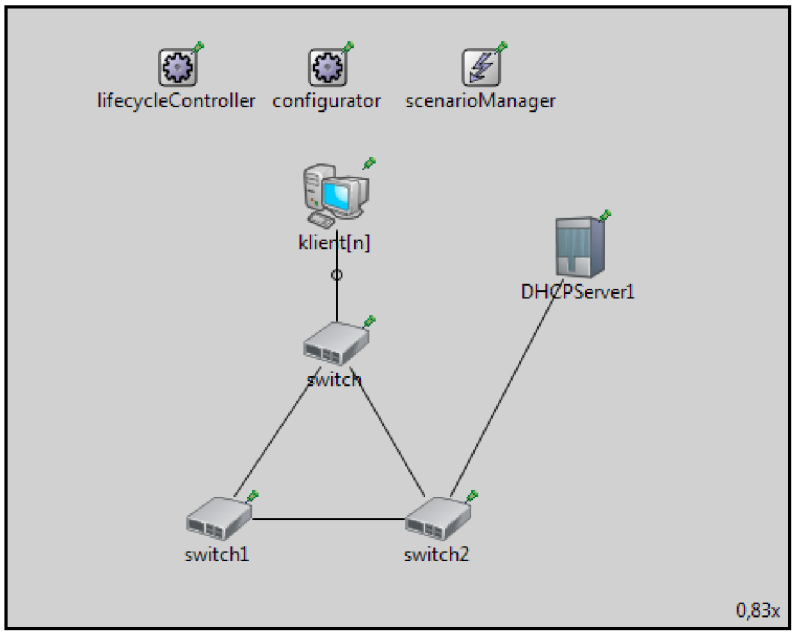
Další část se věnuje počítání podsítí, nejprve neefektivní metodou s použitím classful adresování a možným poukázáním na plýtvání adresového rozsahu, poté pomocí efektivní metody proměnlivé masky (VLSM – Variable Length Subnet Mask).

Poslední scénář se věnuje přímo adresování jednotlivých podsítí v simulačním prostředí (Obrázek 13). K adresování je využit skriptovací soubor.

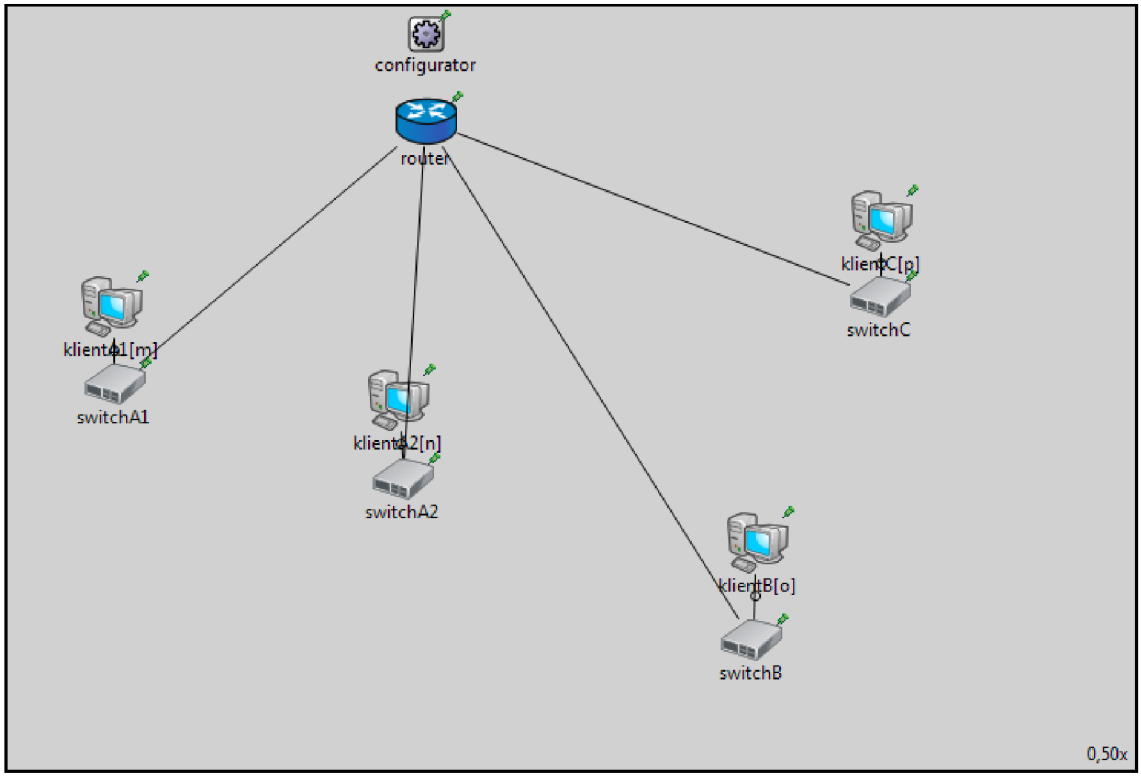
Úloha také zahrnuje samostatný úkol a sadu otázek, na které lze při vypracování najít odpověď.



Obrázek 11: Zapojení topologie s DHCP serverem



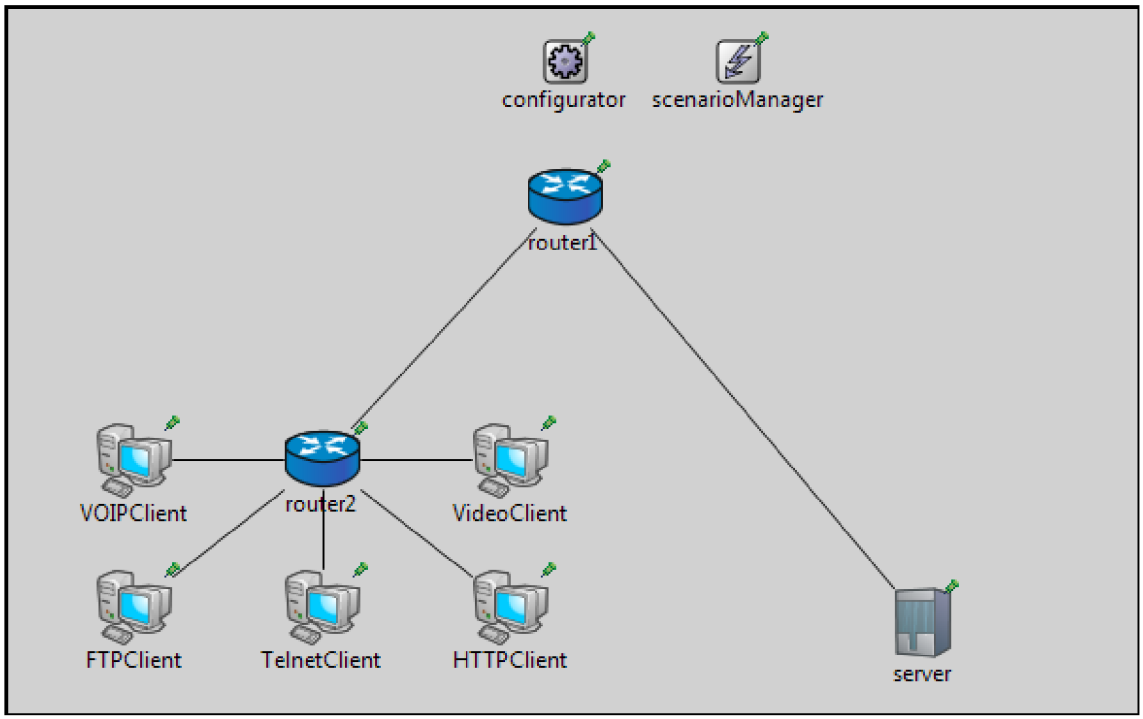
Obrázek 12: Zapojení topologie s DHCP serverem a všesměrovou smyčkou



Obrázek 13: Zapojení topologie pro statickou adresaci

5 LABORATORNÍ ÚLOHA ČÍSLO 1 – APLIKAČNÍ PROTOKOLY

Cílem této laboratorní úlohy je srovnání aplikačních protokolů HTTP, FTP, Telnet a VoIP. K simulaci poslouží jednoduchá síť s 5 klienty generujícími aplikační provoz, dvěma směrovači a jedním aplikačním serverem. Blíže se sleduje provoz při nesouběžném i souběžném toku dat dílčích aplikací, chování přenosu při zvýšené chybovosti linky a reakce aplikací na několikasekundový výpadek linky. Zapojení topologie sítě lze vidět na Obrázek 14.



Obrázek 14: Zapojení sítě

5.1 Teoretický úvod

Aplikační vrstva je 4. vrstvou modelu TCP/IP a 7. vrstvou modelu ISO/OSI. Nachází se na vrcholu obou modelů a zahrnuje komunikační funkce, které jsou specifické pro konkrétní aplikační procesy. Dané aplikace jsou závislé na aplikační vrstvě. Vrstva poskytuje přístup procesům ke komunikačním prostředkům.

Mezi nejznámější protokoly aplikační vrstvy patří HTTP (Hypertext Transfer Protocol), FTP (File Transfer protocol), DNS (Domain Name Systém), SMTP (Simple mail Transfer Protocol), DHCP (Dynamic Host Configuration Protocol) a Telnet (Telecommunication network protocol).

HTTP se používá pro výměnu hypertextových dokumentů a pro přístup na webové služby. Pro přenos využívá port 80 na straně serveru. Další protokol aplikační vrstvy, FTP, je jeden z nejstarších protokolů, využití nachází především při přenosu souborů skrze komunikační síť. K přenosu využívá dvě oddělená spojení – spojení datové (port 20) a spojení řídící (port 21). Telnet slouží pro přístup ke vzdálenému zařízení, dnes jej nahrazuje protokol SSH, využívá port 23 na straně serveru. Všechny tyto protokoly využívají spojově orientované spojení, tedy protokol transportní vrstvy TCP.

V úloze hrají důležitou roli také aplikační protokoly transportní vrstvy UDP, konkrétně protokol VoIP (Voice over Internet Protocol), který se používá pro přenos hlasu přes komunikační síť. Setkáme se i s generováním provozu video streaming.

5.2 Postup řešení

5.2.1 Založení nového projektu

Spustíme program **`omnetpp.exe`**. Pokud se objeví nabídka pro vybrání pracovního adresáře, tento formulář potvrdíme beze změny (vybraná cesta musí být: omnetpp-5.0\samples), jinak vytvořený projekt nebude fungovat.

K vytvoření nového projektu vybereme z nabídky **File » New » OMNeT++ Project…**. Projekt pojmenujeme VUT loginem, dále stiskneme **Next » Empty project » Finish**. V okně Project Explorer se vytvořila námi pojmenovaná složka, ve které budou uloženy všechny části projektu. Nejprve je zapotřebí přiřadit k projektu odkazy na zdrojové soubory, se kterými se bude dále pracovat. To provedeme kliknutím pravým tlačítkem na vytvořenou složku » **Properties » Project References »** vybereme položku **inet » OK**.

Pro vytvoření projektu jsou zapotřebí dva základní soubory. Network Description File, s příponou `.ned`, který definuje topologii sítě, použité technologie a síťové prvky. Druhým je Initialization File, přípona `.ini`, který určuje průběh a parametry simulace.

Klikneme pravým tlačítkem myši na složku a vybereme **New » Network Description File (NED)**, pojmenujeme ji **Network.ned » Next » Empty NED File » Finish**.

V souboru NED lze pracovat ve dvou režimech, v grafickém nebo kódovém. Mezi nimi se lze přepínat v levém dolním rohu volbou Design/Source. Jsou navzájem propojené, pokud tedy přidáme prvky sítě z palety, vytvoří se k nim patřičný kód a naopak vložením kódu lze vytvářet celou síť.

5.2.2 Vypracování 1. část – Nesouběžný běh aplikací

Nejprve musíme importovat do projektu prvky, se kterými se bude pracovat. Do souboru `Network.ned` vložíme následující kód:

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

```
import ned.DatarateChannel;
```

Dále vytvoříme síť pojmenovanou AppNetwork.

```
network AppNetwork
{
    parameters:
        @display("bgb=600,374");
    submodules:
        router1: Router {
            parameters:
                @display("p=296,100;i=abstract/router");
            gates:
                pppg[1];
                ethg[1];
        }
        router2: Router {
            parameters:
                @display("p=167,239");
            gates:
                pppg[1];
                ethg[5];
        }
        FTPClient: StandardHost {
            parameters:
                @display("p=67,318");
            gates:
                ethg[1];
        }
        TelnetClient: StandardHost {
            parameters:
                @display("p=167,318");
            gates:
                ethg[1];
        }
        VOIPClient: StandardHost {
                parameters:
                    @display("p=67,240");
                gates:
                    ethg[1];
            }
        HTTPClient: StandardHost {
                parameters:
                    @display("p=267,318");
                gates:
                    ethg[1];
            }
        VideoClient: StandardHost {
                parameters:
                    @display("p=267,240");
                gates:
                    ethg[1];
            }
```

```
    server: StandardHost {
                parameters:
                    @display("p=486,326;i=device/server");
                gates:
                    ethg[1];
            }
            configurator: IPv4NetworkConfigurator {
                @display("p=296,28");
        }
```

Tímto jsme vytvořili síť o celkem pěti klientech, dvou směrovačích a serveru. Dále provedeme zapojení topologie v těle `connections` a definujeme parametry přenosového média. Jako použitá technologie pro propojení klientů poslouží Ethernet o přenosové rychlosti 10 Mb, mezi směrovači vytvoříme linku ppp, také s přenosovou rychlostí 10 Mb.

```
    scenarioManager: ScenarioManager {
                @display("p=388,28");
            }
        connections:
            router1.ethg++ <--> NormalPath <--> server.ethg++;
            router1.pppg++ <--> ppp <--> router2.pppg++;
            router2.ethg++ <--> NormalPath <--> FTPClient.ethg++;
            router2.ethg++ <--> NormalPath <--> HTTPClient.ethg++;
            router2.ethg++ <--> NormalPath <--> VOIPClient.ethg++;
            router2.ethg++ <--> NormalPath <--> TelnetClient.ethg++;
            router2.ethg++ <--> NormalPath <--> VideoClient.ethg++;
    }

    channel ppp extends ThruputMeteringChannel
    {
        parameters:

            double n;
            datarate = 10Mbps;
            delay = 0.01ms;
            ber = 0;
            per = n;
    }

    channel NormalPath extends DatarateChannel
    {
        parameters:

            datarate = 10Mbps;
            delay = 0.01ms;
            ber = 0;
            per = 0;
}
```

Za povšimnutí zde stojí vlastní proměnná `n` s plovoucí řadovou čárkou, kterou budeme v pozdější části úlohy nastavovat paketovou chybovost přenosového média pro všechny cesty, které přenosové médium využívají. Hodnotu proměnné budeme měnit v souboru `omnet.ini`.

Tímto je vytvořená síť kompletně zprovozněna k provádění simulací, nyní je zapotřebí určit parametry a průběh simulace. Jelikož soubor NED je statický, je vhodné v něm definovat pouze základní parametry, které nechceme dále měnit ani nijak upravovat. Pro nastavení průběhu simulace a síťového provozu slouží konfigurační soubor s příponou `.ini`. Spuštěním tohoto souboru zahájíme simulaci.

Do složky s projektem vytvoříme nový soubor, **New » Initialization File (ini) »** můžete ponechat název **omnetpp.ini » Next » Empty Ini File » Finish**.

Pro potřeby simulace je možné v jednom souboru vytvářet několik různých konfigurací, k vybrání požadované budeme vyzváni po spuštění simulace. Pro vytvoření nové konfigurace se používá syntaxe [Config nazev_konfigurace] zapsaná v hranatých závorkách. Hlavní částí je sekce [General], která obsahuje nastavení společná pro všechny vytvořené konfigurace, jejím nastavením začneme.

Do sekce General vložíme následující kód, který nastavuje jako simulovanou síť námi vytvořenou, povolí výpis problému při chybně nastavené simulaci, nastaví se cesta k potřebným doplňkům a omezí se maximální trvání simulace na 100 sekund.

```
[General]
network = AppNetwork
debug-on-errors = true
tkenv-plugin-path = ../inet/etc/plugins
sim-time-limit = 100s
**.server.numTcpApps = 3
**.server.numUdpApps = 3
```

Na serveru vytváříme 3 TCP a 3 UDP aplikace, které budeme postupně definovat. Server bude potvrzovat provoz HTTP a Telnet klientům, klientovi FTP bude posílat soubor. Mezi VOIP klientem a serverem bude provoz probíhat obousměrně, proto bude jeden datový tok od klienta k serveru a druhý datový tok od serveru ke klientovi. Poslední aplikace bude stream videa od serveru k video klientovi, kdy si klient zažádá o spuštění streamu a server mu následně začne požadovaný stream vysílat.

K rozlišení TCP aplikací využijeme možnost variability nastavení parametru transportní vrstvy mss (Maximal Segment Size). Pomocí něj změníme výslednou velikost TCP segmentu a tím odlišíme jednotlivé aplikace. Aplikace TELNET bude mít velikost mss 450 B, HTTP zase 350 B. Protokol pro přenos souborů necháme na výchozí hodnotě (536 B).

Další nastavení se týká aplikací. Všechny tyto konfigurace stále vkládáme do sekce [General].

```
#TELNET#####################################################
**.TelnetClient.numTcpApps = 1
**.TelnetClient.tcpApp[0].typename = "TCPBasicClientApp"
**.TelnetClient.tcpApp[0].connectAddress = "server"
**.TelnetClient.tcpApp[0].connectPort = 23
**.TelnetClient.tcpApp[0].requestLength = 700KiB
**.TelnetClient.tcpApp[0].numRequestsPerSession = 1
**.TelnetClient.tcpApp[0].thinkTime = 0.001s
**.TelnetClient.tcpApp[0].idleInterval = 0.001s
**.TelnetClient.tcp.mss = 450
```

```
**.server.tcpApp[0].typename = "TCPGenericSrvApp"
**.server.tcpApp[0].localPort = 23
#VOIP#########################################################
**.VOIPClient.numUdpApps = 2
**.VOIPClient.udpApp[0].typename = "SimpleVoIPSender"
**.VOIPClient.udpApp[0].destAddress = "server"
**.VOIPClient.udpApp[0].destPort = 2000
**.VOIPClient.udpApp[0].talkPacketSize = 650B
**.VOIPClient.udpApp[0].packetizationInterval = 5ms
**.VOIPClient.udpApp[1].typename = "SimpleVoIPReceiver"
**.VOIPClient.udpApp[1].localPort = 2001

**.server.udpApp[0].typename = "SimpleVoIPReceiver"
**.server.udpApp[0].localPort = 2000
**.server.udpApp[1].typename = "SimpleVoIPSender"
**.server.udpApp[1].destAddress = "VOIPClient"
**.server.udpApp[1].destPort = 2001
**.server.udpApp[1].talkPacketSize = 800B
**.server.udpApp[1].packetizationInterval = 5ms
#HTTP#########################################
**.HTTPClient.numTcpApps = 1
**.HTTPClient.tcpApp[0].typename = "TCPBasicClientApp"
**.HTTPClient.tcpApp[0].connectAddress = "server"
**.HTTPClient.tcpApp[0].connectPort = 80
**.HTTPClient.tcpApp[0].thinkTime = 0.001s
**.HTTPClient.tcpApp[0].idleInterval = 0.001s
**.HTTPClient.tcpApp[0].requestLength = 1MiB
**.HTTPClient.tcpApp[0].numRequestsPerSession = 3
**.HTTPClient.tcp.mss = 350

**.server.tcpApp[1].typename = "TCPGenericSrvApp"
**.server.tcpApp[1].localPort = 80
#VIDEO#########################################
**.VideoClient.numUdpApps = 1
**.VideoClient.udpApp[0].typename = "UDPVideoStreamCli"
**.VideoClient.udpApp[0].localPort = 1000
**.VideoClient.udpApp[0].serverAddress = "server"
**.VideoClient.udpApp[0].serverPort = 1001

**.server.numUdpApps = 1
**.server.udpApp[2].typename = "UDPVideoStreamSvr"
**.server.udpApp[2].localPort = 1001
**.server.udpApp[2].sendInterval = 1ms
**.server.udpApp[2].packetLen = uniform(400B,800B)
**.server.udpApp[2].videoSize = 1150KiB
```

```
#FTP#####################################
**.server.tcpApp[2].typename = "TCPBasicClientApp"
**.server.tcpApp[2].connectAddress = "FTPClient"
**.server.tcpApp[2].connectPort = 20
**.server.tcpApp[2].thinkTime = 0.001s
**.server.tcpApp[2].idleInterval = 0.001s
**.server.tcpApp[2].requestLength = 1.5MiB
**.server.tcpApp[2].numRequestsPerSession = 1

**.FTPClient.numTcpApps = 1
**.FTPClient.tcpApp[0].typename = "TCPGenericSrvApp"
**.FTPClient.tcpApp[0].localPort = 20
```

Všem vytvořeným TCP aplikacím přiřazujeme typ aplikace, připojovací adresu – jméno modulu a port aplikace. Parametr `requestLength` nastavuje velikost dat, kterou aplikace během relace odešle. Pro každou aplikaci je vytvořena serverová aplikace pro zpracování daných požadavků. K VoIP komunikaci se používá předdefinovaná aplikace `SimpleVoIPSender` pro odesílatele a `SimpleVoIPReceiver` pro příjemce hovoru. Modul s nadefinovanou aplikací `UDPVideoStreamCli` nejprve zažádá server o video, ten následně začne vysílat videostream.

Nyní přejdeme k vytvoření prvního scénáře **[config App_gradually]**, ve kterém jen stačí nastavit časy začátku a konce aplikací a nastavit parametr chybovosti. Chod aplikací je v tomto případě postupný – žádná aplikace, kromě VOIP, by se neměla s jinou překrývat. Parametr `n` zatím ponecháme nastaven na nule.

```
[Config App_gradually]
**.TelnetClient.tcpApp[0].startTime = 0s
**.TelnetClient.tcpApp[0].stopTime = 2s

**.server.udpApp[1].startTime = 2s
**.server.udpApp[1].stopTime = 4s
**.VOIPClient.udpApp[0].startTime = 2s
**.VOIPClient.udpApp[0].stopTime = 4s

**.HTTPClient.tcpApp[0].startTime = 4s
**.HTTPClient.tcpApp[0].stopTime = 6s

**.VideoClient.udpApp[0].startTime = 6s

**.server.tcpApp[2].startTime = 8s
**.server.tcpApp[2].stopTime = 10s
**.n = 0
```

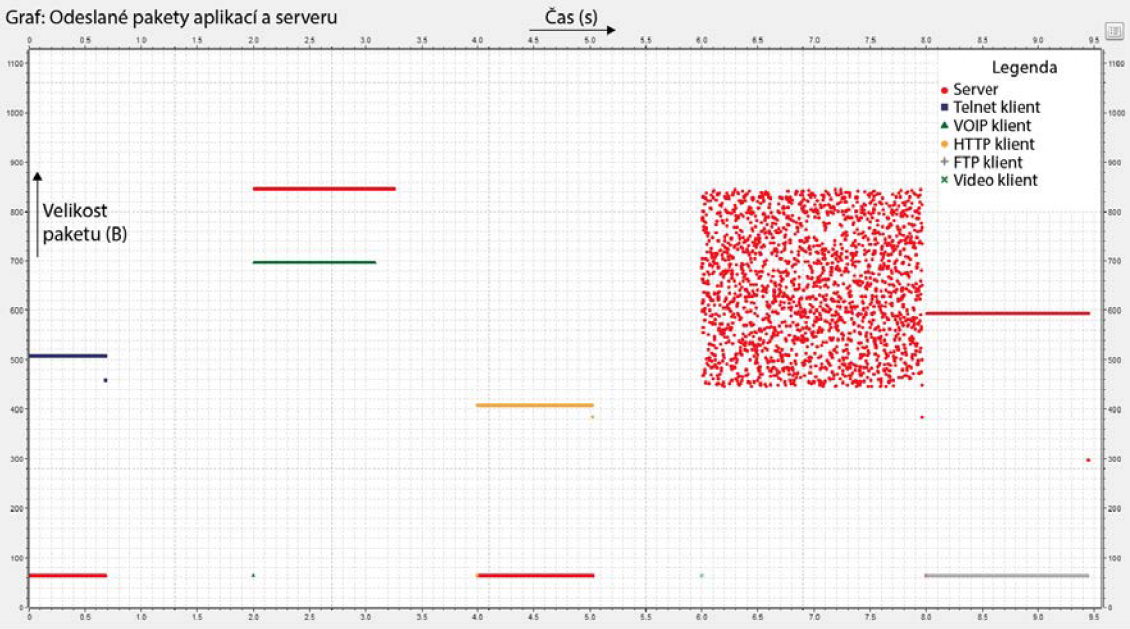
Před spuštěním simulace je potřeba sestavit projekt, to provedeme kliknutím pravým tlačítkem myši na složku s projektem» **Build Configurations » Build Selected… »** zaškrtneme **gcc-debug » OK**.

Po dokončení operace je projekt připraven k simulaci, kliknutím pravým tlačítkem myši na konfigurační soubor **omnetpp.ini » Run As** » **OMNeT++ Simulation »** z kontextové nabídky můžeme vybrat požadovanou konfiguraci, v tomto případě **App_gradually**.

Stisknutím ikony **Run** (nebo klávesou **F5**) se simulace spustí, v simulačním okně probíhá výpis poslaných zpráv, které se zobrazují také v animaci sítě. V simulaci můžeme sledovat chování různých aplikací, jejich navazování a běh. K urychlení dílčích části lze použít ikona **Until** (**Ctrl+F5**) pro posunutí na požadovaný čas, ikona **Fast** (**F6**) a ikona **Express** (**F7**).

Po ukončení simulace se vytvoří soubory s výsledky ve složce **results**. Dvojklikem otevřeme jeden z nich a potvrdíme vytvoření souboru `App_gradually.anf`. V souborech s příponou `.anf` můžeme prohlížet výsledky provedených simulací, v záložce **Browse Data** máme na výběr z výsledků zobrazených vektorově v grafu a skalárně.

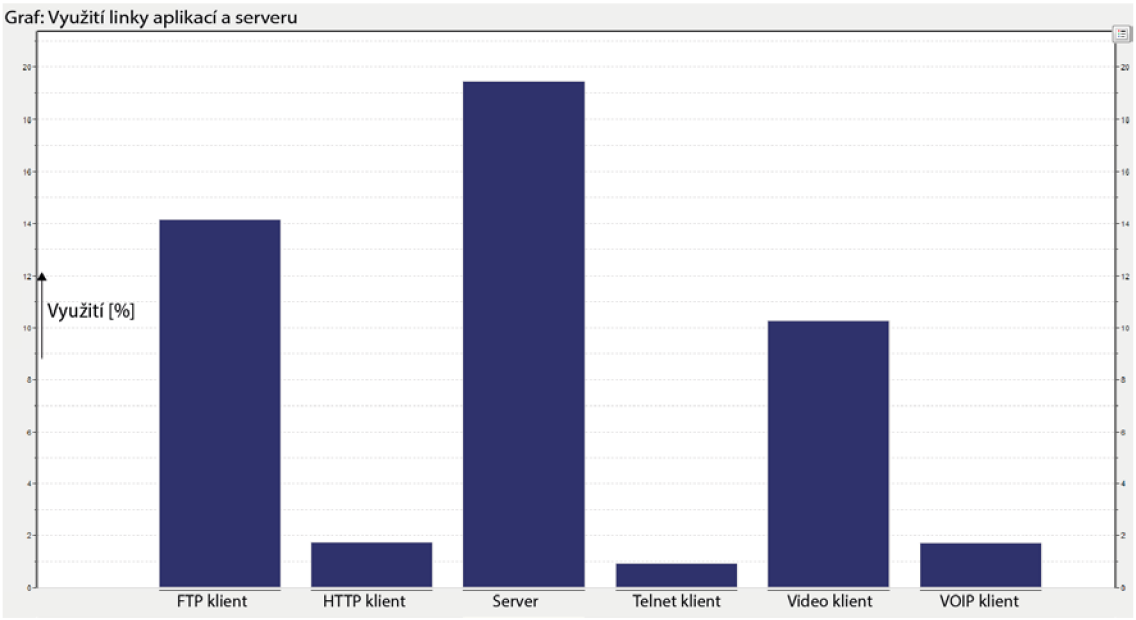
Pro zobrazení provozu klientů a serveru si otevřeme záložku **Vectors** » v poli **statistic name filter** vybereme **txPk:vector(packetBytes)**. Zobrazí se nám přenos všech aktivních rozhraní, jedná se o statistiku odesílaných paketů během simulace. V okně s výsledky postupně označíme všechny klienty a server a klikneme pravým tlačítkem myši » **Plot**. Výsledky zobrazuje Obrázek 15. Legendu položek lze zobrazit pomocí ikony v pravém horním rohu. Z grafu můžeme vypozorovat komunikaci dílčích aplikací se serverem, popř. komunikace UDP aplikací.



Obrázek 15: Grafické zobrazení odeslaných paketů

Následující graf nám umožňuje vypozorovat přenosy jednotlivých aplikací. V čase od 0 sekund do 0,7 sekund probíhá přenos Telnet aplikace. Od dvou vteřin začíná přenos dat hovoru, který končí přibližně ve 3,2 vteřin. Ve čtvrté vteřině začíná přenos protokolu HTTP, který trvá jednu vteřinu. V čase 6 – 8 vteřin se uskutečňuje stream videa ze strany serveru. Posledním přenosem je FTP komunikace (8 – 9,5 vteřin).

Dále si zobrazíme procento využití linek. Přepneme se na záložku **Scalars** » do pole **statistic name filter** vybereme **rx channel utilization (%)**, vybereme stejné statistiky, jako tomu bylo u odesílání paketů a výsledky zobrazíme do grafu. Graf vytížení linek nalezneme na Obrázek 16.



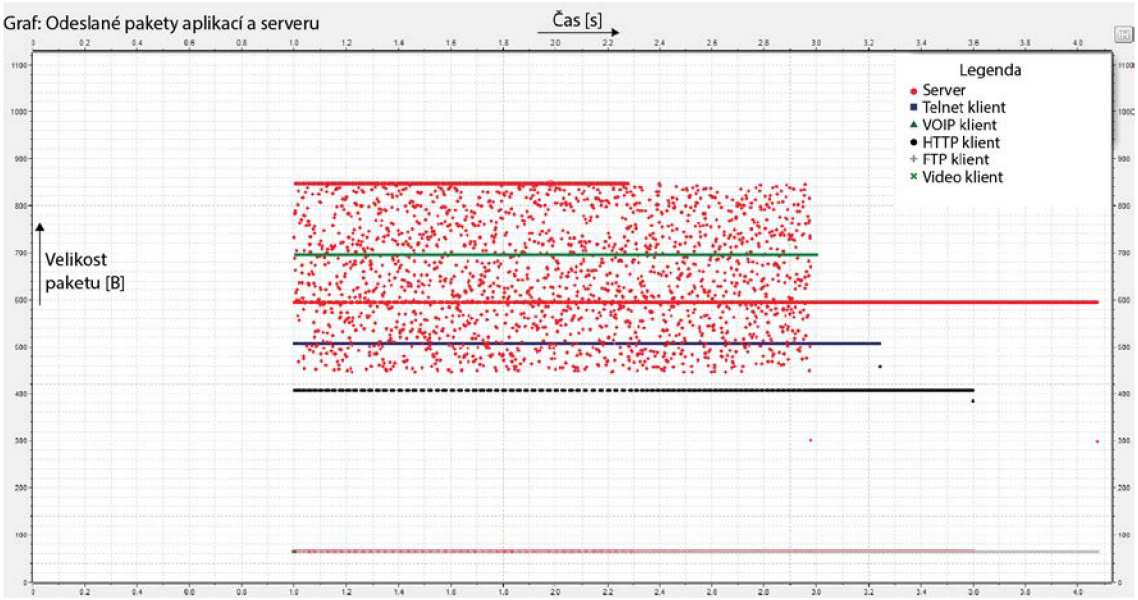
Obrázek 16: Graf využití linek

5.2.3 Vypracování 2. část – Souběžný běh aplikací

V druhé části vytvoříme nový scénář, ve kterém pozměníme časy vysílání paketů. Nastavíme je tak, aby spuštění všech aplikací proběhlo ve stejný okamžik. Obdobným způsobem tedy v souboru **omnetpp.ini** vytvoříme novou konfiguraci **[Config App_parallel]** a přidáme časy spuštění a ukončení aplikací.

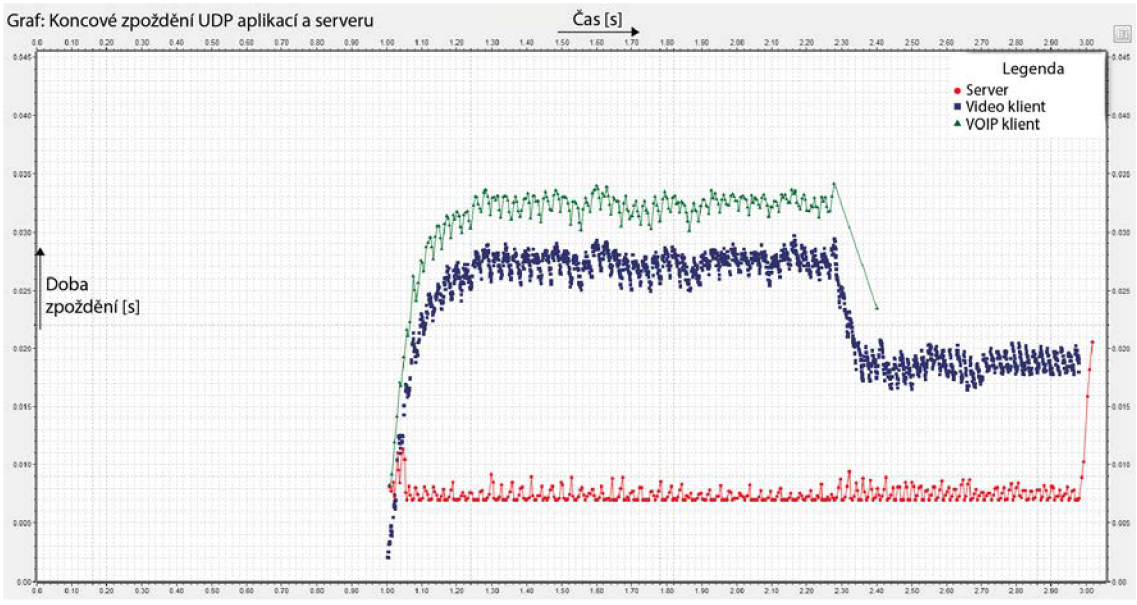
```
[Config App_parallel]
**.server.tcpApp[0].typename = "TCPGenericSrvApp"
**.server.tcpApp[0].localPort = 23

**.TelnetClient.tcpApp[0].startTime = 1s
**.TelnetClient.tcpApp[0].stopTime = 3s

**.server.udpApp[1].startTime = 1s
**.server.udpApp[1].stopTime =  3s
**.VOIPClient.udpApp[0].startTime = 1s
**.VOIPClient.udpApp[0].stopTime = 3s

**.HTTPClient.tcpApp[0].startTime = 1s
**.HTTPClient.tcpApp[0].stopTime = 3s

**.VideoClient.udpApp[0].startTime = 1s

**.server.tcpApp[2].startTime = 1s
**.server.tcpApp[2].stopTime = 3s
**.n = 0
```

Spustíme simulaci podobně, jako tomu bylo v první části. Ve výběru scénáře vybereme právě vytvořený scénář **[Config App_parallel]**. Simulaci můžeme opět urychlit ikonou **Fast (F6)** nebo **Express (F7)**. Po dokončení simulace se přesuneme do složky **results**, kde se vytvořily nové soubory s výsledky pojmenované **App_parallel**. Dvojklikem na jeden z nich otevřeme okno s výsledky. Přepneme se na záložku **Browse Data » Vectors**. V poli **statistic filter** opět zadáme **txPk:vector(packetBytes)**, vybereme statistiky klientů a serveru a hodnoty zobrazíme do grafu pravým klikem myši **» Plot**. Otevřeme si graf z prvního scénáře a grafy porovnáme.

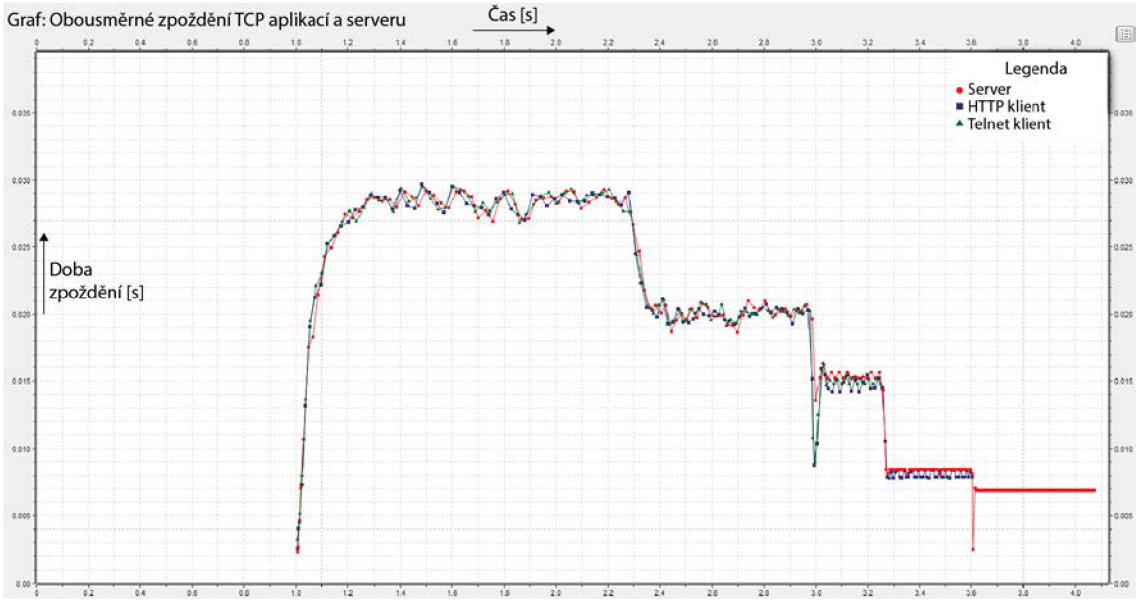


Obrázek 17: Zobrazení odeslaných paketů rozhraní

Dále se ve druhém scénáři podíváme na zpoždění při přenosu a srovnáme ho s prvním scénářem. Ve výsledcích druhého scénáře, v kartě **Vectors** si v poli **statistic filter** vybereme statistiku **endToEndDelay:vector** a všechny nalezené výsledky zobrazíme do grafu. Tato statistika vypovídá o době, kdy je paket přenesen od zdroje k cíli. Zpravidla se jedná o hodnoty UDP aplikací. Pro zobrazení TCP zpoždění slouží statistika RTT, tzv. Round-Trip Time. Ta určuje dobu běhu paketu od zdroje k cíli a zpět jako jeho potvrzení (aplikace TCP). Dobu RTT zobrazíme tak, že v poli **statistic filter** vybereme **measured RTT**. Označíme všechny položky, funkcí **Plot** je znázorníme.



Obrázek 18: Graf koncového zpoždění End to End Delay



Obrázek 19: Graf obousměrného zpoždění Round-Trip Time

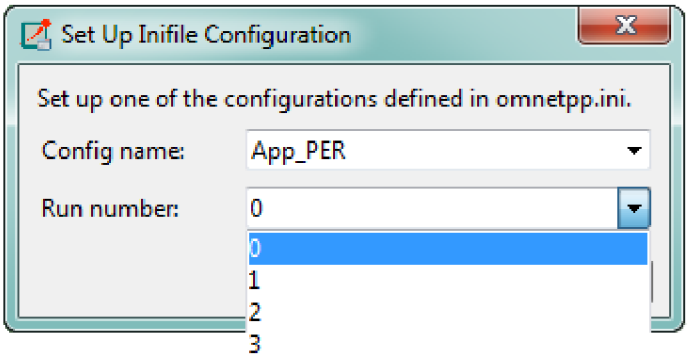
5.2.4 Vypracování 3. část – Zvýšená chybovost

K nastavení chybovosti použijeme vytvořenou proměnnou s pohyblivou řadovou čárkou `n`. Nastavené hodnoty budou simulovat chybovost linky 0.17, 0,20, 0,9 a 2 %. Budeme vycházet z předchozího scénáře (klíčové slovo `extends`). Je důležité podotknout, že se jedná o chybovost linky – hodnota chybovosti bude u paketů růst o nastavený parametr `n` při procházení každé další linky.

V souboru `omnet.ini` vytvoříme třetí scénář **[Config App_PER]** a následujícím kódem nastavíme parametr chybovosti.

```
[Config App_PER]
extends = App_parallel
**.n = ${per=0.0017,0.0020, 0.009, 0.02}
```

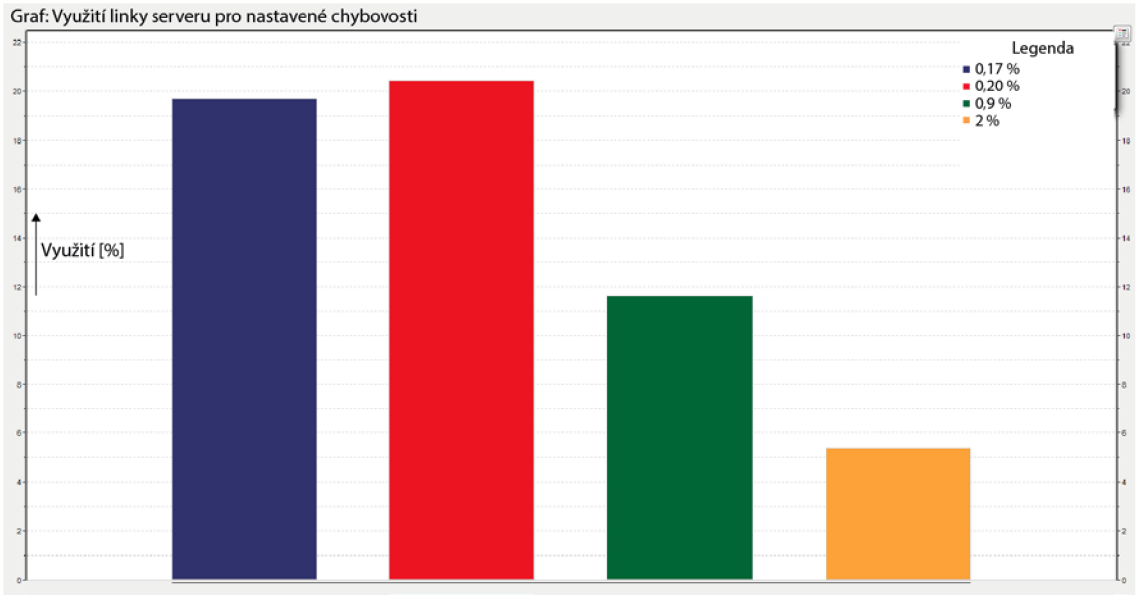
Před spuštěním simulace budeme nově vyzvání také k výběru pořadového čísla simulace, viz Obrázek 20. Každé z nich reprezentuje jinou hodnotu nastavovaného parametru, tedy v tomto případě dílčí chybovosti linky. Vybereme tedy scénář **App_PER** a postupně provedeme simulaci pro všechna pořadová čísla (0 až 3). K urychlení průběhu simulace můžeme využít ikonu **Express (F7)**.



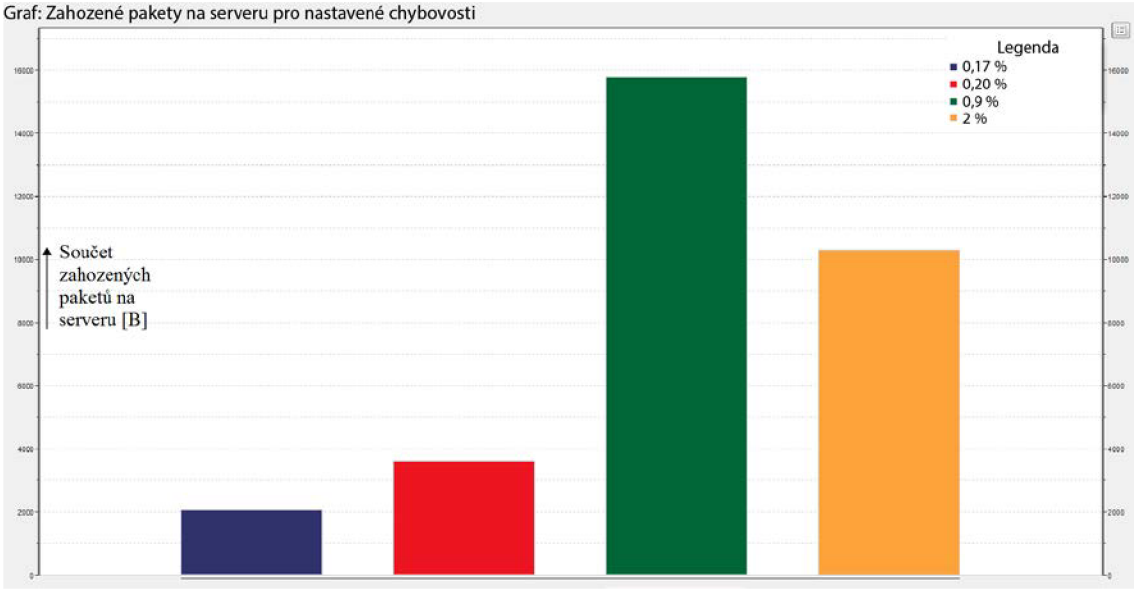
Obrázek 20: Výběr pořadového čísla simulace

Po dokončení simulace se opět přesuneme do složky **results** a otevřeme okno s výsledky. Přepneme se na záložku **Browse Data» Vectors**, postupně si zobrazíme statistiky odesílaných paketů pro všechny parametry. Tyto statistiky mezi sebou porovnáme. Pro zajímavost si zobrazme graf využití linky serveru. V kartě **Scalars** v poli **module filter** vybereme **AppNetwork.server.eth[0].mac** a v poli **statistics name filter** zvolíme **rx channel utilization (%)**. Hodnoty zobrazíme do grafu. Ukázku grafu můžeme vidět na Obrázek 21.

Ještě si zobrazíme celkový počet zahozených paketů na serveru. V kartě **Scalars**, poli **module filter** ponecháme **AppNetwork.server.eth[0].mac** a do pole **statistic name filter** vybereme **droppedPkBitError:sum(packetBytes)**. Výsledek by měl odpovídat grafu na Obrázek 22. Je důležité si uvědomit, že při velké chybovosti nedojde k navázání spojení TCP aplikací a aplikace neposílá žádné data až do chvíle, kdy se znovu pokusí o sestavení relace (viz statistika odeslaných paketů).



Obrázek 21: Graf využití linky serveru pro nastavené chybovosti



Obrázek 22: Graf zahozených paketů na serveru v bajtech

5.2.5 Vypracování 4. část – Výpadek linky

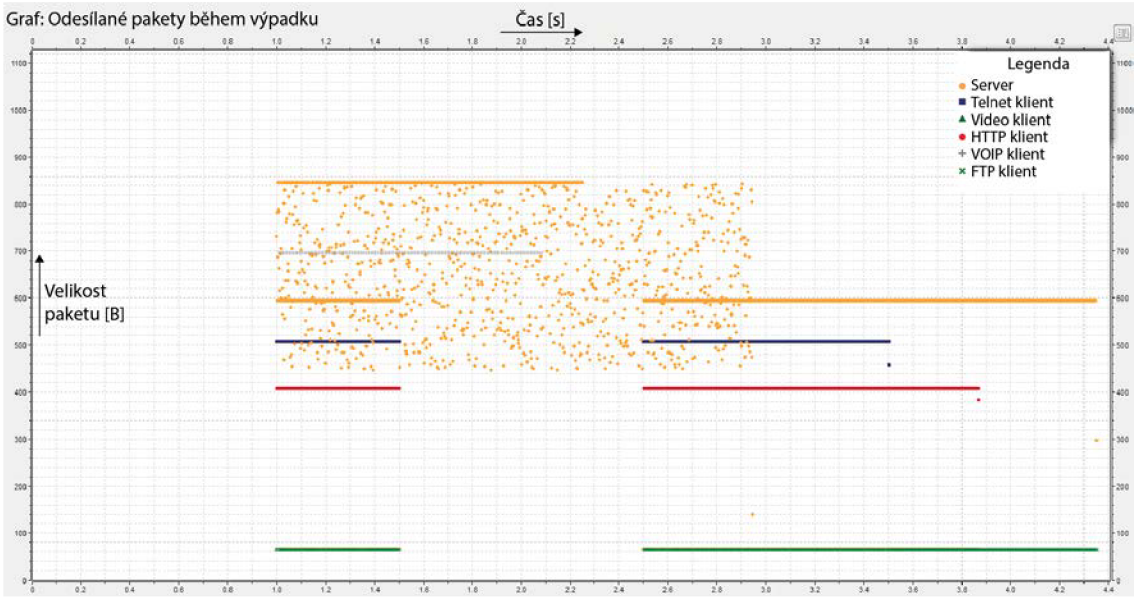
Další vypracování vychází ze scénáře souběžného běhu aplikací, kdy na několik sekund vypadne trasa mezi přepínačem a směrovačem. Budeme sledovat chování aplikací během daného výpadku. Do konfiguračního souboru nakopírujeme následující kód:

```
[Config App_LinkCut]
extends = App_parallel
*.scenarioManager.script = xmldoc("vypadek.xml")
```

Jako další musíme vytvořit XML soubor, který se postará o simulovaný výpadek. Vše je popsáno v XML souboru, který je při startu simulace načten manažerem scénáře. Pokud je dodržena správná syntaxe kódu, manažer tento soubor vykoná. Nyní soubor vytvoříme pravým klikem na projekt, poté **New » File**. Do dialogu vytváření souboru napíšeme jméno souboru **vypadek.xml** a klikneme na tlačítko **Finish**. Do právě vytvořeného souboru vložíme následující kód, kterým nastavujeme dobu výpadku od 1,5 sekundy a opětovné navázání v době 2,4 sekund.

```
<scenario>
<at t="1.5">
<disconnect src-module="router2" src-gate="pppg$o[0]" />
<disconnect src-module="router1" src-gate="pppg$o[0]" />
</at>
<at t="2.4">
<connect src-module="router2" src-gate="pppg[0]"
dest-module="router1" dest-gate="pppg[0]"
channel-type="inet.common.misc.ThruputMeteringChannel">
<param name="delay" value="0.1ms" />
<param name="datarate" value="10Mbps" />
</connect>
</at>
</scenario>
```

Scénář odsimulujeme a zobrazíme si vektorové výsledky odeslaných paketů, viz Obrázek 23. Pro lepší orientaci může pomoci graf přijatých paketů, ten se nachází v kartě **Vectors**, v poli **statistic name filter** zvolíme **rxPkOk:vector(packetBytes)** a vybereme klienty a server.



Obrázek 23: Zobrazení odesílaných dat během výpadku

5.2.6 Vypracování 5. část – Přetížení linky

V posledním scénáři upravíme tok dat linky tak, aby došlo k maximálnímu využití její kapacity. V tomto stavu budeme sledovat chování dílčích aplikací. Pro naše účely si

zvolíme linku ppp.

V souboru `omnet.ini` vytvoříme další scénář, který bude vycházet ze souběžného toku aplikací. Bude se jednat o totožnou konfiguraci, jako tomu bylo ve druhém scénáři, pro přehlednost výsledků je však lepší vytvořit scénář nový, aby se dané výsledky nepřepisovaly. Vytvoříme tak následující konfiguraci:

```
[Config App_LowCapacity]
extends = App_parallel

**.ppp[*].queueType = "DropTailQueue"
**.ppp[*].queue.frameCapacity = 1550
```

Nezbytným je také změnit kapacitu přenosového média. V souboru NED v těle nově vytvořeného modulu `ppp` změníme nastavení velikosti linky `datarate = 1Mbps;`, tím docílíme zmenšení toku dat mezi směrovači, bude tedy omezena každá aplikace. Důležité je připomenout, že jakmile se při přenosu dosáhne maximální kapacity linky, začnou se pakety na směrovačích ukládat do front. Aby ve frontách nedocházelo k zahazování paketů, nastavili jsme velikost fronty na 1550.

Pro získání výsledků scénář odsimulujeme. Zajímají nás odeslané pakety **txPk:vector(packetBytes)** zároveň s přijatými pakety **rxPkOk:vector(packetBytes)**. Výsledky by měli odpovídat Obrázek 24. Pro zajímavost si můžete zobrazit obousměrné zpoždění TCP aplikací a koncové zpoždění UDP aplikací – z grafů lze vypozorovat postupné plnění front a zároveň jistou prioritizaci jednoho z UDP provozů.

5.2.7 Otázky

1. Vyjmenujte alespoň 3 spojově orientované aplikační protokoly, jejich porty a použití.

2. Uveďte zásadní rozdíly mezi TCP a UDP aplikacemi.

3. Jaký parametr (zpoždění, jiter, ztrátovost) je pro danou aplikaci (FTP, video stream, VOIP) kritický.

4. Zdůvodněte jednotlivé toky grafu odeslaných paketů – scénář 1.

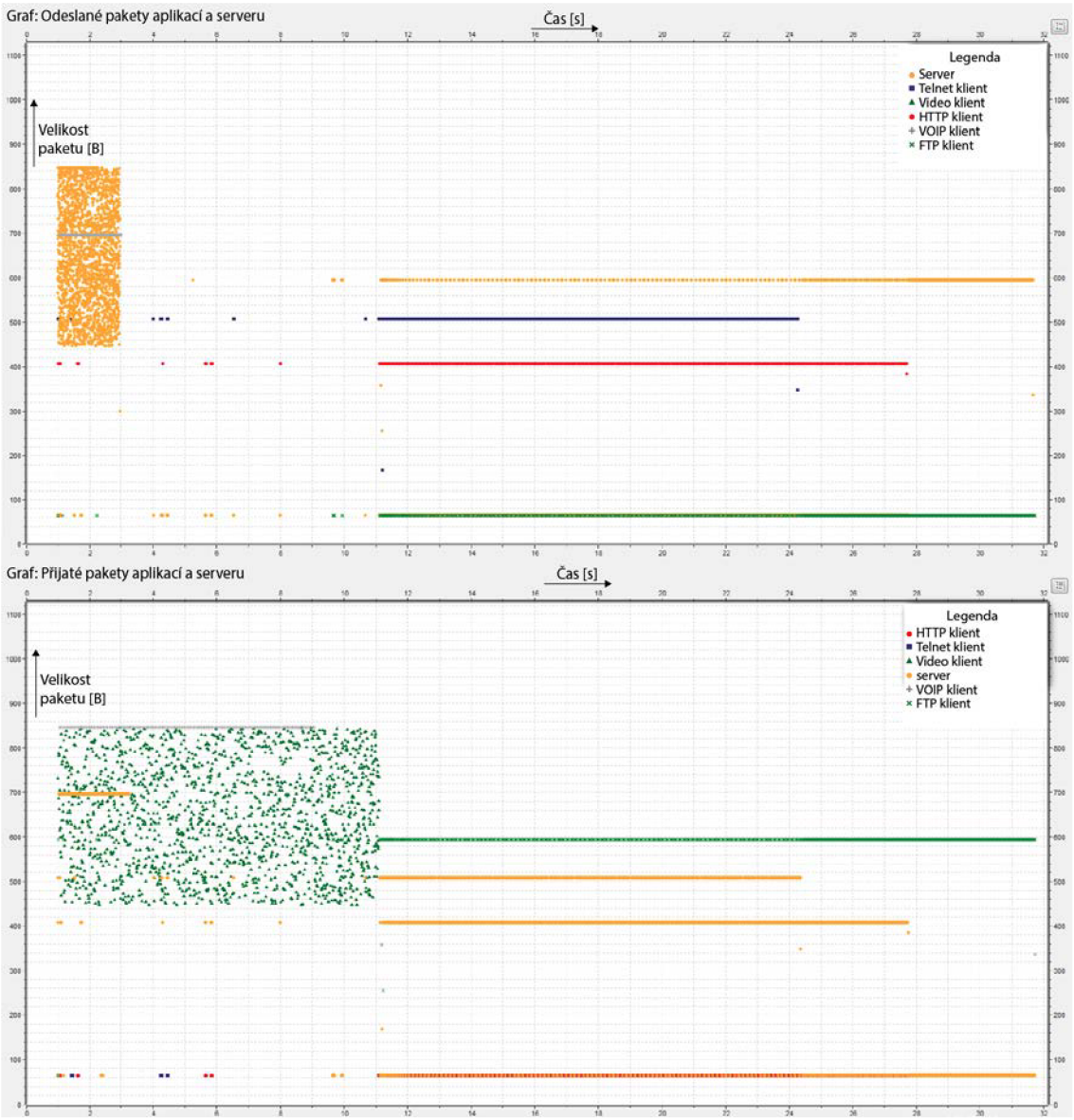
5. Vysvětlete procentuální vytížení grafu využití linek – scénář 1.

6. Konzultujte změny odeslaných paketů, koncového a obousměrného zpoždění oproti prvnímu scénáři – scénář 2.

7. Scénář 3 – proč je využití linky při druhé ztrátovosti větší, než při první?

8. Scénář 3 – zobrazte si graf odeslaných paketů. Diskutujte, k čemu u TCP aplikací došlo při nastavení vyšších hodnot parametrů `n`.

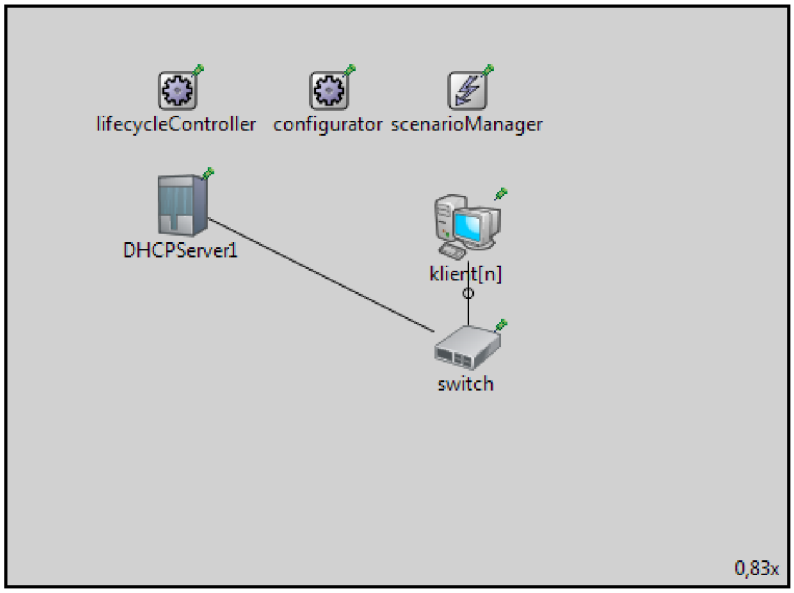
9. Scénář 5 – vyzkoušejte si změnu nastavení kapacity linky `ppp datarate`, konzultujte změny, ke kterým došlo.



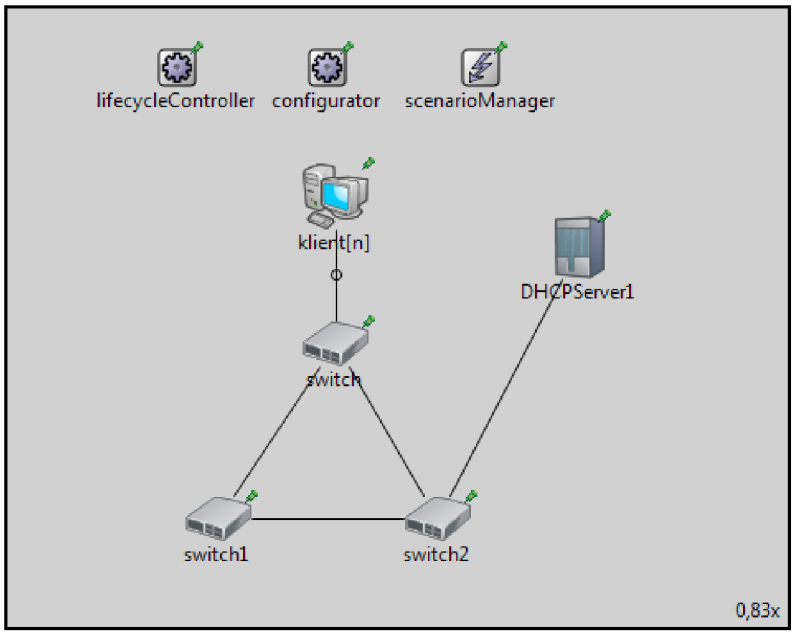
Obrázek 24: Odchozí a příchozí pakety jednotlivých aplikací

6 LABORATORNÍ ÚLOHA ČÍSLO 2 – DYNAMICKÁ A STATICKÁ ADRESACE

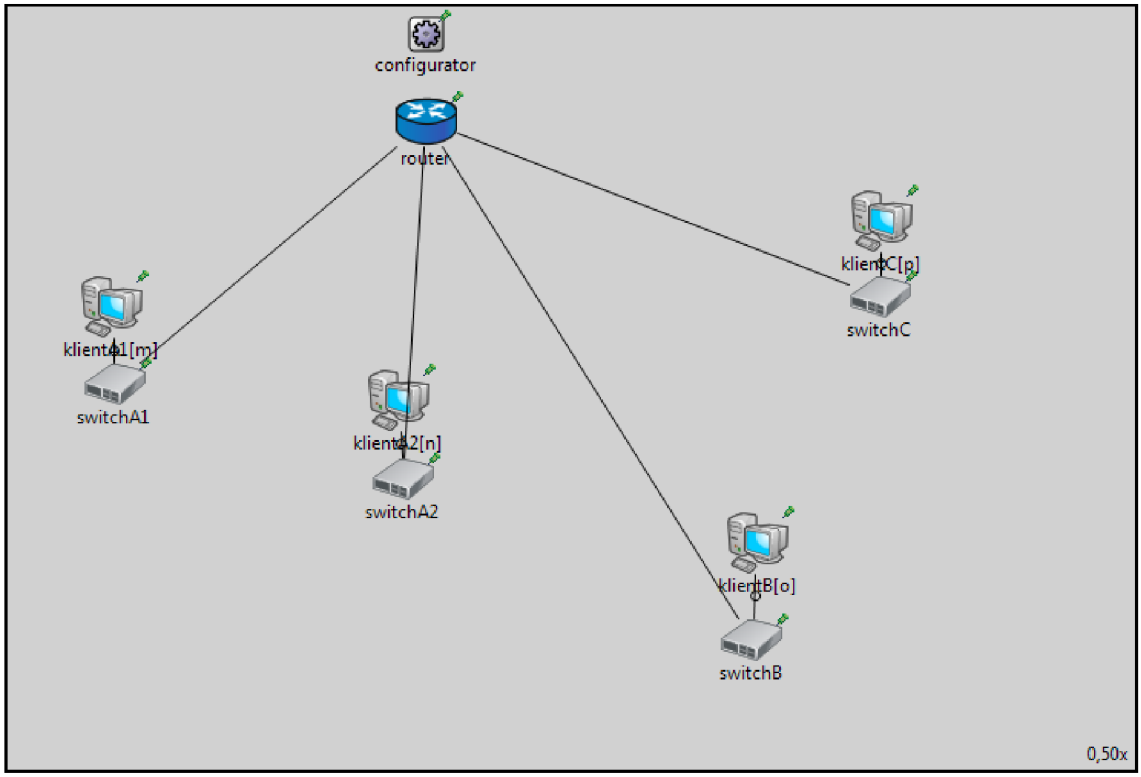
Cílem této laboratorní úlohy je seznámení s protokolem DHCP (Dynamic Host Configuration Protocol), nastavení dynamické adresace sítě a sledování vlivů tohoto protokolu v určitých situacích. Druhá část je věnována statické adresaci, především problematice podsíťování a manuální konfiguraci IP adres. Pro simulaci bude potřeba vytvořit celkem tři topologie, přičemž v každé z nich lze nastavit libovolný počet klientů. Tyto topologie lze vidět na Obrázek 25, Obrázek 26 a Obrázek 27.



Obrázek 25: Zapojení topologie s DHCP serverem



Obrázek 26: Zapojení topologie s DHCP serverem a všesměrovou smyčkou



Obrázek 27: Zapojení topologie pro statickou adresaci

6.1 Teoretický úvod

6.1.1 DHCP (Dynamic Host Configuration Protokol)

Aplikační protokol DHCP slouží k dynamické konfiguraci parametrů sítě. Využívá transportní protokol pro nespolehlivý přenos UDP, klient komunikuje na portu 68 a server naslouchá na portu 67. Jedná se o rozšíření původního protokolu BOOTP. Konkrétně se může jednat o nastavení IP adresy, masky sítě, výchozí brány, DNS serverů atp. Jde o stavový protokol typu klient-server. Protokol pracuje na principu zapůjčování parametrů na určitou dobu (lease time), po vypršení této doby si musí klient o potřebné parametry znovu zažádat (na rozdíl od BOOTP, kdy klient využívá přidělené parametry natrvalo).

DHCP je stavový protokol – pro získání záznamu si musí klient se serverem vyměnit sadu zpráv, aby dané parametry obdržel. Při obdržení parametrů si klient i server ukládá dobu záznamu. Klient si zpravidla žádá o prodloužení záznamu po uplynutí 50 % nastavené doby, po neúspěchu pak ještě při 87,5 % doby.

Pokud má koncová stanice povoleno získávání parametrů z DHCP serveru, vyšle každých 60 sekund všesměrovou zprávu DHCP_DISCOVER. Nachází-li se v dané lokální síti DHCP server, má dostatečný počet adres a nemá koncovou stanici na seznamu zakázaných hostů, odpoví klientovi zprávou DHCP_OFFER (přímo nebo všesměrově). Další stav je žádost klienta o adresu zprávou DHPC_REQUEST (všesměrově). Po přijetí zprávy DHCP_ACK od serveru, může koncová stanice přijaté nastavení využít.

Protokol DHCP také definuje další typy zpráv – DHCP_NAK pro případ, kdy server zamítne požadavek klienta nebo DHCP_RELEASE pro manuální uvolnění adresy klientem. Poslední zpráva DHCP_INFORM slouží pro vyžádání více informací klientem na DHCP server.

6.1.2 Statická adresace

IP adresy verze 4 jsou 32 bitové identifikátory, obvykle slouží k jedinečné identifikaci v celém internetu (vyjma privátních adres). K jednoduššímu zápisu adresy slouží tečkovaná desítková anotace – celé 32 bitové číslo se rozdělí na čtyři části po osmi bitech (nazývané oktet/byte) a každá tato část je přepočítána do desítkové číselné soustavy. Z toho plyne, že každý oktet může nabývat hodnot 0-255 (2^8). Pro zápis IPv6 adres se používá zejména hexadecimální číselná soustava. Úloha se věnuje pouze adresaci v oblasti IPv4.

V dobách vzniku internetu se adresní prostor rozděloval na třídy (třídní adresování), kdy se pracovalo s dvousložkovou adresou – adresou sítě a adresou koncové stanice. Podle prvních bitů adresy se IP adresy dělily na třídy (viz Tabulka 2). Z tabulky lze vypozorovat, že téměř 50 % veškerého rozsahu zabírá třída A (128 sítí, cca 16777214 hostů), 25 % třída B (16383 sítí, 65534 hostů), 12,5 % třída C (2097150 sítí/254 hostů) a zbytek 6,25 %. Z důvodu neefektivního využití se však postupem času přestoupilo k beztřídnímu adresování.

Tabulka 2: Třídy IPv4 adres

Třída	Určující bity	Rozsah adres	Maska	Délka prefixu sítě	Typ adres
Třída A	0xxx	0 - 127.x.x.x	255.0.0.0	/8	Hlavní
Třída B	10xx	128 - 191.x.x.x	255.255.0.0	/16	Hlavní
Třída C	110x	192 - 223.x.x.x	255.255.255.0	/24	Hlavní
Třída D	1110	224 - 239.x.x.x			Multicast
Třída E	1111	240 - 225.x.x.x			Rezervováno

Nepřiměřené rozdělení adres do tříd vedlo ke vzniku podsíťování (subnetting). Pomocí podsíťování lze přiřazený adresní rozsah rozdělit na více menších dílčích bloků a vytvořit tak několik podsítí. Podsítě usnadňují správu, implementace bezpečnostních pravidel je také jednodušší. K podsíťování slouží maska sítě.

Hlavním účelem masky sítě je oddělení adresy sítě od adres koncových stanic. Tohoto aspektu se využívá k rozdělení jedné fyzické sítě na více logických – podsíťování. Lze jí zapisovat pomocí desítkové tečkové anotace, nebo jednodušeji jako tzv. délku prefixu (IP adresa 192.168.1.1 s maskou zapsána v desítkovém tvaru 255.255.255.192, IP adresa 192.168.1.1/26 zapsána s maskou v prefixovém tvaru). V dané síti jsme schopni pomocí IP adresy koncové stanice a jeho masky určit adresní prostor celé sítě, tedy adresu sítě (první adresa rozsahu sítě), všesměrovou adresu (poslední adresa rozsahu sítě) a adresy koncových stanic (adresní prostor mezi první a poslední adresou).

6.2 Postup řešení

6.2.1 Založení nového projektu

Spustíme program `omnetpp.exe`. Pokud se objeví nabídka pro vybrání pracovního adresáře, tento formulář potvrdíme beze změny (vybraná cesta musí být: `omnetpp-5.0\samples`), jinak vytvořený projekt nebude fungovat.

K vytvoření nového projektu vybereme z nabídky **File » New » OMNeT++ Project…**. Projekt pojmenujeme VUT loginem, dále stiskneme **Next » Empty project » Finish**. V okně Project Explorer se vytvořila námi pojmenovaná složka, ve které budou uloženy všechny části projektu. Nejprve je zapotřebí přiřadit k projektu odkazy na zdrojové soubory, se kterými se bude dále pracovat. To provedeme kliknutím pravým tlačítkem na vytvořenou složku » **Properties » Project References »** vybereme položku **inet » OK**.

Pro vytvoření projektu jsou zapotřebí dva základní soubory. Network Description File, s příponou `.ned`, který definuje topologii sítě, použité technologie a síťové prvky. Druhým je Initialization File, přípona `.ini`, který určuje průběh a parametry simulace.

Klikneme pravým tlačítkem myši na složku a vybereme **New » Network Description File (NED)**, pojmenujeme ji **Network.ned » Next » Empty NED File » Finish**.

V souboru NED lze pracovat ve dvou režimech, v grafickém nebo kódovém. Mezi nimi se lze přepínat v levém dolním rohu volbou Design/Source. Jsou navzájem propojené, pokud tedy přidáme prvky sítě z palety, vytvoří se k nim patřičný kód a naopak vložením kódu lze vytvářet celou síť.

6.2.2 Vypracování 1. část – DHCP server

Nejprve musíme importovat do projektu prvky, se kterými se bude pracovat. Do souboru `Network.ned` vložíme následující kód:

```
import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.common.lifecycle.LifecycleController;
import inet.common.scenario.ScenarioManager;
import inet.node.ethernet.EtherSwitch;
import inet.node.inet.StandardHost;
import inet.node.inet.Router;
import ned.DatarateChannel;
```

Dále vytvoříme síť pojmenovanou DHCPNetwork:

```
network DHCPNetwork
{
    parameters:
        @display("bgb=562.8,416.40002");
        int n;
    submodules:
        klient[n]: StandardHost {
            @display("p=231.6,158.40001,ri,100,100");
        }
        DHCPServer: StandardHost {
            parameters:
                @display("p=126,142.8;i=device/server");
            gates:
                ethg[1];
        }
        switch: EtherSwitch {
            parameters:
                @display("p=331.2,244.8");
        }
        configurator: IPv4NetworkConfigurator {
            parameters:
                config = xml("<config><interface hosts='DHCPServer'
address='192.168.1.1' netmask='255.255.255.0' /></config>");
                @display("p=231.6,60");
        }
        lifecycleController: LifecycleController {
            @display("p=122.4,60");
        }
```

```
        scenarioManager: ScenarioManager {
            @display("p=331.2,60");
        }

    connections:
        for i=0..n-1 {
            klient[i].ethg++ <--> NormalPath <--> switch.ethg++;
        }
        switch.ethg++ <--> NormalPath <--> DHCPServer.ethg++;
}
channel NormalPath extends DatarateChannel
{
    parameters:

        datarate = 10Mbps;
        delay = 0.01ms;
        ber = 0;
        per = 0;
}
```

Upřesníme si možné nejasnosti v kódu. Tímto kódem vytváříme síť DHCPNetwork, ve které bude proměnlivý počet klientů – ten stanovuje celočíselná proměnná `n`. Dále vytváříme klienty o daném počtu `n`, proměnnou budeme později nastavovat v `.ini` konfiguračním souboru. Za povšimnutí také stojí `@display`, kterému kromě pozice nastavujeme parametr rozmístění `ri` (kruh) o délce a šířce 100. Jako DHCP server poslouží obyčejný klient s ethernetovým rozhraním. Pro toto rozhraní nastavujeme adresu v konfigurátoru. Bez nastavené adresy by DHCP server nebyl schopný přiřadit adresy.

Pomocí `lifecycleController` můžeme vyvolat události na koncovém bodě. Pomocí něj lze zajistit vypnutí/zapnutí zařízení, restart, stav poruchy či obnovy. Aby mohl `lifecycleController` vyvolat změnu stavu v určitém čase, musí využít služeb manažéru scénáře. Cyklus `for` zde slouží pro propojení proměnlivý počet koncových stanic s přepínačem. V případě, kdy by počet klientů `n` bylo 5, by cyklus proběhl právě pětkrát, pokaždé připojující jiného klienta k přepínači.

Tímto je vytvořená síť kompletně zprovozněna k provádění simulací, nyní je zapotřebí určit parametry a průběh simulace. Jelikož soubor NED je statický, je vhodné v něm definovat pouze základní parametry, které nechceme dále měnit ani nijak upravovat.
Pro nastavení průběhu simulace a síťového provozu slouží konfigurační soubor s příponou `.ini`. Spuštěním tohoto souboru zahájíme simulaci.

Do složky s projektem vytvoříme nový soubor, **New » Initialization File (ini) »** můžete ponechat název **omnetpp.ini » Next » Empty Ini File » Finish**.

Pro potřeby simulace je možné v jednom souboru vytvářet několik různých konfigurací, k vybrání požadované budeme vyzvání po spuštění simulace. Pro vytvoření nové konfigurace se používá syntaxe `[Config nazev_konfigurace]` zapsaná v hranatých závorkách. Hlavní částí je sekce `[General]`, která obsahuje nastavení společná pro všechny vytvořené konfigurace, jejím nastavením začneme.

Do sekce General vložíme následující kód, který povolí výpis problému při chybně nastavené simulaci, nastaví se cesta k potřebným doplňkům a omezí se maximální trvání simulace na 200 sekund.

```
[General]
debug-on-errors = true
tkenv-plugin-path = ../../../etc/plugins
sim-time-limit = 200s

[Config DHCP_function]
network = DHCPNetwork
*.n = 5
**.numUdpApps = 1
**.klient[*].udpApp[0].typename = "DHCPClient"
**.klient[0].udpApp[0].startTime = 0s
**.klient[1].udpApp[0].startTime = 20s
**.klient[2].udpApp[0].startTime = 40s
**.klient[3].udpApp[0].startTime = 60s
**.klient[4].udpApp[0].startTime = 80s

**.DHCPServer.udpApp[0].typename = "DHCPServer"
**.DHCPServer.udpApp[0].subnetMask = "255.255.255.0"
**.DHCPServer.udpApp[0].ipAddressStart = "192.168.1.101"
**.DHCPServer.udpApp[0].maxNumClients = 100
**.DHCPServer.udpApp[0].gateway = "192.168.1.1"
**.DHCPServer.udpApp[0].dns = ""
**.DHCPServer.udpApp[0].leaseTime = 80s
```

Nyní přejdeme k vytvoření prvního scénáře `[config DHCP_function]` zkopírováním kódu v dané sekci. Pro simulování nastavujeme právě vytvořenou síť DHCPNetwork s celkem pěti klienty. Na všech klientech a serveru definujeme jednu UDP aplikaci. Aplikace všech klientů jsou specifikovány jako DHCPClient aplikace klienti jsou postupně spouštěni k získání adresy.

DHCP serveru naopak definujeme `DHCPServer` aplikaci. Masku sítě nastavujeme pro celkem 254 klientů, nicméně masku lze libovolně přizpůsobit reálnému počtu. Příkaz `ipAddressStart` označuje počáteční přidělovanou adresu. S každou další potřebou se adresa inkrementuje. Pro úpravu počtu klientů lze použít `maxNumClients`. Nezbytnou součástí konfigurace DHCP serveru je nastavit mu bránu, kterou bude klientům přidělovat. Bez brány by klienti nebyli schopni komunikovat s vnějšími sítěmi. Přidělování DNS nenastavujeme, musíme však konfiguračnímu souboru říci, že si toho jsme vědomi. Poslední údaj určuje dobu záznamu (lease time). Tím máme konfigurační soubor kompletní a můžeme přejít k simulaci.

Před spuštěním simulace je potřeba sestavit projekt, to provedeme kliknutím pravým tlačítkem myši na složku s projektem» **Build Configurations » Build Selected…** » zaškrtneme **gcc-debug » OK**.

Po dokončení operace je projekt připraven k simulaci, kliknutím pravým tlačítkem myši na konfigurační soubor **omnetpp.ini » Run As** » **OMNeT++ Simulation** » z kontextové nabídky můžeme vybrat požadovanou konfiguraci, v tomto případě **DHCP_function**.

Stisknutím ikony **Run** (nebo klávesou **F5**) se simulace spustí, v simulačním okně probíhá výpis poslaných zpráv, které se zobrazují také v animaci sítě. Hned po spuštění

simulace první klient kontaktuje DHCP server a po vyměnění určité sady zpráv dostane přidělenou adresu z námi nastaveného rozsahu. Sekvence se opakuje pro všechny nastavené klienty, rozdílem je čas. Po uplynutí 50 % doby záznamu si klienti musí záznam obnovit. V této části je klíčové sledovat poslané zprávy mezi klienty a DHCP serverem. Pro přehlednější výpis zpráv lze sledovat událostní okno.

Vyměněné zprávy mezi koncovou stanicí a serverem si poznamenejte. Také si poznamenejte způsob distribuce dané zprávy (unicast x broadcast). Je nutno zmínit, že v polovině času vypršení záznamu si klient pomocí ARP požadavku a IP adresy zjistí MAC adresu DHCP serveru pro další komunikaci.

Také si vyzkoušejte v souboru `.ini` snížit počet možných adres k přiřazení pod hodnotou vyskytujících se klientů (změnou hodnoty `maxNumClients`), pozorujte změnu při simulaci.

6.2.3 Vypracování 2. část – Vypršení doby záznamu

Ve druhé části budeme pozorovat situaci, kdy koncové stanici vyprší doba záznamu a reakci DHCP serveru na tuto událost. Pro realizaci musíme přidat další scénář do konfiguračního souboru `.ini`, je také nezbytné přidat a upravit soubor `XML`, který se postará o řízení koncových uzlů v síti. K tomuto řízení slouží modul `lifecycleController` a podmodul `scenarioManager`. Začneme konfiguračním souborem, kde za poslední řádek vložíme následující kód:

```
[Config VyprseniAdresy]
network = DHCPNetwork
**.hasStatus = true
*.n = 2
**.scenarioManager.script = xmldoc("DHCPShutdown.xml")
**.numUdpApps = 1

**.klient[0].udpApp[0].typename = "DHCPClient"
**.klient[0].udpApp[0].startTime = 20s

**.klient[1].udpApp[0].typename = "DHCPClient"
**.klient[1].udpApp[0].startTime = 175s

**.DHCPServer.udpApp[0].typename = "DHCPServer"
**.DHCPServer.udpApp[0].subnetMask = "255.255.255.0"
**.DHCPServer.udpApp[0].ipAddressStart = "192.168.1.101"
**.DHCPServer.udpApp[0].maxNumClients = 100
**.DHCPServer.udpApp[0].gateway = "192.168.1.1"
**.DHCPServer.udpApp[0].dns = ""
**.DHCPServer.udpApp[0].leaseTime = 30s
```

Tentokrát budeme vycházet ze dvou koncových bodů. Kromě původně vysvětleného kódu je zde manažeru scénáře přiřazen `XML` skriptovací soubor `DHCPShutdown.xml`, který teď vytvoříme. Pokud je dodržena správná syntaxe kódu, manažer tento soubor vykoná. Soubor vytvoříme pravým klikem na projekt, poté **New » File**. Do dialogu vytváření souboru napíšeme jméno souboru **DHCPShutdown.xml** a klikneme na tlačítko **Finish**. Do právě vytvořeného souboru vložíme následující kód, kterým nastavujeme vypnutí DHCP serveru v čase 80 sekund a následovné zapnutí DHCP serveru v čase 170 sekund.

```
<scenario>
    <at t="80.0s">
        <tell module="lifecycleController" target="DHCPServer"
operation="NodeShutdownOperation"/>
    </at>
    <at t="170s">
        <tell module="lifecycleController" target="DHCPServer"
operation="NodeStartOperation"/>
    </at>
</scenario>
```

Nyní nezbývá nic jiného než scénář odsimulovat. Při výběru simulované topologie z kontextového výběru tentokrát volíme scénář **VyprseniAdresy**. Simulaci provádíme klávesou **F5**, při normální rychlosti pro možnost sledování provozu. K ovládání rychlosti animace slouží posuvník **Animation Speed**.

Jednotlivou situaci si popíšeme. První koncová stanice si v čase započetí zapůjčí adresu od DHCP serveru s určitou dobou záznamu (v našem případě 30 vteřin). Poté pokaždé v polovině doby záznamu si stanice zažádá DHCP server o prodloužení záznamu (to je každých 15 vteřin). Vypnutí serveru v čase 80 sekund způsobí, že klient selže při snaze o prodloužení záznamu v čase 80 sekund (50 % doby záznamu) a v čase zhruba 91 sekund (87,5 % času doby záznamu). Tímto by klientovi měl daný záznam v čase 95 s vypršet a přiřazená adresa by měla v simulaci zmizet, nicméně se tak nestane. Jedná se o chybu, která doposud nebyla opravena, je však reportována a zařazena na opravu s další verzí INET Framework. Situace může připomínat protokol BOOTP, u kterého je přidělen záznam na stálo. Záznam na DHCP server však zmizí a při zapnutí serveru (170 s) a následovné žádosti o zapůjčení záznamu od klienta dvě je druhé koncové stanici přiřazena stejná adresa, jakou má koncová stanice jedna. Tímto nastává konflikt v síti.

6.2.4 Vypracování 3. část – Restart DHCP serveru

Tato část úlohy pojednává o situaci, kdy v čase od 80 s do 90 s proběhne vynucený restart DHCP serveru, čímž server ztratí svoji tabulku záznamů. Následně budeme sledovat reakci klienta a serveru na tuto nečekanou změnu.

Do konfiguračního souboru `.ini` vložíme následující sekvenci příkazů:

```
[Config DHCPRestart]
network = DHCPNetwork
**.hasStatus = true

*.n = 1
**.scenarioManager.script = xmldoc("DHCPRestart.xml")
**.numUdpApps = 1
**.klient[0].udpApp[0].typename = "DHCPClient"
**.klient[0].udpApp[0].startTime = 20s
**.DHCPServer.udpApp[0].typename = "DHCPServer"
**.DHCPServer.udpApp[0].subnetMask = "255.255.255.0"
**.DHCPServer.udpApp[0].ipAddressStart = "192.168.1.101"
**.DHCPServer.udpApp[0].maxNumClients = 100
**.DHCPServer.udpApp[0].gateway = "192.168.1.1"
**.DHCPServer.udpApp[0].dns = ""
**.DHCPServer.udpApp[0].leaseTime = 80s
```

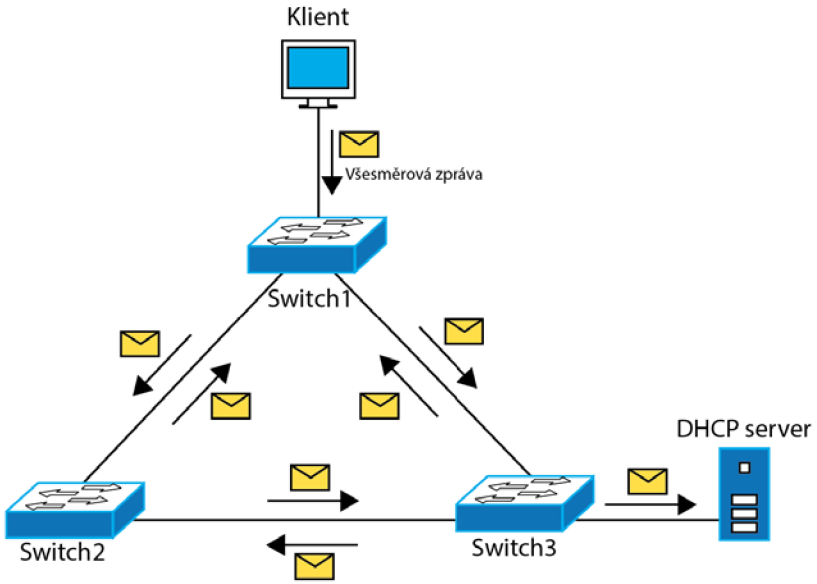
Nastavení scénáře je téměř totožné s předešlým, změna je pouze v počtu koncových zařízení a v jiném skriptovacím souboru. Skriptovací soubor `DHCPRestart.xml` vytvoříme podle předchozího popisu a vložíme do něj kód, který zajistí desetisekundový restart serveru:

```
    <scenario>
        <at t="80.0s">
            <tell module="lifecycleController" target="DHCPServer"
operation="NodeShutdownOperation"/>
        </at>
        <at t="90.0s">
            <tell module="lifecycleController" target="DHCPServer"
operation="NodeStartOperation"/>
        </at>
    </scenario>
```

Syntaxe mezer v jednotlivých sekcích musí být dodržena, jinak simulace skončí chybou. Spustíme simulaci, tentokrát se zaměříme na dobu po výpadku serveru. Pozorujte vyměněné zprávy a zaznamenejte si je.

6.2.5 Vypracování 4. část – Všesměrová bouře

Tato část se zabývá možným problémem smyček. Nejdříve si zkuste představit danou situaci – topologii, ve které je první klient připojen k přepínači switch1 a DHCP server připojen k přepínači switch3, navíc je zde ještě přepínač switch2 (viz Obrázek 28).



Obrázek 28: Přenos všesměrové zprávy ve smyčce

Přepínače budou mezi sebou propojeny každý s každým. Pokud bude první koncová stanice potřebovat získat IP adresu a další údaje od DHCP serveru, musí poslat všesměrovou zprávu pro nalezení DHCP serveru. Přepínač switch1 pošle všesměrovou zprávu ostatním přepínačům připojeným v síti, ty udělají totéž. Tímto způsobem dojde k úplnému zahlcení sítě. V síti se najednou bude vyskytovat více všesměrových zpráv,

než je síť schopna zpracovat. Problém se zahlcením smyčky se nazývá všesměrová bouře, my si takovou situaci vyzkoušíme.

Na konec souboru `Network.ned` vložíme kód, kterým vytvoříme novou síť:

```
network BroadcastStorm
{
    parameters:
        @display("bgb=567.60004,450");
    submodules:
        klient: StandardHost {
            @display("p=39.600002,136.8,ri,200,200,200");
        }
        DHCPServer: StandardHost {
            parameters:
                @display("p=416.40002,174;i=device/server");
            gates:
                ethg[1];
        }
        switch1: EtherSwitch {
            parameters:
                @display("p=238.8,243.6");
        }
        switch3: EtherSwitch {
            parameters:
                @display("p=313.2,370.80002");
        }
        switch2: EtherSwitch {
            parameters:
                @display("p=153.6,370.80002");
        }

        configurator: IPv4NetworkConfigurator {
            parameters:
                config = xml("<config><interface hosts='DHCPServer'
address='192.168.1.1' netmask='255.255.255.0' /></config>");
                @display("p=232.8,43.2");
        }
    connections:
        klient.ethg++ <--> NormalPath <--> switch1.ethg++;
        switch3.ethg++ <--> NormalPath <--> DHCPServer.ethg++;
        switch1.ethg++ <--> NormalPath <--> switch2.ethg++;
        switch2.ethg++ <--> NormalPath <--> switch3.ethg++;
        switch1.ethg++ <--> NormalPath <--> switch3.ethg++;
}
```

Tím jsme vytvořili síť se jménem `BroadcastStorm`. Elementy sítě zůstávají stejné, přibyli pouze dva přepínače a jejich propojení. Už zbývá jen přidat část kódu do konfiguračního souboru a můžeme spustit simulaci.

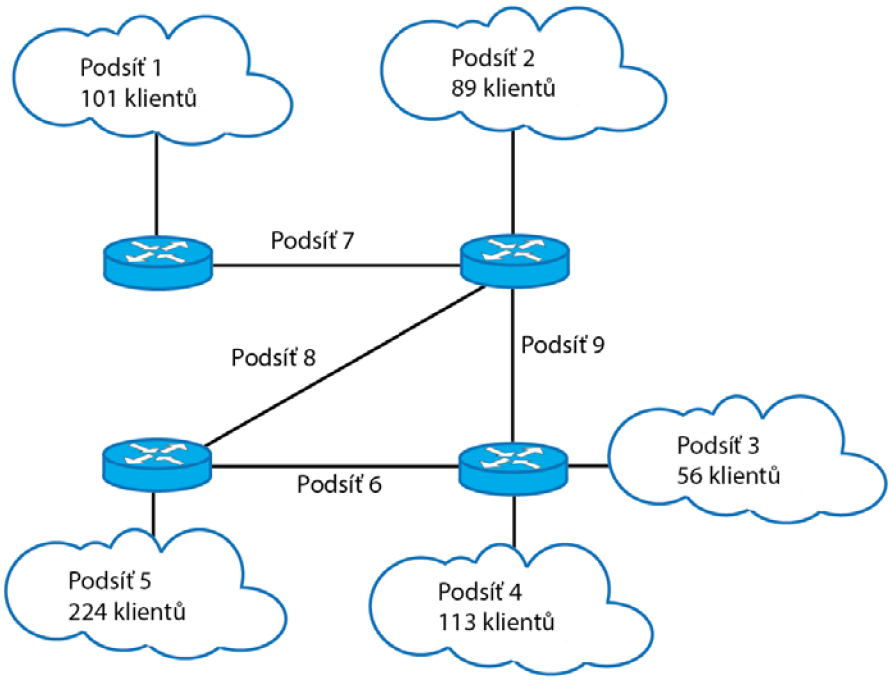
```
[Config BroadcastBoure]
network = broadcaststorm
**.numUdpApps = 1
**.klient.udpApp[0].typename = "DHCPClient"
**.klient.udpApp[0].startTime = 0s
**.DHCPServer.udpApp[0].typename = "DHCPServer"
**.DHCPServer.udpApp[0].subnetMask = "255.255.255.0"
**.DHCPServer.udpApp[0].ipAddressStart = "192.168.1.101"
**.DHCPServer.udpApp[0].maxNumClients = 100
**.DHCPServer.udpApp[0].gateway = "192.168.1.1"
**.DHCPServer.udpApp[0].dns = ""
**.DHCPServer.udpApp[0].leaseTime = 80s
```

Po spuštění simulace můžeme pozorovat první všesměrovou zprávu vyslanou klientem, jak uvízne ve smyčce přepínačů. Každá další všesměrová zpráva dopomáhá k postupnému zahlcení sítě, roste zpoždění přenosu a klient dostane přidělený záznam oproti normální situaci o něco málo později. Nicméně pokud urychlíme simulaci pomocí tlačítka Express (klávesa F7), dojde k nečekanému přerušení simulace z důvodu překročení limitu přednastavené výstupní fronty.

V praxi se výskytu smyček zabraňuje pomocí protokolu STP (Spanning Tree Protocol) a RSTP (Rapid STP), kdy je protokol schopen odstranit smyčky pomocí sestavení topologie sítě do grafu a odstranit redundantní části grafu tak, aby byla kostra grafu zachována.

6.2.6 Vypracování 5. část – Neefektivní adresace podsítí

Tato část si klade za cíl seznámit studenty nejdříve s neefektivní a později s efektivní adresací podsítí. Jako příklad topologie poslouží Obrázek 29.



Obrázek 29: Topologie pro výpočet adres podsítí

Ve stanovené topologii se nachází čtyři směrovače a pět LAN sítí. Nutno podotknout, že jako síť se počítá i trasa mezi směrovači. Pro konkrétní případ budeme vycházet z adresy 183.14.0.0/20.

Ze začátku si musíme přepočítat masku do desítkového tvaru. Víme, že maska se skládá celkem ze 4 oktetů (32 bitů). Pokud tedy rozdělíme masku na 4 stejně dlouhé úseky, tedy po osmi bitech, dostaneme binární zápis masky. První dva oktety budou samé jedničky, třetí oktet je již neúplný – odpočítáním od 16 bitů dostaneme celkem čtyři jedničky, zbytek jsou nuly. Po získání masky v binárním tvaru už můžeme masku jednoduše převést do dekadického zápisu. Dostaneme:

Tabulka 3: Přepočet masky do binárního tvaru

Maska v binárním zápisu	11111111	11111111	11110000	00000000
Maska v decimálním zápisu	255.	255.	240.	0

Pomocí masky se můžeme rozhodnout, kolik podsítí vytvořit. Podsítě vytváříme v masce přidáním binárních jedniček do části klienta. Pokud budeme chtít vytvořit 6 podsítí, ve vztahu $2^n = x$ musí být $x \geq 6$. Podmínce nejblíže vyhovuje číslo $n = 3$, proto do masky postupně přidáme 3 bity, každý bit pak k jednotlivé podsíti. V naší ukázce budeme potřebovat do masky přidat takový počet bitů, aby bylo možné vytvořit celkem devět podsítí. Pokud dosadíme do zmíněné rovnice $x = 9$, zjistíme, že nejbližší možné n je rovno čtyřem. Nyní již můžeme jednotlivé podsítě přiřazovat potřebným sítím:

Tabulka 4: Výpočet adres pomocí masky

Adresy	Oktet 1	Oktet 2	Oktet 3	Oktet 4
Adresa v binárním tvaru	10110111	00001110	00000000	00000000
Maska v binárním tvaru	11111111	11111111	11111111	00000000
Adresa 1. podsítě	10110111	00001110	00000000	00000000
Adresa 2. podsítě	10110111	00001110	00000001	00000000
Adresa 3. podsítě	10110111	00001110	00000010	00000000
Adresa 4. podsítě	10110111	00001110	00000011	00000000
Adresa 5. podsítě	10110111	00001110	00000100	00000000
Adresa 6. podsítě	10110111	00001110	00000101	00000000
Adresa 7. podsítě	10110111	00001110	00000110	00000000
Adresa 8. podsítě	10110111	00001110	00000111	00000000
Adresa 9. podsítě	10110111	00001110	00001000	00000000

Tím máme vypočítanou adresu všech potřebných sítí. Pomocí adres dílčích sítí můžeme stanovit adresy klientů a všesměrovou adresu sítě. Všesměrová adresa je zpravidla poslední možnou adresou v síti. Pokud tedy máme adresu sítě a víme, že v podsíti je celkově 256 adres (2^8 adres), odpočítáním dospějeme ke všesměrové adrese. Využití adres se vypočítá jako podíl celkového počtu adres hostů v síti a skutečného počtu hostů v jednotlivé síti. Možným výpočtem dostaneme následující adresy:

Tabulka 5: Vypočtené adresy sítě, všesměrové adresy a rozsahy

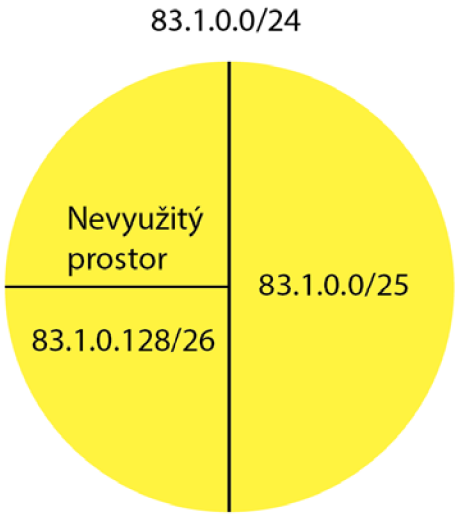
Podsíť/Počet hostů	Adresa sítě	Všesměrová adresa	Rozsah adres hostů	Využití adres [%]
Podsíť 1/101 hostů	183.14.0.0/24	183.14.0.255	183.14.0.1 - 183.14.0.254	39,8
Podsíť 2/89 hostů	183.14.1.0/24	183.14.1.255	183.14.1.1 - 183.14.1.254	35
Podsíť 3/56 hostů	183.14.2.0/24	183.14.2.255	183.14.2.1 - 183.14.2.254	22
Podsíť 4/113 hostů	183.14.3.0/24	183.14.3.255	183.14.3.1 - 183.14.3.254	44,5
Podsíť 5/224 hostů	183.14.4.0/24	183.14.4.255	183.14.4.1 - 183.14.4.254	88,2
Podsíť 6/2 hosti	183.14.5.0/24	183.14.5.255	183.14.5.1 - 183.14.5.254	0,8
Podsíť 7/2 hosti	183.14.6.0/24	183.14.6.255	183.14.6.1 - 183.14.6.254	0,8
Podsíť 8/2 hosti	183.14.7.0/24	183.14.7.255	183.14.7.1 - 183.14.7.254	0,8
Podsíť 9/2 hosti	183.14.8.0/24	183.14.8.255	183.14.8.1 - 183.14.8.254	0,8

Z výpočtu využití adres lze vidět neefektivnost rozdělení, kdy pro síť, která potřebuje pouze dvě adresy, existuje dalších 252 nevyužitých adres. Pro zvýšení efektivnosti přidělování se používá metoda adresování podsítě s maskou proměnlivé délky (VLSM – Variable-Length Subnet Mask).

Celkový počet využitých adres je 591, celkový počet možných adres u námi přiděleného adresního rozsahu je 2286. Po odečtení dostáváme celkem 1695 nevyužitých adres.

6.2.7 Vypracování 6. část – Efektivní adresace podsítí VLSM

Efektivní adresace pomocí metody VLSM využívá nevyužitých adres v daném rozsahu. Představte si dvě sítě, jedna s potřebou 100 hostů, druhá s potřebou 50 hostů a přidělenou adresou 83.1.0.0/24. Tuto adresu můžeme rozdělit do několika podsítí, které v součtu dají počet původního rozsahu. Tuto adresu lze rozdělit na dvě adresy o prefixu /25, čtyři adresy o prefixu /26 atd. Adresy pro dané sítě by tedy byly 83.1.0.0/25 a 83.1.0.128/26. Lze tak ušetřit obrovský adresový prostor.



Obrázek 30: Ukázka rozložení adresového prostoru

Uvažujme stejnou topologii se stejným počtem hostů a stejnou přiřazenou adresou jako při neefektivním adresování. V prvním kroku musíme opakovat postup – přepočítat masku do binární podoby. Další krok je obdobný, tedy zjistíme si potřebný počet bitů pro všechny sítě. Se zjištěnými bity již lze počítat adresy jednotlivých sítí. Vhodným způsobem je počítat adresy od sítí s nejvyšší potřebou hostů po ty s nejnižší potřebou. Nejbližší vhodný prefix pro podsíť 5 se 224 koncovými stanicemi je /24, tím je vhodné začít. Další tři podsítě mají adresovou potřebu do 128 hostů (prefix /25), jedna podsíť potřebu s prefixem 64 klientů a zbylé podsítě každá po dvou adresách (prefix /30). Po implementaci VLSM bude tabulka adres vypadat následovně:

Tabulka 6: Vypočtené adresy sítě, všesměrové adresy a rozsahy

Podsíť/Počet hostů	**Adresa sítě**	**Všesměrová adresa**	**Rozsah adres hostů**	**Využití adres [%]**
Podsíť 5/224 hostů	183.14.0.0/24	183.14.0.255	183.14.0.1 - 183.14.0.254	88,2
Podsíť 4/113 hostů	183.14.1.0/25	183.14.1.127	183.14.1.1 - 183.14.1.126	89,7
Podsíť 1/101 hostů	183.14.1.128/25	183.14.1.255	183.14.1.129 - 183.14.1.254	80,2
Podsíť 2/89 hostů	183.14.2.0/25	183.14.2.127	183.14.2.1 - 183.14.2.126	70,6
Podsíť 3/56 hostů	183.14.2.128/26	183.14.2.191	183.14.2.129 - 183.14.2.190	90,3
Podsíť 6/2 hosti	183.14.2.192/30	183.14.2.195	183.14.2.193 - 183.14.2.194	100
Podsíť 7/2 hosti	183.14.2.196/30	183.14.2.199	183.14.2.197 - 183.14.2.198	100

Podsíť 8/2 hosti	183.14.2.200/30	183.14.2.203	183.14.2.201-183.14.2.202	100
Podsíť 9/2 hosti	183.14.2.204/30	183.14.2.207	183.14.2.205 - 193.14.2.206	100

Celkový počet využitých adres je opět 591, celkový počet možných adres u námi přiděleného adresního prostoru je 702. Po odečtení dostáváme celkem 111 nevyužitých adres. Srovnáme-li výsledky, můžeme pozorovat mnohonásobně nižší plýtvání adres u VLSM metody.

6.2.8 Vypracování 7. část – OMNet++ adresace

V poslední části si vytvoříme ještě jednu topologii a seznámíme se s podrobnější adresací pomocí skriptovacího souboru a nastavování podmodulu `IPv4NetworkConfigurator`. Adresy budeme nastavovat celkem čtyřem podsítím, které jsou propojeny směrovačem.

```
network StaticNetwork
{
    parameters:
        int m;
        int n;
        int o;
        int p;

        @display("bgb=1470,996");
    submodules:
        switchA: EtherSwitch {
            parameters:
                @display("p=141.6,493.2");
        }
        switchB: EtherSwitch {
            parameters:
                @display("p=516,616.8");
        }
        switchC: EtherSwitch {
            parameters:
                @display("p=970.80005,826.80005");
        }
        switchD: EtherSwitch {
            parameters:
                @display("p=1138.8,378");
        }
        router: Router {
            @display("p=546,150");
        }
        klientA[m]: StandardHost {
            @display("p=36,393.6,ri,100,100,100");
        }
        klientB[n]: StandardHost {
             @display("p=411.6,514.8,ri,100,100,100");
        }
        klientC[o]: StandardHost {
             @display("p=858,700.8,ri,120,120,120");
        }
```

```
        klientD[p]: StandardHost {
              @display("p=1040.4,280.80002,ri,100,100,100");
        }
        configurator: IPv4NetworkConfigurator {
            parameters:
                config = xmldoc("Adresace.xml");
                @display("p=546,36");
        }

    connections:
        switchA.ethg++ <--> NormalPath <--> router.ethg++;
        switchB.ethg++ <--> NormalPath <--> router.ethg++;
        switchC.ethg++ <--> NormalPath <--> router.ethg++;
        switchD.ethg++ <--> NormalPath <--> router.ethg++;

        for i=0..m-1 {
            klientA[i].ethg++ <--> NormalPath <--> switchA.ethg++;
        }

        for i=0..n-1 {
            klientB[i].ethg++ <--> NormalPath <--> switchB.ethg++;
        }
        for i=0..o-1 {
            klientC[i].ethg++ <--> NormalPath <--> switchC.ethg++;
        }

        for i=0..p-1 {
            klientD[i].ethg++ <--> NormalPath <--> switchD.ethg++;
        }
}
```

Proměnné *m*, *n*, *o*, *p* zde slouží pro vlastní nastavení počtu koncových stanic na každou podsíť. Důležité je zmínit možné omezení simulačního programu při nastavení více než 30 koncových stanic, kdy se program stává výpočetně náročným. Proto budeme zamýšlet podsítě s menším množstvím klientů.

Další nastavení se týká konfiguračního souboru `.ini`. Vytvoříme zde část pro nastavení parametrů sítě:

```
[Config StatickaKonfigurace]
network = StaticNetwork
*.m = 6
*.n = 15
*.o = 17
*.p = 3
```

Pro účely adresace musíme vytvořit skriptovací soubor se jménem `Adresace.xml`, do něj vložíme následující kód:

```
<config>
  <interface hosts='router' names='eth0' address='#.#.#.#'
netmask='#.#.#.#'/>
  <interface hosts='klientA[*]' address='#.#.#.x'
netmask='#.#.#.#'/>
  <interface hosts='router' names='eth1' address='#.#.#.#'
netmask='#.#.#.#'/>
  <interface hosts='klientB[*]' address='#.#.#.x'
netmask='#.#.#.#'/>
  <interface hosts='router' names='eth2' address='#.#.#.#'
netmask='#.#.#.#'/>
  <interface hosts='klientC[*]' address='#.#.#.x'
netmask='#.#.#.#'/>
  <interface hosts='router' names='eth3' address='#.#.#.#'
netmask='#.#.#.#'/>
  <interface hosts='klientD[*]' address='#.#.#.x'
netmask='#.#.#.#'/>
</config>
```

Nyní je vaším úkolem nastavit adresy jednotlivých sítí. Na pozici # nastavujte adresu, pozici x neměňte – dojde tak k vypočítání zbylého rozsahu automaticky. Adresu si zvolte vlastní. Nezapomeňte však, že je nutné počítat také s adresou rozhraní na směrovači (nastavuje vždy první řádek, volte jako adresu sítě + 1). Nastavení adres a masky v části nastavení rozhraní (`<interface hosts='router' names='eth0' address='#.#.#.x'`) a v části nastavení klientů (`<interface hosts='klientA1[*]' address='#.#.#.x' netmask='#.#.#.#'/>`) bude vždy stejné.

Jako kontrolu správné konfigurace adres indikuje bezchybné spuštění simulace. Lze také použít lupu v simulaci, pokud se v daném kruhu nachází příliš mnoho koncových stanic, pokud je kontrola adresace nepřehledná.

6.2.9 Samostatný úkol

Vytvořte scénář vycházející z prvního scénáře (Obrázek 25), kde počet koncových stanic bude 5 a počet adres k přiřazení 4. Dobu záznamu nastavte na 30 vteřin. Zbytek nastavení může odpovídat prvnímu scénáři. Zajistěte, aby se v čase 100 s klient1 odpojil (například pomocí skriptovacího souboru a příkazu `NodeShutdownOperation`).

6.2.10 Otázky

1. Pomocí jakých portů protokol DHCP komunikuje?

2. Jakým způsobem šíření a jaké zprávy si musí klient s DHCP serverem vyměnit, aby klient získal patřičné údaje?

3. Co se stane, když je počet koncových stanic větší, než možné přidělené adresy DHCP?

4. Jakým způsobem je zabráněno, aby paket (síťová vrstva) nebyl v síti v případě výskytu směrovacích smyček směrován do nekonečna?

5. K čemu slouží maska sítě?

6. K jakému účelu slouží následující adresy: 173.16.5.63/29, 192.168.1.20/30, 98.10.57.1/26, 127.0.0.1/8 (adresa sítě x adresa koncové stanice x všesměrová adresa).

7. Kolik bitů je potřeba vzít z části adresy klienta pro vytvoření celkem jedenácti podsítí?

7 ZÁVĚR

Cílem bakalářské práce bylo vytvořit dvě plnohodnotné úlohy v simulačním prostředí OMNeT++ pro studenty druhého ročníku bakalářského studia. Aby bylo tohoto cíle dosaženo, musel autor práce nastudovat tematické okruhy, jako aplikační vrstvu modelu TCP/IP a seznámit se blíže s protokoly aplikační vrstvy, dále obecně s IP adresací a rozdělováním sítí na menší podsítě. Nezbytným bylo pochopení funkcionality simulačního prostředí OMNeT++, které bylo třeba důkladně prozkoumat. Na základě získaných informací a zkušeností potom vytvořit laboratorní úlohy v síťovém simulátoru OMNeT++, které se zaobírají prostudovanou problematikou. Práce se výhradně člení na část teoretickou (kapitola 1-4) a část praktickou (kapitola 5 a 6).

První navržená úloha zkoumá diference aplikačních protokolů, konkrétně aplikací HTTP, Telnet, FTP, VOIP a streamování videa. První scénář sleduje provoz aplikací při nesouběžném běhu, na konci scénáře je zobrazen graf odeslaných paketů a graf využití linek. Tyto grafy slouží pro porovnání s druhým scénářem, který je zaměřen na pozorování běhu aplikací při souběžném provozu, navíc se sleduje koncové a obousměrné zpoždění aplikací. Ve třetím scénáři úlohy se pozoruje chování aplikací při zvýšené chybovosti přenosových cest, konec třetího scénáře je zaměřen na zahazování paketů a využití linky serveru pro předem stanovené parametry chybování. Předposlední část zkoumá chování aplikací při výpadku trasy serveru. Poslední scénář sleduje chování aplikací při snížení kapacity přenosového média.

Druhá navržená úloha se zabývá dynamickou a statickou adresací koncových stanic. Scénář 1 se soustředí na výměnu zpráv mezi DHCP serverem a koncovými stanicemi. Druhý scénář se zaměřuje na vypršení doby záznamu (lease time) DHCP serveru. Je potřeba zmínit, že se v rozšíření INET Framework nachází chyba, kdy adresy po vypršení doby záznamu nezmizí – očekává se, že při začlenění laboratorních úloh do výuky bude tato chyba již opravena. Scénář 3 je založen na restartu DHCP serveru a následné ztrátě tabulky vypůjčených adres, pozoruje se zamítnutí žádosti koncové stanice o prodloužení záznamu. Účelem čtvrtého scénáře je poukázání na problém všesměrových smyček. Pátá a šestá část úlohy se věnuje neefektivní a efektivní adresaci podsítí a výpočtem využití adres pro jednotlivé podsítě. Úkolem posledního scénáře je staticky nastavit adresy v simulačním prostředí pro jednotlivé podsítě.

Na začátků každého návodu je v krátkosti shrnuta teorie zabývající se danou problematikou. V postupu je pak podrobně rozepsáno založení nového projektu, vytvoření a definování souboru pro popis topologie sítě (souboru NED) a nastavení konfiguračního souboru. Pokud je tomu třeba, zakládá se v některých částech `.xml` skriptovací soubor. Na konci se pak nachází otázky, na které by měli studenti na konci hodiny odpovědět.

LITERATURA

[1] *OMNeT++ Simulation Manual* [online]. Budapest: OpenSim, 2016 [cit. 14. 11. 2016]. Dostupné z URL: <https://omnetpp.org/doc/omnetpp/manual/#sec:introduction:what-is-omnetpp>.

[2] *INET Framework for OMNeT++* [online]. Budapest, 2015 [cit. 14. 11. 2016]. Dostupné z URL: <https://omnetpp.org/doc/inet/api-current/neddoc/index.html>.

[3] JEŘÁBEK, J. *Komunikační technologie* [online]. Brno: Vysoké učení technické v Brně, 2013 [cit. 2016-11-14]. ISBN 978-80-214-4713-4. Dostupné z URL: <https://www.vutbr.cz/www_base/priloha.php?dpid=71518>.

[4] TANENBAUM, Andrew S. a D. WETHERALL. *Computer networks*. 5th ed. Boston: Pearson Prentice Hall, c2011 [cit. 6. 4. 2017]. ISBN 978-0-13-212695-3.

[5] COMER, D. *Internetworking with TCP/IP* 5th ed. Upper Saddle River, N.J.: Pearson Prentice Hall, 2006 [cit. 14. 11. 2016]. ISBN 01-318-7671-6.

[6] RANSOME, J. F., RITTINGHOUSE, J. W. *VOIP security* Boston: Elsevier, 2005 [cit. 17. 11. 2016]. ISBN 15-555-8332-6. (PŮVODNĚ 5)

[7] FOROUZAN, Behrouz A. TCP/IP *Protocol Suite*. Fourth edition, Boston: McGraw-HillHigherEducation, 2010 [cit. 6. 4. 2017], xxxv, 979 s. ISBN 978-0-07-337604-2.

[8] TANENBAUM, Andrew S. a D. WETHERALL. *Cisco Networking Academy Program: CCNA 1 and 2 companion guide*. 3rd ed. Indianapolis, Ind.: Cisco, c2003 [cit. 9. 4. 2017]. ISBN 15-871-3110-2.

[9] DOSTÁLEK, Libor a Alena KABELOVÁ. *Velký průvodce protokoly TCP/IP a systémem DNS*. 2. aktualiz. vyd. Praha: Computer Press, 2000 [cit. 9. 4. 2017]. ISBN 80-722-6323-4.

[10] Morreale, P., Terplan, K.: *The CRC Handbook of Modern Telecommunications, CRC Press*, New York 2001 [cit. 17. 4. 2017]. ISBN 978-142-0078-008.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ASCII	American Standard Code for Information Interchange
BGP	Border Gateway Protocol
BOOTP	Bootstrap Protocol
CSV	Comma-Separated Values
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
Diffserv	Differentiated services
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO/OSI	International Standards Organisation / Open System Interconnection
NED	Network Description
NVT	Network Virtual Terminal
OSPF	Open Shortest Path First
POP3	Post Office Protocol 3
PPP	Point-to-Point Protocol
QoS	Quality of Services
RSTP	Rapid Spanning Tree Protocol
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol

TCP/IP	Transmission Control Protocol/Internet Protocol
Telnet	Telecommunication Network Protocol
UDP	User Datagram Protocol
VLSM	Variable Length Subnet Mask
VOIP	Voice Over Internet Protocol
VPN	Virtual Private Network

SEZNAM PŘÍLOH

A OBSAH PŘÍLOHY NA CD

A.1 Zdrojové soubory

Zdrojové soubory jednotlivých úloh jsou uloženy ve složkách Lab*_Zdrojove_kody. Složky obsahují jak soubory topologie sítě *.NED, tak i konfigurační soubory *.ini a skriptovací soubory *.xml.

A.2 Laboratorní návody

Návody k oběma laboratorním úlohám se nacházejí ve složce Navody/Lab*.docx. Pro možnost pozdější úpravy jsou oba návody ve formátu dokumentu.

A.3 Dokumentace

Přiložené CD obsahuje také dokumentaci ve formátu pdf, která se nachází v kořenovém adresáři CD.