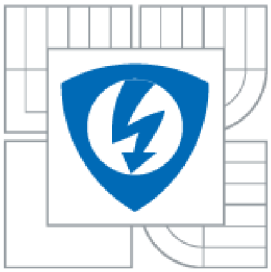
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

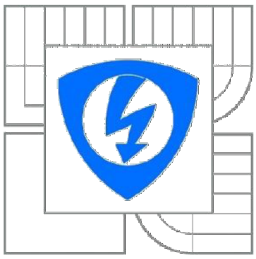
SIMULACE DATOVÝCH SÍTÍ S VYUŽITÍM PŘÍMÉHO VYKONÁVÁNÍ KÓDU

DIPLOMOVÁ PRÁCE
MASTER`S THESIS

AUTOR PRÁCE
AUTHOR
Bc. MICHAL TRÁVNÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. RADKO KRKOŠ

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Michal Trávníček	***ID:***	115298
Ročník:	2	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Simulácia dátových sietí s využitím priameho vykonávania kódu

POKYNY PRO VYPRACOVÁNÍ:

Obeznamte se s diskrétní simulací, výkonným nástrojem pro analýzu a syntézu datových sítí. Zamerte se na zvýšení hodnovernosti simulací získaných dat použitím techniky prímého vykonávání kódu a prostudujte možnosti této technologie. Analyzujte podporu techniky prímého vykonávání kódu v simulacních nástrojích, zamerte se zejména na nástroj NS-3. Experimentálne overte použití prímého vykonávání kódu v diskrétní simulaci. Vytvorte v simulacním nástroji NS-3 topologii pozustávající z uzlu zabezpecujících smerování. S využitím techniky prímého vykonávání kódu spustte na techto uzlech smerovací démon Quagga a jeho podporu pro OSPFD a porovnejte výsledky simulace s využitím démonu s klasickou implementací OSPF protokolu v NS-3. Podrobne popište postup rozchození podpory prímého vykonávání kódu v NS-3 a tvorbu scénáru s využitím prímého vykonávání kódu, speciálne smerovacího démonu Quagga.

DOPORUČENÁ LITERATURA:

[1] STALLINGS, William. High-speed Networks and internets: Performance and Quality of Service. 2nd ed. Upper Saddle River: Prentice Hall, 2002, 715 s. ISBN 01-303-2221-0.

[2] BERTSEKAS, Dimitri P. a Robert G. GALLAGER. Data Networks. 2nd ed. Englewood Cliffs, N.J.: Prentice Hall, 1992, xix, 556 p. ISBN 01-320-0916-1. Dostupné z: http://web.mit.edu/dimitrib/www/datanets.html

Termín zadání:	11.2.2013	***Termín odevzdání:***	29. 5. 2013

Vedoucí práce: Ing. Radko Krkoš
Konzultanti diplomové práce:

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

ANOTACE

Cílem diplomové práce bylo popsat simulace reálné sítě, poukázat na chyby a nedokonalosti při těchto simulacích a snažit se najít technologie a řešení, které by přispělo ke zvýšení hodnověrnosti simulací. Pro účel simulací byl vybrán simulační nástroj ns-3, který podporuje metodu přímého vykonávání kódu spolu se softwarovým balíkem Quagga.

V diplomové práci byl popsán internetový protokol IPV4, protokol UDP, základní principy směrování a podrobně vysvětlen směrovací protokol OSPF. Dále byla v práci byla věnována pozornost obecnému popisu simulací, jeho rozdělení a následně detailnějšímu popisu diskrétní simulaci, která je využívána simulačním nástrojem ns-3. Teoretický popis simulačního nástroje ns-3 byl obsahem další kapitoly, byl popsán vývoj, koncepce ns-3 a jednotlivé objekty využívané při vytváření modelů. Následující kapitola popisuje metodu přímého vykonávání kódu a balíku Quagga, které slouží k dosažení hodnověrnějších výsledku při simulaci.

Praktická část diplomové práce přináší vysvětlení jednotlivých kroků při instalaci a nastavení u všech tří implementací ns-3 v linuxové distribuci Ubuntu. Na základě teoretických znalostí byl navrhnut odpovídající model sítě, detailně demonstrováno vytváření simulace s následnou analýzou a srovnáním výsledků mezi implementacemi v ns-3.

KLÍČOVÁ SLOVA

diskrétní simulace, IPv4, UDP, OSPF, ns-3, přímé vykonávání kódu, DCE, Quagga

ABSTRACT

Purpose of this master thesis was to explain simulation of real network, point to the imperfection during these simulations and try to find out technologies and solutions that could improve credibility of simulation. For this purpose, network simulator ns-3 has been chosen which supports direct code execution together with Quagga software suite.

Master thesis describes internet protocol IPv4, UDP protocol, basics of routing and detailed explanation of OSPF routing protocol. Focus in this thesis is also on general description of simulation, its kinds and detail explanation of discrete simulation that is used by ns-3 simulator. Next chapter explains theory of ns-3 simulator, its development, conceptual overview and objects. Direct code execution and Quagga that both help improve credibility of simulation, are described in last chapter of theoretical part of thesis.

In practical part of thesis is explained how to install and configure all three implemantaions of ns-3 simulator on Ubuntu, linux distribution. On the basis of obtained knowledge, the simulation model has been designed, also there has been given detailed explanation of how to create new model of simulation and how to analyze its results for all implementation.

KEYWORDS

descrete simulation, IPv4, UDP, OSPF, ns-3, direct code execution, DCE, Quagga

Bibliografická citace mé práce:

TRÁVNÍČEK, M. *Simulace datových sítí s využitím přímého vykonávání kódu.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 70 s. Vedoucí diplomové práce Ing. Radko Krkoš.

Prohlášení

Prohlašuji, že svoji diplomovou práci na téma Simulace datových sítí s využitím přímého vykonávání kódu jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb. Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonu (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 diplomové práce Ing. Radku Krkošovi, za velmi užitečnou metodickou pomoc a cenné rady při zpracování práce.

V Brně dne …… ……..

podpis autora

OBSAH

1 Úvod

Simulace reálné sítě nabízí možnost pomocí simulačních nástrojů vidět a analyzovat funkčnost a provoz v síti bez nutnosti samotného fyzického zapojování a nastavování síťových aktivních prvků. Problémem při simulování namodelovaných reálných sítí v simulačních nástrojích je však jejich nepříliš velká spolehlivost a nepřesnost výsledků s porovnáním s reálnými podmínkami. Tento problém se snaží co nejvíce eliminovat projekt vývoje nového simulačního nástroje ns-3. Tento nástroj byl vyvíjen od samotného počátku s velkým důrazem na přesnost a realističnost simulace a výsledků oproti starším simulačním nástrojům.

Ns-3 nabízí podporu přímého vykonávání kódu, což přináší z hlediska výsledků a průběhu simulace velké zpřesnění a vylepšení. Kombinací simulačního nástroje ns-3 a přímého vykonávání kódu se tedy lze využívat software přímo implementovaný v používaném systému a tím se zvyšuje hodnověrnost simulací.

Simulátor ns-3 navíc umožňuje použít metodu přímého vykonávání kódu spolu se softwarovým balíkem Quagga, který slouží k zajištění směrování v unixových systémech. Tato implementace by ještě dále měla zvýšit přesnost výsledků.

Diplomová práce je rozdělena na dvě části, na část teoretickou a praktickou. V části teoretické je za cíl popsat technologie, nástroje a programy, které budou použity v praktické části pro simulování vytvořených modelů sítí za pomocí přímého vykonávání kódu a balíčku Quagga. Teoreticky bude popsán protokol síťové vrstvy IPv4, protokol transportní vrstvy UDP, principy směrování a především protokol OSPF, který bude v simulaci využíván. Dále bude podrobně popsán princip simulací, diskrétní simulace a její rozdělení, bude představen simulační nástroj ns-3, jeho vývoj a koncepce. Nakonec přijde vysvětlení principu přímého vykonávání kódu a softwarového balíčku Quagga a jejich použití v simulátoru ns-3.

Praktická část diplomové práce má za cíl vysvětlit a detailně popsat docela složitou instalaci a nastavení třech používaných programů ns-3, ns-3 s využitím přímého vykonávání kódu a ns-3 s využitím přímého vykonávání kódu spolu s Quaggou. Následně bude důkladným popisem vysvětleno vytvoření modelu sítě a následných scénářů pro simulaci všech implementací, budou vytvořeny scénáře bez výpadku linky a s výpadkem linky. Na závěr praktické části budou demonstrovány výsledky všech scénářů a jejich vzájemné srovnání.

2 Internet Protokol (IP)

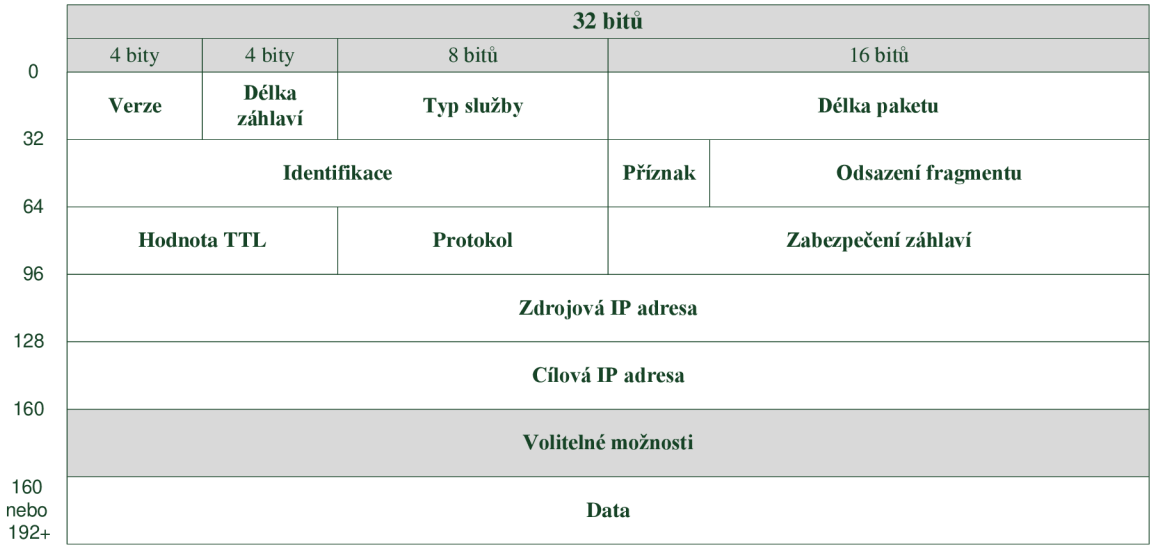
Internet Protokol je v dnešní době nejpoužívanější komunikační protokol. Jedná se o protokol síťové vrstvy poskytující funkci nespolehlivého, nespojovaného doručování paketů mezi koncovými body v TCP/IP sítích. Spojení mezi vysílací stanicí a cílovou stanicí není předem navázáno a protokol se také nestará o úspěšné doručení paketů sítí [1]. Jedná se o základní protokol používaný v síti Internet a téměř v každé nově instalované síti. Existují dvě varianty protokolu IP:

- Internet protokol verze 4 (IPv4)
- Internet protokol verze 6 (IPv6)

S již nastalým vyčerpáním IPv4 adres se IPv6 dostává do stále důležitější a nástupnické role, mimo jiné také díky většímu adresovému rozsahu. Většinové postavení v dnešních sítích však stále náleží čtvrté verzi internetového protokolu, proto bude pouze tato verze popsána a následně použita v simulacích v programu ns-3.

2.1 Internetový protokol verze 4

IPv4 je datově orientovaný protokol síťové vrstvy. Jedná se samozřejmě také o nespolehlivý, nespojovaný protokol. Paket je sítí směrován na základě síťových adres obsažených v záhlaví paketu.



Obr. 2.1: *Formát IPv4 paketu*

2.1.1 Formát paketu IPv4

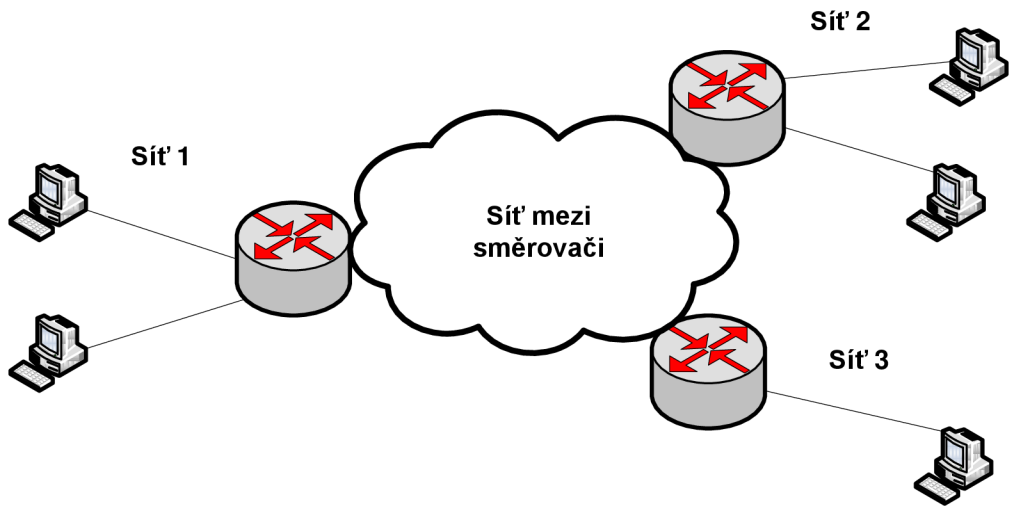
Paket IPv4 se skládá ze dvou základních částí:

- hlavičky paketu,
- datové částí.

Celková délka paketu včetně záhlaví je vždy násobkem 32 bitů. Maximální délka paketu je 65535 oktetů. Formát paketů je zobrazen na obr. 2.1 [2]. Popis jednotlivých polí zde nebude zahrnut, protože není pro tuto práci příliš podstatný. Podstatná jsou pro nás pole se Zdrojovou a Cílovou IP adresou, ve které jsou obsaženy IP adresy nezbytné pro správné fungování směrování na směrovačích v síti [1].

2.2 Směrování v IPv4 síti

Směrování je proces, který provádí síťové prvky (směrovače), ale také koncové stanice při průchodu paketů sítí. Každé síťové rozhraní, komunikující prostřednictvím protokolu IP, má přiřazeno jednoznačný identifikátor, tzv. IP adresu. Na základě těchto IP adres se na každém směrovači vytváří a udržuje směrovací tabulka. Směrovací tabulka obsahuje záznamy o cestách do různých sítí [2]. Bez možnosti směrování by nebylo možné doručit paket mezi různými sítěmi. Na obr. 2.2 lze vidět topologii sítě, ve které je nutné použít směrování pro přenos paketu z jedné sítě do jiné.



Obr. 2.2: *Schéma sítě pro ukázku směrování*

Při posílání paketu do jiné sítě je paket poslán na výchozí bránu (směrovač) v rámci své sítě a zde je porovnána cílová IP adresa (IP adresa cílové sítě) z hlavičky paketu s údaji obsaženými ve směrovací tabulce. Pokud je nalezena shoda, je paket

poslán na adresu sousedního směrovače, označována jako „next-hop“ adresa, a stejný proces se opakuje s paketem znovu. Podstata byla řečena, avšak stále nebyl popsán způsob, jakým se samotné směrovací tabulky naplňují údaji na směrovači [2].

Existují dva způsoby naplnění směrovací tabulky:

- **Statické** – směrovací tabulka se musí vyplňovat manuálně administrátorem na počátku a také vždy, když dojde ke změně topologie sítě, např. při výpadku linky, na změny události v síti tedy nebude nijak reagováno, vše se musí opravit nebo přidat opět ručně. Na první pohled tato možnost vypadá jako nevhodné řešení, avšak v určitých situacích je statická cesta výhodnější a šetří velikost směrovací tabulky a výpočetní zdroje směrovače [1],
- **Dynamické** – směrovací tabulky jsou naplněny automaticky (dynamicky) při zapojení směrovačů do sítě a jsou schopny reagovat na změny topologie sítě i změny v aktuálním provozu (zatížení sítě) zcela automaticky, využitím dynamických směrovacích protokolů [2].

2.3 Dynamické směrovací protokoly

Dynamické směrovací protokoly mají za úkol nalézt pro paket procházející sítí nejlepší cestu do sítě cílové. Pro tento úkol využívá aktuální záznamy ze směrovací tabulky. Směrovače, které jsou nastaveny na používání dynamických směrovacích protokolů, si mezi sebou vyměňují informace a udržují tak své směrovací tabulky aktuální. Při jakékoliv změně v síti se o této události směrovače vzájemně informují a aktualizují se směrovací tabulky. Způsob a detaily zasílání informací mezi směrovači se liší od použitého směrovacího protokolu, jelikož každý směrovací protokol pracuje odlišně a také pro výběr cesty používá odlišné metody.

Obecně směrovací protokol pro výběr nejlepší cesty k cíli používá kritérium zvané cena. Směrovací protokoly se od sebe liší používanou metrikou pro výpočet ceny trasy, proto by se měl směrovací protokol vybírat podle požadavků a rozsahu sítě [17].

Mezi metriku směrovacích protokolů pro výpočet ceny trasy patří například:

- počet skoků,
- kapacita linky,

- zpoždění,
- spolehlivost,
- zátěž linky.

Základem pro výpočet metriky a určení nejlepší cesty je algoritmus dynamického směrovacího protokolu. Používají se dva základní typy směrovacích algoritmů a jeden hybridní algoritmus [1]:

- **algoritmy pracující s vektorem vzdálenosti (distance vector)** – mezi směrovací protokoly využívající tento algoritmus, nazývaný také Bellman-Fordův algoritmus, patří Routing Information Protocol (RIP). RIP používá metriku počet skoků mezi směrovači a vybírá tedy vždy cestu, která je k cíli nejkratší a nebere v potaz žádný jiný parametr.
- **algoritmy stavu linky (link-state)** – mezi směrovací protokoly využívající tento algoritmus, nazývaný také Shortest Path First (SPF), patří OSPF a IS-IS. Protokol OSPF bude popsán podrobně, jelikož tento směrovací protokol bude používán v simulacích v praktické části této práce.
- **hybridní algoritmus** - kombinuje výhody obou základních předchozích algoritmů vektoru vzdálenosti a stavu linky, používá difuzní algoritmus zvaný Diffusing Update algorithm (DUAL) a využívá ho protokol EIGRP od společnosti Cisco [3].

2.3.1 Open Shortest Path First (OSPF)

OSPF pracuje na algoritmu využívající stavy spojů (Shortest Path First, SPF), nazývaný také Dijsktrův algoritmus. Tento algoritmus umožňuje protokolu OSPF rychlou konvergenci při změně topologie, např. přerušení fyzického média nebo výpadku části sítě. Protokol OSPF sleduje stav linky a při každé změně v topologii se posílají informace o změně sousedním směrovačům okamžitě. Pokud se žádná mimořádná událost v síti nevyskytuje, posílají se směrovací informace každých 30 minut z důvodu kontroly stavu linek [3].

Při nakonfigurování směrovacího protokolu OSPF na směrovačích v síti nejdříve směrovače vysílají svými rozhraními tzv. Hello pakety pro nalezení sousedních směrovačů. Tento proces slouží k navázání sousedského vztahu. V Hello paketech se nachází určité parametry a nastavení OSPF, které se musí shodovat mezi sousedními

směrovači. Až poté, co směrovače naváží sousedský vztah, mohou mezi sebou vyměňovat směrovací informace a začít plnit své směrovací tabulky. Zprávy obsahující tyto informace se označuje jako LSA (Link State Advertisement). LSA popisuje stav rozhraní směrovače nebo seznam směrovačů připojených k dané síti. Směrovače si informace z LSA ukládají do své topologické databáze (Link-state database, LSDB) a dál se posílají sousedním směrovačům všemi směry. Protokol OSPF tedy pomocí LSA vytváří a udržuje databáze topologických informací o dané síti. Pokud nastane změna topologie v určité části sítě, směrovač, na kterém ke změně došlo, vyšle zprávu LSA svým sousedům a ti ji rozšíří dále postupně do celé sítě [3][4].

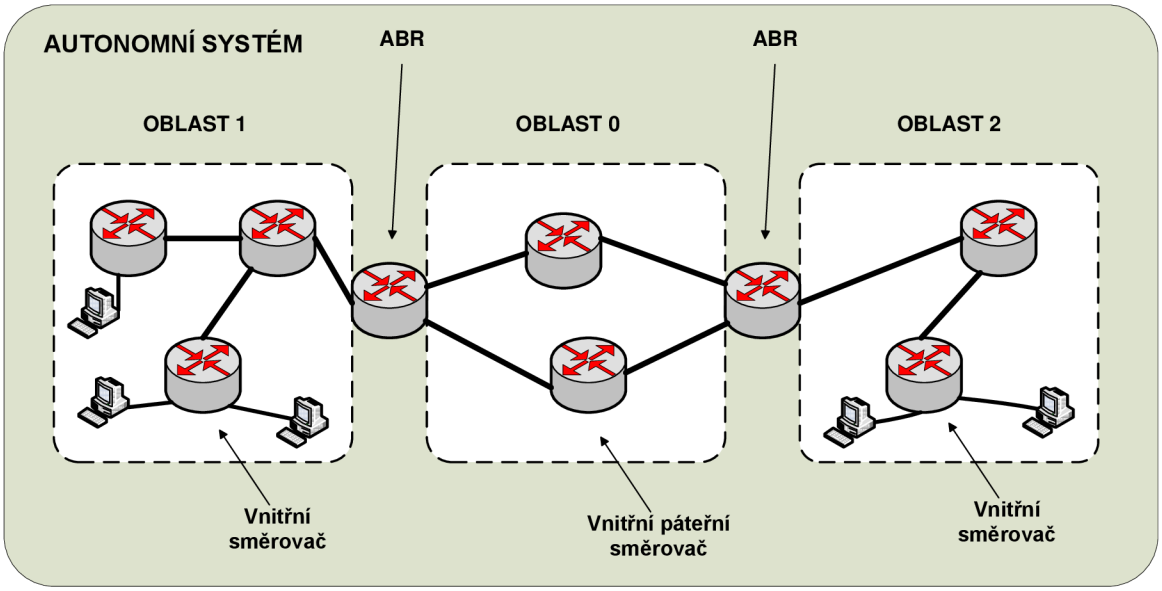
32 bitů			
8 bitů	8 bitů	8 bitů	8 bitů
Verze	**Typ**	**Délka paketu**	
Router ID			
Area ID			
Zabezpečení		**Typ autentizace**	
Autentizace			
Autentizace			

Obr. 2.3: *Formát záhlaví OSPF paketu [5]*

Protokol OSPF pro svou činnost využívá pět typů paketů, všechny tyto typy OSPF zprávy mají společné záhlaví (viz Obr. 2.3). Mezi OSPF zprávy patří [5]:

- **pakety HELLO** – zjišťování nových sousedních směrovačů, výměna parametrů OSPF,
- **pakety Database Description** – využívány pro budování databáze o topologii sítě,
- **pakety Link State Request** – pokud sousední směrovače po porovnání svých databází o topologie zjistí, že jim chybí nějaký záznam nebo jsou informace zastaralé, vyžádají si příslušné informace zasláním paketu Link State Request,
- **pakety Link State Update** – tímto paketem směrovač posílá příslušné informace, které žádá sousední směrovač,
- **pakety Link State Acknowledge** – potvrzení přijetí vyžádaných informací.

Protokol OSPF může být použit v malých i rozsáhlých sítích. Aby bylo dosaženo co nejefektivnějšího směrování jak v menších tak velmi rozsáhlých sítí, byly zavedeny v rámci sítě tzv. oblasti. Určité části sítě lze do těchto oblastí sdružovat. Příklad sítí rozdělení oblastí je na obr. 2.4 [4].



Obr. 2.4: *Hierarchie v OSPF*

V OSPF je použita hierarchická architektura (viz obr. 2.4), jejíž základ tvoří oblast 0, nazývaná páteřní síť. Na okraji sítě jsou hraniční směrovače ABR (Area Border Router), které k páteřní síti připojují další oblasti, které jsou označeny jednoznačnými číselnými identifikátory, tedy např. oblast 1. Všechny oblasti pod jednotnou správou se nazývají autonomní systém [4][3].

Topologie jednotlivých oblastí zůstává skryta pro ostatní oblasti a každá oblast tedy reaguje pouze na změny ve svojí vlastní topologii a nestará se o topologie ostatních oblastí. Tím se snižuje objem provozu nutného k práci OSPF, urychlí se proces konvergence a případné chyby se řeší právě v dané oblasti.

Výpočet nejvýhodnější cesty do všech dostupných sítí se děje pomocí algoritmu SPF nebo též Dijkstrův algoritmus. Jediný parametr výpočtu je bezrozměrná metrika označovaná jako cena. Cena je číslo přiřazené na každé rozhraní směrovače, které podporuje protokol OSPF. V drtivé většině případů se počítá automaticky z kapacity připojené linky na rozhraní. Může být však nastaven také ručně např. podle potřeb správce sítě. Hodnota ceny je v rozsahu 1 až 65535 [3]. Jako výslednou nejlepší cestu ze směrovače do cílové stanice algoritmus SPF vyhodnotí tu s nejnižší celkovou

cenou, která je dána součtem všech cen odchozích rozhraní, přes které bude procházet síťový provoz. Každá oblast používá individuální algoritmus SPF (Shortest Path First) a po výměně informací jsou tedy topologické databáze všech směrovačů v jedné oblasti identické [3][4].

3 User Datagram Protocol (UDP)

Protokol UDP je protokolem transportní vrstvy. Poskytuje nespolehlivé a nespojované přenosové služby. Pokud je tedy UDP datagram během přenosu zahozen nebo ztracen, přenos tohoto datagramu se neopakuje. Tímto se tedy liší od protokolu TCP (Transmission Control Protocol), který je spojovaný a spolehlivý [1]. Svými jednoduchými vlastnostmi se protokol UDP hodí pro aplikace vyžadující rychlý přenos bez přílišného zabezpečení přenosu, jako navazování spojení, kontrolu pořadí příchozích dat a potvrzování příchozích dat, popř. opakování přenosu ztracených dat. Mimo jiné je využíván aplikačními protokoly SNMP, DNS nebo DHCP [17]. Protokol UDP bude právě díky své jednoduchosti využíván také v praktické části diplomové práce při simulování metod používaných v programu ns-3.

3.1 Formát datagramu UDP

Protokol UDP využívá čísla portů k identifikaci aplikačních protokolů, tedy aby bylo možné rozlišit v počítači jednotlivé aplikace pro správné doručení dat. Jedná se o 16-bitovou hodnotu, tedy rozsah 0-65535, přičemž porty 1-1023 jsou přidělena specificky jednotlivým protokolům vyšších vrstev, jedná se o tzv. well-known porty [17]. Čísla portů vyšší jak 1024 již mohou být dynamicky přidělovány na základě momentální potřeby a dynamicky také zanikají po ukončení přenosu. Tyto porty tedy slouží pro dočasnou komunikaci. Na obr. 3.1 lze vidět formát UDP datagramu [17].

32 bitů	
16 bitů	16 bitů
Zdrojový port	**Cílový port**
Délka	**Kontrolní součet**
Data	

Obr. 3.1: *Formát UDP datagramu*

Záhlaví protokolu UDP se skládá z 5 polí:

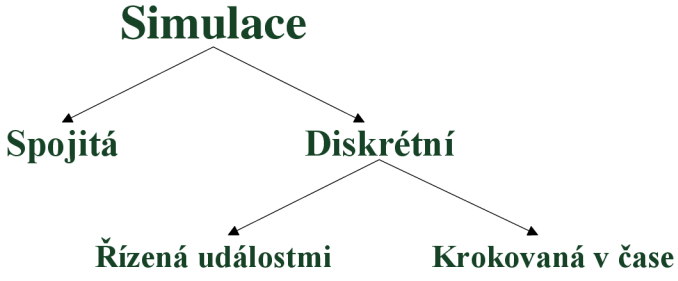
- **zdrojový port** – identifikuje zdrojový aplikační proces,
- **cílový port** – identifikuje cílový aplikační proces, tedy kterému procesu na cílovém uzlu jsou data určena,
- **délka** – délka celého datagramu (násobky 32 bitů),
- **kontrolní součet** – zabezpečení datagramu, není povinné,
- **data** – samotná přenášená data [17].

4 Simulace

Simulace je nahrazení operací reálných procesů či systémů v čase simulačním modelem, s cílem získat informace o zkoumaném systému [12]. Počítačová simulace je výpočetní proces, který umožňuje modelovat rozsáhlý systém a analyzovat chování systému. Nesporné výhody simulace jsou v možnosti neomezeného opakování simulace, kontroly nad simulací a také je levnější variantou zkoumání reálného systému. Čas simulace neboli virtuální čas, je v některých případech kratší než reálný čas systému [13]. Znalosti získané ze simulace lze potom využít k analýze chování systému, důkladnému prostudování a pochopení fungování systému, následné implementaci nových prvků či konfiguraci stávajících prvků a tím vylepšení celého systému. Simulované systémy lze rozdělit podle jejich charakteru na:

- **spojitý systém** – hodnoty proměnných se v čase mění spojitě, simulace pro tento typ systému se nazývá spojitá simulace,
- **diskrétní systém** – hodnoty proměnných se mění pouze v určitých (diskrétních) bodech na časové ose, simulace pro tento typ systému se nazývá diskrétní simulace.

Rozdělení popsaných druhů simulace lze vidět na obr. 4.1.



Obr. 4.1: *Rozdělení druhů simulace*

4.1 Diskrétní simulace

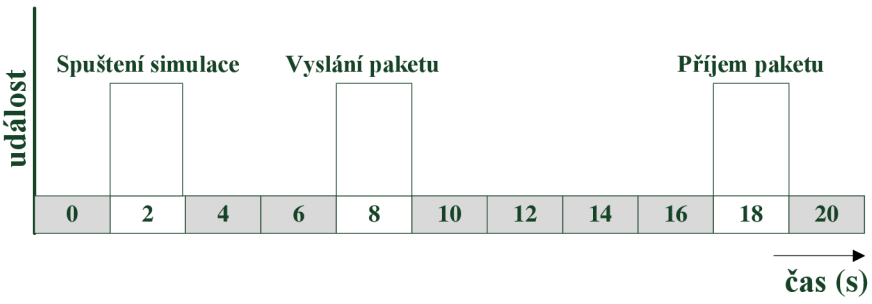
Diskrétní simulace vnímá čas jako osu s body, ve kterých nastala změna stavu sledovaného systému. V případě diskrétní simulace předpokládáme, že se stav systému během konečného časového intervalu mění jen v konečně mnoha okamžicích. To znamená, že na spojité časové ose existuje pouze konečně mnoho časových okamžiku, v nichž dojde ke změně stavu proměnné [15]. Diskrétní simulace je charakteristická tím, že na časové ose přeskakuje z jednoho bodu změny stavu na další. Proměnné se tedy mění skokově [14].

Existuje několik druhů diskrétních simulací, liší se mezi sebou ve způsobu sběru proměnných během simulace, resp. jak program definuje sběr těchto dat v závislosti na čase. Nejběžnějšími typy diskrétní simulace jsou:

- Krokovaná v čase – body na časové ose simulace mají mezi sebou rozestup velikosti stejně velkých časových kroků, simulace pak tedy sbírá data v přesně určených bodech [14],
- Řízena událostmi – tento typ diskrétní simulace bude popsán v další kapitole, jelikož je využíván simulátorem ns-3.

4.1.1 Diskrétní simulace řízená událostmi

Diskrétní simulace řízená událostmi probíhá v čase tak, že se proměnné v modelu mění pouze tehdy, nastane-li nějaká událost. Mezi dvěma událostmi jsou různé časové intervaly, co se děje během nich není pro simulaci podstatné [11]. Pod pojmem událost si lze u simulace sítě představit například vysílání či přijímání paketu, vypršení časovače, či k nějaké nečekané události, jako například výpadek linky nebo připojení stanice do sítě [7]. Událost je změna, která je okamžitá, má nulovou dobu trvání. Ukázka simulace řízená událostmi je zobrazena na obr. 4.2.



Obr. 4.2: *Ukázka simulace řízené událostmi*

Diskrétní simulace se skládá z několika základních složek, které jsou pro ni typické [11]:

- **Čas** – čas se mění skokově,
- **Události** – změny v daném systému, které se simulují, jednotlivé události jsou zpravidla prováděny postupně v čase,
- **Procesy** – posloupnost několika událostí v závislosti na simulačním čase, které ovlivňují běh simulace,
- **Generátor náhodných čísel** – pseudonáhodná čísla se využívají k napodobení reálných podmínek,
- **Statistiky** – statistická data jsou nejčastěji výstupem simulace, aby bylo možné simulaci analyzovat a zpracovat události, které se při simulaci sledují,
- **Koncové podmínky** – nastavení podmínek, po jejichž naplnění se simulace ukončí [11].

5 Simulační nástroj ns-3

Ns-3 je volně šiřitelný síťový simulátor založený na diskrétní simulaci řízenou událostmi (angl. discrete-event), sloužící pro simulování provozu v síti. Je určen primárně pro výzkumné a vzdělávací projekty [6]. Tento simulátor bude použit v této diplomové práci k simulování datového provozu a následné analýze. Ns-3 byl z mnoha alternativních simulátorů vybrán především z důvodu vysokého stupně přesnosti při simulacích a také možnosti využití přímého vykonávání kódu (Direct Execution Code, DCE) [6]. Mnoho lidí, kteří již přišli do styku se simulačním nástrojem ns-2, by se mohlo domnívat, že ns-3 je pouhé vylepšení staršího simulátoru ns-2. Avšak jak bude vysvětleno dále, tato domněnka je mylná a síťový simulátor ns-3 je odlišný a nově vyvinutý software.

5.2 Vývoj ns-3

Vývoj simulátoru začal v roce 2005 s cílem vytvořit modernější alternativu již zastaralého simulátoru ns-2. Autoři se po uvážení rozhodli vyvinout zbrusu nový simulátor, ne pouze rozšířit simulátor ns-2. Tímto krokem sice znemožnili zpětnou kompatibilitu se simulátorem ns-2, ale vytvořili zcela nový a přesnější simulační nástroj [7]. Oba simulátory jsou napsány ve stejném programovacím jazyce C++,

avšak rozdílnost simulátorů spočívá v použití jazyků pro vytváření simulací. Starší ns-2 používá skriptovací jazyk Tcl (Tool Command Language), zatímco nový ns-3 používá jazyk C++. Porovnání simulačních nástrojů ns-3 a ns-2 lze vidět v Tab. 5.1 [7].

Tab. 5.1: *Srovnání simulátorů ns-3 a ns-2*

	VÝHODY	NEVÝHODY
ns-3	• Vysoká přesnost simulací • Podpora přímého vykonávání kódu • Vylepšená podpora prvků s více rozhraními • Vylepšené možnosti sledovaných parametrů a jejich vykreslování • Jednodušší psaní simulací v jazyku C++ • Volně šiřitelný software	• Podpora menšího množství síťových protokolů • Nekompabilita s projekty z ns-2
ns-2	• Podpora velkého množství síťových protokolů • Volně šiřitelný • Velké množství existujících projektů	• Vytváření simulací je komplexní a časově náročné • Výsledky simulací jsou v některých případech nekonzistentní • Náročné nalézání chyb v projektu • Jádro programu neudržováno

5.3 Diskrétní simulace v ns-3

V simulátoru ns-3 nám diskrétní simulace umožní využívat operace, které nám usnadní analýzu a vyhodnocování simulací. Mezi tyto operace například patří:

- **Spuštění** – zaznamenává jednotlivé události s přibývajícím časem až do konce simulace nebo do pozastavení simulace,
- **Pozastavení** – umožní pozastavení simulace, v simulaci se již nachází předchozí zaznamenané události, které lze zkoumat,
- **Zjištění aktuálního času simulace**
- **Plánování** – umožňuje vytvoření určité události v určitém času simulace [7].

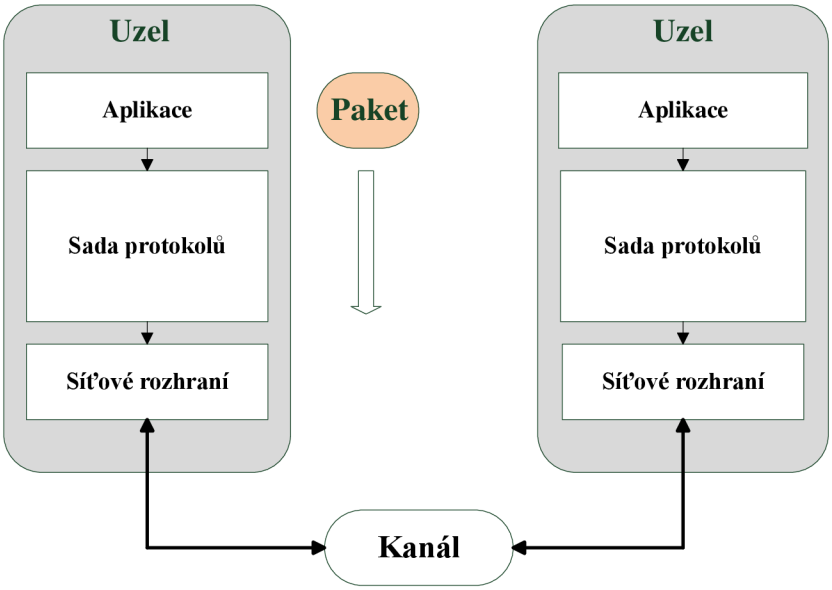
5.4 Koncepce ns-3

V této kapitole budou vysvětleny pojmy používající se při vytváření projektu v simulátoru ns-3 a také struktura programu a začlenění jednotlivých procesů do základního modelu simulace.

5.4.1 Základní model koncepce

Na obr. 5.1 je zobrazen základní model koncepce simulátoru ns-3. Klíčovými objekty simulátoru jsou:

- Uzel (Node),
- Kanál (Channel),
- Paket (Packet) [6][7].



Obr. 5.1: *Základní model koncepce ns-3*

5.4.2 Uzel

V simulátoru ns-3 výraz uzel reprezentuje síťové výpočetní zařízení. Uzel poskytuje metody pro správu výpočtů v simulacích. O uzlu se uvažuje jako o koncovém počítači, kterému je možné přiřadit tyto funkce:

- Aplikace,
- Sadu protokolů (Protocol-stack),
- Síťové rozhraní (NetDevice).

V jazyku C++ je tato abstrakce reprezentovaná třídou ***Node*** [7].

5.4.3 Aplikace

V simulátoru ns-3 se pojmem aplikace rozumí generátor a příjemce paketů, který běží na uzlu a komunikuje s právě používanou sadou protokolů. Každý uzel může využívat služeb jedné nebo více aplikací. Aplikace vytvářejí sockety API (Application

Programming Interface), které reprezentují koncové body komunikace a používají se pro vysílání a přijímání provozu mezi aplikací a sadou protokolů na tomtéž uzlu.

Mezi aplikace patří například:

- Generování provozu,
- Vypínání a zapínání simulace,
- Směrovací agenti.

V jazyku C++ je tato abstrakce reprezentovaná třídou ***Application*** [7].

5.4.3 Sada protokolů

Pro správné fungování sítě je potřeba uzlu přiřadit síťové protokoly, pomocí kterých bude komunikace probíhat. Mezi tyto protokoly patří například IPv4, IPv6, ICMP, TCP nebo UDP.

5.4.4 NetDevice

V reálně síti je nutné mít v koncovém zařízení kartu, přes kterou je možné se do sítě připojit a která je schopna zpracovávat procházející provoz. Tato hardwarová karta se nazývá NIC (Network Interface Card), avšak ke správnému fungování je potřeba také softwarová podpora v podobě ovladačů pro danou kartu.

V simulátoru ns-3 se tato hardwarová karta i softwarové funkce vyjadřují pomocí jedné abstrakce síťového rozhraní. Jak již bylo uvedeno, síťové rozhraní se nachází v uzlu a umožňuje komunikaci s jinými uzly v simulaci přes kanál. Na jednom uzlu se může vyskytovat více síťových rozhraní a tím komunikovat s více uzly.

V jazyku C++ je tato abstrakce reprezentovaná třídou ***NetDevice*** [7].

5.4.5 Kanál

Jedná se komunikační kanál spojující síťová rozhraní a umožňuje uzlům mezi sebou komunikovat. Jako příklad komunikačního kanálu může být uveden ethernetový kanál (CsmaChannel), point-to-point kanál (PointToPointChannel), wifi kanál (WifiChannel).

V jazyku C++ je tato abstrakce reprezentovaná třídou ***Channel*** [7].

5.4.6 Topology Helpers

Tímto procesem nazýváme výrazné zjednodušení, které přináší simulátor ns-3. V simulačním modelu tyto funkce usnadňují základní operace vedoucí k úspěšnému fungování sítě. Například, když v reálné síti má každý síťový prvek svou MAC adresu a IP adresu a samotný přenos přes fyzické rozhraní je také přesně definován. Při vytváření rozsáhlé sítě v ns-3 by tedy přiřazování MAC a IP adres každému síťovému prvku zabralo velké množství času. Topology helper starající se o přiřazování IP adres tento proces udělá za uživatele. Ns-3 nabízí velkou sadu funkcí topology helper, které se starají o to, aby tyto běžné úkony byly co nejrychlejší a nejsnazší [6]. Veškeré topology helpery dostupné v simulátoru ns-3 lze nalézt v dokumentaci tříd na oficiálních stránkách [18].

5.4.7 Paket

Paket v simulátoru ns-3 je generován při vysílání aplikací, která do paketu uloží potřebné informace. Paket dále putuje vertikálně přes jednotlivé vrstvy uzlu, až je následně vyslán síťovým rozhraním přes kanál na další uzel, kde je tento paket dále zpracován a postupuje od síťového rozhraní až k aplikaci. Každý paket vysílaný v simulaci obsahuje velké množství informací. Tyto informace jsou v paketu rozděleny do tří skupin:

- **Buffer** – obsahuje hlavičku paketu (header), data (payload) a zakončení paketu (trailer),
- **Metadata** – obsahují informace o typu hlavičky a zakončení uložené v bufferu,
- **Tagy** – jedná se o sadu dat poskytnutých uživatelem, většinou jde o doplňující informace jako například číslování jednotlivých paketů nebo identifikaci jednotlivých toků [6].

5.5 Operace s pakety v ns-3

Při vývoji simulátoru ns-3 se bral velký ohled na operace s pakety při simulaci, protože na tomto procesu velice úzce závisí výkonnost a efektivnost samotné simulace. Ns-3 oproti ostatním dostupným simulátorům přináší řadu vylepšení z pohledu práce s paketem. Mezi vylepšení patří například:

- Podpora fragmentování a znovu sestavování paketu na cílové stanici,

- Ukládání některých simulačních informací do posílaných paketů a tím zefektivnění simulování,
- Přehlednější zobrazování informací o simulaci, čímž se výrazně zjednoduší analýza chyb v simulaci,
- Efektivní využití paměti,
- Efektivní rozložení CPU zátěže pro lepší průběh simulace [7].

5.5.1 Hlavička paketu

Při putování paketu sítí a při jeho zpracovávání v uzlech se neustále mění hlavička protokolů při vertikálním průchodu vrstvovým modelem. V jednotlivých hlavičkách jsou uloženy důležité informace pro správné fungování doručení paketu od zdrojové aplikace na počáteční stanici až po cílovou aplikaci na koncové stanici. V rámci této diplomové práce, jak již bylo řečeno, bude použit protokol IPv4, takže hlavička 3. vrstvy bude odpovídat hlavičce IPv4 protokolu na obr. 2.1. Na 4. Vrstvě bude použit protokol UDP (User Datagram Protocol) nebo TCP (Transmission Control Protocol). V simulátoru ns-3 existují tři funkce pro práci s hlavičkami v paketu [7]:

- **AddHeader** – funkce pro přidání nové hlavičky protokolu paketu, na obr. 5.2 zobrazena ukázka přidání hlavičky 4. vrstvy (Layer 4, L4), tedy například UDP protokolu,

Obr. 5.2: *Ukázka funkce AddHeader*

- **RemoveHeader** – funkce pro odstranění hlavičky, na obr. 5.3 je zobrazena ukázka odebrání hlavičky 4. vrstvy,

Obr. 5.3: *Ukázka funkce RemoveHeader*

- **PeekHeader** – funkce, která zachová stávající hlavičku a uchová ji v paměti, aby mohla být použita pro další analýzu, na obr. 5.4 je zobrazena ukázka zachování hlavičky 4. vrstvy.

Obr. 5.4: *Ukázka funkce PeekHeader*

6 Přímé vykonávání kódu

Tato kapitola je již z hlediska zadání diplomové práce velmi důležitá, jelikož zde bude vysvětlena podstata podpory přímého vykonávání kódu (Direct Code Execution, DCE) v simulátoru ns-3. Právě tato podpora DCE ve spolupráci s ns-3 nabízí o mnoho realističtější simulování sítí s mnohem přesnějšími výsledky oproti starším simulátorům. Tyto simulátory nebyly schopny splnit všechna kritéria implementace DCE, jak je popsáno dále v této kapitole.

Přímé vykonávání kódu je v podstatě modul, který uvnitř ns-3 umožňuje využívat existující protokolové implementace systému (v tomto případě Linux) bez nutnosti zasahovat do zdrojového kódu. Nezasahovat do zdrojového kódu byla přímo jedna z podmínek při vytváření modulu DCE. Používáním nezměněné systémové implementace lze dosáhnout i při komplexní simulaci stále stejné výsledky, což u starších simulátorů při komplexních simulacích nebylo zaručeno a výsledky byly často nekonzistentní. Výhoda konzistentních výsledků u DCE se využívá při jemném nastavování síťových parametrů, při hledání chyby v síti a také při srovnávání jednotlivých síťových implementací mezi sebou [7].

V ns-3 lze využít několika operačních módů pro simulace. Z teoretického hlediska jsou čtyři operační módy kombinující použitou sadu síťových implementací a použitou metodu vykonávání aplikací. Z pohledu síťových implementací záleží, zda se využívá linuxová sada síťových protokolů nebo je využita sada implementovaná v simulátoru ns-3. Z pohledu metody vykonávání aplikací je možné použít buď metodu DCE nebo opět využít aplikace implementované v ns-3. Mimo mód, který využívá samotné ns-3 bez přídavných implementací, tedy sadu protokolů i aplikací implementovaných v simulátoru ns-3, se používají dva hlavní operační módy, jak je vysvětleno dále.

6.1 Operační módy DCE v simulátoru ns-3

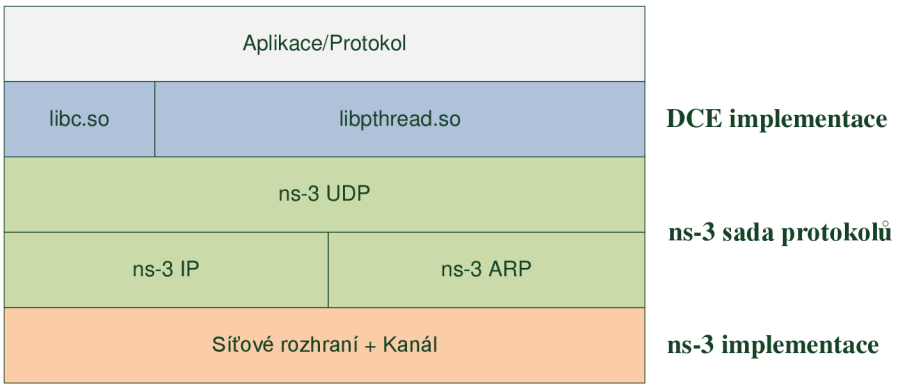
Při využívání přímého vykonávání kódu v ns-3 má uživatel na výběr ze dvou hlavních operačních módů. Výběr operačních módů je umožněn především z důvodu

kompromisu mezi realističností a časovou úsporou u simulace. DCE v ns-3 nabízí dva hlavní operační módy [8]:

- Základní – používá se sada protokolů TCP/IP, která je implementována v ns-3,
- Pokročilý – používá sadu linuxových síťových protokolů.

6.1.1 Základní operační mód

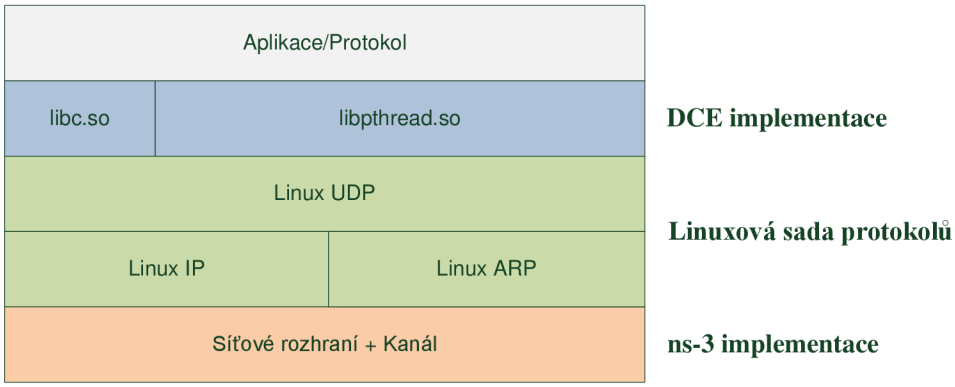
Tento mód využívá sadu protokolů implementovanou přímo v ns-3. Základní mód je rychlejší, není tak výpočetně náročný, ale také není tak realistický jako pokročilý mód. Vrstvový model základního operačního módu lze vidět na obr. 6.1. DCE implementace se nachází mezi aplikací a vrstvou UDP a jeho úkol je zachytávat komunikaci od protokolu (aplikace) a upravovat ji tak, aby mohla být dále zpracována ns-3 implementací [8].



Obr. 6.1: *Vrstvový model DCE pro základní operační mód v ns-3*

6.1.2 Pokročilý operační mód

Tento mód využívá implementace realistických linuxových protokolů implementovaných přímo v jádře systému. Díky tomu je také samotná simulace přesnější a realističtější než základní mód, ale je výpočetně náročnější. Vrstvový model pokročilého operačního módu lze vidět na obr. 6.2 [8].



Obr. 6.2: *Vrstvový model DCE pro pokročilý operační mód v ns-3*

6.2 Využití DCE v síťových simulátorech

Integrování přímého vykonávání systémových implementací do síťových simulátorů pro zvýšení přesnosti simulací není žádná nová myšlenka. Mnoho jiných simulátorů má tuto možnost využití, přesto žádný jiný simulátor nedokázal naplnit všechny kritéria DCE integrovaného prostředí nezbytných pro přesné, stabilní a konzistentní simulace [7].

Mezi tyto kritéria patří:

1. **žádné manuální zasahování do zdrojových kódů implementací**,
2. **integrovaný debugging** – pro snadnou analýzu kódu a nalezení chyby,
3. **efektivita** – zátěž procesoru a paměti by měl být co nejmenší i při vykonávání náročných rozsáhlých simulací,
4. **možnost použít systémové implementace nebo implementace simulátoru** – zvolení základního nebo pokročilého módu, u základního módu jsou samozřejmě protokolové implementace daného simulátoru,
5. **podpora C a C++** – protokolové implementace jsou napsané v jazyce C a C++, proto je podpora těchto jazyků v simulátoru nezbytná [7].

Zajištění všech kritérií byl tedy opravdu náročný úkol a byl to jeden ze stěžejních bodů při vytváření simulátoru ns-3. Ns-3 měl tedy výhodu v tom, že se již od počátku s těmito kritérii počítalo při vývoji programu. To však nic nemění na tom, že autoři čelili velké výzvě a nakonec dokázali vytvořit simulátor, který všechny tyto kritéria naplňuje. Ostatní simulátory nedokážou nabídnout podobné vlastnosti implementace DCE a proto byl pro diplomovou práci vybrán právě simulační nástroj ns-3.

Pro úplnost jsou níže uvedeny příklady jiných simulátorů, které nabízí možnost použití DCE, avšak nenaplňují jedno či více kritérií [7]:

- **ns-2** – nesplňuje kritéria 1 a 4,
- **GTNeS -** nesplňuje kritéria 1 a 4,
- **Nfsim -** nesplňuje kritérium 3,
- **COOJA -** nesplňuje kritérium 4,
- **NCTUns -** nesplňuje kritéria 1 a 2.

6.3 Implementace DCE v ns-3 simulátoru

Již bylo řečeno, že cílem DCE je zvýšení přesnosti simulací spouštěním dané linuxové síťové implementace přímo v simulátoru ns-3. Aby toto provedení bylo možné, DCE vykonává několik úkolů:

1. nahrává kódy a data linuxových implementací do paměti,
2. zastává funkci zprostředkovatele mezi simulátorem a systémovými funkcemi,
3. řídí a kontroluje vykonávání procesů, paměť a ukončování procesů,
4. řídí plánování různých nastavených procesů.

V samotném simulačním programu ns-3 dochází k propojení s DCE pomocí tříd **DceManagerHelper** and **DceApplicationHelper** [8].

První jmenovaná, **DceManagerHelper,** je jakýsi vstupní bod DCE. Všechny uzly, které chtějí využívat metodu DCE, musí být součástí této třídy. Třída obsahuje veškeré funkce nutné pro správné proběhnutí simulace.

Druhá třída **DceApplicationHelper** představuje vstupní bod pro programy spouštějící se přes DCE a tato třída obsahuje funkce pro správný běh těchto aplikací.

6.4 Podporované balíky v ns-3 DCE

Pro přímé vykonávání kódu byla umožněna spolupráce s dalšími softwarovými balíky, které přináší další možnosti využití simulátoru ns-3. Mezi nejpoužívanější podporované balíky patří:

- **Quagga** – softwarová sada pro zajištění směrování pro unixové platformy,

- **CCNx** – implementace pro architekturu CCN (content-centric networking),
- **Linux Kernel implementace** – použití reálné protokolové sady Linuxu.

Pro další vývoj této diplomové práce bude popsán balík Quagga, který bude v pozdějších simulacích využíván.

6.5 Quagga

Softwarový balík Quagga slouží pro zajištění směrování pro operační systémy Linux, Solaris, NetBSD. Podporuje směrovací protokoly RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, BGP a BGP+ a samozřejmě tedy síťové protokoly IPv4 a IPv6 [9]. Znovu připomenu, že v diplomové práci bude použit internetový protokol IPv4 ve spolupráci se směrovacím protokolem OSPFv2. Quagga je volně distribuována pod otevřenou licencí GPL (General Public License) v podobě zdrojových kódů a binárních balíčků. Díky programu Quagga je možné v systému využívat démonu, které zajišťují funkci směrování podobně jako směrovač.

Program Quagga je založen na systému GNU Zebra, kterému skončila podpora vývoje v roce 2005. Architektura programu se skládá z několika démonů. Hlavní roli hraje démon zebra, který shromažďuje směrovací informace, spolupracuje s jádrem systému a upravuje jeho směrovací tabulky. Ostatní démoni (ripd, ripngd, ospfd, bgpd) slouží jako rozhraní centrálního démona pro konkrétní směrovací protokoly. Nevýhodou tohoto distribuovaného uspořádání je poněkud chaotická konfigurace, každý démon má vlastní konfigurační soubor a vlastní rozhraní pro interaktivní komunikaci [16].

Démon protokolu OSPFv2 se nazývá ospfd. Démon ospfd podporuje jak základní operační mód, tak pokročilý operační mód přímého vykonávání kódu [9]. Každý uzel ve vytvořené simulaci, na kterém poběží démon ospfd, se tedy bude chovat jako jednotlivý směrovač, který musí být pro správnou činnost řádně nakonfigurován. Konfigurace probíhá pomocí speciálních příkazů definovaných v balíku Quagga přímo na směrovači. Při vytváření simulace v ns-3 se však parametry uzlů (směrovačů) definují ve zdrojovém kódu projektu a k samotnému nastavení směrovače dochází až při kompilaci kódu. Propojením ns-3 a Quaggy se zabývá následující kapitola.

Podpora programu Quagga s přímým vykonáváním kódu v ns-3 započala v roce 2008 a přináší další zpřesnění výsledků a také snižuje celkový čas simulace [9].

6.5.1 Použití v ns-3 simulátoru

Balík Quagga bude použit ve spojení se simulátorem ns-3 s podporou DCE v pokročilém operačním módu. V tomto případě se DCE stará o vykonávání linuxových implementací, jako v případě bez použití Quaggy. Aby bylo možné spojit quaggu, resp. démona běžícího na jednotlivých uzlech, byla vytvořena topology helper třída s názvem **QuaggaHelper**. Tato třída umožní pomocí kódu vytvořeného v prostředí ns-3 nastavení směrovacího protokolu pro možnost správného směrování v síti. Pomocí funkce **EnableOspf** dochází ke spuštění a nastavení démona **ospfd** zajišťující směrování pomocí protokolu OSPF. Konkrétní použití bude vysvětleno a ukázáno v praktické části [9].

7 Praktická část

Nyní již bude pozornost zaměřená na praktickou část diplomové práce. Simulátor ns-3 a především jeho dodatečné balíky, DCE a Quagga, jsou stále ve vývoji a postupy pro instalaci a také vytváření simulací se postupně vylepšují, zjednodušují a především se odstraňují nalezené chyby. Za dobu psaní této práce vyšlo mnoho aktualizací a některé postupy při vytváření simulace se změnily. Správná instalace a především nastavení simulátoru a jeho součástí je tedy velmi důležitý a komplexní úkol. Praktická část tedy bude postupně popisovat úplnou nynější podobu instalace ns-3, ns-3-DCE a ns-3-DCE-Quagga. Každá tato implementace bude brána jako samostatný program, jelikož budou instalovány samostatně a pokaždé od počátku. Instalace těchto programů se od sebe liší, je to dáno tím, že se o jednotlivé implementace starají rozdílné vývojové týmy. Aktualizace, odstraňování chyb a vylepšování různých procesů tedy probíhá vždy nezávisle na dalších implementacích a platí pouze pro daný program.

V dalších kapitolách praktické části budou popsány jednotlivé kroky při vytváření kompletních modelů až po simulaci a zobrazení výsledků. Budou podrobně popsány odlišnosti při vytváření modelů s podporou DCE a Quagga.

7.1 Instalace ns-3 a přídavných balíků

Jak již bylo řečeno, simulátor ns-3 je sám o sobě velmi komplexní systém a má dopady na mnoho dalších dílčích komponent. Ke svému fungování vyžaduje mnoho knihoven v systému a také další programy, bez kterých by simulátor nepracoval. Je tedy vhodné uvést ukázku instalace v používaném systému a také seznam přídavných nástrojů nutných k bezchybnému chodu programu ns-3.

Používaným systémem je linuxová distribuce Ubuntu 12.04, který je byl volen také s ohledem na jeho úplnou kompatibilitu dle oficiálních stránek [10]. Před samotnou instalací ns-3 tedy nainstalujeme následující požadované nástroje [6]:

- **Mercurial** – nástroj pro spravování a organizaci zdrojových kódů a dokumentaci ns-3,
- **Waf** – nástroj pro kompilaci, konfiguraci a instalování aplikací pomocí jazyka Python, ns-3 pro kompilaci používá jazyk Python a nástroj Waf z důvodů jeho jednoduchosti,

- **Vývojové prostředí** – psaní skriptů modelů v ns-3 je prováděno především pomocí jazyka C++, je nutné tedy stáhnout kompilátor kódů C++ [6].

Získání těchto nástrojů zajistíme v terminálu příkazem:

```
apt-get install g++ python mercurial
```

Poté co jsou všechny potřebné nástroje nainstalovány, již lze přejít k samotné instalaci simulátoru ns-3. Instalace se provede příkazem `hg` programu Mercurial, který stáhne skripty v jazyce Python a umístí do složky `ns-3-allinone`. Příkaz vypadá takto:

```
hg clone http://code.nsnam.org/ns-3-allinone
```

Nyní se použije soubor se skriptem ze složky `ns-3-allinone`, který již stáhne aktuální verzi simulátoru ns-3 z odkazu, který byl uveden v kódu a celý program nainstaluje. Provedení skriptu opět zajišťuje Mercurial. Tento proces provedeme příkazem:

```
./download.py
```

Po úspěšném proběhnutí tohoto skriptu je simulátor ns-3 nainstalován, jako poslední krok však zbývá sestavit konfiguraci pro ns-3 projekty, aby bylo možné využívat kompilaci přes nástroj Waf. Příkaz pro vykonání tohoto kroku je:

```
./build.py
```

Tímto procesem se vytvoří adresář `ns-3-dev`, který bude již využíván pro samotné vytváření projektů, a také bude z tohoto adresáře volán příkaz `./waf` pro kompilaci vytvořeného projektu [10]. Nyní je simulator ns-3 připraven k použití.

7.2 Instalace ns-3 s podporou DCE

Podobně jako samotný ns-3 se musí nainstalovat ns-3-DCE, který se bude vytvářet a instalovat jako samostatný program a nebude rozšiřovat původní instalace ns-3. Toto je především z toho důvodu, že se již nyní bude využívat jádra linuxu a instalace a sestavování programu je odlišné oproti původnímu ns-3. Je také důležité původní instalaci zanechat pro možnost simulovat bez použití jádra linuxu. Metoda přímého vykonávání kódu v pokročilém módu (použití linuxového jádra) využívá velké množství dalších programů a balíků. Při počátku psaní diplomové práce bylo nutné před instalací stáhnout několik potřebných balíků před samotnou instalací ns-3

DCE. Avšak díky aktualizaci instalačního procesu, by se nyní všechny potřebné balíky měly pomocí Mercurialu stáhnout po zadání jediného příkazu [8]:

```
hg clone http://code.nsnam.org/furbani/ns-3-dce
```

Nyní jsou tedy připraveny všechny potřebné věci pohromadě a program může být nainstalován. Před zadáním příkazu k instalaci je nutné se rozhodnout, zda chceme používat DCE v základním nebo pokročilém módu. Základní mód je v instalaci nastaven standardně, ale pro praktickou část diplomové práce bude nastaven pokročilý mód pro využívání linuxových implementací. V tomto případě je nutné změnit v konfiguračním souboru instalace, který má v adresáři s programem cestu `ns-3-dce/utils/clone_and_compile_ns3_dce.sh`, výchozí hodnotu `USE_KERNEL=NO` na hodnotu `USE_KERNEL=YES` [8]. Nyní je program ns-3 DCE připraven k instalaci. Instalaci vyvoláme příkazem:

```
./clone_and_compile_ns3_dce.sh
```

Na konci instalačního procesu proběhne série několika testů funkčnosti ns-3 DCE. Pokud jsou tyto testy provedeny úspěšně, program je připraven k používání. Je nutné říci po mnoha instalacích během psaní diplomové práce, že především při prvních instalací se často objeví nějaká chyba, většinou v podobě chybějícího balíku, který se nestáhl nebo se stáhl chybně. V tomto případě je nutné buď celý proces opakovat od začátku po vymazání aktuálních dat ns-3 DCE nebo pokud je chyba pouze v chybějícím balíku, je možné tento balík stáhnout ručně pouze a poté znovu vyvolat instalaci. Nyní tedy budeme předpokládat, že všechny potencionální chyby byly odstraněny a instalace proběhla úspěšně. Program je tedy připraven k simulování vytvořených modelů [8].

7.3 Instalace ns-3 s podporou DCE a Quagga

Instalace tohoto programu je oproti ostatním nejvíce odlišná. S touto instalací byly dříve největší potíže a bylo obtížné program sestavit bez chyb. Je to samozřejmě opět dáno tím, že s podporou další implementace do programu ns-3 přichází nutnost využívat další množství balíků a funkcí nutných pro správnou funkčnost. Proto se instalační proces z velké části předělal a zjednodušil, také díky testování a hlášení chyb od uživatelů využívajících tento program. I přes výrazné zjednodušení instalace je stále nutné pro samotnou instalaci stáhnout pomocí příkazu `apt-get install` balíky `autoconf`, `automake`, `flex`, `git-core`, `wget`, `g++`, `libc-dbg`,

`bison`, `indent`, `pkgconfig`, `libssl-dev`, `libsysfs-dev`, `gawk` [9]. Je tedy opět důležité postupovat pozorně před samotnou instalací, jinak s velkou pravděpodobností skončí chybně. Oproti instalaci ns-3 DCE to platí pro program ns-3 DCE Quagga dvojnásob. Ale jak již bylo řečeno, veškeré chyby, které jsou nalezeny při instalaci, mohou uživatelé hlásit a autoři se intenzivně snaží odstranit veškeré neobvyklé a chybné chování pomocí aktualizací. Nyní k samotnému procesu instalace. Nejprve se stáhnou konfigurační soubory potřebné k instalaci pomocí Mercurial do složky `bake` a dalšími příkazy se definují proměnné, aby bylo možné použít příkaz pro instalaci kdekoliv v adresářové struktuře [9]:

```
hg clone http://code.nsnam.org/bake bake
export BAKE_HOME=`pwd`/bake
export PATH=$PATH:$BAKE_HOME
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
```

Je vhodné si vytvořit adresář, kam se celý program ns-3 DCE Quagga stáhne a sestaví, tedy např. adresář `dce-quagga` a poté pomocí instalačního souboru `bake.py` postupně vyvolat typ instalace, stažení programu a sestavení programu. Toto provedeme pomocí následujících příkazů:

```
mkdir ns3-dce-quagga
cd ns3-dce-quagga
bake.py configure -e dce-linux -e dce-quagga
bake.py download
bake.py build
```

V programu ns-3 DCE Quagga chceme jako v předchozích programech používat protokolové implementace linuxového systému, toto nastavení je indikováno výše pomocí `-e dce-linux`. Pro úplnost, pokud bychom chtěli využívat protokoly ns-3 simulátoru, použijeme `-e dce-ns3` [9].

Po stažení a sestavení programu je vhodné opět spustit otestování programu, zda byl sestaven správně. Ve vytvořeném adresáři `ns-3-dce-quagga` je soubor s testy nachází ve složce `source/dce`, ve které použijeme příkaz:

```
./test.py -s dce-quagga
```

Pokud tento test projede úspěšně, program je sestaven správně a připraven k používání. Celkový instalační proces je docela časově náročný, proto je vhodné opravdu postupovat pozorně, jelikož je poté často obtížné chybu dohledat a opravit a je lepší celý proces instalace opakovat od začátku.

7.4 Vytváření modelu simulace

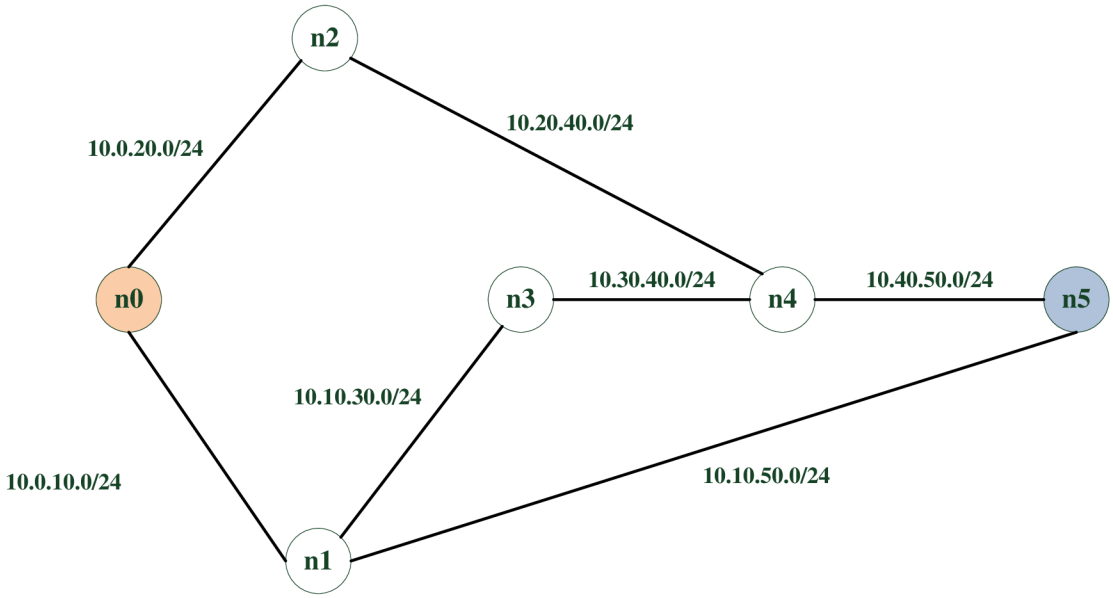
Nyní bude nastíněn postup vytváření simulačního modelu v ns-3. Simulátor ns-3 nemá žádné grafické rozhraní pro zobrazování navrhnutých modelů. Vlastní model je tedy lepší si důkladně připravit pro lepší představu při samotném vytváření kódu v simulátoru. V ns-3 jsou samozřejmě dostupné různé grafické nástavby pro vizualizaci vytvořeného modelu, jelikož není příliš pohodlné navrhovat celý model abstraktně pouze pomocí kódu. Nejpoužívanější přídavný program pro vizualizaci modelů v ns-3 je NetAnim. Tento program umožňuje zobrazit graficky umístění uzlů i samotný průběh simulace v čase i přenos jednotlivých paketů mezi uzly. Program je však určen spíše pro jednodušší modely sítí a nabízí pouze jednoduché statistické funkce. V diplomové práci tedy program není používán.

Program tedy bude bez grafického zobrazení. Pro sledování průběhu simulace a sběr výsledků budou použity dvě základní funkce, které jsou podporovány ve všech třech scénářích, tedy scénáře s programy ns-3, ns-3 DCE a ns-3 DCE Quagga. Tyto funkce se nazývají PcapHelper a AsciiTraceHelper [6][7]. Na sběr dat a zobrazení výsledků jsou opět různé nástavby, které zjednodušují vyhodnocování simulací, avšak nejsou podporovány ve všech třech programech, proto nebudou používány. Mezi nejznámější program pro sledování dat v simulaci a pro jednotlivé statistiky patří například FlowMonitor.

Funkce PcapHelper slouží pro záznam událostí simulace do výstupního souboru typu `.pcap`. Tento soubor je možný otevřít v programu Wireshark a dále analyzovat provoz v síti. Tyto výsledky budou používány jako hlavní zdroj pro další práci na grafech a statistikách. Program Wireshark je všeobecně známý a často používaný, proto se bude předpokládat určitá zkušenost a práce s programem samotným zde nebude rozebírána.

Funkce AsciiTraceHelper slouží pro záznam událostí simulace jako čistý text do výstupního souboru typu `.txt`. Tyto data budou použity pro kontrolu předchozích dat.

Nyní k samotnému návrhu sítě. Na obr. 7.1 lze vidět návrh sítě, který se bude využívat při simulování ve všech scénářích. Rozložení uzlů bylo voleno tak, aby šla názorně ukázat funkčnost směrování, výpadku při směrování a nalezení alternativní cesty k cíli. Všechny linky mezi uzly jsou typu point-to-point a každé spojení dvou uzlů se nachází v samostatné síti. Toto řešení je zvoleno také především pro demonstraci směrování a adresové schéma je řešeno přehledně pomocí čísel uzlu pro lepší orientaci.



Obr. 7.1: *Návrh simulované sítě*

Z hlediska role jednotlivých uzlů budou pro simulaci důležité pouze dva uzly, uzel 5, označený modře, a uzel 0, který je označený červeně. Ostatní uzly slouží pouze jako směrovače rozhodující o dalším průchodu paketů sítí k cíli. Uzel 5 bude plnit funkci klienta, který bude vysílat data v podobě UDP datagramů sítí směrem k serveru. Uzel 0 tedy bude plnit funkci serveru, který tyto data bude přijímat. IP adresa serveru je 10.0.10.1.

Klient vysílá data pouze na známou IP adresu serveru, a jak již bylo vysvětleno, využívaný protokol UDP nesestavuje spojení s cílovou stanicí, pakety tedy budou vyslány do sítě na základě informací nacházejících se ve směrovacích tabulkách. V těchto směrovacích tabulkách se nachází k dané cílové IP adrese serveru pouze adresa next-hop směrovače, kam budou pakety poslány. Další směrovače na cestě znovu rozhodují o možných cestách k cíli na základě jejich vlastních směrovacích tabulek. Klient na začátku vysílání tedy předem neví, jakou cestou se paket k cílovému serveru dostane. Používaný směrovací protokol v simulaci je OSPF.

Nyní bude zaměřen pohled na parametry vytvořené sítě. Již bylo řečeno, že všechny linky jsou stejného typu point-to-point, z hlediska kapacity linky je to stejné. Kapacita všech linek je uměle omezena na hodnotu 768 kb/s. Tato hodnota není zvolena náhodně, je to kompromis po mnoha simulacích. Podmínky pro vhodnou kapacitu byly takové, aby byla síť dostatečně zatížena, a aby byla ztrátovost v přijatelných hodnotách. Pokud by byla kapacita linek menší, docházelo by již k příliš velké ztrátě paketů při přenosu a výsledky ze simulace by pro tuto práci nebyly vhodné. Kapacita linky je samozřejmě volena s ohledem na nastavení aplikace na klientovi, který generuje provoz. Aby bylo možné porovnávat výsledky jednotlivých implementací, bylo nutné nastavit pevnou délku paketů, aby byly podmínky pro všechny tři implementace stejné. Pro paket byla zvolena velikost 1024 bytů, nejvyšší možná velikost paketu v aplikaci generující pakety. Toto byl jediný zásah do generující aplikace na klientu, žádné jiné nastavení pro pakety nebylo, aby se stav co nejvíce blížil reálným podmínkám. Aplikace na klientovi od spuštění své činnosti generuje maximální možný počet paketů a vysílá je do sítě. Celkový počet vygenerovaných paketů není nijak omezen a samozřejmě se bude mezi jednotlivými implementacemi lišit, tento údaj však není podstatný pro měřené výsledky. Ve vytvořených simulacích je pozornost zaměřena na zkoumání velikosti zpoždění paketů, kolísání zpoždění a ztrátovosti paketů. Tyto tři sledované parametry tedy budou ve výsledku srovnávány. Simulace bude mít dobu trvání 100 s, což je dostatečně dlouhá doba na to, aby výsledky měly vypovídající hodnotu. Za tuto dobu je vygenerováno obrovské množství paketů a příliš dlouhá doba simulace by měla za následek pouze zmnohonásobení počtu paketů pro analýzu. Na obr. 7.2 lze vidět přehledné shrnutí parametrů simulace, která bude stejně vytvořena pro všechny tři implementace zkoumané v této praktické části diplomové práce.

Simulace	
Směr provozu:	Uzel 5 --> Uzel 0
Typ linek:	Point-to-point
Kapacita linky:	768 kb/s
Transportní protokol:	UDP
Velikost paketu:	1024 B
IP adresa serveru:	10.0.10.1
Směrovací protokol:	OSPF
Doba simulace:	100 s

Obr. 7.2: *Shrnutí parametrů simulace*

7.4.1 Úvod do vytvoření projektu

Výše tedy byla simulace popsána teoreticky a nyní musíme tyto teoretické návrhy převést do skriptu pomocí jazyka C++ spustitelného v simulačním nástroji ns-3. Tento soubor se skriptem C++ bude mít tedy příponu `.cc`. Při psaní je nutné dodržovat danou strukturu programu, na začátku programu je potřeba přidat dodatečné externí moduly z knihovny ns-3 podle funkcí využívajících se v modelu. Děje se tak pomocí kódu `#include` a moduly se budou mírně mezi jednotlivými scénáři lišit. Přesněji budou oproti prvnímu scénáři ns-3 přibývat u dalších scénářů DCE a DCE Quagga. V úvodním scénáři s ns-3 budou přidány následující moduly:

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

Moduly zajišťují veškerou funkčnost a správné chování v síti při simulacích tak, jak bychom čekali v reálné síti. V kódu je využíváme v podobě topology helperů, které byly popsány již dříve. Výrazně ulehčují práci tím, že se nemusíme starat o základní fungování sítě.

Po přidání požadovaných modulů následuje definice oboru názvů `namespace`, ve kterém se bude nacházet celý kód projektu. Poté následuje definice hlavní funkce programu, kde bude napsán celý skript prvního scénáře ns-3:

```
int main (int argc, char *argv[])
```

7.4.1 Vytvoření uzlů a linek

V hlavní funkci programu budou nejprve definovány uzly, jejich vzájemné spojení pomocí linek a nastavení parametrů linek. Tento kód bude společný pro všechny tři scénáře, jelikož uzly i linky budou stále stejné. K definování veškerých prvků v síti, linek, IP adres, sady protokolů, aplikací a dalších funkcí v síti se budou využívat topology helpery. Nejprve je vytvořen kontejner pro všechny uzly s názvem `c` a poté je vytvořeno 6 uzlů:

```
NodeContainer c;
c.Create (6);
```

Nyní je tedy v kontejneru `c` vytvořeno 6 uzlů, nyní je potřeba vytvořit vazby mezi jednotlivými uzly, opět využijeme `NodeContainer` a vytvoříme pro každou vazbu samostatný kontejner:

```
NodeContainer n0n2 = NodeContainer (c.Get (0), c.Get (2));
NodeContainer n0n1 = NodeContainer (c.Get (0), c.Get (1));
NodeContainer n2n4 = NodeContainer (c.Get (2), c.Get (4));
NodeContainer n1n3 = NodeContainer (c.Get (1), c.Get (3));
NodeContainer n3n4 = NodeContainer (c.Get (3), c.Get (4));
NodeContainer n1n5 = NodeContainer (c.Get (1), c.Get (5));
NodeContainer n4n5 = NodeContainer (c.Get (4), c.Get (5));
```

Tímto je vytvořeno 7 kontejnerů pro vazby mezi jednotlivými uzly, pomocí `c.Get` určujeme konkrétní uzel. Číslování uzlů je samozřejmě stejné jako čísla uzlů v návrhu sítě z obr. 7.1. Dále nadefinujeme linku pomocí topology helperu PointToPointHelper a poté určíme parametry linky. Linky mezi uzly budou stejné, proto bude stačit tato jedna definice:

```
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("768kbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("0ms"));
```

Linka je tedy nastavena na kapacitu neboli `DataRate` 768 kb/s. Zpoždění linky neboli `Delay` je nastaveno na 0 ms, jelikož nechceme žádné uměle vytvořené zpoždění. Nakonec zbývá linky přiřadit jednotlivým vazbám `NodeContainer` vytvořeným výše. Pro přiřazení linek je potřeba vytvořit kontejnery NetDeviceContainer, což lze považovat za rozhraní na daných uzlech a pomocí `p2p.Install` určit vazbu mezi kterou bude linka přiřazena:

```
NetDeviceContainer d0d2 = p2p.Install (n0n2);
NetDeviceContainer d0d1 = p2p.Install (n0n1);
NetDeviceContainer d2d4 = p2p.Install (n2n4);
NetDeviceContainer d1d3 = p2p.Install (n1n3);
NetDeviceContainer d3d4 = p2p.Install (n3n4);
```

```
NetDeviceContainer d1d5 = p2p.Install (n1n5);
NetDeviceContainer d4d5 = p2p.Install (n4n5);
```

Nyní jsou tedy vytvořeny všechny uzly, vazby mezi nimi a také nastaveny všechny linky.

7.5 Vytvoření sady protokolů, IP adres a aplikací

V další části je potřeba uzlům přiřadit sadu protokolů, kterou budou pro komunikaci využívat, nastavit rozsah IP adres jednotlivých sítí a poté přiřadit rozhraní uzlů do dané sítě. Po tomto kroku již následuje vytvoření aplikací pro server a klienta a nastavit tyto aplikace na uzly. Kód odpovídající těmto procesům se již mezi scénáři bude lišit a každý scénář tedy bude vysvětlen a ukázán zvlášť.

7.5.1 Scénář ns-3

První scénář ns-3 využívá internetovou sadu protokolů neboli TCP/IP, které jsou implementované v simulátoru ns-3. Využije se topology helper s názvem `InternetStackHelper`, který tedy zajišťuje správnou funkčnost těchto protokolů. Poté se určí uzly, na kterých budou tyto sady protokolů nastaveny. V našem případě se samozřejmě jedná o všechny uzly v síti, pokud má síť fungovat správně. Všechny uzly jsou vytvořené v kontejneru `c`, jak bylo ukázáno výše. Kód vypadá následovně:

```
InternetStackHelper internet;
internet.Install (c);
```

K práci s IPv4 adresami se používá topology helper `Ipv4AddressHelper`. Jako dvojice rozhraní, které jsou spojené linkou, se budou uvažovat vytvořené kontejnery pomocí `NetDeviceContainer`. Pro každý kontejner se zvolí vhodný adresní rozsah, jak je ukázáno v obr. 7.1, tak, aby se jednalo o samostatné sítě. Tyto adresní rozsahy jsou přiřazeny jednotlivých dvojicím rozhraní v dalším vytvořeném kontejneru s názvem `Ipv4InterfaceContainer`. Samotné přiřazení konkrétních IP adres rozhraním probíhá automaticky při spuštění simulace díky používanému topology helperu. Odpovídající kód:

```
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.0.20.0", "255.255.255.0");
Ipv4InterfaceContainer i0i2 = ipv4.Assign (d0d2);
ipv4.SetBase ("10.0.10.0", "255.255.255.0");
```

```
Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);
ipv4.SetBase ("10.20.40.0", "255.255.255.0");
Ipv4InterfaceContainer i2i4 = ipv4.Assign (d2d4);
ipv4.SetBase ("10.10.30.0", "255.255.255.0");
Ipv4InterfaceContainer i1i3 = ipv4.Assign (d1d3);
ipv4.SetBase ("10.30.40.0", "255.255.255.0");
Ipv4InterfaceContainer i3i4 = ipv4.Assign (d3d4);
ipv4.SetBase ("10.10.50.0", "255.255.255.0");
Ipv4InterfaceContainer i1i5 = ipv4.Assign (d1d5);
ipv4.SetBase ("10.40.50.0", "255.255.255.0");
Ipv4InterfaceContainer i4i5 = ipv4.Assign (d4d5);
```

Jelikož se jednotlivá rozhraní na uzlech nachází v jiných sítích, dostáváme se do situace, kdy je zapotřebí zprovoznit funkce směrování, aby bylo možné na uzlech směrovat pakety procházející sítí. Funkčnost směrování v simulátoru ns-3 zajišťuje `Ipv4GlobalRoutingHelper`. Tento topology helper pro směrování využívá směrovací protokol OSPF a využitím tohoto helperu se OSPF protokol aktivuje na všechny uzly v síti a činnosti protokolu OSPF fungují automaticky. Kód pro tuto konfiguraci je jednoduchý a je to názorný příklad toho, jak jsou užitečné topology helpery v simulátoru ns-3:

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

Tímto krokem jsme tedy dokončili nastavování uzlů, přiřadili IP adresy a nastavili směrování. Zbývá tedy již vytvořit aplikace pro generování paketů pro klienta a přijímání paketů pro server. Budou využity dvě aplikace z jádra ns-3 s názvem `OnOffHelper` a `PacketSinkHelper`. Aplikace `OnOffHelper` slouží pro generování paketů na klientovi. Pro generování paketů je dostupných několik aplikací, `OnOffHelper` byl vybrán z toho důvodu, že je dostupný ve všech třech vytvářených scénářích. Tato aplikace umožňuje vysílat pakety v nastaveném čase při stavu `On` a přerušit vysílání v nastaveném čase ve stavu `Off`. V tomto scénáři ovšem přerušení vysílání nebude využito a bude nastaveno na 0 sekund. Velikost paketu bude nastavena na 1024 B a cílová destinace bude mít adresu rozhraní z kontejneru `Ipv4InterfaceContainer i0i1`, konkrétně to je IP adresa 10.0.10.1 a port, na kterém server čeká na příchod paketů, je nastaven na 9. Aplikace

`PacketSinkHelper` bude nastavena na serveru a jejím jediným úkolem je přijímat pakety. Odpovídající kód pro popsané kroky, nejprve pro aplikaci na klientovi:

```
OnOffHelper onoff ("ns3::UdpSocketFactory", InetSocketAddress
(i0i1.GetAddress (0), port1));

onoff.SetAttribute ("PacketSize", UintegerValue (1024));

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

ApplicationContainer apps = onoff.Install (c.Get (5));

apps.Start (Seconds (1.0));

apps.Stop (Seconds (100.0));
```

Nyní kód pro aplikaci na serveru:

```
PacketSinkHelper sink ("ns3::UdpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), port1)));

apps = sink.Install (c.Get (0));

apps.Start (Seconds (1.0));

apps.Stop (Seconds (100.0));
```

Obě aplikace se spustí v čase 1 s od začátku simulace a ukončí svou činnost v čase 100 s. Nastavení sady protokolů, IP adres, jejich přiřazení a dále vytvoření aplikací je pro první scénář hotové.

7.5.2 Scénář ns-3 DCE

Druhý scénář již bude využívat metody přímého vykonávání kódu. Z pohledu vytváření scénáře pro simulátor ns-3 se bude zdrojový kód od prvního scénáře lišit v použitém topology helperu pro sadu protokolů a ve vytvořených aplikacích. Přiřazení IP adres rozhraním a zprovoznění směrování zůstane stejné jako v prvním scénáři. Pro implementování DCE do scénáře se využívá topology helper. Avšak v prvním scénáři nebyl v úvodu přidán modul, který by umožňoval daný topology helper v programu využívat. V první řadě je tedy v druhém scénáři nutné přidat daný modul pomocí kódu:

```
#include "ns3/dce-module.h"
```

Nyní podobně jako v prvním scénáři zvolíme sadu protokolů využívající se při simulaci. Budou využity protokoly implementovány přímo v linuxovém jádře a toto

nastavení provedeme pomocí topology helperu s názvem `DceManagerHelper` a dále `LinuxStackHelper`. Tyto jsou stěžejní pro fungování DCE s pokročilém módu v programu ns-3. Pro běžného uživatele tedy stačí vyvolat tyto helpery a vše by mělo fungovat jak má, bez velké námahy pro uživatele. Tak jak přesně bylo zamýšleno autory simulátoru. Ale za těmito topology helpery se skrývá veškerá náročná práce implementace DCE a jsou zde řešeny a nakonec také splněny všechna kritéria pro využití DCE v síťových simulátorech. Díky těmto topology helperům simulátor ns-3 umožňuje jako jediný síťový simulátor využívat plnohodnotné implementace přímého vykonávání kódu. Před používáním těchto funkcí je tedy dobré si uvědomit, jakou obrovskou práci autoři programu udělali a jak snadné pro uživatele je tyto funkce využívat pro simulování. Níže je kód, který implementuje do programu DCE s podporou implementací linuxového jádra:

```
DceManagerHelper dceManager;
dceManager.SetNetworkStack ("ns3::LinuxSocketFdFactory", "Library",
StringValue ("liblinux.so"));
LinuxStackHelper stack;
stack.Install (c);
```

Funkce `SetNetworkStack` topology helperu `DceManagerHelper` určuje sdilenou knihovnu, která bude definovat, jak správně zpracovávat linuxové sockety v ns-3 při simulaci. `LinuxStackHelper` je poté aktivován na všech uzlech z kontejneru `c` pomocí funkce `Install`. Tímto je tedy implementace ns-3 DCE hotová a může se přejít k vytváření aplikací.

Vytváření aplikací ve scénáři s DCE probíhá pomocí topology helperu `DceApplicationHelper`. Tento topology helper umožňuje spustit přímo binární kód linuxové programu. V tomto scénáři bude využíván program `udp-perf`, který odpovídá aplikaci OnOffHelper, používanou v předešlém scénáři, a nabízí stejné možnosti nastavení. Níže je vidět kód programu pro server a klient a jeho popis:

```
//nastavení serveru
DceApplicationHelper process;
ApplicationContainer apps;
process.SetBinary ("udp-perf");
process.AddArgument ("--duration=100");
```

```
apps = process.Install (c.Get(0));
apps.Start (Seconds (1.0));
//nastavní klienta
process.SetBinary ("udp-perf");
process.ResetArguments ();
process.AddArgument ("--pktsize=1024");
process.AddArgument ("--port=9");
process.AddArgument ("--client");
oss << "--host=" << Ipv4AddressToString ("10.0.10.1");
process.AddArgument (oss.str ().c_str ());
process.AddArgument ("--duration=100");
apps = process.Install (c.Get(5));
apps.Start (Seconds (1.0));
```

Jako první je nastavena aplikace jaká se bude využívat, to se děje pomocí funkce `SetBinary("udp-perf")`. Nastavení aplikace pro funkci serveru je ve výchozím stavu, proto je pouze nutné zvolit dobu trvání dané aplikace pomocí funkce `AddArgument("--duration=100")`. Aplikace tedy bude běžet 100 s. Poté se již aplikace přiřadí uzlu 0 a zvolí se počátek spuštění aplikace v čase 1 s od spuštění simulace.

Pro nastavení klienta použijí rozšířené nastavení aplikace, jelikož klient opět musí generovat UDP datagramy o velikosti 1024 B na IP adresu 10.0.10.1 a port 9 uzlu 0. Nastavení aplikace pro funkci klienta na uzlu, velikosti datagramu a portu se provede přes funkci `AddArgument`, konkrétně pomocí třech argumentů `--client`, `--pktsize=1024`, `--port=9`. Nastavení cílové adresy serveru probíhá také pomocí `AddArgument`, avšak IP adresa přímo zadaná v argumentu `--host` musí být nejprve předena na řetězec `string` a poté je tento převedený řetězec nastaven v `AddArgument`. Nakonec je opět nastavena doba trvání 100 s, spuštění aplikace v čase 1 s a aplikace je přiřazena na uzel 5. Nyní je tedy scénář ns-3 DCE nastaven pro využití přímého vykonávání kódu v pokročilém módu, jsou vytvořené aplikace, přiřazeny IP adresy a linuxová sada protokolů.

7.5.2 Scénář ns-3 DCE Quagga

Poslední scénář se vytvářením bude nejvíce lišit od předešlých scénářů, jelikož se bude vedle DCE využívat také implementace Quagga, která využívá reálná rozhraní dostupná v Linuxu. Pro nastavování IP adres rozhraní a aktivování či deaktivování rozhraní se budou využívat linuxové příkazy odpovídající těmto operacím. Pro směrování se bude využívat protokol OSPF, který je v balíku Quagga reprezentován démonem ospfd. Nejprve je do scénáře opět nutné oproti předchozímu scénáři ns-3 DCE přidat modul pro podporu Quagga topology helperů:

```
#include "ns3/quagga-helper.h"
```

Nastavení sady protokolů bude stejné jako v předchozím scénáři, bude využíván topology helper `DceManagerHelper`. Pro implementaci balíku Quagga se bude využívat topology helper s názvem `QuaggaHelper`, jehož funkce se využijí na aktivování protokolu OSPF na jednotlivých uzlech. Příkazy přiřazení IP adresy na rozhraní směrovače a aktivování jednotlivých rozhraní budou řešeny ve speciálních třídách `AddAddress` a `RunIp`. Tyto dvě třídy jsou pro tento scénář velmi důležité a zajišťují jednu ze stěžejních konfigurací. Je proto důležité pochopit tyto funkce. Konfiguraci adres a aktivování rozhraní zajišťuje linuxový program s názvem `ip`. Právě tento program bude volán v třídě `RunIp` pomocí DCE, konkrétně funkce `SetBinary`, a dále budou využívány funkce programu `ip` na daném uzlu definovaném v proměnné `node`. V třídě `AddAddress` budou na základě jména rozhraní `name` a IP adresy s prefixem `address` přiřazeny IP adresy na daném uzlu. Odpovídající linuxový příkaz programu `ip` pro výše popsané operace:

```
ip -f inet addr add address dev name
```

Ve třídě `RunIp` jsou poté aktivována či deaktivována chtěná rozhraní, opět pomocí programu `ip`, konkrétně příkaz `ip link set`, poté zvolit jedno z rozhraní na daném uzlu a pro aktivaci rozhraní slouží příkaz `up`, resp. `down` pro deaktivaci. Kompletní zdrojové kódy těchto tříd a scénáře lze nalézt na přiloženém DVD.

Nyní již přímo k samotnému nastavení adres na rozhraní pro lepší ukázku fungování tříd popsaných výše. Ukázka je pro nastavení uzlu 1, který bude mít tři aktivní rozhraní s přiřazenými IP adresy:

```
AddAddress (c.Get (1), Seconds (0.1), "sim0", "10.0.10.2/24");
AddAddress (c.Get (1), Seconds (0.1), "sim1", "10.10.30.1/24");
```

```
AddAddress (c.Get (1), Seconds (0.1), "sim2", "10.10.50.1/24");
RunIp (c.Get (1), Seconds (0.11), "link set lo up");
RunIp (c.Get (1), Seconds (0.11), "link set sim0 up");
RunIp (c.Get (1), Seconds (0.11), "link set sim1 up");
RunIp (c.Get (1), Seconds (0.11), "link set sim2 up");
```

Lze tedy vidět, že konfigurace se dosáhne voláním tříd s nastavenými danými parametry. Jsou určeny názvy třech rozhraní na uzlu 1, zvolené vhodné IP adresování a poté jsou rozhraní aktivována parametrem `up`. Podobně se nastaví i ostatní uzly a model je po této stránce připraven. Dále je tedy důležité nastavit směrovací protokol OSPF pomocí topology helperu `QuaggaHelper`, odpovídající kód je následující:

```
QuaggaHelper quagga;
quagga.EnableOspf (c, "10.0.0.0/8");
quagga.Install (c);
```

Funkce `EnableOspf` zajišťuje na daném uzlu přiřazení všech připojených sítí do směrovacího procesu OSPF. Pokud by nějaká síť nebyla přidaná do směrování OSPF, směrovací protokol OSPF by ji ignoroval a nebyla by součástí směrovacích tabulek. Zvolený adresový prostor 10.0.0.0/8 však zajišťuje, že tomuto parametru budou odpovídat všechny dostupné sítě na každém uzlu a všechny sítě ve scénáři budou součástí směrování pomocí OSPF. Funkce `Install` poté aktivuje Quaggu, potažmo démona ospfd, na uzlech v kontejneru `c`, tedy všech uzlech ve scénáři.

Tímto krokem tedy máme zprovozněné směrování pomocí OSPF a zbývá vytvořit aplikace serveru a klienta pro generování provozu. Tyto aplikace budou vytvořeny totožně jako v předešlém scénáři ns-3 DCE, jelikož pro generování provozu bude využíváno opět programu `udp-perf` přes DCE s totožným nastavením aplikací. Toto vytváření tedy nebude znovu popisováno. Jedno nastavení se přece u aplikací bude lišit, a to začátek spuštění aplikací. V tomto scénáři po spuštění simulace trvá směrovacímu protokolu konvergování okolo 40 s. Počátek spuštění aplikací serveru a klienta je tedy nastaveno na 100 s od začátku simulace, doba trvání aplikací se však nezměnila a je stále nastavena na 100 s. Celá simulace v tomto scénáři tedy bude trvat celkově 200 s, avšak provoz z klienta na server bude trvat 100 s jako v předešlých scénářích.

Nyní tedy již je scénář hotový z hlediska zprovoznění balíku Quagga, přiřazení IP adres pro dané rozhraní, aktivování všech rozhraní a vytvoření aplikací.

7.6 Nastavení výpadku linek

Veškeré předešlé scénáře budou probíhat v podmínkách, kdy za dobu simulace nenastane žádný výpadek linky na směrovací trase ani žádná jiná linka. Pro lepší zhodnocení a porovnání výsledků by bylo dobré získat sledované parametry simulace, tedy zpoždění, kolísání zpoždění a ztrátovost, také ze scénářů s náhlým výpadkem linky na trase a nutným přesměrováním provozu po alternativní cestě sítí. Nastavení výpadku linky není nijak složité a stačí do každého scénáře přidat několik řádků kódu. Provedení ovšem nebude možné realizovat v jednom ze scénářů. V době psaní této diplomové práce není možné provést kompletní simulaci s výpadkem linky ve druhém scénáři ns-3 DCE. Tato skutečnost byla pro tento scénář potvrzena jedním z autorů ns-3 na oficiální fóru ns-3-users, který má na starost podporu metody DCE [19]. Problém je v implementaci DCE, která nebere v potaz stav deaktivovaného rozhraní určitého zařízení z kontejneru `NetDevice`. Linka samotná tedy má interní stav nestavený na neaktivní, avšak nikdo tento stav neregistruje. Směrovací protokol poté nemá jak zjistit, že určitá linka na trase je neaktivní a dále směřuje provoz přes onu linku, jako kdyby žádný výpadek nenastal. Autoři přislíbili prozkoumání této chyby, avšak nebyli schopni říci časový horizont nápravy. Pro scénář ns-3 DCE tedy simulace s výpadkem nebyla provedena. Pro ostatní dva scénáře ns-3 a ns-3 DCE Quagga je simulace provedena bezproblémově a postup provedení výpadku je popsán níže.

7.6.1 Scénář ns-3 s výpadkem linky

V prvním scénáři bude proveden výpadek linky mezi uzly 1 a 5. Je nutné zvolit správné rozhraní na daném uzlu, které má být deaktivováno. Rozhraní jsou číslované od 1 a na uzlu 1 má rozhraní linky spojující uzly 1 a 5 má hodnotu 3. Právě toto rozhraní uzlu 1 bude deaktivováno. Deaktivace linky proběhne v čase 60 s od počátku simulace a provede se funkcí `SetDown` topology helperu `Ipv4`. Následující kód odpovídá popisu výše:

```
Ptr<Node> n1 = c.Get (1);
Ptr<Ipv4> ipv43 = n1->GetObject<Ipv4> ();
uint32_t index3 = 3;
Simulator::Schedule (Seconds (60),&Ipv4::SetDown,ipv43, index3);
```

7.6.1 Scénář ns-3 DCE Quagga s výpadkem linky

U třetího scénáře s využítím Quaggy je nastavení výpadku jednoduché. Stačí pomocí třídy `RunIp` zvolit uzel, nastavit čas výpadku a určit rozhraní na uzlu, které se deaktivuje parametrem `Down`. Čas výpadku bude nastaven na dobu 160 s po spuštění simulace. Jak již bylo popsáno, samotné aplikace spouští svou činnost v čase 100 s, výpadek tedy nastane opět 60 s po spuštění aplikací jako v předchozím scénáři. Výpadek bude nastaven na rozhraní uzlu 1, jehož linka spojuje uzly 1 a 5, níže je odpovídající kód:

```
RunIp (c.Get (1), Seconds (160.0), "link set sim2 down");
```

7.7 Nastavení výstupních souborů a spuštění simulací

Všechny scénáře jsou již téměř kompletní, poslední, co zbývá nastavit, jsou výstupní soubory simulací s daty potřebnými pro analýzu výsledků. Poté bude ukázáno, jak spustit vytvořené simulace v simulátoru ns-3.

Výstupní soubory, jak již bylo řečeno, budou dvojího typu. První typ bude výstupní soubor .pcap, který je možné otevřít v programu Wireshark. Druhý typ bude textový soubor s výsledky, který slouží pro dodatečné kontroly a je užitečné tyto soubory s výsledky v čistém textu mít. Nastavení výstupních souborů s výsledky bude stejný pro všechny scénáře. Bude se využívat topology helperů `AsciiTraceHelperForDevice` a `PcapHelperForDevice`, první bude pro textový soubor s výsledky a druhý pro soubor typu `.pcap`. Budou použity funkce, které vytvoří výstupní soubory pro rozhraní každého uzlu v síti a zaznamená všechen provoz, který přes ně prochází. Možností nastavení výstupních souborů je více, např. pouze pro konkrétní uzel nebo konkrétní rozhraní, zde však budou generovány výstupní soubory ze všech rozhraní na uzlech. Odpovídající kód:

```
p2p.EnablePcapAll ("mereni/uzel");
p2p.EnableAsciiAll ("mereni/uzel");
```

Je vidět, že zachytávání provozu je nastaveno na linky `p2p` topology helperu `PointToPointHelper`, který byl nastaven na začátku modelu v dřívějších kapitolách. Dále je z kódu vidět umístění výstupních souborů ve složce `mereni` a v této složce budou soubory pojmenovány uzel a za tento název budou automaticky

doplněna dvě čísla, první číslo označující číslo uzlu a druhé číslo označující číslo rozhraní.

7.7.1 Spuštění simulací v simulátoru ns-3

Nyní jsou tedy všechny scénáře kompletně hotové a připravené na spuštění. Pro sestavení a kompilaci vytvořených modelů se používá Waf, který již byl popsán. Každá implementace má mírně odlišný proces spuštění modelů, avšak rozdíl je zejména v prvotním sestavení a kompilaci modelů. Tyto procesy se stejně jako instalace těchto implementací postupně mění, zejména zjednodušují, jelikož často může být samotné spuštění vytvořených modelů docela komplikované.

Nejjednodušší je spuštění prvního scénáře pouze implementace ns-3. Soubor typu `.cc` se zdrojovými kódy je potřeba pouze zkopírovat do adresáře `scratch`, který je umístěn v hlavní složce nainstalovaného programu ns-3. Poté stačí již využít programu Waf a pomocí příkazu `./waf --run` s názvem souboru se provede spuštění simulace.

V druhém scénáři ns-3 DCE je nutné nejprve modifikovat konfigurační soubor `clone_and_compile_ns3_dce.sh` popsaný dříve při instalaci programu. Sem je nutné dopsat cestu k vytvořenému modelu spolu s moduly použitými v modelu. Poté je nutné celý program ns-3 DCE opět znovu sestavit a po tomto sestavení již je vytvořený model sestaven také a opět pomocí příkazu `./waf --run` a název modelu spustit simulaci.

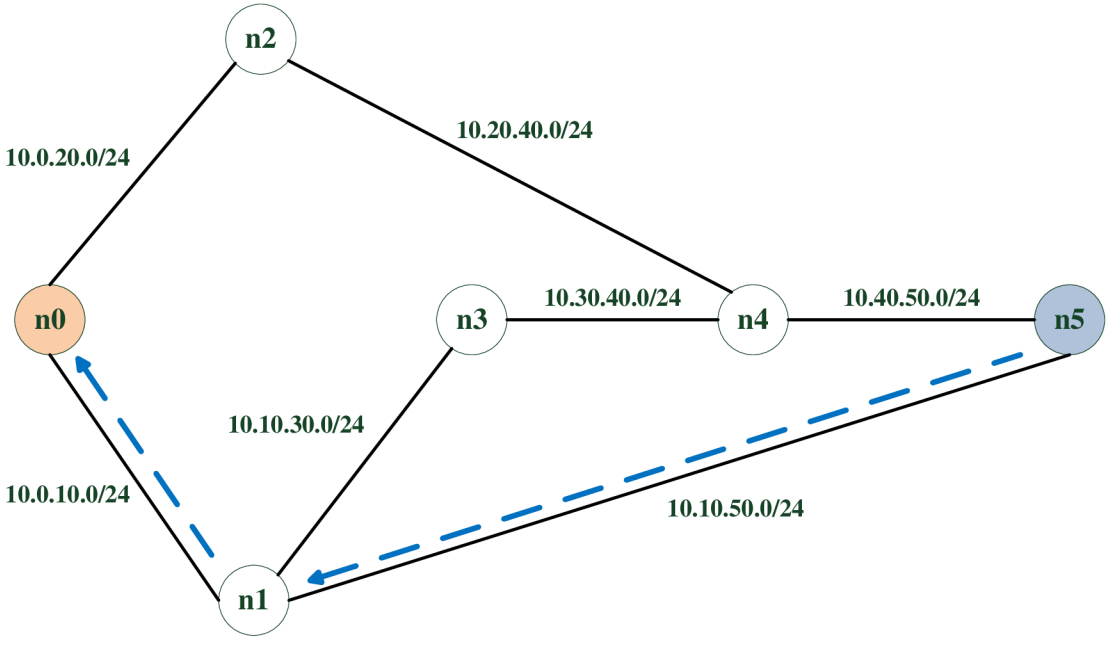
Ve třetím scénáři ns-3 DCE Quagga je potřeba model zkopírovat do vytvořené nové složky v adresáři `myscripts` umístěný v hlavním adresáři programu. V této nové složce s již zkopírovaným modelem je nutné vytvořit dodatečný konfigurační soubor, který opět musí obsahovat moduly použité v simulaci a také název modelu. Poté tento konfigurační využijeme k sestavení vytvořeného modelu. Bude sestaven pouze daný model bez nutnosti sestavovat celý program ns-3 DCE Quagga jako v předchozím scénáři, což ušetří spoustu času. Po sestavení se simulace spustí příkazem `./waf --run` a název modelu.

Nyní se již předpokládá, že všechny simulace proběhly úspěšně, všechny výstupní soubory jsou vygenerovány a nyní se může přejít k vyhodnocení výsledků.

8 Vyhodnocení výsledků simulací

V poslední kapitole diplomové práce dojde k analýze výsledků a shrnutí poznatků z této analýzy. Na následujících řádcích budou zobrazeny a popsány grafy sestavené z výsledných zaznamenaných hodnot ze simulací. Sledovanými parametry ve výsledcích pro všechny scénáře bude zpoždění, kolísání zpoždění a ztrátovost. V dalších kapitolách budou nejprve ukázány výsledky pro jednotlivé scénáře a poté budou zobrazeny výsledky srovnání všech scénářů pro lepší demonstraci v jednom grafu. Kapitoly budou rozděleny na výsledky simulací sítě bez výpadku linky a sítě s výpadkem linky.

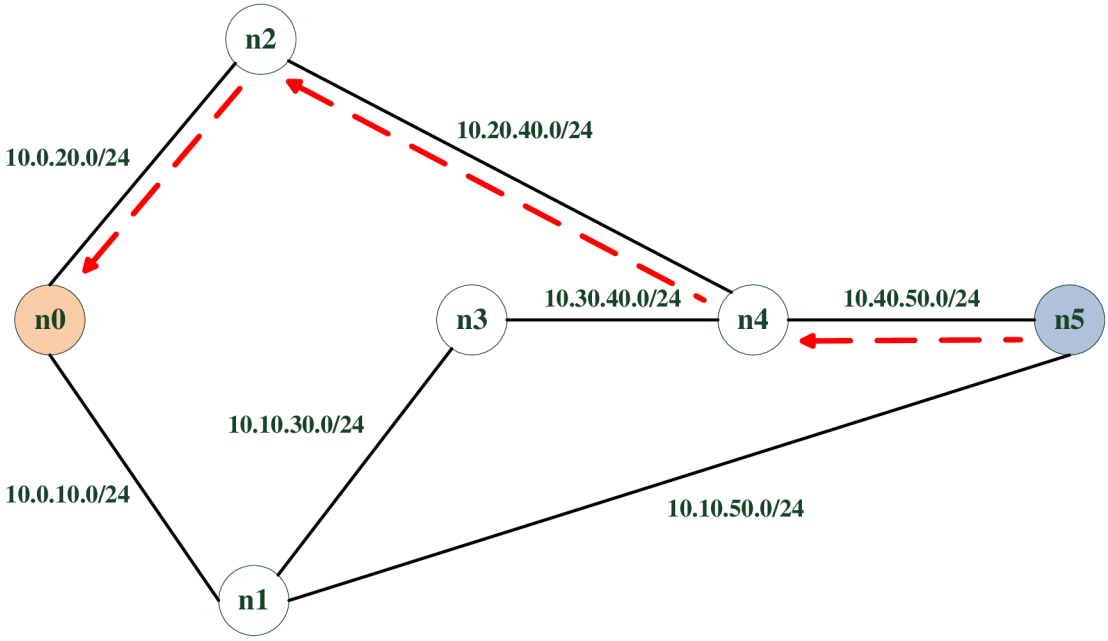
Obecně ke společnému modelu sítě pro všechny scénáře lze říci, že protokol OSPF bude směrovat provoz ve všech scénářích bez výpadku linky přes uzel 1. Je to dáno tím, že všechny rozhraní na uzlech mají stejnou cenu, což je metrika OSPF. Cena je totiž určena z kapacity linky, která je nastavená pro všechny linky na stejnou hodnotu. Jelikož celková cena cesty přes uzel 1 je nejnižší, protokol OSPF ji bude využívat jako primární cestu pro provoz z uzlu 5 na uzel 1. Na obr. 8.1 je v modelu sítě naznačen průchod paketů sítí bez výpadku linky.



Obr. 8.1: *Primární cesta paketů v síti bez výpadku*

Takto jsou pakety směrovány v ideálním případě, kdy v síti nenastane žádný výpadek linky, přes kterou provoz prochází. Ve scénářích s výpadkem linky však taková událost nastane a v čase 60 s od spuštění generování paketů dojde k výpadku

linky mezi uzly 1 a 5. Tímto výpadkem linky bude muset směrovací protokol OSPF přesměrovat generovaný provoz po alternativní cestě k cíli. Jako alternativní cesta bude zvolena cesta přes uzel 4 a dál přes uzel 2, jelikož má z dostupných cest nejnižší cenu. Na obr. 8.2 je znázorněna alternativní cesta při výpadku linky mezi uzly 1 a 5. V dalších kapitolách již tedy cesty směrovacího protokolu OSPF nebudou nijak detailněji rozebírány.



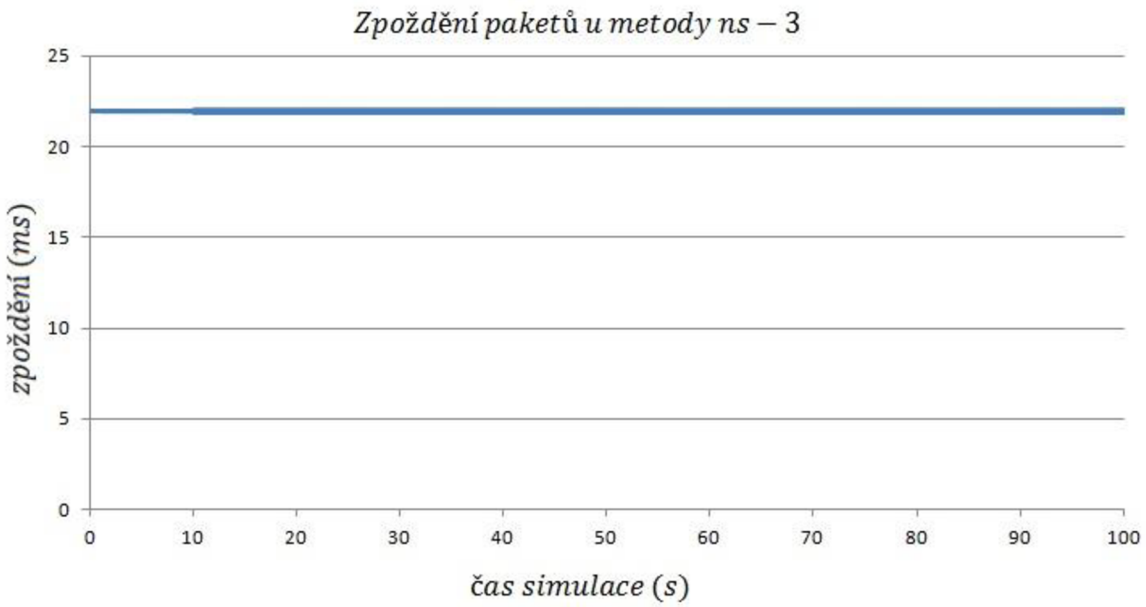
Obr. 8.2: *Alternativní cesta paketů po výpadku linky*

8.1 Analýza prvního scénáře ns-3

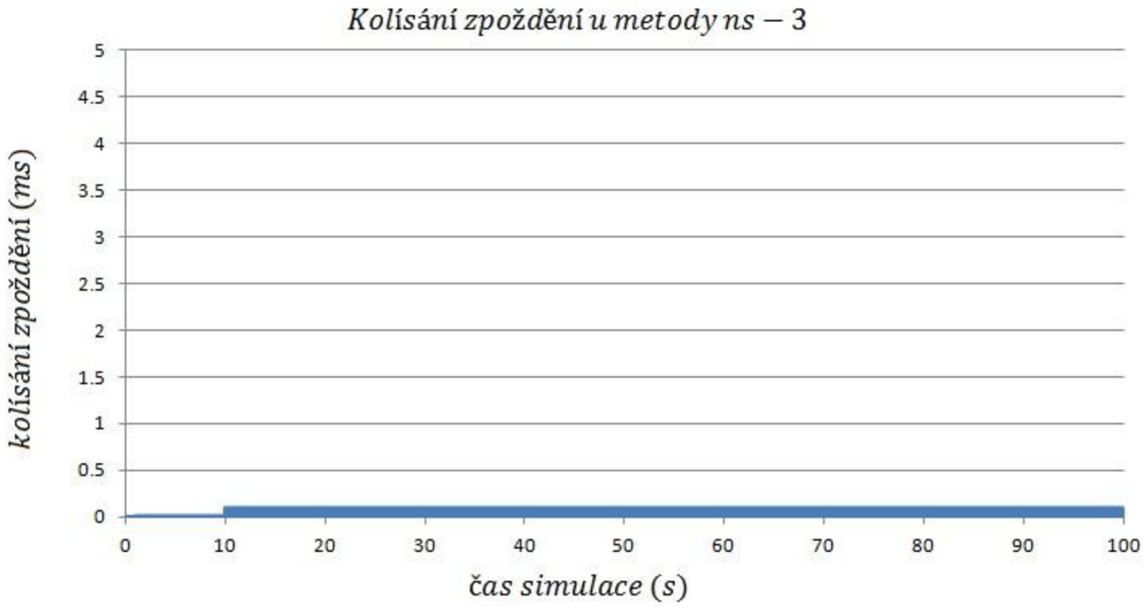
Scénář ns-3 využívá pouze základních protokolových implementací a aplikací ze simulátoru ns-3, lze tedy očekávat, že výsledky budou ze všech scénářů nejméně přesné a průběhy budou hladší než by v reálných sítích opravdu byly. Neznamená to však, že by měli být výsledky nějak výrazněji nepřesné, ale spíše mají informativní hodnotu o parametrech simulované sítě.

8.1.1 Výsledky sítě bez výpadku linky

V prvním grafu na obr. 8.3 lze vidět průběh paketového zpoždění při průchodu sítí z uzlu 5 na uzel 0. Z grafu opravdu vyplývá, že průběh je po celou dobu simulace téměř neměnný, průměrná hodnota zpoždění je 21,96 ms. Po dobu simulace neproběhla v síti žádná nečekaná událost a zpoždění je tedy téměř konstantní. Kolísání zpoždění se z důvodu konstantního zpoždění bude samozřejmě pohybovat kolem nulových hodnot, viz obr. 8.4.



Obr. 8.3: *Zpoždění paketů scénáře ns-3 bez výpadku linky*



Obr. 8.4: *Kolísání zpoždění paketů scénáře ns-3 bez výpadku linky*

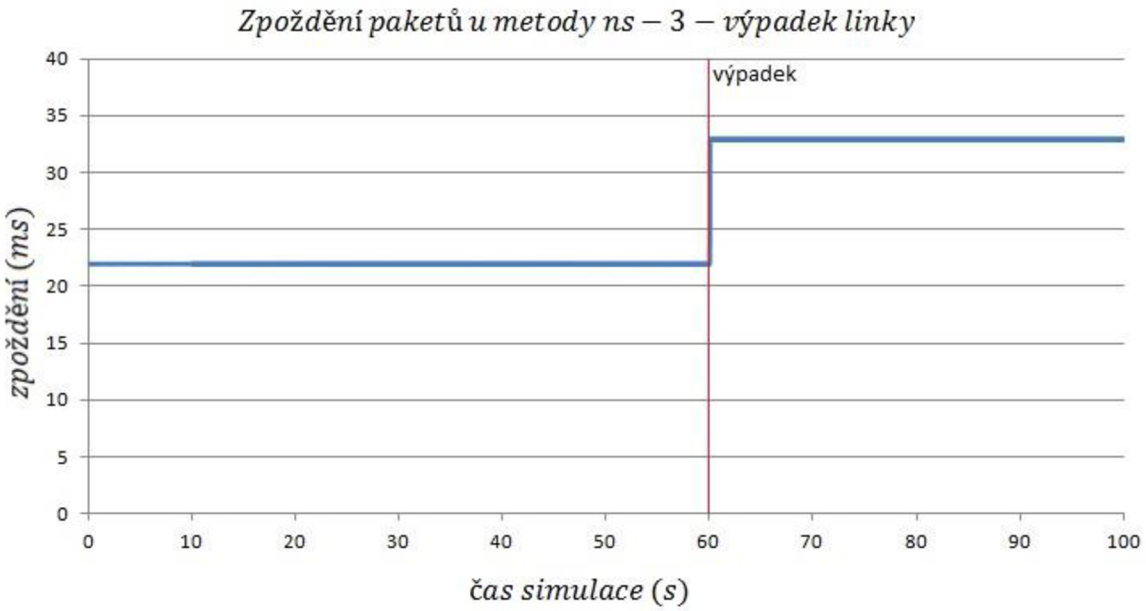
V tabulce tab. 8.1 jsou výsledné sledované parametry pro tento scénář bez výpadku linky. Průměrný jitter nebo kolísání zpoždění je 0,029 ms, což by v reálné síti byl opravdu dobrý výsledek a to samé platí pro ztrátovost, která odpovídá hodnotě 0,017 %.

Tab. 8.1: Sledované parametry pro scénář 1 bez výpadku linky

Sledovaný parametr	Hodnota
Průměrné zpoždění:	21,96 ms
Průměrný jitter:	0,029 ms
Ztrátovost:	0,017 %

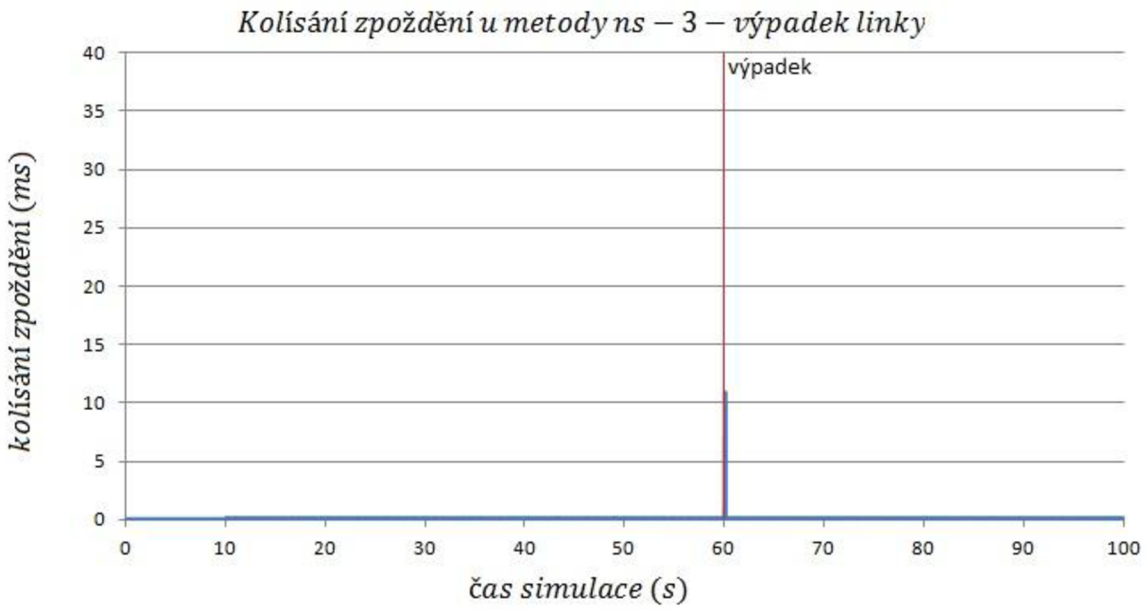
8.1.2 Výsledky sítě s výpadkem linky

Při výpadku linky v čase 60 s se dá očekávat, že tato nečekaná událost změní průběhy sledovaných parametrů v době výpadku a průměrná hodnota zpoždění, kolísání zpoždění i ztrátovosti se zvýší. Na obr. 8.5 je zobrazen průběh zpoždění v sítí s výpadkem linky mezi uzlem 1 a 5. V grafu je naznačen výpadek v čase 60 s. V tomto čase tedy musí dojít k přesměrování provozu na alternativní cestu a dojde tím také ke zvýšení paketového zpoždění, jelikož provoz prochází přes dva směrovače a cesta se tím časově prodlouží, např. odbavováním na směrovačích. V čase 60 s je tedy vidět nárůst z hodnoty 22 ms na hodnotu okolo 32 ms. Opět je vidět ideální průběh před výpadkem, po výpadku i samotný nárůst zpoždění je ideální a skokový, což by v reálné síti nebylo možné.



Obr. 8.5: Zpoždění paketů scénáře ns-3 s výpadkem linky

Kolísání zpoždění má v tomto případě pouze jeden významný skok v čase 60 s, jak lze vidět na obr. 8.6. Toto je dáno výpadkem linky a skokové změně zpoždění.



Obr. 8.6: *Kolísání zpoždění scénáře ns-3 s výpadkem linky*

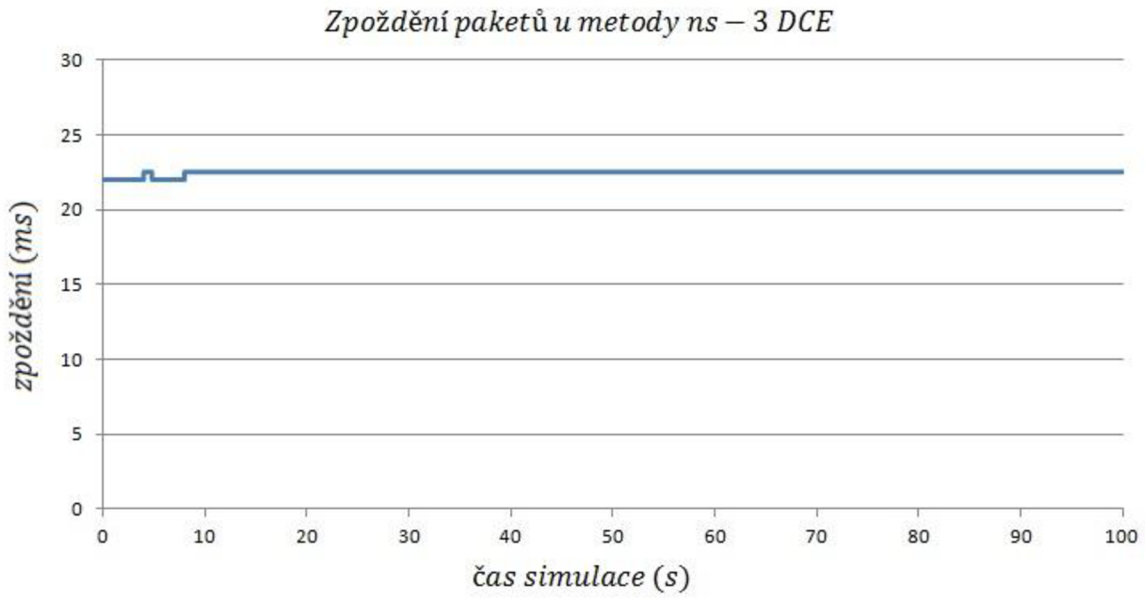
Nyní zbývá pouze ukázat průměrné hodnoty sledovaných parametrů. V tab. 8.2 jsou výsledky pro scénář ns-3 s výpadkem linky. Podle očekávání jsou hodnoty vyšší, ztrátovost se stále drží na velmi nízkých hodnotách i při výpadku.

Tab. 8.2: *Sledované parametry pro scénář ns-3 s výpadkem linky*

Sledovaný parametr	Hodnota
Průměrné zpoždění:	26,39 ms
Průměrný jitter:	0,031 ms
Ztrátovost:	0,05 %

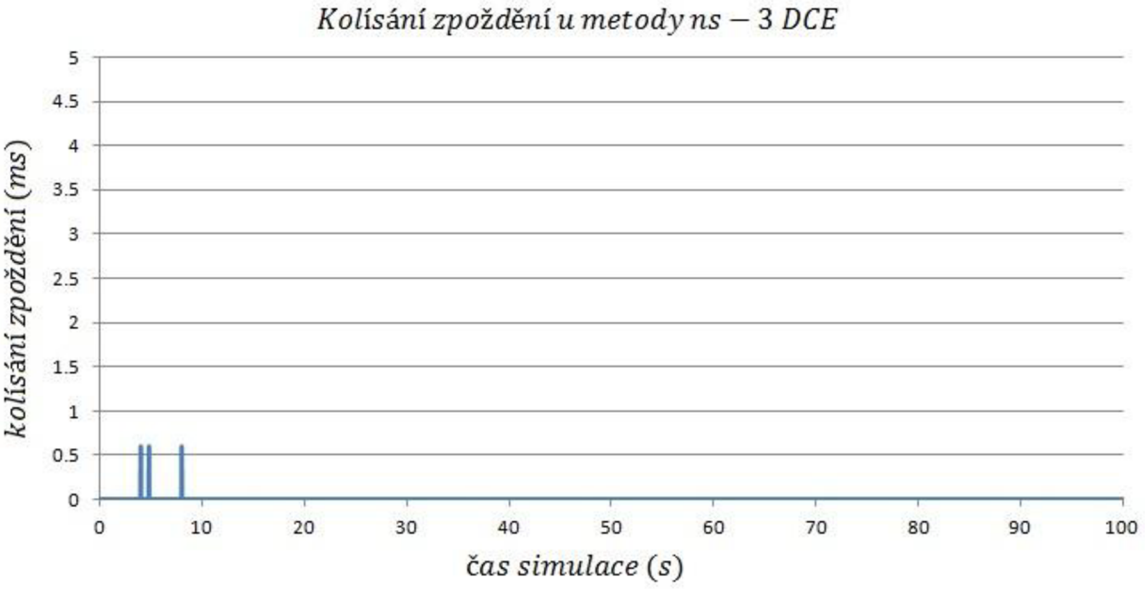
8.2 Analýza druhého scénáře ns-3 DCE

Druhý scénáři by měl s využitím přímého vykonávání kódu a linuxových implementací přinést přesnější výsledky simulací. V tomto scénáři nebylo možné provést simulaci s výpadkem linky, jak již bylo popsáno dříve, takže budou ukázány pouze výsledky sítě bez výpadku linky. Na obr. 8.7 lze vidět průběh zpoždění po dobu simulace.



Obr. 8.7: *Zpoždění paketů scénáře ns-3DCE bez výpadku linky*

Z průběhu vyplývá, že zpoždění je opět téměř konstantní, pouze z počátku simulace se objevují mírné výkyvy, které ale nejsou nijak výrazné. Na obr. 8.8 jsou tyto výkyvy vidět v průběhu kolísání zpoždění. Mimo tyto výkyvy je kolísání zpoždění téměř nulové a velmi podobné prvnímu scénáři.



Obr. 8.8: *Kolísání zpoždění scénáře ns-3 DCE bez výpadku linky*

V tab 8.3 jsou uvedeny výsledné průměrné hodnoty sledovaných parametrů. Podle očekávání jsou mírně odlišné od prvního scénáře. Průměrné zpoždění se nepatrně zvýšilo při snížení průměrného kolísání zpoždění, což znamená, že interval y

mezi přijímanými pakety byly téměř stejné. Ztrátovost se opět pohybuje u nulových hodnot.

Tab. 8.3: *Sledované parametry pro scénář ns-3 DCE bez výpadku linky*

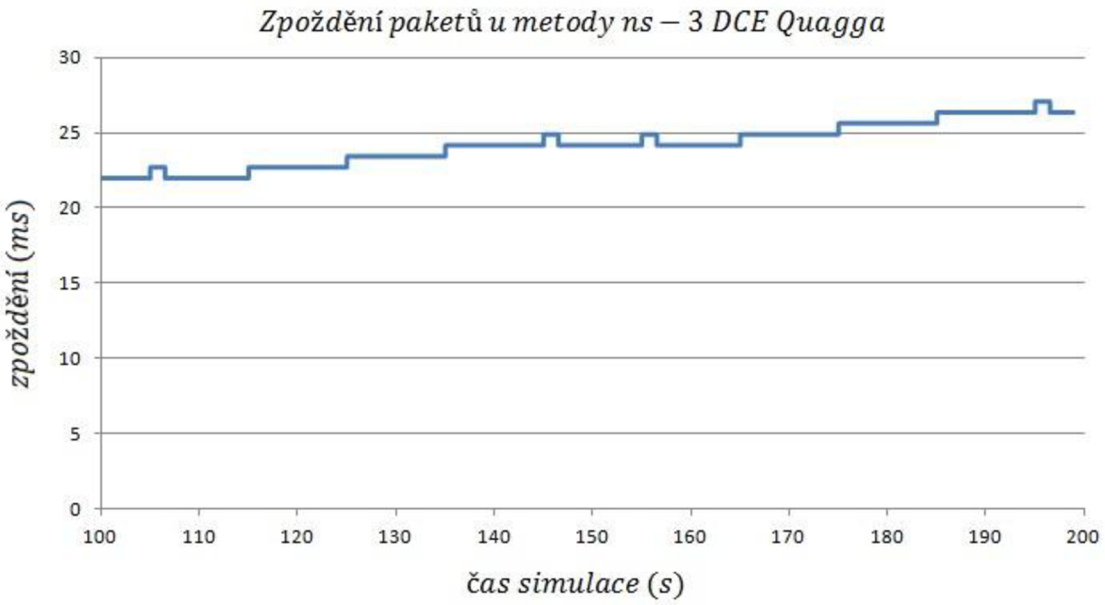
Sledovaný parametr	Hodnota
Průměrné zpoždění:	22,52 ms
Průměrný jitter:	0,0005 ms
Ztrátovost:	0,0222 %

8.3 Analýza třetího scénáře ns-3 DCE Quagga

Třetí scénář by měl přinést nejpřesnější výsledky. Využití přímého vykonávání kódu spolu s balíkem Quagga nabízí pro simulační model reálné prostředí, jak již bylo vysvětleno dříve. Je dobré připomenout, že počátek generování provozu v tomto scénáři je v čase 100 s a doba trvání generování provozu je 100 s. V grafech je tedy znázorněn průběh mezi časy 100 s a 200 s od počátku spuštění simulace.

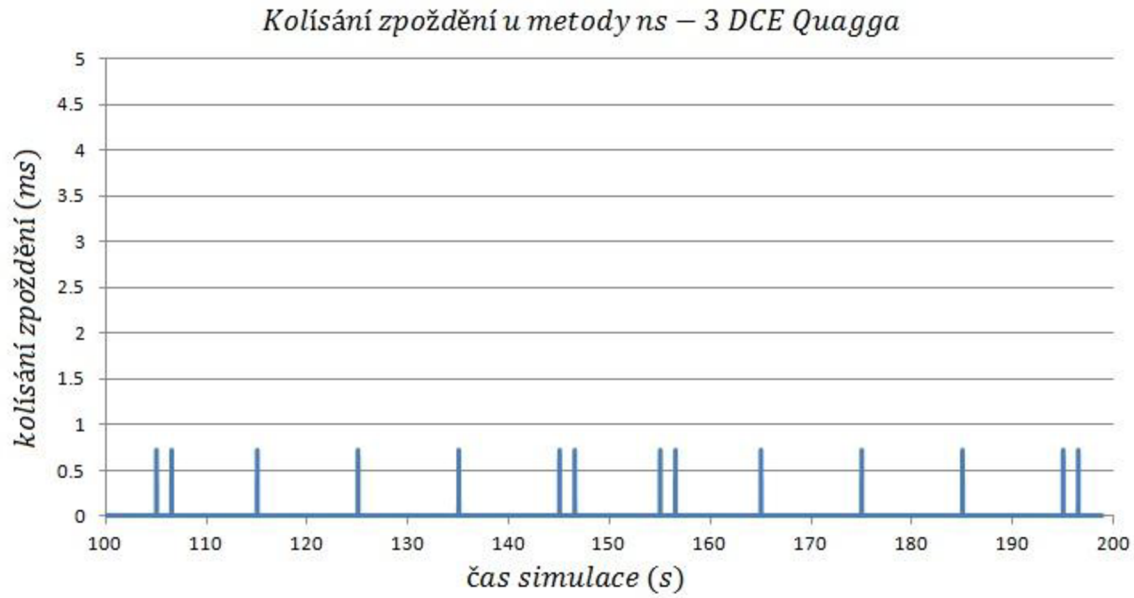
8.3.1 Výsledky sítě bez výpadku linky

Na obr. 8.9 je zobrazen průběh zpoždění pro síť bez výpadku linky. Průběh má mírně vzrůstající charakter po celou dobu simulace. Průběh již není ideálně rovný, ale má různé výkyvy ve zpoždění paketů, nejsou však nijak výrazné a odpovídají reálnějším podmínkám sítě.



Obr. 8.9: *Zpoždění paketů scénáře ns-3DCE Quagga bez výpadku linky*

Na obr. 8.10 je dále znázorněn průběh kolísání zpoždění a zaznamenává výkyvy z průběhu zpoždění. Maximální hodnota kolísání nepřekračuje hodnotou 1 ms, což značí, že v intervalech mezi přijatými pakety nebyly velké rozdíly a tato hodnota by dostačovala i pro nejnáročnější aplikace. Průměrné hodnoty sledovaných parametrů jsou pak ukázány v tab. 8.4 a lze vidět, že se průměrně zpoždění oproti předchozím scénářům zvýšilo. Ztrátovost v tomto scénáři zůstává na nízkých hodnotách okolo nuly.



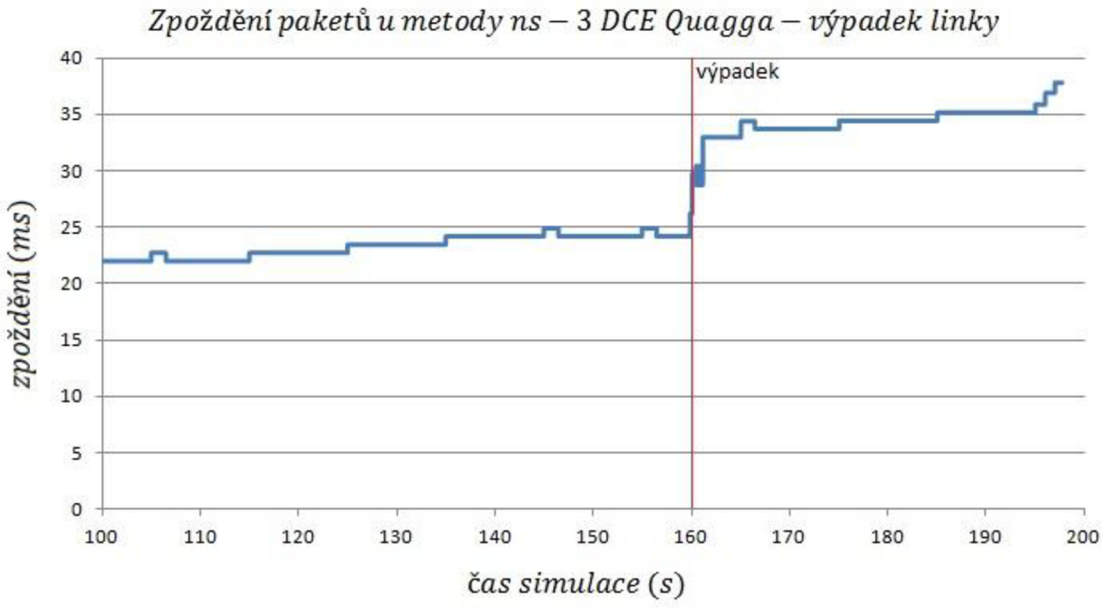
Obr. 8.10: *Kolísání zpoždění scénáře ns-3 DCE Quagga bez výpadku linky*

Tab. 8.4: *Sledované parametry pro scénář ns-3 DCE Quagga bez výpadku linky*

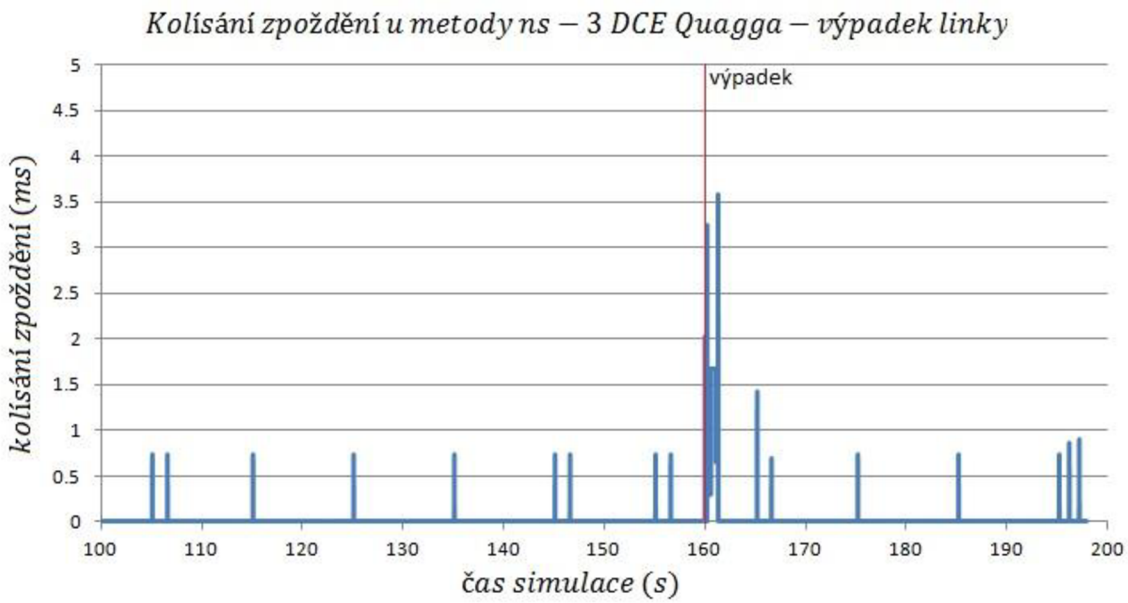
Sledovaný parametr	Hodnota
Průměrné zpoždění:	24,227 ms
Průměrný jitter:	0,0014 ms
Ztrátovost:	0,0215 %

8.3.2 Výsledky sítě s výpadkem linky

Výpadek linky nastane v čase 60 s po spuštění aplikací generující provoz. V tomto scénáři tedy nastane výpadek linky v čase 160 s. Průběh zpoždění lze vidět na obr. 8.11. V čase 160 s je opravdu vidět, jak po výpadku linky dojde ke zvýšení hodnoty zpoždění po přesměrování provozu na alternativní cestu. Skok ve zpoždění už není ideální tak jako v prvním scénáři, ale je vidět postupné zvýšení až se průběh ustálí na nové hodnotě. Protokolu OSPF totiž samozřejmě nějaký čas trvá, než se o výpadku dozví a než provoz přesměruje na novou cestu. Proto se také očekává, že v tuto dobu výpadku bude zahozeno určité množství paketů, které jsou směrovány přes cestu, na které nastal výpadek. Na obr. 8.12 je zobrazen průběh kolísání zpoždění, kde jsou samozřejmě největší výkyvy v době výpadku linky a při přesměrování provozu. Hodnoty se však pohybují stále v přiměřeném rozsahu a dle očekávání.



Obr. 8.11: *Zpoždění paketů scénáře ns-3DCE Quagga s výpadkem linky*



Obr. 8.12: *Kolísání zpoždění scénáře ns-3 DCE Quagga s výpadkem linky*

Tab. 8.5 potom na závěr opět ukazuje průměrné hodnoty sledovaných parametrů. Průměrné zpoždění i kolísání zpoždění se dle očekávání při simulaci s výpadkem zvýšily. Nečekaná hodnota se však vyskytla u ztrátovosti. Ztrátovost se téměř vůbec nezvýšila, což neodpovídá předpokladům. Je těžké najít příčinu tohoto stavu, jak již ale bylo napsáno, implementace DCE spolu s Quaggou se stále vyvíjí a odstraňují se chyby. Je proto možné, že program není správně odladěn pro tuto situaci, jelikož zdrojový kód scénáře je v pořádku a výpadek linky je řešen pouze dopsáním jednoho řádku. Chybu se každopádně nepodařilo vyhledat a odstranit.

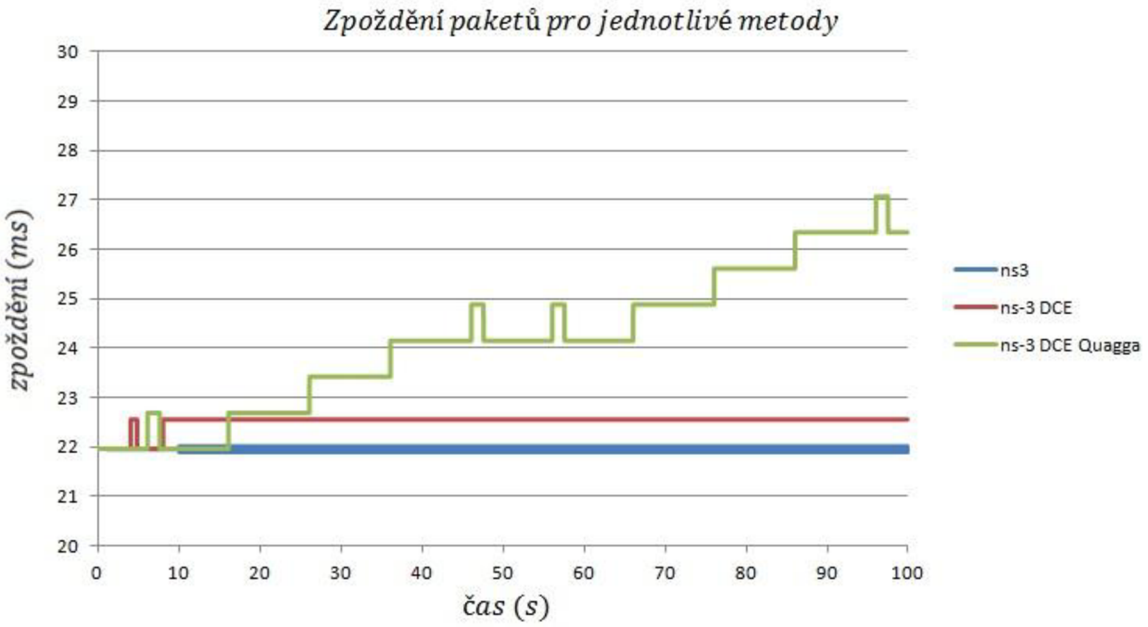
Tab. 8.5: *Sledované parametry pro scénář ns-3 DCE Quagga s výpadkem linky*

Sledovaný parametr	Hodnota
Průměrné zpoždění:	27,957 ms
Průměrný jitter:	0,0238 ms
Ztrátovost:	0,0213 %

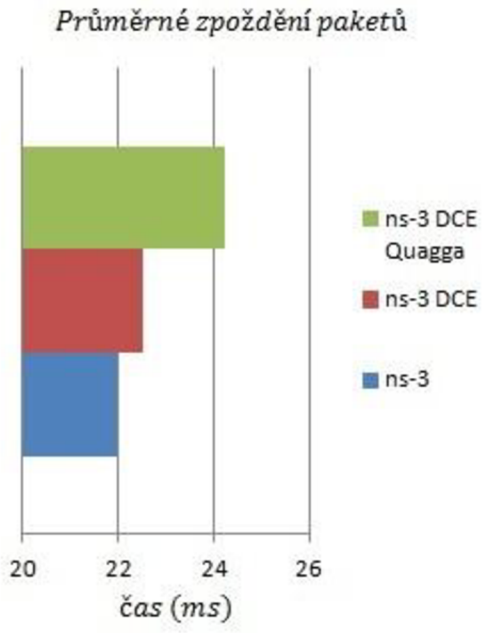
8.4 Závěrečné srovnání scénářů

V této poslední kapitola budou v grafu zaneseny průběhy ze všech scénářů pro lepší demonstraci rozdílností. Na obr. 8.13 jsou průběhy zpoždění společně ze všech scénářů bez výpadku linky. Je vidět, že první dva scénáře mají pouze mírně odlišný průběh, třetí scénář ns-3 DCE Quagga se průběhem liší od ostatních scénářů.

Samozřejmě nejdůvěryhodnější by bylo srovnání s průběhem z reálné sítě, což z časové náročnosti nebylo možné provést. Jelikož jsou však zdrojové kódy implementací odzkoušené a funkční, lze tedy dle teoretických předpokladů vyhodnotit, že průběh ns-3 DCE Quagga je možné brát jako nejpřesnější a nejvíce odpovídající realitě. Na obr. 8.14 jsou poté srovnány průměrné hodnoty zpoždění ze všech scénářů. U zpoždění je tedy mezi jednotlivými implementacemi vidět rozdílnost v řádu milisekund.

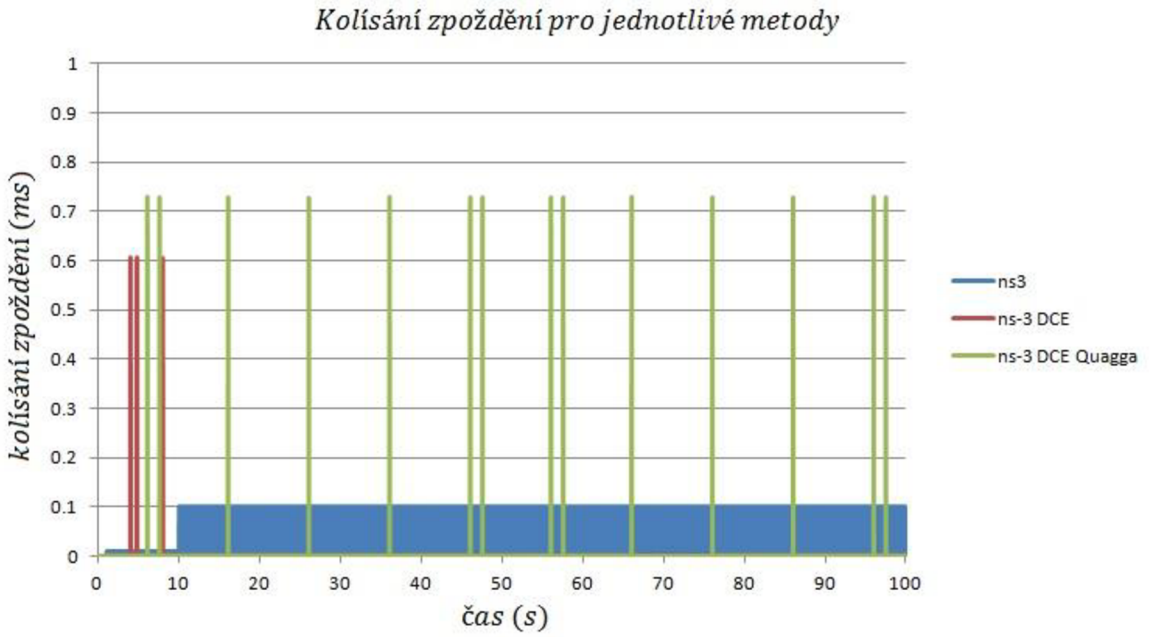


Obr. 8.13: *Srovnání zpoždění mezi jednotlivými scénáři*

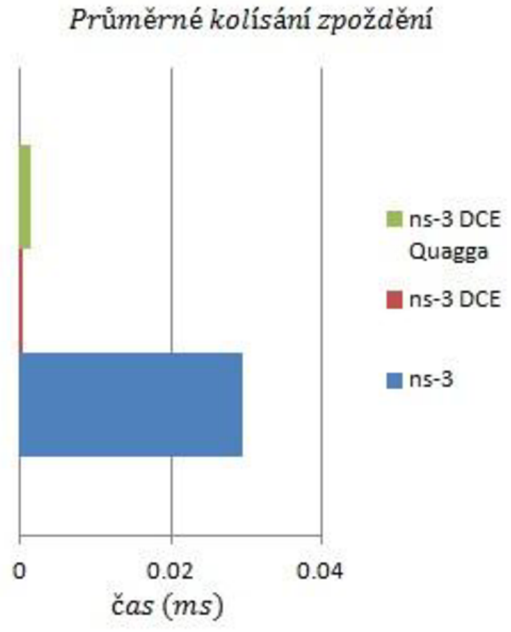


Obr. 8.14: *Srovnání průměrných hodnot zpoždění*

Jako další bude ukázáno srovnání hodnot kolísání zpoždění na obr. 8.15 a také průměrných hodnot kolísání zpoždění na obr. 8.16 pro všechny scénáře bez výpadku linky. U všech scénářů maximální hodnota nepřesáhne po celou dobu simulace hodnotu 1 ms. Kolísání je tedy téměř nulové pro všechny implementace.

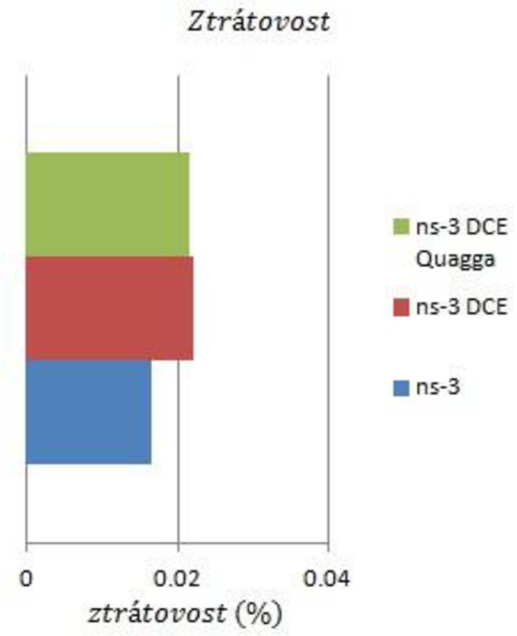


Obr. 8.15: *Srovnání kolísání zpoždění mezi jednotlivými scénáři*



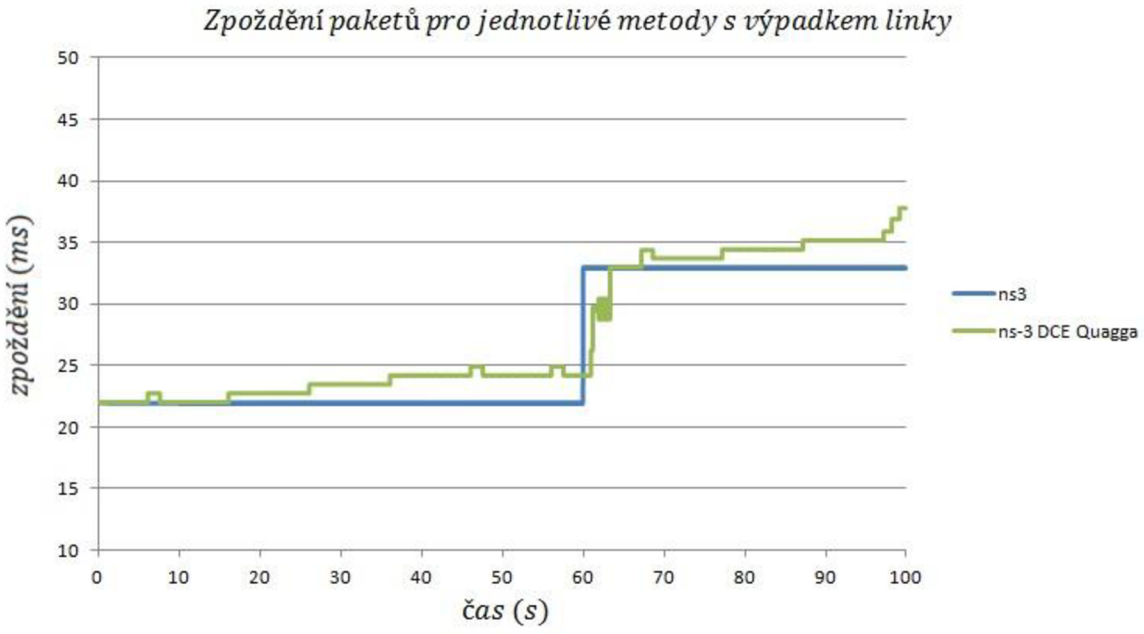
Obr. 8.16: *Srovnání průměrných hodnot kolísání zpoždění*

Jako poslední pro scénáře bez výpadku budou srovnány hodnoty ztrátovosti na obr. 8.17. Všechny implementace mají téměř identické hodnoty okolo 0,02 %.

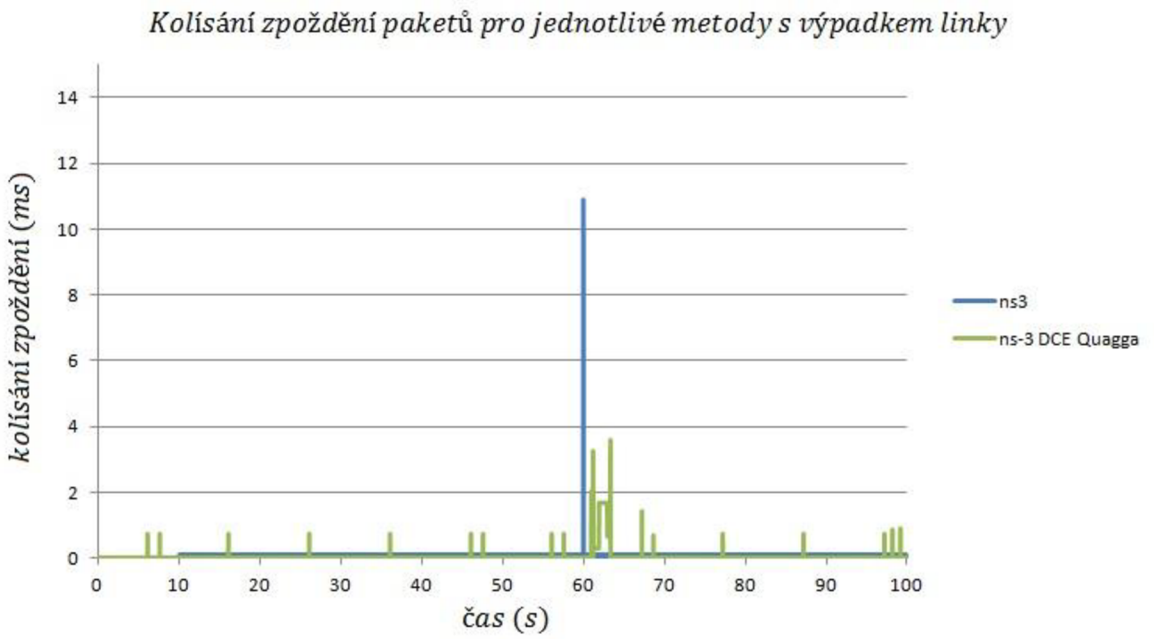


Obr. 8.17: *Srovnání hodnot ztrátovosti*

Nyní dojde na srovnání výsledků mezi implementacemi při výpadku linky. Průběhy zpoždění jsou vidět na obr. 8.18 a mají velmi podobný průběh. Rozdíly jsou v řádu milisekund. Průběhy kolísání zpoždění jsou poté zobrazeny v grafu na obr. 8.19. V čase výpadku má kolísání zpoždění ve scénáři ns-3 několikanásobnou hodnotu oproti scénáři ns-3 DCE Quagga.



Obr. 8.18: *Srovnání zpoždění mezi jednotlivými scénáři s výpadkem linky*



Obr. 8.19: *Srovnání kolísání zpoždění mezi jednotlivými scénáři s výpadkem linky*

Ztrátovost zde nebude srovnávána z důvodu uvedeného dříve, neměl by totiž žádnou vypovídající hodnotu. Tímto je tedy srovnání výsledků ze všech scénářů kompletní. Potvrdilo se, že ns-3 DCE Quagga má přesnější a reálnější průběhy sledovaných parametrů, oproti ostatním implementacím se však většinou liší pouze nepatrně. Z výsledků tedy vyplývá, že pro tyto návrhy v diplomové práce se nevyplatí používat implementace DCE nebo DCE Quagga. Je to dáno především časovou náročností simulací, která je oproti implementaci ns-3 několikanásobně vyšší. Tyto závěry ale samozřejmě platí pro nastavené parametry těchto vytvořených modelů, které by v reálných sítích byly odlišné.

Závěr

Cílem diplomové práce bylo teoreticky popsat a vysvětlit princip fungování simulátoru ns-3 a zaměřit se na zvýšení hodnověrnosti simulací pomocí implementace metody přímého vykonávání kódu a softwarového balíku Quagga. Měly být vytvořené scénáře pro jednotlivé implementace a poté provedeno srovnání výsledků těchto simulací.

V první části byl popsán protokol IPv4 a základy směrování v sítích využívající tento protokol. Byl vysvětlen transportní protokol UDP a následně princip statického a dynamického směrování. Stručně byl nastíněn rozdíl mezi dynamickými protokoly využívajícími odlišné algoritmy. Důkladně byl popsán směrovací protokol OSPF, diskrétní simulace a představen simulátor ns-3 a jeho implementace metody přímého vykonávání kódu a Quagga.

V praktické části byla pozornost zaměřena na poměrně komplikované sestavení a instalaci simulátoru ns-3 a jeho implementací. Byl navrhnut model sítě se 6 uzly pro zajištění směrování. Velmi detailní popis byl věnován celkovému procesu vytváření a nastavování simulačních scénářů pro všechny implementace, bylo poukázáno na rozdíly ve vytváření scénářů. Pro každou implementaci byl vytvořen scénář bez výpadku linky a s výpadkem linky. Scénář s výpadkem linky nebylo možné vytvořit o implementace ns-3 DCE. Sledovanými parametry v simulacích bylo zpoždění, kolísání zpoždění a ztrátovost, které jsou po provedení simulací analyzovány a zpracovány do výsledných grafů a tabulek.

Výsledky simulací jsou přehledně vyhodnoceny v závěrečných kapitolách. Ze vzájemných srovnání jednotlivých implementací lze vidět mírné odchylky v řádu ms, což není hodnota, která by ovlivňovala funkčnost většiny síťových aplikací. Proto především vzhledem k výpočetní náročnosti implementací DCE a DCE Quagga by nebylo vhodné je využívat pro navržený a vytvořený model sítě v této diplomové práci. Samotná implementace ns-3 vykazuje vzhledem k ostatním implementacím podobnou přesnost sledovaných parametrů a výpočetní náročnost je menší.

Literatura

[1] PUŽMANOVÁ, R. *Moderní komunikační sítě od A do Z.* 2. aktualizované vydání. Brno: Computer Press, 2006. 430 s. ISBN 80-251-1278

[2] WENDELL, O. *CCENT/CCNA ICND1 640-822 Official Cert Guide*. Third Edition. Cisco Press, 2011. 736 s. ISBN 978-1-58720-425-8

[3] WENDELL, O. *CCENT/CCNA ICND2 640-822 Official Cert Guide*. Third Edition. Cisco Press, 2011. 736 s. ISBN 978-1-58720-425-7

[4] GRYGAREK, J. *Směrovací protokol OSPF: Princip, oblasti, topologická databáze, budování směrovací tabulky; funkce na různých typech sítí* [online]. [cit. 2012-30-10]. Dostupné z URL: <http://www.cs.vsb.cz/grygarek/SPS/lect/OSPF/ospf.html>

[5] MOY, J. *RFC 1583: OSPF Version 2* [online]. 1994 [cit. 2012-30-10]. Dostupné z URL: <http://www.ietf.org/rfc/rfc1583.txt>

[6] *ns-3 Project: Network Simulator: ns-3 Tutorial* [online]. 2012 [cit. 2012-15-11]. Dostupné z URL: < http://www.nsnam.org/ns-3-15/documentation/>

[7] LACAGE, M. *Experimentation Tools for Networking Research* [online]. University of Nice Sophia Antipolis. 2010. Doctor`s Thesis. Dostupné z URL: <http://www.mendeley.com/profiles/mathieu-lacage>

[8] URBANI, F. *Direct Code Execution* [online]. Dostupné z URL: <http://www-sop.inria.fr/members/Frederic.Urbani/DCE/>

[9] *NS-3 Quagga Documentation* [online]. 2012 [cit. 2012-20-11]. Dostupné z URL: <http://www.nsnam.org/~thehajime/ns-3-dce-quagga/getting-started.html>

[10] *NS-3 Wiki. Installation* [online]. 2012 [cit. 2012-20-11]. Dostupné z URL: <http://www.nsnam.org/wiki/index.php/Installation>

[11] DOUŠA, J. *Diskrétní simulace* [online]. 2010. Dostupné z URL: <https://dsn.felk.cvut.cz/wiki/vyuka/cviceni/x36dsi/start>

[12] BANKS, J., CARSON J., NELSON B. *Discrete-Event System Simulation*. Fourth Edition. Prentice Hall, 2005. 624 s.

[13] GARRETT Y. *Efficient large-scale computer and network models using optimistic parallel simulation*. ProQuest Company, 2006. 136 s.

[14] FUJIMOTO, R. *Parallel and Distributed Simulation systems.* A Wiley-Interscience Publication. 2000. 300 s.

[15] KŘIVÝ I., KINDLER E. *Simulace a modelování.* Učební texty Ostravské univerzity. 2001. 146 s.

[16] Quagga Routing Suite [online]. 2013 [cit. 2013-08-05]. Dostupné z URL: <http://www.nongnu.org/quagga/>

[17] SOSINKY, B. *Počítačové sítě.* Brno: Computer Press, 2010. 840 s. ISBN 978-80-251-3363-7

[18] *ns3 Documentation* [online]. 2013 [cit. 2013-01-05]. Dostupné z URL: <http://www.nsnam.org/doxygen/index.html>

[19] *ns-3-users* [online]. 2013 [cit. 2013-20-04]. Dostupné z URL: <https://groups.google.com/d/topic/ns-3-users/SeQiyTntqa8/discussion>

Seznam zkratek

ABR	Area Border Router
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access With Collision Detection
DCE	Direct Code Execution
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DUAL	Diffusing Update Algorithm
EIGRP	Enhanced Interior Gateway Routing Protocol
GPL	General Public License
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate Systém
LSA	Link State Advertisement
LSDB	Link State Database
LSR	Link State Request
LSU	Link State Update
MAC	Media Access Control
NIC	Network Interface Card
OSPF	Open Shortest Path First
P2P	Point-to-Point
RIP	Routing Information Protocol
SNMP	Simple Network Management Protocol
SPF	Shortest Path First
Tcl	Tool Command Language
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol

Seznam obrázků

Seznam tabulek