VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Dávid Vaľuš

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

LABORATORNÍ ÚLOHY PRO VÝUKU SÍŤOVÝCH TECHNOLOGIÍ S POUŽITÍM RŮZNÝCH SIMULAČNÍCH NÁSTROJŮ

LABORATORY EXERCISES EXPLAINING NETWORK TECHNOLOGIES USING VARIOUS SIMULATION INSTRUMENTS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Dávid Vaľuš**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Zuzana Bečková**

SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Student: Bc. Dávid Vaľuš ***ID:*** 164430
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Laboratorní úlohy pro výuku síťových technologií s použitím různých simulačních nástrojů

POKYNY PRO VYPRACOVÁNÍ:

Zadání práce spočívá v návrhu dvou až tří nových laboratorních úloh včetně kompletních návodů vhodných pro studenty zahrnující výchozí scénář, doplňující úkoly a kontrolní otázky. Nastudujte problematiku síťových technologií a vyberte vhodné simulační prostředí, ve kterém budou laboratorní úlohy vytvořeny. Doporučené je prostředí NS3. Obsah úloh zaměřte na okruhy: rozbor aplikačních protokolů (FTP, HTTP, SMTP, DNS, popřípadě další - konkrétní ukázky komunikačních algoritmů), srovnání transportních protokolů TCP, UDP, SCTP, porovnání technologií Ethernet a WLAN, porovnání technik multiplexování, simulace technik ARQ atd. Časová náročnost každé úlohy musí být přibližně dvě hodiny.

DOPORUČENÁ LITERATURA:

[1] FOROUZAN, B. A. TCP/IP Protocol Suite. Fourth edition, Boston: McGraw-Hill Higher Education, 2010, 979 stran. ISBN 978-0-07-337604-2.

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

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: Ing. Zuzana Bečková
Konzultant:

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

ABSTRAKT

Táto diplomová práca sa zaoberá prípravou operačného systému, simulačneho prostredia a prípravou dvoch laboratórnych úloh na výuku sieťových technológií. V teoretickej časti sú popísané protokoly DHCP, FTP, TFTP, HTTP a služba NAT. Praktická časť sa zaoberá inštaláciou programov ns-3, GNS3, ich súčastí a ich nastavením na OS Ubuntu 18.10. Táto časť sa ďalej venuje samotnej príprave laboratórnych úloh s návodmi a kontrolnými otázkami s použitím týchto dvoch simulačných programov a HTTP servera v programovacom jazyku Python.

KĽÚČOVÉ SLOVÁ

GNS3, ns-3, server, klient, DHCP, NAT, FTP, TFTP, HTTP, laboratórna úloha, VMware, GNS3 VM, Cisco, Wireshark, Linux, Ubunutu, Python

ABSTRACT

This diploma thesis focuses on the preparation of an operating system, a simulation environment and the creation of two laboratory exercises explaining network technologies. In the theoretical part protocols DHCP, FTP, TFTP, HTTP, and NAT service are described. The practical part focuses on the installation of ns-3 and GNS3 programmes and their utilization and settings on OS Ubuntu 18.10. Moreover, it deals with the preparation of laboratory exercises created in the simulation programmes mentioned above together with tutorials, self-check questions and HTTP server in the Python programming language.

KEYWORDS

GNS3, ns-3, server, client, DHCP, NAT, FTP, TFTP, HTTP, laboratory exercises, VMware, GNS3 VM, Cisco, Wireshark, Linux, Ubunutu, Python

VAĽUŠ, Dávid. *Laboratorní úlohy pro výuku síťových technologií s použitím různých simulačních nástrojů*. Brno, 2019, 75 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Zuzana Bečková

VYHLÁSENIE

Vyhlasujem, že som svoju diplomovú prácu na tému „Laboratorní úlohy pro výuku síťových technologií s použitím různých simulačních nástrojů“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som sa poďakoval vedúcej diplomovej práce pani Ing. Zuzane Bečkovej, za odborné vedenie, konzultácie, trpezlivosť a predmetné návrhy k práci.

Brno ....

podpis autora

Obsah

Úvod **10**

1 Network Address Translation (NAT) **11**
1.1 Princíp fungovania 11
1.1.1 NAT so zdrojovým prekladom 12
1.1.2 NAT s cieľovým prekladom 14

2 Protokol DHCP **16**
2.1 Princíp fungovania 17

3 Protokol FTP **19**
3.1 Prenosové režimy FTP 19

4 Protokol TFTP **23**
4.1 Porovnanie FTP a TFTP 24

5 Protokol HTTP **26**
5.1 Vytváranie HTTP spojenia 26

6 Simulačné nástroje **28**
6.1 Network Simulator (ns-3) 28
6.1.1 Inštalácia ns-3 a potrebných nástrojov 29
6.2 Graphical Network Simulator (GNS3) 33
6.2.1 Porovnanie grafických simulačných nástrojov 33
6.2.2 Inštalácia a nastavenie GNS3 36
6.2.3 Quick emulator (QEMU) 36
6.2.4 GNS3 VM 37

7 Laboratórne úlohy **41**
7.1 Laboratorní úloha `DHCP a NAT` 41
7.1.1 Teoretický úvod 41
7.1.2 Topologie sítě 42
7.1.3 Hlavní cíle 42
7.1.4 Úvod k laboratorní úloze 42
7.1.5 Konfigurace směrovače R1 42
7.1.6 Konfigurace směrovače R2 44
7.1.7 Testovací scénáře 44
7.1.8 Samostatná práce 45
7.1.9 Kontrolní otázky 46

7.1.10 Kontrolní otázky - odpovědi 46
7.1.11 Konfigurace Linuxového DHCP serveru 47
7.1.12 Simulace v programu ns-3 50
7.1.13 Simulace v programu NetAnim 55
7.2 Laboratorní úloha `FTP, TFTP, HTTP` 56
7.2.1 Teoretický úvod 56
7.2.2 Topologie sítě 57
7.2.3 Hlavní cíle 57
7.2.4 Příprava programu GNS3 57
7.2.5 Konfigurace FTP serveru 58
7.2.6 Konfigurace FTP klienta 58
7.2.7 Konfigurace TFTP klienta 59
7.2.8 Samostatná práce 59
7.2.9 Kontrolní otázky 60
7.2.10 Kontrolní otázky - odpovědi 60
7.2.11 Metody protokolu HTTP 61

8 Záver **63**

Literatúra **64**

Zoznam symbolov, veličín a skratiek **66**

Zoznam príloh **67**

A HTTP server v jazyku Python **68**

B Obsah priloženého CD **75**

Zoznam obrázkov

1.1 Statický NAT 13
1.2 Preťažovaný NAT 14
3.1 FTP model 19
3.2 FTP v aktívnom móde 20
3.3 FTP v pasívnom móde 21
4.1 Štruktúra TFTP paketu 24
6.1 Terminálový výpis po spustení simulácie 31
6.2 Ukážka programu Ostinato v TCL 40
7.1 Topológia laboratórnej úlohy DHCP a NAT 42
7.2 Topologie laboratorní úlohy FTP, TFTP, HTTP 57

Zoznam tabuliek

2.1 Komunikácia DHCP 17
3.1 Typy FTP správ 22
4.1 Typy TFTP správ 23
4.2 Chybové kódy TFTP 23
4.3 Porovnanie FTP a TFTP 25
5.1 Metódy HTTP 27
6.1 Porovnanie GNS3 34
6.2 Porovnanie Cisco Packet Tracer 35
6.3 Porovnanie Cisco VIRL 35

Úvod

Protokol dynamického prideľovania adries (DHCP – Dynamic Host Configuration Protocol), hypertextový prenosový protokol (HTTP – Hypertext Transfer Protocol) a služba prekladu sieťových adries (NAT – Network Address Translation) sú v súčasnosti samozrejmosťou a mnoho ľudí si ani neuvedomuje, že sú s týmito technológiami v dennodennom styku a ako by to v dnešnej dobre internetu bez nich vyzeralo. Nemenej dôležité sú protokoly pre prenos súborov (FTP – File Transfer Protocol) a jednoduchý protokol pre prenos súborov (TFTP – Trivial File transfer Protocol). Vďaka týmto protokolom je náš každodenný život pohodlnejší a naša bezpečnosť a súkromie na internete na vyššej úrovni.

Táto práca sa v úvode teoretickej časti venuje práve týmto protokolom, ich štruktúre, možnostiam a ďalším detailom, ktoré s nimi úzko súvisia.

Ďalšia časť je venovaná simulačnému prostrediu, prostredníctvom ktorého by mal študent lepšie pochopiť spôsob ako tieto systémy komunikujú po sieti. V tejto časti je popísaná inštalácia simulačných nástrojov, ich súčastí a porovnanie s konkurenčnými nástrojmi.

Posledná kapitola sa venuje realizácií samotných laboratórnych úloh, kde bol dôraz kladený na spojenie teoretických a praktických znalosti. Úlohy sú štrukturované po častiach, obsahujú teoretický úvod, postup, samostatnú prácu a kontrolne otázky.

1 Network Address Translation (NAT)

NAT je služba, ktorá umožňuje preklad sieťových adries. NAT zvyčajne spája dve siete. Znamená to, že adresy privátnej siete, ktoré nie sú registrovanými adresami sa preložia na jedinečnú verejnú adresu, ktorá slúži na prístup k verejnej sieti, napríklad k Internetu. NAT môže byť nakonfigurovaný tak, že celá privátna sieť vystupuje pod jednou verejnou adresou. Táto možnosť poskytuje lepšiu bezpečnosť efektívnym skrývaním celej privátnej siete.[2]

NAT rieši problém s nedostatkom verejných adries. Umožňuje pripojiť už vytvorenú privátnu sieť k verejnej sieti, pomocou mapovania tisícok skrytých privátnych adries do niekoľkých verejne dostupných adries triedy C. NAT dáva administrátorom privátnych sietí možnosť ich rozširovania v rozsahu triedy A. Výhodou NAT je, že môže byť nakonfigurovaný na hraničných smerovačoch bez zásahu do ďalších častí siete.[2]

Priraďovaním rozsahu vnútorných adries jednej verejnej adrese, môžeme rozložiť záťaž. Ďalšou možnosťou je tzv. dvojnásobný NAT, ktorý slúži na spájanie privátnych sietí, ktorých rozsah sa prekrýva.[2]

1.1 Princíp fungovania

Smerovač, na ktorom je NAT nakonfigurovaný, má aspoň jedno rozhranie pripojené do verejnej a jedno do privátnej siete. Tento smerovač je tiež nazývaný hraničným smerovačom. Keď paket opúšťa privátnu sieť, smerovač si privátnu adresu uloží do prekladovej tabuľky. Pri odpovedi si smerovač prekladá celosvetovo unikátnu adresu, ktorú pomocou prekladovej tabuľky preloží na privátnu adresu. V prípade, že je v sieti viac ako jeden hraničný smerovač, tak všetky musia mať rovnakú prekladovú tabuľku. Záznamy prekladovej tabuľky sa odstraňujú, ak nie sú po dlhšiu dobu používané. Ak NAT nemôže prideliť verejnú adresu, pretože ich má vyčerpané, paket zahodí. Následne pošle Internet Control Message Protocol (ICMP) správu o nedosiahnutí cieľa.[2][1]

Poznáme štyri druhy NAT adries:

- Vnútorná lokálna – adresa nakonfigurovaná na stanici v privátnej sieti.
- Vnútorná globálna – adresa viditeľná z verejnej siete.
- Vonkajšia lokálna – adresy staníc vonkajšej siete, v tvare ako sa javia vo vnútornej sieti.
- Vonkajšia globálna – adresy nakonfigurované na staniciach vonkajšej siete. Táto možnosť je voliteľná.

Ak paket putuje z privátnej siete do verejnej siete, zdrojová adresa sa preloží z vnútornej lokálnej na vnútornú globálnu. Cieľová adresa sa preloží z vonkajšej

lokálnej sa vonkajšiu globálnu.

Ak paket putuje z verejnej siete do privátnej siete, zdrojová adresa sa preloží z vonkajšej globálnej na vonkajšiu lokálnu. Cieľová adresa sa preloží z vnútornej globálnej na vnútornú lokálnu.[2]

NAT môžeme rozdeliť do dvoch kategórií:

1. NAT so zdrojovým prekladom.
    - Statický NAT.
    - Dynamický NAT.
2. NAT s preťažovaním vnútornej globálnej adresy.

1.1.1 NAT so zdrojovým prekladom

NAT so zdrojovým prekladom, nazývaný Source NAT (SNAT). Používa sa pre prístup klientov z privátnej siete do verejnej siete. Zdrojová adresa (prípadne port) je v tomto prípade preložená a udržaná v tajnosti. [1]

Statický NAT

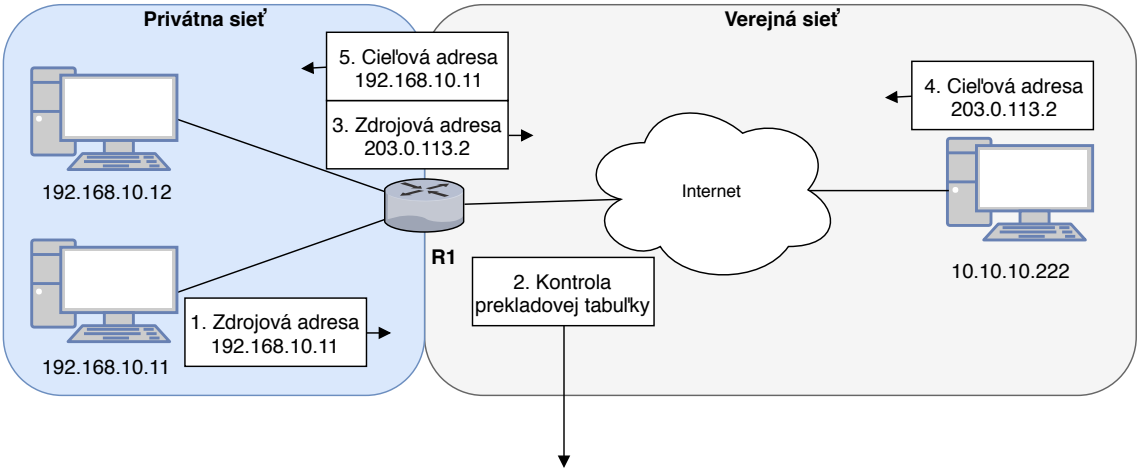
Je permanentné definovanie prevádzacej tabuľky administrátorom. Jedna privátna adresa, je pevne pridelená jednej verejnej adrese. Používa sa pokiaľ má byť nejaká vnútorná lokálna adresa dostupná pod stále rovnakou vnútornou globálnou adresou. Napríklad pre sprístupnenie servera z verejnej siete. V tomto prípade zostáva port nezmenený.

Dynamický NAT

Jedna privátna adresa je dynamicky pridelená jednej verejnej adrese z povoleného rozsahu. Záznamy prekladovej tabuľky sa môžu vytvárať len smerom z vnútornej siete do vonkajšej. V tomto prípadne je nutné, aby komunikáciu začal klient s lokálnou vnútornou adresou, inak hraničný smerovač nebude vedieť, komu má paket doručiť a zahodí ho. Pomocou dynamického NAT máme možnosť rozložiť záťaž a definovať vnútorné adresy, ktoré sa majú prekladať. Vytvorené záznamy sú len dočasné a adresy sa recyklujú.[2]

Popis fungovania služby NAT so zdrojovým prekladom: Viď. Obr. 1.1

1. Užívateľ v privátnej sieti s vnútornou lokálnou adresou 192.168.10.11 zahajuje komunikáciu s užívateľom vo verejnej sietí s vonkajšou lokálnou adresou 10.10.10.222.
2. Smerovač R1 dostane paket a porovná ho s jeho prekladovou tabuľkou. Ak smerovač nájde záznam v tabuľke, bude pokračovať k ďalšiemu kroku. Ak



Protokol	Vnútorná lokálna IP	Vnútorná globálna IP	Vonkajšia lokálna IP	Vonkajšia globálna IP
-	192.168.10.12	203.0.113.3	-	-
-	192.168.10.11	203.0.113.2	-	-

Obr. 1.1: Fungovanie statického NAT.[2]

smerovač nenájde záznam v tabuľke, tak vyhodnotí, že adresa musí byť preložená dynamicky. Zariadenie vyberie voľnú verejnú adresu a vytvorí záznam v prekladovej tabuľke.

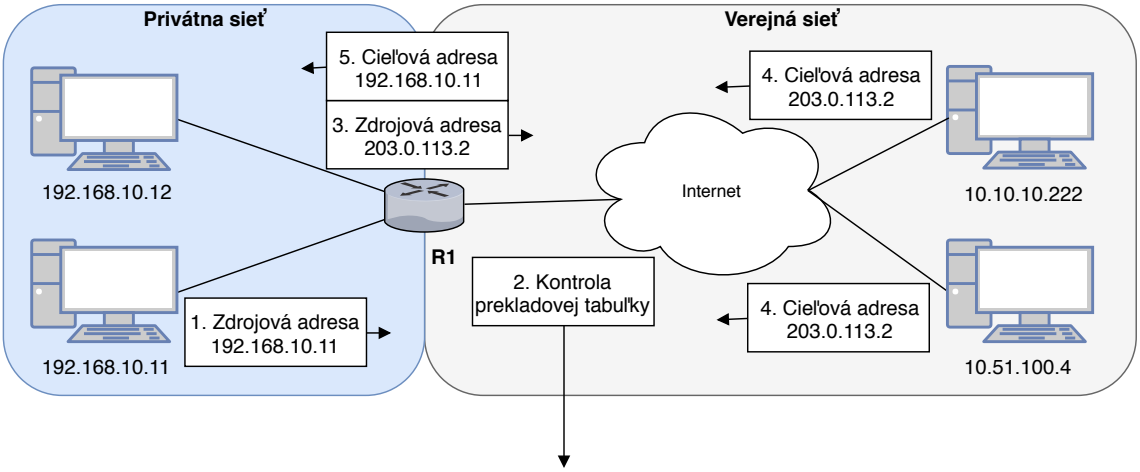
3. Smerovač nahradí vnútornú lokálnu IP adresu užívateľa vnútornou globálnou adresou.
4. Užívateľ vo verejnej sieti dostane paket a odpovedá užívateľovi na vnútornú globálnu adresu.
5. Keď smerovač dostane paket od užívateľa z verejnej siete, porovná jeho vonkajšiu lokálnu adresu so záznamom v prekladovej tabuľke. Následne paket prepošle na vnútornú lokálnu adresu užívateľovi v privátnej sieti.

Preťažovaný NAT

Viacero privátnych adries je v tomto prípade priradených jednej verejnej adrese. V rozšírenej prekladovej tabuľke je ku každej privátnej adrese priradený náhodný port. Pri odpovedi z vonkajšej siete si smerovač v prekladovej tabuľke vyhľadá port a paket prepošle na adresu k nemu priradenú. Táto metóda je tiež označovaná ako Port Address Translation (PAT).[2]

Popis fungovania služby preťažovaný NAT: Viď. Obr. 1.2

1. Užívateľ v privátnej sieti s vnútornou lokálnou adresou 192.168.10.11 zahajuje komunikáciu s užívateľom vo verejnej sieti s vonkajšou lokálnou adresou 10.10.10.222.
2. Smerovač R1 dostane paket a porovná ho s jeho prekladovou tabuľkou. Ak



Protokol	Vnútorná lokálna IP	Vnútorná globálna IP	Vonkajšia lokálna IP	Vonkajšia globálna IP
TCP	192.168.10.12:1723	203.0.113.3:1723	10.51.100.4:23	10.51.100.4:23
TCP	192.168.10.11:1024	203.0.113.2:1024	10.10.10.222:23	10.10.10.222:23

Obr. 1.2: Fungovanie preťažovaného NAT.[2]

smerovač nenájde záznam v tabuľke, tak vnútornú lokálnu adresu prevedie na vnútornú globálnu adresu z povoleného rozsahu. Ak je povolené preťažovanie a smerovač dostane ďalší paket, znovu použije tú istú vnútornú globálnu adresu. Zároveň si urobí ďalší záznam do prekladacej tabuľky s iným, náhodne vybraným číslom portu.

3. Smerovač nahradí vnútornú lokálnu IP adresu užívateľa vnútornou globálnou adresou a paket prepošle na verejnú lokálnu adresu druhého užívateľa.
4. Užívateľ vo verejnej sieti dostane paket a odpovedá užívateľovi na vnútornú globálnu adresu.
5. Keď smerovač dostane paket od užívateľa z verejnej siete, porovná jeho vonkajšiu lokálnu adresu so záznamom v prekladovej tabuľke. Ak nájde zhodný protokol, číslo portu a vonkajšiu lokálnu adresu, tak paket prepošle na vnútornú lokálnu adresu užívateľovi v privátnej sieti.

1.1.2 NAT s cieľovým prekladom

Nazývaný tiež Destination NAT (DNAT). Používa sa na sprístupnenie viacerých zariadení privátnej siete vo verejnej sieti pod jednou verejnou adresou. DNAT ponúka dve možnosti prekladu adries.[1]

Preposielanie portu

V angličtine nazývaný Port Forwarding. Hraničný smerovač prekladá paket prichádzajúci z verejnej siete do privátnej siete na základe čísla portu. Využíva sa pokiaľ

chceme pod jednou verejnou IP adresou sprístupniť viacero služieb. Napríklad paket, ktorý príde na hraničný smerovač s adresou 14.15.16.17:80 je na základe portu preložený na inú privátnu adresu s rovnakým portom.

Preklad portu

Nazývaný Port Translation. Táto možnosť je podobná predošlej možnosti s tým rozdielom, že s adresou sa prekladá rovnako aj číslo portu.

2 Protokol DHCP

Dynamic Host Configuration Protocol (DHCP) je aplikačný protokol TCP/IP (Transmission Control Protocol/Internet Protocol), umožňujúci zariadeniu vyžiadať si a získať voľnú IP adresu od DHCP servera. [1]

DHCP server môže byť nakonfigurovaný buď na smerovači, alebo na serveri (Linux server, Windows server). DHCP server vykonáva automatické pridelenie IP adresy, masky siete, implicitnej brány, primárneho a sekundárneho DNS (Domain Name System) servera klientom. DHCP klient si tieto parametre musí vyžiadať od DHCP servera a ten mu ich na časovo obmedzenú dobu pridelí. Server udržuje informácie o tom, ako dlho má klient adresu požičanú.[3]

DHCP server môže klientovi prideliť vždy tú istú IP adresu, pokiaľ je klient v zozname, ktorý obsahuje súhrn MAC adries a k ním pevne nedefinované IP adresy. Tento proces sa nazýva statická alokácia.[4]

V prípade, že správca siete vymedzí rozsah adries, ktoré budú prideľované klientom. Časové obmedzenie prenájmu IP adresy umožní DHCP serveru už nepoužívané adresy prideľovať iným staniciam. Tento proces sa nazýva dynamická alokácia.[1]

DHCP adresné rozsahy sú uložené v NVRAM (non-volatione Random Access memory). Staticky alokované adresy sú rovnako uložené v NVRAM pamäti. Dynamicky pridelené adresy sú uložené na vzdialenom zariadení, nazývanom databázový agent. Databázový agent môže byť klasický FTP server, ktorý ukladá DHCP databázu. DHCP databáza je textový súbor pre jednoduchú údržbu. Databázových agentov môže byť niekoľko a môžeme voliť interval ich vzájomnej synchronizácie.[3]

Databáza servera DHCP má stromové usporiadanie. Koreňom stromu je rozsah adries pre základné siete, konármi sú rozsahy podsietí a listy sú manuálne väzby pre klientov. Podsiete dedia parametre sietí a klienti dedia parametre podsietí. Preto by bežné parametre (napríklad názov domény) mali byť nakonfigurované na najvyššej (sieťovej alebo podsieťovej) úrovni stromu. Zdedené parametre je možné prepísať. Ak je napríklad parameter definovaný v základnej sieti aj v podsieti, používa sa definícia podsiete. Informácie o prenájmoch adries sa nededia. Ak pre IP adresu nie je určená doba prenájmu, server DHCP štandardne priradí adresu na jeden deň.[3]

Protokol definuje tiež DHCP relay agenta. Používa sa v situácii, keď exitujú dve, alebo viac sietí oddelených smerovačom a len jedna obsahuje DHCP server. V takom prípadne je nutné na smerovači nastaviť, aby všesmerové DHCP pakety preposielal DHCP serveru. Agent k preposielanému paketu pridá sieťovú adresu a masku, aby bol DHCP server schopný rozpoznať, z ktorého adresného rozsahu ma prideliť adresu klientovi.[3][4]

Komunikácia prebieha na portoch:

- 68 – Klient,

- 67 – Server.

2.1 Princíp fungovania

DHCP protokol používa štyry základné správy na vyjednanie konfigurácie medzi klientom a DHCP serverom. V tabuľke 2.1 je súhrná konverzácia medzi klientom a DHCP serverom:[4]

Zdroj. MAC	Cieľová MAC	Zdroj. IP	Cieľová IP	Popis paketu
Klient	FF:FF:FF:FF:FF:FF	0.0.0.0	255.255.255.255	DHCP Discover
Server	FF:FF:FF:FF:FF:FF	Server	255.255.255.255	DHCP Offer
Klient	FF:FF:FF:FF:FF:FF	0.0.0.0	255.255.255.255	DHCP Request
Server	FF:FF:FF:FF:FF:FF	Server	255.255.255.255	DHCP ACK

Tab. 2.1: Komunikácia DHCP klienta a servera.

1. **DHCPDISCOVER** – Po pripojení do siete vyšle klient tzv. DHCPDISCOVER paket, ktorý je doručený na všetky zariadenia v sieti.
2. **DHCPOFFER** – Server odpovie paketom DHCPOFFER s ponukou konfigurácie. Server použije svoju adresu ako zdrojovú a paket pošle všesmerovým vysielaním. Server identifikuje klienta na základe MAC (Media Access Control) z predošlej správy.
3. **DHCPREQUEST** – Klient si teoreticky z niekoľkých možností jednu vyberie a požiada o pridelenie konfigurácie. Jeho zdrojová adresa je stále 0.0.0.0, pretože zatiaľ nedostal potvrdenie od servera, že adresu môže používať. Cieľová adresa je stále všesmerová, pretože odpovedať môže viac ako jeden server.
4. **DHCPACK** – Server mu pridelenie konfigurácie potvrdí paketom. V správe sú aj informácie o dobe, kedy je klientovi adresa zapožičaná.[1]

Časť správy DHCPACK s konfiguráciou pridelenou klientovi:[4]

```
DHCP: DHCP Message Type      = DHCP ACK
DHCP: Renewal Time Value (T1) = 8 Days,  0:00:00
DHCP: Rebinding Time Value (T2) = 14 Days,  0:00:00
DHCP: IP Address Lease Time  = 16 Days,  0:00:00
DHCP: Server Identifier      = 157.54.48.151
DHCP: Subnet Mask            = 255.255.240.0
DHCP: Router                 = 157.54.48.1
DHCP: NetBIOS Name Service   = 157.54.16.154
DHCP: NetBIOS Node Type      = (Length: 1) 04
DHCP: End of this option field
```

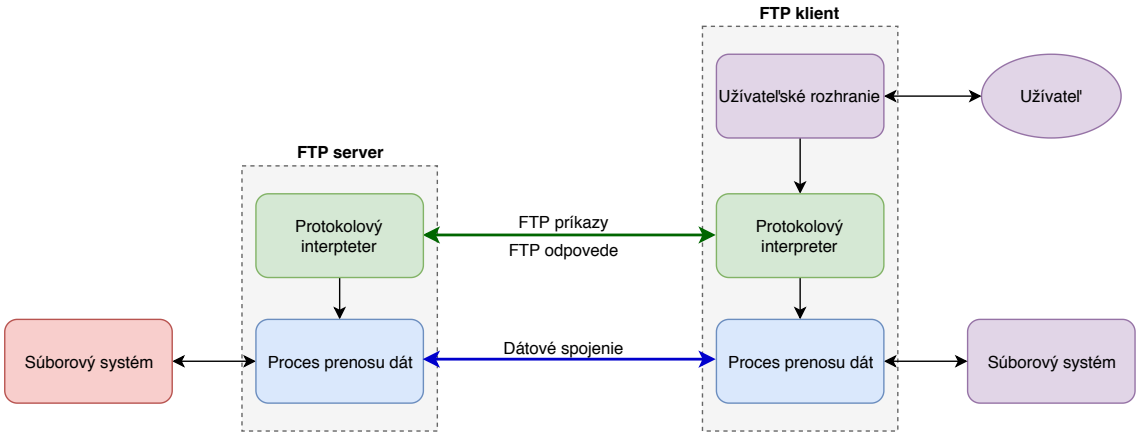
Po doručení paketu DHCPACK môže klient konfiguráciu používať. Klient musí pred uplynutím doby zapožičania konfigurácie znovu požiadať o pridelenie. Pokiaľ doba uplynie bez toho, aby dostal nové potvrdenie, musí prestať túto konfiguráciu používať.[3]

Pokiaľ dôjde napríklad k reštartovaniu klientského počítača, môže klient znovu poslať serveru DHCPREQUEST paket s jeho pôvodnou adresou. V prípade, že je adresa stále voľná, server odpovie klientovi správou DHCPACK. V prípade, že požadovaná adresa už voľná nie je, server odošle paket DHCPNAK.[1][4]

Klient posiela DHCPRELEASE paket o uvoľnení jemu priradenej konfigurácie.[1]

3 Protokol FTP

FTP (File Transfer Protocol), v preklade protokol pre prenos súborov. Využíva transportný protokol Transmission Control Protocol (TCP), ktorý garantuje spoľahlivé doručovanie v správnom poradí. FTP využíva TCP porty 20 a 21. Port 20 slúži k prenosu dát a port 21 k riadeniu komunikácie pomocou FTP správ. viď. Obr. 3.1 Prenos dát môže byť binárny, alebo textový v ACII znakoch. [7]



Obr. 3.1: FTP klient server model. [7]

3.1 Prenosové režimy FTP

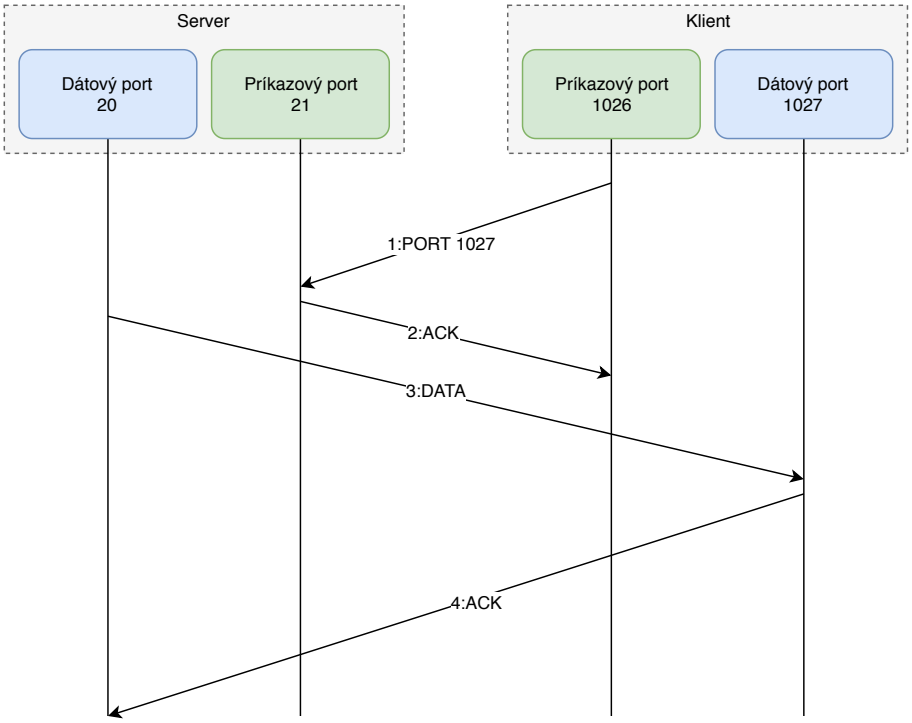
FTP podporuje dva prenosové režimy, aktívny a pasívny. Tieto režimy sa líšia v tom, ktorá stana inicializuje dátové spojenie.

Popis fungovania FTP v aktívnom režime. Viď. Obr. 3.2

1. Klient na príkazovom porte kontaktuje server na jeho príkazový port 21 a posiela správu PORT X, kde X je port na dátovú komunikáciu na strane klienta. V našom prípade `PORT 1027`.
2. Server posiela správu `ACK` späť na klientov príkazový port.
3. Server inicializuje spojenie na jeho dátovom porte s dátovým portom klienta, ktorý bol špecifikovaný v prvej správe.
4. Klient posiela potvrdzovaciu `ACK` správu. [8]

Popis fungovania FTP v pasívnom režime. Viď. Obr. 3.3

1. Klient na príkazovom porte kontaktuje server na jeho príkazový port 21 a posiela správu `PASV`.
2. Server odpovedá správou PORT X, čo je jeho dátový port. V našom prípade `PORT 2024`.



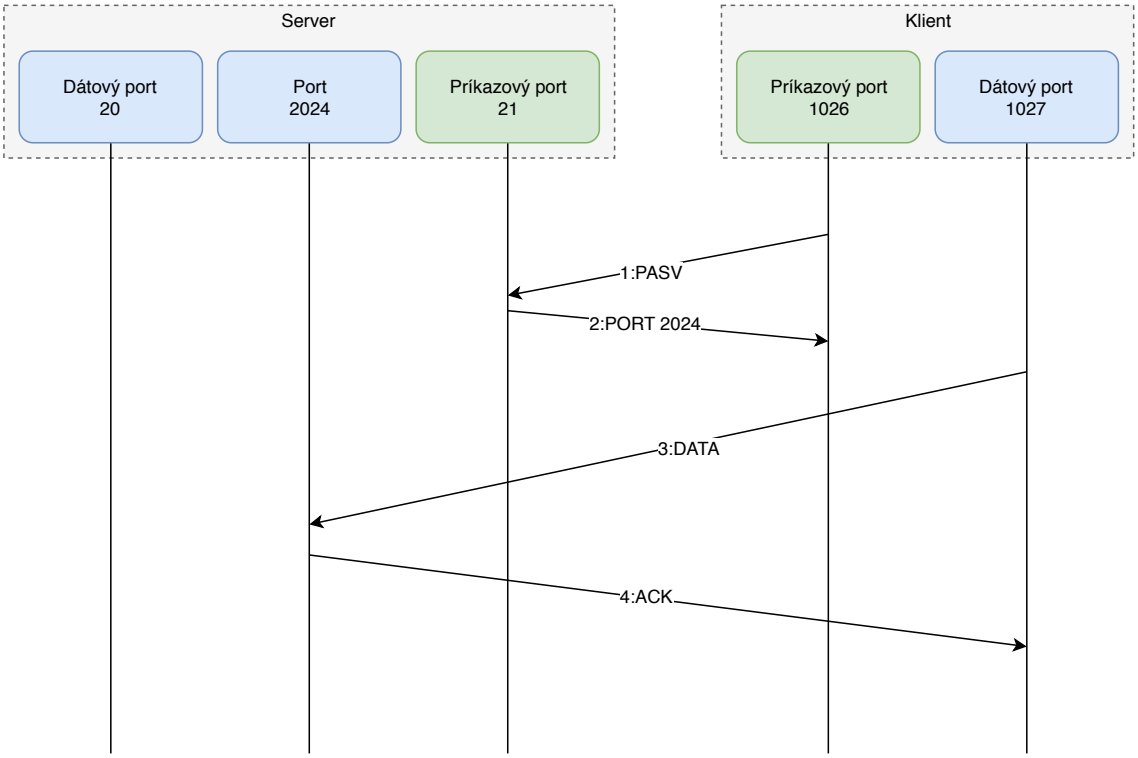
Obr. 3.2: FTP v aktívnom režime. [8]

3. Klient inicializuje dátové spojenie na jeho porte s portom 2024, ktorý dostal v druhej správe od servera.
4. Server posiela potvrdzovaciu `ACK` správu. [8]

Každý z týchto režimov ma svoje výhody a nevýhody.

- **Aktívny**- Problém aktívneho režimu je, že klient neinicializuje dátové spojenie. Pre firewall na strane klienta sa to javí, že sa zariadenie z verejnej siete snaží nadviazať spojenie s klientom v privátnej sieti a preto takéto spojenie zablokuje.
- **Pasívny**- Vzhľadom k tomu, že klient inicializuje všetky spojenia, tento režim je vhodný, ak ide komunikácia cez firewall, alebo NAT. No napriek tomu, že rieši mnoho problémov na strane klienta, spôsobuje niekoľko problémov na strane servera. Najväčší problém je, že server musí povoliť všetky pripojenia z dynamických portov. Druhým problémom môže byť, ak server nepodporuje pasívny mód. Vzhľadom k veľkej popularite HTTP(Hypertext Transfer Protocol), mnoho klientov používa webový prehliadač na prístup k FTP serveru a väčšina z nich používa len pasívny mód. [8]

FTP nebol navrhnutý ako bezpečný protokol a má mnoho bezpečnostných slabín. FTP nešifruje svoju komunikáciu, takže užívateľské mená, heslá aj dáta dokáže prečítať ktokoľvek, kto má k ním prístup. Riešením je pre prenos dát použiť FTPS (FTP Secured), SSH FTP (Secured Shell FTP) alebo vytvoriť šifrovanú komunikáciu



Obr. 3.3: FTP v pasívnom režime. [8]

medzi klientom a serverom, napríklad použitím VPN(Virtual Private Network).

Správy FTP servera sú vždy trojciferné a každé číslo má špeciálny význam. Prvé číslo vždy vyjadruje či je správa dobrá, zlá, alebo nekompletná. Viď. Tab. 3.1 Typy FTP správ: [7]

Tab. 3.1: Typy FTP správ. [5]

Číselný rozsah	**Význam**
1xx	Pozitívna predbežná odpoveď. Požadovaná aktivita sa zahajuje, očakáva ďalšiu odpoveď pred spracovaním následujúceho kódu.
2xx	Pozitívne doručenie. Požadovaná akcia bola úspešne dokončená.
3xx	Pozitívna prostredná odpoveď. Príkaz bol akceptovaný, ale požadovaná akcia je pozdržaná až do prijatia ďalších informácií.
4xx	Prechodná negatívna odpoveď. Príkaz nebol akceptovaný a požadovaná akcia nebola spracovaná, ale chybový stav je dočasný a akcia môže byť požadovaná znovu.
5xx	Permanentná negatívna odpoveď. Príkaz nebol akceptovaný a požadovaná akcia nebola spracovaná.
6xx	Chránená odpoveď.

4 Protokol TFTP

Trivial File Transfer Protocol je jednoduchý protokol na prenos dát. Obsahuje len niektoré základné funkcie protokolu FTP. Je určený pre prenos súborov v prípade, keď je protokol FTP nevhodný pre svoju komplikovanosť. Žiadosť o spojenie je vždy iniciovaná na cieľovom porte 69. [9]

Protokol TFTP funguje nad protokolom UDP, ktorý pre svoju komunikáciu nevytvára spojenie, takže protokol TFTP musí obsahovať vlastné riadenie spojenia. V jednom spojení sa vždy prenáša jediný súbor a pri komunikácii na sieti sa vždy prenáša len jeden paket a pred poslaním ďalšieho sa čaká na potvrdenie o doručení. Kvôli tomuto zjednodušeniu poskytuje tento protokol len malú prenosovú rýchlosť a zvyčajne sa používa na spúšťanie bezdiskových počítačov v sieti (BOOTP). [9]

TFTP podporuje tri prenosové módy:

- **netascii** – Text v ASCII znakoch.
- **octet** – 8b binárne dáta.
- **mail** – Pre zasielanie emailovej správy (Nemal by sa už používať). [9]

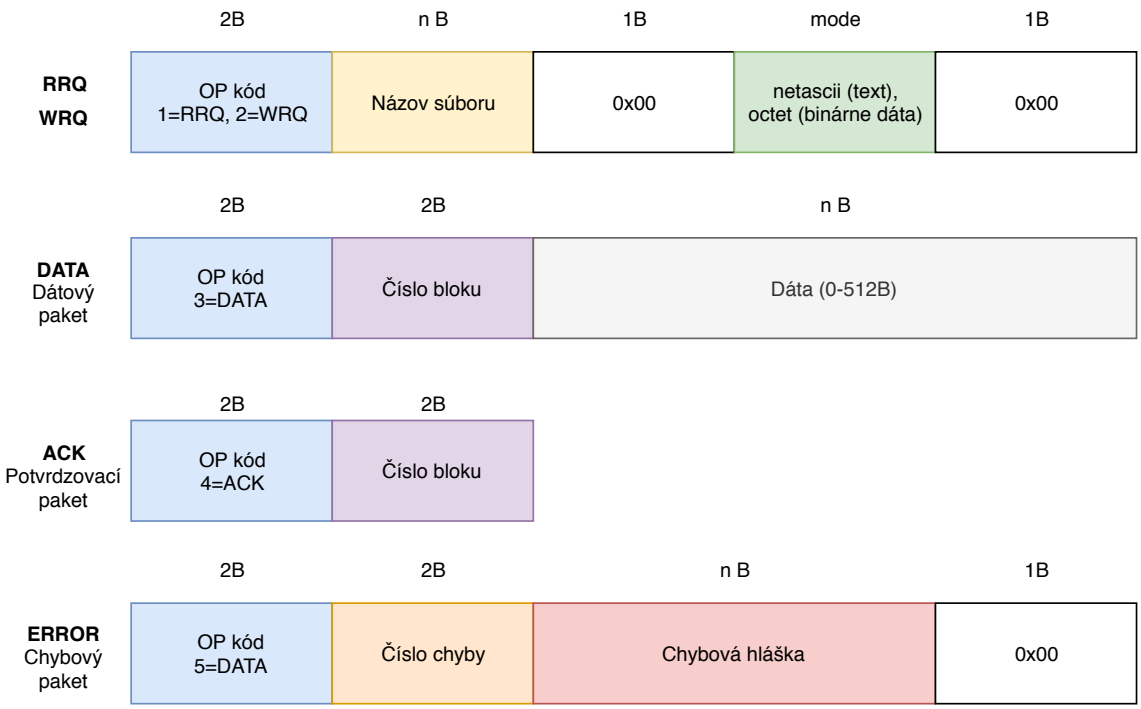
Typy TFTP paketov.

OP kód	Operácia	Popis
1	Read request (RRQ)	Požiadavka na čítanie
2	Write request (WRQ)	Požiadavka na zápis.
3	Dáta (DATA)	Blok dát.
4	Acknowledgment (ACK)	Potvrdzovací paket.
5	Error (ERROR)	Chybový paket

Tab. 4.1: Typy TFTP správ.

Tab. 4.2: Chybové kódy TFTP. [9]

Hodnota	**Význam**
0	Nie je definovaná. Popis v chybovej hláške.
1	Súbor nebol nájdený.
2	Chyba zdieľania.
3	Plný disk.
4	Ilegálna TFTP operácia.
5	Neznáme TID.
6	Súbor už existuje.
7	Neexistujúci užívateľ.



Obr. 4.1: Štruktúra TFTP paketov. [9]

4.1 Porovnanie FTP a TFTP

Protokol FTP je robustnejší, čo ale nie je vždy potrebné. V niektorých aplikáciách je výhodnejšie použiť protokol TFTP, ktorý ma značne menšie pamäťové nároky, čo umožňuje jeho využitie napríklad vo vstavaných systémoch, alebo pri vzdialenom zavádzaní systému na bezdiskové zariadenia. Podrobnejšie porovnanie týchto protokolov sa nachádza v tab. 4.3.

Hodnota	FTP	TFTP
Autentifikácia	Autentifikácia založená na prihlasovacom mene a hesle.	Nepodporuje autentifikáciu.
Spojenie	Používa spoľahlivé spojenie TCP. O chyby pri prenose sa stará vrstva TCP.	Používa nespoľahlivé spojenie UDP. O chyby pri prenose sa stará TFTP server.
Fungovanie	Prenos dát a kontrolných informácií je v réžií vrstvy TCP. Tá zaručuje maximálnu priepustnosť a riadenie prenosu pri chybách.	Používa jednoduchú potvrdzovaciu metódu. Pri komunikácii po sieti sa vždy prenáša len jeden paket a pred poslaním ďalšieho sa čaká na potvrdenie o doručení.
Dátový prenos	Je komplexnejší ako TFTP, takže potrebuje viac pamäte. Nie je vhodný pre bezdiskové zariadenia, kde sa musí zmestiť do obmedzenej EEPROM pamäte.	Je veľmi jednoduchý, pretože používa transportný protokol UDP. Vďaka tomu je vhodný na spúšťanie bezdiskových zariadení, kde sa celý protokol musí zmestiť do malej EEPROM pamäte.
Riadenie prenosu	Používa dve rozdielne TCP spojenia. Jedno na prenos používateľských dát, druhé na prenos kontrolných informácií.	Používa len jedno spojenie, kde kontroluje tok paketov v jednom smere a dáta posiela v opačnom smere pomocou rovnakých UDP soketov.

Tab. 4.3: Porovnanie FTP a TFTP.

5 Protokol HTTP

HTTP (Hypertext Transfer Protocol) je protokol aplikačnej vrstvy, ktorý slúži na prenos HTML (HyperText Markup Language) dokumentov v systéme www (World Wide Web). Je to primárna metóda prenosu informácií na webe. Jedná sa o klient-server architektúru, kde klient, webový prehliadač, otvára TCP spojenie na port servera 80, alebo port 443 pre HTTPS (HTTP Secure). Každá aktivita musí byť vyvolaná klientom. Úplná otázka a odpoveď musí mať špecifikovanú metódu URI(absolútna alebo relatívna cesta k súboru, alebo úplné URL dokumentu). [12]

Server podporujúci protokol 1.0 pošle odpoveď a spojenie ihneď uzavrie. Pred každou žiadosťou a odpoveďou musí byť zostavené spojenie TCP a tým sa zvyšuje réžia. Protokol 1.1 vytvára tzv. perzistentné spojenie a preto sa spojenia hneď neuzavrú, ale čakajú na ďalšie príkazy. To umožňuje klientovi poslať požiadavku a dostať odpoveď a následne tým istým spojením posielať ďalšie požiadavky a prijímať odpovede. Vďaka tomu sa znižuje réžia TCP. Tiež je možné poslať viacero požiadaviek pred obdržaním odpovede. Táto metóda sa nazýva zreťazené spracovanie (pipelining). Klient vďaka tomu môže pokračovať v požiadavkách na ostatné prvky HTML stránky a spojenie ukončiť sám. [12]

HTTP 2 je binárny protokol, nie textový. Kvôli tomu ho už nie je možné čítať a vytvárať manuálne. Táto zmena ale umožňuje, aby mohli byť implementované rôzne optimalizačné techniky. Protokol 2 narozdiel od verzie 1.1 podporuje paralelne požiadavky cez rovnaké spojenie. Keďže hlavičky sú často podobné, HTTP 2 ich dokáže komprimovať a tým sa odstraňuje duplicita a nároky na réžiu. V neposlednom rade umožňuje serveru uložiť údaje, napríklad obrázky do klientskej pamäte cache skôr, ako sú vyžiadané a to pomocou mechanizmu tlačenia serverom (server push). [13]

HTTP sa od ostatných protokolov založených na TCP líši tým, že spojenia sa ukončia potom, ako sa dokončí vykonanie požiadavky, alebo série požiadaviek. Tento návrh ho robí ideálny pre web, kde stránky často odkazujú na ďalšie stránky, na iných serveroch. To niekedy spôsobuje problémy webovým návrhárom, keďže chýbajúce perzistentné spojenie si vynucuje alternatívne prístupy udržania informácie o užívateľovi. Na to sa zvyčajne používajú súbory cookies. [12]

5.1 Vytváranie HTTP spojenia

HTTP server nasluchajúci na porte 80 čaká, kým klient pošle reťazec s požiadavkou `GET / HTTP/1.1`, ktorá žiada o zaslanie domovskej stránky servera. Táto požiadavka je nasledovaná sériou hlavičiek opisujúcich požiadavky klienta. Po prijatí požiadavky pošle server správu s odpoveďou `200 OK`, ktorá je nasledovaná hlavičkami spolu so samotnou správou. [12] Tento základný scernár sa môže modifikovať.

- **keep-alive**— Server nechá spojenie otvorené a klient ho môže využiť pre ďalšie požiadavky.
- **pipelining**— Klient nemusí čakať na odpoveď na predchádzajúcu požiadavku GET a hneď môže poslať ďalšie požiadavky. Server posiela odpovede v poradí v akom k nemu prišli.
- **Upgrade**— klient pomocou hlavičky Upgrade iniciuje prechod na iný protokol ako HTTP. Pokiaľ sa prechod podarí, komunikácia sa riadi pravidlami nového protokolu.
- **Connect**— Klient si cez proxy server vytvorí TCP tunel. Obvykle proxy povolí tunel len na TCP porte 443(HTTPS).

Tab. 5.1: Metódy HTTP. [12]

Príkaz	**Význam**
GET	Požiadavka na HTML stránku na danej URL.
HEAD	Požiadavka na hlavičku HTML stránky.
POST	Predávanie dát na server.
PUT	Nahranie súboru na server.
DELETE	Mazanie súboru zo servera.
OPTIONS	Zistenie možnosti spojenia, alebo informácií o HTML stránke bez toho, aby bola priamo vytvorená požiadavka.
TRACE	Sledovanie požiadavky predávanej na server.
CONNECT	Nadviazanie TCP spojenia cez HTTP proxy. (Nutnosť pre HTTPS)
PATCH	Požiadavka na čiastočnú aktualizáciu zdroja. Metóda PUT prepíše súbor, ale metóda PATCH ho modifikuje.

6 Simulačné nástroje

V tejto časti diplomovej práce sa budeme venovať príprave výukového prostredia. Táto príprava sa skladá z dvoch hlavných častí. Prvá časť sa venuje inštalácií, konfigurácií a skúšobnému scenáru simulačného prostredia ns-3. Ďalej sa venuje inštalácií a konfigurácií pomocných programov ako napríklad vývojové prostredie Eclipse a pomocných programov Wireshark a NetAnim.

V tejto kapitole je neskôr popísaná inštalácia a nastavenie programu GNS3 (Graphical Network Simulator) a jeho porovnanie s konkurenčnými programami.

6.1 Network Simulator (ns-3)

Simulátor ns-3 je simulátor siete zameraný na výskum a vzdelávacie účely. Jedná sa o softvér s otvoreným zdrojovým kódom a je stále vyvíjaný. Je primárne určený pre GNU/Linux rozhranie pod licenciou GNU GPLv2 (GNU's Not Unix General Public License v2). Nahrádza predošlú verziu ns-2, takže sa nejedná o jej rozšírenie. ns-3 je napísaný v jazyku C++ a python. Najnovšia verzia je ns3.29, ktorej dátum vydania bol 4.9.2018 a ktorú používame v tejto práci. Simulácie sa implementujú v jazyku C++ a niektoré časti simulácií môžu byť tiež implementované v jazyku python. [6]

Hlavným cieľom je vytvoriť pevné simulačné jadro, ktoré je dobré zdokumentované, jednoducho použiteľné s možnosťou dolaďovania a ktoré vyhovuje potrebám celého pracovného postupu od konfigurácie cez simuláciu až po analýzu dát. ns-3 tiež dokáže generovať súbory pcap, ktoré zaznamenávajú tok paketov v sieti a následne sa dajú analyzovať v programe Wireshark.[6]

Softvérové rozhranie ns-3 podporuje vývoj simulačných modelov, ktoré sú dostatočne realistické, aby mohli byť použité ako emulátor siete. Tieto modely môžu byť v reálnom čase prepojené so skutočnou sieťou a umožňujú použitie mnohých existujúcich implementácii protokolov. [6]

Simulátor ns-3 podporuje IP aj non-IP siete. Väčšina užívateľov sa zameriava na bezdrôtové IP simulácie ako napríklad Wi-Fi, WiMAX, alebo LTE pre prvú a druhú vrstvu modelu ISO/OSI a kombináciu statických a dynamických smerovacích protokolov ako napríklad Routing Information Protocol (RIP), Open Shortest Path first (OSPF) a ďalšie. [6]

ns-3 podporuje aj interakciu s reálnymi systémami, pomocou plánovača v reálnom čase, kde beží simulácia v slučke. V praxi to znamená, že užívateľ dokáže prijať dáta generované ns-3 na skutočnom zariadení v sieti a ns-3 môže slúžiť ako prepojovací článok na pridanie udalostí medzi dvoma zariadeniami. Ďalšou výhodou simulátora je použitie nemodifikovanej reálnej aplikácie z celého sieťového zásobníka jadra Linuxu.

Postup vytvárania simulácie by sa dal rozdeliť do šiestich základných krokov.

1. **Definícia topológie** – Vytvorenie základných zariadení.
2. **Definícia vzťahov** – Vytvorenie vzťahov a prepojení medzi zariadeniami.
3. **Konfigurácia zariadení** – Nastavenie zariadení a ich parametrov.
4. **Spustenie simulácie** – Simulácia generuje užívateľom zadané udalosti.
5. **Vyhodnocovanie výsledkov** – Po skončení simulácie sú dáta uložené s časovou značkou a detailami o udalosti.
6. **Grafická vizualizácia** – Dáta zozbierané počas simulácie si môžeme zobraziť graficky.

6.1.1 Inštalácia ns-3 a potrebných nástrojov

V tejto časti si popíšeme inštaláciu ns-3 a potrebných nástrojov na OS Ubuntu 18.10 64-bit. Počas inštalácie je nutné mať pripojenie k internetu. Inštaláciu ns-3 by som rozdelil do troch hlavných častí:

- Inštalácia ns-3.29,
- inštalácia a konfigurácia Eclipse 2018-09,
- inštalácia Wireshark a NetAnim.

Inštalácia ns-3.29

Inštalácia programu ns-3 prebieha cez terminál. Na začiatku je dôležité si aktualizovať systémové balíčky, takže v terminále zadáme následujúci príkaz:

```
$ sudo apt-get update && sudo apt-get dist-upgrade
```

Pre správne fungovanie ns-3 budeme potrebovať veľké množstvo knižníc. Knižnice sú predkompilované časti kódu, ktoré môžu byť použité v programe a ich úlohou je zjednodušiť prácu vývojárovi. Potrebné knižnice sa stiahnú a nainštalujú automaticky po zadaní následujúceho príkazu, ktorý následne potvrdíme:

```
$ sudo apt install build-essential autoconf automake
libxmu-dev python-pygoocanvas python-pygraphviz cvs
mercurial bzr git cmake p7zip-full python-matplotlib
python-tk python-dev python-kiwi python-gnome2
python-gnome2-desktop-dev python-rsvg qt4-dev-tools
qt4-qmake qt4-qmake qt4-default gnuplot-x11
```

Vo webovom prehliadači si otvoríme `https://www.nsnam.org/releases/` a vyberieme požadovanú verziu ns-3. V našom prípade sa jedná o ns3-29. Súbor ns-allinone-3.29.tar.bz2 si uložíme.

V termináli vojdeme do priečinka, kde sme si súbor uložili a pomocou príkazu rozbalíme.

```
$ tar jxvf ns-allinone-3.29.tar.bz2
```

Rozbalený súbor si následne presunieme do priečinka `/home$` a vojdeme do priečinka `/home/ns-allinone-3.29$`.

V tomto kroku je potrebné si program skompilovať. Znamená to, že zdrojový kód je konvertovaný do formátu, ktorý je spustiteľný. Tento proces je riadený kompilačným nástrojom a trvá v závislosti na hardvéri približne 90 minút. Kompiláciu spustíme následujúcim príkazom:

```
$ sudo ./build.py --enable-examples --enable-tests
```

Ak všetko prebehlo v poriadku, tak vo výpise terminálu budeme vidieť skompilované moduly.

Následne vojdeme do priečinka `/home/ns-allinone-3.29/ns-3.29$` a zadaním príkazu `ls` overíme, že sa v priečinku nachádza súbor `waf`. Súbor waf je automaticky kompilátor s otvoreným zdrojovým kódom napísaný v jazyku Python. Pomocou tohto súboru sme schopní kompilovať a spúšťať jednotlivé ns-3 simulácie. Súbor waf je nutné najprv nakonfigurovať následujúcim príkazom. V závisloti na hardvéri trvá tento krok približne 30 minút.

```
$ sudo ./waf --build-profile=debug --enable-examples
--enable-tests configure
```

Pokiaľ v termináli dostaneme výpis `'configure' finnished successfully`, všetko prebehlo v poriadku a môžeme pokračovať.

Aby sme overili funkčnosť programu ns-3, tak si spustíme skúšobný súbor. Skúšobý súbor spustíme príkazom:

```
$ sudo ./waf --run hello-simulator
```

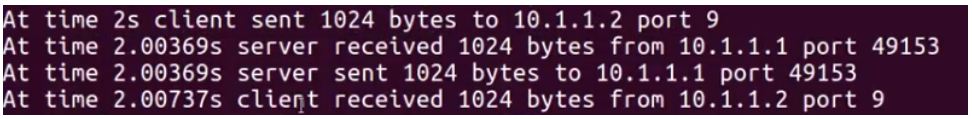
Ak všetko prebehlo v poriadku, dostaneme terminálový výpis `Hello Simulator`. Teraz si môžeme skúsiť spustiť reálnu simuláciu siete, ktorá je uložená v priečinku `/home/ns-allinone-3.29/ns-3.29/examples/tutorial$`. Jedná sa o jednoduchú komunikáciu medzi klientom a serverom, ktorý je dostupný na porte 9. V simulácii klient posiela serveru 1024B, ktoré server príjme a následne odošle späť klientovi. Tento tutoriál je nutné skopírovať do zložky `scratch`, z ktorej sa simulácie spúšťajú. To dosiahneme zadaním následujúceho príkazu v priečinku `/tutorial`.

```
$ cp first.* ../../scratch/
```

Teraz máme súbory `first.cc` a `first.py` skopírované v priečinku scratch a môžeme ich spustiť v priečinku `/home/ns-allinone-3.29/ns-3.29` príkazom:

```
$ ./waf --run scratch/first
```

Ak prebehlo všetko v poriadku, tak dostaneme terminálový výpis komunikácie klienta a servera, viď. Obr. 6.1



Obr. 6.1: Výpis terminálu po spustení simulácie first.cc.

Inštalácia a konfigurácia Eclipse

V tejto časti si popíšeme inštaláciu vývojového prostredia Eclipse 2018-09, ktorého dátum vydania bol 19.9.2018. Eclipse je IDE (integrated developement enviroment) s otvoreným zdrojovým kódom používaný v programovaní v jazyku Java. Obsahuje základný pracovný priestor s možnosťou rozšírenia pomocou pluginov, napríklad o C++, alebo PHP. Pre funkčnosť ns-3 vývojové prostredie Eclipse nepotrebujeme, ale výrazne nám to uľahčí a sprehľadní prácu pri kompilovaní zdrojového kódu projektu.

Pred samotnou inštaláciou Eclipse si potrebujeme nainštalovať Oracle Java 8 JDK (Java Development Kit). Tu si môžeme stiahnúť na odkaze `https://www.oracle.com/technetwork/java/javase/downloads/index.html`. Stiahnutý súbor skopírujeme do priečinka `/usr/local/java$`, kde ho rozbalíme. Následne si príkazom `$ sudo gedit /etc/profile` otvoríme textový súbor profile a na jeho koniec pridáme následujúce riadky a súbor uložíme.

```
JAVA_HOME=/usr/local/java/jdk1.8.0_191
JRE_HOME=$JAVA_HOME/jre
PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
export JAVA_HOME
export JRE_HOME
export PATH
```

Spustíme terminal a zadáme následujúce príkazy:

```
$ sudo update-alternatives --install "/usr/bin/java"
"java" "/usr/local/java/jdk1.7.0_45/jre/bin/java" 1
$ sudo update-alternatives --set java
/usr/local/java/jdk1.7.0_45/jre/bin/java
$ . /etc/profile
java -version
```

Po zadaní posledného príkazu by sme mali vidieť aktuálnu verziu java. Aby sa aplikovali zmeny, je nutné reštartovať operačný systém.

Teraz prejdeme k samotnej inštalácii Eclipse. Eclipse 2018-09 si stiahneme pomocou webového prehliadača na adrese `https://www.eclipse.org/downloads/`. Stiahnutý súbor si, pomocou grafcikého rozhrania rozbalíme a skopírujeme do priečinka `/home$`. Eclipse otvoríme dvojklikom na súbor eclipse.

V programe Eclipse si potrebujeme nainštalovať plugin Mercurial. Jendá sa o nástroj s otvoreným zdrojovým kódom, ktorý slúži na efektívne spravovanie projektov. Kliknutím na `"Help-> Install new Software..."` si zvolíme Mercurial a potvrdíme jeho inštaláciu. Po vyzvaní reštartujeme Eclipse.

Eclipse je teraz pre správne fungovanie potrebné nastaviť. Na začiatok si vytvoríme nový C++ projekt. Pravým tlačidlom klikneme na náš novovytvorený projekt a následne na `"Team-> Share project"`, otvorí sa nám okno kde zvolíme Mercurial a potvrdíme tlačidlom Finnish. Pravým tlačidlom znovu klikneme na náš projekt a následne na `Properties`. Otvorí sa nám nové okno kde zvolíme `C/C++ Build`. V záložke `Build Settings` zvolíme `Build command`, zadefinujeme cestu k `waf` súboru a následne nastavíme cestu `Build directory` k súboru `build`.

V záložke `Behavior` vložíme do poľa `Build (Incremental build)` slovo `build` a potvrdíme tlačidlom `OK`.

Ako posledný krok si nakonfigurujeme debugger. Z hornej lišty zvolíme `"Run-> Debug Configurations..."` kde zadefinujeme cestu k projektu a k spustiteľnému súboru. V záložke `Enviroment` klikneme na `New` a ako meno zvolíme `LD_LIBRARY_PATH` a hodnotu vyberieme cestu k priečinku `scratch`.

Inštalácia Wireshark a NetAnim

Na zobrazovanie súborov pcap, ktoré zachytávajú tok paketov s časovou hlavičkou budeme v tejto práci používať program Wireshark. Program stiahneme a nainštalujeme pomocou nasledujúcich príkazov:

```
$ sudo add-apt-repository ppa:wireshark-dev/stable
$ sudo apt-get install wireshark
```

Program NetAnim slúži na grafické zobrazenie simulovaného scenára. Ako zdrojový súbor NetAnim používa súbor XML, ktorý je vygenerovaný počas simulácie a obsahuje časovú známku a informácie o ceste rámcov sieťou. Inštalácia programu NetAnim je rozdelená do následujúcich bodov:

1. V termináli vojdeme do priečinka `NetAnim`, ktorý je umiestnený v priečinku `ns-allinone-3-29`.
2. Zadáme príkaz `make clean`, argument clean zmaže súbory z predošlej kompilácie. Ďalej zadáme príkaz `qmake NetAnim.pro` ktorý vytvorí súbor Makefile.
3. Príkazom `make` skompilujeme súbor Makefile.
4. NetAnim spustíme príkazom `./NetAnim`.

6.2 Graphical Network Simulator (GNS3)

GNS3 je grafický simulátor siete s otvoreným zdrojovým kódom. Slúži na emuláciu, simuláciu, konfiguráciu, testovanie a vyhľadávanie porúch virtuálnych a reálnych sietí. GNS3 umožňuje spustiť malú sieť obsahujúcu zopár zariadení, ktorá beží len na lokálnom počítači a zároveň obrovskú sieť, ktorá má zariadenia na niekoľkých serveroch alebo na cloude. GNS3 je v súčasnej dobe vyvíjaný a podporovaný. [5]

Prostredie GNS3 je v porovnaní s ns-3 viac užívateľsky prívetivé práve kvôli grafickému rozhraniu. V hornej časti je lišta s nástrojmi, v ľavej je ponuka zariadení, ktoré sú rozdelené podľa funkcií a v pravej časti vidíme zariadenia v našej aktuálnej topológii, ich aktuálny stav a hardvérové vyťaženie.

Pôvodne GNS3 umožňoval len emuláciu Cisco zariadení pomocou softvéru Dynamips. Dnes je už vyvinutých a podporovaných niekoľko nástrojov na emuláciu zariadení od rôzných výrobcov ako napríklad Cisco virtuálne prepínače, Windows klient, Linux server, Juniper, PIX firewallov a mnoho ďalších. Aktuálne GNS3 podporuje viac ako 20 rôznych výrobcov a ďalší stále pribúdajú. Vďaka tomu si môžeme skúsiť spoluprácu zariadení od rôznych výrobcov. Pomocou virtualizačných programov môžeme do našej simulovanej siete implementovať virtuálne zariadenia. [5]

GNS3 obsahuje dva hlavné softvérové komponenty:

- GNS3-all-in-one software (GUI),
- GNS3 virtual machine (VM).

GNS3-all-in-one je klientská časť GNS3 a tvorí ju grafické užívateľské rozhranie. Je možné ju nainštalovať na OS Windows, MAC aj Linux na lokálnom zariadení. Po vytvorení topológie použitím GNS3-all-in-one použité zariadenia potrebujú byť umiestnené a spustené na hosťovskom serveri. Máme tri možnosti hosťovského servera:

- **Lokálny GNS3 server** – Server beží na tom istom zariadení na ktorom beží GNS3-all-in-one. Ďalší proces ako napríklad Dynamips beží tiež na tom istom zariadení.
- **Lokálny GNS3 virtuálny stroj** – Server beží na virtuálnom lokálnom stroji (GNS3 VM) pomocou virtualizačného softvéru.
- **Vzdialený GNS3 virtuálny stroj** – Server beží na na vzdialenom virtuálnom stroji alebo na cloude.

6.2.1 Porovnanie grafických simulačných nástrojov

Existujú rôzne nástroje na študovanie a testovanie sietí. Niektoré umožňujú simulovanie a niektoré emulovanie sieťových zariadení. Simulovanie znamená, že na zariadení nebeží reálny operačný systém, simuluje sa len jeho funkcionalita. Emulovanie

znamená, že na virtuálnom zariadení môžeme spustiť kópiu Cisco IOS, stiahnutú z fyzického Cisco smerovača. Toto virtuálne zariadenie dokáže emulovať hardvér smerovača Cisco. Výber závisí na preferenciách užívateľa a každé riešenie ponúka nejaké výhody a nevýhody. Hlavné nástroje na simuláciu sietí sú napríklad:

- GNS3,
- Cisco Packet Tracer,
- Cisco VIRL (Virtual Internet Routing Lab),
- Fyzické zariadenia.

GNS3

GNS3 sa teší podpore veľkej komunity s viac ako 800 000 členmi. Je tiež podporovaný komerčnými subjektami ako AT&T, Google, VISA a v neposlednom rade NASA. Zdrojový kód GNS3 je dostupný na GitHub, vďaka čomu môžu členovia prispievať ku vývoju softvéru. Zhrnutie výhod a nevýhod v následujúcej tabuľke, viď. Tab. 6.1

Tab. 6.1: Porovnanie výhod a nevýhod GNS3. [5]

Výhody	**Nevýhody**
Voľne dostupný softvér s otvoreným zdrojový kódom, bez ročných poplatkov.	Obrazy Cisco zariadení si musí zaobstarať užívateľ sám.
Bez limitu na počet použitých zariadení.	Nie je samostatný balík, vyžaduje lokálnu inštaláciu softvéru (GUI).
Podporuje všetky Cisco VIRL obrazy (IOSv, IOSvL2, IOS-XRv, CSR1000v, NX-OSv, ASAv).	GNS3 môže byť ovplyvnený nastavením lokálneho počítača. Napr. nastavenie firewall, antivírusových programov a pod.
Popporuje viacerých výrobcov.	
Server môže byť spustený na lokálnom aj virtuálnom stroji.	

Packet Tracer

Cisco Packet Tracer je oficiálny produkt Cisco Academy, ktorý simuluje cisco siete. Nedokáže emulovať Cisco hardvér a nepodporuje obrazy Cisco zariadení a tiež iných výrobcov. Cisco Packet Tracer je zdarma, ale je potrebná registrácia do Cisco Akadémie. Výhody a nevýhody Cisco Packet Tracer vidíme v následujúcej tabuľke, viď. Tab. 6.2

Tab. 6.2: Porovnanie výhod a nevýhod Cisco Packet Tracer. [5]

Výhody	**Nevýhody**
Jednoduché nastavenie.	Proprietárny zdrojový kód.
Podporuje simuláciu Cisco smerovačov a prepínačov.	Simuluje len zariadenia Cisco.
Vhodný na štúdium CCNA.	Je možná len simulácia, nie emulácia zariadení.
Simuluje viacero zariadení a protokolov.	Nemôže byť prepojený s reálnymi zariadeniami.
Malé hardvérové nároky.	Nie je podporovaný na Mac OS.
Je zdarma po registrovaní sa do Cisco Akadémie.	

Cisco VIRL

Spoločnosť Cisco vytvorila ďalší oficiálne podporovaný simulačný nástroj nazývaný Cisco Virtual Internet Routing Lab. Je oveľa komplexnejši ako Cisco Packet Tracer. Produkt je podobný GNS3 a umožňuje užívateľovi simulovať reálne siete a zároveň sa učiť. Prehľad výhod a nevýhod Cisco VIRL je zhrnutý v tabuľke, viď. Tab. 6.3

Tab. 6.3: Porovnanie výhod a nevýhod Cisco VIRL. [5]

Výhody	**Nevýhody**
Podporuje cisco smerovače, prepínače, firewally a počítačové simulácie.	Cena osobnej licencie je $199,99 na rok a akademickej je $79,99.
Podporuje veľké množstvo protokolov a funkcií.	S osobnou a akademickou verziou je počet zariadení limitovaný na 20.
Vhodný na štúdium CCNA, CCNP a CCIE.	Vyžaduje virtualizačný softvér VMware Workstation alebo ESXi. Nepodporuje VirtualBox.
Podporuje najnovšie verzie Cisco IOS (15.X)	Vysoké harvérové nároky.
	Podporuje len Cisco zariadenia.
	Zložité nastavenie a konfigurácia.

6.2.2 Inštalácia a nastavenie GNS3

GNS3 stiahneme a nainštalujeme spustením následujúcich príkazov v terminále nášho systému Linux Ubuntu.

```
$ sudo add-apt-repository ppa:gns3/ppa
$ sudo apt-get update
$ sudo apt-get install gns3-gui
```

Používaný Cisco IOS potrebuje mať pre svoje spustenie na lokálnom serveri aj 32-bitové knižnice. Vzhľadom k tomu, že používame 64- bitovú verziu OS Ubuntu, knižnice musíme dodatočne nainštalovať. Naištalujeme tiež 32-bitovú verziu programu Dynamips, ktorý je v GNS3 implementovaný na simuláciu Cisco zariadení.

```
$ sudo apt install libc6-dev-i386
$ sudo apt install libelf-dev:i386 libpcap0.8-dev:i386
$ sudo apt install dynamips:i386
```

Pridanie Cisco IOS obrazu do programu GNS3.

- Spustíme GNS3 a vytvoríme si nový projekt.
- V hornej lište vyberieme Ëdit-> `Preferences...`".
- V novo otvorenom okne zvolíme `IOS routers`.
- Zvolíme `New` a následne `New Image`.
- Klikneme na Browse a zvolíme Cisco IOS obraz.
- Zadefinujeme minimálne množstvo pamäte RAM podľa Cisco dokumentácie.
- Klikneme na tlacidlo `IdlePC finder`. To slúži na výpočet tzv. IdlePC hodnoty z dôvodu, že Dynamips nevie efektívne určiť kedy Cisco IOS vykonáva efektívnu činnosť a kedy sa nachádza v procese nečinnej slučky. V dôsledku toho by sa procesor hostiteľského počítača plne vyťažil.

6.2.3 Quick emulator (QEMU)

QEMU je voľne dostupný emulátor s otvoreným zdrojovým kódom, ktorý podporuje virtualizáciu hardvéru. Je to hosťujúci virtuálny stroj, ktorý emuluje procesor zariadenia a pomocou binárnych prevodov umožňuje spustenie rôznych operačných systémov, ktoré potrebujú rôzny hardvér. Tiež môže byť použitý na spustenie KVM (Kernel-based Virtual Machine), ktoré dosahujú takmer svoje prirodzené rýchlosti pri použití hardvérového rozšírenia Intel VT-x, (Virtualization Technology) alebo AMD-V (AMD virtualization). Vďaka tomu je možné napríklad spustiť OS určený pre ARM (Advanced RISC Machine) procesor na stolovom počítači a dosiahnúť vysokú výkonnosť.

QEMU pracuje v 4 operačných módoch:

- **Emulácia používateľského módu** - V QEMU je spustený jeden Linux alebo macOS program, ktorý bol skompilovaný pre rozdielnu inštrukčnú sadu. Systémové volania sú ošetrené pre endianitu a 32/64 bitové nesúlady. Hlavnými cieľmi pre emuláciu používateľského režimu sú krížová kompilácia a krížové odladovanie.
- **Systémová emulácia** - V tomto móde QEMU emuluje celý počítačový systém vrátane periférií. Môže byť použitý na poskytnutie niekoľkých hosťujúcich zariadení na jednom počítači. QUEMU dokáže spúšťať viacero hosťujúcich OS ako napríklad Linux, Solaris, Windows, DOS a BSD a poskytuje emuláciu niekoľkých inštrukčných sád ako napríklad x86, MIPS, 32-bit ARMv7, PowerPC a ďalších.
- **KVM hosťovanie** - V tomto móde QEMU pracuje s nastavením a migráciou KVM obrazov. Je to stále vložené do emulácie hardvéru, ale spustenie hosťa je dané KVM na vyžiadanie QEMU.
- **Xen hosťovanie** - QEMU len emuluje hardvér, spustenie hosťujúceho OS je urobené Xen a je úplne skryté pred QEMU.

6.2.4 GNS3 VM

Pokiaľ chceme spúšťať komplexnejšie zariadenia v GNS3, musíme použiť GNS3 VM (Virtual Machine). Tento program je vyžadovaný pre spúšťanie zariadení založených na QEMU. GNS3 VM je navrhnutý pre prekonanie nedostatkov hostiteľského zariadenia. Beží na Ubuntu a vyžaduje spustenie pomocou virtualizačných nástrojov.

Požiadavky GNS3 VM sú procesor Intel Core i3/i5/i7/i9, 8/16GB pamäte RAM, ideálny OS je Windows 7 a VMWare workstation je taktiež odporúčaný. Na GNS3 VM je možné spustiť rôzne zariadenia ako napríklad Cisco IOSvL2, OS Ubuntu, Radius Server, Juniper a iné.

Zariadenia pre GNS3

GNS3 je možné rozšíriť rôznymi zariadeniami(appliances), ktoré boli vyvinuté komunitou. Všetky tieto zariadenia sú voľne dostupné, ale niektoré, napríklad Cisco zariadenia môžu vyžadovať originálny operačný systém. Väčšina z nich vyžaduje spustenie pomocou GNS3 VM a môžu podporovať aj grafické rozhranie pomocou aplikácie VNC viewer. Medzi tieto zariadenia patria napríklad Linux servery, firewally od rôznych výrobcov, OS pre stolové počítače, webové prehliadače a iné programy.

Zariadenia si môžeme stiahnuť priamo zo stránok GNS3 na odkaze: `https://gns3.com/marketplace/appliances`. Pomocou menu programu `"File-> Import Appliance"` vyberieme stiahnuté zariadenie a nainštalujeme.

Networkers' Toolkit

Toto zariadenie funguje na OS Ubuntu 18.04, ale nepodporuje grafické rozhranie. Obsahuje následovný serverový softvér pre manažment rôznych služieb:

- **www** (nginx)— je softvérový webový server s otvoreným zdrojovým kódom. Vďaka prepracovanej metóde rozloženia záťaže ho častejšie používajú väčšie firmy. Pracuje s protokolmi HTTP, HTTPS, SMTP, POP3, IMAP a SSL. Okrem rozloženia záťaže podporuje aj reverzné proxy. Zameriava sa predovšetkým na vysoký výkon a nízke nároky na pamäť. Dostupný je pre systémy na báze Unixu pod BSD, ale existujú aj varianty pre Solaris, macOS a MS Windows.
- **FTP** (Very Secure FTP Daemon)— Jedna sa o FTP server používaný v Unixových systémoch a patrí pod GPL licenciu. Podporuje IPv6, aktívny a pasívny mód prenosu, SSL, lokálnych a anonymných užívateľov, virtuálne IP adresy, osobitné nastavenie pre každého používateľa a dátové limity pre užívateľa, alebo IP adresu.
- **TFTP** (tftpd)— server pre TFTP protokol. Je rozsiahlo používaný pre podporu vzdialeného spúšťania bezdiskových zariadení.
- **Syslog** (rsyslog (Rocket-fast system for log processing))— Poskytuje vysokú výkonnosť, bezpečnosť a modulárny dizajn. Dokáže prenášať viac ako milión správ za sekundu.
- **DHCP** (isc-dhcp (Internet System Consorcium-DHCP deamon))— Podporuje IPv4 a IPv6, poskytuje vysokú spoľahlivosť a výkon. Jedna sa o softvér s otvoreným zdrojovým kódom a patrí pod MPL 2.0 licenciu.
- **SNMP** (Simple Network Management Protocol) (snmpd + snmptrapd)— jedna sa o SNMP agenta ktorý sa viaže na port a čaká na požiadavky od SNMP softvéru. Po prijatí žiadosti ju spracuje, zhromaždi požadované informácie, alebo vykoná požadovanú operáciu a informácie vráti.

Tiny Core Linux

Tiny Core Linux (TCL) je minimalistický Linuxový systém zameraný na poskytovanie základných služieb. Bol vytvorený aby mohol fungovať aj na vstavaných zariadeniach a aby bol schopný bežať primárne na RAM. TCL sa spúšta veľmi rýchlo a podporuje inštaláciu ďalších aplikácií a hardvéru. TCL existuje vo viacerých verziách [10]:

- **Tiny Core** — 16MB verzia obsahuje GUI a je pre zariadenia ktoré majú ethernetové pripojenie. Minimálne hardvérové nároky sú 46MB RAM, no odporúča sa 128MB RAM.
- **Core** — 11MB verzia, známa tiež ako Micro Core. Menšia verzia Tiny Core bez GUI, ale s možnosťou rozšírenia o GUI pomocou prídavných rozšírení.

Vyžaduje aspoň 28MB RAM.

- **dCore** — 12MB verzia s jadrom z Debian alebo Ubuntu, ktoré využívajú SCE balíčkový formát.
- **CorePure64** — rieši použiteľnosť verzie Core na architektúre x86_64.
- **Core Plus** — nejedná sa o distribúciu, ale o inštalačný súbor. Je to Tiny Core s ďalšími funkciami ako podpora bezdrôtových sietí a regionálnych znakov na klávesnici.
- **piCore** — verzia pre Raspberry Pi.

Ostinato

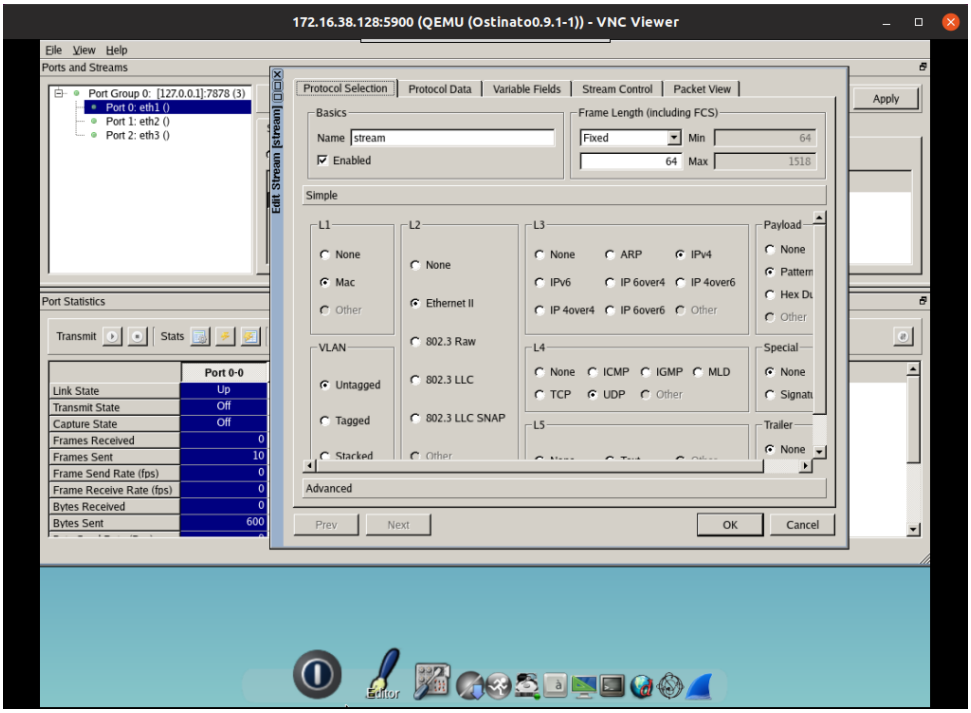
Je generátor paketov a sieťovej prevádzky s jednoduchým GUI a s otvoreným zdrojovým kódom pod GPL licenciou. Dokáže vytvárať a posielať pakety viacerými prúdmi s rozdielnými protokolmi a v inej frekvencii. Pre jednoduchú predstavu, funkčnosťou je to opak programu Wireshark. [11]

Ostinato nie je stavovo orientovaný a TCP spojenie nie je podporované. Dokáže vytvárať TCP pakety a posielať ich do siete, ale nie na nich reagovať. Ostinato nemôže byť použitý na generovanie falošnej prevádzky na web stránky. Podporuje protokoly Ethernet, VLAN, ARP, IPv4, IPv6, IP Tunnelling, TCP (bezstavový), UDP, ICMPv4, ICMPv6, IGMP, MLD, HTTP, SIP, RTSP a iné. Ostinato môže byť vďaka svojej jednoduchosti spustený aj na TCL. [11]

Ukážka programu Ostinato spusteného na TCL s možnosťami nastavenia detailov paketu, viď. Obr. 6.1

Webterm

Webterm je zariadenie na báze Debian 8 (jessie). Toto zariadenie obsahuje plne hodnotný webový prehliadač Mozilla Firefox a tiež nástroje ako ping, traceroute, ssh klienta, nástroje pre správu siete net-tools a iproute2 a iné.



Obr. 6.2: Ukážka programu Ostinato v TCL.

7 Laboratórne úlohy

V tejto kapitole sa nachádzajú laboratórne úlohy pre výuku sieťových technológií. Úlohy obsahujú teoretický úvod, samostatnú prácu a kontrolné otázky. Laboratórne úlohy sú navrhnuté na približne dve hodiny a sú písané v Českom jazyku z dôvodu, aby mohli byť bez väčších úprav použité na výuku.

7.1 Laboratorní úloha DHCP a NAT

Tato laboratorní úloha se zabývá konfigurací služby DHCP na směrovači Cisco a Linuxovém serveru. V úloze si nejprve vyzkoušíme konfiguraci služby NAT, poté hledání problémů v síti a statické směrování. Nakonec je úloha doplněna o simulaci v programu ns-3.

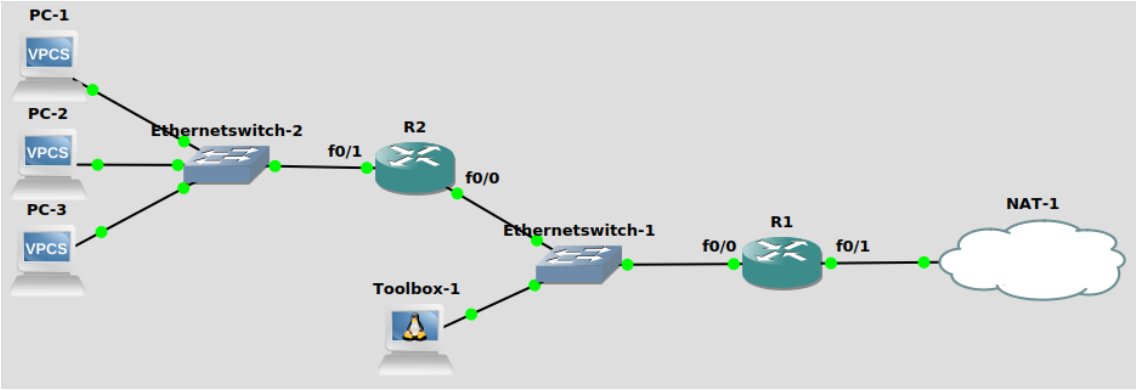
7.1.1 Teoretický úvod

Dynamic Host Configuration Protocol (DHCP) je aplikační protokol TCP/IP (Transmission Control Protocol/Internet Protocol) umožňující zařízení vyžádat si a získat volnou IP adresu od DHCP serveru. DHCP server může být nakonfigurován buď na směrovači nebo na serveru (Linux server, Windows server). Klient si tyto parametry musí vyžádat od DHCP serveru a ten mu je na časově omezenou dobu přidělí. Klient komunikuje na portu UDP 68, server na portu UDP 67. DHCP protokol používá čtyři základní zprávy na vyjednání konfigurace mezi klientem a serverem.

1. **DHCPDISCOVER** - Po připojení do sítě vyšle klient tzv. DHCPDISCOVER paket, který je doručen na všechna zařízení v síti.
2. **DHCPOFFER** - Server odpoví paketem DHCPOFFER s nabídkou konfigurace.
3. **DHCPREQUEST** - Klient si požádá o přidělení konfigurace.
4. **DHCPACK** - Server mu přidělení konfigurace potvrdí paketem. Ve zprávě jsou i informace o době, kdy je klientovi adresa zapůjčena.

Network address translation (NAT) je služba, která umožňuje překlad síťových adres, někdy bývá také nazývaná network masquerading (síťové maskování). NAT obvykle spojuje dvě sítě. NAT může být nakonfigurován tak, že celá privátní síť vystupuje pod jednou veřejnou adresou. Tato možnost poskytuje lepší bezpečnost efektivním skrýváním celé privátní sítě. Když paket opouští privátní síť, směrovač si privátní adresu uloží do překladové tabulky. Při odpovědi si směrovač překládá celosvětově unikátní adresu, kterou pomocí překladové tabulky přeloží na privátní adresu.

7.1.2 Topologie sítě



Obr. 7.1: Topologie laboratorní úlohy.

7.1.3 Hlavní cíle

1. Vytvoření statických cest.
2. Konfigurace NAT.
3. Konfigurace DHCP serveru.
4. Analýza výsledků pomocí Wireshark.

7.1.4 Úvod k laboratorní úloze

Z lišty v levé části obrazovky spustíme program GNS3 a otevřeme projekt `Uloha1-student`. Zeleným tlačítkem v horní nabídce programu GNS3 spustíme všechna zařízení. Po kliknutí se nám u všech zařízení otevřou terminálová okna. Směrovače Cisco se nespustí ihned, protože se na nich emuluje reálný Cisco IOS operační systém, jehož zavedení trvá přibližně minutu.

Směrovač R1 je již předem nakonfigurován pro připojení do veřejné sítě prostřednictvím rozhraní f0/1. PC-1 má staticky nakonfigurovanou adresu IP 192.168.10.11/24 a bránu. PC-2 a PC-3 mají nastavené dynamické přidělování adresy IP.

7.1.5 Konfigurace směrovače R1

(a) Připojení do veřejné sítě.
Připojení do veřejné sítě otestujeme příkazem:

```
R1# ping 8.8.8.8
```

V případě úspěšného připojení nakonfigurujeme DNS server.

```
R1# configure terminal
R1(config)# ip domain lookup
R1(config)# ip name-server 8.8.8.8
```

Ověříme správnost konfigurace:

```
R1# ping vutbr.cz
```

Nakonfigurujeme rozhraní f0/0, které je připojeno ke směrovači R2.

```
R1# configure terminal
R1(config)# interface f0/0
```

Možnost popisu je volitelná. V praxi je dobrým zvykem si psát k rozhraním popisky, protože to může značně usnadnit práci při hledání problému.

```
R1(config-if)# description Route to R2
```

Nakonfigurujeme IP adresu rozhraní a masku. Důležité je nezapomenout na příkaz `no shutdown`, který aktivuje rozhraní.

```
R1(config-if)# ip address 10.10.10.1 255.255.255.252
R1(config)# no shutdown
```

V této chvíli ještě nebude možné připojení ke směrovači R2, protože jeho rozhraní zatím není nakonfigurováno.

(b) Konfigurace NAT.
Nejprve si nakonfigurujeme rozhraní f0/1, které slouží k připojení do veřejné sítě.

```
R1# conf t
R1(config)# int f0/1
R1(config-if)# ip nat outside
```

Dále si nakonfigurujeme rozhraní f0/0 s parametrem `inside`.

```
R1(config-if)# int f0/0
R1(config-if)# ip nat inside
R1(config-if)# exi
```

Následujícími příkazy si nakonfigurujeme, že se všechny adresy privátní sítě budou překládat na jednu veřejnou adresu.

```
R1(config)# ip nat inside source list 1 interface
f0/1 overload
R1(config)# access-list 1 permit any
```

7.1.6 Konfigurace směrovače R2

(a) Konfigurace statické cesty ke směrovači R1.
Nakonfugurujeme rozhraní f0/0.

```
R2(config-if)# des Route to R1
R2(config-if)# ip add 10.10.10.2 255.255.255.252
R2(config-if)# no sh
```

(b) Konfigurace DHCP serveru.
Příkazem `service dhcp` spustíme DHCP službu směrovače R2. Tato služba je implicitně spuštěna, ale z bezpečnostních důvodů může být deaktivována.

```
R2# conf t
R2(config)# service dhcp
```

Dále si vytvoříme DHCP pool s názvem `studentpool1`, který si nakonfigurujeme. Nadefinujeme síť, ze které může DHCP server přidělovat adresy. Nastavíme výchozí bránu, která je adresa směrovače R2. Zvolíme rozsah adres, z nichž DHCP přidělovat nemá, a DNS server.

```
R2(config)# ip dhcp pool studentpool1
R2(dhcp-config)# network 192.168.10.0 /24
R2(dhcp-config)# default-router 192.168.10.1
R2(dhcp-config)# dns 8.8.8.8
R2(dhcp-config)# exi
R2(config)# ip dhcp excluded-address
192.168.10.1 192.168.10.11
```

7.1.7 Testovací scénáře

V tomto bodě je naše síť téměř hotová. Zkuste si otestovat následující scénáře. V případě, že některý z nich nebude fungovat, se zamyslete a pokuste se najít řešení v časovém limitu 30 minut.

Zadání.

1. Ping výchozí brány (192.168.10.1) z PC-1, který ma statickou IP (192.168.10.11).
2. Ping výchozí brány (192.168.10.1) z PC-2.
3. Ping PC-1 (192.168.10.11) z PC-2.
4. Ping R1 (10.10.10.1) z R2.
5. Ping 8.8.8.8 z R2.
6. Ping R1 (10.10.10.1) z PC-1.
7. Ping www.vutbr.cz z PC-2.

Řešení.

1. Výchozí brána není dostupná kvůli tomu, že rozhraní f0/1, které je připojeno k Eth.Switch 2, není nakonfigurováno. Na rozhraní nastavíme adresu 192.168.10.1 a masku 255.255.255.0, zároveň nezapomeneme aktivovat rozhraní příkazem `no sh`. Funkčnost ověříme příkazem `ping 192.168.10.1` na PC-1.
2. PC-2 se snažil získat konfiguraci dříve, než byl DHCP server aktivní. V terminálu PC-2 si příkazem `sh ip` můžeme zobrazit aktuální konfiguraci IP. Dále zadáme příkaz `set dump file`, který nám bude ukládat veškerý datový tok na PC-2 do souboru spustitelného programu Wireshark, který využijeme později. Následně příkazem `dhcp -r` (renew) vyžádáme novou adresu v síti. Funkčnost ověříme příkazem `ping 192.168.10.1` na PC-2.
3. Vzhledem k tomu, že obě tato zařízení již mají nakonfigurovány IP adresy, vzájemný ping bude možný. Funkčnost ověříme příkazem `ping 192.168.10.11` na PC-2.
4. Ping R1 z R2 bude možný, protože se jedná o přímo propojené sítě. Funkčnost ověříme příkazem `ping 10.10.10.1`.
5. Ping 8.8.8.8 z R2 nebude možný, protože nemáme nastavenou defaultní cestu. Směrovač R2 ve své směrovací tabulce nenalezne záznam o 8.8.8.8, a proto paket zahodí. Defaultní cestu nakonfigurujeme příkazem `ip route 0.0.0.0 0.0.0.0 10.10.10.1` v konfiguračním módu směrovače R2.
6. Ping R1 z PC-1 nebude možný. Směrovač R1 nezná síť 192.168.10.0/24. Pokud na jeho rozhraní přijde paket z této sítě, zahodí ho. Do směrovací tabulky směrovače R1 proto přidáme záznam o síti 192.168.10.0 dostupné přes rozhraní f0/0. Záznam přidáme příkazem `ip route 192.168.10.0 255.255.255.0 f0/0` v konfiguračním módu.

   V bodě 5 jsme použili při konfiguraci defaultní cesty jako výstupní rozhraní IP adresu. V příkladu v bodě 6 jsme u konfigurace statické cesty použili jako rozhraní port f0/0. Obě tyto možností jsou správné.
7. Ping www.vutbr.cz z PC-2 bude možný. Na DHCP serveru jsme při konfiguraci `ip dhcp pool` příkazem `dns 8.8.8.8` nastavili, aby server v paketu DHCPOFFER posílal i DNS server.

7.1.8 Samostatná práce

Zadání

1. Zjistěte, jaká je defaultní doba pronájmu adresy od DHCP serveru.
2. Nakonfigurujte DHCP server tak, aby byla doba pronájmu konfigurace 3 minuty.

3. Získejte konfiguraci pro PC-3 z DHCP serveru a ověřte dobu pronájmu konfigurace.
4. V programu Wireshark zobrazte komunikaci mezi klientem a serverem před vypršením doby pronájmu konfigurace.

Řešení

1. Dobu pronájmu adresy můžeme zjistit třemi způsoby. Na směrovači R2 příkazem `sh ip dhcp binding`. Na PC-2 příkazem `dhcp -d` (decode) v paketu ACK. Třetí možností je pak otevření souboru `/home/GNS3/projects/Uloha1-student/project-files/vpcs/PC-2/PC-2.pcap` a analýzou paketu `DHCP ACK`.
2. Dobu pronájmu (lease time) změníme na směrovači R2. V konfiguračním módu vstoupíme do námi vytvořeného poolu studentpool1. Tam nastavíme dobu pronájmu konfigurace příkazem `lease 0 0 3` (DD HH MM).
3. Na PC-3 zadáme příkaz `dhcp -r` následovaný příkazem `dhcp -d`, kde v paketu DHCPACK zjistíme dobu pronájmu dobu pronájmu.
4. Na PC-3 zadáme příkaz `set dump` file pro ukládání komunikace do souboru PCAP. Soubor *.pcap najdeme v `/home/GNS3/projects/Uloha1-student/project-files/vpcs/PC-3/PC-3.pcap`.
   Komunikaci můžeme zobrazit také kliknutím pravým tlačítkem na spojení PC-3 a Eth.Switch2, poté výběrem možnosti `Start capture` a `Start Wireshark`.

7.1.9 Kontrolní otázky

1. Liší se hodnoty času pronájmu konfigurace PC-2 na směrovači, ve Wireshark a v dekódování DHCP paketů na PC-2?
2. Proč se po zadání příkazu `ping` v některých případech ztratí první paket (.!!!!)?
3. Jaký vliv má změna doby má změna doby pronájmu konfigurace DHCP serveru na PC-2?
4. Jaké typy paketů si vyměňují DHCP klient a DHCP server před uplynutím doby pronájmu?
5. Jaký je rozdíl mezi Lease Time a Renewal Time a jaký je mezi nimi poměr?

7.1.10 Kontrolní otázky - odpovědi

1. Hodnoty se neliší, jsou stejné. Obě jsou generovány programem GNS3.
2. Zdrojové zařízení nezná MAC adresu cílového zařízení a snaží se ji zjistit. Všesměrovou zprávou posílá ARP Request paket, kde se podle cílové IP adresy poptává na MAC adresu cílového zařízení. Směrovač mu odpoví jedmosměrovým vysíláním s MAC adresou cílového zařízení. V této chvíli má zdrojové zařízení dostatek informací, aby vygenerovalo ping paket. Vzhledem k tomu,

že časovač byl zapnutý dříve než zařízení získalo MAC adresu cílového zařízení a poslalo první paket ping, obvykle 1s, zařízení považuje tento paket za ztracený. Při opakovaném příkazu ping se neztratí žádný paket neztratí(!!!!!).

3. Nemá žádný vliv. PC-2 má konfiguraci pronajatou na 24 hodin. PC-2 to ovlivní až po uplynutí času Renewal Time. Pokud PC-2 pošle DHCPREQUEST paket s žádostí o pronájem jeho aktuální konfigurace znovu, DHCP server mu odpoví zprávou DHCPACK, ve které mu potvrdí, že konfiguraci může používat. Tato zpráva bude také obsahovat už novou dobu pronájmu konfigurace.
4. Před uplynutím pronájmu si DHCP klient a Server vyměňují jen zprávy DHCPREQUEST a DHCPACK.
5. Renewal Time je polovina Lease Time. Po uplynutí Lease Time klient nesmí adresu dále používat. Z toho důvodu si o její používání žádá předem, a to po uplynutí doby Renewal Time.

7.1.11 Konfigurace Linuxového DHCP serveru

Zařízení Toolbox-1 je Linuxový server s OS Ubuntu, na kterém si nakonfigurujeme službu DHCP. Na směrovači R2 si nakonfigurujeme `dhcp relay` agenta, který bude přeposílat `DHCP` pakety serveru v jiné síti.

(a) Připojení serveru do sítě.
Mezi směrovači R1 a R2 je nakonfigurována síť 10.10.10.0/30. To znamená, že v síti mohou být připojeny maximálně 2 zařízení, protože první adresa sítě 10.10.10.0 je síťová a poslední adresa 10.10.10.4 je všesměrová. Na portu f0/0 směrovače R1 a R2 zadáme příkaz `ip add` se stejnou IP adresou, ale maskou /24, díky které můžeme do sítě připojit až 254 zařízení.

```
R1(config)#int f0/0
R1(config-if)#ip add 10.10.10.1 255.255.255.0

R2(config)#int f0/0
R2(config-if)#ip add 10.10.10.2 255.255.255.0
```

(b) Smazání poolu Studentpool1.
V konfiguračním módu směrovače R2 zadáme příkaz:

```
R2(config)#no ip dhcp pool studentpool1
```

Pozor! Služba `service dhcp` musí zůstat zapnuta. Části této služby je také dhcp relay agent, který by pak nefungoval správně.

(c) Konfigurace DHCP relay na směrovači R2.
Jelikož DHCP server bude v jiné síti než klienti, je třeba nastavit, aby směrovač R2 přeposílal DHCP zprávy z portu f0/1 serveru, který komunikuje na IP

adrese 10.10.10.3. Konfiguraci portu ověříme.

```
R2(config)#int f0/1
R1(config-if)#ip helper-address 10.10.10.3
R2(config-if)#do sh run int f0/1
```

(d) Konfigurace statické adresy pro DHCP server.
Ještě před tím, než začneme s konfigurací DHCP, je potřeba nastavit serveru statickou IP adresu. Příkazem `nano /etc/network/interfaces` si otevřeme konfigurační soubor, ve kterém je již předpřipravená konfigurace. Odkomentovat řádky od `auto eth0` až po `gateway 192.168.0.1` (včetně), serveru přiřadíme adresu `10.10.10.3` a nastavíme i další parametry sítě. Výsledný soubor by měl vypadat přibližně takto:

```
#
# This is a sample network config
# uncomment lines to configure the network


# Static config for eth0
auto eth0
iface eth0 inet static
    address 10.10.10.3
    netmask 255.255.255.0
    gateway 10.10.10.2
#   up echo nameserver 192.168.0.1 > /etc/resolv.conf

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```

(e) Připojení DHCP serveru do sítě.
Správnost konfigurace serveru a směrovačů ověříme příkazem `ping`.

```
root@Toolbox-1:~# ping 10.10.10.1
root@Toolbox-1:~# ping 10.10.10.2
```

(f) Konfigurace rozhraní DHCP serveru.
Pomocí příkazu `nano /etc/default/isc-dhcp-server` si otevřeme konfigurační soubor DHCP serveru. V tomto souboru je třeba definovat rozhraní, na kterém má DHCP server naslouchat - v našem případě `eth 0`.

```
INTERFACESv4="eth0"
INTERFACESv6=""
```

(g) Nastavení parametrů DHCP serveru.
Pro vytvoření podsítě slouží samostatný konfigurační soubor, který si otevřeme pomocí příkazu `nano /etc/dhcp/dhcpd.conf`. Na začátek souboru je třeba překopírovat předpřipravenou konfiguraci.

```
option domain-name-servers 8.8.8.8;
default-lease-time 3600;
max-lease-time 7200;
authoritative;

subnet 192.168.10.0 netmask 255.255.255.0 {
        option routers                  192.168.10.1;
        option subnet-mask              255.255.255.0;
        option domain-name-servers      8.8.8.8;
        range   192.168.10.2    192.168.10.9;
        range   192.168.10.12   192.168.10.200;
}
subnet 10.10.10.0 netmask 255.255.255.0 {
}
```

V této konfiguraci se vytvoří podsíť `192.168.10.0/24`. Výchozí brána se nastaví na adresu `192.168.10.1` a vytvoří se dva rozsahy přidělovaných adres `192.168.10.2 - 192.168.10.9` a `192.168.10.12 - 192.168.10.200`. Je třeba definovat také síť, ve které je server `(10.10.10.0)` i přesto, že z ní server nebude přidělovat adresy.

(h) Spuštění DHCP serveru.
Po konfiguraci serveru je třeba ho v programu GNS3 vypnout a znovu zapnout, aby se projevily všechny změny.
Po restartu si opět otevřeme konzoli serveru a zobrazíme si stav DHCP serveru. Občas se totiž stane, že se po startu systému sám nespustí.
Pokud server běží, vše je v pořádku a server je úspěšně nakonfigurován. V případě, že server neběží, na konzoli můžeme vidět následný výpis:

```
root@Toolbox-1:~# service isc-dhcp-server status
Status of ISC DHCPv4 server: dhcpd is not running.
```

V takovém případě je třeba server jednoduše spustit. Pokud se však při zobrazení stavu DHCP serveru zobrazí následující zpráva, je třeba odstranit soubor, který systému vadí, a až potom je možné server spustit.

```
root@Toolbox-1:~# service isc-dhcp-server start
Launching IPv4 server only.
 * Starting ISC DHCPv4 server dhcpd
```

```
 * dhcpd service already running
      (pid file /var/run/dhcpd.pid currenty exists)

root@Toolbox-1:~# service isc-dhcp-server status
Status of ISC DHCPv4 server:
      dhcpd is not running but /var/run/dhcpd.pid exists.

root@Toolbox-1:~# rm /var/run/dhcpd.pid

root@Toolbox-1:~# service isc-dhcp-server start
Launching IPv4 server only.
 * Starting ISC DHCPv4 server dhcpd              [ OK ]
```

7.1.12 Simulace v programu ns-3

1. Z lišty vlevo spustíme program Eclipse Luna.
2. Zvolíme Workspace `/home/student/eclipse-workspace` a klikneme na tlačítko launch `Launch`.
3. Z levého menu programu ns3 zvolíme `ns3/scratch/dhcp.cc`.

Stručně si popíšeme zdrojový kód dhcp.cc.

```
#include "ns3/core-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"
```

Knihovna ns3 je pro lepší orientaci logicky rozdělena na několik podmodulů. První věc, kterou tedy v kódu musíme udělat, je vložit všechny potřebné moduly této knihovny.

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("DhcpExample");
```

Protože všechny funkce a definice, které jsme v předchozí části vložili, patří do knihovny ns3, musíme při jejich použití v dalším kódu vždy určit, že jsou z této knihovny. K tomu, abychom to nemuseli pokaždé nastavovat ručně, slouží příkaz `using namespace ns3`. Ten překladači řekne, aby předpokládal, že funkce, které používáme, jsou z knihovny ns3. Druhý příkaz jen spustí a pojmenuje LOG.

```
int
main (int argc, char *argv[])
{
  bool verbose = false;

  if (verbose)
    {
      LogComponentEnable ("DhcpServer", LOG_LEVEL_ALL);
      LogComponentEnable ("DhcpClient", LOG_LEVEL_ALL);
      LogComponentEnable ("UdpEchoServerApplication",
      LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoClientApplication",
      LOG_LEVEL_INFO);
    }
```

Na začátku programu je proměnná `verbose`, která slouží k zapnutí/ vypnutí dodatečných výpisů programu. Další řádky pak slouží pro samotné výpisy.

```
Time stopTime = Seconds (20);

NS_LOG_INFO ("Create␣nodes.");
NodeContainer nodes;
NodeContainer router;
nodes.Create (3);
router.Create (1);

NodeContainer net (nodes, router);
```

Nastavíme délku simulace, vytvoříme 4 uzly (tři PC a jeden směrovač). Nakonec přidáme všechny PC i směrovač do jedné skupiny s názvem `net`, aby bylo možné nastavovat je všechny současně.

```
NS_LOG_INFO ("Create␣channels.");
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("5Mbps"));
csma.SetChannelAttribute ("Delay", StringValue ("2ms"));
csma.SetDeviceAttribute ("Mtu", UintegerValue (1500));
NetDeviceContainer devNet = csma.Install (net);
```

Vytvoříme WiFi síť a nastavíme její parametry. K této síti posledním příkazem připojíme všechna zařízení.

```
NodeContainer p2pNodes;
```

```
p2pNodes.Add (net.Get (3));
p2pNodes.Create (1);

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

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Vytvoříme ještě jeden uzel (PC4), vytvoříme peer to peer síť, nastavíme její parametry a propojíme s ní směrovač a nový uzel (PC4).

```
InternetStackHelper tcpip;
tcpip.Install (nodes);
tcpip.Install (router);
tcpip.Install (p2pNodes.Get (1));
```

Na všech uzlech v síti nainstalujeme TCP/IP stack, aby byly schopny spolu komunikovat.

```
Ipv4AddressHelper address;
address.SetBase ("10.10.10.0", "255.255.255.252");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
```

Vytvoříme IPv4 síť a necháme program automaticky přiřadit IP adresy z této sítě směrovači a PC4.

```
Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4> ipv4Ptr = p2pNodes.Get (1)->GetObject<Ipv4> ();
Ptr<Ipv4StaticRouting> staticRoutingA =
ipv4RoutingHelper.GetStaticRouting (ipv4Ptr);
staticRoutingA->AddNetworkRouteTo (Ipv4Address ("192.168.10.0"),
                                   Ipv4Mask ("/24"),
                                   Ipv4Address ("10.10.10.1"),
                                   1);
```

Na směrovači se vytvoří statická cesta k propojení peer to peer a WiFi sítě.

```
DhcpHelper dhcpHelper;
```

```
Ipv4InterfaceContainer fixedNodes =
dhcpHelper.InstallFixedAddress (devNet.Get (0),
Ipv4Address ("192.168.10.11"), Ipv4Mask ("/24"));

fixedNodes = dhcpHelper.InstallFixedAddress (devNet.Get (3),
Ipv4Address ("192.168.10.1"), Ipv4Mask ("/24"));
```

Na směrovači se nastaví statická IP adresa 192.168.10.1 a na PC1 statická adresa 192.168.10.11.

```
ApplicationContainer dhcpServerApp =
dhcpHelper.InstallDhcpServer
(devNet.Get (3), Ipv4Address ("192.168.10.1"),
Ipv4Address ("192.168.10.0"), Ipv4Mask ("/24"),
Ipv4Address ("192.168.10.12"), Ipv4Address ("192.168.10.254"),
Ipv4Address ("192.168.10.1"));
```

Na směrovači se spustí DHCP server. Adresa DHCP serveru je nastavena na 192.168.10.1, adresy se přidělují z rozsahu 192.168.10.11 - 192.168.10.255, maska sítě je 255.255.255.0 a adresa default gateway 192.168.10.1.

```
dhcpServerApp.Start (Seconds (1.0));
dhcpServerApp.Stop (stopTime);
```

DHCP server se spustí až po jedné sekundě od začátku simulace a na konci simulace se zastaví.

```
NetDeviceContainer dhcpClientNetDev1, dhcpClientNetDev2;
dhcpClientNetDev1.Add (devNet.Get (1));
dhcpClientNetDev2.Add (devNet.Get (2));

ApplicationContainer dhcpClient1 =
dhcpHelper.InstallDhcpClient (dhcpClientNetDev1);
dhcpClient1.Start (Seconds (0.0));
dhcpClient1.Stop (stopTime);

ApplicationContainer dhcpClient2 =
dhcpHelper.InstallDhcpClient (dhcpClientNetDev2);
dhcpClient2.Start (Seconds (2.0));
dhcpClient2.Stop (stopTime);
```

Zařízení PC2 a PC3 se nastaví jako DHCP klienti. PC2 si vyžádá IP adresu ještě dříve, než se spustí DHCP server a nedostane tedy žádnou odpověď. PC3 si vyžádá

IP adresu dvě sekundy po startu simulace, kdy už běží DHCP server, a obdrží tak IP adresu.

```
ApplicationContainer serverApps =
echoServer.Install (p2pNodes.Get (1));
serverApps.Start (Seconds (0.0));
serverApps.Stop (stopTime);

UdpEchoClientHelper echoClient
(p2pInterfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval",
TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps =
echoClient.Install (nodes.Get (1));
clientApps.Start (Seconds (15.0));
clientApps.Stop (stopTime);
```

PC4 se nastaví jako echo server, bude tedy odpovídat na echo požadavky. PC1 se nastaví jako echo klient, který pošle jeden echo paket o velikosti 1024B. Interval odesílání v tomto případě není potřebný a je zde nastaven pouze pro úplnost. PC1 začne posílat echo požadavky v 15. sekundě.

```
Simulator::Stop (stopTime + Seconds (10.0));

csma.EnablePcapAll ("dhcp-csma");
pointToPoint.EnablePcap ("dhcp-csma", p2pDevices.Get(0));

AnimationInterface anim ("dhcp-NetAnim.xml");
anim.SetConstantPosition (nodes.Get (0), 10, 10);
anim.SetConstantPosition (nodes.Get (1), 10, 40);
anim.SetConstantPosition (nodes.Get (2), 10, 80);
anim.SetConstantPosition (p2pNodes.Get (0), 40, 40);
anim.SetConstantPosition (p2pNodes.Get (1), 80, 40);
anim.EnablePacketMetadata (true);

NS_LOG_INFO ("Run␣Simulation.");
Simulator::Run ();
Simulator::Destroy ();
```

```
NS_LOG_INFO ("Done.");
return 0;
 }
```

Nastavíme čas konce simulace (10 s po zaslání poslední zprávy, aby bylo jisté, že stihne doběhnout), zapneme ukládání paketů do souborů `*.pcap`, nastavíme pozici jednotlivých uzlů pro program NetAnim, aby odpovídala znázornění v topologii sítě, a nakonec spustíme simulaci.

7.1.13 Simulace v programu NetAnim

1. Z lišty vlevo spustíme program NetAnim.
2. Z horního menu programu vybereme `Open XML trace file` a ze složky `/home/student/ns-allinone-3.29/ns-3.29` zvolíme soubor `dhcp-NetAnim.xml`.
3. Animaci si spustíme tlačítkem `Play Animation`.

7.2 Laboratorní úloha `FTP, TFTP, HTTP`

Tato laboratorní úloha se zabývá konfigurací FTP serveru. Vytvářením systémových uživatelů, nastavováním jejich přístupových práv k souborovému systému a porovnáním FTP a TFTP protokolů. Úloha obsahuje i vlastní HTTP server v jazyce Python, kde si vyzkoušíme metody HTTP.

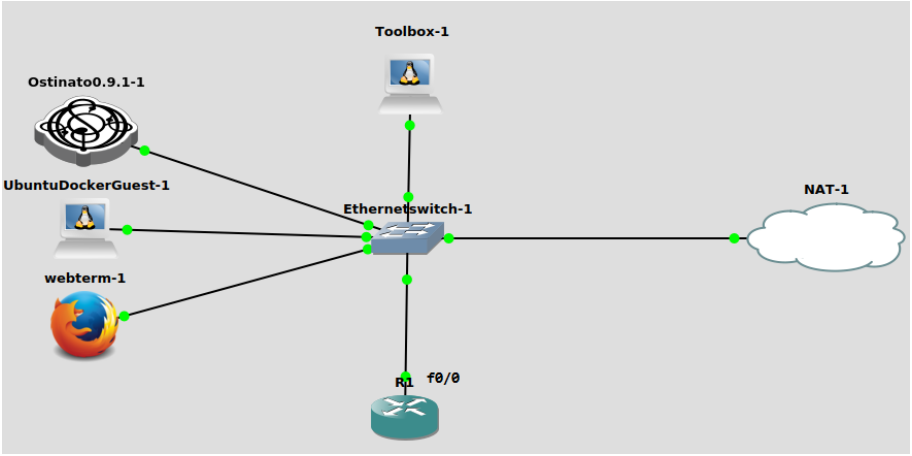
7.2.1 Teoretický úvod

FTP využívá transportní protokol Transmission Control Protocol (TCP), který garantuje spolehlivé doručování ve správném pořadí. FTP využívá **TCP** porty 20 a 21. Port 20 slouží k přenosu dat a port 21 k řízení komunikace pomocí FTP zpráv, takže vytváří dvě spojení. FTP podporuje dva přenosové režimy, aktivní a pasivní. V **aktivním** režimu klient na příkazovém portu kontaktuje server na jeho příkazový port 21 a posílá zprávu `PORT X`, kde X je port na datovou komunikaci. Server následně pošle potvrzovací zprávu ACK, a poté vytvoří spojení s datovým portem klienta, který klient specifikoval v první zprávě. V **pasivním** režimu klient na příkazovém portu kontaktuje server na jeho příkazový port 21 a posílá zprávu `PASV`. Server odpovídá zprávou PORT X, kde X je jeho datový port. Klient následně vytvoří datové spojení se serverem na portu, který byl specifikován v předešlé zprávě.

TFTP obsahuje jen některé základní funkce protokolu FTP. Je určen pro přenos souborů v případě, kdy je protokol FTP nevhodný pro svou komplikovanost. Žádost o spojení je vždy inicializována na cílovém portu 69. Protokol TFTP funguje nad protokolem UDP, který pro svou komunikaci nevytváří spojení, takže protokol TFTP musí obsahovat vlastní řízení spojení. V jednom spojení se vždy přenáší jediný soubor, při komunikaci na síti se vždy přenáší jen jeden paket a před posláním dalšího se čeká na potvrzení o doručení. TFTP podporuje 3 přenosové módy: **neascii** (text v ASCII znacích), **octet** (8b binární data) a **mail** (neměl by se již používat). TFTP narozdíl od FTP nepodporuje autentizaci a používá jen jedno spojení, kde se v jednom směru kontroluje tok paketů a data se posílají v opačném směru pomocí stejných UDP soketů.

HTTP slouží pro přenos HTML dokumentů v systému www. Od ostatních protokolů založených na TCP liší tím, že spojení se ukončí poté, co se dokončí provedení požadavky, nebo série požadavků. HTTP server naslouchající na portu 80 čeká, až klient pošle řetězec s požadavkem **GET / HTTP / 1.1**, který žádá o zaslání domovské stránky serveru. Tento požadavek je následován sérií hlaviček popisujících požadavky klienta. Po přijetí požadavku pošle server zprávu s odpovědí 200 OK, která je následována hlavičkami spolu se samotnou zprávou.

7.2.2 Topologie sítě



Obr. 7.2: Topologie laboratorní úlohy FTP, TFTP, HTTP

7.2.3 Hlavní cíle

1. Konfigurace FTP serveru.
2. Nastavení uživatelského přístupu k adresářům.
3. Porovnání FTP a TFTP přenosu.
4. Zkoumání HTTP metod.

7.2.4 Příprava programu GNS3

1. V programu VMware spustíme OS GNS3 VM.
2. Po načtení OS spustíme program GNS3.
3. Úlohu zapojíme podle obrázku 7.2.
4. Spustíme všechna zařízení.
5. Na portu f0/0 Cisco směrovače zadáme příkaz `ip add dhcp` a `no sh`, kterými nastavíme získání IP adresy od DHCP serveru, který je spuštěn na zařízení NAT.
6. Na zařízeních Toolbox, Webterm, UbuntuDocker a Ostinato také nastavíme automatické přidělování adresy IP. Konfiguraci změníme příkazem `nano/etc/network/interfaces` a odkomentováním řádků `auto eth0` a `IFAC eth0 inet dhcp`.
7. Všechna Linuxové zařízení restartujeme.
8. Ověříme, zda všechna zařízení mají konfiguraci IP. Na zařízení Cisco příkazem `show ip interface brief` a na linuxových zařízeních příkazem `ifconfig`.

7.2.5 Konfigurace FTP serveru

Zařízení `Toolbox-1` je linuxový server, na kterém běží služby enginx (HTTP), vsftpd (FTP) a tftpd (TFTP). Otevřeme si jeho konzoli a nakonfigurujeme následující parametry.

(a) Vytvoření systémového uživatele.

```
useradd Student1
passwd Student1
```

(b) Vytvoříme složku a přiřadíme ji uživateli jako domovskou.

```
cd /var
mkdir ftp
cd ftp
mkdir folder1
usermod -d /var/ftp/folder1 Student1
```

(c) Pomocí souboru `vsftpd.conf` můžeme definovat chování FTP démona. Soubor si otevřeme příkazem `nano /etc/vsftpd.conf`. Soubor `vsftpd.conf` upravíme podle následující šablony.

```
listen=YES
anonymous_enable=NO
local_enable=YES
chroot_local_user=NO
write_enable=YES
local_umask=022
pam_service_name=ftp
chroot_list_enable=YES
chroot_list_file=/etc/vsftpd.chroot_list
allow_writeable_chroot=YES
```

(d) Vytvoříme soubor `vsftpfd.chroot_list`, ve kterém definujeme, kteří uživatelé budou mít přístup pouze do své domovské složky. Uživatelé, kteří v listu definováni nejsou, budou mít přístup do celého souborového systému.

```
touch /etc/vsftpfd.chroot_list
```

Soubor otevřeme příkazem `nano /etc/vsftpfd.chroot_list` a dopíšeme tam uživatele `Student1`. Soubor zavřeme a uložíme pomocí `ctrl + x`.

7.2.6 Konfigurace FTP klienta

`UbuntuDocker` je zařízení s OS Ubuntu 16.04, ale nepodporuje grafické rozhraní. My si na něm nakonfigurujeme FTP klienta.

(a) Instalace FTP klienta.

```
apt-get update
apt install ftp
```

(b) Vytvoříme si textový dokument `file1.txt`, do něhož vložíme krátký text.

```
echo "0123456789" > file1.txt
```

(c) Z klientského PC se připojíme na FTP server. Po zadání příkazu ftp a IP se autentizujeme jménem Student1 a heslem, které jsme si vytvořili na serveru. Příkazem `put` nahrajeme soubor na server.

```
ftp 192.168.122.x
pwd
put file1.txt
ls -al
```

Po zadání příkazu `pwd` bychom neměli vidět celou cestu do uživatelské složky, ale ten by se měl tvářit jako kořenový adresář.
Nahrání souboru ověříme na straně serveru tak, že si zobrazíme obsah složky `folder1`.

```
cd /var/ftp/folder1
ls -al
```

7.2.7 Konfigurace TFTP klienta

V praxi je zvykem zálohovat konfiguraci síťových zařízení na TFTP server, aby bylo v případě poruchy možné zařízení jednoduše nahradit. Zkusíme si zálohovat konfiguraci směrovače `R1`.

```
copy running-config tftp
```

Po potvrzení příkazu zadáme IP adresu TFTP serveru a soubor nazveme `R1config`.
Nahrání souboru si ověříme na straně serveru.

```
more /tftpboot/R1config
```

7.2.8 Samostatná práce

1. Na FTP serveru vytvořte dalšího uživatele Student2, který bude mít svoji domovskou složku folder2 a bude mít přístup jen do ní.
2. Do složky folder2 nahrajte jako FTP klient soubor `file2.txt`.
3. Vytvořte uživatele Ucitel, který bude mít svoji domovskou složku `folder` a bude mít přístup do celého souborového systému.

4. Na zařízení webterm spusťte webový prohlížeč Mozila a prostřednictvím něj se připojte na FTP server. Připojení realizujte zadáním ftp://192.168.122.x do vyhledávacího pole a autentizujte se jako uživatel Ucitel.
5. Přes grafické rozhraní zařízení webterm zkuste vstoupit do složky `folder1` a kliknutím na `file1.txt` zobrazit obsah souboru.
6. V souboru `vsftpd.conf` změňte parametr `chroot_local_user=NO` na `YES`. Sledujte, jak se změnila práva uživatelů na procházení souborového systému.
7. Otestujte, co se stane při nahrávání konfigurace Cisco směrovače na TFTP server, pokud zvolíme název, který již existuje.

7.2.9 Kontrolní otázky

Před kontrolními otázkami doporučuji spustit odchytávání paketů na rozhraních jednotlivých zařízení. Pravým tlačítkem klikneme na spojení zařízení s přepínačem a zvolíme `Start capture`. Pokud se pakety odchytávají, uvidíme nad spojením ikonu lupy. Program Wireshark spustíme kliknutím pravým tlačítkem na spojení a volbou nabídky `Wireshark`.

1. Který přenosový režim je použit při nahrávání souboru `file1.txt` na FTP server přes terminál zařízení UnuntuDocker?
2. Který přenosový režim je použit při stahování souboru `file1.txt` z FTP serveru do zařízení UbuntuDocker?
3. Který přenosový režim je použit při otevíraní souboru `file1.txt` z FTP serveru na zařízení webterm?
4. Jak se změnila práva uživatelů na procházení kořenového systému při změně parametru na `chroot_local_user=YES`?
5. Co se stane s původním souborem v případě, že zvolíme stejný název souboru při nahrávání konfigurace Cisco směrovače na TFTP server?
6. V jakém přenosovém módu je přenášena konfigurace Cisco směrovače na TFTP server?
7. Jaké jsou hlavní rozdíly mezi FTP a TFTP při ukončování spojení?

7.2.10 Kontrolní otázky - odpovědi

1. Aktivní přenosový režim.
2. Aktivní přenosový režim.
3. Pasivní přenosový režim.
4. Uživatelé definováni v souboru `nano /etc/vsftpfd.chroot_list` mohli nejprve vidět jen svůj adresář, nyní mohou vidět celý souborový systém. U uživatelů, kteří definováni nebyli, je situace přesně opačná.
5. Soubor se přepíše.

6. Octet (8b binární data).
7. FTP: Odesílající strana ukončuje datové TCP spojení paketem s příznaky FIN a ACK, na což mu příjmající strana odpovídá také paketem s příznaky FIN a ACK. Následně na řídicím spojení informuje server klienta o ukončení přenosu zprávou FTP "Transfer Complete". TFTP spojení se neukončuje. Spojení je ukončeno UDP zprávou, která potvrzuje doručení posledních přijatých dat.

7.2.11 Metody protokolu HTTP

V této části laboratorní úlohy si vyzkoušíme komunikaci s HTTP serverem. Otestujeme si, jaké různé metody tento protokol podporuje a jaký je mezi nimi rozdíl. Jednotlivé požadavky lze simulovat pomocí programu `telnet`.

1. Spustíme si program Wireshark
2. Jako zařízení pro záznam si vybereme Loopback:lo a spustíme záznam
3. Do filtru zobrazení zadáme filtr `tcp.port eq 8080`, který nám vyfiltruje jen naše pakety
4. Přes Terminal si příkazem `python /Desktop/DHCP server/HTTP-server.py` spustíme přiložený program HTTP serveru.
5. V dalším terminálovém okně si příkazem `telnet 127.0.0.1 8080` spustíme HTTP klienta. Po přijetí odpovědi program telnet vždy spojení ukončí a je třeba jej spustit znovu.

Nejzákladnější HTTP metodou je GET, která slouží k získání zdroje ze serveru. Například po zadání adresy do jakéhokoli webového prohlížeče. Ten pošle na server zadané domény požadavek na vybranou stránku. Metoda GET je velmi jednoduchá, protože nepotřebuje žádné další části. Pro odeslání jednoduché GET požadavky stačí do programu telnet napsat jediný příkaz. Příkaz se potvrdí odesláním dvou nových řádků.

```
GET / HTTP/1.1
```

Formát příkazu pro telnet je `METHOD resource HTTP_version`, kde `METHOD` je metoda požadavků, `resource` je zdroj stránka, o kterou klient žádá, a `HTTP_version` je použitá verze protokolu HTTP. Server by měl na tento požadavek odpovědět kódem 200 - OK a zaslat klientovi obsah souboru index.html. Pokud chceme na server spolu s požadavkem poslat i data (například obsah vyplněného formuláře po kliknutí na tlačítko `submit`), můžeme jej připojit na konec požadavku. Pro oddělení požadavku a dat se používá znak '?' a pro oddělení jednotlivých položek znak '&'. Položky mají formát `název=hodnota`, což znamená, že do položky `název` se má přiřadit `hodnota`.

Server je nastaven tak, aby si všechna tyto data ukládal do své vnitřní databáze. Po obdržení takové žádosti data přidá ke starým a následně výsledek odešle zpět klientovi. Kód zprávy nastaví na 200 - OK.

Pro odlišení metody GET a PUT je server naprogramován tak, že pro ukládání dat na server slouží metoda PUT, zatímco pro metodu GET server jen zašle zpět aktuální hodnoty požadovaných parametrů. Pro požadavky GET proto server neočekává žádnou hodnotu parametrů. Výsledné požadavky pak mohou vypadat například takto:

```
PUT /?name=Student&age=65&sex=M HTTP/1.1
GET /?name&sex HTTP/1.1
```

Další metodou HTTP je HEAD. Server na takový požadavek odpovídá stejnými hlavičkami jako v případě zaslání požadavku GET, neposílá však žádná data. V telnetu si tuto metodu můžeme vyzkoušet velmi jednoduše. Příkaz se opět potvrdí odesláním dvou nových řádků.

```
HEAD / HTTP/1.1
```

Stejně jako GET fungují i metody PUT a DELETE, které slouží pro přidání (PUT) nebo odstranění (DELETE) dat v databázi serveru. Data do metody DELETE server očekává ve stejném formátu jako pro GET - `?Name&age&sex`, tedy jen názvy bez hodnoty. Mírně odlišná je metoda POST, která umožňuje v hlavičce požadavku uvést velikost dat, která chce klient připojit ke svému požadavku, a následně přenést binární data. Výhoda tohoto přístupu je, že data nejsou reprezentovány pomocí ASCII znaků a je tedy možné přenést jakákoli data. Další výhodou je, že je takto možné přenést větší množství dat než metodou GET. Požadavek POST včetně jeho přídavných hlaviček je třeba opět potvrdit odesláním dvou nových řádků a následně je možné začít odesílat data.

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 25

name=Student&age=65&sex=M
```

Pozor! Při změně dat na posledním řádku je třeba změnit jejich velikost v hlavičce požadavku.

8 Záver

Zoznámili sme sa s protokolmi DHCP, TFTP, FTP, HTTP a službou NAT. V teoretickej časti práce boli popísané základné parametre a princípy fungovania týchto protokolov.

Cieľom diplomovej práce bolo pripraviť výukové prostredie a laboratórne úlohy za použitia rôznych simulačných nástrojov. Prvým nástrojom je program GNS3, ktorý slúži na emuláciu rôznych zariadení od viacerých výrobcov. Vďaka tomu je možné spájať zariadenia s rôznymi operačnými systémami a testovať ich spoluprácu v sieti. GNS3 je voľne dostupný a predstavuje alternatívu voči iným komerčným programom, ktoré sú porovnané v teoretickej časti.

Nástroj ns3 slúži na simuláciu sieťovej prevádzky a zaťaženia siete, ktorý môže byť v reálnom čase prepojený so skutočnou sieťou.

Teoretická časť plynulo prechádza do praktickej časti, kde je v úvode popísaná inštalácia a správne nastavenie simulačných nástrojov a ďalších pomocných programov v OS Ubuntu 18.10. Toto výukové prostredie je vďaka spomínaným simulačným nástrojom univerzálne a v budúcnosti ho je možné veľmi jednoducho rozširovať pre ďalšie úlohy.

V závere diplomovej práce sú dve laboratórne úlohy s teoretickým úvodom, podrobnými návodmi a kontrolnými otázkami, ktoré overia samostatnú prácu študenta a porozumenie danej problematike. Súčasťou prvej úlohy je hľadanie problémov v sieti, ktoré súvisí s logickým uvažovaním. Okrem simulačných nástrojov je v laboratórnej úlohe použitý aj vlastný server HTTP v jazyku Python kvôli lepšiemu porozumeniu metódam HTTP. Laboratórne úlohy sú napísané v Českom jazyku z dôvodu, aby mohli byť bez väčších úprav použité na výuku. Dôraz bol kladený na prepojenie teoretických a praktických znalostí.

Literatúra

[1] Ing. Jan Jeřábek, Ph.D. Pokročilé komunikační techniky. Vysoké učení technické v Brně Fakulta elektrotechniky a komunikačních technologií Ústav telekomunikací Purkyňova 118, 612 00 Brno, 2015, 2012(1.), 96 [cit. 2019-01-29]. ISBN 978-80-214-4636-6.

[2] IP Addressing: NAT Configuration Guide, Cisco IOS Release 15M&T [online]. 2018, Edited 16.10.2018 [cit. 2018-12-02]. Dostupné z: <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/15-mt/nat-15-mt-book/iadnat-addr-consv.html>.

[3] DHCP Overview. Cisco Content Hub [online]. 2018, 15.8.2018 [cit. 2018-12-02].Dostupné z URL: Dostupné z: <https://content.cisco.com/chapter.sjs?uri=/searchable/chapter/content/en/us/td/docs/ios-xml/ios/ipaddr_dhcp/configuration/15-mt/dhcp-15-mt-book/dhcp-overview.html.xml>.

[4] Základní informace o protokolu DHCP (Dynamic Host Configuration Protocol) [online]. 2018, 18.4.2018 [cit. 2018-12-02]. Dostupné z: <https://support.microsoft.com/cs-cz/help/169289/dhcp-dynamic-host-configuration-protocol-basics>.

[5] Getting Started with GNS3 [online] August 18, 2018 [cit. 2018-11-17].Dostupné z URL: Dostupné z: <https://docs.gns3.com/1PvtRW5eAb8RJZ11maEYD9_aLY8kkdhgaMB0wPCz8a38/index.html#>.

[6] About: What is ns-3? [online]. 2018 [cit. 2018-12-02].Dostupné z URL: Dostupné z: <https://www.nsnam.org/about/>.

[7] J. Postel a J. Reynolds. FILE TRANSFER PROTOCOL (FTP). Www.w3.org [online]. 1985 [cit. 2019-04-29]. Dostupné z: <https://www.w3.org/Protocols/rfc959/Overview.html>.

[8] Jay Ribak. Active FTP vs. Passive FTP, a Definitive Explanation. Www.slacksite.com [online]. Nashua, NH [cit. 2019-04-29]. Dostupné z: <https://slacksite.com/other/ftp.html>.

[9] K. Sollins. THE TFTP PROTOCOL (REVISION 2). Www.tools.ietf.org [online]. MIT, 1992 [cit. 2019-04-29]. Dostupné z: <https://tools.ietf.org/html/rfc1350>.

[10] Robert Shingledecker. The Core Project - Tiny Core Linux. Http://tinycorelinux.net [online]. December 01, 2008 [cit. 2019-04-30]. Dostupné z: <http://tinycorelinux.net/welcome.html>.

[11] Ostinato Packet Generator. Ostinato.org/ [online]. 2019 [cit. 2019-04-30]. Dostupné z: <http://https://ostinato.org/>.

[12] Hypertext Transfer Protocol – HTTP/1.1. Network Working Group [online]. 1999, 1999, 176 [cit. 2019-04-29]. Dostupné z: <https://tools.ietf.org/html/rfc2616?spm=5176.doc32013.2.3.Aimyd7>.

[13] Hypertext Transfer Protocol Version 2 (HTTP/2). Internet Engineering Task Force (IETF) [online]. 2015, 2015, 96 [cit. 2019-04-29]. Dostupné z: <https://tools.ietf.org/html/rfc7540>.

Zoznam symbolov, veličín a skratiek

NAT	Network Address Translation
ns-3	network simulator 3
GNS3	Graphic Network Simulator 3
DHCP	Dynamic Host Configuration Protocol
ICMP	Internet Control Message Protocol
PAT	Port Address Translation
TCP/IP	Transmission Control Protocol/Internet Pro-tocol
DNS	Domain Name System
NVRAM	Non-Volatione Random Access memory
FTP	File Transfer Protocol
TFTP	Trivial File Transfer Protocol
MAC	Media Access Control
GNU	GNU's Not Unix
GPLv2	General Public Licence v2
LTE	Long Term Evolution
OS	Operating System
IDE	Integrated Developement Enviroment
PHP	PHP: Hypertext Preprocessor
JDK	Java Development Kit
GUI	Graphical User Interface
VM	Virtual Machine
XML	eXtensible Markup Language
VIRL	Virtual Internet Routing Lab
IOS	Internetwork Operating System
RAM	Random Access Memory
HTTP	Hypertext Transfer Protocol
HTML	yperText Markup Language)
HTTPS	HTTP Secure
URL	Uniform Resource Locator
AMD-V	AMD virtualization
ARM	Advanced RISC Machine)
SNMP	Simple Network Management Protocol
SSH FTP	Secured Shell FTP
FTPS	FTP Secured
VPN	Virtual Private Network

Zoznam príloh

A HTTP server v jazyku Python **68**

B Obsah priloženého CD **75**

A HTTP server v jazyku Python

V tejto kapitole si popíšeme fungovanie programu HTTP servera ktorý bol vytvorený v jazyku Python pre laboratórnu úlohu č. 2.

```
#!/usr/bin/python
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
from urlparse import urlparse
from os import curdir, sep
import cgi
```

Prvý riadok kódu slúži ako informácia pre operačný systém v akom programe má tento súbor spustiť. Na začiatku kódu musíme naimportovať všetky potrebné moduly a ich funkcie. V jazyku Python na to slúži príkaz import, ktorý je možné používať v dvoch formách.

```
import MODULE1
from MODULE2 import FUNC1, FUNC2
```

Prvý príkaz naimportuje celý modul `MODULE1`. Funkcie modulu následne voláme príkazom `MODULE1.FUNC()`. Druhý príkaz slúži na naimportovanie len vybraných funkcii modulu. Tieto funkcie je následne možné volať bez názvu modulu príkazmi `FUNC1()` alebo `FUNC2()` rovnako ako naše vlastné funkcie.

```
PORT_NUMBER = 8080


# Dictionary in which all the data from user is stored
my_dict = {}
```

Do premennej `PORT_NUMBER` si uložíme číslo portu na ktorom chceme spustiť HTTP server. Tento krok nie je potrebný, kód je takto ale prehľadnejší než keby sme všade ďalej používali priamo číslo 8080. Výhodou je, že sa dá číslo portu v prípade potreby jednoducho zmeniť z jedného miesta. Štandardný port pre HTTP server je 80, tento je však často obsadený operačným systémom a preto použijeme port 8080.

```
# This class will handle any GET/POST/HEAD/DELETE
# incoming requests from the browser
class HTTP_handler(BaseHTTPRequestHandler):
```

Pri vytváraní HTTP servera na konci nášho kódu je potrebné definovať triedu v ktorej nájde funkcie pre ich spracovanie. Po prijatí HTTP požiadavky s metódou `METHOD` server zavolá funkciu `do_METHOD` z našej triedy (napríklad pre požiadavku `GET` zavolá funkciu `do_GET`). Vytvoríme si teda vlastnú triedu, ktorú nazveme `HTTP_handler`.

```
# GET request handler
def do_GET(self):
  # Parts of the URL:
  # <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
  path = urlparse(self.path).path
  query = urlparse(self.path).query
```

Následujúca časť kódu sa vďaka svojej dĺžke môže zdať zložitá, preto si ju rozdelíme na menšie celky a vysvetlíme postupne. V prvej časti sa pomocou funkcie `urlparse` rozdelí URL adresa na jednotlivé časti. Ich názvy sú uvedené v komentári nad príkazom. Nás zaujímajú časti path a query.

```
if query != '':
  if path == "/":
    query_components = list(query.split("&"))
    self.send_response(200) # 200 = OK
    self.send_header('Content-type', 'text/html')
    self.end_headers()
    response_dict = {}
    for key in my_dict:
      if key in query_components:
        response_dict[key] = my_dict[key]
    self.wfile.write(str(response_dict))
    # Send the current dictionary back
```

Ak URL adresa smeruje na domovskú stránku (`/`) a obsahuje parametre, rozdelíme si časť s parametrami na jednotlivé prvky. Tie si uložíme v premennej `query_components`. Nastavíme kód odpovede na 200 - OK a typ odpovede na `'text/html'`. Vytvoríme si novú dočasnú premennú `response_dict` do ktorej si nakopírujeme všetky požadované prvky z premennej `my_dict`. Napokon odošleme obsah novej premennej klientovi ako odpoveď na požiadavku.

```
else:
  if path == "/":
    path = "/index.html"
```

Ak URL adresa nešpecifikuje cestu k žiadnemu súboru (cesta je nastavená na `/`), presmerujeme užívateľa na stránku `/index.html`.

```
try:
  # Check the file extension required and
  # set the right mime type
```

```
  sendReply = False
  if path.endswith(".html"):
    mimetype='text/html'
    sendReply = True
  elif path.endswith(".jpg"):
    mimetype='image/jpg'
    sendReply = True
  elif path.endswith(".ico"):
    mimetype='image/ico'
    sendReply = True
  elif path.endswith(".gif"):
    mimetype='image/gif'
    sendReply = True
  elif path.endswith(".js"):
    mimetype='application/javascript'
    sendReply = True
  elif path.endswith(".css"):
    mimetype='text/css'
    sendReply = True
```

Podľa prípony požadovaného súboru nastavíme typ odpovede a nastavíme premennú `sendReply` na hodnotu `True`. Ak teda server dostane požiadavku na nepodporovaný typ súboru neodošle na ňu žiadnu odpoveď.

```
if sendReply == True:
  # Open the static file requested and send it
  f = open(curdir + sep + path)
  self.send_response(200)
  self.send_header('Content-type', mimetype)
  self.end_headers()
  self.wfile.write(f.read())
  f.close()
  return
```

Ak sa má zaslať odpoveď, server sa pokúsi otvoriť a poslať späť požadovaný súbor. V prípade, že je všetko v poriadku, vyskočí z funkcie a požiadavka sa považuje za vybavenú. Všimnite si že server nijako nekontroluje o aké súbory ho klient žiada a bez problémov mu pošle aj súbor z iného priečinka. V reálnom použití sa toto musí vždy ošetriť.

```
except IOError:
  self.send_error(404, 'File␣Not␣Found:␣%s' % self.path)
```

Ak pri pokuse o otvorenie alebo čítanie súboru nastane chyba, server skočí na tento kód. Tu sa odošle chyba 404 - File Not Found.

```
# HEAD request handler
def do_HEAD(self):
  # Parts of the URL:
  # <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
  path = urlparse(self.path).path
  query = urlparse(self.path).query
  if query != '':
    if path == "/":
      self.send_response(200) # 200 = OK
      self.send_header('Content-type', 'text/html')
      self.end_headers()
  else:
    if path == "/":
      path = "/index.html"

    try:
      # Check the file extension required and
      # set the right mime type

      sendReply = False
      if path.endswith(".html"):
        mimetype='text/html'
        sendReply = True
      elif path.endswith(".jpg"):
        mimetype='image/jpg'
        sendReply = True
      elif path.endswith(".ico"):
        mimetype='image/ico'
        sendReply = True
      elif path.endswith(".gif"):
        mimetype='image/gif'
        sendReply = True
      elif path.endswith(".js"):
        mimetype='application/javascript'
        sendReply = True
      elif path.endswith(".css"):
```

```
        mimetype='text/css'
        sendReply = True

      if sendReply == True:
        # Open the static file requested and send it
        f = open(curdir + sep + path)
        self.send_response(200)
        self.send_header('Content-type', mimetype)
        self.end_headers()
        f.close()
        return

    except IOError:
      self.send_error(404,'File␣Not␣Found:%s' % self.path)
```

Z definície požiadavka HEAD reaguje rovnako ako požiadavka GET, no do odpovede nezahrnie telo správy, len jej hlavičky. Kód je preto takmer identický s predošlou funkciou, sú z neho len odstránené všetky volania funkcie `self.wfile.write()`.

```
# PUT request handler
def do_PUT(self):
  # Parts of the URL:
  # <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
  path = urlparse(self.path).path
  query = urlparse(self.path).query
  if query != '':
    if path == "/":
      query_components = dict(qc.split("=")
          for qc in query.split("&"))
      self.send_response(200) # 200 = OK
      self.send_header('Content-type', 'text/html')
      self.end_headers()
      my_dict.update(query_components)
      self.wfile.write(str(my_dict))
      # Send the current dictionary back
```

Funkcia pre požiadavku PUT je opäť takmer totožná s funkciou `do_GET`. Neobsahuje však časť kódu ktorá by klientovi zasielala požadovaný súbor (tá totiž slúži hlavne pre webové prehliadače) a zameriava sa len na spracovanie parametrov. Hlavný rozdiel oproti metóde GET je v tom, že ku každému zaslanému parametru očakáva aj jeho novú hodnotu. Príkaz `my_dict.update(query_components)` potom

pridá do premennej `my_dict` prvky, ktoré tam ešte nie sú a upraví hodnotu prvkov ktoré tam už sú na prijaté hodnoty.

```
# POST request handler
def do_POST(self):
  ctype, pdict = cgi.parse_header
  (self.headers.getheader('content-type'))
  # Check if the content type is application
  #/x-www-form-urlencoded
  if ctype == 'application/x-www-form-urlencoded':
    length = int(self.headers.getheader('content-length'))
    query = self.rfile.read(length)
    postvars = dict(qc.split("=")for qc in query.split("&"))
  else:
    postvars = {}
  # Add the new elements to the global dictionary
  my_dict.update(postvars)
  self.send_response(200) # 200 = OK
  self.send_header('Content-type', 'text/html')
  self.end_headers()
  self.wfile.write(str(my_dict))
  # Send the current dictionary back
```

Táto funkcia najprv zistí aký typ dát má očakávať (premenná `ctype`), ich dĺžku (`length`) a následne ich prijme. Ďalej je funkcia zhodná s predošlou funkciou - teda aktualizuje premennú `my_dict`, vytvorí hlavičku odpovede a zašle ju spolu s obsahom premennej `my_dict` klientovy.

```
# DELETE request handler
def do_DELETE(self):
  # Parts of the URL:
  # <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
  path = urlparse(self.path).path
  query = urlparse(self.path).query
  if query != '':
    if path == "/":
      query_components = list(query.split("&"))
      self.send_response(200) # 200 = OK
      self.send_header('Content-type', 'text/html')
      self.end_headers()
      # Remove the received elements from the g. dictionary
```

```
      for key in query_components:
        if key in my_dict:
          del my_dict[key]
      self.wfile.write(str(my_dict))
      # Send the current dictionary back
  else:
    self.wfile.write('Please␣tell␣me␣what␣to␣delete')
```

Posledná funkcia tejto triedy slúži na spracovanie požiadaviek typu DELETE. Po prijatí požiadavky od klienta spracuje jej parametre a následne odstraní z premennej `my_dict` všetky kľúče z požiadavky.

```
try:
  # Create a web server and define the handler to manage the
  # incoming requests
  server = HTTPServer(('', PORT_NUMBER), HTTP_handler)
  print('Started␣HTTP␣server␣on␣port␣' + str(PORT_NUMBER))

  # Listen forever for incoming HTTP requests
  server.serve_forever()

except KeyboardInterrupt:
  print('\nShutting␣down␣the␣web␣server\n')
  server.socket.close()
```

Príkazom `HTTPServer((ADDRESS, PORT), HANDLER)` vytvoríme HTTP server a povieme mu aby na spracovanie požiadaviek používal našu vytvorenú triedu. Keďže je ako adresa použitý prázdny reťazec, server bude počúvať na všetkých dostupných adresách počítača. Nakoniec nastavíme server aby počúval navždy. Posledné tri riadky ošetrujú prípad ak užívateľ zadá klávesovú skratku na prerušenie behu programu (`Ctrl+C` alebo `Ctrl+Z`).

B Obsah priloženého CD

```
HTTP server.zip
    - favicon.ico
    - HTTP-server.py
    - index.html
Uloha1-student.zip
    - Uloha1.gns3
    - index.html
    - dhcp.cc
Uloha1-vypracovana.zip
    - Uloha1.gns3
    - index.html
Uloha2-vypracovana.zip
    - Uloha2.gns3
    - Project files
```