VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Distribuované spojovací pole pro virtuální laboratoř počítačových sítí

Bakalářská práce

2008 Václav Bortlík

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 9. května 2008 .....

Rád bych na tomto místě poděkoval všem, kteří přispěli svými nápady, radami a připomínkami k vytvoření této práce. Nerad bych jmenovitě uváděl všechny, kterých si vážím – nechtěl bych totiž na někoho zapomenout, ale pár výjimek přesto udělám. Největší poděkování patří tvůrci Virtlabu Ing. Petru Grygárkovi Ph.D., který celou práci vymyslel a poté i koordinoval její vývoj. Bez jeho trpělivosti by tato práce nevznikla. Dále bych chtěl poděkovat Ing. Tomáši Kučerovi a Bc. Adamu Janoškovi za jejich neocenitelnou pomoc při řešení řady problémů objevených během tvorby programu.

Abstrakt

Smyslem této práce je vytvoření nového modulárního tunelovacího serveru distribuované virtuální laboratoře, neboť stávající architektura neumožňuje postupné přidávání dalších modulů pro předávání provozu jiných než ethernetových portů. Součástí této práce je vytvoření modulů pro tunelování mezi ethernetovými porty a mezi WAN porty. Konfigurace technologií spojování použitých v jednotlivých lokalitách mají být na sobě nezávislé.

Klíčová slova: virtuální síťová laboratoř, Virtlab, distribuované spojovací pole, tunelovací server, VLANy

Abstract

The purpose of this work is creation of new modular tunneling server distributed virtual laboratory, because actual architecture doesn't make possible in succesive steps to add next modules for transfer running other than ethernet ports. This work includes creation of modules for tunneling between ethernet ports and between WAN ports. Configuration of connecting technologies used in individual locations are to be independent on each other.

Keywords: virtual network laboratory, Virtlab, distributed virtual crossconnect, tunneling server, VLANs

Seznam použitých zkratek a symbolů

ASSSK	–	Automatizovaný Systém pro Správu Síťových Konfigurací
BASH	–	Bourne Again SHell
GCC	–	GNU Compiler Collection
GNU	–	GNU's Not UNIX
HDLC	–	High-Level Data Link Control
L2	–	Layer 2
PPP	–	Point-to-Point Protocol
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
VLAN	–	Virtual Local Area Network
WAN	–	Wide Area Network

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

1.1 Cíle této práce

Cílem této bakalářské práce bylo navrhnout a implementovat modulární a rozšiřitelnou architekturu pro vytváření dočasných virtuálních topologií mezi lokalitami distribuované virtuální laboratoře počítačových sítí. Jednotlivé části tunelovacího serveru měly být od sebe oddělené a navzájem na sobě nezávislé.

Jelikož jsme museli brát ohled na možnost postupného zapojování dalších modulů pro tunelování provozu převedeného z laboratorních prvků lokality rozličnými kanály, musel být celý systém navržen modulárně. Zaměřili jsme se na tunelování provozu mezi Ethernetovými porty (trunk portu a internetového portu) a pro budoucí potřebu také na tunelování HDLC/PPP mezi WAN porty. Další z modulů, který budeme v budoucnu potřebovat k tunelovacímu serveru připojit je virtuální sonda pro sledování síťového provozu[1].

Konfigurace celého mechanismu měla být nezávislá na technologiích spojování použitých v jednotlivých lokalitách virtuální laboratoře. Mezi jednotlivými lokalitami by měl být provoz předáván ve vhodně navrženém univerzálním formátu. Celý tento enkapsulační mechanismus by měl vždy určovat zdrojové a cílové rozhraní rámce a případně další volitelné parametry vztažené k přenášenému rámci.

[1] Touto problematikou se zabývá bakalářská práce Radka Nováka pod vedením Ing. Martina Milaty[1].

2 Základní princip distribuovaného spojovacího pole

2.1 Lokality

Jelikož cílem distribuovaného řešení je zpřístupnění síťových zařízení různých síťových laboratoří ve světě, je systém rozdělen na základní jednotky, které jsou nazvány *lokality*, a které reprezentují právě tyto síťové laboratoře. Tedy lokalita je místo, kde jsou fyzicky umístěna síťová zařízení, která se zpřístupňují místním uživatelům i uživatelům z ostatních lokalit. Každá lokalita má svá pravidla, jež definují, která zařízení a ve kterém čase jsou k dispozici vybraným vzdáleným lokalitám. V každé lokalitě proto běží Rezervační server, který obhospodařuje práva uživatelů a lokalit a zabezpečuje zapůjčování prvků. Dále je zde spuštěn Konzolový server, Konfigurační server a Webový server, který je určen jen pro místní uživatele[2].

2.2 Logické topologie

Jestliže chce uživatel vytvořit jistou úlohu, musí vytvořit návrh žádané síťové topologie. Ten neobsahuje přímo fyzická zařízení, ale určuje pouze typy síťových zařízení a způsob jejich propojení, proto jej nazýváme logická topologie. Teprve až je vytvořena tato logická topologie, zjišťuje se, je-li ji v daném čase možno realizovat, vyberou se vhodná síťová zařízení - a to jak lokální, tak i vzdálená a prostřednictvím rezervačních serverů se tyto prvky pro danou topologii vyhradí[3].

2.3 Princip propojení fyzických zařízení

Ke spojení zařízení v rámci více lokalit jsou určena automatická spojovací zařízení - distribuované virtuální spojovací pole, která dokážou samočinně spojit libovolná sériová rozhraní i ethernetové porty. V současné době je používán tunelovací server, který implementoval v rámci své diplomové práce Ing. Tomáš Hrabálek. Tunelovací server má vyvedeno minimálně jedno ethernetové rozhraní, které je spojeno s ethernetovým rozhraním switche Cisco Catalyst 2950. Toto rozhraní je nastaveno do trunk módu, komunikace s tunelovacím serverem je tagována a vybrané VLANy[4] se přenáší tímto rozhraním. Ostatní ethernetové porty switche přijímají netagované pakety, které jsou následně zařazeny do jednoho VLAN ID[5]. Tunelovací server má potom na starosti vzdálené propojení dvou takovýchto segmentů distribuovaného spojovacího pole, který z těchto dvou segmentů v různých VLAN udělá jeden logický ethernetový segment. Takto můžeme propojit například jeden ethernetový port přepínače v jedné lokalitě s ethernetovým portem přepínače v jiné lokalitě, přičemž tato zařízení pracují, jako by byla spojena přímo přes ethernetový

[2] Tato podkapitola je převzata z diplomové práce Ing. Tomáše Hrabálka[2].

[3] Tato podkapitola je převzata z diplomové práce Ing. Tomáše Hrabálka[2].

[4] Virtuální LAN - je logický segment sítě oddělen od fyzické topologie. Standard IEEE 802.1q definuje, že v jedné fyzické topologii může být až 4096 virtuálních sítí.

[5] VLAN ID určuje číslo virtuální sítě

bridge, jehož existence je, pro protokoly přes něj běžící, utajena[6]. V rámci jedné lokality je používáno zařízení ASSSK, které umožňuje propojení synchronních sériových portů (RS232) přepínačů. Toto zařízení funguje na první vrstvě ISO/OSI referenčního modelu – nijak se tedy nezabývá daty, která jsou přes něj přenášena, pouze slepě kopíruje napěťové úrovně z daného vstupu na daný výstup[3].

Vytvoření nového modulárního distribuovaného virtuálního spojovacího pole, jenž je cílem této práce, bude navázání na část diplomové práce Ing. Tomáše Hrabálka. Systém bude umožňovat v rámci více lokalit spojit jak virtuální ethernetové segmenty, tak propojení synchronních sériových portů v rámci více laboratoří. Pro distribuovaný systém sériových rozhraní se již nebude používat zařízení ASSSK, ale Ethernet-RS232 převodníky, které jsou ve fázi vývoje. Ethernetové porty vyvedené z převodníků budou propojeny s L2 switchem a s jedním ethernetovým rozhraním tunelovacího serveru. Konfigurace topologií ve všech lokalitách bude nezávislá na ostatních lokalitách. Tunelovacímu serveru můžeme postupně přidávat další moduly, například virtuální sondu pro sledování síťového provozu. Všechny moduly se budou přidávat tak, aby se neporušila architektura celého systému. Tunelování provozu mezi lokalitami skrz Internet bude probíhat pomocí internetového protokolu UDP jako doposud.

2.4 Aktivace topologií

K aktivaci virtuálního distribuovaného spojovacího pole je potřeba, aby po spuštění všech tunelovacích serverů byl zavolán aktivační skript. Tento skript zajistí, aby tunelovacím serverům ve všech lokalitách byly přiděleny příslušná přesměrování.

[6] Popis stávající architektury nemodulárního distribuovaného spojovacího pole je podrobně vysvětlen diplomové práci Ing. Tomáše Hrabálka[2, 8].

3 Tunelování provozu mezi lokalitami

3.1 Princip funkce

Všechny moduly, které jsme aktivovali při spuštění tunelovacího serveru naslouchají na ethernetových rozhraních podle konkrétních požadavků jednotlivých modulů[7]. Po přijetí dat musí tyto moduly zjistit zdrojové rozhraní. Toto rozhraní je textový řetězec v předem daném formátu, jenž všechny moduly musí splnit. Jak se zdrojové rozhraní určí, záleží na konkrétním modulu[8]. Podle zdrojového rozhraní se z tabulky přesměrování zjistí cílové rozhraní, které obsahuje informaci, kam se mají data přesměrovat. Cílové rozhraní se může nacházet buď v místní, nebo také v jiné lokalitě. Pokud se nachází v místní lokalitě, předáme data určenému modulu. Ten už je přepošle cíli v závislosti na své konfiguraci. Tímto se vytvoří virtuální most v rámci jedné lokality. Pokud se cílové rozhraní nachází v jedné z ostatních lokalit předáme data modulu internetovému portu. Ten je přepošle i s informacemi o přesměrování druhému tunelovacímu serveru zabalená do UDP paketu. Na druhé straně modul internetového portu přijatá data rozbalí a dále s nimi pracuje jako u prvního tunelovacího serveru. Pomocí tabulky přesměrování ze zdrojového rozhraní zjistí cílové rozhraní a odešle je na ten modul kam patří. Tímto způsobem se vytvoří virtuální most v rámci více lokalit.

Tento systém je plně modulární a do budoucna se plánuje přidávání dalších částí. Jedna z těchto částí je virtuální sonda pro sledování síťového provozu. Tato sonda bude přizpůsobena tak, aby přijímala všechen provoz, který projde tunelovacím serverem[9].

Jelikož budeme chtít fyzické prvky vzájemně propojovat jen na určitou dobu, bude se muset tabulka přesměrování modifikovat za běhu programu. K tomuto účelu je vytvořena vzdálená konzole, která je z větší části převzata z diplomové práce Ing. Tomáše Hrabálka[2]. Démon vzdálené konzole naslouchá na příslušném portu. Přístup k ní je možný přes program telnet, nebo netcat. Kvůli modulárnosti systému musela být také trochu upravena vzdálená konzole. Ta umožňuje modifikovat také tabulky, které jsou potřebné pro správný chod jednotlivých modulů. Proto bylo nutné zajistit, aby také vzdálená konzole byla nezávislá na právě připojených modulech.

3.2 Objektový návrh

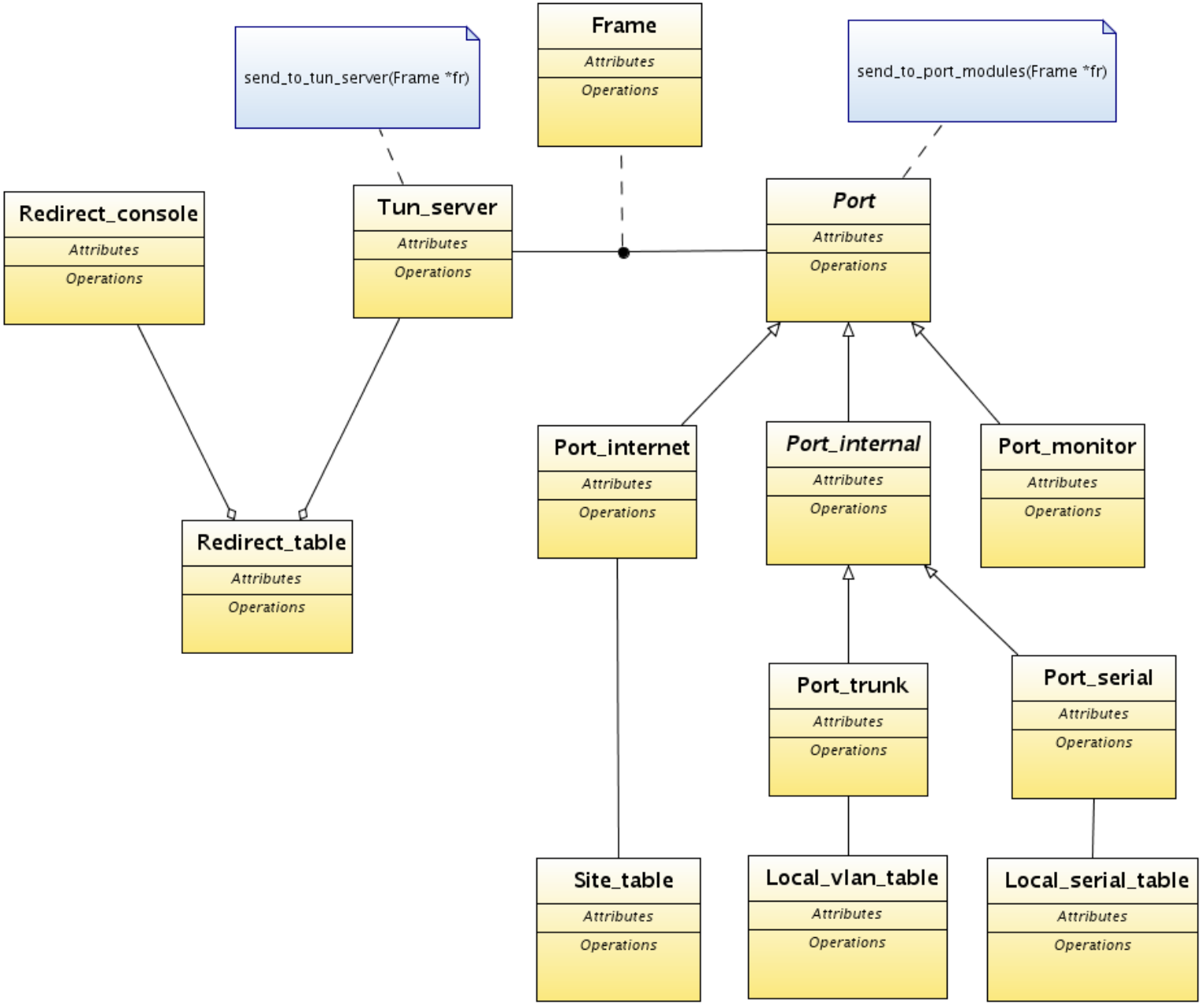
Objektový návrh je řešený v jazyce UML (grafický jazyk pro návrh programů). Na obrázku 1 je znázorněn třídní diagram tunelovacího serveru. Popis jednotlivých tříd je uveden v implementaci v kapitole 4. Na obrázcích 2, 3 a 4 jsou sekvenční diagramy, které znázorňují komunikaci objektů mezi jednotlivými moduly a třídou Tun_server. Na obrázku 2 je zobrazen průběh komunikace mezi objekty při přeposílání dat v rámci jedné lokality. V tomto případě vytvoření tunelu mezi rozhraními sériových portů. Na obrázku 3 je

[7] Např. modul trunk portu přijímá označkované ethernetové VLAN rámce dle normy IEEE 802.1q, nebo modul sériového portu, který přijímá pakety přes protokol UDP.

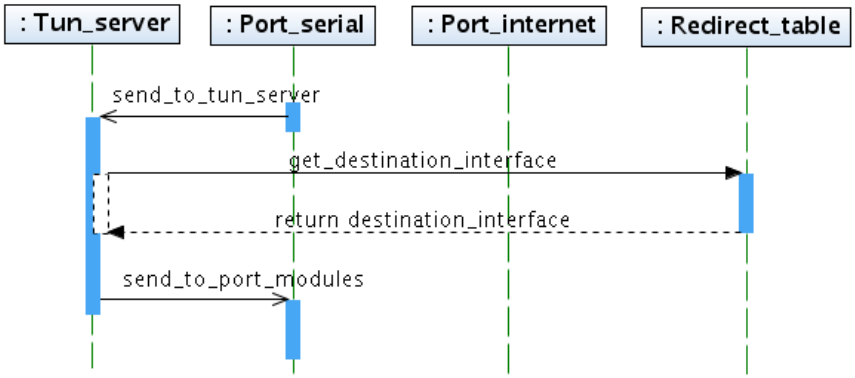
[8] K modulu trunk portu se vztahuje tabulka lokálních VLANů, ve které je každému použitému VLAN ID přiřazeno jedno rozhraní.

[9] Více informací o virtuální sondě pro sledování síťového provozu poskytne bakalářská práce Radka Nováka[1].

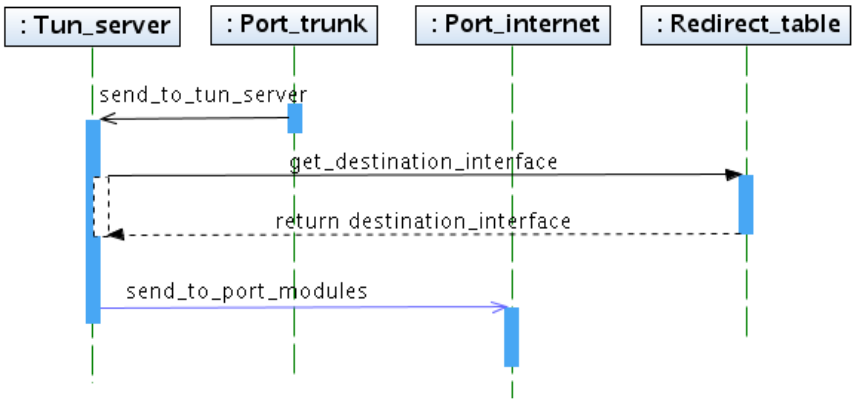
znázorněno přeposílání dat do jiné lokality a na obrázku 4 příjem dat z jiné lokality a odeslání na modul vnitřního trunk portu.



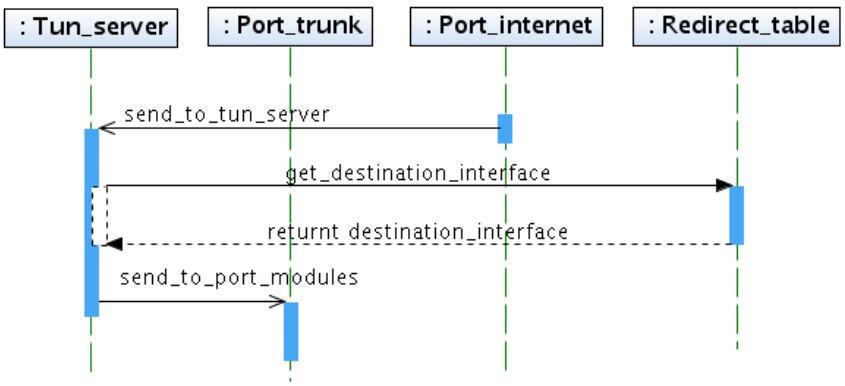
Obrázek 1: Třídní diagram



Obrázek 2: Sekvenční diagram serial-serial



Obrázek 3: Sekvenční diagram trunk-internet



Obrázek 4: Sekvenční diagram internet-trunk

4 Implementace

Program tunelovacího serveru je napsán v jazyce C/C++, z jazyka C++ využíváme prvky objektového programování. Skript pro aktivaci topologií jednotlivých lokalit je napsán v interpretru Bash. Celá práce byla vytvořena v operačním systému GNU/Linux Debian. Kompilace programu byla vyzkoušena jak na stabilní distribuci Debian 4.0 Etch[10], tak na testovací distribuci Debian Lenny 4.1[11]. Všechny zdrojové kódy jsou psané v kódování UTF-8.

4.1 Algoritmus přesměrování

V tunelovacím serveru máme vytvořenou třídu Tun_server, kterou bychom mohli nazvat řídicí třídou. Tato třída přijímá data z připojených modulů a následně je zpětně na některý z modulů přeposílá. Metodě, která tuto funkci plní, předáváme informace o zdrojovém rozhraní, abychom věděli odkud nám data přišla. Zdrojové rozhraní je řetězec v předem dohodnutém formátu, který musí dodržet všechny moduly, když přeposílají data metodě řídicí třídy. Formát musí být ve tvaru `prvek@lokalita:fyzicke_rozhrani_prvku`.

První část řetězce před zavináčem definuje název fyzického síťového prvku, které chceme propojit. Druhá část mezi zavináčem a dvojtečkou značí jméno lokality, ve které je toto zařízení připojeno. Třetí část určuje fyzické rozhraní síťového prvku. Podle tohoto celého řetězce zdrojového rozhraní vyhledáme řetězec cílového rozhraní v tabulce přesměrován. Popis této tabulky je uveden v kapitole 4.4.

Jeden z parametrů, které musíme tunelovacímu serveru předat při spuštění, je název lokality, ve které se tunelovací server nachází. Tento povinný parametr předává řídicí třídě modul internetového portu, kapitola 4.8. Název lokality, ve které se nacházíme, se porovnává se jménem lokality cílového rozhraní, tímto se zjistí, jestli se mají data přeposlat modulu internetového portu (ten je odešle jinému tunelovacímu serveru), nebo jednomu z vnitřních modulů.

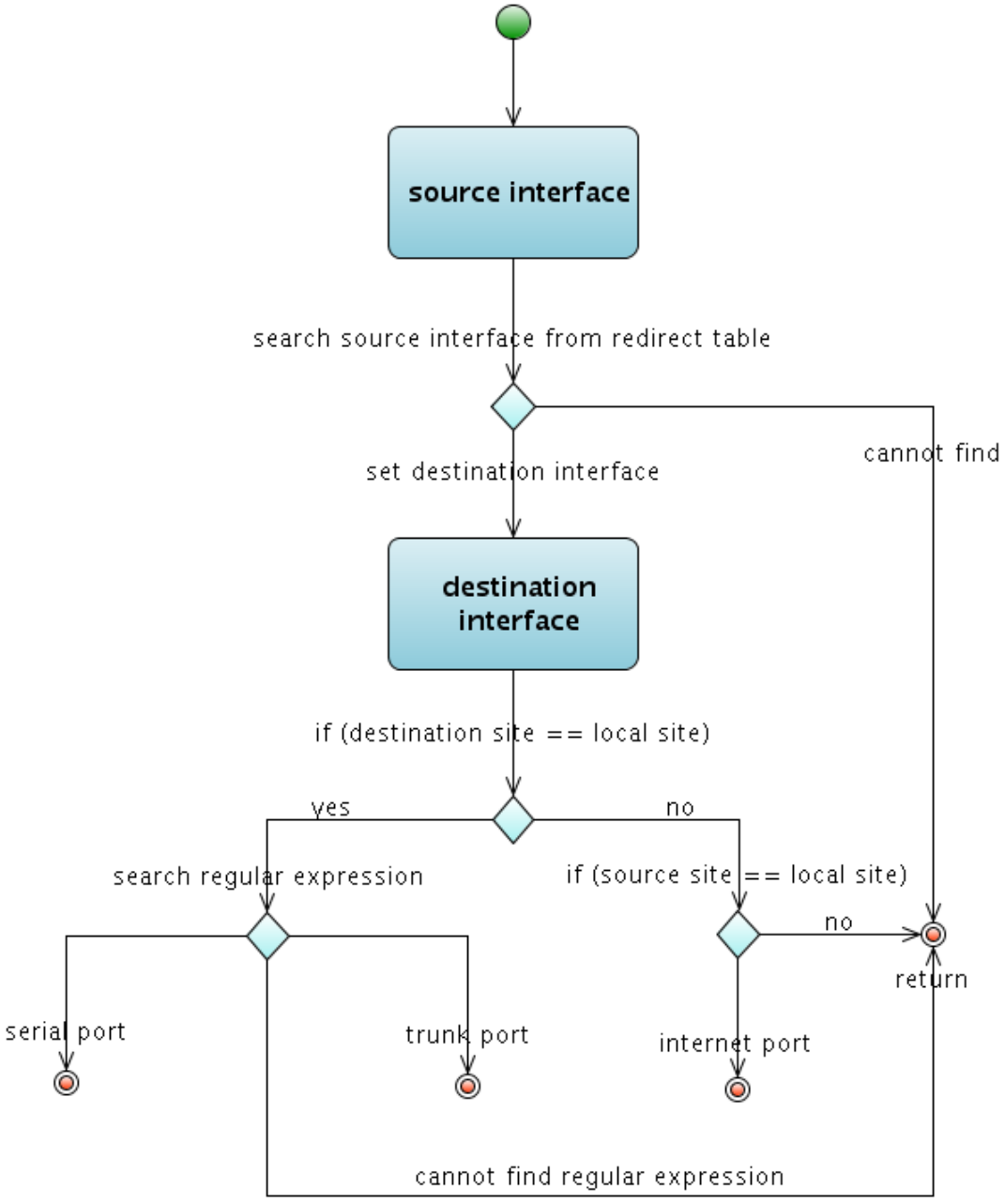
Všechny interní moduly, které registrujeme v řídicí třídě, musíme nějak rozlišit. Registrace probíhá tak, že každému modulu se přiřazuje minimálně jeden regulární výraz. Tyto výrazy se postupně porovnávají s třetí částí řetězce cílového rozhraní. Tímto zajistíme, aby se přesměrování přiřadilo správnému internímu modulu. Podrobný popis registrace modulů je uveden v kapitole 4.6.

Celý průběh mechanismu, podle kterého se rozhoduje, na jaký modul se mají data odeslat je znázorněn na stavovém diagramu obrázku 5.

4.1.1 Třída pro jednodušší komunikaci mezi objekty

Pro jednodušší komunikaci mezi jednotlivými moduly a řídicí třídou Tun_server se před zavoláním metody pro přesměrování vytvoří objekt třídy Frame. Do tohoto objektu se vkládají takové informace, které jsou důležité pro přesměrování a samozřejmě data, které

[10] verze gcc 4.1.2
[11] verze gcc 4.2.3



Obrázek 5: Stavový diagram algoritmu pro přesměrování

chceme přeposlat[12]. Nekopírují se přímo řetězce, ale ukládají se jen ukazatele na adresu řetězců vytvořených v modulu, ze kterého nám data přišla. Tento způsob byl zvolen kvůli rychlosti, protože nechceme ztrácet čas kopírováním řetězců.

Mezi všemi objekty si předáváme pouze ukazatel na instanci objektu Frame a vybíráme taková data, která zrovna potřebujeme. Případně můžeme do objektu nějaká data dodatečně doplnit.

Může nám nastat taková situace, že instanci objektu Frame dealokuje jakákoliv třída, která bude mít ukazatel na její instanci. Proto je dohodnuté pravidlo, že instanci objektu Frame bude dealokovat vždy jen ta třída, která ji vytvořila.

4.2 Aktivační skript

Pro správný chod tunelovacího serveru, musí po jeho spuštění administrátor nastavit, co a kam se má přeposílat. Toto shromáždiště informací se nazývá tabulka přesměrování, kapitola 4.4. Naplnění tabulky při spuštění serveru probíhá přes aktivační skript vytvořený v příkazovém shell interpretru Bash v Linuxu. Tento skript je vytvořený tak, že prochází soubor s tabulkou všech lokalit, která se váže na modul internetového portu – kapitola 4.8. Do privátní proměnné si uloží název lokality, na kterou právě ukazuje a postupně vyhledává v centrální tabulce přesměrování (ukázka 1) všechna přesměrování dané lokality. Tato přesměrování si ukládá do dočasného souboru, jeho název je "nazev_lokality.cmd". Až skript projde celou tabulku přesměrování v dané lokalitě, začne postupně číst jednotlivé řádky dočasného souboru a odešle je přes program netcat na vzdálenou konzoli příslušného tunelovacího serveru. Poté přejde v souboru tabulky lokalit na další řádek a opakuje cyklus dokud nenarazí na konec souboru. Tímto se naplní tabulky přesměrování všech tunelovacích serverů.

```
#
# zapis je ve tvaru "r1@ostrava:e0,r1@ostrava:e1"
#

pc1@karvina:ethernet0,pc3@ostrava:ethernet0
pc1@ostrava:ethernet0,pc2@ostrava:ethernet0
pc4@ostrava:ethernet0,rn@havirov:ethernet0
pc5@ostrava:serial0,pc6@ostrava:serial0
```

Výpis 1: Ukázka konfiguračního souboru tabulky přesměrování "stat_redirect.conf"

4.3 Vzdálená konzole

Uživatelská konzole běží jako samostatný proces, tudíž byla naprogramována pomocí vláken (standard pthreads). Démon pro přístup na řádkovou konzoli naslouchá na portu 40001, který je definován v globálním hlavičkovém souboru pro společné konstanty. Na konzoli je možné se připojit programem telnet, nebo programem netcat. Přes řádkovou konzoli můžeme konfigurovat tabulku přesměrování i tabulky, které jsou nutné pro

[12] standardně do objektu Frame ukládáme zdrojové rozhraní, cílové rozhraní, data a velikost dat

Příkaz	Argumenty	Popis
exit	-	Ukončí spojení s klientem
help	název příkazu	Vypíše obecnou nápovědu, nebo nápovědu k příkazu
show	jméno tabulky	Vypíše obsah zadané tabulky
reload	jméno tabulky	Opětovné načtení zadané tabulky
redir	zdrojové rozhraní, cílové rozhraní	Přidá záznam do tabulky přesměrování – záznam se přidá duplicitně se záměnou pořadí zdrojového a cílového rozhraní
noredir	(zdrojové — cílové) rozhraní	Vymaže záznam z tabulky přesměrování – nezáleží na tom, jestli zadáme zdrojové, nebo cílové rozhraní

Tabulka 1: Příkazy vzdálené konzole

správný chod ostatních modulů. Tabulky vnitřních modulů, například lokální tabulka VLANů pro modul trunk portu se do vzdálené konzole přidává jen v případě, když modul použijeme. Naopak tabulka lokalit, která je sdružena s modulem internetového portu, je připojena na vzdálenou konzoli trvale.

Po připojení klienta ke vzdálené konzoli se vytvoří nové vlákno a klient již běží v tomto samostatném vlákně. Počet klientů, kterých se může najednou připojit je definován v globálním hlavičkovém souboru. Standardně je nastaven na pět zároveň připojených klientů.

4.3.1 Vyřešení problému s připojením na konzoli

Jelikož jsem vzdálenou konzoli z větší části převzal z diplomové práce od Ing. Tomáše Hrabálka, opsal jsem i jeho chybu, na kterou mě upozornil až Radek Novák, když testoval tunelovací server. Jak jsem se již zmínil, po připojení klienta na konzoli se vytvořilo nové vlákno, po následném ukončení práce a odpojení ze serveru se vlákno neukončilo a běželo dále. To zapříčinilo, že po zhruba třech až čtyřech stech připojeních na konzoli se již dále nešlo připojit, protože nešlo vytvořit nové vlákno. Problém se jednoduše vyřešil tak, že vláknu se nastavily atributy pro detached vlákna – následkem toho je, že struktura vlákna se uvolní, jakmile vlákno skončí, jak je vidět ve výpise 2.

```
pthread_attr_t   attr ;
pthread_attr_init ( &attr ) ;
pthread_attr_setdetachstate( &attr , PTHREAD_CREATE_DETACHED ); // detached
int ret=pthread_create( &this->main_thread, &attr, Redirect_console::main_pthread, (void*) this );
{
   /* osetreni chyby pthread_create */
}
pthread_detach( this->main_thread );
```

Výpis 2: Přidání atributu pro detached vlákna

4.4 Tabulka přesměrování

Samotná tabulka přesměrování obsahuje dva parametry. Jedním z nich je zdrojové rozhraní a druhým je rozhraní cílové. Jako nejvhodnější datový kontejner pro uložení obou řetězců bylo asociativní pole. V C++ se tato pole nazývají mapy, hodnoty jsou uloženy ve tvaru `klíč -> hodnota`; klíč lze nazvat indexem hodnoty prvku. Oproti poli může být tento klíč libovolným objektem, v našem příkladě string. Protože se v tomto kontejneru vyhledává hodnota podle klíče, ale ne naopak, musí se prvky vkládat do pole dvakrát – z toho jednou v obráceném pořadí.

4.4.1 Synchronizace přístupu k datům v tabulce přesměrování

Protože pracujeme v prostředí, ve kterém pracuje současně více vláken zároveň, které přistupují k tabulce přesměrování, je synchronizace pro bezproblémový chod aplikace nutností. Potřebujeme zakázat, aby více než právě jeden proces číst a modifikovat sdílená data. K zajištění přístupu ke sdílenému prostředku se použil synchronizační objekt Mutex – vzájemné vyloučení. Mutex může být v jednom okamžiku vlastněn pouze jedním vláknem, což umožňuje vzájemnou koordinaci přístupu ke sdíleným zdrojů. Zde byl použit mutex, jenž je součástí knihovny vláken pthreads API `pthread_mutex`. Pro přístup do sdílené tabulky přesměrování se použil model procesů – problém čtenářů a spisovatelů. Ten připouští, aby více procesů mohlo zároveň v jednu dobu z tabulky číst. Pokud ale bude jeden proces do tabulky zapisovat, zajistí, aby žádný jiný proces nemohl do tabulky přistupovat, včetně čtení.[4, 5]

4.5 Multithreading

Aby mohly všechny použité moduly být spuštěny současně, bylo nutné přidat funkci pro souběžné zpracování více procesů. Tyto funkce řeší procesy nebo vlákna. Pro jednodušší strukturu programu byly zvoleny vlákna vyhovující standardu POSIX (Pthreads), které jsou zahrnuty ve standardní knihovně C++.

Abychom se vyhnuli opakování psaní částí zdrojových kódů při vytváření vláken, byla vytvořena abstraktní třída Port. V této třídě je zahrnuto vytvoření vláken. Tuto funkci z ní dědí všichni potomci.[6] Vlákna se vytváří v konstruktoru objektu. Při vytvoření vlákna se volá statická metoda, která již běží jako samostatný proces a předáváme jí parametr ukazatele na instanci příslušného modulu. Pomocí tohoto parametru si každý modul potom zavolá svou metodu.

Při překladu programu se musí zadat parametry `-D__USE_REENTRANT -lpthread`, které umožňují následné použití vláken při spuštění serveru.

4.5.1 Nedostatky při konstrukci objektů

Problém nastává v takovém případě, kdy použijeme v dědičnosti polymorfismus. Když vytváříme objekt, zavolá se konstruktor předka v konstruktoru potomka. V našem případě v konstruktoru předka vytváříme vlákno a předáme mu parametr `this` – instanci vytvářeného objektu. Ve statické metodě vlákna tento parametr využijeme pro zavolání

metody konkrétního objektu (využití pozdní vazby – metoda je deklarována jako virtual). V době, kdy se volá tato metoda, nemusí být objekt zcela zkonstruován a místo toho dojde k zavolání metody v abstraktní třídě předka, která je ovšem prázdná. Problém jde vyřešit jednoduše, a to tak, že po vytvoření nového vlákna se zavolá funkce sleep, čímž se proces uspí na stanovenou dobu. Po uplynutí tohoto času bude objekt s největší pravděpodobností již zkonstruován.

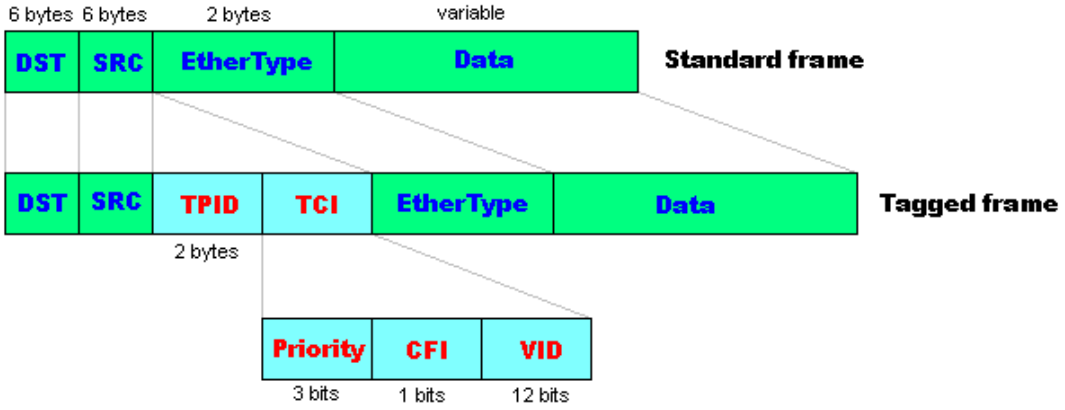
4.6 Registrace modulů

O všech modulech, které chceme aktivovat po spuštění tunelovacího serveru, musí vědět třída Tun_server, která určuje, na který modul se mají odesílat přijatá data. Tuto problematiku řeší registrace modulů. Rozlišujeme dva druhy modulů, a to pro vnější a vnitřní komunikaci. Vnější komunikace slouží k tunelování provozu mezi dvěma tunelovacími servery – modul internetového portu. Ukazatel na jeho instanci dostává řídicí třída natvrdo a pokaždé. Zato ve vnitřní komunikaci bude možno připojit ty moduly, které zrovna potřebujeme. V této práci se zaměřujeme hlavně na tunelování mezi trunk porty a tunelování mezi WAN porty. Protože jsme chtěli u vnitřních modulů zachovat modulárnost objektů, vytvořili jsme abstraktní třídu Port_internal, ze které budou všechny vnitřní moduly dědit vlastnost registrace pro třídu Tun_server.

Pro rozlišení vnitřních modulů zasíláme do konstruktoru předka předem dané parametry. Tyto parametry jsou řetězce, jenž mají podobu regulárních výrazů (příklad regulárních výrazů - `^ethernet*`, `^fast*`, `^gigabit*`, `^serial*`). Tyto regulární výrazy porovnáváme v řídicí třídě s třetí částí cílového rozhraní – čili s rozhraním fyzických prvků (příklad rozhraní fyzických prvků – `fastethernet0/1`, `serial0/0/0`). Každému regulárnímu výrazu přiřadíme příslušný ukazatel na instanci modulu, a když se nějaký regulární výraz bude rovnat s rozhraním fyzického prvku, víme, na jaký modul se data mají odeslat. Ukazatel na instanci modulu a tento řetězec regulárního výrazu ukládáme do setříděného asociativního pole. Každému modulu je možno přiřadit více regulárních výrazů.

4.7 Modul pro tunelování mezi trunk porty

Ethernetová rozhraní fyzických prvků, které chceme tunelovat, připojíme do jednoho z portů switche Cisco 2950. Každý paket vycházející trunk rozhraním tohoto switche je zabalen do takzvaného tagovaného rámce typu IEEE 802.1q. Oproti standardnímu ethernetovému rámci se liší tím, že má hlavičku rozšířenou o 4 bajty, které nazýváme tagem – rozdíl je znázorněn na obrázku 6. První tři bity určují prioritu rámců. Dále následuje CFI bit, jenž definuje, v jakém pořadí jsou bity v rámci přenášeny. Další části je dvanáctibitové číslo, které nám udává VLAN ID. Maximální počet takovýchto VLANů je 4096, opravdu využitelných je pouze 4094, protože VLAN ID 0 poukazuje na to, že rámec není označen jako VLAN. VLAN ID 0xFFF je rezervován pro budoucí použití. Poslední dva bajty určují typ protokolu již samotného datagramu ethernetového rámce.



Obrázek 6: Rozdíl mezi standardním ethernetovým rámcem a tagovaným rámcem

Trunk rozhraní switche je připojeno k jednomu ethernetovému rozhraní tunelovacího serveru a přijímá veškerý provoz na druhé vrstvě ISO OSI modelu včetně hlaviček. Tuto úlohu nám zajišťuje třída trunk portu.

Při vytváření socketu pro příjem ethernetových rámců nastavíme rodinu protokolů na `AF_PACKET`, s níž se můžeme dostat na linkovou vrstvu ISO OSI referenčního modelu. V této rodině se dají vytvořit dva druhy socketů `SOCK_RAW` a `SOCK_DGRAM`. Pro příjem syrových rámců vybereme `SOCK_RAW`, který pracuje se záhlavím i zápatím linkových rámců. Protokol nastavíme na `ETH_P_ALL`, protože při vybrání protokolu `ETH_P_8021Q` rozhraní nezachytávalo žádné rámce. Na tuto skutečnost, kterou jsem si sám ověřil, přišel již Ing. Tomáš Hrabálek ve své diplomové práci.

Pro přístup k datům v hlavičce rámce vytvořil Ing. Tomáš Hrabálek datovou strukturu vlanethhdr, která se nachází v hlavičkovém souboru "ieee8021q.h".

Dále voláme funkci `ioctl`, pomocí které zjišťujeme parametry síťového rozhraní. Pomocí atributu `SIOCGIFINDEX` dostaneme index toho rozhraní, atribut `SIOCGIFFLAGS` nám vrátí nastavení síťového rozhraní, abychom ho mohli při ukončení vrátit do původního stavu. Pomocí tohoto atributu nastavíme zařízení do promiskuitního režimu přidáním příznaku pro promiskuitní režim. Tímto se síťové zařízení nastaví tak, aby přijímalo všechen provoz bez ohledu na fyzickou adresu síťového zařízení.

V rodině protokolů `AF_PACKET` se používá adresa struktury `sockaddr_ll`. Atribut `sll_family` nastavíme na rodinu `AF_PACKET`, protokol `sll_protocol` na `ETH_P_ALL` a `sll_ifindex` nastavíme číslo síťového rozhraní. Před samotným příjmem musíme pomocí příkazu `bind` svázat adresu a port s vytvořeným socketem.

Příjem rámců se provádí pomocí zavolání funkce `recvfrom`, která čeká, dokud nepřijde nějaký ethernetový rámec. Při úspěšném přijetí se rámec uloží do předem vytvořeného bufferu a ověří, zdali typ rámce odpovídá protokolu 802.1q, v opačném případě se zahodí. Poté se přijatý rámec přetypuje na strukturu vlanethhdr, z níž vybereme VLAN ID[2, 7].

Toto číslo VLANu se vyhledá v tabulce lokálních VLANů – třída Local_vlan_table. Hodnoty v tabulce jsou pár – VLAN ID a řetězec rozhraní. Datová struktura, do které se tyto hodnoty ukládají, je setříděné asociativní pole. Jelikož se vyhledává rozhraní dle VLAN ID a také i VLAN ID podle rozhraní jsou vytvořeny dva kontejnery. Tabulka VLANů se načítá ze souboru "localvlans.conf", kterou má každý tunelovací server jedinečnou. Hodnoty jsou v souboru uloženy ve formátu `retezec_rozhrani VLAN_ID`. Příklad konfigurace souboru je v ukázce 3.

Nesmíme však zapomenout na první řádek tohoto souboru, na němž se nachází hodnota řetězce fyzického ethernetového zařízení, které bude naslouchat v promiskuitním režimu.

```
eth0
#
# zapis je ve tvaru "r1@ostrava:e0_123"
#

pc3@ostrava:ethernet0 13
pc1@ostrava:ethernet0 14
pc2@ostrava:ethernet0 15
pc4@ostrava:ethernet0 10
```

Výpis 3: Ukázka konfiguračního souboru tabulky VLANů "localvlans.conf"

Po vyhledání VLAN ID v tabulce se mu vrátí zdrojové rozhraní. Pokud je VLAN ID úspěšně nalezeno, ořízneme z datového rámce ty čtyři bajty, které nám standardní ethernetový rámec rozšiřují na tagovaný rámec. Ořezaný datový rámec, jeho velikost a zdrojové rozhraní vložíme jako parametry pro vytvoření nového objektu Frame. Ukazatele na instanci tohoto objektu poté zašleme přijímací metodě řídicí třídy Tun_server. Po úspěšném dokončení této metody musíme objekt dealokovat.

V případě, že řídicí třída rozhodne, že se mají data odeslat přes trunk port, proběhne přesně opačný proces příjmu rámců. Nejdříve řídicí třída zavolá metodu trunk portu, která má odeslání na starosti. Parametrem bude ukazatel na instanci objektu typu Frame. Z tohoto objektu vybere cílové rozhraní, podle kterého se vyhledá v tabulce VLANů, příslušné VLAN ID. Datový rámec se vloží do bufferu a rozšíří se o tag pro VLANy s vyhledaným VLAN ID. Tento buffer se poté odešle zavoláním funkce sendto.

4.7.1 Problémy s protokolem IEEE 802.1q

Jak již naznačil Ing. Tomáš Hrabálek ve své diplomové práci[2], narazil jsem také na stejný problém s odesíláním datagramu rámců větších než MTU(1500 bajtů). Tento problém nastává v případě, když ethernetový rámec otagujeme. Jeho velikost se zvětší na hodnotu MTU+4 bajtů. Podle Ing. Tomáše Hrabálka je problém ve špatné implementaci tagovacího protokolu IEEE 802.1q v jádře Linuxu. Pro ověření jsem vyzkoušel test. Pomocí příkazu ping jsem zkoušel latenci druhého počítače. Přidal jsem mu parametr pro nastavení velikosti paketu, který se bude odesílat. Tento parametr jsem nastavil na takovou hodnotu, aby velikost celkového rámce i s hlavičkou byla alespoň 1518 bajtů

Obrázek 7: Formát rámce přenášeného mezi lokalitami

(což je velikost otagovaného rámce). Zároveň jsem spustil program Wireshark pro sledování provozu na síťové kartě. Zajímala mně velikost přenášeného rámce, která byla 1514 bajtů. IP protokol si s větším rámcem poradil sám a přefragmentoval jej na dvě části. Tudíž nešlo odeslat větší rámce než nastavené MTU. V druhém příkladě jsem zavedl do jádra modul 8021q. Poté jsem pomocí utility vconfig přidal jednu VLANu a příslušnému subinterfacu přidělil IP adresu. Na druhém počítači jsem nastavil stejné údaje, ale jinou IP adresu, aby mohly spolu navzájem komunikovat. Poté jsem znovu spustil program ping. Tentokrát měl výsledný odeslaný rámec velikost 1518 bajtů. Z toho jsem odvodil, že podpora pro větší rámce je, ale až při použití utility vconfig. Tento závěr nám ale příliš nepomůže, protože utilita vconfig je pro tunelovací server nepoužitelná. Dokud nebude zvýšení MTU na 1518 bajtů implementováno v jádře, budeme muset MTU snížit na všech laboratorních zařízeních na hodnotu 1496. Zajímavé je však to, že příjem rámců větších než MTU bez utility vconfig je možné, pouze je nevyřešeno odesílání tagovaných rámců.

Další problémem, s který mi pomohl Tomáš Kučera, byl takový, že jsem musel vypnout protokol Spanning Tree na přepínači 2950, který je přímo propojen s tunelovacím serverem.

4.8 Modul pro tunelování mezi lokalitami

Pro tunelování provozu mezi lokalitami bylo nutné vytvořit mechanismus, pomocí kterého by dokázaly tunelovací servery mezi sebou komunikovat. Internetový port, který tuto funkci naplňuje, přijímá a odesílá pakety pomocí protokolu UDP. Tento bezestavový protokol byl vybrán kvůli své rychlosti a jednoduchosti. Přenášený datagram obsahuje zdrojové rozhraní, cílové rozhraní, datový rámec a velikosti všech těchto údajů. Formát rámce přenášeného mezi lokalitami je zobrazen na obrázku 7.

Vytváříme dva druhy socketů, jeden pro příjem a jeden pro odesílání dat. Oběma socketům nastavíme parametry atributů pro rodinu protokolů `AF_INET`. Typ socketů bude `SOCK_DGRAM` pro datagramový socket. Poslední parametr je protokol, který je socketem používán. Zde vložíme hodnotu `IPPROTO_UDP`. Přijímací socket poté svážeme příkazem `bind` s lokální adresou a portem tunelovacího serveru. Socket pro odesílání dat svážeme adresní konstantou `INADDR_ANY`, která se bude měnit podle toho, na jaký tunelovací server data odešleme[2, 7].

Řídicí třída Tun_server dává pokyn k odeslání dat jinému tunelovacímu serveru tak, že zavolá příslušnou metodu, které předá parametr ukazatele na instanci objektu Frame. Ten má v sobě uložené odkazy na data, která se mají odeslat. Dále se vytvoří buffer, do kterého

se podle daného formátu postupně vkládají data z objektu Frame, a která chceme zaslat druhému tunelovacímu serveru. Jsou to zdrojové rozhraní, cílové rozhraní, datový rámec a jejich velikosti. Přesný formát uložení dat je na obrázku. Tento buffer poté odešleme pomocí funkce sendto tunelovacímu serveru jiné lokality.

Pro určení IP adresy lokality slouží tabulka lokalit – třída Site_table. Tato tabulka se souběžně naplní v konstrukci objektu internetového portu při spuštění tunelovacího serveru. Tabulka lokalit data získá ze souboru "tunservers.conf", který je pro všechny lokality stejný. Formát dat uložených v tomto souboru je v přesně daném tvaru. Tento tvar je `nazev_lokality ip_adresa_lokality`. Jak vypadá konfigurační soubor můžeme shlédnout v ukázce 4. První řádek tohoto souboru je rezervován pro určení lokality, ve které je tunelovací server spuštěn. Tuto hodnotu předá modul internetového portu řídicí třídě, která s ní poté pracuje. Pokud bychom chtěli tabulku lokalit za běhu upravit, můžeme soubor jednoduše přepsat, a potom přes vzdálenou konzoli tabulku lokalit znovu načíst ze souboru pomocí příkazu `reload sites`.

```
ostrava
#
# zapis je ve tvaru "ostrava 1.1.1.1"
#

ostrava 10.0.0.10
karvina 10.0.0.20
havirov 10.0.0.30
```

Výpis 4: Ukázka konfiguračního souboru tabulky lokalit "tunservers.conf"

Samotný příjem dat se provádí příkazem `recvfrom`, ve kterém uložíme přijatá data do předem připraveného bufferu. Ten potom rozparsujeme na jednotlivé části. Jejich odkazy poté uložíme do vytvořeného objektu typu Frame. Instance objektu se odešle řídicí třídě Tun_server k dalšímu zpracování.

4.9 Modul pro tunelování mezi sériovými porty

Pro budoucí potřebu tunelování mezi WAN porty byla vytvořena třída Port_serial. Principiálně tunelování mezi WAN porty by mělo fungovat tak, že se vytvoří RS232/Ethernet převodníky, které překládají sériový synchronní přenos na ethernetový protokol. Přes ethernetový port tohoto převodníku se budou zasílat data pomocí UDP protokolu na jedno ethernetové rozhraní tunelovacího serveru. Tunelovací server má tudíž vytvořený socket, na kterém naslouchá a přijímá UDP pakety podobně jako modul internetového portu. V případě, že nám přijde UDP paket, zjistí se zdrojová adresa odesílatele a podle ní se vyhledá řetězec rozhraní v tabulce lokálních sériových portů. Tato tabulka se načítá ze souboru "localserials.conf". Příklad konfigurace je v ukázce 5. Přesný formát dat v souboru je `retezec_rozhrani IP_adresa_prvku`. První řádek tabulky je však rezervován pro nastavení IP adresy, na které má modul pro sériové porty naslouchat.

UDP datagram paketu, jeho velikost a rozhraní z kterého nám data přišla vložíme jako parametry do nově vytvořeného objektu Frame. Instanci tohoto objektu zašleme metodě řídicí třídy pro další zpracování. V případě, že řídicí třída dá pokyn k odeslání paketů,

předá modulu sériového portu instanci na objekt Frame, ze kterého vybere rozhraní, na která se mají data odeslat. Ta rovněž vybere z objektu Frame.

```
192.168.100.1
#
# zapis je ve tvaru "r1@ostrava:serial0 192.168.1.101"
#

pc6@ostrava:serial0 192.168.100.106
pc5@ostrava:serial0 192.168.100.105
```

Výpis 5: Ukázka konfiguračního souboru tabulky "localserials.conf"

5 Ukázkové příklady

Pro demonstraci správného chodu modulárního distribuovaného spojovacího pole byly vytvořeny následující úlohy. Testování probíhalo v laboratorním sálu A1028 pomocí aktivních prvků Cisco. Jako tunelovací servery byly použity počítače, na kterých běžel operační systém Linux. Na všech prvcích, které jsme tunelovali, se musela snížit hodnota MTU na 1496[9]. K tomuto jsme v Linuxu v superuživatelském režimu použili následující příkaz `ip link set <ethernetove_rozhrani_prvku> mtu 1496`. Aby se některé výpisy konfiguračních souborů zbytečně neopakovaly, rozčlenil jsem je podle lokalit. Jednotlivé typy úloh a obrázky topologií jsou rozčleněny do kapitol.

Lokalita OSTRAVA

```
192.168.100.1
#
pc5@ostrava:serial0 192.168.100.105
pc6@ostrava:serial0 192.168.100.106
```

Výpis 6: Výpis konfiguračního souboru ”localserials.conf” lokalita OSTRAVA

```
ostrava
#
ostrava 10.0.0.10
karvina 10.0.0.20
havirov 10.0.0.30
```

Výpis 7: Výpis konfiguračního souboru ”tunservers.conf” lokalita OSTRAVA

```
eth0
#
pc1@ostrava:ethernet0 14
pc2@ostrava:ethernet0 15
pc3@ostrava:ethernet0 13
pc4@ostrava:ethernet0 10
```

Výpis 8: Výpis konfiguračního souboru ”localvlans.conf” lokalita OSTRAVA

Lokalita KARVINA

```
karvina
#
ostrava 10.0.0.10
karvina 10.0.0.20
havirov 10.0.0.30
```

Výpis 9: Výpis konfiguračního souboru ”tunservers.conf” lokalita KARVINA

```
eth0
#
pc1@karvina:ethernet0 13
pc2@karvina:ethernet0 11
```

Výpis 10: Výpis konfiguračního souboru ”localvlans.conf” lokalita KARVINA

Lokalita HAVIROV

```
havirov
#
ostrava 10.0.0.10
karvina 10.0.0.20
havirov 10.0.0.30
```

Výpis 11: Výpis konfiguračního souboru ”tunservers.conf” lokalita HAVIROV

```
eth0
#
rn@havirov:ethernet0 10
rn@havirov:ethernet1 11
```

Výpis 12: Výpis konfiguračního souboru ”localvlans.conf” lokalita HAVIROV

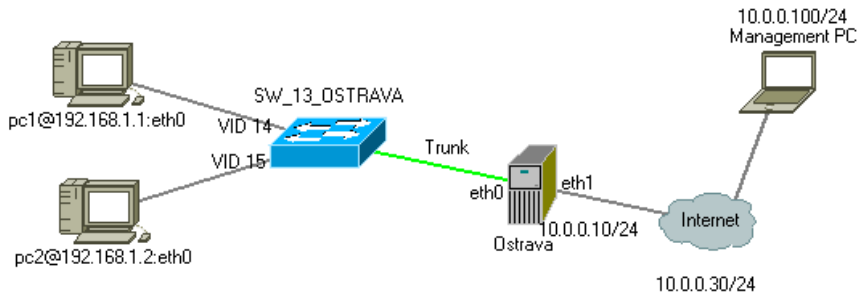
Pomocí aktivačního skriptu jsem do jednotlivých lokalit nalil přesměrování.

```
eth0
# presmerovani ethernetu v ramci jedne lokality
pc1@ostrava : ethernet0,pc2@ostrava : ethernet0
# presmerovani ethernetu v ramci vice lokalit
pc1@karvina : ethernet0,pc3@ostrava : ethernet0
# presmerovani ethernetu v ramci vice lokalit se smerovanim
pc4@ostrava : ethernet0,rn@havirov : ethernet0
rn@havirov : ethernet1,pc2@karvina : ethernet0
# presmerovani seriovych portu v ramci jedne lokality
pc5@ostrava:serial0,pc6@ostrava:serial0
```

Výpis 13: Výpis konfiguračního souboru přesměrování ”statredirect.conf”

5.1 Tunelování provozu ethernetových portů v rámci jedné lokality

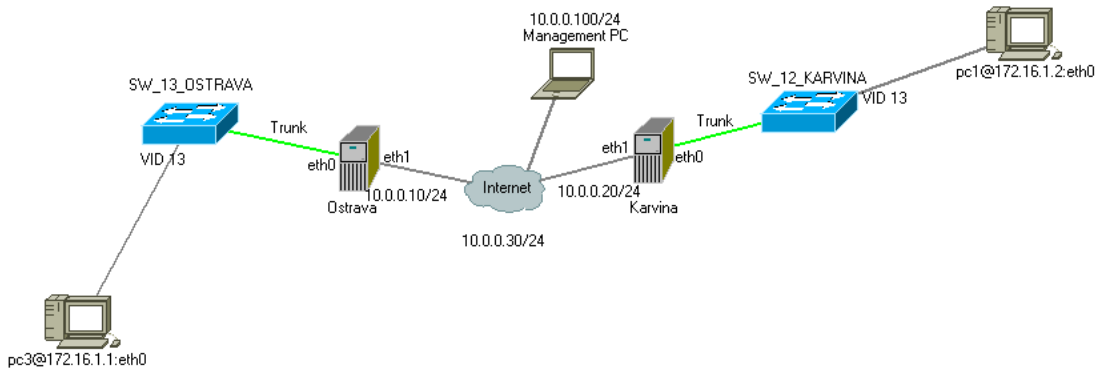
První úloha demonstruje propojení dvou počítačů s různými VLAN ID v rámci jedné lokality. Topologie této úlohy je znázorněna na obrázku 8



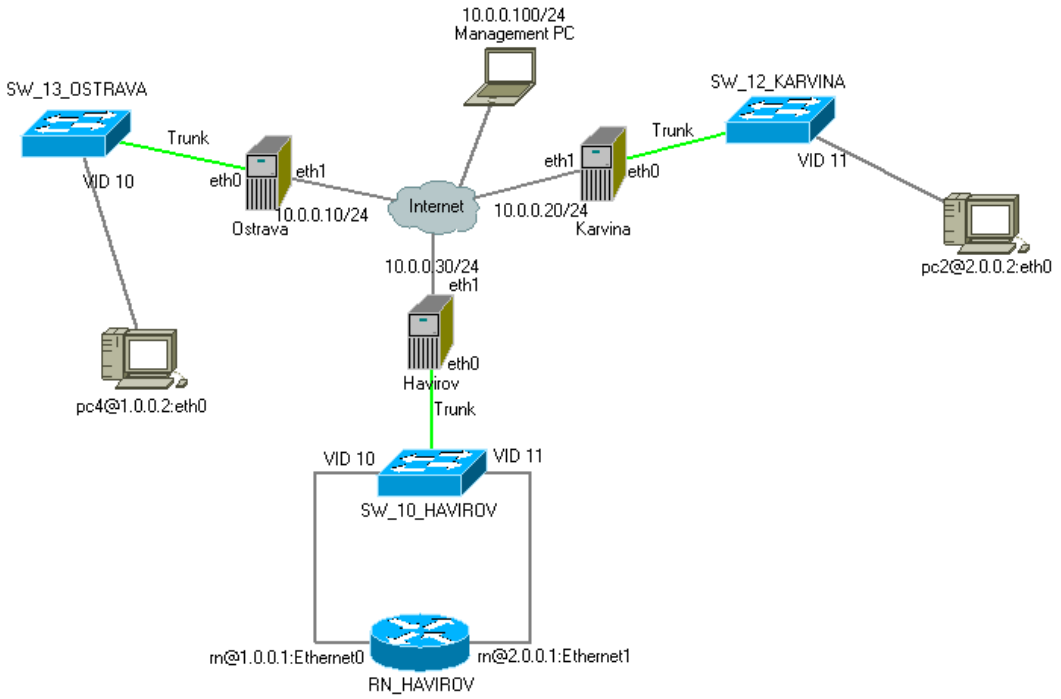
Obrázek 8: Tunelování síťového provozu v rámci jedné lokality

5.2 Tunelování provozu ethernetových portů mezi lokalitami

Na obrázku 9 a 10 je znázorněno přesměrování v rámci více lokalit. V topologii na obrázku 10 je ukázka směrování.



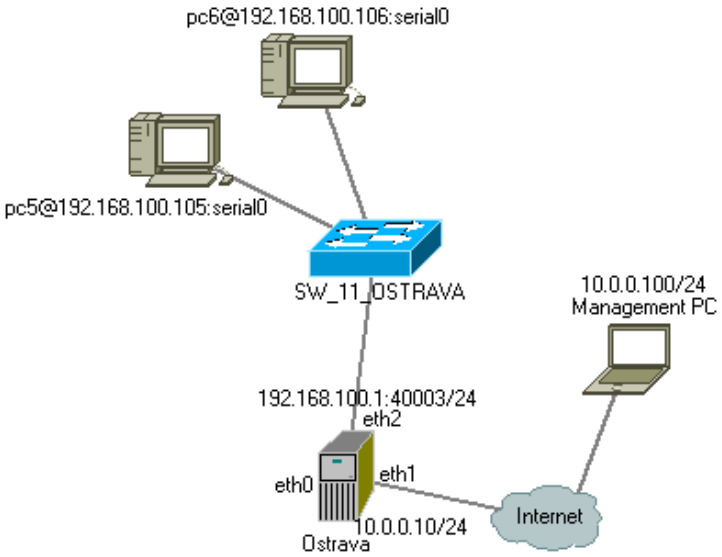
Obrázek 9: Tunelování provozu ethernetových portů mezi lokalitami



Obrázek 10: Tunelování provozu ethernetových portů mezi lokalitami se směrováním

5.3 Tunelování sériových portů v rámci jedné lokality

Další úloha demonstruje propojení sériových portů v rámci jedné lokality. Topologie této úlohy je znázorněna na obrázku 11. Tato topologie se testovala následujícím způsobem. Vytvořili jsme klienta a server. Klient zasílal data přes protokol UDP na zvoleném portu tunelovacímu serveru. Tunelovací server data přeposílal serveru, který je přijímal a zobrazoval výsledky na obrazovce.



Obrázek 11: Tunelování sériových portů v rámci jedné lokality

6 Závěr

V této práci jsem se snažil navrhnout a vytvořit novou architekturu tunelovacího serveru pro distribuovanou virtuální laboratoř počítačových sítí. Zvolil jsem takový způsob, aby tunelovací server byl v budoucnu snadno rozšiřitelný o další komponenty, aniž by vývoj nemusel začít opět od začátku. Stávající architektura to totiž neumožňuje. Návrh programu byl jeden z nejdůležitějších součástí této práce, protože od něj se vyvíjel celý průběh implementace.

Na počátku jsem řešil problém, jak vytvořit takovou aplikaci, při které by její rozšíření nemělo důsledky pro napsání zcela nového programu, nebo hromadění nepřehledného zdrojového kódu. Rozhodl jsem se proto, že využiji prvky objektového programování. Výsledkem toho byl objektový návrh, který můžeme shlédnout v kapitole 3.2 třídního diagramu na obrázku 1.

Jedna z částí, na kterou se bylo nutné pořádně zaměřit, byl obecný algoritmus přesměrování pro předávání dat jednotlivým modulům. Základní filozofie algoritmu spočívá v nezávislosti na konkrétních modulech.

Při vytváření komponent vzdálené konzole, modulu trunk portu a modulu internetového portu pro tunelování provozu skrz Internet, jsem zjistil, že mohu využít většinu zdrojového kódu diplomové práce Ing. Tomáše Hrabálka[2]. Ostatní části bylo nutné naprogramovat znovu.

Modul sériového portu, který nemohl být vyzkoušen v reálném nasazení, byl vyvinut pro budoucí použití tunelování mezi WAN porty (HDLC/PPP). Pro správnou komunikaci se musí vytvořit Ethernet-RS232 převodníky. Otestování tohoto modulu probíhalo pomocí simulovaného klienta a serveru. Klient posílal data serveru prostřednictvím UDP protokolu.

V nejbližší době se novému modulárnímu tunelovacímu serveru připojí modul pro monitorování síťového provozu[1]. Prozatím je sonda vytvořena pro původní tunelovací server Ing. Tomáše Hrabálka[2] a nic ji nebrání připojení k nové architektuře, která s ní počítá.

Dalšímu rozvoji tohoto projektu bych se chtěl věnovat v rámci diplomové práce v magisterském studiu.

Václav Bortlík

7 Reference

[1] Novák, Radek, *Bakalářská práce - Implementace sledování provozu na VLAN ve virtuální laboratoři počítačových sítí pomocí tethereal/tcpdump*, VŠB-TU Ostrava, FEI, 2008.

[2] Hrabálek, Tomáš, *Diplomová práce - Podpora vytváření virtuálních topologií ve virtuální laboratoři počítačových sítí s využitím technologie tunelování*, VŠB-TU Ostrava, FEI, 2007.

[3] Seidl, David, *Diplomová práce - Systém pro automatizovanou správu síťových konfigurací*, VŠB-TU Ostrava, FMMI, 2005.

[4] Olivka, Petr, *Operační systémy*, VŠB-TU Ostrava, FEI, http://poli.cs.vsb.cz/edu/osy/

[5] Kváš, David, *Synchronizace vláken*, http://nb.vse.cz/ zelenyj/it380/eseje/xkvad01/Vlakna.htm

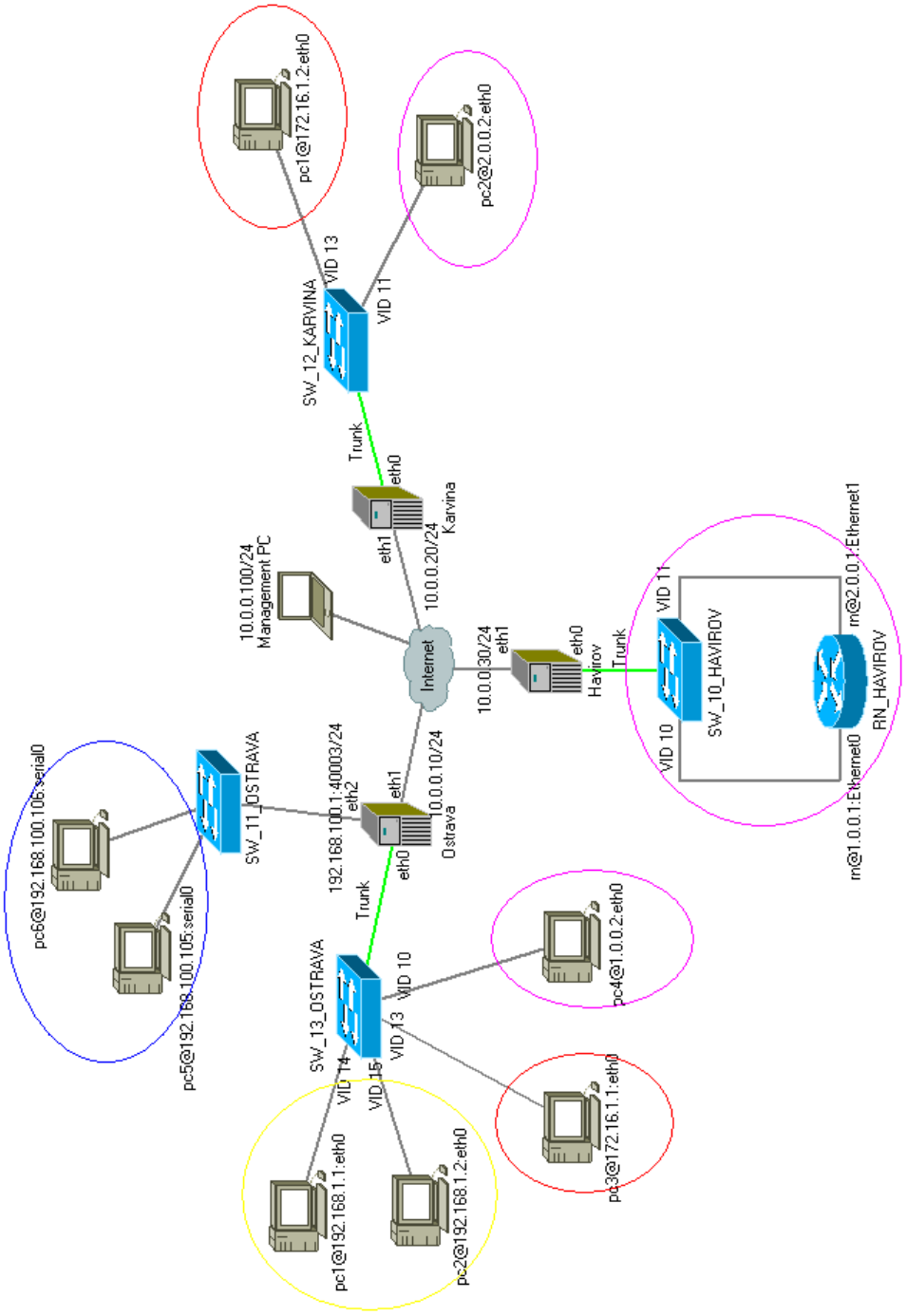
[6] Dostál, Radim, *Objektově orientované programování v C++*, http://www.builder.cz/art/cpp/cpp_oop.html

[7] Dostál, Radim, *Seriál Sokety a C/C++*, http://www.root.cz/serialy/sokety-a-cc/

[8] Grygárek, P., Milata, M., Vavříček, J., *The Fully Distributed Architecture of Virtual Network Laboratory*, In proceedings of ICETA, Stara Lesna, High Tatras, Slovakia, 2007.

[9] Linux manual pages, klíčová slova: *"ip"*, *"ifconfig"*, *"vconfig"*, *"netcat"*, *"telnet"*

A Obrázky



Obrázek 12: Kompletní topologie ukázkové sítě