VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DOKUMENTACE PROJEKTU DO PŘEDMĚTU POČÍTAČOVÉ KOMUNIKACE A SÍTĚ

Duben 2022 Matěj Konopík

1. Implementace

1. Zvolený programovací jazyk

K implementaci projektu jsem zvolil jazyk C++ z důvodu snadné manipulace s poli oproti jazyku C a dalším strukturálním výhodám. Do jazyka C# jsem bohužel neměl odvahu se pouštět, jelikož se v něm považuji za nováčka.

2. Základní implementační údaje

K vývoji jsem velice intenzivně používal knihovnu pcap.h, určenou původně pro jazyk C, ale díky zpětné kompatibilitě jsem neměl problém v C++. Knihovna velice dobře zapouzřdila funkce pro interakci s nižšími vrstvami síťového stacku. Pro parsing argumentů jsem využil knihovnu getopts a její funkci getopts_long(), jenž se volá ve smyčce pro načtení argumentů ve funkci main. Dále jsem využil knihovnu arpa/inet.h, jenž se velmi dobře hodila pro práci s daty vytažených z packetů, například její struktury pro čtení hlaviček vrstev sítě byly velice užitečné.

3. Struktura programu

Program jsem strukturoval na více funkcí, ale ve výsledku jsem nebyl spokojený s jejich délkami, jenž mohou omezit srozumitelnost a do budoucna by určitě stálo za to některé funkce lépe zdokumentovat nebo vytvořit další pomocné.

3a. Funkce main

Ve funkci main začítám definicí struktury pro načítáním argumentů pomocí getopt_long, kde definuji všekny přepínače. Dle nich se poté ve switchi nastavují flagy, jenž se poté předávají dále nebo upravují tok programu. Například samotné -i přímo volá funkci pro výpis dostupných rozhraní, print_interfaces(), jenž voláním pcap_findalldevs() tyto najde, vypíše, a ukončí program. Pokud načtení proběhne v pořádku, vytvoří síťový filtr, vytvoří pcap descriptor síťového zařízení na němž filtr nastaví a zkontroluje datalink. Tento descriptor je globální proměnná. Poté se již volá funkce pcap_loop(), která která automaticky při nalezení packetu volá callback funkci handle_packet(), jenž obsahuje hlavní logiku pro zpracování síťovích rámců, jenž od pcap_loop() obdrží.

3b. Funkce load_filtr()

Dle zadaných argumentů programu vytvoří pcap filtr pro filtrování.

3c. Funkce set_pcap_handle()

Procedurálně zavolá potřebné funkce knihovny pcap pro nastavení descriptoru zařízení.

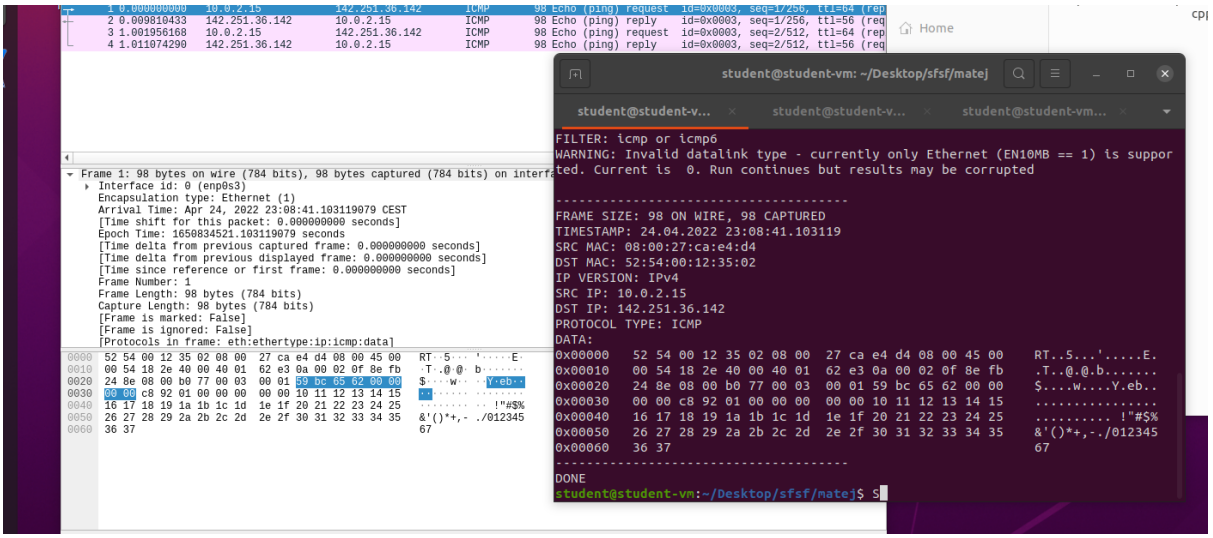
3d. Funkce handle_packet()

Tato funkce je hlavní logikou pod parsováním packetů. Každou hlavičku v síťovém stacku typecastne do odpovídající inet struktury. Tyto struktury umožňují snadný přístup k polím hlaviček bez používání offsetů.

Takto postupně vypíše potřebná data. U hlavičky ethernetu vyhledá ether_type dle nějž rozliší mezi verzí IP (4 a 6) nebo ARP packetem. Dle nich nastaví délky hlaviček síťové vrstvy a z těchto poté stejným způsobem vytáhne data o transportní vrstvě. Po rozlišení mezi protokoly transportní vrstvy vypíše další informace, například port a poté také celý paket jako data.

2. Testování

1. Pro testování jsem využil poskytnutý virtuální stroj a nástroj Wireshark, s jehož výstupy jsem výstupy svého snifferu porovnával.



Obrázek 1: snímek obrazovky z testování snifferu.

3. Přejímání pasáží kódu z internetu

V programu jsem použil kód ze stránky https://www.tcpdump.org/ pro vypisování packetů a vše jsem řádně v kódu zdokumentoval. Celkově mi tato stránka pomohla s tvorbou, jelikož se jedná o oficiální zdroj ke knihovně pcap.

4. Závěr:

Tvorba snifferu byla velice přínosná zkušenost nejen v rámci sítí a používaných knihoven, ale zároveň se mi podařilo vytvořit nástroj na úrovni splňující standardy argumentů programu, vyzkoušel jsem si také přejímání kódu z internetu , zvládnutí licencí a copyrightů a jejich zapracování do programu. Také použití jazyka C++, ač bylo minimální, bylo velice přínosné.