POČÍTAČOVÉ A KOMUNIKAČNÉ SIETE

Fakulta informatiky a informačných technológií Slovenskej technickej univerzity v Bratislave

Ak. Rok 2020/2021, zimný semester

Zadanie 1

Analyzátor sieťovej komunikácie

Autor: Pavol Belej

Cvičiaci: Ing. Dominik Macko, PhD.

Cvičenie: Štvrtok 8:00

Zadanie úlohy

Navrhnite a implementujte programový analyzátor Ethernet siete, ktorý analyzuje komunikácie v
sieti zaznamenané v .pcap súbore a poskytuje nasledujúce informácie o komunikáciách.
Vypracované zadanie musí spĺňať nasledujúce body:
1) **Výpis všetkých rámcov v hexadecimálnom tvare** postupne tak, ako boli zaznamenané v súbore.
Pre každý rámec uveďte:
a) Poradové číslo rámca v analyzovanom súbore.
b) Dĺžku rámca v bajtoch poskytnutú pcap API, ako aj dĺžku tohto rámca prenášaného po médiu.
c) Typ rámca – Ethernet II, IEEE 802.3 (IEEE 802.3 s LLC, IEEE 802.3 s LLC a SNAP, IEEE 802.3 – Raw).
d) Zdrojovú a cieľovú fyzickú (MAC) adresu uzlov, medzi ktorými je rámec prenášaný.
Vo výpise jednotlivé **bajty rámca usporiadajte po 16 alebo 32 v jednom riadku**. Pre prehľadnosť
výpisu je vhodné použiť neproporcionálny (monospace) font.

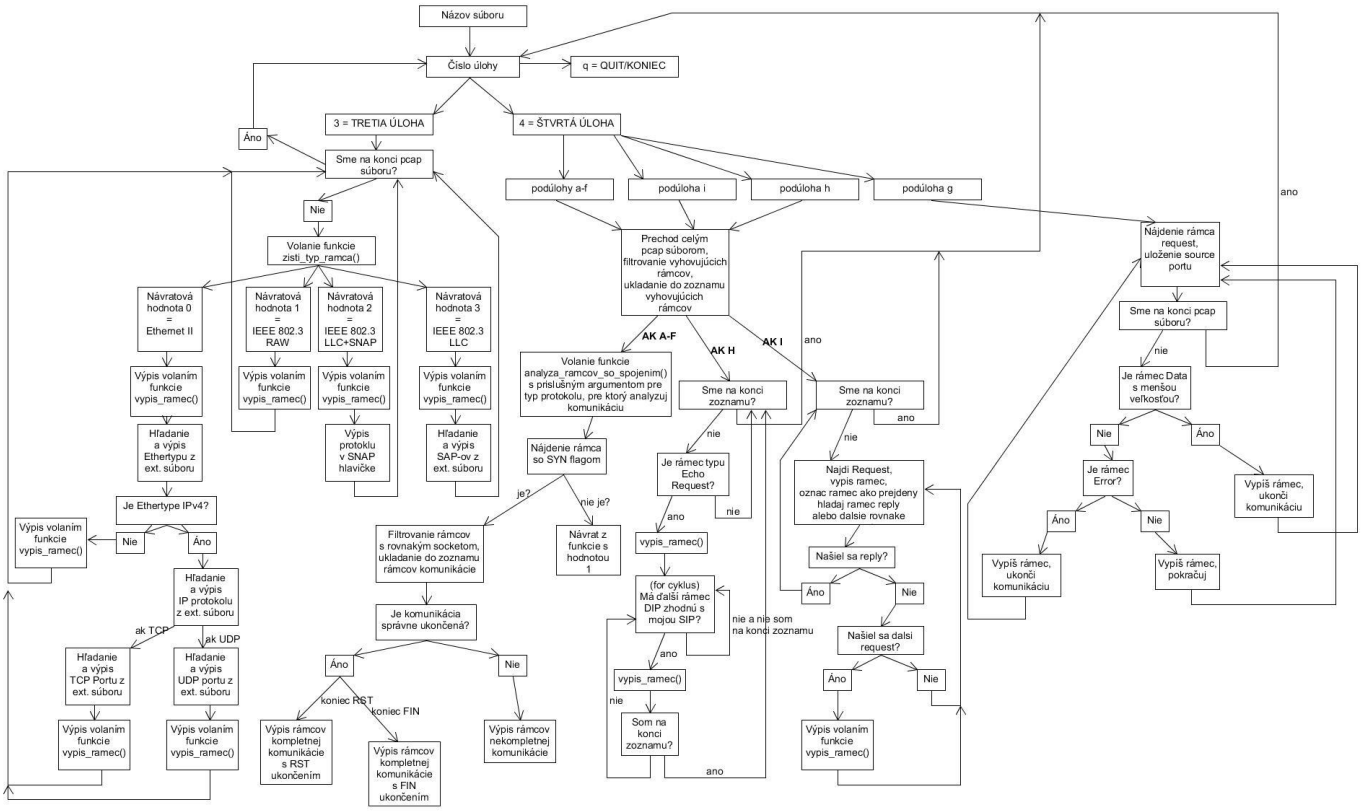
2) Pre rámce typu **Ethernet II a IEEE 802.3 vypíšte vnorený protokol**. Študent musí vedieť vysvetliť,
aké informácie sú uvedené v jednotlivých rámcoch Ethernet II, t.j. vnáranie protokolov ako aj ozrejmiť dĺžky týchto rámcov.

3) Analýzu cez vrstvy vykonajte pre rámce Ethernet II a protokoly rodiny TCP/IPv4:
Na konci výpisu z bodu 1) uveďte pre IPv4 pakety:
a) Zoznam IP adries všetkých prijímajúcich uzlov,
b) IP adresu uzla, ktorý sumárne prijal (bez ohľadu na odosielateľa) najväčší počet paketov a koľko paketov prijal (berte do úvahy iba IPv4 pakety).
IP adresy a počet poslaných paketov sa musia zhodovať s IP adresami vo výpise Wireshark -> Statistics -> IPv4 Statistics -> Source and Destination Addresses.

4) V danom súbore analyzujte komunikácie pre zadané protokoly:
a) HTTP
b) HTTPS
c) TELNET
d) SSH
e) FTP riadiace
f) FTP dátové
g) TFTP, **uveďte všetky rámce komunikácie**, nielen prvý rámec na UDP port 69
h) ICMP, uveďte aj typ ICMP správy (pole Type v hlavičke ICMP), napr. Echo request, Echo reply, Time exceeded, a pod.
i) **Všetky** ARP dvojice (request – reply), uveďte aj IP adresu, ku ktorej sa hľadá MAC (fyzická) adresa a pri ARP-Reply uveďte konkrétny pár - IP adresa a nájdená MAC adresa. V prípade, že bolo poslaných viacero rámcov ARP-Request na rovnakú IP adresu, vypíšte všetky. Ak sú v súbore rámce ARP-Request bez korešpondujúceho ARP-Reply (alebo naopak ARPReply bez ARP-Request), vypíšte ich samostatne.

Bloková schéma riešenia

Nasledujúca schéma popisuje postup interakcie užívateľa s programom a následný postup pri riešení jednotlivých úloh zadania



Mechanizmus analyzovania protokolov na jednotlivých vrstvách

Pomocné funkcie

int zisti_real_dlzku()

Funkcia slúži na vypočítanie veľkosti/dĺžky rámca, ako sa prenáša po médiu. Keďže winpcap knižnica a jej patriace funkcie ako aj štruktúra *pktheader* poskytujú iba dĺžku, ktorá sa dá spracovať, teda významné bajty. Chýba tam však header checksum a rovnako tak môže byť veľkosť menšia než 64 B, čo v skutočnosti nikdy nie je. Táto funkcia teda prepočíta veľkosť rámca tak, ako sa prenáša po médiu, berúc ohľad na tieto skutočnosti.

int zisti_typ_ramca()

Funkcia slúži na získanie informácie, o aký rámec ide: Ethernet II, IEEE 802.3 RAW, IEEE 802.3 LLC alebo IEEE 802.3 LLS+SNAP. To robí podľa porovnávania hodnôt na príslušnom mieste v rámci s predpokladanými hodnotami, prislúchajúcimi k jednotlivým typom. Pre rámec Ethernet II vracia funkcia hodnotu 0, pre IEEE 802.3 RAW hodnotu 1, pre IEEE 802.3 LLC+SNAP hodnotu 2 a pre IEEE 802.3 LLC hodnotu 3.

void vypis_obsah_ramca()

Funkcia vypisuje jednotlivé bajty rámca v nami zvolenom formáte, a to takom, že na jeden riadok vypíše maximálne 16 bajtov. To je docielené za pomoci modula po delení šestnástimi, kedy ak je výsledok takej operácie 0, vloží znak nového riadku. Používa argument *dlzka_ramca*, podľa ktorého pomocou for cyklu bajty vypisuje.

void vypis_mac_adresy()

Funkcia iba vypisuje MAC adresy rámca, ktorý dostane ako parameter. Formátovaný výpis do súboru vykoná pomocou prehľadania príslušných bajtov v rámci, ktoré prislúchajú pre MAC adresy.

void vypis_ramec()

Funkcia vypisuje informácie o rámci: jeho dĺžku, vypočítanú dĺžku prenášanú po médiu, typ rámca a volaním funkcie *vypis_mac_adresy()* aj cieľovú a zdrojovú MAC adresu.

Štruktúry

V riešení používame rôzne štruktúry, pre uľahčenie práce a efektívne uchovávanie údajov. Tieto štruktúry slúžia na:

- Uchovávanie čísiel a názvov ethertypov, IP protokolov, DSAP a SSAP polí, TCP a UDP portov,
- Uchovávanie informácií o ARP, resp. ICMP rámcoch pri analýze rámcov príslušných typov v bode 4h, resp. 4i. Uchováva sa poradové číslo v súbore, veľkosť rámca jeho jednotlivé bajty, pre ICMP rámce typ správy a pre ARP rámce typ rámca,
- Uchovávanie poradového čísla, veľkosti a jednotlivých bajtov rámcov pri analýze komunikácií so spojením v bodoch 4a-4f
- Uchovávanie konkrétnej komunikácie so spojením, a teda zdrojového a cieľového portu, rámca s flagom SYN a poľom štruktúr vyššie spomenutého typu jednotlivých rámcov, ktoré sú súčasťou komunikácie.

Knižnice

Pri riešení sme potrebovali zahrnúť 2 knižnice, a to knižnicu *stdio.h* pre načítavanie vstupu od používateľa a knižnicu *pcap.h* na prácu s rámcami. V rámci knižnice na práce s rámcami v súbore .pcap sme použili funkcie:

- pcap_open_offline() na otvorenie súboru
- pcap_next_ex() na prechádzanie jednotlivých rámcov po poradí

Rovnako sme okrem funkcií použili aj typy pcap, a to *pcap_pkthdr** na pridelenie hlavičiek rámcov, *pcap_t* handle* na umožnenie čítania jednotlivých rámcov so súboru a do istej časti aj *PCAP_ERRBUF_SIZE* ako parameter pre nutný argument do funkcie *pcap_open_offline()*.

Analýza

Vo väčšej časti riešenia sme pracovali s pozíciami jednotlivých bajtov, aby sme určili kľúčové charakteristiky konkrétneho rámca. S pomocou programu Wireshark, ktorý sme sa snažili do istej miery napodobiť, resp. implementovať podobnú funkciu sme porovnávali naše výsledky s očakávanými výsledkami výpisov a analýzy.
Na 13. a 14. bajte v rámci (12. a 13. ak číslujeme od nuly) sa nachádza dvojbajtová hodnota, hovoriaca o type rámca. Pomocou bitového posunu 13. bajtu o osem bitov a bitového operátora OR sme dostali hexadecimálnu hodnotu, ktorú sme porovnali s očakávanými hodnotami podľa typov rámcov, a tak určili typ konkrétneho rámca. Ak je číslo väčšie ako 640 hexadecimálne, resp. 1600 desiatkovo, vieme, že ide o rámec Ethernet II, inak ide nie o hodnotu Ethertype, ale o dĺžku. Pri tejto dĺžke analýzou nasledujúcich bajtov vieme zistiť o aký typ IEEE 802.3 ide.

Podobne ako pri určovaní typu rámca sme postupovali aj pri určovaní vnorených protokolov na 2. a 4. vrstve, portov pri TCP a UDP protokoloch. Pri analýzach komunikácií konkrétnych protokolov sme pomocou porovnávania bajtov na príslušných miestach v rámci a očakávaných, resp. požadovaných hodnôt, určovali a filtrovali rámce komunikácie konkrétnych protokolov ako HTTP, HTTPS, TELNET, SSH, FTP riadiace, FTP dátové. Tieto protokoly komunikujú pomocou špecifických flagov na nadviazanie a ukončenie spojenia. Tieto flagy sme taktiež vyhľadávali a porovnávali tak, aby sme našli vyhovujúcu komunikáciu, ktorá okrem flagov musela spĺňať aj požiadavky zhodnosti portov a IP adries tak, aby komunikáciu bolo možné považovať za kompletnú, teda korektne nadviazanú aj ukončenú. Ak sa vyskytla komunikácia, ktorá tieto požiadavky nespĺňala a išlo teda o nekompletnú komunikáciu, takúto sme pri prvom výskyte taktiež, podľa zadania, vypísali.
Pre skoro všetky úlohy patriace pod úlohu 4, konkrétne úlohy 4a-4f,4h,4i; sme použili rovnaký princíp dvojstupňovej analýzy. Tá spočíva v tom, že si rámce v súbore najskôr podľa daného protokolu prefiltrujeme a uložíme do príslušného poľa vyhovujúcich rámcov. Toto pole následne v druhom kroku analýzy prechádzame, označujeme si prejdené rámce a hľadáme, už podľa kritérií špecifických pre danú úlohu, vyhovujúce rámce.
Pre úlohu 4g sme využili vlastnosti komunikácie TFTP rámcov. Vieme totiž, že komunikácia začína rámcom s opcode-om pre request. Rámce, ktoré nasledujú po tomto rámci s request-om, sú TFTP rámcami a vyhovujú podmienkam pre zaradenie do komunikácie (zhodné IP adresy) môžu teda byť buď rámce obsahujúce dáta alebo rámce s opcode-om acknowledgment. Komunikácia pokračuje, až kým nenarazíme na rámec s opcode-om ERROR alebo na dátový rámec, ktorý má veľkosť menšiu než obvyklú (bežne 512 B, 558 B s hlavičkou). Takýto rámec signalizuje koniec komunikácie, po ktorom by mal nasledovať ešte jeden rámec s opcode-om acknowledgment.

Štruktúra externých súborov

Rozhodli sme sa pre štruktúru externých súborov takú, ktorá obsahuje čo najmenej informácií, s ktorými zároveň bez problémov dokážeme určiť výskyt Ethertypov pre rámce Ethernet II,DSAP a SSAP polí pre rámce typu IEEE 802.3 LLC, IP protokolov, resp. portov na TCP alebo UDP protokoloch.

Externé súbory teda obsahujú číslo protokolu, uvedené v hexadecimálnom formáte, nakoľko práve v hexadecimálnom formáte sú uložené aj čísla protokolov v rámcoch načítaných zo súboru; rovnako ako názov daného protokolu.

Ako príklad uvádzame štruktúru externého súboru s názvom *ipprotokoly.txt*, slúžiaceho na použitie pri analýze protokolov vnorených do IP paketu.

Príklad štruktúry:

01 ICMP

02 IGMP

06 TCP

09 IGRP

11 UDP

2F GRE

32 ESP

33 AH

39 SKIP

58 EIGRP

59 OSPF

73 L2TP

Pri načítavaní zo súboru sa teda najprv načíta číslo protokolu v hexadecimálnom formáte ako hexadecimálne číslo pomocou „%X“, následne dočítame všetky znaky až do konca riadku, označeného „\n“ pomocou „%[^\n]“. Tieto informácie sme si ešte pred spustením programu uchovali v príslušných dátových štruktúrach, konkrétne to predstavovali polia štruktúr *struct*, ktoré sme si nastavili tak, aby uchovali ako číslo protokolu, tak aj jeho názov v poli znakov.

Používateľské rozhranie

Keďže sme zvolili jazyk C ako jazyk implementačného prostredia, naše riešenie neposkytuje pre používateľa grafické používateľské rozhranie. Všetka interakcia používateľa a nášho programu sa uskutočňuje cez konzolu. Pri typickom využití programu ako sieťového analyzátora, by po spustení na vypísanú výzvu udelil používateľ **názov súboru**, s ktorým chce pracovať. V ideálnom prípade, by sa tento súbor mal nachádzať v priečinku, kde sa nachádza aj zdrojový súbor programu, ak nie, tak je nutné zadať absolútnu cestu v správnom formáte. Je nutné podotknúť, že nami zvolené riešenie neposkytuje kontrolu správnosti zadaného vstupu v podobe názvu pcap súboru, čiže sa vyžaduje obozretnosť používateľa pri zadávaní, prípadne spätná kontrola správnosti vstupu.
Po zadaní názvu súboru sa opäť zobrazí výzva na bližšie špecifikovanie žiadanej akcie, ktorá sa má so súborom vykonať. Pre **ukončenie programu** je nutné stlačiť tlačidlo ***q***, inak používateľ špecifikuje, ktorú z úloh si žiada vykonať, stlačením príslušného čísla úlohy. Pozn.: Úlohy 1 a 2 sú plne obsiahnuté v úlohe 3, a teda sa samostatne nevykonávajú.
Pri možnosti vykonania úlohy 4 je taktiež nutné zadať podúlohu, ktorú si používateľ želá vykonať, stlačením príslušného malého písmena, označujúceho danú podúlohu. Ako príklad uvedieme ukážkovú sekvenciu zadaných vstupov pre vykonanie úlohy 4, podúlohy s označením *i* nad súborom „*trace-8.pcap*“: trace-8.pcap -> 4 -> i, prípadne trace-8.pcap -> 4i.
Po vykonaní akejkoľvek z úloh sa vstupné menu zobrazí znova a program sa teda úspešne skončí až po stlačení ukončovacieho tlačidla ***q***.

Implementačné prostredie

Ako implementačné prostredie sme si zvolili Integrated Development Environment, teda IDE, konkrétne Visual Studio Enterprise 2019 a programovacím jazykom je jazyk C. Do implementačného prostredia je zahrnutá knižnica pre spracovanie sieťovej premávky *Winpcap*. Z tejto knižnice sme čerpali nástroje pre otváranie a spracovanie zachytených súborov vo formáte .pcap.
Implementácia v jazyku C bola zvolená predovšetkým s prihliadnutím na skúsenosti v tomto programovacom jazyku a možnosti práce s prideľovaním pamäte a prácou s bajtami.