Slovenská Technická Univerzita v Bratislave
Fakulta Informatiky a Informačných Technológií

Počítačové a komunikačné siete

Zadanie 1 – Analyzátor sieťovej komunikácie

Marek Adamovič

Cvičenie: Piatok 10:00 – 11:40
Ing. Peter Kaňuch

2020/2021

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 príjemcu) 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.

Vo všetkých výpisoch treba uviesť aj IP adresy a pri transportných protokoloch TCP a UDP aj porty komunikujúcich uzlov. V prípadoch komunikácií so spojením vypíšte iba jednu kompletnú komunikáciu – obsahuje otvorenie (SYN) a ukončenie (FIN na oboch stranách alebo ukončenie FIN a RST alebo ukončenie iba s RST) spojenia a aj prvú nekompletnú komunikáciu, ktorá obsahuje iba otvorenie spojenia. Pri výpisoch vyznačte, ktorá komunikácia je kompletná. Ak počet rámcov komunikácie niektorého z protokolov z bodu 4 je väčší ako 20, vypíšte iba 10 prvých a 10 posledných rámcov tejto komunikácie. (Pozor: toto sa nevzťahuje na bod 1, program musí byť schopný vypísať všetky rámce zo súboru podľa bodu 1.) Pri všetkých výpisoch musí byť poradové číslo rámca zhodné s číslom rámca v analyzovanom súbore.

5) Program musí byť organizovaný tak, aby čísla protokolov v rámci Ethernet II (pole Ethertype), IEEE 802.3 (polia DSAP a SSAP), v IP pakete (pole Protocol), ako aj čísla portov v transportných protokoloch boli programom načítané z jedného alebo viacerých externých textových súborov. Pre známe protokoly a porty (minimálne protokoly v bodoch 1) a 4) budú uvedené aj ich názvy. Program bude schopný uviesť k rámcu názov vnoreného protokolu po doplnení názvu k číslu protokolu, resp. portu do externého súboru. Za externý súbor sa nepovažuje súbor knižnice, ktorá je vložená do programu.

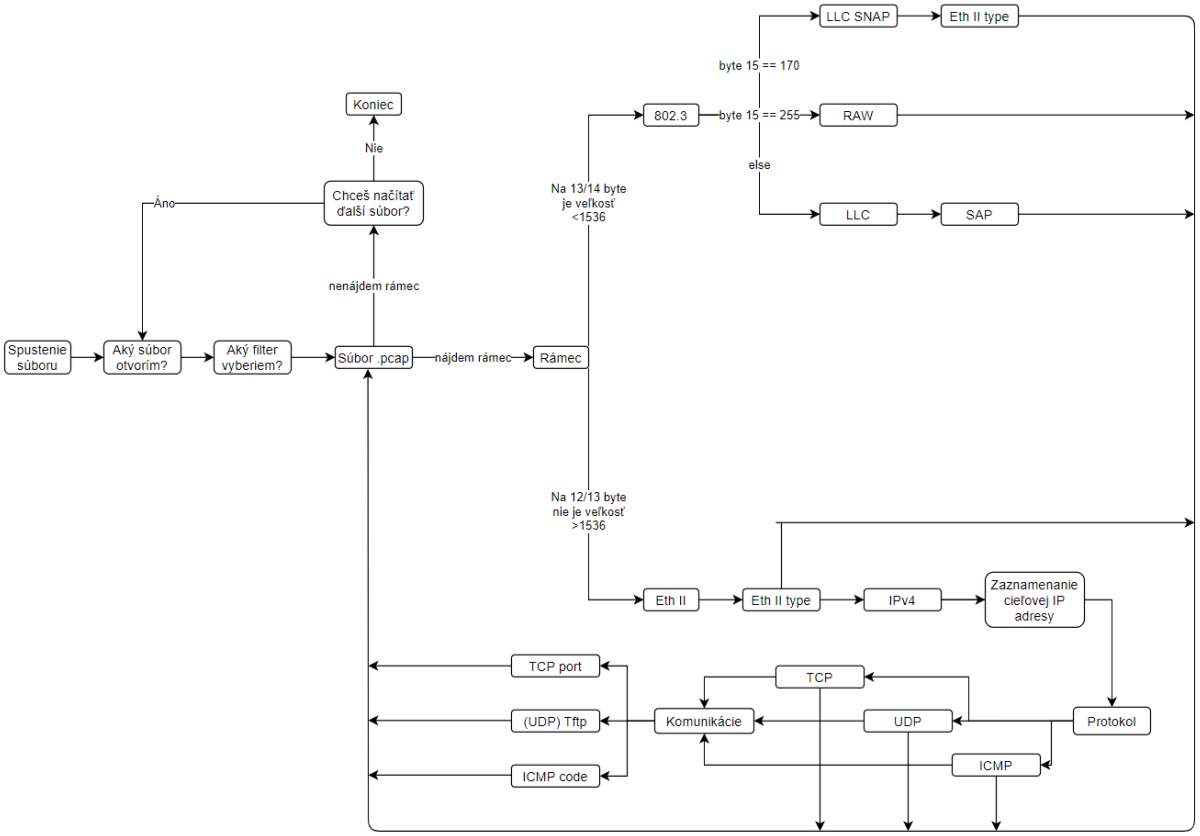
6) V procese analýzy rámcov pri identifikovaní jednotlivých polí rámca ako aj polí hlavičiek vnorených protokolov nie je povolené použiť funkcie poskytované použitým programovacím jazykom alebo knižnicou. Celý rámec je potrebné spracovať postupne po bajtoch.

7) Program musí byť organizovaný tak, aby bolo možné jednoducho rozširovať jeho funkčnosť výpisu rámcov pri doimplementovaní jednoduchej funkčnosti na cvičení.

8) Študent musí byť schopný preložiť a spustiť program v miestnosti, v ktorej má cvičenia. V prípade dištančnej výučby musí byť študent schopný prezentovať podľa pokynov cvičiaceho program online, napr. cez Webex, Meet, etc. V danom týždni, podľa harmonogramu cvičení, musí študent priamo na cvičení doimplementovať do

funkčného programu (podľa vyššie uvedených požiadaviek) ďalšiu prídavnú funkčnosť.

Blokový návrh



Mechanizmus analyzovania protokolov na jednotlivých vrstvách

Ako vidíme na blokovom návrhu, algoritmus na druhej vrstve je veľmi jednoduchý .. pozriem sa na 13 a 14 byte, skontrolujem, či sa tu nachádza číslo väčšie/menšie ako 1536 a podľa toho daný rámec zaradím, či patrí do skupiny 802.3 (číslo je menšie ako 1536, tým pádom predstavuje veľkosť) alebo do eth II (číslo je väčšie ako 1536 tým pádom predstavuje eth type). Táto logika sa rieši vo funkcii ries_ramec_1().

```
if(kontrolna_velkost < 1536){                    //zistujem podla toho, ci je velko
    if(filter)
        return;
    fprintf(vystupny_subor, "Jedna sa o skupinu 802.3 ");
    ries_802_1(data_ramca, vystupny_subor, ieee, eth);
}
else{
    if(!filter)
        fprintf(vystupny_subor, "Jedna sa o ethernet II\n");
    ries_eth2_1(data_ramca, vystupny_subor, eth, 0, ipv4, destination_adresy,
}
```

Pri 802.3 ďalej sledujem 15ty byte, ktorý bude mať maximálnu veľkosť (FF) pri RAW a inak bude predstavovať SAP, teda Service Access Point (napríklad SNAP, LAN a podobne). Toto všetko sa rieši vo funkcii ries_802_1(). V prípade, že mám SNAP, tak ešte musím ísť pozrieť jeho eth type, ktorý sa nachádza na 21. a 22. byte (posielam do funkcie ries_eth2_1).

```
void ries_802_1(const u_char *data_ramca, FILE *vystupny_subor, EXTERNY *ieee, EXTERNY *eth){
    while(ieee != NULL){
        if(data_ramca[14] == ieee->value)
            break;
        ieee = ieee->dalsi;
    }
```

Pri eth II sa viem priamo dostať k eth type cez 13. a 14. byte (ak sa nejedná o SNAP). Ak bude hodnota na týchto byteoch zodpovedať hodnote určenej pre IPv4 protokol, tak pokračujem v analýze na ďalšej vrstve. Riešim vo funkcii ries_eth2_1 a IPv4 protokoly posielam do funkcie ries_ipv4.

```
void ries_eth2_1(const u_char *data_ramca, FILE *vystupny_subor, EXTERNY *eth, int snap, EXTERNY *ipv4,
DESTINATION_ADRESA *destination_adresy, EXTERNY *tcp, EXTERNY *icmp, EXTERNY *udp, char filter, int cislo_ramca, int velkost, KOMUNIKACIE *zaciatok_komunikacii){
    int eth_value;
    if(snap)
        eth_value = data_ramca[20] * 256 + data_ramca[21];
    else
        eth_value = data_ramca[12] * 256 + data_ramca[13];

    while(eth != NULL){
        if(eth_value == eth->value)
            break;
        eth = eth->dalsi;
    }
```

Na IPv4 nie je typ portu vyššej vrstvy vždy na rovnakom mieste, musím sa k nemu dopočítať. Zoberiem veľkosť ip headeru z druhej polovice 15. byteu a prenásobím ho štyrmi. Nazvem číslo, ktoré mi vyšlo číslom velkost_ip_headeru, budem ho potrebovať pre neskoršiu analýzu. Protokol na IPv4 zistím priamo na 24. byte. Používam podobnú logiku ako v predošlých funkciách.

```
void ries_ipv4(const u_char *data_ramca, FILE *vystupny_subor, EXTERNY *ipv4, DESTINATION_ADRESA *destination_adresy,
EXTERNY *udp, char filter, int cislo_ramca, int velkost, KOMUNIKACIE *zaciatok_komunikacii){
    DESTINATION_ADRESA *predosli = destination_adresy;
    int velkost_ip_headeru = (data_ramca[14] % 16) * 4;
    int ipv4_value = data_ramca[23];

    while(ipv4 != NULL){
        if(ipv4_value == ipv4->value)
            break;
        ipv4 = ipv4->dalsi;
    }
```

Ak sa chcem dostať na začiatok protokolu v nasledujúcej vrstve, použijem číslo velkost_ip_headeru, ktoré som si vypočítal skôr, pripočítam k nemu 15 a vyjde mi číslo byteu, na ktorom začína ďalšia vrstva, teda pri indexovaní od 0 by to vyzeralo takto:

data_ramca[14 + velkost_ip_headeru]

Túto informáciu posielam podobným spôsobom do ďalších funkcií, v závislosti od voľby filtra od užívateľa.

```
if(ipv4 != NULL){
    if(ipv4->value == 6 && filter > 0 && filter < 7)
        ries_tcp(data_ramca, data_ramca + 14 + velkost_ip_headeru, vystupny_subor, tcp, filter, cislo_ramca, velkost, zaciatok_komunikacii);
    if(ipv4->value == 17 && filter == 7)
        ries_udp(data_ramca, data_ramca + 14 + velkost_ip_headeru, vystupny_subor, udp, filter, cislo_ramca, velkost);
    if(ipv4->value == 1 && filter == 8)
        ries_icmp(data_ramca, data_ramca + 14 + velkost_ip_headeru, vystupny_subor, icmp, filter, cislo_ramca, velkost);
}
```

Ak som sa dostal na ICMP, tak hneď na prvom byte nájdem typ správy, ktorú predstavuje (echo, echo reply, ...). Nachádzam sa vo funkcii ries_icmp().

```
while(icmp != NULL){
    if(vnorene_data_ramca[0] == icmp->value)
        break;
    icmp = icmp->dalsi;
}
if(icmp == NULL)
    fprintf(vystupny_subor, "Protokol je ICMP\nNEZNAMY\n");
else
    fprintf(vystupny_subor, "Protokol je ICMP\n%s\n", icmp->name);
```

Ak sa nachádzam na začiatku TCP, riadim sa podľa portov na prvých 4 byteoch (stačí, ak identifikujem aspoň jeden a podľa toho viem typ). Vo funkcii ries_tcp() sa nachádzam kvôli tomu, že užívateľ očakáva výstup v komunikáciach podľa filtru, preto z tejto funkcie pokračujem do komunikacia_tcp(), ktorá následne používa pomocné funkcie vytvor_komunikaciu() a najdi_komunikaciu(). Pre vyznanie sa v komunikáciach používam flagy z jednotlivých paketov a istý stav, v ktorom sa nachádzam (začal som komunikáciu? začal som ukončovať komunikáciu? a podobne).

```
void ries_tcp(const u_char *data_ramca, const u_char *vnorene_data_ramca, FILE *vystupny_subor, EXTERNY *tcp, char filter, int cislo_ramca,
    // +13 z vnorene_data_ramca, aby som sa dostal na flag byte

    int source = vnorene_data_ramca[0] * 256 + vnorene_data_ramca[1], destination = vnorene_data_ramca[2] * 256 + vnorene_data_ramca[3];
    EXTERNY *pomocna = tcp;
    while(pomocna != NULL){
        if(destination == pomocna->value){
            tcp = pomocna;
            break;
        }
        pomocna = pomocna->dalsi;
    }
    if(pomocna == NULL){
        while(tcp != NULL){
            if(source == tcp->value)
                break;
            tcp = tcp->dalsi;
        }
    }
```

Logika tu je pomerne náročná, keďže možných stavov môže byť viacero.

```
//ak som nasiel, tak upravim strukturu, ak bude co upravovat
if(akt){
    //printf("cisielko %d\n", cislo_ramca);
    akt->zoznam_ramcov[akt->pocet_ramcov++] = cislo_ramca;

    //nie je reset?
    if(flagy[2] == 1){
        akt->stav[0] = 2;
        akt->stav[1] = 2;
        akt->stav[2] = 2;
        akt->stav[3] = 2;
        akt->uzavreta = 1;
        //printf("Zatvaram komunikaciu, kvoli resetu, zacinajucu na porte %d a konciacu na porte %d\n", akt->zoznam_ramcov[0], cislo_ramca);
    }

    //syn ack combo
    else if(akt->stav[0] == 1 && akt->stav[1] == 0 && akt->stav[2] == 0 && akt->stav[3] == 0 && flagy[0] == 1 && flagy[3] == 1 && *(int *)adresa1 == *(int *)akt->adresa2){
        akt->stav[2] = 1;
        akt->stav[3] = 1;
        //printf("Vypis 1 = som na ramci %d\n", cislo_ramca);
    }
    //ack solo
    else if(akt->stav[0] == 1 && akt->stav[1] == 0 && akt->stav[2] == 1 && akt->stav[3] == 1 && flagy[0] == 1 && *(int *)adresa1 == *(int *)akt->adresa1){
        akt->stav[1] = 1;
        //printf("Vypis 2 = som na ramci %d\n", cislo_ramca);
    }

    //zacnem zatvaranie, prve fin
    else if(akt->stav[0] == 1 && akt->stav[1] == 1 && akt->stav[2] == 1 && akt->stav[3] == 1 && flagy[4] == 1){
        if(*(int *)adresa1 == *(int *)akt->adresa1)
            akt->stav[0] = 2;
        else if(*(int *)adresa1 == *(int *)akt->adresa2)
            akt->stav[2] = 2;
        //printf("Vypis 3 = som na ramci %d info: %d %d %d %d\n", cislo_ramca, akt->stav[0], akt->stav[1], akt->stav[2], akt->stav[3]);
```

Pri UDP sledujem len TFTP, teda číslo 69 na destination porte. Ak taký port nájdem, tak mi tu začína komunikácia a musím si zapamätať source port a ip adresy, aby som ju vedel sledovať.

```
if(tftp_port == -1 && destination == 69){
    tftp_port = source;
    fprintf(vystupny_subor, "KOMUNIKACIA\n");
}
else if(tftp_port > -1 && destination == 69){
    tftp_port = -2;
}
if(source == tftp_port || destination == tftp_port){
```

Príklad štruktúry externých súborov pre určenie protokolov a portov

00 Null_SAP

02 LLC_Sublayer_Management_Individual

03 LLC_Sublayer_Management_Group

06 IP

0E PROWAY_Network_Management

42 BPDU

4E MMS

5E ISI_IP

7E X.25_PLP

8E PROWAY_Active_Station_List_Maintenance

AA SNAP

E0 IPX

F4 LAN

FE ISO

FF RAW

Štruktúra je nasledovná, súbor obsahuje ľubovoľné množstvo dvojíc, pozostávajúcich z jedného hexadecimálneho čísla a prislúchajúcemu názvu, ktorý nesmie obsahovať medzery (používam podtržníky namiesto medzier), keďže názvy načítavame ako jeden string (načítanie stringu sa stopne na prvom „white space“).

Používateľské rozhranie

Používateľské rozhranie je veľmi ľahko pochopiteľné, keďže nám program sám vypíše (do konzoly), aké vstupy očakáva. Najskôr nám vypíše všetky predpripravené .pcap súbory a nechá nám z nich jeden vybrať (napíšeme číslo príslušného súboru, taktiež do konzoly). Následne sa nás program spýta, či chceme použiť nejaký filter, alebo spraviť kompletný výpis podľa bodu 1 (taktiež našu voľbu vyjadrujeme príslušným číslom). Keď skončí s touto konfiguráciou, tak sa nás opýta,

či máme záujem o analýzu ďalšieho súboru a pokračuje odznova. Pri správnom ukončení programu, vypíše „koniec“ do konzoly. Príklad:

```
1   =>  eth-1                   2   =>  eth-2                   3   =>  eth-3
4   =>  eth-4                   5   =>  eth-5                   6   =>  eth-6
7   =>  eth-7                   8   =>  eth-8                   9   =>  eth-9
10  =>  trace_ip_nad_20_B       11  =>  trace-1                 12  =>  trace-2
13  =>  trace-3                 14  =>  trace-4                 15  =>  trace-5
16  =>  trace-6                 17  =>  trace-7                 18  =>  trace-8
19  =>  trace-9                 20  =>  trace-10                21  =>  trace-11
22  =>  trace-12                23  =>  trace-13                24  =>  trace-14
25  =>  trace-15                26  =>  trace-16                27  =>  trace-17
28  =>  trace-18                29  =>  trace-19                30  =>  trace-20
31  =>  trace-21                32  =>  trace-22                33  =>  trace-23
34  =>  trace-24                35  =>  trace-25                36  =>  trace-26
37  =>  trace-27
Napis cislo suboru, ktory chces analyzovat:
10
Aky filter z bodu 4 chces pouzit na subor?
0  =>  Nechcem filter
1  =>  HTTP          2  =>  HTTPS          3  =>  TELLNET
4  =>  SSH           5  =>  FTP riadiace   6  =>  FTP datove
7  =>  TFTP          8  =>  ICMP
0
Chces zvolit dalsi subor?
0 -> nie
1 -> ano
0
Koniec
PS C:\Users\Admin\Desktop\Adamovic_PKS_1>
```

Implementačné prostredie

Program som implementoval v jazyku C za pomoci knižnice pcap.h.