VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikácie a siete

Sniffer paketov

22. apríla 2022

Igor Hanus (xhanus19)

Obsah

1 Úvod **1**

2 Spracovanie argumentov **1**
2.1 Výpis dostupných rozhraní 1
2.2 Nastavenie filtru 1

3 Sniffovanie **2**
3.1 Spracovanie rámcu 2
3.2 Výpis timestampu 2
3.3 Výpis MAC adries 2
3.4 Sniffing IPv4 a IPv6 3
3.5 Sniffing ARP 4

4 Výpis rámcu **5**

5 Testovanie **5**

1 Úvod

Projekt sa zaoberá odchytávaním internetových rámcov a ich nasledným spracovaním a vypísaním na základe nastaveného filtra zo vstupných argumentov. Projekt je písaný v jazyku C++ 17 za pomocou knižnice PCAP.

2 Spracovanie argumentov

Pri spracovaní vstupných argumentov je použitá funkcia `parseArgs`. Prvotne sú preddefinované nastavenia pre jednotlivé argumenty pomocou `parseArgs` do štruktúr typu `option`, kde sa nastavia potrebné parametre pre jednotlivé argumenty, ktorými sú napríklad flag, ktorý je nastavený na `nullptr` aby getops vracial jeho honotu, ktorá je pri každom argumente nastavená ako malé písmeno abecedy.

Následne sa prechádzajú jednotlivé argumenty a podľa jednotlivých hodnôt sa nastavujú hodnoty v statickej triede `ArgData`. Implementácia obsahuje jeden argument navyše a tým je `sniff-only`, podľa ktorého sa následne filtruje výpis jednotlivých rámcov. Viac informácií v sekcii o testovaní.

Celé zpracovanie argumentov vychádza z dokumentácie `getopt`, viď [1], v časti s príkladom na `getopt_long`. V rámci kontoly či číslo portu je reálne číslo sa používa pomocná funkcia `isNumber`, ktorá bola inšpirovaná podľa odpovede na fóru, viď [3].

2.1 Výpis dostupných rozhraní

V prípade potreby vypísať všetky dostpné rozhrania, volá sa funkcia `printAllInterfaces`, ktorá používa funkciu `pcap_indalldevs`, z dokumentácie pre pcap, viď [8], pre získanie rozhraní a následne sa cez while cyklus vypíšu na štandardný výstup.

2.2 Nastavenie filtru

Keď už je vybrané špecifické rozhranie tak je volaná funkcia `setFilter`, ktorá následne volá ďalšie funkcie na nastavenie vyžiadaného filtra, ktoré používaju implementácie podľa [8]. Ako prvá sa zavolá funkcia `openInterfaceForSniffing`, kde sa volá `pcap_open_live` s čítacou dobou 1000ms, ktorá otvorí prepojenie s vybraným rozhraním. Následne sa vytvorí výraz špecifický pre filter podľa vstupných argumentov vo funkcii `setFilterArg`. Reťazec `arg` sa rozširuje podľa použitých protokolov. Pre ICMP protokol sa nastavuje ako `icmp` tak aj `icmp6` kvôli podpore IPv6. Následne sa na koniec argumentu pridá port (ak je vyžadovaný). Bližšie šepecifikácie pre výrazy pre filtre nájdete na [11]. Následne sa pomocou `pcap_compile` a `pcap_setfilter` nastavia definované nastavenie vo výraze pre filter a už len stačí začat sniffovať.

Príklad nastavenia filtru pri použití ICMP a TCP a portu 443:
icmp or icmp6 or (tcp and port 443)

V tomto prípade program bude vypisovať rámce TCP na porte 443 a ICMP rámce (port sa neberie do úvahy).

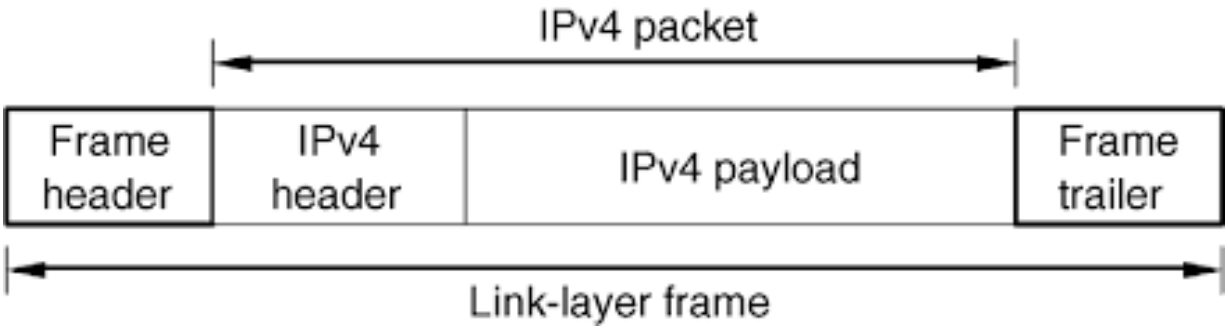
3 Sniffovanie

Celé sniffovanie je spustené z funkcie `startSniffing`, kde sa zavolá funkcia `pcap_loop`, ktorej argumenty obsahujú funkciu, ktorá spracováva jednotlivé rámce a počet rámcov, ktorý chceme odchytiť. Pre jednoduchšie zpracovanie rámcov sa používajú `netinet` štruktúry, viac na [4]. Ďalej pre definovanie jednotlivých protokolov, resp. ich čísiel sú v `ipk-sniffer.h` definované makra. Jednotlivé čísla pre protokoly boli získane primárne z wikipédie [5] a [9]. Ďalej sú definované čísla typov jednotlivých rámcov, ktoré boli prevedené do desiatkovej sústavy pre jednoduchšiu manipuláciu v rámci kódu. [6].

3.1 Spracovanie rámcu

Po zachytení rámcu najskôr získame ethernetovú hlavičku z ukazateľa na samotný rámec. Podľa obrázku je vidno, že samotná hlavička je na začiatku a jej dĺžka je 14 bytov podľa dokumentácie knižnice `pcap` [8].

Na základe získanej hlavičky vieme určiť typ packetu, ktorý sa nachádza v odchytenom rámci. Z hlavičky naďalej vieme získať timestamp, zdrojovú a cieľovú MAC adresu.



Obr. 1: Názorné zobrazenie rámcu (v tomto prípade IPv4)

3.2 Výpis timestampu

Pred samotným výpisom timestampu je potrebné si previesť dátum z Unix času na normálny kalendárny čas. To sa docielilo použitím funkcie `localtime` a následne získaný čas sa formátoval pomocou `strftime` do tvaru YYYY-mm-ddT HH:MM:SS. Ku samotnému naformátovanému času sa na koniec manuálne pridal časový offset podľa zadania.

3.3 Výpis MAC adries

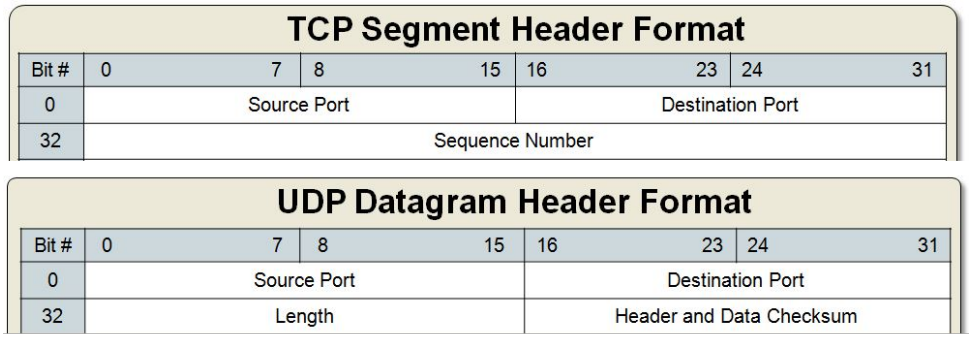
Vo funkcii `printMacAddress` sa po získaní dvoch MAC adries pomocou funkcie `printf` správne naformátujú na výstup. Na výstup sa teda vypisovali jednotlivé byty MAC adresy, viď [7].

3.4 Sniffing IPv4 a IPv6

V rámci IPv4 a IPv6 sa odchytávajú 3 protokoly a tými sú TCP, UDP a ICMP (ICMP6 pre IPv6). Samotná implementácia nepočíta s rozšírenou IPv6 hlavičkou, teda IPv6 next sa nekontroluje.

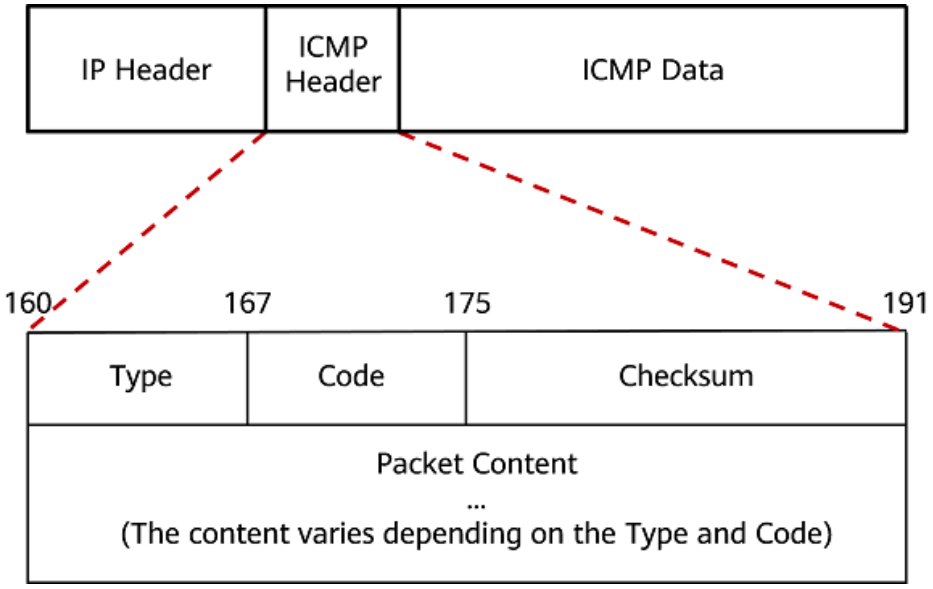
Na základe čísla protokolov, ktoré sú uložené v makrách, ktoré boli spomenuté v sekcii o spracovaní argumentov sa rozlišuje, ktorý protokol enkapsulovaný v IPv6 alebo IPv4 sa bude spracovávať. Tieto protokoly sa v samotnom rámci nachádzajú vedľa hlavičky IP adresy, viď obrázok číslo 3. Preto je potrebné získať veľkosť štruktúry samotnej hlavičky pomocou knižnice `netinet`. Pre IPv4 sa jedná o štruktúru `struct ip` a pre IPv6 `struct ip6_hdr`. Po získaní veľkosti IP hlavičky sme schopní získať hlavičky samotných protokolov. Predtým než sa vypíšu informácie z hlavičiek protokolov, vypíšu sa zdrojové a cieľové IP adresy.

Na výpis IPv6 adresy sa používa identický spôsob ako pri MAC adrese. Pri výpisu IPv4 adresy vo funkcii `printSrcDstIP` používa funkcia `inet_ntoa`, ktorá konvertuje byty adresy, ktoré sú zoradené vo forme Big-endian na čitateľnú pododobu IPv4, viac na [10]. Tento problem u IPv6 nenastáva, nakoľko všetky byty sú "normálne"zoradené čiže není potrebné ich prevádzať. Taktiež sa vypisuje aj typ protokolu daného rámcu.



Obr. 2: TCP a UDP packet

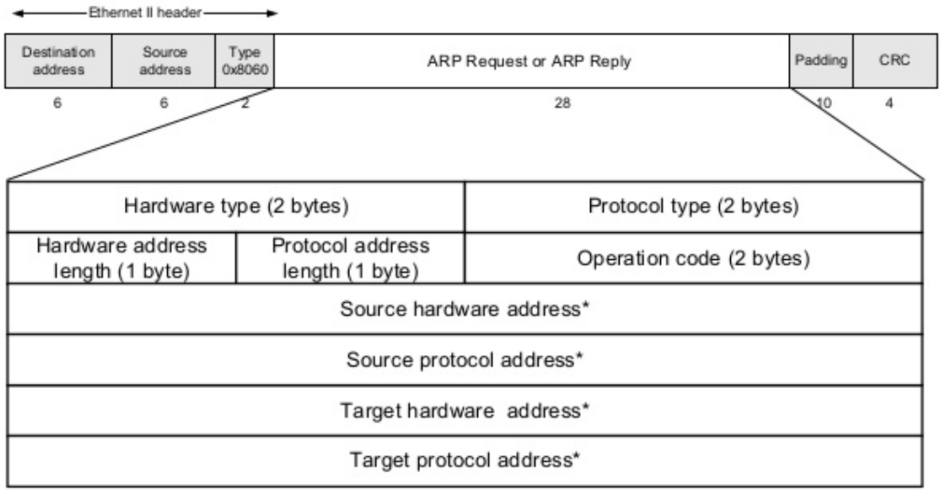
Pre UDP a TCP nám stačí zistiť len port procesu, ktorý packet odosiela a port procesu, ktorý ho príjma. Z hlavičky TCP/UDP packetu vyberieme port a pomocou funkcie `nthos` [2] prekonvertujeme port zo štruktúry, ktorá je znova zoradená vo forme Big-endian na formát, ktorý je podporovaný procesorom, na ktorom je sniffer spustený.



Obr. 3: ICMP packet

Pre ICMP a ICMPv6 sú na výstup vypísane len zdrojové a cieľové IP adresy.

3.5 Sniffing ARP



Obr. 4: ARP packet

Pre ARP sa vypisujú len zdrojové a cieľové MAC adresy.

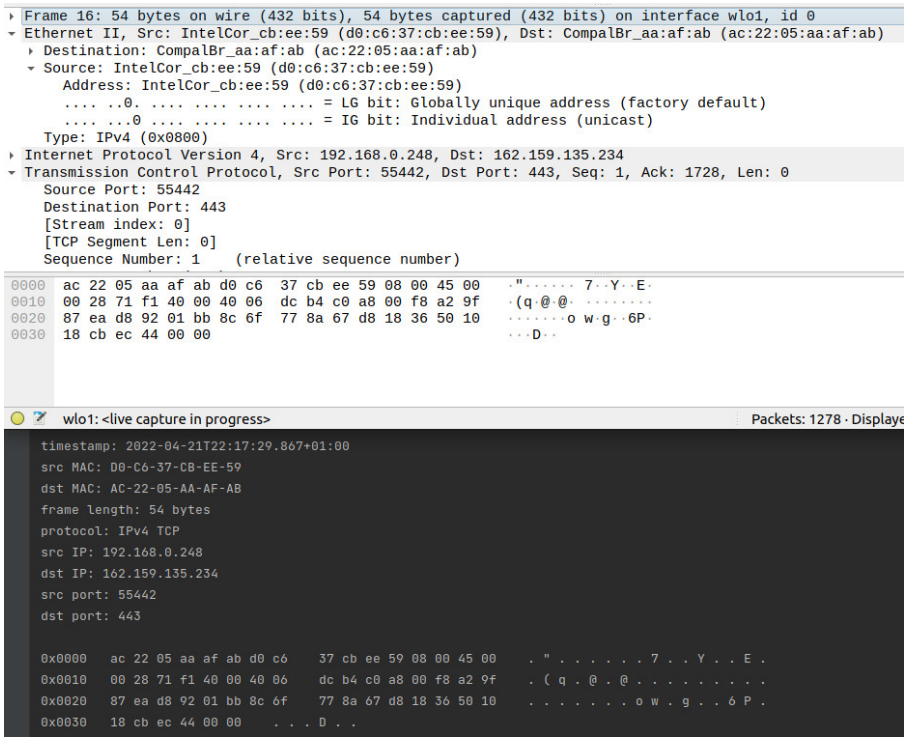
4 Výpis rámcu

Pre výpis rámcu sa používa funkcia `printFrameData`. Pre každý vypísaný riadok sa vyhradí maximálne 16 bytov. Na výpis riadkov sa volá funkcia `printFrameLine`.

Vo funkcii `printFrameLine` sa najprv vypíše offset, ktorý začína od 0 a je pri každom volaní tejto funkcie zvýšený o 16. Ďalej sa podla žiadanej dĺžky riadku vypisujú jendotlivé byty (v hexadecimálnej sústave) rámecu. Po výpise bytov sa vrátime späť na začiatok riadku a začneme vypisovať ich hodnoty, resp. charaktery. Aby sa zaistilo, že sa vypisujú len validné charaktery tak sa používa funkcia `isprint`.

5 Testovanie

Testovanie projektu bolo bolo založené na porovnávaní výstupu snifferu v rámci projektu s referenčným výstupom programu WireShark. Pre testovanie bol vytvorený jeden argument navyše čisto pre testovacie účely. Na základe argumentu `sniff-only` sa mohli filtrovat výpisi podľa typov packetov a to IPv4, IPv6 (aj ARP ale to už je možné filtrovať pomocout `--arp` argumentu). Uľahčilo to prácu, najmä časovo, a nebolo potrebné použiť filtrovacie argumenty podľa zadania, kde pre TCP, UDP a ICMP vypisovalo packety ako pre IPv4 tak aj pre IPv6.



Obr. 5: Porovnanie výstupu projektu s programom WireShark

Literatúra

[1] getopt_long(3) - Linux man page. [online].
URL https://linux.die.net/man/3/getopt_long

[2] How do I get sender's UDP port in C? [online].
URL https://stackoverflow.com/questions/702451/how-do-i-get-senders-udp-port-in-c

[3] How to determine if a string is a number with C++? [online].
URL https://stackoverflow.com/questions/4654636/how-to-determine-if-a-string-is-a-number-with-c

[4] IBM Documentation - Header files. [online].
URL https://www.ibm.com/docs/en/zos/2.2.0?topic=zxcrlr-header-files

[5] IPv4. [online].
URL https://en.wikipedia.org/wiki/IPv4#Header

[6] List of IP protocol numbers. [online].
URL https://en.wikipedia.org/wiki/EtherType

[7] MAC Address printing. [online].
URL https://stackoverflow.com/questions/6063039/mac-address-printing

[8] Programming with pcap. [online].
URL https://www.tcpdump.org/pcap.html

[9] Protocol Numbers. [online].
URL https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

[10] Why do inet_ntoa and inet_ntop "reverse"the bytes? [online].
URL https://stackoverflow.com/questions/30153966/why-do-inet-ntoa-and-inet-ntop-reverse-the-bytes

[11] Sultana, N.: What we talk about when we talk about pcap expressions. [online].
URL https://www.seas.upenn.edu/~nsultana/files/pcap_semantics.pdf