VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě – 2. projekt

Sniffer paketů – manuál

3. května 2020 Richard Klem

Obsah

1 Úvod

Data, která se po síti přenáší jsou členěna na drobnější části. K těmto segmentům dat se pak připojují další informace jako je třeba cílová adresa(kam mají být data doručena), zdrojová IP adresa(odkud se data posílají), velikost přenášených dat apod. Takovýto „balíček“ je pak odeslán po síti.

Sniffer paketů neboli *síťový analyzátor* je aplikace, která na určitém síťovém rozhraní zachytává, filtruje a dále zpracovává příchozí a odchozí pakety. Motivace k vytvoření takovéto aplikace může být například

- detekování podezřelého chování v síti,
- ochrana vnitřní sítě před útoky zvenčí,
- analýza provozu a jeho špiček,
- shromažďování statistik k další analýze.

Mezi známé analyzátory patří například *Wireshark* nebo *tcpdump*[3].

Síťová komunikace pracuje na několika vrstvách na základě OSI modelu. Tento model definuje 7 vrstev, od fyzické až po aplikační. Analyzátor tedy musí na základě znalostí o struktuře dat v paketu rekonstruovat jednotlivé struktury a dále je případně zpracovat a analyzovat[4].

2 Návrh

2.1 Výběr jazyka

Moje implementace analyzátoru je v jazyce C++. Ačkoli není naplno využito možností tohoto jazyka, zvolil jsem si ho právě pro jeho širší spektrum možností, které nabízí. Z C++ bylo využito například třídy `string`, která poskytuje automatické dynamické alokace paměti a práce s instancemi `string` je tedy přehlednější a snazší než s `char *`. Nebo metody `c_str` třídy `string` pro převod instance `string` na „céčkové“ pole znaků.

I když jazyk C není podmnožinou jazyka C++, knihovny pro zpracování paketů jako např. *libpcap*(v jazyce C), jsou komplatibilní s C++. Anebo jsou kompatibilní po drobných úpravách, např. explicitní přetypování apod. Pro možnost dalšího vývoje a rozšíření jsem tedy zvolil paradigmatický jazyk C++.

2.2 Struktura a jména zdrojových souborů

Zdrojový kód aplikace je členěn do několika samostatných zdrjovových souborů reprezentujících logické části aplikace. Zdrojové soubory mají zpravidla i korespondující hlavičkový soubor. Vyskytují se ale i samostatné hlavičkové soubory obsahující například definice struktur.

Ilustrace adresáře se zdrojovými soubory:

```
.
|-- ipk-sniffer.cpp
|-- ipk-sniffer.h
|-- Makefile
|-- my_arp.h
|-- my_dns_cache.cpp
|-- my_dns_cache.h
|-- my_getnameinfo.cpp
|-- my_getnameinfo.h
|-- my_string.cpp
'-- my_string.h
```

Jména souborů začínající předponou `my_` prezentují soubory, ve kterých jsou implementovány funkce podobné jako v některých již vytvořených knihovnách(ne nutně systémových).
Kupříkladu v souboru `my_getnameinfo.cpp` je implementovaná funkce stejné funcionality jako GNU `getnameinfo`, ale navíc jsou zde různá omezení nebo rozšíření, aby byly lépe pokryty případy užití této aplikace.

2.3 Implementace z pohledu vyšší abstrakce

Zde bude vysvětlen průběh programu z pohledu vyšší abstrakce, bude se jednat o slovní popis funkčnosti. Popsána bude pouze implementace základní specifikace. Rozšíření budou popsána v závěrečné části, která je jim věnována.

Aplikace načte a zpracuje vstupní argumenty a zkontroluje jejich korektnost. Napojí se na specifikované rozhraní a aplikuje na něj patřičné filtry. Následně se volá funkce `pcap_loop`, která zachytává filtrované pakety. Nad každým paketem zavolá tzv. `pcap_handler` fukci. To je funkce, která obstarává samotné zpracování paketu. V mojí implementaci se funkce jmenuje `callback`. Funkce `callback` využívá řadu podpůrných funkcí pro zpracování konkrétních typů paketů a výsledkem je výpis dat z paketu v požadovaném formátu na výstup programu.

3 Implementace – zajímavé pasáže

V této sekci jsou uvedena implementačně zajímavá řešení standartní specifikace.

Jedním z mnoha kroků, které jsou potřeba udělat během zpracování paketu je i překlad IP adresy na doménové jméno(pokud je to možné).

Bez DNS mezipaměti vypadá scénář následovně. Dotaz na doménové jméno je odeslán jako paket do sítě. Tento odchozí paket je ale zachycen snifferem a zpracován, včetně opětovného dotazu na překlad adresy na dom. jméno. Odešle se další dotaz na dom. jméno. A tímto neustálým DNS dotazováním sniffer generuje vyskový provoz.

Jakmile se zachytí příchozí paket s (ne)přeloženou adresou, tak je potřeba uložit tento výsledek do nějaké paměti. Jinak se bude neustále posílat, zachytávat a analyzovat paket s dotazem na jméno a program bude takto cyklit a zatěžovat zbytečně síť! Aby se zabránilo tomuto nechtěnému konání je implementována DNS mezipamět, která uchovává již přeložené adresy, to i v případě, že se ji nepodařilo přeložit. Samotná mezipaměť je implementována jako cyklický registr struktur obsahujích dvojici `adresa:jméno` nebo `adresa:adresa` v případě, že se adresu nepodařilo přeložit. Pro IPv4 a IPv6 jsou oddělené mezipaměti. Kapacita pro IPv4 adresy je 64 záznamů, pro IPv6 pak 16 záznamů.

Ukládání přeložených adres obstarává vlastní funkce `getnameinfo`, která využívá struktur a funkcí ze souborů `my_dns_cache.cpp/h`, kde jsou implementovány obslužné funkce pro DNS mezipaměti.

Další zajímavostí může být třeba využití bitového posuvu. Tzv. „ethertype“ je definován na 13. a 14. bajtu(poslední dva) v ethernetové hlavičce(linková vrstva, OSI vrstva 2).
Získání této hodnoty je realizováno takto:

```
eth->h_proto = (packet[12] << (unsigned int) 8) + packet[13];
```

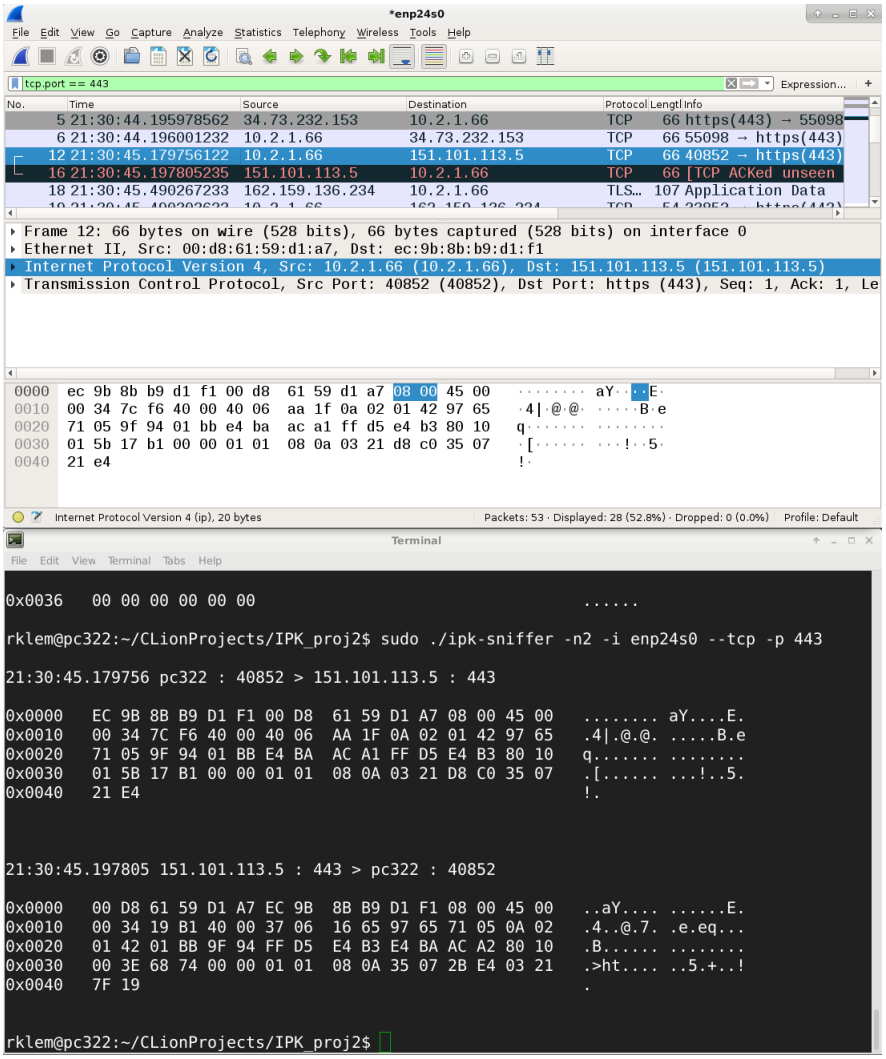
Ten samý výsledek jsme schopni dostat i jinak, např.:

```
eth->h_proto = (unsigned short)*(packet + 13)*0x100+(unsigned short)*(packet + 12));
```

Mně se více líbil bitový posun. Také i proto, že když je potřeba něco vynásobit mocninou dvojky($0\text{x}100 = 2^8$), tak je efektivnější právě bitový posuv.

4 Testování

Svoji aplikaci jsem testoval pomocí reálného provozu a aplikace Wireshark, oproti které jsem porovnával výsledky. Zpravidla jsem pustil Wireshark s právě testovanou kombinací filtrů a pak následně pustil na stejném rozhraní a se stenými filtry vlastní aplikaci. Poté jsem nalezl první zachycený paket podle času a kontroloval, zda jsou si ekvivaltní jednotlivé části. Zaměřoval jsem se na časovou stopu paketu, IP adresy, porty a pak na samotná data.



Obrázek 1: Porovnání výstupu z Wiresharku a výstupu z mého snifferu

5 Rozšíření

V této sekci jsou popsána implementovaná rozšíření, ale i další možnosti, které vidím jako dobré rozšíření, ale byli již nad rámec mé kapacity v tomto projektu. Dali by se tedy doimplementovat ve volném čase nebo tento projekt rozvinout např. do bakalářské práce.

5.1 Implementovaná rozšíření

Nad rámec původní specifikace jsem implementoval několik dalších funkcí, kterými sniffer disponuje. Nejprve budou v bodech představeny jednotlivá rozšíření a následně budou podrobně popsána.

Rozšíření:

- Parametr `-a/--arp` – přidává podporu zobrazit i ARP pakety
- Parametr `-6/--ip6` – přidává podporu filtrovat pouze IPv6 adresy
- Parametr `-4/--ip4` – přidává podporu filtrovat pouze IPv4 adresy
- Parametr `-A/--all` – přidává podporu nefiltrovat nijak, zachytávají se úplně všechny pakety
- Parametr `-s/--stats` – přidává podporu zobrazení jednoduchých statistik o síťovém provozu

Podpora ARP paketů umožňuje uživateli filtrovat ARP pakety. ARP je zkratka pro Address Resolution Protocol, který patří do linkové vrstvy OSI modelu. Nicméně je balen do ethernetové hlavičky a může se na první pohled zdát, že spadá, stejně jako IP protokol, do vrstvy síťové, ale není tomu tak. Pracuje totiž s fyzickými adresami a proto patří do vrstvy č.2. ARP je protokol spravující IPv4 adresy[2].

Pro IPv6 existuje podobný protokol, NDP – Neighbor Discovery Protocol. Ten využívá ICMPv6 síťový protokol. ICMPv6 je balen do IPv6 protokolu a proto se může zdát, že ICMPv6 je transportní protokol. Není tomu tak, ICMPv6 pracuej s IPv6 adresami a patří proto do 3. vrstvy OSI modelu[1].

Parametr `-a` v mém snifferu pracuje pouze samostatně. To znamená, že jej nelze kombinovat například s `-u`. Pokud bude nějaká kombinace zadaná, bude ignorována. Stejně tak i parametry pro filtrování IP adres nebo dokonce portu. Pokud je zadán tedy parametr pro filtrování ARP paketů, pak se budou zachytávat pouze tyto pakety a žádné jiné.

Podpora filtrování IPv6 umožňuje uživateli filtrovat pakety zabalené do síťového IPv6 protokolu. Platí restrikce uvedené výše. Parametr lze kombinovat s filtrujícími parametry `-u -t -p -4` a jejich dlouhými variantami.

Podpora filtrování IPv4 umožňuje uživateli filtrovat pakety zabalené do síťového IPv4 protokolu. Platí restrikce uvedené výše. Parametr lze kombinovat s filtrujícími parametry `-u -t -p -6` a jejich dlouhými variantami[1].

Podpora zachytávání všech paketů umožňuje uživateli zachycení všech paketů. Vypíšou se ale jen ty podporované, tedy UDP, TCP i ARP. Parametr `-A/--all` má nejvyšší prioritu a znehodnocuje všechny předchozí i následující volby filtrování!

Podpora výpisu statistik síového provozu uživateli dává možnost zobrazit si počet zachycených paketů jednotlivých typů. Kategorie jsou *Celkový počet*, *TCP*, *UDP*, *ARP* a *Nepodporované*. Tento parametr je vhodné používat společně s parametrem `-A`, kdy se zachytávají všechny pakety a statistiky mají tedy největší informační hodnotu. Parametr `-s/--stats`

5.2 Další možná rozšíření

Jako další možná rozšíření vidím určitě podporu více druhů paketů, například již zmiňované ICMPv6 nebo ICMP, STP či LLDP. Ale také filtrování na nižších vrstvách jako například možnost filtrovat SSH, DHCP, DNS, LDAP apod.

Aplikace s grafickým rozhraním, ale stále s podporou konzolového režimu by byla určitě přínosná. Dále by byla užitečná větší interaktivita s uživatelem, například možnost zachytávat všechny pakety, ale mít možnost si v průběhu vykonávání filtrovat pouze zobrazování bybraných paketů.

[1] Při parametru pro filtrování IPv4 i IPv6 zároveň se filtruje jejich logický součet, tedy buď IPv4 nebo IPv6. Nezachytí se tak třeba ARP paket.

Reference

[1] ComputerNetworkingNotes: IPv6 Neighbor Discovery Protocol Explained. [online], [vid. 30.dubna 2020]. Dostupné z: <https://www.computernetworkingnotes.com/networking-tutorials/ipv6-neighbor-discovery-protocol-explained.html>

[2] FAIRHURST, G.: Address Resolution Protocol (arp). [online], [vid. 30.dubna 2020]. Dostupné z: <https://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html>

[3] MITTAL, D.: What is Packet Sniffing ? [online], [vid. 24.dubna 2020]. Dostupné z: <https://www.geeksforgeeks.org/what-is-packet-sniffing/>

[4] Wikipedia: OSI model. [online], [vid. 30.dubna 2020]. Dostupné z: <https://en.wikipedia.org/wiki/OSI_model>