VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEPARTMENT OF INFORMATION SYSTEMS

ANALYZÁTOR PAKETŮ

PACKET ANALYZER

PROJEKTOVÁ DOKUMENTACE

PROJECT DOCUMENTATION

AUTOR PRÁCE **PETR BUCHAL**

AUTHOR

BRNO 2017

Obsah

Kapitola 1

Úvod

V dnešní době prostupuje internet napříč téměř všemi sférami každodenního života a je tedy důležité, zabývat se tím, co sebou tato skutečnost nese. Běžného člověk většina věcí týkající se této záležitosti zajímat nemusí, stačí mu, aby to prostě fungovalo. Poté jsou zde ale lidé, jejichž prací je, zařídit právě, aby nebyla ohrožena efektivita, bezpečnost a obecně funkčnost internetu a potažmo všech sítí. Správa sítí je složitá disciplína, ale existuje spousta prostředků, které tuto činnost usnadňují. Každý informatik nicméně využívá ke své práci odlišné nástroje podle toho, které se na danou práci hodí nejlépe. Je tedy nápomocné, když tito lidé mají co největší počet kvalitních prostředků a mohou si vybrat opravdu ten, který je pro danou činnost nejlepší.

1.1 Motivace

Klíčový prvek sítí jako takových je skutečnost, že zařízení mezi sebou vzájemně komunikují. Posílají si zprávy různých typů a ty poté interpretují. Co ale když nastane nějaký problém v síti a je třeba zjistit jeho příčinu? Jednou z variant při pátrání po chybě je právě analýza paketů, tedy zpráv, které směřují přes síťovou kartu do zařízení a ze zařízení. Jenže bez nějakého podpůrného prostředku se jedná pouze o binární informace, které člověk nepřečte. Zde se nabízí možnost vytvoření programu, který bude binární formu transformovat do podoby, kterou nebude mít člověk problém přečíst.

1.2 Cíl práce

Vytvořený prostředek bude pracovat offline, tedy bude zpracovávat soubory se zachycenou síťovou komunikací. Vytvoření programu na analýzu paketů chci docílit s pomocí knihovny libpcap, která tento proces značně usnadňuje. Vytváří totiž nad daty jakousi abstraktní vrstvu s jejíž pomocí se s nimi lépe pracuje. Tato knihovna ovšem neusnadní problematiku sítí jako takových. Nachází se zde příliš mnoho protokolů, abych je zvládl ve školní práci analyzovat všechny, a tudíž dělám analýzu pouze několika základních protokolů. Z vrstvy síťového rozhraní analyzuji Ethernet a IEEE 802.1Q (včetně IEEE 802.1ad), z vrstvy síťové analyzuji IPv4, IPv6, ICMPv4 a ICMPv6 a z vrstvy transportní TCP a UDP. Uživatel programu si rovněž bude moci nechat agregovat, filtrovat nebo seřadit analyzovaná data podle různých parametrů.

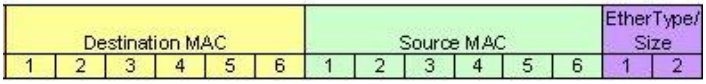
Kapitola 2

Návrh Aplikace

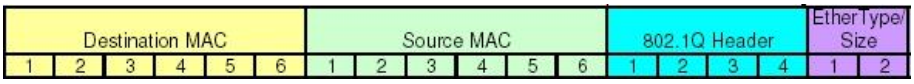
Program se zabývá analýzou několika síťových vrstev. Tato analýza probíhá postupně pro každou vrstvu zvlášť. Rozebereme si tedy, vrstvu po vrstvě, postup, jakým se analýza bude ubírat.

2.1 L2 vrstva

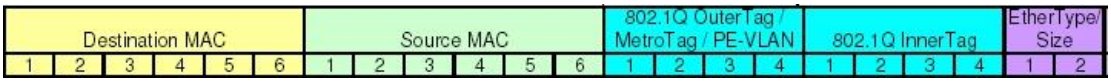
V L2 vrstvě se můžeme setkat se třemi typy hlaviček. Základní hlavička je typu Ethernet a obsahuje pole pro cílovou MAC adresu, zdrojovou MAC adresu a EtherType. Políčko EtherType nám pro naše účely říká, jaký protokol následuje, popřípadě zdali se jedná o jiný typ ethernetové hlavičky. Kód 0x8100 nám říká, že se bude jednat o hlavičku standartu IEEE 802.1Q a kód 0x88a8 indikuje hlavičku standartu IEEE 802.1ad (nejedná se tedy o ethernetovou hlavičku). V obou hlavičkách se vyskytuje EtherType na posledním místě (mezi ním a poli s MAC adresami jsou značky, které nás nezajímají). To znamená, že musíme identifikovat hlavičku, posunout se o daný počet bytů a získat hodnotu EtherType. Ten je totiž důležitý pro zjištění, zdali následující vrstva obsahuje IPv4 nebo IPv6 protokol.



Obrázek 1: Hlavička typu Ethernet Zdroj: https://upload.wikimedia.org/wikipedia/commons/f/f8/EthernetFrame.jpg



Obrázek 2: Hlavička typu IEEE 802.1Q Zdroj: https://upload.wikimedia.org/wikipedia/commons/2/23/TCPIP_802.1Q.jpg



Obrázek 3: Hlavička typu IEEE 802.1ad Zdroj: https://upload.wikimedia.org/wikipedia/commons/1/1b/TCPIP_802.1ad_DoubleTag.jpg

2.2 L3 vrstva

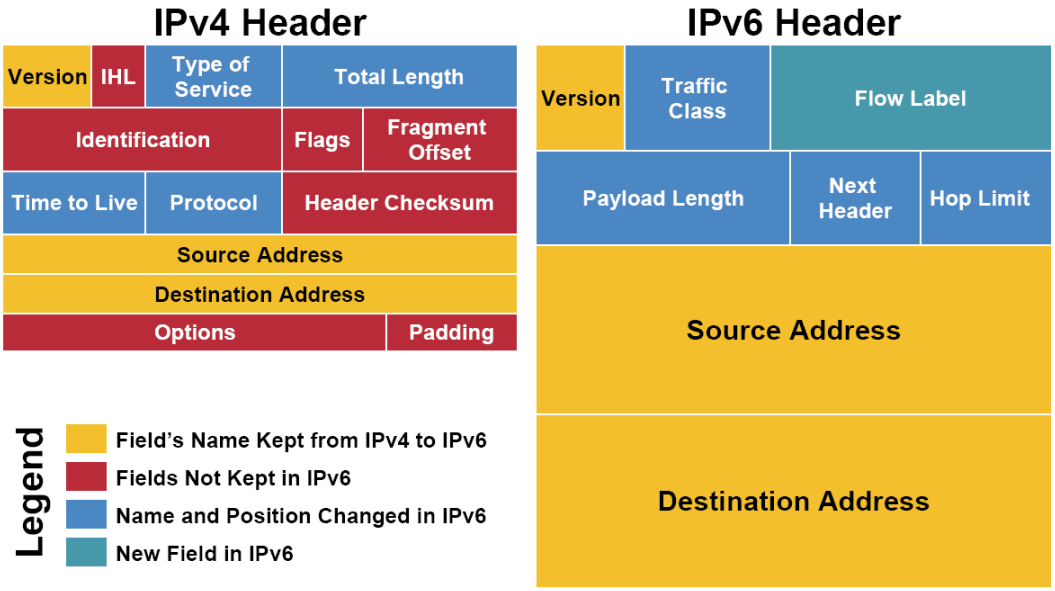
V L3 vrstvě se můžeme setkat se dvěma typy protokolů, konkrétně s IPv4 (RFC 791) a IPv6 (RFC 2460, RFC 5952). Ty jsou od sebe velmi odlišné a nesetkáme se tu jen s přidanými políčky jakožto změnou. Každý protokol je tedy nutné řešit zcela zvlášť.

Ještě než se začneme zabývat položkami hlaviček, je důležité zmínit se o samotných délkách hlaviček, protože ty jsou klíčové pro nalezení začátku hlavičky z vrstvy L4. V případě IPv4 je délka proměnná 40 – 60 bytů. V případě IPv6 je délka hlavičky fixních 40 bytů, ale tento protokol může obsahovat rozšiřující hlavičky a díky těm může být značně delší. Délka rozšiřujících hlaviček pro protokol IPv6 se určuje z jejich políčka délky. Konkrétně se k jeho hodnotě přičte 8 bytů a tohle číslo tvoří celkovou délku rozšiřující hlavičky. Rozšiřující hlavička osahuje pro nás ještě jednu důležitou informaci, a to kód následující hlavičky. Může se jednat buďto o kód hlavičky z následující L4 vrstvy nebo o kód další rozšiřující hlavičky.

Nyní se podíváme na pro nás důležité položky protokolu IPv4. Důležitým políčkem je IHL, z něj získáváme délku IPv4 hlavičky (po vynásobení čtyřmi). Položku identifikace bychom využili jako

část identifikace fragmentovaných paketů, ale fragmentaci jsem v tomto projektu nedělal, a tudíž ho více popisovat nebudu. Příznaky a offset fragmentu slouží rovněž pro práci s fragmentací. Další důležité políčko je TTL (time to live), značí, kolik skoků může paket provést, než se zahodí. Tohle políčko je pro nás důležité, protože ho tiskneme. Poslední dvě důležitá políčka jsou pro nás zdrojová a cílová adresa IPv4, opět z důvodu tisku.

Z IPv6 hlavičky nás zajímají 4 políčka, konkrétně další hlavička, maximum skoků a zdrojová a cílová IPv6 adresa. Maximum skoků je IPv6 varianta TTL a stejně jako TTL ji tiskneme. O položce další hlavička jsme se již bavili výše. Jde o kód hlavičky, která následuje po současné.



Obrázek 4: Srovnání IPv4 a IPv6 Zdroj: https://i2.wp.com/www.ebrahma.com/wp-content/uploads/2013/12/ipv4-ipv6-header.gif

2.3 L4 vrstva

V L4 vrstvě řešíme čtyři různé protokoly. První dvojicí protokolů je TCP a UDP, které sebou většinou přenášejí data na aplikační vrstvě. Druhou kategorií jsou protokoly ICMPv4 (IPv4) a ICMPv6 (IPv6), které ve většině případů oznamují chyby v síti.

Jako první se podíváme na hlavičku protokolu TCP (RFC 793). Z té jsou pro nás důležité položky zdrojový a cílový port, číslo sekvence, potvrzený bajt a příznaky indikující různé situace. Ostatní věci jsou pro nás nedůležité stejně jako jakákoli data uložená v paketu. Analýza paketu zde končí.

V UDP (RFC 768) hlavičce je pro nás důležitý jen zdrojový a cílový port, zbytek paketu opět zahazujeme a nezpracováváme.

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Obrázek 5: Srovnání TCP a UDP hlavičky Zdroj: https://skminhaj.files.wordpress.com/2016/02/92926-tcp_udp_headers.jpg

ICMPv4 (RFC 792) a ICMPv6 (RFC 4443) jsou protokoly, které nás informují o chybách, které nastaly při přenosu dat na síti. Pro nás důležité informace z těchto protokolů se nacházejí v obou hlavičkách hned na začátku, dokonce na stejných místech. Jedná se o typ a kód. Na základě těchto dvou polí se specifikuje, jaká chyba nastala. Nejdříve se chyba určí na základě typu a poté se specifikuje podle hodnoty kódu. Čísla specifikující jednotlivé chyby se pro ICMPv4 a ICMPv6 liší. V programu budeme obě dvě hodnoty tisknout a podle nich budeme vypisovat i definice chyb z RFC.

2.4 Práce s daty

Po proběhnutí analýzy je data buďto možné vytisknout anebo dále zpracovávat. První možností dalšího zpracování je filtrace dat, té uživatel docílí zadáním filtru vyhovujícího libpcap knihovně. Poté je data možné agregovat. Agregace je možná podle zdrojové a cílové MAC adresy nebo podle cílové a zdrojové IP adresy anebo podle cílového a zdrojového portu. Další možná manipulace s daty je řazení podle velikosti paketů nebo podle jejich počtu. Seřazení je vždy sestupné. A poslední možností je vypsání omezeného množství položek.

Kapitola 3

Popis implementace

Popis implementace programu rozdělím do dvou částí. První se bude zabývat strukturami a důvody jejich využití a druhá se bude zabývat implementací samotného analyzátoru.

3.1 Struktury

Hlavičky jednotlivých protokolů jsem sestavil na základě příslušných RFC dokumentů, ze kterých jsem si odnesl informace o polích, které hlavičky obsahují. Datové typy jsem pak volil na základě průzkumu nejrůznějších implementací daných hlaviček v jazyce C. Má implementace hlavičky IPv4 je například velmi podobná implementaci hlavičky v knihovně netinet/ip.h. Jako důležitý vnímám fakt, že pro unsigned hodnoty typu char, short a int používám u_int8_t, u_int16_t, uint32_t. Je tomu tak, protože standartní datové typy mohou mít na různých platformách různou délku, oproti tomu použité mají délku fixní. Za zmínku stojí i skutečnost, že v rámci hlaviček TCP a IPv4 mám definované masky pro snazší práci s jednotlivými položkami hlaviček.

```
47  //TCP Hlavicka
48  struct TCPHeader {
49      u_int16_t th_sport;  //TCP Source port
50      u_int16_t th_dport;  //TCP Destination port
51      u_int32_t th_seq;  //TCP Sequence number
52      u_int32_t th_ack;  //TCP Acknowledgement number
53      u_int8_t th_off;  //TCP Data offset
54      u_int8_t th_flags;  //TCP Flags
55      u_int16_t th_win;  //TCP Window
56      u_int16_t th_sum;  //TCP Checksum
57      u_int16_t th_urp;  //TCP Urgent pointer
58      //Flag masks
59      #define TH_FIN 0x01
60      #define TH_SYN 0x02
61      #define TH_RST 0x04
62      #define TH_PUSH 0x08
63      #define TH_ACK 0x10
64      #define TH_URG 0x20
65      #define TH_ECE 0x40
66      #define TH_CWR 0x80
67  };
```

Nyní si popíšeme struktury, do kterých se ukládají zpracované pakety. Jako první zde máme strukturu, do které se ukládají všechny důležité informace pro budoucí tisk (***PacketData***). Struktura obsahuje struktury hlaviček všech zpracovávaných protokolů. Data každého zpracovávaného paketu se ukládají do struktury hlavičky příslušného protokolu. Zároveň s nimi se ukládá informace o typu protokolu, který je na dané vrstvě přítomen. Této znalosti se poté využívá například při tisku nebo při dalším zpracování paketů. Kromě struktur hlaviček a informací o využitých protokolech struktura obsahuje časovou značku paketu, délku paketu a jeho pořadové číslo zpracování. Typy protokolů jednotlivých vrstev jsou nainicializovány s hodnotou -1 kvůli prevenci chyb v programu.

```
143  //Struktura pro ukladani zpracovanych paketu
144  struct PacketData {
145      int Layer1 = -1;  //Druh L2 vrstvy
146      struct ETHHeader eptr;  //Ethernet hlavicka
147      struct QHeader qptr;  //IEEE 802.1Q Hlavicka
148      struct ADHeader adptr;  //IEEE 802.1ad Hlavicka
149      int Layer2 = -1;  //Druh L3 vrstvy
150      struct IPv4Header ipv4ptr;  //IPv6 Hlavicka
151      struct IPv6Header ipv6ptr;  //IPv4 Hlavicka
152      int Layer3 = -1;  //Druh L2 vrstvy
153      struct TCPHeader tcpptr;  //TCP Hlavicka
154      struct UDPHeader udpptr;  //UDP Hlavicka
155      struct ICMPv4Header icmpv4ptr;  //ICMPv4 Hlavicka
156      struct ICMPv6Header icmpv6ptr;  //ICMPv6 Hlavicka
157      int PacketNumber;  //Cislo zpracovaneho paketu
158      struct timeval ts;  //Casova znacka v mikrosekundach
159      bpf_u_int32 len;  //Delka paketu v bajtech
160  };
```

Druhou strukturou, kterou si popíšeme, je struktura ***AggrData***. Ukládají se do ní všechny typy agregovaných dat. Není zde potřeba uchovávat informaci o tom, který typ dat je agregován, protože na konci agregace každého typu dat se rovnou provádí tisk. Nachází se zde ovšem položka ***IpType*** kvůli rozhodování, jaký typ IP adresy byl agregován. Dále se zde nachází políčka čítající počet bytů a počet paketů pro danou agregovanou hodnotu.

```
162 //Struktura pro ulozeni agregovanych zpracovanych paketu
163 struct AggrData {
164     u_int8_t Aggr_shost[ETHER_ADDR_LEN];  //Agregovana zdrojova MAC adresa
165     u_int8_t Aggr_dhost[ETHER_ADDR_LEN];  //Agregovana cilova MAC adresa
166     struct in_addr Aggr_ip_src;  //Agregovana zdrojova IPv4 adresa
167     struct in_addr Aggr_ip_dst;  //Agregovana cilova IPv4 adresa
168     int IpType = -1;  //Druh IP adresy
169     struct in6_addr Aggr_ip6_src;  //Agregovana zdrojova IPv6 adresa
170     struct in6_addr Aggr_ip6_dst;  //Agregovana cilova IPv6 adresa
171     u_int16_t   Aggr_sport;  //Agregovany zdrojovy port
172     u_int16_t   Aggr_dport;  //Agregovany cilovy port
173     int NumberOfPackets;  //Pocet agregovanych paketu
174     bpf_u_int32 len;  //Delka agregovanych paketu
175 };
```

3.2 Implementace analyzátoru paketů

Program začíná zpracováním argumentů. Jsou zde provedeny testy, zdali nebyly argumenty zadány duplicitně, zda jsou jejich hodnoty korektní a zdali mají řetězce rozumnou délku. Pokud byla nalezena chyba program vypíše chybovou hlášku a končí. Dále se provádí test, zdali nebyl zadán argument na vypsání nápovědy společně s nějakým dalším, pokud byl, program končí opět s chybovou hláškou. Před samotnou analýzou se provede ještě test validity souboru/souborů a validity filtru, pokud nebyla nalezena chyba pokračuje se k samotné analýze.

Nejdříve se pomocí funkce ***GetNumberOfPackets()***, zjistí počty paketů v jednotlivých souborech. Tyto hodnoty jsou sečteny a je nainicializován vektor struktur ***PacketData*** o velikosti součtu všech paketů. Do tohoto vektoru se následně ukládají data ze zpracovaných paketů z jednotlivých souborů.

Jeli zadán nějaký filtr, vrací funkce ***GetNumberOfPackets()*** počet paketů vyhovujících danému filtru. Smyčka programu, která analyzuje data rovněž zpracovává jen pakety vyhovující zadanému filtru. Filtr se nastavuje pomocí funkcí knihovny libpcap ***pcap_compile()*** a ***pcap_setfiler()***.

V hlavní smyčce programu je několik počítadel paketů. V proměnné ***PacketNumber*** je index zpracovávaného paketu, s nímž se po zpracování bude ukládat do vektoru zpracovaných paketů, v proměnné ***PacketNumberVerified*** je uložen počet úspěšně zpracovaných paketů s podporovanými protokoly a v proměnné ***PacketNumberActual*** je uschováno číslo celkového pořadí paketu ze všech zpracovávaných. Narazí-li program na paket obsahující nepodporovaný protokol nabude příznak ***WrongProtocol*** hodnoty true a na konci smyčky se proměnná ***PacketNumber*** sníží o hodnotu jedna. Tím pádem budou uložená data paketu s nepodporovaným protokolem dalším paketem přepsána.

Samotná analýza probíhá vrstvu po vrstvě, stejně jako bylo uvedeno v části dokumentace „Návrh Aplikace“. Jen si dovolím zmínit, že při analýze IPv4 fragmentovaného paketu se paket zahodí a na chybový výstup se vypíše hlášení „Fragmented packet.“.

Po skončení smyčky analyzující pakety se program přesouvá k tisku. Pokud nebyla zadána agregace, kontroluje se, zdali bylo zadáno řazení. Pokud bylo zadáno, pomocná funkce ***GetMax()*** určuje pořadí ve kterém se budou pakety tisknout, pokud nebylo, vytisknou se pakety v pořadí ve kterém byli zpracovány. Funkce ***GetMax()*** prochází vektor struktur ***PacketData*** jako lineární seznam a vrací index prvku s největší délkou, daný paket se vytiskne a jeho délka se vynuluje, tudíž je eliminována duplicita hodnot při tisku paketů. Při tisku se kontroluje rovněž, zdali byl zadán limit maximálního počtu vytištěných paketů.

Pokud byla zadána agregace vstoupí program do funkce ***printPacketAggr()***, která se sama postará o agregaci, řazení a ohlídání limitu maximálního počtu vytištěných paketů. Agregované pakety se ukládají do vektoru struktur ***AggrData***, ze kterého se na konci každé konkrétní agregace tisknou. Při agregaci se prochází vektor agregovaných struktur jako lineární seznam a hledá se stejná hodnota agregačního klíče. Datový typ agregačních klíčů je určen datovými typy ve struktuře ***AggrData***. Pro následné řazení se používají funkce ***GetMaxAggrLen()*** pro řazení podle délky a ***GetMaxAggrPac()*** pro řazení podle počtu odeslaných paketů. Algoritmy pro řazení jsou takřka stejné jako u tisku neagregovaných položek.

3.3 Seznam využitých knihoven

- stdio.h
- stdlib.h
- stdbool.h
- ctype.h
- unistd.h
- string.h
- pcap/pcap.h
- netinet/ether.h
- arpa/inet.h
- inttypes.h
- vector
- string
- iostream

Kapitola 4

Návod na použití

Program je nutné spouštět alespoň s jedním analyzovatelným souborem (pokud nechceme vypsat nápovědu). Analyzovaný soubor musí být čitelný knihovnou libpcap.

4.1 Parametry programu

-h — Vypíše nápovědu a ukončí program. Tento parametr nelze kombinovat s žádným jiným.

-a aggr-key — Zapnutí agregace podle klíče aggr-key, což může být *srcmac* značící zdrojovou MAC adresu, *dstmac* značící cílovou MAC adresu, *srcip* značící zdrojovou IP adresu, *dstip* značící cílovou IP adresu, *srcport* značící číslo zdrojového transportního portu nebo *dstport* značící číslo cílového transportního portu.

-s sort-key — Zapnutí řazení podle klíče sort-key, což může být packets (počet paketů) nebo bytes (počet bajtů). Řadit lze jak agregované, tak i neagregované položky. Řadí se vždy sestupně.

-l limit — Nezáporné celé číslo v desítkové soustavě udávající limit počtu vypsaných položek.

-f filter-expression — Program zpracuje pouze pakety, které vyhovují filtru danému řetězcem filter-expression.

file — Cesta k souboru ve formátu pcap (čitelný knihovnou libpcap). Možné je zadat jeden a více souborů.

Kapitola 5

Závěr

Program úspěšně provádí offline analýzu pcap souborů, ovšem u fragmentovaných paketů protokolu IPv4 vypíše pouze, že jsou fragmentované. Pro překlad slouží soubor Makefile, program je překládán překladačem g++. Prostředek byl otestován na referenčním serveru Merlin (CentOS/Linux).

5.1 Metriky kódu

Počet souborů: 1 soubor

Počet řádků zdrojového textu: 2590 řádků

Velikost spustitelného souboru: 145 184 bajtů

Literatura

[1] *Síťové aplikace a jejich architektura*. Brno: VUTIUM, 2014. ISBN 978-80-214-3766-1.