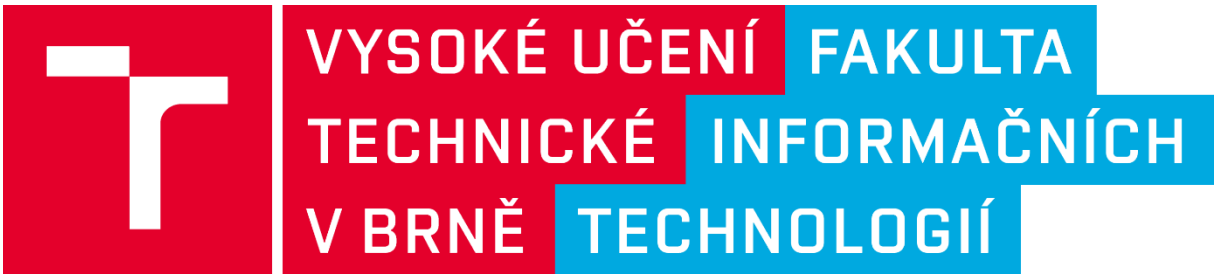



# IPK – projekt 2

# Varianta ZETA: Sniffer paketů

# 2021/2022

Tetauer Pavel (xtetau00) 24.04.2022

# Obsah

# Úvod

Úkolem bylo navrhnout a implementovat síťový analyzátor v C/C++/C#, který bude schopný na určitém síťovém rozhraním zachytávat a filtrovat pakety. Je implementována podpora jak IPv4, tak IPv6. Program defaultně zachycuje pakety TCP, UDP, ICMPv4, ICMPv6 a ARP. Tyto jdou dále filtrovat pomocí argumentů programu.  Zachytávání dalších typů paketů není podporováno. Taktéž je podporován pouze link-type ethernet. Projekt je vypracován v jazyce C++.

# Volání programu

*sudo ./ipk-sniffer {--help} [-i rozhraní | --interface rozhraní] {-p port} {[--tcp|-t] [--udp|-u]  [--arp] [--icmp] } {-n num}*

- -i *eth0* (právě jedno rozhraní, na kterém se bude poslouchat. Nebude-li tento parametr uveden, či bude-li uvedené jen -i bez hodnoty, vypíše se seznam aktivních rozhraní)
- -p *23* (bude filtrování paketů na daném rozhraní podle portu; nebude-li tento parametr uveden, uvažují se všechny porty; pokud je parametr uveden, může se daný port vyskytnout jak v source, tak v destination části)
- -t nebo --tcp (bude zobrazovat pouze TCP pakety)
- -u nebo --udp (bude zobrazovat pouze UDP pakety)
- --icmp (bude zobrazovat pouze ICMPv4 a ICMPv6 pakety)
- --arp (bude zobrazovat pouze ARP rámce)
- Pokud nebudou konkrétní protokoly specifikovány, uvažují se k tisknutí všechny (tj. veškerý obsah, nehledě na protokol)
- -n *10* (určuje počet paketů, které se mají zobrazit, tj. i "dobu" běhu programu; pokud není uvedeno, uvažujte zobrazení pouze jednoho paketu, tedy jakoby -n 1)
- argumenty mohou být v libovolném pořadí

# Příklady volání

*./ipk-sniffer --help*

*sudo ./ipk-sniffer -i eth0*

*sudo ./ipk-sniffer -i*

*sudo ./ipk-sniffer -i eth0 --tcp -p 80*

*sudo ./ipk-sniffer -i eth0 --arp*

# Implementace

1. Funkce `main()`

   Ve funkci `main()` je nejprve volána funkce `parse_arguments()`, která zajistí rozparsování argumentů a jejich následné uložení do odpovídajících globálních proměnných. Následuje kontrola, zda byl zadán argument *-i* či *--interface*, následovaný rozhraním. Pokud tento argument zadán nebyl, nebo po něm nenásledovalo rozhraní, následuje funkce `get_all_interfaces()`, která vypíše seznam všech dostupných rozhraní a skončí. Naopak pokud bylo rozhraní specifikováno, provede se několik funkcí z pcap.

   Jako první z pcap funkcí je `pcap_lookupnet()`, která zjistí IP adresu a masku pro zadaný interface. Následuje funkce `pcap_open_live()`, která otevře zadané rozhraní pro sniffing v promiskuitním módu. Po tomto otevření jsem implementoval signal handling, který při signálu SIGINT korektně nejprve uzavře otevřené rozhraní. Poté je použita funkce `pcap_datalink()`, ověřující, zda otevřené rozhraní poskytuje link-type ethernet, který je jediný podporovaný.

   Pokud se jedná o ethernet, provede se funkce `create_filter()`, která vytvoří filtr na základě zadaných argumentů. Pokud žádné specifikující argumenty nebyly zadány, vytvoří filtr "*tcp or udp or arp or icmp or icmp6".* Tento filtr je pak zkompilován za použití funkce `pcap_compile()` a následně aplikován pomocí `pcap_setfilter()`. Jako další následuje funkce `pcap_loop()`, která pro každý packet volá implementovanou callback funkci a nakonec je použita funkce `pcap_close()`, která uzavře otevřené rozhraní a ukončí sniffing session.

2. Funkce `callback()`

   Jedná se o funkci pro `pcap_loop()`, která je volána pro každý packet. V této funkci je nejprve volána funkce `print_timestamp()`, která zjistí a následně převede čas a datum zachycení packetu do specifikovaného formátu. Následně je obdržený packet přetypován na *ether_header**, pomocí kterého jsou následně vytištěny MAC adresy. Následně je z parametru *header* vytištěna *caplen*, která představuje délku daného packetu v bytech.

   Následuje *switch*, ve kterém se na základě *ether_type* vybere, zda se jedná o IPv4, IPv6 nebo ARP. V každém případě se nejprve přeskočí ethernet header, který má délku 14 bajtů a takto „posunutý“ packet se přetypuje na odpovídající header a vypíše se cílová a počáteční IP adresa. V případě IPv4 se musí vypočítat proměnlivá délka hlavičky, která je chována v *ip->ip_hl*. V IPv6 je fixní délka hlavičky 40. Tyto délky hlaviček se přeskočí a takto vzniklý packet se přetypuje na TCP či UDP header, z kterých se poté vypisují porty. Pokud se jedná o ICMP, nemusí se nic přetypovat,

jelikož žádný port nemá a nejsou v něm uložená žádná potřebná data. Nakonec jsou vypsána data z celého packetu, tedy od 0 do *caplen*.

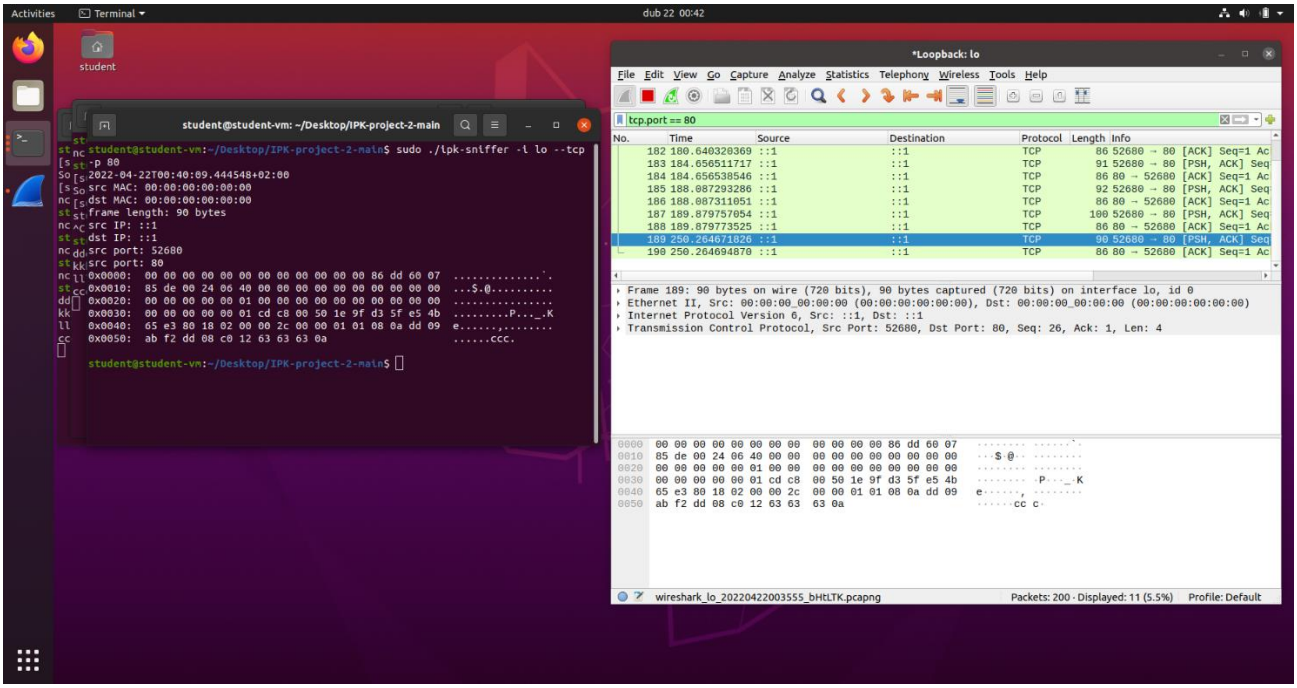
V případě ARP se nemusí nic nadále počítat ani přeskakovat, pouze se vypíší IP adresy a data.

3. Funkce `print_data()`
   Všechna data z packetů jsou vypisována pomocí této funkce. Nachází se v ní cyklus while, který vypisuje data, dokud nenarazí na caplen. Data jsou tisknuta po 16 bajtech, nejprve hexadecimálně, poté následuje mezera a data v Ascii. Takto se opakují až do konce packetu. Na začátku každého řádku se nachází offset vypsaných bajtů, který je v hexadecimální formě.

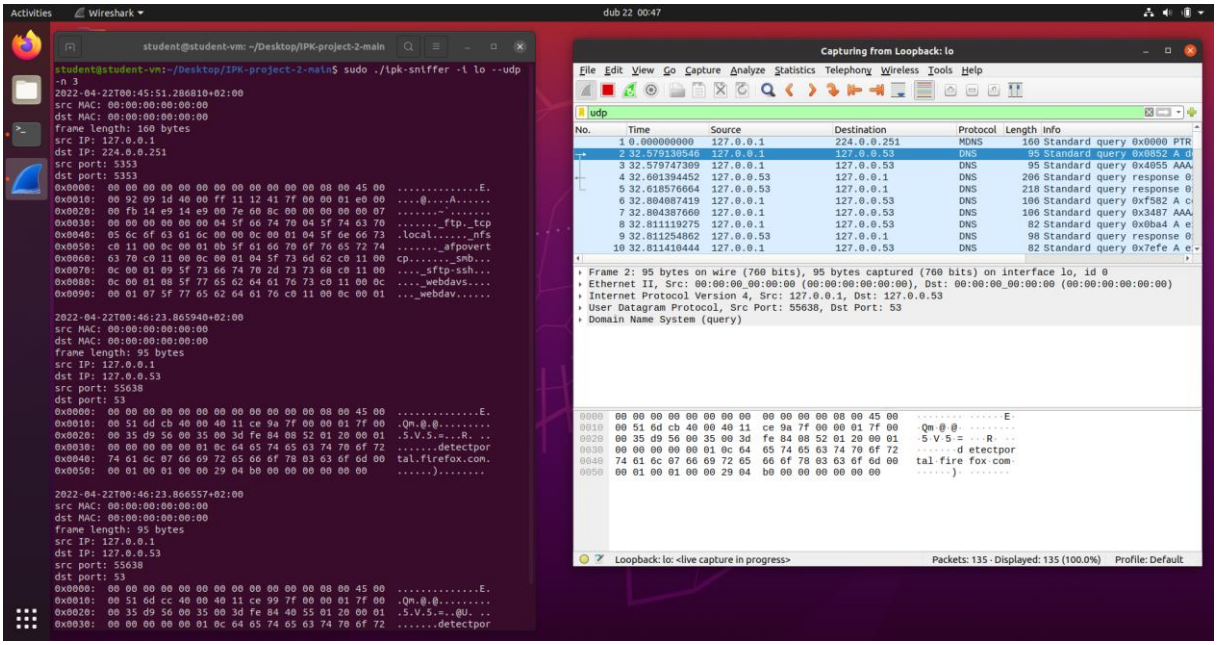
# Testování

Na závěr příkládám několik snímků z testování. Testoval jsem pochopitelně mnohem více případů, nicméně přikládám jenom několik pro ukázku.

1. **TCP se specifikovaným portem**




2. **První 3 UDP packety**



**Zdroje**

https://www.tcpdump.org/pcap.html

https://en.wikipedia.org/wiki/IPv4

https://www.geeksforgeeks.org/internet-protocol-version-6-ipv6-header/

https://cs.wikipedia.org/wiki/Ethernetov%C3%BD_r%C3%A1mec