

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě  
ZETA - Sniffer paketů

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Naštudovaná literatúra</b>	<b>2</b>
2.1	IPv4 . . . . .	2
2.2	UDP . . . . .	2
2.3	TCP . . . . .	3
<b>3</b>	<b>Implementácia</b>	<b>3</b>
3.1	Spracovanie argumentov . . . . .	3
3.2	Odchytávanie paketov . . . . .	4
<b>4</b>	<b>Testovanie</b>	<b>4</b>



## 2.3 TCP

Aplikácie posielajú sieťou pomocou TCP toky dát s 8-bitovými slabikami a TCP rozdeľuje tok bajtov do segmentov s vhodne zvolenou veľkosťou. TCP potom podáva výsledné pakety Internet protokolu na doručenie internetom TCP modulu na opačnom konci spojenia. TCP vykonáva kontrolu, aby sa uistil, že sa žiaden paket nestratí tak, že dá každému paketu poradové číslo, ktoré na druhom konci opäť TCP modul kontroluje a zabezpečuje tiež, že dáta sú doručené v správnom poradí. TCP spojenie má tri fázy: nadviazanie spojenia, prenos dát a ukončenie spojenia. Na nadviazanie spojenia sa používa tzv. 3-way handshake. 4-way handshake sa používa na ukončenie spojenia. [3][4]

		TCP segment header																															
Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0		N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size																
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if <i>data offset</i> > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

Obrázek 3: TCP formát hlavičky [4]

## 3 Implementácia

Implementácia bola rozdelená do dvoch celkov, spracovanie argumentov a odhytávanie paketov. Hlavný súbor programu je `main.c`, z ktorého sa volajú nosné funkcie modulov. V súbore `error.h` sú zadefinované chybové kódy, ktoré program vracia ak nastane jedna z chýb. Program sa prekladá za pomoci nástroja **Make**. Príkazy na preloženie sú definované v súbore `Makefile`. Ak nie je programu odovzdaný parameter *interface*, program vypíše všetky rozhrania, ktoré nájde funkcia `pcap_findalldevs()`. Celková štruktúra programu na odchytyvanie packetov bola inšpirovaná zdrojmi [5],[6].

### 3.1 Spracovanie argumentov

Implementované v súboroch `args.c` a `args.h`. Spracovanie argumentov prebieha za pomoci využitia knižnice `getopt.h`. Používa sa funkcia `getopt_long()` z dôvodu podpory dlhých (`--opt`) ako aj krátkych (`-o`) argumentov. V prípade nevalidných vstupných argumentov sa ukončí program s chybovým kód **ERR\_ARGS**, ktorý je zadefinovaný ako **1**.

## 3.2 Odchytyvanie paketov

Implementované v súboroch `sniffer.c` a `sniffer.h`. K odchytyvaniu paketov sa využíva knižnica `pcap.h`. Z hlavnej funkcie `start_sniffing()` sa volajú podfunkcie na nastavenie filtra, otvorenie rozhrania na odchytyvanie paketov a odchytyvací cyklus `pcap_loop()`. Funkciu `loop_callback()` volá cyklus vždy keď zachytí packet. Táto funkcia packet spracuje a vypíše jeho údaje na štandardný výstup. Preklad IP adres na hostname je uskutočnený pomocou funkcií `getaddrinfo()` a `getnameinfo()`. V prípade, že sa nepodarí preložiť, je vypísaná IP adresa. Ak je nastavený filter na UDP packety, tak sa preklad IP adres zacyklý, pretože DNS využíva UDP. Pri každom zachytenom packete sa vytvorí ďalší na preloženie adresy. Odchytyvanie IPv6 paketov nie je podporované, keďže sme to nemali ako testovať. Nepodarilo sa nám na našej domácej sieti na žiadnom počítači (testované boli 3) prijať či odosielať IPv6 packety. Či už s využitím `tcpdump`, `Wireshark` alebo online testerov.

## 4 Testovanie

Program bol testovaný za pomoci nástroja **Wireshark**. Obom boli zadané rovnaké filtre a boli spustené. Zachytené packety v oboch programoch boli navzájom porovnané, či sa zhodujú.

The image displays two side-by-side windows. The left window is Wireshark, showing a packet capture on interface 'eth0'. The filter is 'tcp.port == 80'. The packet list shows several HTTP GET requests to '192.168.0.105'. The selected packet (No. 868) is expanded, showing the raw data and the packet details pane. The details pane shows the Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol layers. The packet bytes pane shows the raw data in hexadecimal and ASCII. The right window is a custom application titled 'proj2: ./sniffer'. It displays the same network traffic captured by the custom sniffer. The output shows the source and destination IP addresses, the source port, and the destination port, along with the raw data in hexadecimal and ASCII.

## Reference

- [1] “IPv4,” April 30, 2020. <https://en.wikipedia.org/wiki/IPv4>.
- [2] “User Datagram Protocol.” Wikipedia. Wikimedia Foundation, April 23, 2020. [https://en.wikipedia.org/wiki/User\\_Datagram\\_Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol).
- [3] Mitchell, Bradley. “An Inside Look at TCP Headers and UDP Headers.” Lifewire. Lifewire, April 1, 2020. <https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970>.
- [4] “Transmission Control Protocol.” Wikipedia. Wikimedia Foundation, April 29, 2020. [https://en.wikipedia.org/wiki/Transmission\\_Control\\_Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol).
- [5] Tcpdump Group. “Tcpdump/Libpcap Public Repository.” TCPDUMP/LIBPCAP public repository. The Tcpdump Group. Accessed May 2, 2020. [https://www](https://www.tcpdump.org/)
- [6] NanoDano. “Using Libpcap in C.” DevDungeon, October 28, 2018. [https://www.devdungeon.com/content/using-libpcap-c?fbclid=IwAR2NP-LDruBjX-Hwa1To67uX6\\_4KmmxAxWzbT\\_6TIpOkG93p2bR3o4A9-XU](https://www.devdungeon.com/content/using-libpcap-c?fbclid=IwAR2NP-LDruBjX-Hwa1To67uX6_4KmmxAxWzbT_6TIpOkG93p2bR3o4A9-XU).