ISA PROJEKT 2023/2024

MONITORING TOOL FOR DHCP POOL UTILIZATION

Matěj Nešuta
Vysoké učení technické v Brně, Fakulta informačních technologií
Síťové aplikace a správa sítí
20. 11. 2023

1 Project assignment

The goal of this project is to implement a simple program in C/C++, which provides statistics about the utilizations of a DHCP pool. The program should work as a console application when analyzing a network interface. When utilization of a prefix surpasses 50% the program should send a syslog message. The program should have an option to parse a .pcap file, instead of choosing a network interface. In this case, the results are printed on the standard output after the parsing is finished.

2 Theoretical basis

The theory needed for the project implementation mainly revolves around the theme of DHCP communication and also the format of DHCP messages. In order to properly monitor the DHCP communication, some theory related to the UDP and IP protocol is also needed.

2.1 L3 and L4 protocols

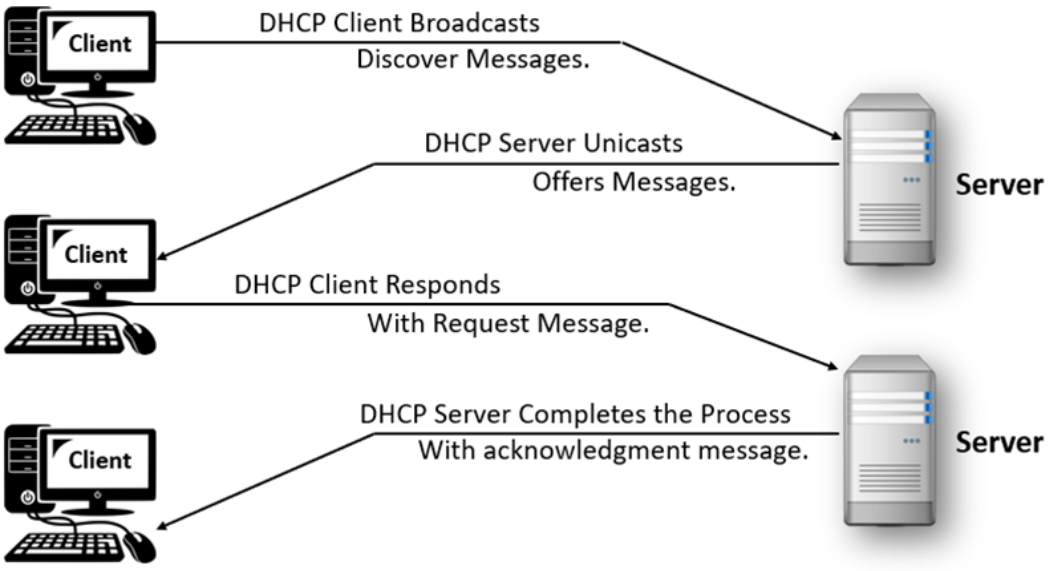
In order to properly get to DHCP messages, IP and UDP header must be removed properly. The length of the IP header is contained in the IP protocol.[1] The length of the UDP header is 8 bytes.[2]

2.2 Dynamic Host Configuration Protocol

The Dynamic Host Configuration Protocol (DHCP) provides a framework for passing configuration information to hosts on a TCP/IP network. DHCP is based on the Bootstrap Protocol (BOOTP), adding the capability of automatic allocation of reusable network addresses and additional configuration options. DHCP captures the behavior of BOOTP relay agents, and DHCP participants can interoperate with BOOTP participants.[3]

2.2.1 DHCP communication

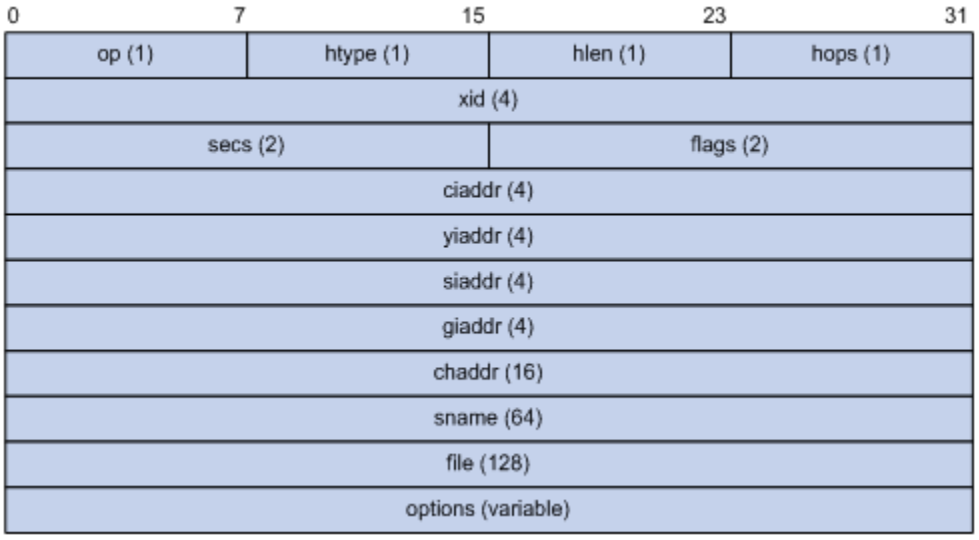
The client broadcasts a DHCPDISCOVER message on its local physical subnet.
The DHCPDISCOVER message MAY include options that suggest values for the network address and lease duration. BOOTP relay agents may pass the message on to DHCP servers not on the same physical subnet. Each server may respond with a DHCPOFFER message that includes an available network address in the 'yiaddr' field (and other configuration parameters in DHCP options). The client receives one or more DHCPOFFER messages from one or more servers. The client broadcasts a DHCPREQUEST message that MUST include the 'server identifier' option to indicate which server has been selected, and that MAY include other options specifying desired configuration values. The server selected in the DHCPREQUEST message commits the binding for the client to persistent storage and responds with a DHCPACK message containing the configuration parameters for the requesting client. The 'yiaddr' field in the DHCPACK messages is filled in with the selected network address. Due to this, the application only checks for the DHCPACK messages. The DHCPACK messages generated as a response to DHCPINFORM messages are ignored by the application.[3]



Picture no. 1: Simple DHCP communication between client and server.[4]

2.2.2 DHCP message structure

The DHCP message structure (https://datatracker.ietf.org/doc/html/rfc2131) consists of 236 octets long header, but my application is mainly interested in “yiaddr”, “file” and “sname” fields. The message also contains an “options” section, which must always start with the so-called “Magic Cookie” and end with the option 255 (End). These options can be further extended into the file and sname sections, given that option 52 (Option Overload) is present in the option field. Another important option is option 53 (DHCP Message Type), which must always be included in the options menu.[5]



Picture no. 2: DHCP message format.[6]

3 Implementation

The application was written in the C language with the use of the libpcap and ncurses libraries.

3.1 The libpcap library

The library is mainly used for monitoring network traffic. The application mainly uses the pcap_open_live[7] or pcap_open_offline[8] function to either listen to a network interface, or parse a .pcap file. The application also uses the pcap_setfilter[9] function to filter some of the traffic from its source. I decided to filter by port 68, because the application is looking for DHCPACK messages. Finally, the pcap_loop[10] function is used upon every packet, which satisfies the filtering condition. Unfortunately, VLAN tagged traffic is not supported.

3.2 The ncurses library

The library gives a robust framework to create a nice looking UI (User Interface) in text mode. It provides functions to create windows etc. These libraries usually come along with curses. One can create applications that contain multiple windows, menus, panels and forms.[11]

The application only uses the basic capabilities of the library, mainly grid options and refreshing. The terminal window is refreshed upon every new pcap_loop function call.

3.3 Source code

My code is split into three .c files (main.c, parser.c and utils.c) and their corresponding header files.

3.3.1 main.c

This file calls the function, which is responsible for argument parsing, then opens the pcap_t * handle in either online or offline mode and then runs the pcap_loop function until a kill signal is sent or the .pcap file is parsed.

3.3.2 utils.c

This file contains smaller functions for the IPv4 comparison, operations related to my bit field data structure, which is used for storing the allocated IPv4 addresses, and also functions related to argument parsing and printing on the stdout/stderr.

3.4 Argument parsing

The application is executed by running

```
./dhcp-stats [-r <filename>] [-i <interface-name>] <ip-prefix> [ <ip-prefix> [ ... ] ]
```

where the -r flag is used for reading the .pcap file, while the -i flag is used for choosing an interface. Only one file/interface can be passed per execution and either interface or file must be supplied. One or more IP prefixes must be supplied as well. The syntax of the prefix is <IP address>/<CIDR prefix> (for example 192.168.0.0/24). When a non-network IP address is supplied (for example 192.168.0.30/24), the rightmost bits are zeroed out by the prefix (resulting in 192.168.0.0/24). Root privileges are also necessary when using the -i flag. When wrong arguments are detected, the application prints help for the user and exits.

This function also allocates all the necessary data structures for the application. I decided to create a bit field, which has the number of bits equal to the number of IP addresses in a certain network pool. When a certain IP address is found in the yiaddr field, the corresponding bit is set to 1.

3.5 DHCP traffic analysis

Firstly, the presence of a specific network interface/.pcap file is checked. Then the already mentioned pcap filter is applied and the main loop is started using the pcap_loop function.

This function invokes a “packet_handler” function, which is responsible for getting to the DHCP message. To get to the DHCP message, the function must remove Ethernet, IPv4 and UDP headers. If a UDP message is not found in the IP payload, the packet is discarded. Then, if the payload is at least 241 octets long (length of the DHCP header + cookie + terminating END option), the function responsible for DHCP parsing is called.

This function first checks if the first 4 octets of the options field contains the DHCP cookie. Then the function checks for option 53 in the options. If not found, the application checks for option 52 and if option 52 is found, then the search for option 53 continues in sname/file fields of the DHCP header. When option 53 is finally found, the application checks if the DHCP message is DHCPACK. If all of these conditions are met, the application compares the IP address found in yiaddr field with every network pool, and if there is a match, it is marked in the bitfield.

After the .pcap file has been parsed in offline mode or on every iteration of the pcap_loop function in the online mode, the total number of bits equal to 1 is counted in every network pool and the final results are printed to stdout. If running in the online mode, when a certain prefix has an allocation of 50% or more, a syslog message is sent.

Sources

[1] RFC 791: Internet Protocol. (1981, September 1). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc791#section-3.1

[2] RFC 768: User Datagram Protocol. (1980, August 1). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc768

[3] RFC 2131: Dynamic Host Configuration Protocol. (1997a, March 1). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc2131

[4] What is DHCP | Relay Agent | DORA Process - A complete Guide. (n.d.). PyNet Labs. https://www.pynetlabs.com/what-is-dhcp/

[5] RFC 2132: DHCP options and BOOTP vendor extensions. (1997, March 1). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc2132

[6] DHCP message format. (n.d.). https://techhub.hpe.com/eginfolib/networking/docs/switches/5120si/cg/5998-8491_l3-ip-svcs_cg/content/436042653.htm

[7] pcap_open_live(3PCAP) man page | TCPDUMP & LIBPCAP. (n.d.). https://www.tcpdump.org/manpages/pcap_open_live.3pcap.html

[8] pcap_open_offline(3PCAP) man page | TCPDUMP & LIBPCAP. (n.d.). https://www.tcpdump.org/manpages/pcap_open_offline.3pcap.html

[9] pcap_setfilter(3PCAP) man page | TCPDUMP & LIBPCAP. (n.d.). https://www.tcpdump.org/manpages/pcap_setfilter.3pcap.html

[10] pcap_loop(3PCAP) man page | TCPDUMP & LIBPCAP. (n.d.). https://www.tcpdump.org/manpages/pcap_loop.3pcap.html

[11] Introduction. (n.d.). https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/intro.html