BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

ISA – Documentation

Jan Kalenda
xkalen07

November 17, 2023

Contents

Intro

The dhcp-stats application is made to monitor the usage rate of specified subnets. This is achieved through capturing DHCP packets by utilizing pcap C library [6].

DHCP protocol

The Dynamic Host Configuration Protocol (DHCP) provides configuration parameters to Internet hosts. DHCP consists of two components: a protocol for delivering host-specific configuration parameters from a DHCP server to a host and a mechanism for allocation of network addresses to hosts.

DHCP is built on a client-server model, where designated DHCP server hosts allocate network addresses and deliver configuration parameters to dynamically configured hosts. Throughout the remainder of this document, the term "server" refers to a host providing initialization parameters through DHCP, and the term "client" refers to a host requesting initialization parameters from a DHCP server.[4]

Our monitor will run on the serverside and monitor client and server traffic.

DHCP packet structure

As stated by the RFC2131[4], the following is the diagram of a DHCP packet:

DHCP Packet Structure			
Offset	Length	Field	Description
0	1	op	
1	1	htype	
2	1	hlen	
3	1	hops	
4	4	xid	Transaction ID
8	2	secs	
10	2	flags	
12	4	ciaddr	Client IP Address
16	4	yiaddr	Your IP Address
20	4	siaddr	Server IP Address
24	4	giaddr	Gateway IP Address
28	16	chaddr	Client Hardware Address
44	64	sname	Server Name
108	128	file	Boot File Name
236	variable	options	DHCP Options

Table 1: DHCP Packet Structure

For our monitor only yiaddr, sname, file, and options are of importance. yiaddr stores the IPv4 address that is being assigned to the client, sname and file are important for use due to option overload. options are variable-sized field of individual options. Each of the options starts with their opcode of the length of one octet. After this octet, another one indicates the length of the option, the maximum being 255 bits. Only options 0, padding, and 255, end, have a fixed size of 0. The most important options for our monitor are options 53, 52, and 255. Other options of interest are 3, 6, and 54.

The client and server should be ready to receive a packet of up to 576 octets. However, the client and server can negotiate larger packets. This includes the ethernet header and UDP header, as DHCP communication happens over the UDP protocol, IP header, and the DHCP datagram itself.

Important options

As stated before, for our monitor, only some of the options are important. They are defined within RFC1533[1][5]

- Option 53 DHCP Message Type: takes on values of 1 to 7. See table 2. Our monitor looks for DHCPACK and stores the assigned address for the subnet use calculation. This option should be the first option in the options. However, it might occur elsewhere as well.
- Option 52 Option Overload: values 1 to 3, value 1 indicates that some options are in the overloaded file field, value 2 indicates that some options are in the overloaded sname field, and value 3 indicates that both sname and file fields are overloaded and additional options can be found there.
- Option 255 End: a fixed size option indicating the end of the option field.
- Option 3 - 11: Numerous options like a router, DNS server, and other server addresses. They can be parts of the specified subnets, and as such, they are being accounted for
- Option 54 Server Identifier: an option of static length of 4, present in DHCOFFER, DHCREQUEST and optionally in DHCPACK and DHCPNAK. The value is an address of the DHCP server.

Value	**Message Type**
1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE

Table 2: DHCP Message Types

Application design

The dhcp-stats monitor is a C++ application built on C++20 standard and utilizes mostly standard library and then ncurses[2] for live updates of statistics and libpcap[6] to capture the packets. The program is divided into three classes: ArgParse, DHCPStats, and Subnet.

Classes

ArgParse

The ArgParse class has a single purpose, and that is to parse the argument line values and validate them. It checks whether only one of the -r -i arguments is being used, stores the subnet addresses with masks as strings, and saves the file or interface to be accessed later. Additionally, a help option -h has been added.

API:

- **ArgParse(int, char)** - constructor of the class, parses the command line arguments
- **get_interface()** - getter for interface
- **get_filename()** - getter for filename
- **get_ips()** - getter for a vector of specified subnet addresses

Noteworthy parts:

- arguments are being checked by regexes, including the expected addresses
- conflicting flags and missing flags are being reported, and the application shuts down
- added help option

Subnet

The Subnet class acts as a storage class with direct access to its public attributes. It has a simple API to calculate usage percentage and the number of used addresses.

API:

- **Subnet(string)** - constructor, takes in the address from the command line arguments and parses it into a usable structure
- **calculate_subnet_fullness()** - returns the percentage of how full the subnet is
- **get_subnet_used_count()** - returns the number of allocated addresses in the subnet
- **ip** - IPv4 address if the subnet
- **capacity** - the maximum capacity of the subnet
- **subnet_mask** - binary representation of the subnet mask
- **prefix** - prefix of the subnet
- **first_ip** - binary representation of the network address of the subnet
- **last_ip** - binary representation of the broadcast address of the subnet

Noteworthy parts:

- The percentage calculation counts with the possibility of /32 prefix

- Usage of a map for faster search of an address

DHCPStats

The core singleton class of the monitor. It captures, processes, and prints the data. During its initialization, the ArgParse class is used to get the monitored subnet addresses from which the actual Subnet objects are being constructed. Based on the selection of either interface, -i, or file, -r, option, the DHCPStats class opens ncurses[2] and listens to the traffic on ports 68 a 67 or parses the file and outputs the results to the standard output.

The class header file also contains a declaration of a DHCPHeader structure, which follows the structure of the packet according to the specified structure 1. The only difference is that our option field starts 4 octets further as our structure takes the magic cookie as part of the structure. The magic cookie is being checked and used to recognize the DHCP packet. Additionally, the entire incoming packet is checked to determine whether the DHCP data can fit in the allocated space.

The class is initiated as a global unique pointer to manage deallocation on exit.

API:

- **DHCPStats(int, char**)** - constructor takes argc and argv from the main function and passes it into ArgParse; a vector of Subnets is also created in the process.
- **DHCPStats()** - The destructor of the DHCPStats object prints the final statistics to the standard output.
- **filename_is_set()** - returns true if the file (-r flag) is set and false otherwise
- **sniffer()** - live packet capturing, utilizes ncurses[2] to update the statistics in real-time. Each of the updates is being separated by a 20ms delay due to buffering issues on some terminals.
- **read_file()** - reads the pcap file and outputs the results on the standard output

Private methods

- **parse_packet(const unsigned char)** - this method initially checks whether the DHCP data at all fits into the captured packet, then checks for the magic cookie. If the checks pass, options offset is calculated, and the method parse_options is called. Based on the return of the parse_options, the DHCP message type is detected, and if it is of value 5 (DHCPACK), the addresses from the packet (client address, addresses from options 3 to 11) are added to the statistics.
- **parse_options(unsigned char, int*, int)** - this method parses the options and looks for options 52, 53, 54, and 3 to 11. This method runs a loop that ends when the END option is encountered or the entire length of the option field has been parsed. Addresses from options 3 to 11 and 54 are stored in a class attribute vector.
- **print_stats()** - this method serves to update the ncurses[2] output of live capture or print the initial statistics.
- **update_stats()** - this method updates the internal status of the Subnets with the vector of addresses from parse_options().

Attributes

- lines - variable holding number of lines managed by ncurses[2]
- subnets - map of all monitored subnets
- interface - a string representing the interface

- filename - string representing the file name and path
- handle - pcap handle used to loop over packets

Noteworthy parts

- Usage of the same parsing method for all options, even overloaded ones
- Usage of macros to recognize and count with a single layer of VLAN, calculating offset for options
- Decision to print data directly to standard output when launched to read a file.
- Waiting 20ms when updating live status
- Subnets sorted by their capacity

Capabilities and limitations

This implementation of a DHCP server monitor is not a fully feature-rich one, but it offers basic functionality with the following capabilities and limitations:

Capabilities

- Monitor the usage rate of any subnet specified in the arguments
- Detect invalid IPv4 addresses in arguments
- Capture packets from UDP traffic on ports 67 and 68
- Detect router, time server, name server, domain name server, log server, cookie server, LPR server, impress server, and location server addresses from the packets and count them toward the statistics
- Parse overloaded file and sname parts of the DHCP packet
- Ignore malformed packets, based on the length and magic cookie
- Update statistics in real-time in ncurses[2], updating only required lines
- Logging of a warning when subnet usage is more than 50%
- Smooth management of SIGINT, SIGTERM, and SIGKILL signals

Limitations

- Missing support for the lease time
- Missing support for other options[5][1]
- Missing support for IPv6
- Ncurses library still leaks memory[3]
- More advanced support for argument checking
- Missing support for IP tunneling and numerous layers of VLAN

UML

The following is the simple UML class diagram of the monitor:

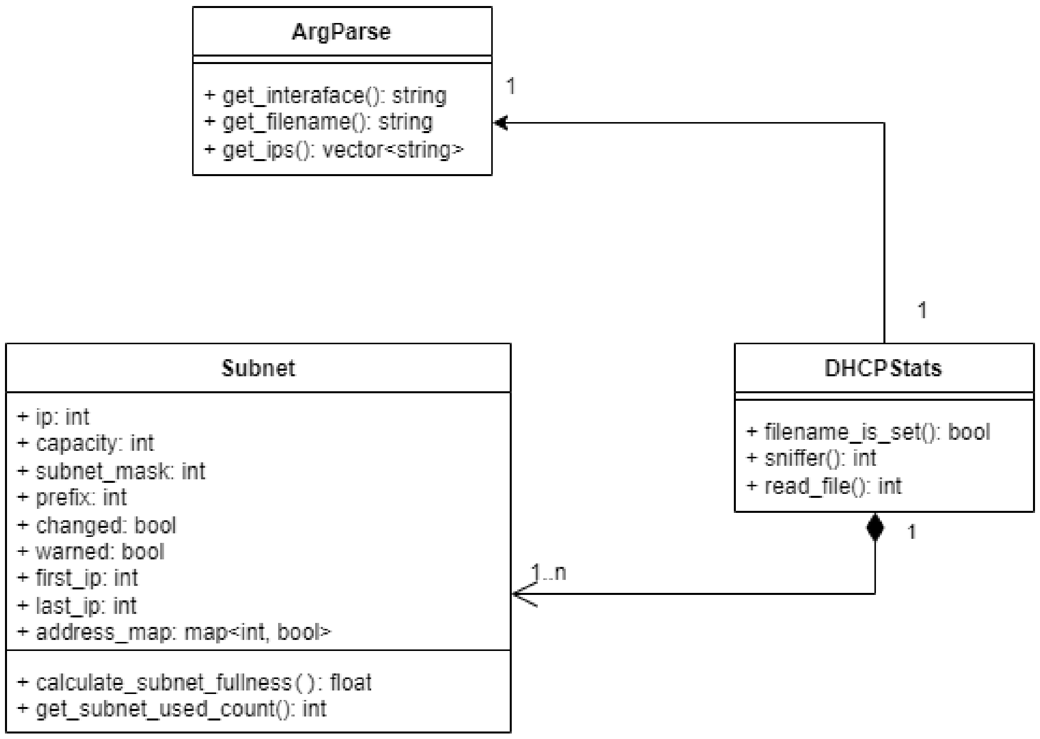

Figure 1: UML diagram of the monitor

Use and examples

To use the monitor you need UNIX based system and a compiler supporting at least C++20. You will also need ncurses and pcap libraries.

Requirements

- C++ compiler with C++20 standard or higher
- libpcap.h library
- ncurses.h library

Steps:

- Compile the code: The monitor comes with a Makefile, so you can simply type "make"
- To run the monitor, type: "./dhcp-stats" with the following options. At least one subnet has to be specified and either interface or file has to be specified, but never both.
  - **-r FILE** where the FILE is a pcap file with recorded traffic
  - **-i INTERFACE** interface to listen on and capture packets on
  - **-h** prints out help
- When started on live capture, to exit, send SIGINT (Control + c) to the monitor window

Example output:

```
IP-Prefix        Max-hosts        Allocated addresses        Utilization
192.168.0.0/22   1022             50                         4.89%
192.168.1.0/24   254              50                         19.69%
172.16.32.0/24   254              0                          0.00%
192.168.1.0/26   62               50                         80.65%
192.168.1.0/27   30               30                         100.00%

prefix 192.168.1.0/27 exceeded 50% of allocations
prefix 192.168.1.0/26 exceeded 50% of allocations
```

References

[1] Steve Alexander and Ralph Droms. *DHCP Options and BOOTP Vendor Extensions*. RFC 1533. Oct. 1993. DOI: `10.17487/RFC1533`. URL: `https://www.rfc-editor.org/info/rfc1533`.

[2] Thomas E. Dickey. *ncurses*. online. Mar. 2022. URL: `https://invisible-island.net/ncurses/`.

[3] Thomas E. Dickey. *ncurses memory leak*. online. Mar. 2022. URL: `https://invisible-island.net/ncurses/ncurses.faq.html#config_leaks`.

[4] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Mar. 1997. DOI: `10.17487/RFC2131`. URL: `https://www.rfc-editor.org/info/rfc2131`.

[5] Ralph Droms and Steve Alexander. *DHCP Options and BOOTP Vendor Extensions*. RFC 2132. Mar. 1997. DOI: `10.17487/RFC2132`. URL: `https://www.rfc-editor.org/info/rfc2132`.

[6] Van Jacobson, Craig Leres, and Steven McCanne. *libpcap documentation*. online. Mar. 2023. URL: `https://www.tcpdump.org/manpages/pcap.3pcap.html`.