CAS765 Programming Assignment 1 – Trace Analysis

Due date: Oct 21st, 2013

In this programming assignment, you will develop C code for packet analysis instead of using Wireshark. You will be provided with sample codes needed for completing this task except for a few coding blocks to fill in.

I. Programming environments

Download program_assignment.tgz from the course website. Decompress the file using "tar xfz *filename*". In the folder you will find the following files:

- *frame_handler.c* and *frame_handler.h* contain the main routines to decode frames. *frame_handler.c* is the main file you will modify.
- *radiotap-parser.c* and *radiotap-parse.h* contain the routines to handle the radiotap headers.
- *packet_analyzer.c* is the entry point of the analyzer code. It takes a single argument specifying the pcap trace file and passes it to the *handle_packet* routine, which in turns call *process_ratiotap_header* and routines in *frame_handler.c* to further decode the frame.
- *ieee802_11.h* and *ieee80211_radiotap.h* define data structures related to 802.11 frames
- *Makefile* You can type "make" to compile all the files.
- *ieee80211_frame_format.pdf* is abridged from the IEEE 802.11 standard documenting the 802.11 frame format.

The sample codes have been tested on Red Hat Enterprise Linux Server release 5.9 (kernel 2.6.18-348.12.1.el5), Mac OSX mountain lion (10.8.4), and OSX snow lepard (10.6.8) with GCC 4.1.2 and 4.2.1. I did not make the extra effort to make it platform independent. The best option would be to use mills.cas.mcmaster.ca. If you choose to use OSX, please make sure to install the libpcap library, which should've been there if you already had Wireshark running.

For development and debugging, you may use command line tools such as Vim and gdb. Alteratively, you may use Eclipse CDT as IDE (http://www.eclipse.org/cdt/) (not installed on mills).

II. Overview

In class, we have introduced MAC layer frames. For the purpose of completing the assignment, you need to be familiar with the format of the MAC layer frames, management frame header and body. You also need to pay attention to byte ordering.

In the IEEE 802.11 standard, the following convention is defined regarding the bit and byte order (Page 34, 7.1.1 [1]).

> *The MAC protocol data units (MPDUs) or frames in the MAC sublayer are described as a sequence of fields in specific order. Each figure in Clause 7 depicts the fields/subfields as they appear in the MAC frame and in the order in which they are passed to the physical layer convergence protocol (PLCP), from left to right.*
>
> *In figures, all bits within fields are numbered, from* 0 *to* k*, where the length of the field is* $k+1$ *bit. The octet boundaries within a field can be obtained by taking the bit numbers of the field modulo 8. Octets within numeric fields that are longer than a single octet are depicted in increasing order of significance, from lowest numbered bit to highest numbered bit. The octets in fields longer than a single octet are sent to the PLCP in order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.*

In other words, 802.11 frames follow the *little endian byte order* (i.e., the least significant byte comes first) as opposed to the *network byte order* (*big endian)*, where the most significant byte comes first. The functions

©2014 Rong Zheng

e16_to_cpu, *le32_to_cpu*, *pletohs* defined in byteorder.h and frame_handler.h can be used to convert from little endian bit and byte orders to host orders, respectively. The host byte order is architecture dependent.

Figure 1 and Figure 2 give the MAC frame format and frame control (FC) field. In Figure 2, the order of the bits is also shown. As an example, consider the valid type and subtype combination in Figure 3. For a probe response frame (type 0, subtype 0101), the first byte in the FC field from low (B0) to high (B1) is $00|00|1010$, which is equivalent to 0x50.

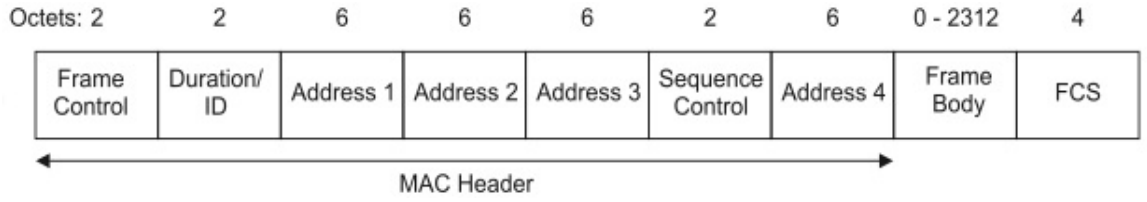

Fig. 1. MAC frame format

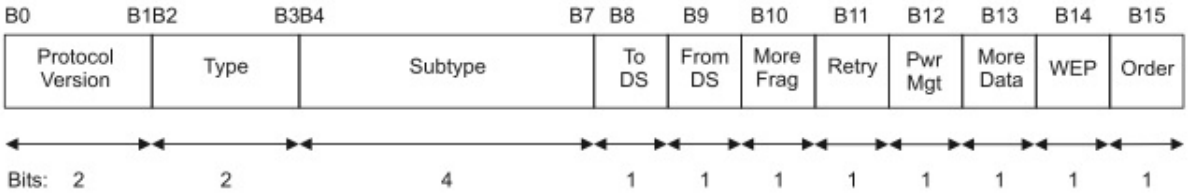

Fig. 2. Frame control field

III. Parsing 802.11 management frame

In this project, you will complete the code *frame_handler.c* to process MAC management frames (Figure 4). For simplicity, you only need to handle management frames originated from the access point, namely, association and re-association response, probe response, and beacon messages. Within management frames, fixed-length mandatory frame body components are defined as fixed fields; variable length mandatory and all optional frame body components are defined as information elements. The fixed fields are different for different frame types and subtypes. Details can be found in [1] (chapter 7). Elements are defined to have a common general format consisting of a 1 octet Element ID field, a 1 octet length field, and a variable-length element-specific information field. Each element is assigned a unique Element ID as defined in this standard. The Length field specifies the number of octets in the Information field. A vendor specific information field with element ID 221 consists of 3-byte Organizationally Unique Identifier (OUI) followed by a variable number of three-tuples <info_tag (1 byte), info_length (1 byte), info_content)>. Based on the information from [2], we know that the first tuple for Merunetwork (OUI = 00-0c-e6) corresponds to the ID of the AP, the 2nd 3-tuple corresponds to the AP's serial MAC address, and the 3rd 3-tuple gives radio index and remaining tuples are unknown.

In handling each management frame (DECODE_MGMT_FRAME), one needs to first decode and print out the associated fixed fields, and then the information element (PARSE_ELEMENTS). The MGMT_BODY_T structure is defined to store information from the fixed field. In parsing the information elements, we only decode the vendor specific elements with element ID 221 and vendor (OUI = 00-0c-e6). A sample output from the field is given below:

```
(MERUNet) AP ID: 39 AP MAC_ADRESS: 00:0c:e6:07:90:fd  Radio Index: 1
```

Requirements: Your code should perform the following tasks:

- Decode and print out the fixed frame body components of association and re-association response, probe response, and beacon messages.

Fig. 3. Valid type and subtype combinations

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0000	Association request
00	Management	0001	Association response
00	Management	0010	Reassociation request
00	Management	0011	Reassociation response
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	0110–0111	Reserved
00	Management	1000	Beacon
00	Management	1001	Announcement traffic indication message (ATIM)
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101–1111	Reserved
01	Control	0000–1001	Reserved
01	Control	1010	Power Save (PS)-Poll
01	Control	1011	Request To Send (RTS)
01	Control	1100	Clear To Send (CTS)
01	Control	1101	Acknowledgment (ACK)
01	Control	1110	Contention-Free (CF)-End
01	Control	1111	CF-End + CF-Ack
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null function (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack + CF-Poll (no data)
10	Data	1000–1111	Reserved
11	Reserved	0000–1111	Reserved

- Decode and print out the vendor specific elements with element ID 221 and vendor (OUI = 00-0c-e6) in the variable length frame body components.
- Put all codes in a tar.gz ball and email to the instructor.

REFERENCES

[1] Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications
[2] Making Sense of Meru Decodes, http://www.cwnp.com/cwnp_wifi_blog/making-sense-of-meru-decodes/

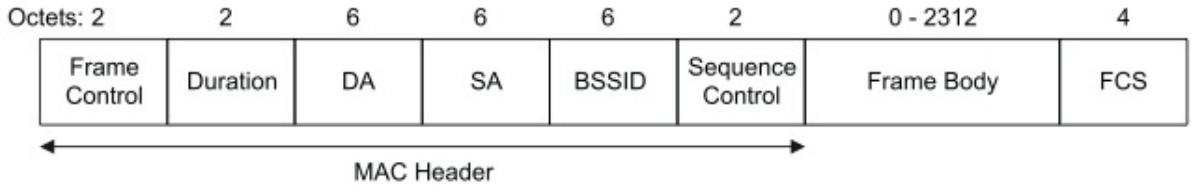

Fig. 4. Management frame format