Part 1.

1. (a) Refer to line 24: CNET_start_timer(EV_TIMER1, 1000000, 0);
   This line is responsible for rescheduling EV_TIMER1 to occur again, so we can change the 2nd parameter to 10000000 (10s).

   (b) Refer to line 21: printf("%3d.\t%s\n", which, ((which%2) == 1) ? "tick" : "\ttock");
   This printf statement need to be modified in order to represent the new output structure. Also, before the printf statement, there are some extra statements we need to add, just for the purpose of reverting time into string format.

   The modified program will look like:

```
#include <cnet.h>
static EVENT_HANDLER(timeouts)
{
    static int which = 0;
    static CnetTime time = 0;
    char *timeString;

    ++which;

    if (which == 1){
        time = (CnetTime) which * 1000000;
    } else {
        time = (CnetTime) time + 10000000;
    }
    timeString = CNET_format64(time);

    printf("%3d.\t%s (time = %s)\n", which, ((which%2) == 1) ? "tick" : "\ttock", timeString);

//  RESCHEDULE EV_TIMER1 TO OCCUR AGAIN IN 1SEC
    CNET_start_timer(EV_TIMER1, 10000000, 0);
}

EVENT_HANDLER(reboot_node)
{
//  INDICATE THAT WE ARE INTERESTED IN THE EV_TIMER1 EVENT
    CHECK(CNET_set_handler(EV_TIMER1, timeouts, 0));

//  REQUEST THAT EV_TIMER1 OCCUR IN 1SEC, IGNORING THE RETURN VALUE
    CNET_start_timer(EV_TIMER1, 1000000, 0);
}
```

2.

```
typedef struct {
        CnetPosition position; // a struct that stores the device coordinates in 3D:
                                    // x, y, and z
        CnetNodeType devicetype; // device type, e.g., NT_MOBILE
        CnetNodeInfo *deviceinfo; // a pointer to a struct that stores device's information,
                                   // e.g., nodenumber, address, nodename, etc.
        CnetLinkInfo connection[32]; // an array of structs, each struct stores
                                      // information about each indicent link
                                      // (e.g., linktype, bandwidth, etc.)
        CnetTimerID mainTimer; // a timer identifier
} DEVICE;
```

3.

The CHECK macro will pop up a new window highlighting the file and line number of runtime error in order to indicate the cause of the returned error value(typically -1).

If the purpose is to report the API error and not exiting from the simulation, then the program should report the error message from cnet_errno, and dose not call the CNET_exit() function.

4.

FRAME_SIZE(f) is different from sizeof(f). FRAME_SIZE(f) can always precisely calculate the size of a frame(the size the frame should be equal to), while sizeof(f) might provide undesired result if there is something bad happen to the message part(e.g loss of message).
So, sizeof(f) cannot record the original size of a frame, while FRAME_SIZE(f) is able to do that. The existence of difference between FRAME_SIZE(f) and sizeof(f) can prove the message is corrupted.

5. (a) Observation of **beta**: (Min -> 1578258; Max -> 1834093)
   The range of observed values of beta is determined by the setting of propagation delay and wan-jitter, and the impact of wan-jitter is much larger.
   (b) In order to make **beta** approximated equal to 4 seconds, we need to change the wan-jitter, and also modify propagation delay. For instance, setting propagation delay to 1600 msec and setting wan-jitter to 400 msec will work.

6.

unsigned short msg_1[] = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46};
unsigned short msg_2[] = {'A', 'B', 'C', 'D', 'E', 'F'};
unsigned short msg_3[] = {'@', 'A', 'B', 'C', 'D', 'K'};

CNET_IP_checksum(msg_1, 6) -> 65337
CNET_IP_checksum(msg_2, 6) -> 65337
CNET_IP_checksum(msg_3, 6) -> 65340

The checksum of msg_1 and msg_2 are the same, which is not really a surprise, because the 6 values stored in msg_1 are just ascii representation of the values in msg_2, which means they are the same.

7.

unsigned short msg_1[] = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46};
unsigned short msg_2[] = {'A', 'B', 'C', 'D', 'E', 'F'};
unsigned short msg_3[] = {'@', 'A', 'B', 'C', 'D', 'K'};

CNET_crc16(msg_1, 6) -> 22826
CNET_crc16(msg_2, 6) -> 22826
CNET_crc16(msg_3, 6) -> 23594

The checksum of msg_1 and msg_2 are the same, which is not really a surprise, because the 6 values stored in msg_1 are just ascii representation of the values in msg_2, which means they are the same.

Program Report

- Design Overview

- Program requires user providing topology file as input file.
- Program mainly focuses on the physical layer protocol.
- Program defines global variables for the purpose of recording information of neighbours of each node.
- Design of program mimics real world frame transmission within internet
- The main idea of discovering the info of neighbours is sending back the info of such neighbour via the DISCOVER_ACK frame.
- Program will check if each node successfully discover all neighbours, and if it does not because of the lose of transmitted frame, program will redo the discovery process for that specific node again.
- User can check the neighbour info of certain node by clicking the button within GUI.

- Program Status

The program is basically complete at this point, witch means that it fulfill the requirements and instructions of the specification.

There are indeed many difficulties during the implementation of the program, I ran into may problems of understanding how the simulation of physical layer of cnet actually works since I am not very familiar with cnet.

To be more specific, I did lots of reading specifically on the function CNET_write_physical and function CNET_read_physical, since these two functions are absolutely significant for this lab. My misunderstanding of these two leaded me to design and implement incorrect physical_ready function, which is the function will be called when there is a message ready for delivery. Luckily, the difficulties are finally resolved and conquered.

- Testing and Results

The overall testing strategy is basically try and error. I tried serval different topology file as input file and check the performance of the program. If there exists such unexpected result, I used printf function to discover the reason of that and making correct modification.

API error I encountered:
CNET_write_physical()
CNET_read_physical()

- Acknowledgement

The design and implementation of the program are completed individually with consultation to lab slides/videos, eclass slides and official cnet document.

References:

http://webdocs.cs.ualberta.ca/~c313/cnet-3.4.1-php/index.php (cnet official website)
https://eclass.srv.ualberta.ca/course/view.php?id=67469(CMPUT313 eclass page)