EECS 325/425: Computer Networks
Project #2
Due: March 3, by 3:00pm

This project involves analyzing a *packet trace* that contains Ethernet and IP packet headers as observed on an operational network. The packet trace your program will operate on will be given by the "`-r tracefile`" command-line option. If this option is not present your program should generate an error and exit.

The format of this packet trace is as follows:

Each packet in the trace is prefaced by twelve bytes of meta information about the packet, as follows:

- **Bytes 1–4:** The number of seconds since Unix epoch (midnight GMT on January 1 1970). This value is in *network byte order*. See *ntohl()*.
- **Bytes 5–8:** The number of microseconds since the second given in the first field. This value is in *network byte order*. See *ntohl()*.
- **Bytes 9–10:** The number of bytes captured in the trace file for this packet (`caplen`). This value *does not* include the 12 bytes of meta information included for the given packet (i.e., it is *only* the packet portion). This will dictate the degree to which the packet can be processed. This value is in *network byte order*. See *ntohs()*.
- **Bytes 11-12:** These bytes are not used for this project and while they are present, must be ignored.

The above information will dictate how much can be understood about the included packet. If the `caplen` is at least 14 bytes the full Ethernet frame header is included in the trace file, as follows:

- **Bytes 13–26:** The Ethernet header of the packet: 6 bytes for the destination address, 6 bytes for the source address and 2 bytes for the type, in this order.
  Note: The Ethernet *preamble* is not included in the trace file. Likewise, the Ethernet trailer that includes the CRC is also not included in the trace file.
  Note: The type field is encoded in *network byte order* in the trace file. See *ntohs()*.

If the `caplen` is at least 34 bytes and the Ethernet "type" field indicates IP (0x0800) you will also be able to process the IP header, as follows:

- **Bytes 27–46**: fixed IP header
  Note: Look for "struct iphdr" in /usr/include/netinet/ip.h for a handy way to deal with the IP header in memory.
  Note: The 16 and 32 bit fields in the header are in network byte order and you'll need to use *ntohs()* and *ntohl()* to find the proper values.

- **Bytes 47**–: IP options
  The presence of options is dictated by the IP header length field and the value of `caplen`.

Note: You will need to use the `caplen` to guide you to the next packet in the trace. That is, after `caplen` bytes you will find the twelve bytes of meta information for the next packet.

Your program will operate in one of the five modes described next. These options are to work independently and cannot be combined. That is, if more than one of these command line options appears your program must generate an error and exit. The order of the command line options can be arbitrary (with the caveat that a filename must follow the `-r` option).

Part 1: Trace Summary

When the "`-s`" option is given on the command line, the program must print the following four line summary of the trace file to standard output:

```
PACKETS: TOT_NUM_PKTS
FIRST: TIMESTAMP_FIRST_PKT
LAST: TIMESTAMP_LAST_PKT
DURATION: TRACE_DURATION
```

Each label should start at the beginning of the line and end with a colon. One space follows the colon. The value of the field follows the space. The "`PACKETS`" line indicates the number of packets found in the trace file in decimal without any padding. The "`FIRST`" line indicates the time since Unix epoch of the first packet in the trace. The time should be printed as a floating-point decimal number to microsecond precision—i.e., six digits after the decimal point—and without padding, which will require you to leverage the first two fields in the meta information. The "`LAST`" line indicates the time since Unix epoch of the last packet in the trace. Again, the time should be printed as a floating-point decimal number to microsecond precision—i.e., six digits after the decimal point—and without padding. Finally, the "`DURATION`" field should indicate the number of seconds—to microsecond precision—covered by the trace. That is, the number of seconds that elapse between the first and last packets that appear in the trace. Again, this value should be reported as a floating point decimal number without padding.

Sample output:

```
PACKETS: 45
FIRST: 1105102605.095006
LAST: 1105102605.152978
DURATION: 0.057972
```

Part 2: Dumping Ethernet Headers

When the "`-e`" option is given your program will simply dump information from the Ethernet frames found in the trace to standard output in the following format:

```
timestamp eth_src_addr eth_dst_addr type
```

The `timestamp` should be printed as a floating point number to microsecond resolution—i.e., six decimal places—that represents the time since epoch for the given packet. The `eth_src_addr` and `eth_dst_addr` are the Ethernet addresses of the source and destination of the packet, respectively. These addresses should be presented in standard MAC address format, as follows: (*i*) each byte in the address should be represented by a 2-digit hexadecimal number (i.e., pad if necessary), (*ii*) any letters that appear in the address must be printed in lower case and (*iii*) a colon (":") must separate the bytes in the address. A example of a properly formatted address is "00:23:df:0f:7c:f7". The `type` should be presented as a four-digit hexadecimal number prepended by "0x" (e.g., "0x0800"). For numbers with less than four digits, padding is required.

The output should include one line for each packet in the given trace file. The fields on each line are to be separated by a single space. No extraneous output at the beginning or end of the lines may occur. The order of the lines should be the same as in the given packet trace.

In the case when the trace does not contain the entire Ethernet header, your program must print "Ethernet-truncated" after the timestamp.

Sample output:

```
1313035171.359437 00:1e:4f:20:33:51 ff:ff:ff:ff:ff:ff 0x0800
1313035172.215284 00:22:19:02:73:5b 33:33:ff:96:21:82 0x86dd
1313035173.452592 Ethernet-truncated
```

Note: Padding with zeros on the left to always print two or four byte numbers is easy with the right format specifier within the *printf()* routine. See the man page.

Part 3: Dumping IP Headers

When the "`-i`" option is given your program will simply dump information from the IP headers found in the trace to standard output in the following format:

```
timestamp ip_src ip_dst ip_hl ip_prot ip_ttl
```

Where the above fields are:

- `timestamp`: The packet's timestamp, printed as a floating point number to microsecond resolution (i.e., six decimal places) that represents the time since epoch without padding.
- `ip_src`, `ip_dst`: The source and destination IP addresses. These should be printed in "dotted decimal" notation without padding.
- `ip_hl`: The length of the IP header *in bytes* (i.e., not precisely what is in the packet header), printed as a decimal number without padding.
- `ip_prot`: The protocol number of the transport protocol encapsulated within this IP packet, printed as a decimal number without padding.
- `ip_ttl`: The time-to-live value present in the packet, printed as a non-padded decimal number.

Your output should include one line per packet. The order of the packets in your output must match the original order of the packets in the trace file.

Note: If the trace does not contain the full Ethernet header your code will not be able to determine whether the given packet is an IP packet and therefore you should print "unknown" after the timestamp.

Note: If the trace contains the full Ethernet header but the packet is not an IP packet, your program should print "non-IP" after the timestamp.

Note: If the trace contains a full Ethernet header indicating the packet is an IP packet, but does not contain the full IP header—including options—your program should print "IP-truncated" after the timestamp.

Examples:

```
1313035170.893478 unknown
1313035171.359437 129.22.150.71 129.22.175.13 20 6 127
1313035171.592724 129.22.175.13 129.22.150.71 20 6 62
1313035172.215284 non-IP
1313035173.894241 IP-truncated
```

Part 4: Counting Packet Types

When the "`-t`" option is given your program will dump information about the packet types found in the trace file to standard output in the following format:

```
ETH: ETH_FULL_PKTS ETH_PARTIAL_PKTS
NON-IP: NUM_NON_IP_PKTS
IP: IP_FULL_PKTS IP_PARTIAL_PKTS
SRC: NUM_IP_SRC_ADDRS
DST: NUM_IP_DST_ADDRS
TRANSPORT: NUM_TCP_PKTS NUM_UDP_PKTS NUM_OTHER_PKTS
```

Each label will be printed as shown above, followed by a colon (":") and a single space. The value(s) will then printed as non-padded decimal numbers. There should be no extraneous characters at the beginning or end of the lines. The values are as follows:

- The "ETH" line will report two values: (*i*) the number of packets in the trace that have a fully intact Ethernet header (i.e., `caplen` is at least 14 bytes) and (*ii*) the number of packets in the trace that do not have a fully intact Ethernet header (i.e., `caplen` is less than 14 bytes).
- The "NON-IP" line will report the number of non-IP packets (i.e., those with a full Ethernet header, but an Ethernet type that is not 0x0800).
- The values reported on the "IP" line will be: (*i*) the number of packets in the trace that have their IP headers fully intact and (*ii*) the number of packets in the trace that do not have their IP headers fully intact.
- The "SRC" line will report the number of IP addresses observed in the trace sending at least one packet. Source addresses from packets with partial IP headers must be ignored.
- The "DST" line will report the number of IP addresses observed in the trace receiving at least one packet. Destination addresses from packets with partial IP headers must be ignored.
- Finally, the "TRANSPORT" line will include three values, as follows: (*i*) the number of TCP packets included in the trace (i.e., IP protocol 6), (*ii*) the number of UDP packets included in the trace (i.e., IP protocol 17) and (*iii*) the number of packets in the trace that are neither TCP nor UDP.

  Note: Only count packets for which the full IP headers are present.

The following equalities hold:

$$\texttt{NUM\_NON\_IP\_PKTS} \leq \texttt{ETH\_FULL\_PKTS} \quad (1)$$

$$(\texttt{IP\_FULL\_PKTS} + \texttt{IP\_PARTIAL\_PKTS}) \leq \texttt{ETH\_FULL\_PKTS} \quad (2)$$

$$(\texttt{NUM\_TCP\_PKTS} + \texttt{NUM\_UDP\_PKTS} + \texttt{NUM\_OTHER\_PKTS}) = \texttt{IP\_FULL\_PKTS} \quad (3)$$

$$\texttt{NUM\_IP\_SRC\_ADDRS} \leq \texttt{IP\_FULL\_PKTS} \quad (4)$$

$$\texttt{NUM\_IP\_DST\_ADDRS} \leq \texttt{IP\_FULL\_PKTS} \quad (5)$$

Example output:

```
ETH: 10 5
NON-IP: 2
IP: 7 1
SRC: 3
DST: 4
TRANSPORT: 4 1 2
```

Part 5: Producing a Traffic Matrix

When the "`-m`" option is given your program will print a traffic matrix of the given trace file. A traffic matrix shows the amount of traffic sent from each sending IP address found in the trace to each recipient with which it communicates. Your program will report the amount of traffic both in terms of (*i*) the number of packets and (*ii*) the total IP-level bytes (including the IP headers) carried by the packets recorded in the trace. The format of your output will be:

```
SRC_IP DST_IP NUM_PKT IP_DATA_VOL
```

The "`SRC_IP`" and "`DST_IP`" fields report the hosts involved in this line of output. The addresses must be reported in dotted-decimal notation without padding. The "`NUM_PKT`" field represents the total number of packets the given source sent to the given destination. Finally, the "`IP_DATA_VOL`" field is the total number of IP-level bytes—including IP headers—that the given source sent to the given destination. All fields must be separated by a single space.

Note: The number of IP-level bytes actually present n the trace is different from the number of bytes the packets actually carry. The volume is derived from the packet length in the IP header.

Each (source,destination) IP pair will be printed on its own line of output. The fields will be separated by a single space character. No extraneous characters should appear before the "`SRC_IP`" or after the "`IP_DATA_VOL`" fields. The order of the lines in the output is not important.

Example output:

```
129.22.156.173 129.22.150.71 10 8652
129.22.150.71 129.22.156.173 12 3202
129.22.150.71 192.150.187.12 1 40
```

Notes:

1. Your submission must be a gzip-ed tar file called "CaseID-proj2.tar.gz" that contains all code and a Makefile that by default produces an executable called "`proj2`". Do not include executables or object files in the tarball. Do not include sample input or output files in your tarball. Do not include multiple versions of your program in your submission. Finally, do not include any directories in the tarball.

2. Any notes you wish to include with your submission should be put in a text file called "notes.txt" that is included in your tarball.

3. There will be sample input files and reference output on the class web page by the end of the day on February 7. (Not all sample input will have corresponding reference output.)

4. Print only what is described above. Extra debugging information must not be included. Adding an extra option (e.g., "`-v`" for verbose mode) to dump debugging information is always fine.

5. Do not make assumptions about the size of the input. Your project should handle trace files of arbitrary length.

6. Every source file must contain a header comment that includes (*i*) your name, (*ii*) your Case network ID, (*iii*) the filename, (*iv*) the date created and (*v*) a brief description of the code contained in the file.

7. Do not use spaces in file/directory names.

8. If you encounter or envision a situation not well described in this assignment, please Do Something Reasonable in your code and include an explanation in the "notes.txt" file in your submission. If you'd like to ensure you're on the right track, please feel free to discuss these situations with me.

9. Remember to use good programming practices—including meaningfully naming variables/constants, using modularity, checking for errors, using consistent style, avoiding repetitive code, leveraging reasonable data structures, etc.

10. Remember that your project must build and run on the class server or it will not be graded.

11. Hint: Octal Dump (`od`) will help you understand the packet trace files and therefore help you check that your program is correct.

12. Hint: Errors will be thrown at your project.

13. *WHEN STUCK, ASK QUESTIONS!*
    When asking question via email, please include all your source code and a Makefile.

EECS 425: Computer Networks
Project #2
Due: March 3, by 3:00pm

Tell Me Something I Don't Know!

Project 2 will parse and report various facets of a trace file. Usually, this is just the first step in understanding some networking phenomena. For this portion of the assignment, you will write a short report outlining five aspects of the network you have determined from a reference trace file. These aspects will be over and above what is reported by the analyzer's base functionality.

Notes:

- Your report must be included as a PDF document called "report.pdf" in your submitted tarball. Your report should not only give the properties of the network you uncover, but also sketch how you arrived at those properties.
- The aspects of the trace that you report should be larger than those about a single packet. That is, your "five things" should not be the checksum value from the first five packets. Rather, think about distributions, averages, summations or rankings across many packets in the trace.
- Your results can be reported as numbers, tables or plots.
- Writing clarity is crucial.
- The output of your trace file analyzer will be required to draw out broader insights. Therefore, you will need to plan to have your analyzer finished before the deadline to allow time for using your new tool to analyze the reference trace.
- While the output of your analyzer will be necessary for this portion of the project, it will not likely be sufficient. You may add additional options to your project to track additional features. Or, you may write additional tools—in whatever language you like—to further analyze the output of your trace file analyzer.

  Include the source for any extra tools you build with your submission.

  Explain the additional tools and/or options in your report.
- A reference trace file will be provided on the project web page by Thursday February 9.

Extra Credit: Undergraduates may complete this portion of the assignment for extra credit.