EECS 325/425: Computer Networks
Project #3
Due: Friday April 7, by 3:00pm

This project involves extending your previous packet trace analyzer to analyze the transport layer and the network path. As with the previous project, your program will operate on a trace file given via the "`-r tracefile`" command-line option. If this option is not present your program should generate an error and exit. The trace files will now contain transport layer headers in addition to the previous link and network layer headers.

The basic format of the packet trace is the same as we used for project 2. However, in this project, after the IP header you will find a transport layer header (e.g., TCP or UDP). The format for these headers is given in your book. Additionally, you may find the `struct udphdr` in /usr/include/netinet/udp.h and the `struct tcphdr` in /usr/include/netinet/tcp.h handy. A detailed trace format—which is largely the same as for project 2—is given on the project 3 web page.

Your program will operate in one of three 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).

In all three modes, your program must ignore the following packets:

- Packets that do not contain the full Ethernet header.
- Non-IP packets. I.e., those that do not have an Ethernet type of 0x0800.
- Packets that do not contain the full IP header. (Note: this includes any IP options that may be present.)
- Packets with an IP protocol number that does not indicate TCP (6) or UDP (17).
- UDP packets that do not contain the full UDP header.
- TCP packets that do not contain the full TCP header. (Note: The TCP header—like the IP header—is of variable size and the length is contained in the fixed portion of the header.

Part 1: Packet Dumping (`-p`)

The first part of this project involves printing out various information about each packet in the trace (eliding those packets described above as always being ignored). The format of the output depends on whether the packet is a UDP packet or a TCP packet. The output will follow one of these two forms:

```
ts src_ip src_port dst_ip dst_port U app_data_vol
ts src_ip src_port dst_ip dst_port T app_data_vol seqno ackno
```

where:

- `ts` is the timestamp from the trace (in the meta information), printed in decimal to microsecond granularity (six digits after the decimal point).
- `src_ip` is the source IP address in the packet, printed in dotted-decimal notation without padding.
- `src_port` is the source port from the transport layer header, printed in decimal without padding.
- `dst_ip` is the destination IP address in the packet, printed in dotted-decimal notation without padding.
- `dst_port` is the destination port from the transport layer header, printed in decimal without padding.
- The fifth field is either "U" for UDP packets or "T" for TCP packets.
- `app_data_vol` is the number of bytes carried in the transport protocol's *payload*. This value does not include the size of the Ethernet, IP or transport layer headers.
- `seqno` is TCP's sequence number field, printed in decimal with no padding.
- `ackno` is TCP's acknowledgment number field, printed in decimal with no padding.

The fields are to be separated by a single space. Nothing should appear before the timestamp on each line. A newline must follow the last field with no intervening spaces.

The order of the packets in your output must be the order in which the packets appear in the trace file.

Sample output:

```
1105147636.767583 10.1.124.167 5396 10.1.86.253 80 T 327 53463213 3223122
1105147636.780222 10.1.86.253 8912 10.1.124.167 53 U 27
1105147636.934433 10.1.86.253 80 10.1.124.167 5396 T 1322 3223122 53463314
```

Sample invocation:

```
./proj3 -p -r tracefile.dmp
```

Part 2: Connection Summaries (`-s`)

The second part of the project involves aggregating individual packets between two transport endpoints together into "connections" and printing a summary. An "endpoint" is defined as an IP address and transport protocol port number. We will refer to the endpoints as the "originator" and the "responder". The "originator" is the source (IP and port) of the first packet observed between the endpoints. The "responder" is the destination (IP and port) of the first packet observed between the endpoints. Your program will print summaries in the following form:

```
start_ts dur orig_ip orig_port resp_ip resp_port proto \
o_to_r_pkts o_to_r_app_bytes r_to_o_pkts r_to_o_app_bytes
```

where:

- `start_ts` is the timestamp from the meta information of the first packet observed in the connection. This will be printed in decimal to microsecond granularity—six digits after the decimal point—and without any padding.
- `dur` is the duration of the communication between he endpoints. I.e., the span from the first packet observed to the last packet observed in the connection. Again, print to microsecond granularity without padding.
- `orig_ip` is the originator's IP address, printed in dotted-decimal without padding.
- `orig_port` is the originator's transport port number, printed in decimal without padding.
- `resp_ip` is the responder's IP address, printed in dotted-decimal without padding.
- `resp_port` is the responder's transport port number, printed in decimal without padding.
- `proto` is "U" for UDP connections and "T" for TCP connections.
- `o_to_r_pkts` is the number of packets sent from the originator to the responder. This should be printed as a decimal number without padding.
- `o_to_r_app_bytes` is the number of bytes of transport layer payload—i.e., application-level bytes—sent from the originator to the responder. This should be printed as a decimal number without padding.
- `r_to_o_pkts` is the number of packets sent from the responder to the originator. This should be printed as a decimal number without padding.
- `r_to_o_app_bytes` is the number of bytes of transport layer payload—i.e., application-level bytes—sent from the responder to the originator. This should be printed as a decimal number without padding.

It is possible that a connection may not have any traffic sent by the responder to the originator. In this case, the last two fields of the summary should each contain a single question mark ("?").

The fields are to appear on a single line and be separated by a single space. Nothing should appear before the timestamp on each line. A newline must follow the last field with no intervening spaces. Note: The format above is give on two lines with a continuation symbol (backslash) at the end of the first line. This is for presentation only. Your output must not include a backslash and must appear as a single line of output.

The order of the connection summaries in your output can be arbitrary.

Sample output:

```
1105147636.767583 0.166850 10.1.124.167 5396 10.1.86.253 80 T 1 327 1 1322
1105147636.780222 0.000000 10.1.86.253 8912 10.1.124.167 53 U 1 27 ? ?
```

Sample invocation:

```
./proj3 -r tracefile.dmp -s
```

Part 3: Round-Trip Times (`-t`)

The last part of the project builds on part 2, but instead of reporting the amount of traffic you will report the round-trip time (RTT) between two endpoints. We will only consider TCP connections in this part of the project. You will report two RTTs: one from the originator to the responder and one from the responder to the originator. We will define the RTT as beginning when a TCP endpoint sends the first data-carrying packet to its peer. That packet will carry a TCP sequence number X. The end of the RTT is when the other TCP endpoint sends an acknowledgment number that is $> X$. Your program will report the RTTs in the following form:

```
orig_ip orig_port resp_ip resp_port o_to_r_rtt r_to_o_rtt
```

where:

- `orig_ip` is the originator's IP address, printed in dotted-decimal without padding.
- `orig_port` is the originator's transport port number, printed in decimal without padding.
- `resp_ip` is the responder's IP address, printed in dotted-decimal without padding.
- `resp_port` is the responder's transport port number, printed in decimal without padding.
- `o_to_r_rtt` is the RTT as calculated from the time the originator sends its first data-carrying packet to the time when the responder acknowledges the data. The RTT must be reported as a decimal number of seconds to microsecond granularity without padding.
- `r_to_o_rtt` is the RTT as calculated from the time the responder sends its first data-carrying packet to the time when the originator acknowledges the data. The RTT must be reported as a decimal number of seconds to microsecond granularity without padding.

It is possible that a connection may not have data-carrying packets in one or both directions. In this case, the corresponding RTT should contain a single dash ("-").

It is possible that an endpoint sends a data-carrying packet, but this packet is never acknowledged and therefore a RTT cannot be computed. In this case, the corresponding RTT should contain a single question mark ("?").

The fields are to appear on a single line and be separated by a single space. Nothing should appear before the originator's IP address on each line. A newline must follow the last field with no intervening spaces.

The order of the connection summaries in your output can be arbitrary.

Sample output:

```
10.1.124.167 5396 10.1.86.253 80 0.166850 -
```

Sample invocation:

```
./proj3 -r tracefile.dmp -t
```

Notes:

1. Your submission must be a gzip-ed tar file called "CaseID-proj3.tar.gz" that contains all code and a Makefile that by default produces an executable called "`proj3`". 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 March 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.