EECS 325/425: Computer Networks
Project #4
Due: November 13, 11:59 PM

Often when researching or debugging networks it is useful to collect a *packet trace*. This involves monitoring a particular network link and recording each packet that crosses the link as it appears on the network. The fourth project of the semester involves processing a packet trace file. The aim of this project is to think about packets the way devices—e.g., computers, routers, switches, etc.—process packets. Your program will run in one of four modes, as directed on the command line. Each mode will read in one packet at a time from the trace file, analyze the packet and track or print some aspect of the packet. The command line syntax is as follows:

```
./proj4 -r trace_file -s|-l|-p|-c
```

Specifically:

- The "-r" option must always be present and the argument to this option specifies the name of the packet trace file.
- The "-s" option specifies the tool should run in "summary mode".
- The "-l" option specifies the tool will run in "length analysis" mode.
- The "-p" option specifies the tool will run in "packet printing" mode.
- The "-c" option specifies the tool will run in "packet counting" mode.
- A single mode must be specified. If your program is invoked with multiple mode selections an error must be issued and the program must terminate. Specifying no mode will also result in an error.
- Command line arguments can appear in any order.
- Unknown command line arguments must trigger errors.

Summary Mode

When your program is invoked with the "-s" option, it will operate in "summary mode". This mode provides a high-level summary of the trace file. After processing each packet in the trace, your program will print several pieces of information to standard output in this format:

```
time: first: [first_time] last: [last_time] duration: [trace_duration]
pkts: total: [total_pkts] ip: [IP_pkts]
```

The value of "first_time" is the timestamp (found in the trace) of the first packet in the trace file. Similarly, the value of "last_time" is the timestamp of the last packet in the trace. Both of these will be printed in seconds-since-epoch format and include 6 decimal places of precision—e.g., "1103112609.135350". The value of "trace_duration" is the duration of the trace file—i.e., last_time minus first_time. The duration should be printed in seconds and to 6 decimal places of precision. The value of "total_pkts" is the number of packets found in the packet trace file, reported as a decimal number with no padding. Finally, the value of "ip_pkts" is the number of IP packets found in the packet trace file, reported as a decimal number with no padding. Note: ip_pkts $\leq$ total_pkts. Also, note that the braces in the above format are to denote variables and should not appear in your output.

The labels will be presented in lower-case, as shown. A single space will separate all fields / words on each line. Further, no additional white space at the beginning or end of the lines may be printed.

Hint: We determine whether a packet is an IP packet or not by consulting the "type" field in the Ethernet header. See packet trace format document for more information.

Sample output:

```
./proj4 -r trace1.dmp -s
time: first: 1103112609.132870 last: 1103112609.135354 duration: 0.002484
pkts: total: 4 ip: 3
```

Length Mode

When your program is invoked with the "-l" option, it will operate in "length mode". In this mode, you will print length information about each IP packet in the packet trace file. That is, if the Ethernet type field indicates a packet is not an IP packet, you must ignore it. Likewise, if the Ethernet header is not present in the packet trace you will ignore the packet. Each packet will yield a single line of output in this format:

```
ts caplen ip_len iphl transport trans_hl payload_len
```

The fields are defined as follows:

- `ts`: This field is the timestamp of the packet, which is included with the packet's meta information (see packet trace format document). Print this as a decimal number to 6 decimal places of precision.
- `caplen`: This is the number of bytes from the original packet that have been "captured" in the packet trace. This value is included with the packet's meta information. This value must be printed as an unpadded decimal number.
- `ip_len`: This is the total length (in bytes) of the IP packet (from the total length field in the IP header). This should be printed as an unpadded decimal number.

  If the IP header is not included in the packet trace file, you must print a singe dash ("-") for this field.
- `iphl`: This is the total length (in bytes) of the IP header—which can be determined from the IP header.

  As with the ip_len field, if the IP header is not present in the packet trace, this field will be printed as a "-".
- `transport`: This indicates the transport protocol in use for this packet. A field in the IP header will indicate which transport is in use. This field should be printed as a "U" for UDP packets and a "T" for TCP packets. For all other protocols, this value will be a question mark ("?").

  This value will be a "-" if the IP header is not included in the packet trace.
- `trans_hl`: This is the total number of bytes occupied by the TCP or UDP header written as an unpadded decimal value.

  For other transport protocols your program must print a single question mark ("?") for this field.

  When the IP header is not included in the trace the size of the transport's header cannot be determined and therefore this field will be a single dash ("-").

  When the TCP or UDP header is not included in the packet trace the entry in this field will be a single dash ("-").
- `payload_len`: The final value to be printed for each packet is the number of application layer payload bytes present in the original packet. This will be determined by starting with the ip_len value and subtracting all IP and transport layer header bytes.

  Again, if the IP header is not present, this value cannot be determined and a "-" will be printed.

  Likewise, when the packet is a protocol other than TCP or UDP, report "?" for this field.

  Finally, when the TCP or UDP header is not available in the trace file, print a "-" in this field.

Each IP packet must produce a single line of output. The fields must be separated by a single space and no additional whitespace may appear at the beginning or end of the line.

Sample output:

```
./proj4 -l -r trace2.dmp
1103112609.132870 54 240 20 T 20 200
1103112610.465345 34 1000 20 T - -
1103112615.436115 43 308 24 U 8 276
1103112618.029221 20 - - - - -
```

TCP Packet Printing

When your program is run with the "-p" option it will operate in "packet printing mode". In this mode, you will output a single line of information about each TCP packet in the packet trace file. Non-TCP packets will be ignored. Further, TCP packets that do not have the TCP header in the packet trace file will be ignored. The output format for this mode is:

```
ts src_ip src_port dst_ip dst_port ip_id ip_ttl window ackno
```

The fields are defined as follows:

- `ts`: This field is the timestamp of the packet, which is included with the packet's meta information (see packet trace format document). Print this as a decimal number to 6 decimal places of precision.
- `src_ip`: This is the dotted-quad version of the source IP address. E.g., "192.168.1.43". Print the numbers in decimal and do not pad.
- `src_port`: This is the unpadded decimal version of TCP's source port number.
- `dst_ip`: This is the dotted-quad version of the destination IP address. E.g., "192.168.1.43". Print the numbers in decimal and do not pad.
- `dst_port`: This is the unpadded decimal version of TCP's destination port number.
- `ip_id`: This is the unpadded decimal value in IP's ID field.
- `ip_ttl`: This is the unpadded decimal value in IP's TTL field.
- `window`: This is the unpadded decimal value in TCP's advertised window field. (Disregard any window scaling that may be happening.)
- `ackno`: This is the unpadded decimal value in TCP's acknowledgment number field in ACK packets. Most TCP packets are valid ACKs, but not all. You must consult the flags field in the TCP header and ensure the ACK bit is set to "1" before reporting the acknowledgment number. For TCP packets without the ACK flag set, your program will include a single dash ("-") instead of the acknowledgment number.

Each TCP packet must produce a single line of output. The fields must be separated by a single space and no additional whitespace may appear at the beginning or end of the line.

Sample output:

```
./proj4 -p -r some-example.dmp
1103112609.132870 192.168.1.2 4512 192.168.100.34 80 4912 127 16384 3828024032
1103112610.983425 192.168.100.34 80 192.168.1.2 4512 11783 63 32961 3770985881
```

Packet Counting Mode

When your program is run with the "-c" option it will operate in "packet counting mode". In this mode, you will keep track of the number of packets and total amount of application layer data carried from each host to each of its peers using TCP. Non-rCP packets will be ignored by this mode. Likewise, any packets that do not contain TCP headers in the trace file will be ignored. After processing each packet in the trace file, this information will be printed, in this format:

```
src_ip dst_ip total_pkts traffic_volume
```

The fields are defined as follows:

- `src_ip`: The IP address that sends the packets. This will be printed in dotted-quad notation, as it was for the "-p" option.
- `dst_ip`: The IP address that receives the packets. This will be printed in dotted-quad notation, as it was for the "-p" option.

- `total_pkts`: This is the decimal representation of the total number of TCP packets sent from `src_ip` to `dst_ip` across all packets the entire trace file.
- `traffic_volume`: This is the decimal representation of the total number of application layer bytes sent from `src_ip` to `dst_ip` over TCP across all packets in the trace file. Note: The number of payload bytes for each packet is derived for the "-l" option, as well.

Each (src,dst) pair must produce a single line of output. The fields must be separated by a single space and no additional whitespace may appear at the beginning or end of the line. The order of the lines is immaterial.

Note: Traffic *from* Host-A *to* Host-B must not be combined with traffic *from* Host-B *to* Host-A. These will be reported on two different lines of output.

Hint: Choose an appropriate data structure to track the traffic volume.

Sample output:

```
./proj4 -r trace4.dmp -m
192.168.100.34 192.168.1.2 1 1029
192.168.1.2 192.168.100.34 2005 1026522
```

Final Bits

1. The usual generic programming guidelines apply. A copy will be posted to the project 4 web page.
2. Submission specifications:
   (a) All project files must be submitted to Canvas in a gzip-ed tar file called "[CaseID]-proj4.tar.gz".
   (b) Your submission must contain all code and a Makefile that by default produces an executable called "`proj4`" (i.e., when typing "make").
   (c) Do not include executables or object files in the tarball.
   (d) Do not include sample input or output files in your tarball.
   (e) Do not include multiple versions of your program in your submission.
   (f) Do not include directories in your tarball.
   (g) Do not use spaces in file names.
   (h) 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.
3. You may not leverage a third-party libraries in your project. Projects that rely on extra libraries will be returned ungraded.
4. Your submission may include a "notes.txt" file for any information you wish to convey during the grading process. We will review the contents of this file, but not of arbitrary files in your tarball (e.g., "readme.txt").
5. There will be samples on the class web page by the end of the day on October 18.
6. 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.
7. Do not make assumptions about the size of the packet traces.
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. Hints / tips:
   (a) Needlessly reserving large amounts of memory in case it may be needed (e.g., for tracking traffic in packet counting mode) is unreasonable.
   (b) Errors will be thrown at your project.
10. *WHEN STUCK, ASK QUESTIONS!*

EECS 425: Computer Networks
Extension of Project #4

Tell me something interesting!

A packet trace will be designated for this part of the assignment on the project web page. You will analyze this packet trace with your program (any or all of the modes) and report some interesting global observation. In other words, this is not some facet of a single packet or even a small number of packets, but something that spans the packet trace.

You will include a "report.pdf" file in your tarball that includes (*i*) a single plot *or* a single table, *and* (*ii*) a paragraph describing the interesting facet you are reporting.

The facet of the trace file you are reporting must have its roots in the output of your program. However, you may further analyze the packet stream or traffic matrix using one or more additional programs. These do not have to be written in C or C++ (e.g., you could analyze the output of the "./proj4 -c" with a Python script). The additional tools must be included in your tarball and referenced in your report.

Note: Undergraduates may do this portion of the assignment for extra credit.

Note: No extra credit will be given for projects submitted late.