CSDS 325/425: Computer Networks
Project #2
Due: October 7, 11:59 PM

The second project of the semester is to write a simple router simulator. The simulator will take both a router forwarding table and a packet trace file as input. For each packet in the packet trace your program will decide what action the router will take based on inspecting the packet and consulting the given forwarding table. The goal for this project is to start thinking about packets the way routers think about packets as they traverse the network.

Overview

Your program will take a mode selector and one or two filenames as input. The usage of your program—which will be called `proj2`—is as follows:

```
./proj2 [-p] [-r] [-s] [-f forward_file] [-t trace_file]
```

Specifically:

- When "-p" is given the program operates in *packet printing mode* whereby contents of the packet trace file will be printed. When "-p" is given, "-t" must also be given. See below for details.
- When "-r" is given the program operates in *forwarding table printing mode* whereby the contents of the forwarding table will be printed. When "-r" is given, "-f" must also be given. See details below.
- When "-s" is given the program operates in *simulation mode*. When "-s" is given, both "-f" and "-t" must also be given. See details below.
- The "-f" option followed by a filename provides the name of the file containing the router's forwarding table. This option must be given when the "-r" or "-s" options are used. See below for details.
- The "-t" option followed by a filename provides the name of a packet trace. This option must be given when the "-p" or "-s" options are used. See below for details.
- The user must give one of the mode options: "-p", "-r" or "-s". Multiple mode options are not allowed.
- The command line options may appear in any order.
- Unknown command line arguments must trigger meaningful error messages.
- When encountering an error your program will print a meaningful error message and terminate.

Packet Trace File Format

The packet trace file contains a series of IPv4 packet headers captured on an operational network. Operators and researchers use packet traces to better understand networks. Packet traces are binary files. You will not be able to readily inspect their contents without a tool like "od". The packet trace you will use for this project contains a timestamp and a basic IPv4 packet header for each packet. The remainder of the packet has been stripped away (e.g., the link layer header/trailer, transport header, application data, etc.). Therefore, each packet is represented by 28 bytes in the packet trace file. In particular:

- **Bytes 1–4:** The number of seconds since Unix epoch (midnight GMT on January 1 1970). This value must be combined with the fractional value from the next field to form the timestamp. This value is an unsigned 4-byte number encoded in *network byte order*. See *ntohl()*.
- **Bytes 5–8:** The fractional part of the timestamp. This is a number of microseconds (so ranging from 0–999,999) that must be combined with the previous field to form the timestamp. This value is an unsigned 4-byte number encoded in *network byte order*. See *ntohl()*.

- **Bytes 9–28**: The fixed 20 byte IPv4 header from the packet. While IPv4 headers can include options and therefore be longer than 20 bytes, a simplifying assumption for this project is that none of the packets will have options. Additionally, you may assume none of the packet trace files will have partial IPv4 headers. Therefore, each packet header will be exactly 20 bytes long.

  Note: The 16 and 32 bit fields in the IP header are unsigned values encoded in network byte order and you'll need to use *ntohs()* and *ntohl()* to find the proper values.

- After reaching byte 28 in the file, the format repeats for the next packet (unless you have reached the end of the file). That is, bytes 29–36 will be the timestamp for packet #2 and bytes 37–56 will be the IPv4 header for packet #2.

Additional Hints:

- You can find a structure to hold both components of the timestamp in /usr/include/linux/time.h as `struct timeval`.
- You can find a structure for the IP header in /usr/include/netinet/ip.h as `struct iphdr`.

Forwarding Table File Format

The forwarding table file is a binary file that includes a series of eight byte records that represent a router's forwarding table, as follows:

- **Bytes 1–4:** This contains an IPv4 address in network byte order. You will need to use *ntohl()* to turn this into host byte order.
- **Bytes 5–6:** This is an unsigned integer that represents how much of the IP address to use when matching an incoming packet to the router's forwarding table. This value is encoded in network byte order. This value will always be eight in the CSDS 325 input files.[1]
- **Bytes 7–8:** This is an unsigned integer that gives the output interface number on which matching traffic will be forwarded. This value is encoded in network byte order.
- After reaching byte 8 in the file, the format repeats for the next rule (unless you have reached the end of the file). That is, bytes 9–16 will be the second rule in the forwarding table, etc.

You may assume none of the input files will have partial records. That is, all records will be exactly eight bytes long.

Additional Hint:

- It may be useful to define a "`struct`" with one unsigned integer (for the IP address) and two unsigned short integers (for the prefix and interface number).

Packet Printing Mode

When the "-p" option is given your program will operate in packet printing mode. In this mode, you will print information about each packet in the packet trace file specified. Print the packets in the order they appear in the file. Each packet will produce a single line of output, as follows:

```
timestamp src_addr dst_addr checksum ttl
```

The fields in each line will be separated by a single space. Each line will end with a newline character ("`\n`"). Do not include any additional information or whitespace. The fields will be printed as follows:

[1] The prefix in CSDS 425 input files will be different. See below. CSDS 325 students only have to deal with a prefix of length eight bytes.

- **timestamp:** This will be printed as a floating-point decimal number in seconds-since-epoch format to 6 decimal places of precision—e.g., "1103112609.135350". I.e., this number is the combination of the two timestamp fields described in the packet trace format above.
- **src_addr:** This is the IPv4 source address printed in dotted-quad notation. That is, four unpadded decimal integers separated by periods (".") . E.g., "192.168.10.54".
- **dst_addr:** This is the IPv4 destination address printed in dotted-quad notation. That is, four unpadded decimal integers separated by periods (".") . E.g., "132.235.1.2".
- **checksum:** This is an indication of whether the checksum passes or fails. As a simplifying assumption, if the checksum value is 1234 it will be considered to be correct and you will print a single "P" character (pass). Otherwise, the checksum will be considered to be incorrect and you will print a single "F" character (failed).
- **ttl:** This is the TTL field from the IPv4 packet header printed as an unpadded decimal number. E.g., "132".

Example output:

```
1103112609.135350 192.168.10.54 132.235.1.2 P 132
1103112609.274923 192.168.1.12 13.81.40.201 F 63
1103112609.343543 10.0.14.38 128.3.43.3 P 68
```

Forwarding Table Printing Mode

When the "-r" option is given, your program will operate in forwarding table printing mode. In this mode you will print information about each rule in the forwarding table file. Print the rules in the order they appear in the file. Each rule will produce a single line of output, as follows:

```
ip prefix_len interface
```

The fields on each line will be separated by a single space. Each line will end with a newline character ("\n"). Do not include any additional information or whitespace. The fields will be printed as follows:

- **ip**: This is the IPv4 address from the forwarding table printed in dotted-quad notation. That is, four unpadded decimal integers separated by periods (".") . E.g., "192.168.10.54".
- **prefix_len**: The number of bits the router will use when comparing the IP address prefix in the rule with the destination IP address in incoming packets.
- **interface**: The interface number on which to forward traffic matching the rule's IP address and prefix.

Examples:

```
10.0.0.0 8 1
132.0.0.0 8 2
0.0.0.0 8 3
128.1.2.3 8 4
```

Simulation Mode

When the program is run with "-s" it will operate in simulation mode. In this mode the router forwarding table given on the command line (with "-f") will be read and stored. The program will then read each packet in the given packet trace file (as specified with "-t") and decide what to do with the packet by inspecting the packet and consulting the forwarding table. You will need to store each rule in the forwarding table such that you can consult the rules for each packet. The following two stipulations apply:

- Duplicate IP address prefixes are not allowed. If such are given in the input file your program must print a meaningful error message and exit. Note: prefixes of length 8 bits with the following IPs are considered the same: 10.0.0.0, 10.1.1.1, 10.0.0.2. That is, because he rule only covers the first 8 bytes (or 1 byte) and all these start with "10", they are considered duplicates.
- An IP address of "0.0.0.0" indicates the "default" interface to forward traffic on when no more specific rule applies. This may or may not need to be special-cased when storing the table.

The output of the program will consist of a timestamp and an action, as follows:

```
timestamp action
```

The packet's *timestamp* will be printed exactly as it is in packet printing mode. That is, in seconds-since-epoch format with 6 decimal places of precision—e.g., "1103112609.135350". Again, this number is the combination of the two timestamp fields described in the packet trace format above.

The *action* will be one of the following:

- **drop checksum**: When the packet's checksum calculation *fails* the router cannot further process the packet and will drop it. Recall from above that when the checksum is 1234 it is considered to be correct and otherwise it is considered to be incorrect.
- **drop expired**: When the packet's TTL field is "1" the router will decrement this to zero and the packet will have *expired* and will be dropped.
- **drop policy**: When the interface indicated in the forwarding table for the destination IP address in the packet is zero ("null") the packet will be dropped as a matter of *policy*. That is, the router has been configured to not pass traffic to the address (or "null routed" the address).
- **send** [$interface$]: When the destination address in the packet matches an entry in the forwarding table you will list the interface on which the packet will be transmitted by the simulated router. E.g., "send 3".
- **default** [$interface$]: As noted above, a rule in the forwarding table for "0.0.0.0" indicates the "default" rule. That is, if the destination IP address in the packet does not match any specific rule in the given forwarding table, but the forwarding table has an entry for IP address "0.0.0.0" then all packets without specific rules will be transmitted on the given interface. E.g., "default 10".
- **drop unknown**: The router does not know what to do with a packet that doesn't fall into any of the above categories. Therefore, this packet will be dropped as "unknown".

Note: It is possible a packet falls into multiple categories given above. The categories must be applied in the order given above and the first one that applies to a packet will be applied. For instance, if the checksum is bad and the TTL is "1" then the drop should be reported as caused by the bad checksum.

Example output:[2]

```
1103113719.100000 drop policy
1103113719.200000 drop checksum
1103113719.300000 send 13
1103113719.400000 send 2
1103113719.500000 default 1
1103113719.600000 drop expired
1103113719.700000 send 13
1103113719.800000 drop unknown
1103113719.900000 send 13
```

[2]Note: This specific output is impossible. It is meant to show examples of all possible output lines. But, no output will have both "default" and "unknown" lines.

Final Bits

1. Submission specifications:
   (a) All project files must be submitted to Canvas in a gzip-ed tar file called "[CaseID]-proj2.tar.gz" (without the brackets and with your Case Network ID).
   (b) Your submission must contain all code and a Makefile that by default produces an executable called "`proj2`" (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 folders / 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.
2. You may *not* leverage third-party libraries for this project. The standard C library and the C++ STL are fine. Projects that rely on extra libraries or tools will be returned ungraded.
3. Your submission may include a "notes.txt" file for any information you wish to convey to help us during the grading process. We will review the contents of this file, but not of arbitrary files in your tarball (e.g., "readme.txt").
4. We do *not* review Canvas comments during grading. We only look at the contents of the tarball you submit.
5. There will be sample reference output on the class web page by the end of the day on September 16. (Not all sample input will have (all) corresponding reference output.)
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 input.
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 is unreasonable. Reserving memory in case it may be needed is unreasonable.
   (b) Errors will be thrown at your project.
   (c) A simple C program and Makefile are available on the class web page. The program illustrates the use of *getopt()* to parse the command line arguments. The Makefile can be easily adapted for this project.
10. *WHEN STUCK, ASK QUESTIONS!*

CSDS 425: Computer Networks
Project #2
Due: October 7, 11:59 PM

In addition to the CSDS 325 portion of the project, CSDS 425 students will be responsible for a short extension, as follow.

- Instead of the prefix length in the router forwarding tables always being "8", the prefix length can be "8", "16", "24" or "32".
- The prefix length controls the number of bits used to compare the destination IP address and addresses in the router forwarding table when deciding which rule determines the forwarding of each packet in the trace file.
- Recall from lecture that we use "longest prefix match" in the Internet. For instance, consider two rules in the router forwarding table: (*i*) route all 10.0.0.0/8 traffic to interface 100 and (*ii*) route all 10.1.0.0/16 traffic to interface 200. Therefore, a packet destined for 10.10.10.10 would be sent on interface 100 because the address only matches the first rule. Meanwhile, a packet destined for 10.1.2.3 would be sent on interface 200 even though it matches both rules because the second rule represents a longer prefix than the first rule.

Note: When dealing only with prefixes of size 8, the forwarding table cannot be all that big since there are only 256 possible prefixes. However, when allowing more prefixes the tables can grow quite large. For instance, there are 4 billion 32 bit prefixes in the IPv4 address space. The larger the table the more important the choice of data structure to hold and lookup the information. Your program will be stress tested with large forwarding tables.

CSDS 325 students may do this portion of the project for extra credit. However, extra credit is not available for projects turned in late.