Networking Lab 3 – Analysis of TCP

The goal of this laboratory is to understand transport layer protocols, with particular attention to TCP. TCP offers a reliable service, based on a connection oriented paradigm based on a bidirectional connection. It performs flow control, congestion control, error correction, etc.

1. Running some service on the host

In order to use transport protocol, we need some application running at the application layer at the server, which is "listening" for possible requests from clients. To enable some basic services, the O.S. must be correctly configured. To do so, the file `/etc/inetd.conf` must be properly edited. First, check which services are actually already running on your host using the netstat command

1. Check using netstat which are the current state of both TCP and UDP. Check the man page to figure out which parameter to use.
2. Is there any service running in your PC that can be reached using TCP? Or using UDP?

```
netstat - Print network connections, routing tables, interface statistics,
masquerade connections, and multicast memberships
```

```
netstat [address_family_options] [--tcp|-t] [--udp|-u] [--raw|-w]
[--listening|-l] [--all|-a] [--numeric|-n] [--numeric-hosts]
[--numeric-ports] [--numeric-users] [--symbolic|-N]
[--extend|-e[--extend|-e]] [--timers|-o] [--program|-p] [--verbose|-v]
[--continuous|-c]
```

3. Using `gedit`, edit the `/etc/inetd.conf` file to enable the `echo` and `chargen` services. Do you need to be root to edit this configuration file? Do it on all hosts of your network.
4. (Re)start the `inetd` service by running the command
   `/etc/init.d/openbsd-inetd start`
5. Check again the status of the services that are now running on your host using the `netstat` command. Which ports are used by the `echo` and `chargen` service?

WARNING: To avoid some misunderstanding, we need to disable some advanced functionalities that modern linecards implement. This can be done using the ethtool. Check the man page to see which are those advanced features.

1. Checking the OFFLOADING capability of a linecard

```
ethtool -k ethX
```

2. Disable any OFFLOADING capability implemented by any linecard

```
ethtool -K ethX [rx on|off] [tx on|off] [sg on|off] [tso on|off] [ufo
on|off] [gso on|off] [gro on|off] [lro on|off]
```

2. Using services from a client

Now that each host is running different services, you can use a client application to connect to one of the applications running on another host. We can use the `telnet` command to contact the application we are interested into on any host of your LAN. `telnet` is a remote terminal emulator that, using TCP at the

transport layer, allows one accessing a service on a remote host. The `telnet` command is used for interactive communication with another host (the server) using the TELNET protocol. It begins in command mode, where it prints the `telnet` prompt ("`telnet>` "). If `telnet` is invoked with a host argument, it performs an open command implicitly; to terminate using a service, enter into the "**telnet command mode**" by pressing the escape sequence `'^]'+RETURN` (that is, press `CTRL` and `]` keys at the same time on the keyboard and then `RETURN` key). This will force the telnet application running locally to not send character to the server, but to interpret them as command. Use the `quit` command to exit from the telnet application.

`telnet <host> [port]`

will try to setup an application layer communication to `host` on port `port` using TCP at the transport layer.

1. Which well-known port do you know?
2. Which should be the command to be used from `H1` to contact the echo server on `H3`?
3. Check the file `/etc/services`. Which information is present? Which functionality is offered by that file according to the ISO-OSI standard?

3. Analysis of the ECHO service

1. Run wireshark and start capturing packets
2. Using telnet, open a connection from your host to another host in the LAN
    a. Contacting the `time` service
    b. Contacting the `echo` service. Type some strings on your keyboard, then close the connection by entering into command mode, and using the quit command
3. Analyze the packet level trace, identifying
    a. the connection setup phase
    b. the data exchange phase
    c. the connection tear down phase.
4. How many connection request do you see?
    a. When contacting the `echo` service, is the three-way-handshake successful? Why?
    b. When contacting the port 1234, is the three-way-handshake successful? Why?
5. How are sequence numbers chosen during the three-way-handshake? **WARNING**: wireshark uses relative sequence numbering. Which is the actual sequence number?
    a. Why the client and the server perform a `SYN`chronization of sequence number? Why it is deprecated to always start numbering from 0?
    b. How are TCP source port numbers chosen? Hint: open two connections from the same server to the same destination host at the same port.
6. How is it possible to multiplex/demultiplex correctly different TCP connection?
7. How are ACK numbers sent by the receiver?
8. How can the server refuse an incoming connection?
9. When does the application send data to the server? How is the text encoded?
10. How do the sequence number and ACK number evolve over time?

Suggestion: get confident with wireshark functionalities to

- Follow a TCP stream
- Generate statistics about conversations
- Showing IO graphs

- Performing flow analysis

4. Analysis of the CHARGEN service

1. Run wireshark and start capturing packets
    a. **Suggestion**: since we are NOT interested in the application payload, you should limit the packet capture to include all TCP headers, but not all application payload]
2. Open a connection from your host to another host in the LAN contacting the `Chargen` service.
    a. What is the application running on the server doing?
    b. What happens to eventual characters you type on the keyboard?
3. After some time, **maximize** the terminal windows on your screen
4. Now **minimize** the terminal window for about 10s, then bring it back to full screen.
5. Now press CTRL+C on the telnet window [this stops telnet to print characters on the screen – but the chargen server is still sending lot and lot of packets, now at line rate speed]
6. Open a new terminal window, and open a second telnet to the same server
7. Press CTRL+C on the second telnet screen
8. After about 10s, enter into Telnet command mode by pressing the escape sequence `'^]'+ return CONTROL and ‘]’ character`
    a. What is happening now? How can the client stop the server from sending packets?
    b. Wait 10s then press “`return`” on the keyboard. What happens?
9. Enter again into telnet command mode, and close the connection.

Analyze the packet level trace

1. Identify the connection setup phase, the data exchange phase, and the connection tear down phase.
2. How are sequence number chosen during the three-way-handshake? WARNING: wireshark uses relative sequence numbering. Which is the actual sequence number?
3. How are ACK numbers sent by the receiver?
4. When the application sends data to the server? How is the payload encoded?
5. How do the sequence numbers and ACK numbers evolve over time?
6. When the terminal window is bigger/smaller, or when telnet stops printing character, how is the RWND evolving?

Plot the evolution versus time of

- The client sequence number
- The client acknowledge number
- The server sequence number
- The server acknowledgement number
- The size of the packets sent by the client
- The size of the packets sent by the sever
- The receiver window size advertised by the client
- The receiver windows size advertised by the server

Briefly comment the plots.

How to make the plots

To make the above plots, the easiest way is to save the packet trace collected by wireshark on a `.txt` file, and then use some simple Bash scripts to extract files that contain the desired fields. First, select the set of packets you want to consider by using a proper visualization filter in wireshark. Since client-to-server and server-to-client packets have to be *separately* analyzed, export the client and server trace into two separate files, respectively. Use the menu `File->export` entry to select the proper subset of packets that you need, and export only the "summary line".

Once saved, you can use Bash scripting tools to process the file and extract the values that you are interested into. As suggestion, consult the `man` page to see more details and useful options. Some useful commands are:

- **`cat[OPTION]... [FILE]...`**: concatenate files and print on the standard output.
  prints the [FILE] to the screen. Useful to feed the content of [FILE] to another command using a pipe.
- **`cut OPTION... [FILE]...`**: Print selected parts of lines from each FILE to standard output.
  `-d 'DELIM'`: use DELIM instead of TAB for field delimiter
  (e.g., -d ' ' uses space, or –d '=' uses the = char as delimiter)
  `-f` : select only these fields, e.g., -f 1,3-5 prints first, third, forth and fifth fields (notice the different meaning of , and -).
  `cut -d '=' -f 1 data.txt`: chop each line into fields using the '=' as delimiter, then print only the first field.
- **`paste [FILE…]`**: merge lines of files. Write lines consisting of the sequentially corresponding lines from each FILE, separated by TABs, to standard output.
  `paste file1.txt file2.txt` prints each lines of file1.txt and file2.txt on the same line.
- **`grep[OPTIONS] PATTERN [FILE...]`**: searches the FILE for lines containing a match to the given PATTERN. Prints lines of FILE matching the PATTERN
  `grep ACK data.dat`: prints only lines where ACK is present in data.dat file
  `-v`: Invert the sense of matching, to select non-matching lines.
- **`tr[OPTION]... SET1 [SET2]`**: Translate, squeeze, and/or delete characters from standard input, writing to standard output
  `tr 'c' 'C'`: replace every occurrence of `c` with a `C`
  `-s`: replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character
  `tr -s ' '`: replace multiple spaces with a single space

In Bash, it is possible to concatenate commands creating a pipeline, i.e., the ouput of the previous command is used as input of the following one. The pipeline is a sequence of one or more commands separated by the control operator `'|'`. The format for a pipeline is:

```
command | command2 …
```

The standard output of `command` is connected via a pipe to the standard input of `command2`. Similarly, it is possible to redirect the output of a command to a file using `'> OUTFILE'`. If `OUTFILE` already exists, it will be overwritten. To append the results to a file which is already existing simply use `'>> OUTFILE'`. If outfile does not exists, it will be created. If it exists, the output will be appended to it.

For example

```
grep ACK data.dat | tr –s ‘ ‘ | cut –d ‘ ‘ –f 2,4 > results.txt
```

would look for rows in the data.dat file that contains the `ACK` pattern. For each matching line, multiple occurrences of the space character will be replaced with a single space. Then, each line is split into fields by using the space as delimiter, and only the second field will be printed. The result is saved on the `results.txt` file.

Assuming the file data.dat looks like

```
1   pippo    pluto paperino,nonna papera
2     qui        quo,qua    ACK
3 zio paperone ACK pluto      topolino
```

The output of the above command will be save in the results.txt file as

```
qui ACK
zio ACK
```

In the following, you find a sample script that can be used as example and must be properly modified to extract useful values from txt files exported from wireshark.

```
# simple script that extracts useful information from a wireshark exported txt file
# WARNING : THIS HAS TO BE CUSTOMIZED AND VERIFIED!

# extract the time information
# I need to squeeze multiple spaces, then cut on space and get the time column
# I use grep to avoid the first line.
# from the grep man page:
# -v Invert the sense of matching, to select non-matching lines.
# i.e., do NOT print lines that match ...

cat client.data.txt | grep -v Source | tr -s ' ' | cut -d' ' -f 3 > client.time.txt

# extract the seqno
cat client.data.txt | grep -v Source | cut -d'=' -f 2 | cut -d' ' -f 1 > client.seqno.txt
# extract the ackno
cat client.data.txt | grep -v Source | cut -d'=' -f 3 | cut -d' ' -f 1 > client.ackno.txt
# extract the rwin
cat client.data.txt | grep -v Source | cut -d'=' -f 4 | cut -d' ' -f 1 > client.rwin.txt

# paste all columns together
paste client.time.txt client.seqno.txt client.ackno.txt client.rwin.txt > client.dat

#plot it using gnuplot
gnuplot plot.gnu
```

Extend the previous script to generate all data that are needed to produce the required plots, for client and server traffic. Suggestion: check the output of the pipeline piece by piece to verify that there are no errors.