Lab 2 Data processing notes

monitor_tcq.py

The python script, `monitor_tcq.py`, polls the length of the tc queue on the interface "r1-eth2". It does this by running the command '`tc -s qdisc show dev r1-eth2`' in a loop, parsing the results and writing the queue lengths to a file called '`qlen.txt`'.

The format of the qlen.txt file is:

elapsed_time (seconds)	epoch_time (seconds)	queue_length (packets)

The content of `qlen.txt` looks like this:

```
[...]
0.21247386932373047 1668595532.227865 0
0.2290029525756836 1668595532.244394 0
0.245499849319458 1668595532.260891 0
0.2619009017944336 1668595532.277292 0
0.27829480171203613 1668595532.293686 0
0.2947428226470947 1668595532.310134 0
0.3110928535461426 1668595532.326484 0
0.3279588222503662 1668595532.34335 0
0.34428977966308594 1668595532.359681 0
[...]
```

You can read more about epoch time here:
https://en.wikipedia.org/wiki/Unix_time

Epoch time (or Unix time) is the number of seconds since 00:00:00 01.01.1970. It is useful here because it is universal. All timestamps using epoch time are from the same clock. The first column, on the other hand, is the time elapsed since the script was started. If you run two different programs, and start them at different times, the elapsed timestamps will be offset from each other, and it makes it difficult to line up the results.

Processing ping data

When you run the command like 'ping -i 0.1 -c 100 -D rx1', you get results that look like this:

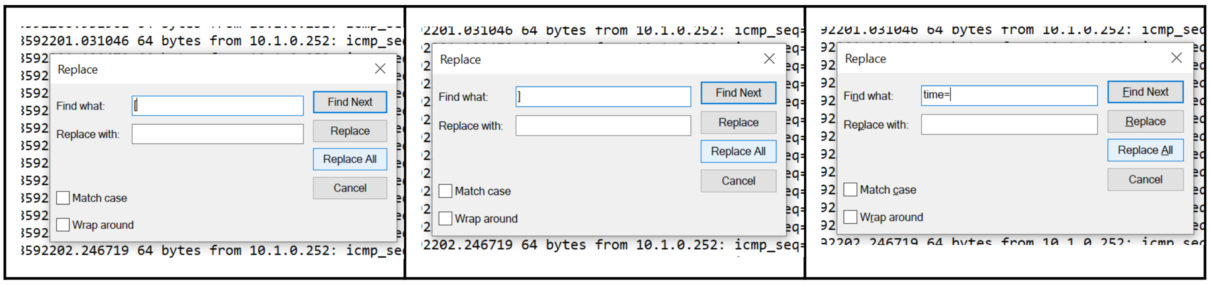
```
PING 10.1.0.252 (10.1.0.252) 56(84) bytes of data.
[1668592200.830226] 64 bytes from 10.1.0.252: icmp_seq=1 ttl=63 time=12.2 ms
[1668592200.931901] 64 bytes from 10.1.0.252: icmp_seq=2 ttl=63 time=12.3 ms
[1668592201.031046] 64 bytes from 10.1.0.252: icmp_seq=3 ttl=63 time=10.9 ms
[1668592201.132479] 64 bytes from 10.1.0.252: icmp_seq=4 ttl=63 time=10.8 ms
[1668592201.234131] 64 bytes from 10.1.0.252: icmp_seq=5 ttl=63 time=10.9 ms
[1668592201.336404] 64 bytes from 10.1.0.252: icmp_seq=6 ttl=63 time=11.7 ms
[1668592201.435863] 64 bytes from 10.1.0.252: icmp_seq=7 ttl=63 time=10.4 ms
[1668592201.536637] 64 bytes from 10.1.0.252: icmp_seq=8 ttl=63 time=10.7 ms
[1668592201.639290] 64 bytes from 10.1.0.252: icmp_seq=9 ttl=63 time=11.7 ms
```

The '`-D`' option told ping to include epoch timestamps at the start of each line. One of those is highlighted in the example above. The end of the line is the measured RTT of the ping (in milliseconds). That is the other number we want to extract, and is also highlighted in the example above.

There are many tools people use to plot data, and some make it easier to extract values automatically. For most, it is easiest if the numbers you want have whitespace (spaces or tabs) around them. Here are two easy ways to achieve that.

1. In a text editor, like Notepad

Use the find/replace function. The things we want to remove are the "`[`", and "`]`" around the timestamp and the "`time=`" before the RTT.
Open the ping log file in your editor. Go to (probably) Edit --> Find/Replace. Enter the thing you want to get rid of, and leave the replacement string blank. Three examples in Notepad are below:



```
3592201.031046 64 bytes from 10.1.0.252: icmp_se
3592202.246719 64 bytes from 10.1.0.252: icmp_se
```

Replace — Find what: [— Replace with:

Replace — Find what:] — Replace with:

Replace — Find what: time= — Replace with:

After you do this, most software will be able to recognize the columns you want. You may have to delete the lines at the start and end of the file, since they are different from the data lines.

2. Use a script or sed/awk/perl in the VM

There are many text processing tools in linux. Here is an example using a perl regular expression that will extract the values you want and print them to STDOUT:

```
perl -pe 's/\[(\d+\.\d+)\] .* time\=(\d+\.\d+) ms/$1\t$2/' file_to_process
```

What is this doing?
The perl command "`s/`*`regular_experssion`*`/`*`replacement_text`*`/`" is just a fancier find/replace. The "s" is the substitution operator. In the example below, the regular expression is highlighted in blue. The replacement text is highlighted in yellow.

```
perl -pe 's/\[(\d+\.\d+)\] .* time\=(\d+\.\d+) ms/$1\t$2/' file_to_process
```

If you are interested, you can read about regular expressions online. In perl, the parts of the regex contained in parentheses, for example "`(\d+\.\d+)`", will be saved to numbered variables, $1, $2, etc. That is what is printed out in the yellow part.

To save the processed results, you need to pipe the output of this command to another file. Here is an example, where we process a file, `pings.log`, and write the result to `pings.dat`.

```
perl -pe 's/\[(\d+\.\d+)\] .* time\=(\d+\.\d+) ms/$1\t$2/' pings.log > pings.dat
```