Wed Oct 21 2020

Nuru Nabiyev 2642819

Matthias Debernardini 2622050

Detjon Shahu 2611771

Milestone 5 for Userspace TCP Stack

To understand how benchmarking works, we were tasked with doing one so that we can see the difficulties that come from trying to clock a computer system. The first step involved choosing the computer to run the benchmark. We decided to use a PC with the following specifications; AMD Ryzen 2700x, 16 GB of 3200MHz CL14 DDR4 RAM, 120GB SATAIII SSD running Ubuntu 18.04.5 LTS. This choice was mainly due to convenience and because that's what I have in my room. The samples were taken from this machine consecutively.

The second step involved choosing how to query the system clock. For this, we picked the monotonic clock in the time library that ships with the Linux kernel. We chose to use a wall clock rather than a user clock because we are interested in how the application uses the userspace library rather than how the CPU executes the code. The next step was deciding where to start the time and when to end it. We were interested in measuring the latency that occurs in the three-way TCP handshake. To measure this, we placed the clocks in the client applications at the beginning and the end of the exchange. Nothing was changed neither in the application nor in the userspace library. The difference between the clocks was printed to standard output, which is then redirected to a file with a timestamp of the sample. The code can be found in the appendix.

The server and client then ran multiple times under a shell script, since the sleep time is the same for all samples no post adjustments were made to the final result. Once we collected the data, we had to decide which samples were representative of the actual latency between the server and the client. It is insufficient to only measure the time and then report one quantity. The environment that the code is running in is constantly changing, cache lines are invalidated at seemingly random intervals, the clock frequency is dynamically changing depending on the load (to increase energy efficiency and overall performance), page faults can occur (or not). This is too much information to try to distill into one number. Therefore, to estimate the run-time of the exchange, we must also report the median of a population sample, the standard deviation, and the uncertainty. Because of how wildly variable the latency can be, outliers will need to be methodically dealt with. Our method for dealing with outliers involved calculating the Median-Absolute-Deviation, using the MAD to decide if a data point is an outlier. This step included deciding how many deviations away from the median are tolerable. In our case it was 3, this was chosen manually by

increasing the number of deviations and seeing when the changes in the number of outliers were small relative to the number of samples. The uncertainty was then calculated by dividing the MAD by the square root of non-outlier values. To verify our results, we re-ran the benchmark and looked at the average, and compared it to our previous median plus or minus the uncertainty. The results from the second sample are within the margin of error which increases our confidence that we modeled the distribution of the process in such a way as to account for the random variables.

In conclusion, exchanging 4096 bytes using our userspace TCP library took 51.03 microseconds +/- 0.51 microseconds. This distribution was obtained with 100 iterations of our program and 36 samples were classified as outliers.

Statistic	Microseconds
Average Latency	51.30
Median Latency	52.57
95 Percentile	57.07
99 Percentile	60.70
Max Observed Latency	62.77
Min Observed Latency	45.53
Sample Size	76

Reproducing the results

- Install `just` to expedite the benchmark otherwise copy and paste the commands found in the `justfile` directly in the shell
- `apt install just` or `cargo install just`
- `just prep`
- `just iterate`
- `cd plotter`
- `python3 main.py`

http://blogs.perl.org/users/steffen_mueller/2010/09/your-benchmarks-suck.html

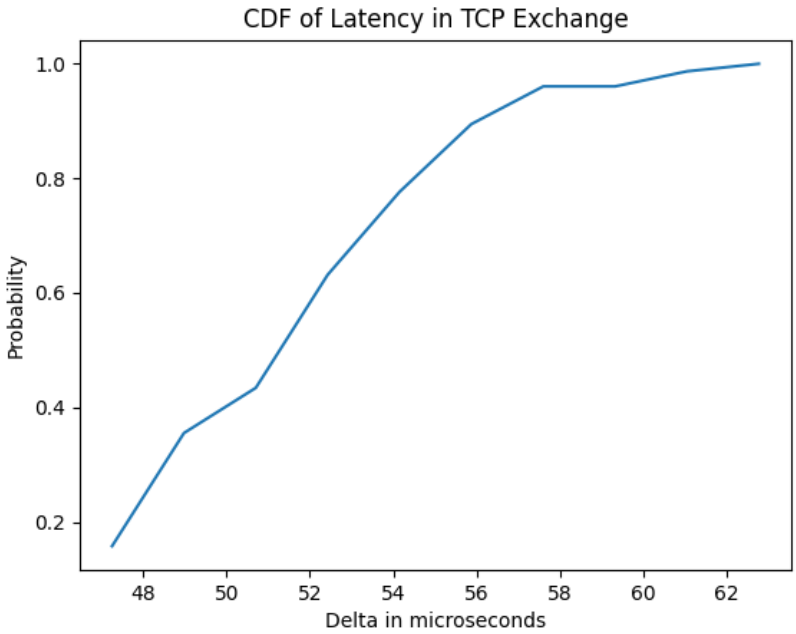
CDF of Latency in TCP Exchange
Probability
1.0
0.8
0.6
0.4
0.2
48
50
52
54
56
58
60
62
Delta in microseconds

Figure 1: CDF