Advanced Networks — Laboratory 5

Juliusz Chroboczek

26 March 2025

Exercice 1.

1. Write a Go program that downloads a file from the URL given as a command-line parameter and saves it to disk.
2. Modify your program so that it downloads the file by making sequential range requests for pieces of the size given by the `-c` command-line option (default 16 kB). Note that you will need to treat the first piece specially. Make sure that your program behaves correctly if the server doesn't obey range requests.
3. Modify your program so that it downloads the file in a single request unless the server provides a strong `ETag` on the first piece.
4. Modify your program so that it uses the `If-Match` header to avoid corrupting the data if the file changes on the server. Ensure that your program does the right thing even if the server doesn't honour `If-Match`.
5. How much slower is piece-wise downloading? Why does it depend on the location of the server?

Exercice 2. Modify your program so that it downloads multiple pieces in parallel. Your program will first download the first piece. If the server provides a strong ETag and obeys range requests, then it will download the file using n simultaneous threads (*goroutines*), where n is the value of the command-line parameter `-n` (default 4).

You may structure your program as follows:

- a main program, that fetches the first piece then produces a stream of piece descriptions to download which it sends over a channel;
- n threads, that compete to read a piece description from the channel.

The main program may signal that there are no pieces left to download by closing the channel. You may use the `(*File).WriteAt` method to avoid concurrency issues when writing data to disk. You may use a `sync.WaitGroup` in order to determine when the worker threads have terminated.

Verify that your program does use the expected number of connections, using either `tcpdump`, *Wireshark*, or simply `netstat`.