Homework 01

The homework requirements are explained in Requirements file

- Homework 01
  - Project Structure
  - Structures
  - Client
  - Server
  - Command line arguments
  - Results (round 1 => streaming UDP vs stop-and-wait TCP)
  - Results (round 2 => streaming and stop-and-wait for both UDP and TCP)
  - Conclusions (round 1)
  - TCP
  - UDP

The project is written in Python. It is a simple client-server application where the client uploads files to a server via TCP (stop-and-wait) or UDP (streaming) protocols.

Project Structure

File	Description
config.cfg	Simple config with settings for client, server & common (alternative to the command line arguments)
structures.py	Definition of packages sent via TCP or UDP
client.py	The client that uploads files once it connects to the server
server.py	The server that receives the files from the client and writes them to disk

Structures

Structure	Description
Protocol	TCP or UDP
TCPState	Corrupted or Good (package)
PackageType	Header (file size, number of packages, etc) or Block (actual indexed data)
HeaderTCP	Filename, file size and number of packages which will be sent

Structure	Description
HeaderUDP	Depending on the type and state => initial header (metadata), actual block or 'Done' (end) marker

Client

Flow	Description
TCP	send_data_via_tcp(source_path) => connection & source path folder iteration => for every file split it in packages and send them with confirmation
UDP	send_data_via_udp(source_path) => connection & source path folder iteration => for every file split it in packages and send them (no confirmation)

Server

Flow	Description
TCP	receive_data_via_tcp(destination_path) => connection => for each file received (identification via initial header) write all its packets (iterative, sorted, with confirmation for every one of them)
UDP	receive_data_via_udp(destination_path) => connection => create a map with all the files and packages received => write everything when the client sends `Done` marker (keeping in mind that packages need to be reordered, some may be missing or maybe the initial header (containing filename, packages count, etc) is missing)

Command line arguments

Script	Description
client.py	python3
client.py	python3 UDP ./test/source 127.0.0.1 7003
server.py	python3
server.py	python3 UDP ./test/destination 127.0.0.1 7003

Results (round 1 => streaming UDP vs stop-and-wait TCP)

Entity	Description	Messages	Bytes	Time
Message	Batches of 100 bytes (+ UDP header to sort blocks)			
Files	8798 files with a total size of 381MB			
TCP (Client)	Via TCP protocol	Messages sent: 4020687	Bytes received: 400869938	Time to receive/process: 384.31 seconds
TCP (Server)	Via TCP protocol	Messages received: 4020687	Bytes received: 400869938	Time to receive/process: 387.39 seconds
UDP (Client)	Via UDP protocol	Messages sent: 4011889	Bytes sent: 551118578	Time to send: 186.11 seconds
UDP (Sever)	Via UDP protocol	Messages received: 750097	Bytes received: 102253457	Time to receive/process: 215.37 seconds

Results (round 2 => streaming and stop-and-wait for both UDP and TCP)

Batches of 512 bytes (+ UDP header to sort blocks).

8798 files with a total size of 381MB.

Protocol	Mechanism	IO (Write files)	Messages	Bytes	Time
TCP (Client)	Streaming	YES	791821	400246350	23.87s
TCP (Server)	Streaming	YES	791821	400246350	25.28s
TCP (Client)	Streaming	NO	791821	400246350	18.97s
TCP (Server)	Streaming	NO	791821	400246350	20.31s
TCP (Client)	ACK	YES	791821	400246350	71.88s
TCP (Server)	ACK	YES	791821	400246350	73.44s
TCP (Client)	ACK	NO	791821	400246350	49.49s
TCP (Server)	ACK	NO	791821	400246350	51.01s
UDP (Client)	Streaming	YES	791821	400246350	18.26s
UDP (Server)	Streaming	YES	103605	53779384	34.01s
UDP (Client)	Streaming	NO	791821	400246350	23.87s
UDP (Server)	Streaming	NO	102254	53078947	25.28s
UDP (Client)	ACK	YES	791821	400246350	716.29s
UDP (Server)	ACK	YES	791821	400246350	732.60s
UDP (Client)	ACK	NO	791821	400246350	656.73s
UDP (Server)	ACK	NO	791821	400246350	658.72s

Conclusions (round 1)

Lost packages via UDP: 81.30%.

Lost packages via TCP: 0%.

TCP

(+) Correct order of packages with confirmation.

(+) Way more reliable and less coding overhead than UDP.

(-) Confirmation and authentication takes a lot of time.

(-) Actual time was 79% slower than UDP.

(+) All packages were delivered regardless of the mechanism.

UDP

(+) Faster than TCP (79% faster).

(-) Lost packages percentage may vary. Not recommendable for files.

(-) Packages are unordered. Even if all of them are received, you'd need extra data for mapping and sorting (along with the overhead to implement these). If you write these packages to disk, the speed you are winning from using this protocol is lost on IO. If you choose not to, it requires a lot of RAM. Even if you somehow track and write the files to disk on completion (all packages received) you still remain with all the incomplete files (random packages from different files).

(-) Without any kind of ACK the end marker package may be lost and the server doesn't know that the client finished sending data.

(-) Even with some sort of ACK.. if the client waits for a confirmation, the confirmation itself may never arrive. This might happen either because the confirmation from the server didn't reach the client or the package from the client did not reach the server - in this case we need a timeout. With a given timeout, we may send unwanted duplicated packages using more bandwith than TCP and also being slower to do so.

(-) Streaming method on UDP had an extremely low percentage regarding the files sent:

(1)

Good files: 186

Corrupted files (missing packages): 5988

Corrupted files (missing header): 2605

(2)

Good files: 193

Corrupted files (missing packages): 5892

Corrupted files (missing header): 2680

This gives us a percentage of 2.11-2.19% of useful data from all the 8798 files sent.

(-) Ack method on UDP was x10 slower than streaming but no packages were lost.