

RDT Programming

学号： N/A

姓名： N/A

专业： 计算机科学与技术

1 Introduction

This is a simple socket programming project that implements a file transfer system. The system consists of a server and multiple clients. The server is responsible for managing the clients and sending files to the clients. The clients can connect to the server, request files from the server, and save the files to the local path. The core of this project is implementing reliable data transfer (RDT) on top of the UDP protocol.

The features of the system include:

- The user can use command-line interface to interact with the system.
- The file transfer is reliable and large binary files are supported.
- The server can handle multiple active clients simultaneously.
- The connection supports both IPv4 and IPv6.
- The project can be compiled and run on both Windows and Posix systems (Linux, macOS). (SO_NO_CHECKS is not supported on Windows, so the client may not correctly calculate the number of corrupted packets on Windows.)

2 Usage

2.1 Commands

The client program can be run with the following command:

- `client-start domain=ipv4|ipv6 port=<number> timeout=[milliseconds] retries=[number]`
Start the client program with the specified domain, port, timeout, and retries.
- `client-request server=<ip> port=<number> file=<path> destination=<path>`
Request a file from a server and save it to the local path.
- `client-statistics`
Display the statistics of the client program, including the number of lost, corrupted, duplicated, and successful packets received.

The server program can be run with the following command:

- `server-start domain=ipv4|ipv6 port=<number> timeout=[milliseconds] retries=[number]`

Start the server program, with the specified domain, port, timeout, and retries.

- `server-stop`

Stop the server program.

Other commands include:

- `help`

Show the help message.

- `exit`

Exit the program.

2.2 Examples

Here we start two hosts *A* and *B* on the mininet environment, with the left one *A* as the client and the right one *B* as the server.

```
root@curriculumpractice:~# ./core/Core
client-start domain=ipv4 port=4567 timeout=1000 retries=5
client-start: success=true
```

```
root@curriculumpractice:~# ./core/Core
server-start domain=ipv4 port=3456 timeout=1000 retries=5
server-start: success=true
```

Then the client can request a file from the server. The progress will be displayed on the console.

```
root@curriculumpractice:~# ./core/Core
client-start domain=ipv4 port=4567 timeout=1000 retries=5
client-start: success=true
client-request server=10.0.0.2 port=3456 file=./core/Core destination=./core/recv
.img
client-progress: total=6774328 received=1485824 progress=21.93%
```

If the server reaches the maximum number of retries, it will stop the connection and display the error message.

```
root@curriculumpractice:~# ./core/Core
server-start domain=ipv4 port=3456 timeout=1000 retries=5
server-start: success=true
server-task: path=./core/Core port=4567 IP=10.0.0.1
server-error: reason=Timeout\ on\ 10.0.0.1:4567\ with\ maximum\ retries,\ connect
ion\ terminated
```

Cases of successful transfers is shown in part 2.3.

2.3 Test cases

All of the following test cases are run on the Mininet environment, with different network conditions. The file transferred is the executable file of this project, which is 6774328 bytes.

- Environment: loss = 0, corrupt = 0
Recommended timeout = 1000, retries = 5
The transfer takes 10784 ms.
Packets: lost = 0, corrupted = 0, duplicate = 2, successful = 6621.

```
root@curriculumpractice:~# ./core/Core
client-start domain=ipv4 port=4567 timeout=1000 retries=5
client-start: success=true
client-request server=10.0.0.2 port=3456 file=./core/Core destination=./core/receive
.img
client-progress: total=6774328 received=6774328 progress=100.00%
client-finish: duration=10784ms size=6774328 file=./core/Core
client-statistics
client-statistics: lost=0 corrupted=0 duplicate=2 successful=6621
```

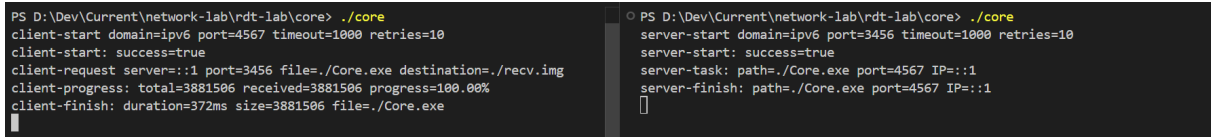
- Environment: loss = 1, corrupt = 1
Recommended timeout = 1000, retries = 5
The transfer takes 33765 ms.
Packets: lost = 489, corrupted = 275, duplicate = 482, successful = 7101.

```
root@curriculumpractice:~# ./core/Core
client-start domain=ipv4 port=4567 timeout=1000 retries=5
client-start: success=true
client-request server=10.0.0.2 port=3456 file=./core/Core destination=./core/receive
.img
client-progress: total=6774328 received=6774328 progress=100.00%
client-finish: duration=33765ms size=6774328 file=./core/Core
client-statistics
client-statistics: lost=489 corrupted=275 duplicate=482 successful=7101
```

- Environment: loss = 5, corrupt = 3
Recommended timeout = 1000, retries = 20
The transfer takes 403218 ms.
Packets: lost = 1859, corrupted = 1138, duplicate = 2485, successful = 9106.

```
root@curriculumpractice:~# ./core/Core
client-start domain=ipv4 port=4567 timeout=1000 retries=10
client-start: success=true
client-request server=10.0.0.2 port=3456 file=./core/Core destination=./core/receive
.img
client-progress: total=6774328 received=6774328 progress=100.00%
client-finish: duration=403218ms size=6774328 file=./core/Core
client-statistics
client-statistics: lost=1859 corrupted=1138 duplicate=2485 successful=9106
```

This is a test on Windows using localhost.



The image shows two terminal windows side-by-side. The left window shows the output of a client program, and the right window shows the output of a server program. Both programs are named 'core' and are running in a PowerShell prompt at the path 'PS D:\Dev\Current\network-lab\rdt-lab\core>'. The client program starts by setting domain=ipv6, port=4567, timeout=1000, and retries=10. It then sends a request to the server at port 3456 for a file named 'Core.exe'. The server program starts by setting domain=ipv6, port=3456, timeout=1000, and retries=10. It then receives the request from the client at port 4567 and sends back the file 'Core.exe'. Both programs finish successfully.

```
PS D:\Dev\Current\network-lab\rdt-lab\core> ./core
client-start domain=ipv6 port=4567 timeout=1000 retries=10
client-start: success=true
client-request server=:1 port=3456 file=./Core.exe destination=./recv.img
client-progress: total=3881506 received=3881506 progress=100.00%
client-finish: duration=372ms size=3881506 file=./Core.exe

PS D:\Dev\Current\network-lab\rdt-lab\core> ./core
server-start domain=ipv6 port=3456 timeout=1000 retries=10
server-start: success=true
server-task: path=./Core.exe port=4567 IP=:1
server-finish: path=./Core.exe port=4567 IP=:1
```

3 Compilation

The project can be compiled with the following commands:

- `make BUILDTYPE=debug`
Compile the project in debug mode.
- `make BUILDTYPE=release`
Compile the project in release mode.

The makefile is compatible with both platforms. `mingw32-make` from MinGW is recommended on Windows.

Be aware that the project uses the C++20 standard, and you need at least GCC 13 to compile the project. You may need to substitute `COMPILER=g++` with `COMPILER=g++-13` or higher in the makefile.

In the given image of the Mininet environment, `g++-13` can be installed with the following commands:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt install g++-13
```