VE489 -- Computer Networks

UM-SJTU Joint Institute

Electrical and Computer Engineering

VE489 Group 16

Demo Link: https://jbox.sjtu.edu.cn/l/V1Ho6O

Author	Student-ID
娄辰飞 Lou Chenfei	518021910610
陆昊融 Haorong Lu	518370910194

Final Project Report

In this project we aim at developing a socket program based on a overlay network. We need to let the client on one PC send a file to the server on another PC. Since in task 2 we've implemented an overlay network across two PCs, so in this task we mainly show how to write a socket program. In this project we use "Go" to write the socket program, due to its high efficiency in socket programming.

1. Configure the Overlay Network across PCs

Since the docker container needs to have access to network configurations, some of which does not gives permission to normal users, so we need to create two containers on two PCs again, with the permission granted.

To grant the permission to configure network, we do the following when creating the docker containers:

```
$ docker run -itd --cap-add=NET_ADMIN --name=server ubuntu:latest # --
name=client on another PC
```

In this way, the created container have access to modify network configurations.

Then, we just repeat what we've done in task 1 and 2 to install `etcd`, `flannel` and `ssh` on both containers.

In the client container, since we need to use udp to simulate the package loss, so we type the following commands to the terminal to set the loss probability to be 10%.

```
$ tc qdisc add dev eth0 root netem loss 10%
```

After this step, we ping the client PC from server PC, and we can see that they succeed to communicate with each other, while at sometimes the delay is larger, because of the loss probability.

2. Install Go inside the Container

For socket programming, we choose to use the language Go. The following procedures from this source shows how to install Go inside the ubuntu container.

```
# download the source files from the official website
$ wget https://golang.org/dl/go1.16.6.linux-amd64.tar.gz

# clear the previously-installed go
$ rm -rf /usr/local/go

# extract the package and copy it to /usr/local
$ tar -C /usr/local -xzf go1.16.6.linux-amd64.tar.gz

# add the command to the system path
$ export PATH=$PATH:/usr/local/go/bin

# check the version
$ go version
```

After the last step, the terminal should show the following message, which indicates that the installation is successful:

```
go version go1.16.6 linux/amd64
```

3. Write the Stop-and-Wait ARQ Programs in Go

The socket program for stop-and-wait ARQ consists of two major functions:

1. for client:
    - sends the packages and waits for ARQ
    - if ARQ timeouts, resends the package again
2. for server:
    - listens to package arrivals and returns ARQ if received any
    - if the sequence number for the received package does not mathc the expected sequence number, drop the package

Our source code can be found on our github repository $VE489$. The codes in the repo are in the following structure:

- /client/clint.go *implement the functions for client*
- /server/server.go *implement the functions for server*
- /output
    - client *binary executable for client*
    - server *binary executable for server*
- util/typeconvert.go *implement the type conversions*
- shakespeare.txt *the text file that we're going to transfer*

We first write and compile the codes on server's PC. After that, we use `scp` command to transfer `client` executable to client's PC.

After we implement the sources and build the corresponding binary executables, we finally get the transmission working as what we expect. The following figure shows the ARQ states during the transmission process:

The left terminal window shows the states of server, while the right terminal window shows the states of client. We can see that in the right terminal window, an ACK timeouts (because ACK is lost), so the client resends the same package again. And in the left window, the server detects that the ACK is duplicated, so it dropped the message.

Transmit Shakespeare's Text from One PC to Another

After we've implemented the stop-and-wait ARQ, now we try to transmit Shakespeare' text from one PC to another.

In the directory $VE489$ on server PC, we type the following commands in server's terminal to start listening

```
./output/server -f /root/VE489/received_text.txt
```

and on client PC, we type the following commands in client's terminal to start trasmissoin

```
./client -f shakespeare.txt
```

And on server's PC, we would get a new file named *received_text.txt*. This file should be identical to *Shakespeare.txt*, so we compare them using `diff` command, and the result is as what we expect – nothing differs between them.