Reliable UDP

Name : Prakash Rajagopal

Usage :

- Run `make scl` on silo and `make` to create app
- Run a `./app` with port as argument. This is the receiver listening on the named port `app -r 9000` The received file is saved with name `temp`
- Run another `./app` with filename, destination ip and port. This is the sender which sends the file to the receiver `app -s file localhost 9000` The file will be found in the `www` directory
- Examples can be found in the makefile

Header structure :

- Present in `header.cpp`
- Sequence number
- Ack represents message type and is an 16bit integer represented as an enum in the program. In the enum if it starts with R then message is transmitted to receiver and if it starts with S then it is for the sender.
  - RhasFileInfo - Represents that packet has file information
  - RData - Represent normal data message
  - ShasFileInfo - A file info ack for the sender, sender starts transmitting message
  - SduplicateAck - Duplicate ack, Go back N and repeat
  - SrepeatAck - A repeat ack to indicate that reciever has already got the packet and ensure there is no wait lock at sender side
  - SsuccessAck - Normal success ack
  - SwrapSequence - Ack to indicate wrap sequnce , the sequence number is used as the new base. The number is set as lastAcked and sequence base and all numbers start from 0

Steps :

Present in `udphandler.cpp` , where sender functions start with `send` and all handler functions for getting messages for both reciever and sender are named as `handler` functions

```
R: Initially Receiver starts up to listen on named port and receive file
S: Sender starts and first sends fileInfo -> Name and Size
R: Receiver receives and sends ack
S: Sender waits for an Ack from receiver before proceeding - This is not part of the transfer
    - This request gives the sender an idea of the RTT
S: Sender starts sending file chunks on receiving ack until window fills up which initially is : Window_Size = 1
    - In each case the RTT is updated whenever a successful ack is received
R: Receiver receives the packet and sends an  ack with header.seq as 1 + last_recieved_seq
S: Sender recieves ack and increases window size

R: In case of out of order packet with seq > lastReceived + 1 , then duplicate ack is set
 : If seq <= lastReceived + 1 then repeating ack is set to ensure that the previous ack was not lost
 : If timeout occurs then duplicate ack is sent similar to out of order packet
 : If seq = lastReceived + 1 , then it is a success . Sends a SsuccessAck.

S: If ack is a duplicate ack then treat it as drop and use congestion control. Sequence reset to lastRecievedAck + 1
 : If ack seq > lastRecievedAck , then lastRecievedAck = seq - 1. Is a success case , reduce window and resume any waiting threads
 : If ack has seq < lastRecievedAck ,i.e ack is Srepeatack then it is just ignored

R: When received bytes = fileSize , then reciever sends a final ack, waits for seq+1 msg from sender and exits
S: When it receives a seq > max_sequence possible then it sends out ack with seq+1 and exits
 : The final steps can be replaced with a special ack message to complete
```

Adaptive retransmission - Jacobson/Karels algo

Timer util methods and jacob karnels is implemented in `timeutil.cpp`

- Initially the sender gets an estimated RTT when it sends file info and gets an ack back
- Each succesful receive and send is used to calcualte RTT - since packets always arrive in order, This is done using jacob karnels
- The RTT is displayed for each change

Congestion control

- Slow start initially
- On each succesful ack the window size is increased by that number
- When a duplicate ack or timeout occurs then we make window size as 1 and make thread wait until it gets acks and window size reduces to 1
- Then we increase window size in same way per ack until it reaches `window_size = congestion_window_size/2`
- On reaching this stage , the app enters AIMD phase - where each ack leads to increase by 1
- Whenever there is a duplicate ack or whenever there is a timeout for an ack at the sender side then the window size is halved.

Simulate loss and delay

- Drop probability set in app in combinations with `srandom(clock())` and `random()` is used to drop packets whenever enabled
- Latency set using `srandom(clock())` in combination with latency time set in config which is used to make the listener thread (on both receiever and sender)
- Latency using `usleep` with time as `(randomnumbr % 10) * 0.1 * latencyTime`

State handling and config

All of the state variables are declared in `state.cpp`

- All the global app config is present in `initApp` function
- The current state of each operations is stored in `senderState` and `recieverState`. This struct also holds RTT , window counter , max size , e.t.c
- This struct also encapsulates the mutex and methods for `thread_wait` , `thread_resume` and `thread_timedwait`