Lab2 – 書面報告

C34031027 資訊四甲 張睿

除了此word檔以外，我繳交了另外5個資料夾：

1. unicast：我寫的code為client1.c和server1.c，執行方式等請參考下方第「一」點。
2. multicast：我寫的code為client2.c和server2.c，執行方式等請參考下方第「二」點。
3. unicastfec：我寫的code為test資料夾中的client3.cpp和server3.cpp，執行方式等請參考下方第「三」點。
4. multicastfec：我寫的code為test資料夾中的client4.cpp和server4.cpp，執行方式等請參考下方第「四」點。
5. file: 存放測試檔，分別為約1.9MB的文字檔“testfile.txt”以及約2MB的圖檔“2MB.jpg”。
6. Unicast

* 每次傳輸大小：UDP datagram 傳輸大小有限。預設是一次傳1/20的檔案，而若仍太大，server中會以udp\_subchunk\_size()再往下切。
* 遺失率紀錄：Server每次做sendto()前，會在該次資料前串接序號（一個int），一起傳給client。Client則會在recv時，紀錄收到資料對應的序號，並在最後印出沒收到的資料序號，並計算遺失率。
* 遺失率資料：
  + 測試檔案一：2MB.jpg (約2MB的圖檔)
  + 測試檔案二：testfile.txt (約1.9MB的文字檔)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 傳輸檔 | 第一次 | 第二次 | 第三次 | 第四次 | 第五次 | 平均 |
| testfile.txt | 0 | 0 | 0 | 0 | 0 | 0 |
| 2MB.jpg | 20.3 | 0 | 19.1 | 10.4 | 8.7 | 11.7 |

註：表格中為百分比

* 執行方式： （在unicast資料夾下執行）
  + 編譯server1.c: gcc -o server1 server1.c
  + 編譯client1.c: gcc -o client1 client1.c
  + 先執行server1: ./server1 localhost 5566 ../file/testfile.txt （或2MB.jpg）
  + 再執行client1: ./client1 localhost 5566
  + 會output出udp\_receiver.txt（或udp\_receiver.jpg）(另外，client端會顯示遺失率和沒收到的封包的序號)

1. Multicast

* 每次傳輸大小和遺失率紀錄同Unicast。
* 輸出檔案：會在檔名後面加5碼亂數，這樣一次執行多個client時，各client的輸出檔才不會因檔名相同而互相覆蓋。
* 遺失率資料
  + 測試檔案一：2MB.jpg (約2MB的圖檔)
  + 測試檔案二：testfile.txt (約1.9MB的文字檔)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 傳輸檔 | 各client | 第一次 | 第二次 | 第三次 | 第四次 | 第五次 | 各client平均 | 總平均 |
| Textfile.txt | clientA | 16.7 | 3.2 | 0 | 5.4 | 4.5 | 6.0 | 4.7 |
| clientB | 0 | 0 | 4.5 | 17.2 | 3.2 | 5.0 |
| clientC | 0 | 0 | 1.3 | 14.9 | 0 | 3.2 |
| 2MB.jpg | clientA | 5.4 | 0 | 16.6 | 19.5 | 19.5 | 12.2 | 11.8 |
| clientB | 4.1 | 18.3 | 10.4 | 15.4 | 7.5 | 11.1 |
| clientC | 0 | 20.0 | 0 | 22.8 | 18.3 | 12.2 |

註：表格中為百分比

* 執行方式：（在mulicast資料夾下執行）
  + 編譯server2.c: gcc -o server2 server2.c
  + 編譯client2.c: gcc -o client2 client2.c
  + 在三個不同視窗分別執行client2: ./client2
  + 再執行server2: ./server2 ../file/testfile.txt （或2MB.jpg）
  + 會output出三個udp\_receiverXXXXX.txt（或udp\_receiverXXXXX.jpg）(另外，client端會顯示遺失率和沒收到的封包的序號)

1. Unicast + FEC

* 參考github檔案：<https://github.com/randombit/fecpp/blob/master/test/benchmark.cpp>
* 傳輸流程：
  + Server: 讀進整個檔案。
  + Server: 將檔案傳給encoder。
  + Server: 每次encoder的callback被呼叫，就會將傳進callback的share、share id、share length包成一個struct Sendele，再將此struct傳給client。
  + Client: 將收到的struct Sendele存進map中。
  + Client: 當不再有struct Sendele傳過來，會將map傳給decoder。
  + Client: 每次decoder的callback被呼叫，會將傳進callback的block存到一個global array的block id那一格。
  + 將global array中的資料逐一fwrite到輸出檔。
* 遺失率紀錄：
  + client會紀錄收到的share的id，並計算遺失率（沒遺失的話，總共要收到FEC\_K個share）
  + client會紀錄從decoder decode出的block的id，並計算遺失率（若沒有遺失資訊，decoder總共要產出FEC\_N個block）
* 限制：
  + 無法進行圖檔傳輸：我的測試結果是，若傳輸檔案中出現null character（整個byte的值為零）的話，從decode出來的結果可看出，「在傳輸時與該character同個share的資料，其中在該character之後的資料」都會變為零。在傳輸txt檔時不會因此出錯（因為null character在檔案的最末），但在傳輸圖檔時資料就會損壞。
  + 檔案大小不能太大（大約能到2MB出頭）：依參考github的readme描述，encoder端會將檔案分成n個share傳出，而n最大只能設成256。而我是直接把檔案切成k份，因為這樣可使每次傳輸都對應到0~255的其中一個shareID，receive端就能依照shareID去排出正確順序。既然是把檔案切成固定的255份，那檔案就不能太大，否則檔案的1/255還是會超出UDP的傳輸上限。有想到的解決方法為（沒有寫在這次的作業裡），可將檔案分兩批傳（也就是傳255\*2 = 510個share），但要如何解決shareID重複的問題呢？可以在send端手動將第二批share的ID改成比如1000~1255。
* 遺失率資料：
  + 測試檔案：testfile.txt (約1.9MB的文字檔)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Loss | 第一次 | 第二次 | 第三次 | 第四次 | 第五次 | 平均 |
| Share loss | 5.9 | 0 | 0 | 31.0 | 2.0 | 7.8 |
| Block loss | 0 | 0 | 0 | X | 0 |  |

註：表格中為百分比

註：X表示client端收到少於K個share，因此decoder無法還原出資訊

* 執行方式：（在unicastfec資料夾下執行）
  + 建議先：make clean，否則編譯可能出錯
  + 編譯server3.cpp: make server3
  + 編譯client3.cpp: make client3
  + 先執行server3: ./server3 localhost 5566 ../file/testfile.txt
  + 再執行client3: ./client3 localhost 5566
  + 會output出udp\_receiver.txt (另外，client端會顯示遺失率)

1. Multicast + FEC

* 參考github檔案、傳輸流程、遺失率紀錄、限制同「三、Unicast+FEC」。
* 遺失率資料：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 各client | loss | 第一次 | 第二次 | 第三次 | 第四次 | 第五次 | 各client  share loss  平均 | share loss  總平均 |
| clientA | Share | 0 | 0 | 25.0 | 0 | 0 | 5.0 | 1.7 |
| Block | 0 | 0 | X | 0 | 0 |
| clientB | Share | 0 | 0 | 0 | 0 | 0 | 0 |
| Block | 0 | 0 | 0 | 0 | 0 |
| clientC | Share | 0 | 0 | 10.6 | 0 | 0 | 0 |
| Block | 0 | 0 | X | 0 | 0 |

註：表格中為百分比

註：X表示client端收到少於K個share，因此decoder無法還原出資訊

* 執行方式： （在mulicastfec資料夾下執行）
  + 建議先：make clean，否則編譯可能出錯
  + 編譯server4.c: make server4
  + 編譯client4.c: make client4
  + 在三個不同視窗分別執行client4: ./client4
  + 再執行server2: ./server2 ../file/testfile.txt
  + 會output出三個udp\_receiverXXXXX.txt (另外，client端會顯示遺失率)

1. 結果討論

* 在做Lab1時有觀察receive端的輸出檔案，本以為資料遺失的情況都是「檔案的前面一大段都有收到，而在檔案的某處之後的封包全部遺失」，但因為這次lab有加入序號，印出沒收到的封包序號後才發現遺失也確實會發生在兩個沒有遺失的封包之間。
* 在報告的Unicast、Multicast部分（一、二部份）可看出圖檔比較容易遺失。
* Unicast的情況下，測試結果顯示有加入FEC反而遺失率較高，可能為封包較大所致。
* Unicast的情況下，測試結果顯示有加入FEC遺失率較低。

三、如果compile沒過請先Make clean:

<https://stackoverflow.com/questions/17126384/g-output-file-not-recognized-file-format-not-recognized>