RESF: Reliable UDP Implementation

Xianghang Mi
xmi@iu.edu

1. Abstract

In this project, I implement a reliable application layer protocol based on UDP. I call this implementation as RESF (reliable, efficient, smart, flexible). To provide reliable data transmission service, RESF implements many features of TCP, such as sliding sending and receiving window, time out and retransmission, three duplicates and fast transmission, flow control and congestion control. The architecture of REFS is as below in Figure 1.

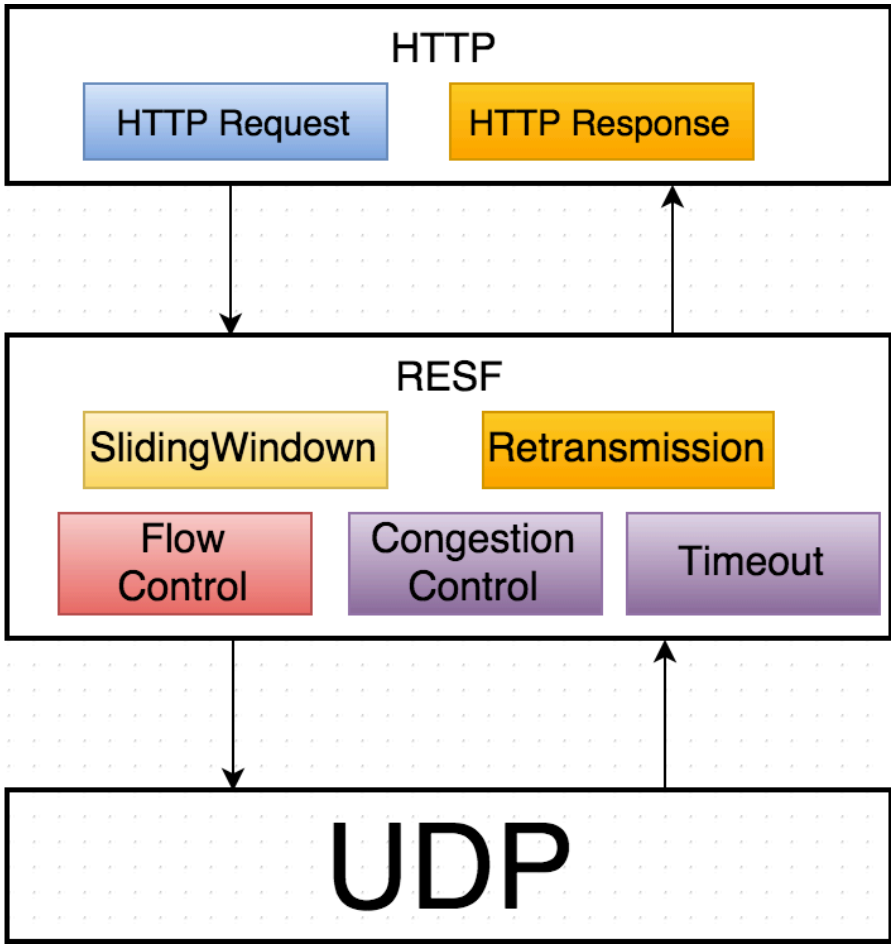

Figure 1.RESF Architecture

2. Design and Implementation

2.1 Header format

The header format is as Figure 2.
Sequence Number: 32bits, used to guarantee the order of received data.
Acknowledge Number: 32 bits, used to for cumulative acknowledge.
Data Length: 16 bits, byte length of the data in this segment. It is used by receiver to do verification.
Receive Window: 16bits, used for flow control
Flags: 8bits, if the first bit is set, then the acknowledge number is valid.

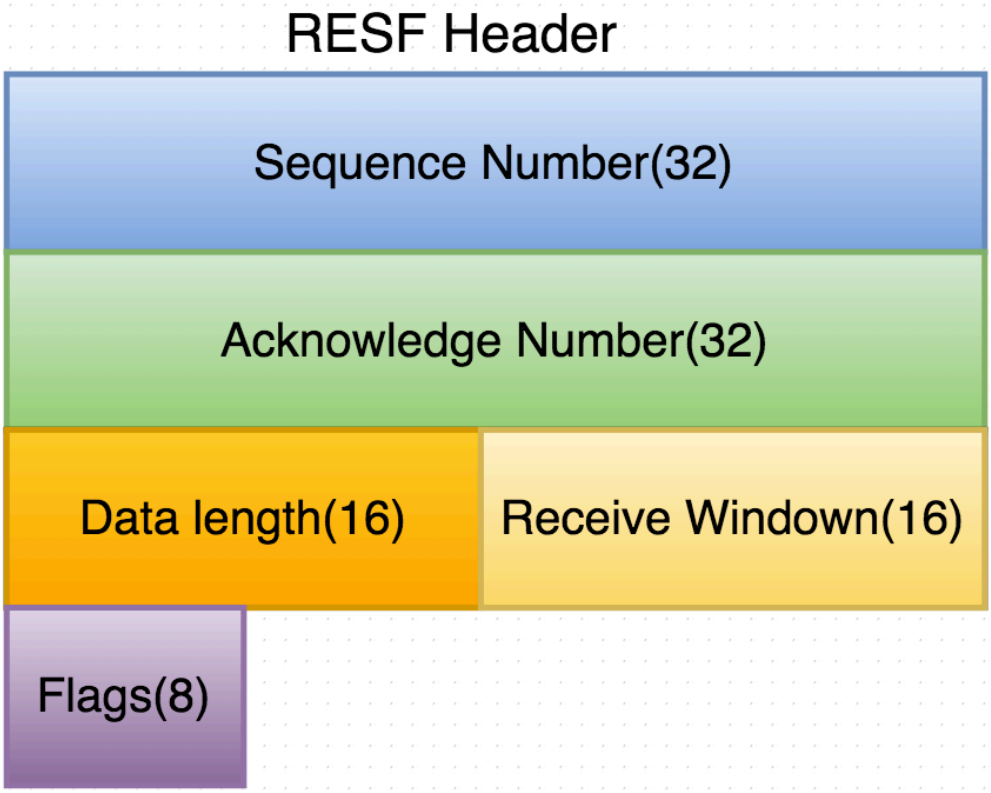

Figure 2. RESF Header Format

2.2 User Guide

- Compile

Run "./run" in the base directory, Then in the "bin" directory, you can find "server" and "client" executables.

- Run Server

Go to bin dir, run "./server port rwndSize"

- Run Client

Go to bin dir, run "./client serverIp serverPort rwndSize remoteFileName [localFileName]" the localFileName is optional, you can use it to indicate the filename where to save the received file content. Without this parameter, the remote file content will be printed on the screen.

3. Experiments

3.1 Timeout and Retransmission

In this experiment, I test the behavior of my program when timeout happens. Just as design requirement, it will retransmission the oldest unacknowledged segment and double the timeout value. The result is shown in Figure 3.

```
../src/http_server.c main lastRecved 29 lastRead 29 lastsent 60478 lastwrite 64953, lastAcked 0
95
before reset, timeout is 1000000, estimateRtt 95 devRtt 0
after reset, timeout is 95, estimateRtt 95 devRtt 0
../src/resf.c timeout timeout and retransmit before 95
before reset, timeout is 95, estimateRtt 100 devRtt 9
../src/resf.c timeout timeout and retransmit after 190
after reset, timeout is 136, estimateRtt 100 devRtt 9
../src/resf.c timeout timeout and retransmit before 136
before reset, timeout is 136, estimateRtt 117 devRtt 38
../src/resf.c timeout timeout and retransmit after 272
after reset, timeout is 269, estimateRtt 117 devRtt 38
../src/resf.c timeout timeout and retransmit before 269
before reset, timeout is 269, estimateRtt 154 devRtt 94
../src/resf.c timeout timeout and retransmit after 538
after reset, timeout is 530, estimateRtt 154 devRtt 94
before reset, timeout is 530, estimateRtt 197 devRtt 147
after reset, timeout is 785, estimateRtt 197 devRtt 147
../src/resf.c timeout timeout and retransmit before 785
before reset, timeout is 785, estimateRtt 237 devRtt 181
../src/resf.c timeout timeout and retransmit after 1570
after reset, timeout is 961, estimateRtt 237 devRtt 181
before reset, timeout is 961, estimateRtt 280 devRtt 212
after reset, timeout is 1128, estimateRtt 280 devRtt 212
../src/resf.c timeout timeout and retransmit before 1128
before reset, timeout is 1128, estimateRtt 320 devRtt 229
../src/resf.c timeout timeout and retransmit after 2256
after reset, timeout is 1236, estimateRtt 320 devRtt 229
../src/resf.c timeout timeout and retransmit before 1236
```

Figure 3. Timeout and Retransmission

3.2 Flow Control

As Figure 4 shows, the program actually uses the rwnd header field to tell the other endpoint how much you can send.

```
../src/resf.c startRecv Line 587:  before refresh rwnd 28055
../src/resf.c startRecv Line 589: after refresh rwnd 26615
../src/resf.c startRecv Line 587:  before refresh rwnd 26615
../src/resf.c startRecv Line 589: after refresh rwnd 25175
../src/resf.c startRecv Line 587:  before refresh rwnd 25175
../src/resf.c startRecv Line 589: after refresh rwnd 23735
../src/resf.c startRecv Line 587:  before refresh rwnd 23735
../src/resf.c startRecv Line 589: after refresh rwnd 22295
../src/resf.c startRecv Line 587:  before refresh rwnd 22295
../src/resf.c startRecv Line 589: after refresh rwnd 20855
../src/resf.c startRecv Line 587:  before refresh rwnd 20855
../src/resf.c startRecv Line 589: after refresh rwnd 19415
../src/resf.c startRecv Line 587:  before refresh rwnd 19415
../src/resf.c startRecv Line 589: after refresh rwnd 17975
../src/resf.c startRecv Line 587:  before refresh rwnd 17975
../src/resf.c startRecv Line 589: after refresh rwnd 16535
../src/resf.c startRecv Line 587:  before refresh rwnd 16535
../src/resf.c startRecv Line 589: after refresh rwnd 15095
../src/resf.c startRecv Line 587:  before refresh rwnd 15095
../src/resf.c startRecv Line 589: after refresh rwnd 13655
../src/resf.c startRecv Line 587:  before refresh rwnd 13655
../src/resf.c startRecv Line 589: after refresh rwnd 12215
../src/resf.c startRecv Line 587:  before refresh rwnd 12215
../src/resf.c startRecv Line 589: after refresh rwnd 10775
../src/resf.c startRecv Line 587:  before refresh rwnd 10775
../src/resf.c startRecv Line 589: after refresh rwnd 9335
../src/resf.c startRecv Line 587:  before refresh rwnd 9335
```

Figure 4. Flow Control

3.3 Congest Control

In this congest control experiment, I test how the congestion window and congestion stage change when different events happen. There are several interesting points.

- Although the initial ssthresh is 64 KB, it can become very small if several timeouts happen in a short time.
- If timeout happens several times in a short time, the transition from slow start to congest avoidance can be very quickly

Some detailed data is as Figure 5.

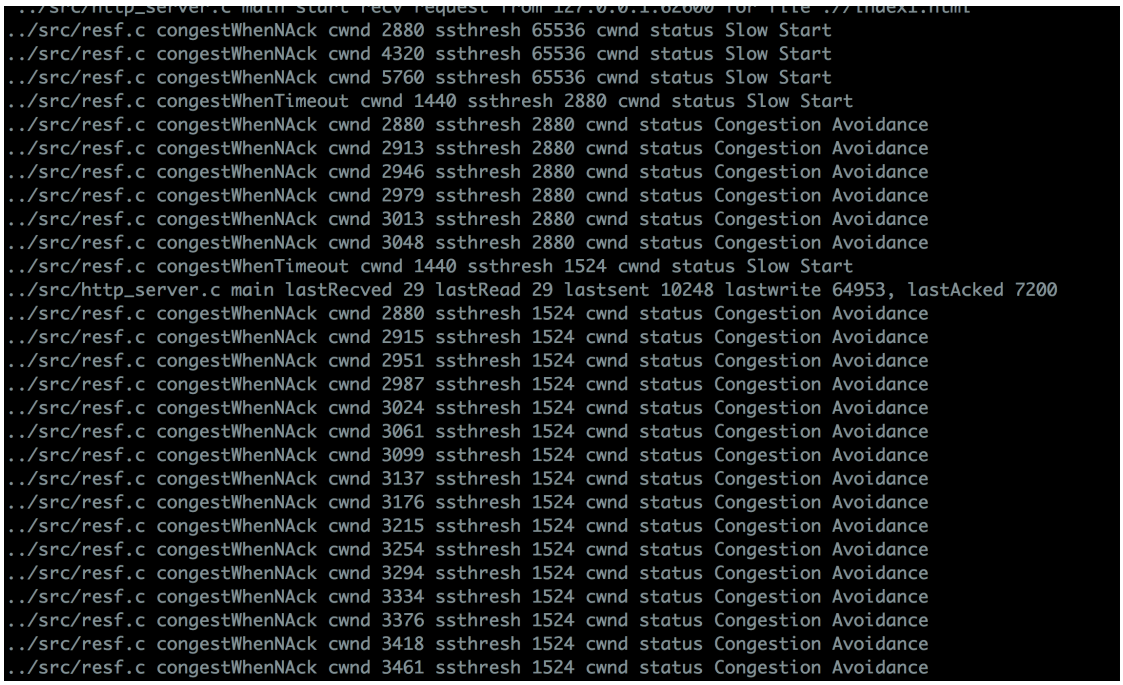

Figure 5. Congestion Control