**南京大学本科生实验报告**

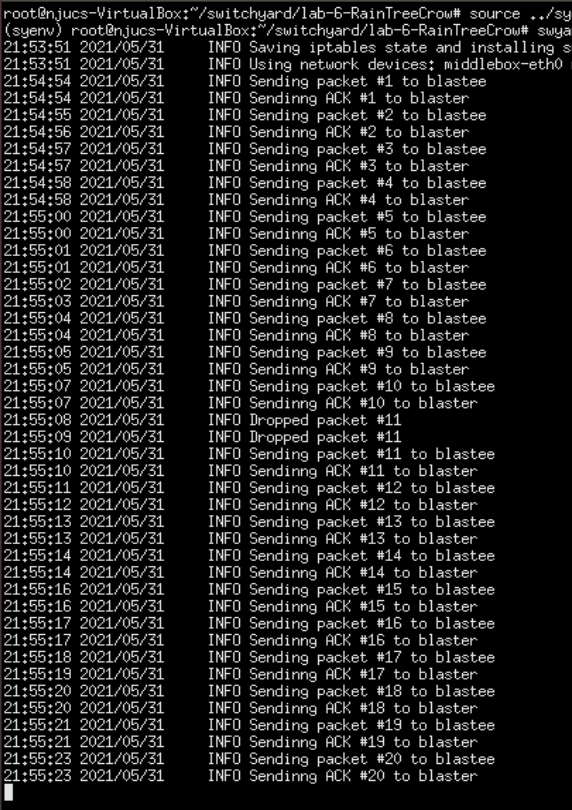
课程名称：**计算机网络** 任课教师：田臣/李文中 助教：

|  |  |  |  |
| --- | --- | --- | --- |
| 学院 | **计算机科学与技术系** | 专业（方向） | **计算机科学与技术** |
| 学号 | **191220154** | 姓名 | **张涵之** |
| Email | **1683762615@qq.com** | 开始/完成日期 | **2021/5/30-2021/6/1** |

1. **实验名称：Lab 6: Reliable Communication**
2. **实验目的：**

Build a reliable communication library in Switchyard.

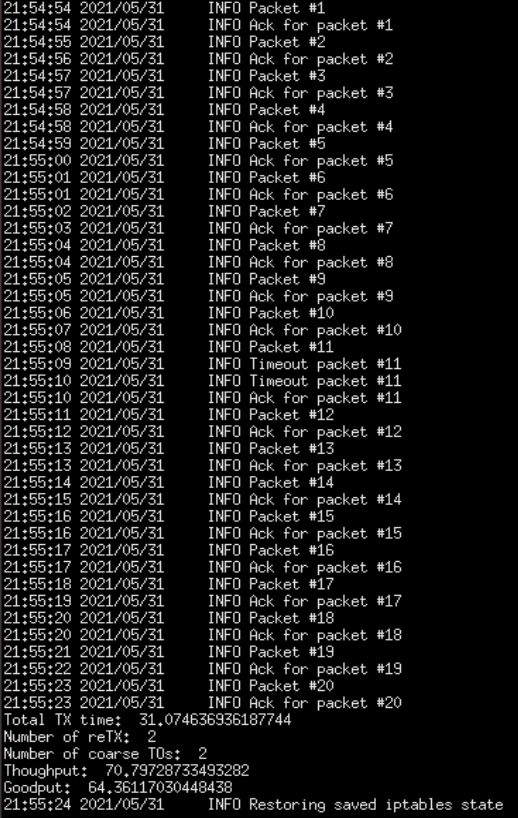
1. **实验内容**
   1. Task 2: Implement the features of middlebox
   2. Task 3: Implement the features of blastee
   3. Task 4: Implement the features of blaster
2. **实验结果**
   1. Running middlebox with the drop rate of 0.19



* 1. Running blastee with blaster IP 192.168.100.1 and num 20



* 1. Running blaster with blastee IP=192.168.200.1, num=20, length=100, sender window=5, timeout=1000 and receive timeout=100



From the log info we can infer that middlebox dropped packet #11 twice, and blaster raised two timeouts for the packet, demanding retransfer of the ack.

1. **核心代码**
   1. Overview

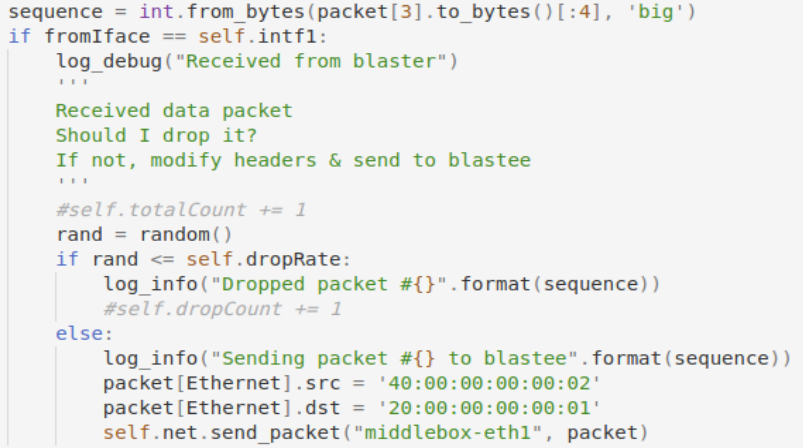
The middlebox, as its name suggests, serves as a transfer station between the blaster and the blastee, it has some of the basic functions of a hub/switch/router such as receiving and forwarding packets, but is a more simplified version. It only has two interfaces, that is to say, no ARP and no forward table, when the middlebox receives a packet from one interface, it sends it out the other.

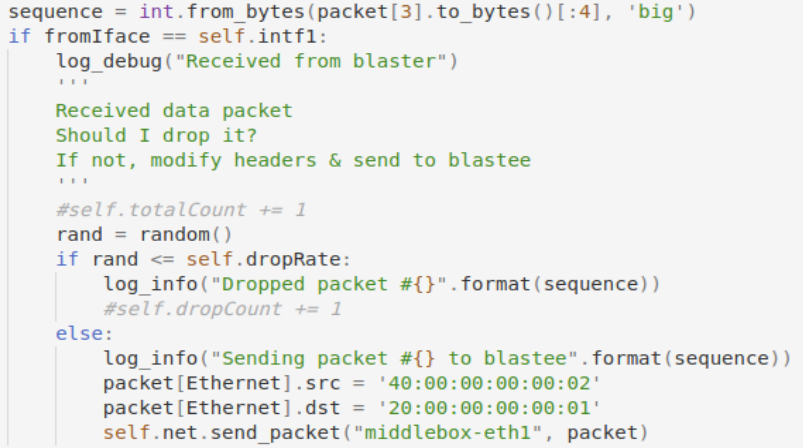
The blastee receives packets and sends back ack, simple as that.

The blaster is the most complicated. Suppose it imitates the transfer process of a file, which it breaks down into num pieces. The pieces are put in a queue and wait to be sent in their original order. The sender window can hold at most () such packets, as they except the corresponding ack. Once a packet has received its ack and there are no other packets waiting to be ack’d before it, it can come out of the queue. The blaster continuously test timeout, which occurs when the time past after the last ack packet exceed the set timeout argument. Once the timeout is detected, the first unack’d packet in the window is resent. Strategies are implemented so that one packet is sent in one round. When all num packets have been ack’d successfully, blaster print some statistics and then shutdown.

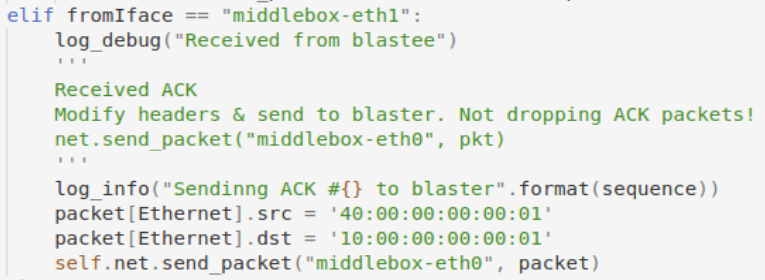
* 1. Task 2: Middlebox

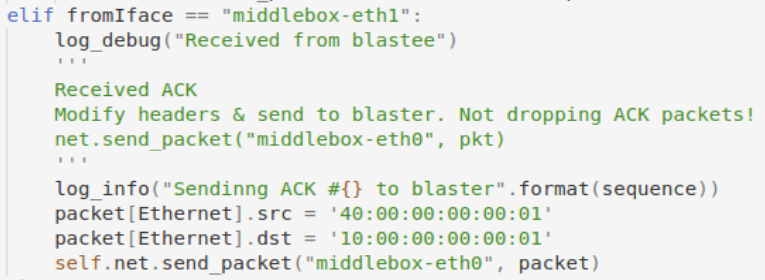
Modification in the function handle\_packet:





This part deals with packets from blaster to blastee. A random number between 0 and 1 is generated, if it is below drop rate the packet is dropped, otherwise it is sent. This is to simulate the possible packet loss in a real network.

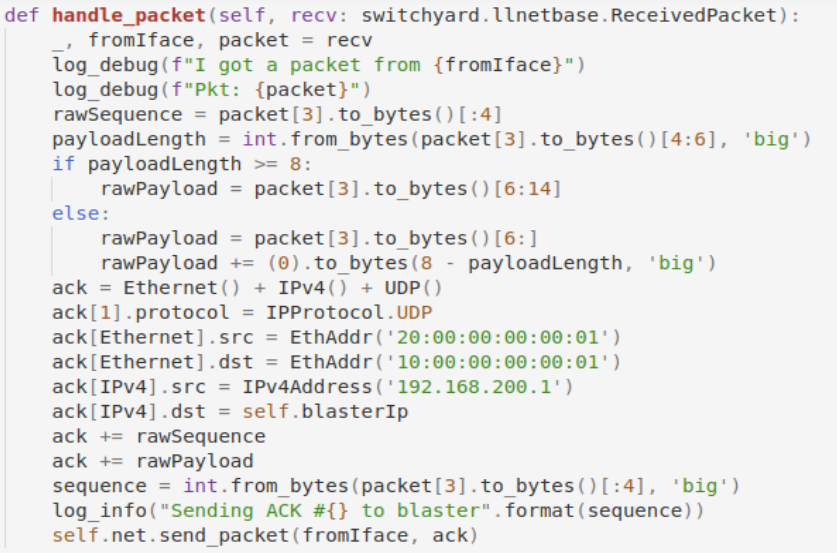




This part deals with ack from blastee to blaster. No ack is dropped here.

* 1. Task 3: Blastee

Modification in the function handle\_packet:



The three packet headers are generate using functions, while other data directly attached to the packet. The RawPacketContents header looks complicated and this works, so anyway. The sequence number is encoded (and decoded) using big-endian format, and the payload is set to a fixed size. If the received packet’s payload is longer than the length, the first 8 bytes are taken. Otherwise, if it is shorter than the length, some 0s are added to the payload to fill the length.

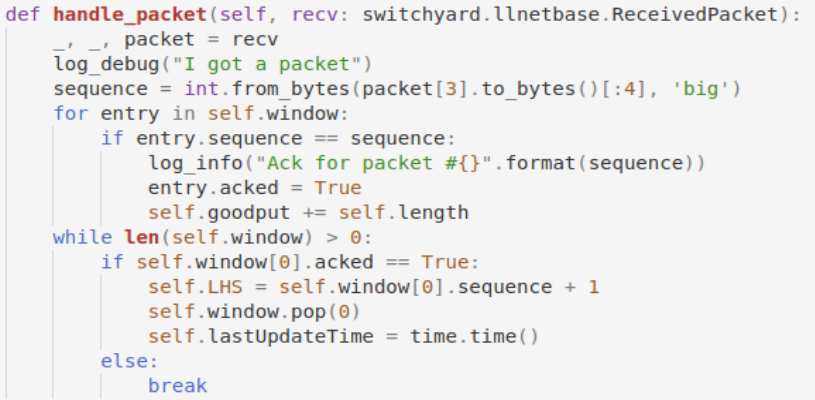
* 1. Task 4: Blaster

First, we look at the parameters and variables added to the init function:



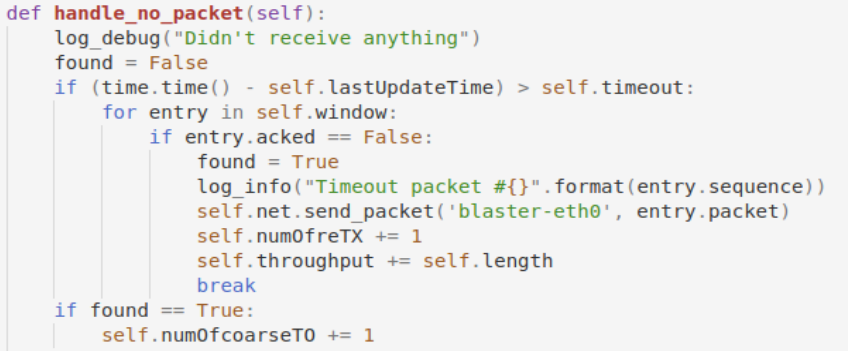
As we can see the sequence start from 1, the number of the first packet.

Then we modify the handle\_packet function:

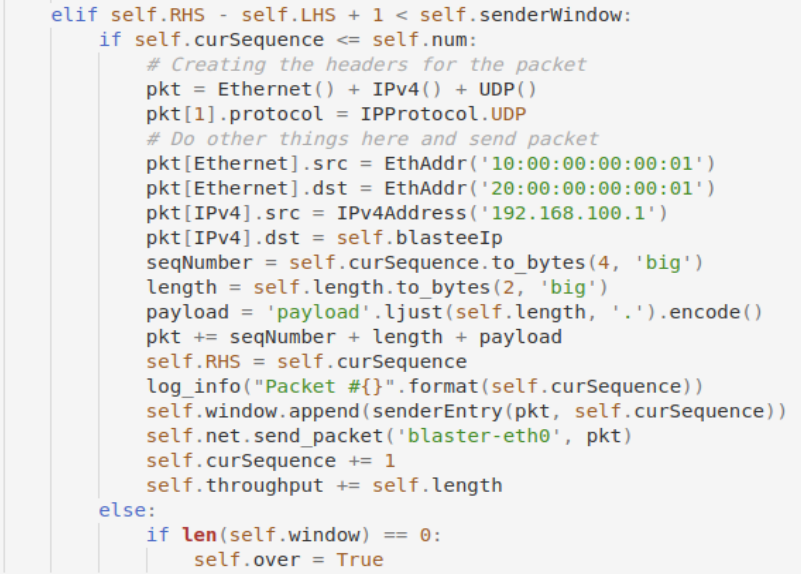


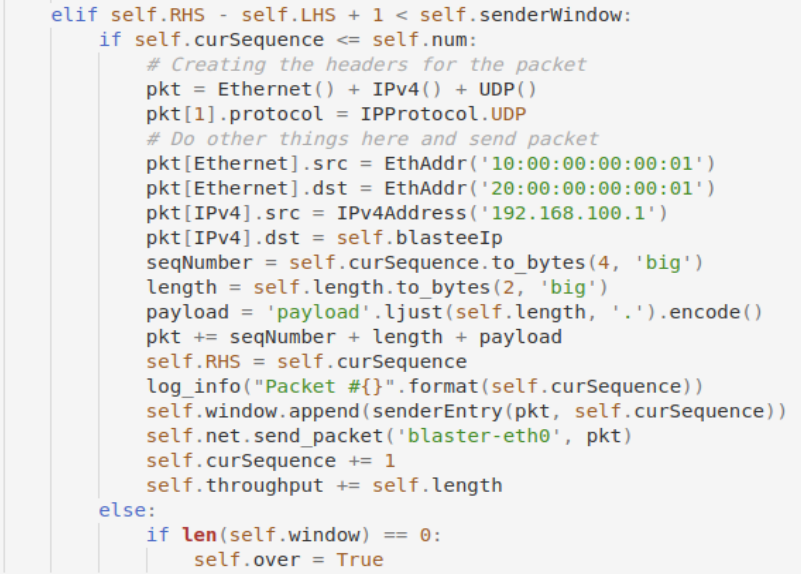
Each time an ack is received, the blaster checks its queue for the corresponding packet. The packet is marked “ack’d”, and length is added to the total goodput. Blaster then checks each packet in the front of the queue, if it has been ack’d it is removed, otherwise the loop stops, for every packet after an unack’d one should be blocked there until each entry before it has been removed.

Then we modify the handle\_no\_packet function:



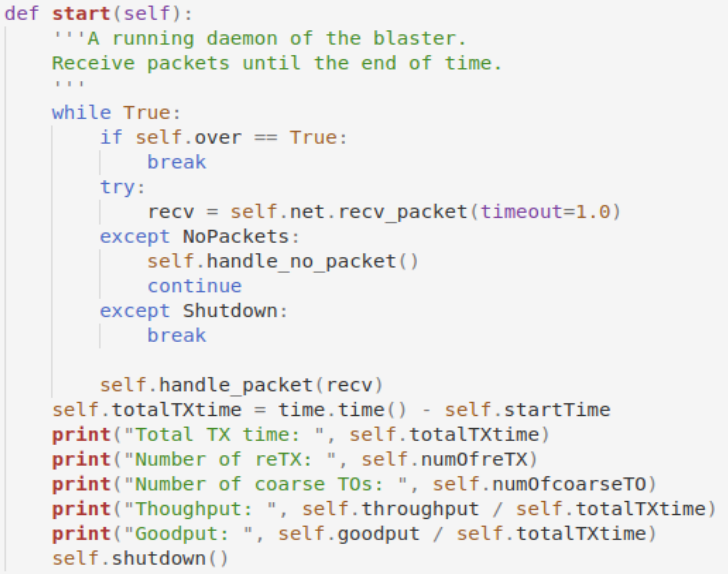
First, we check if there is a timeout, if so, check the sender window for the first unack’d packet, if found, resend packet, update number of reTX and the total throughput. After the search, also update the number of coarse timeout.





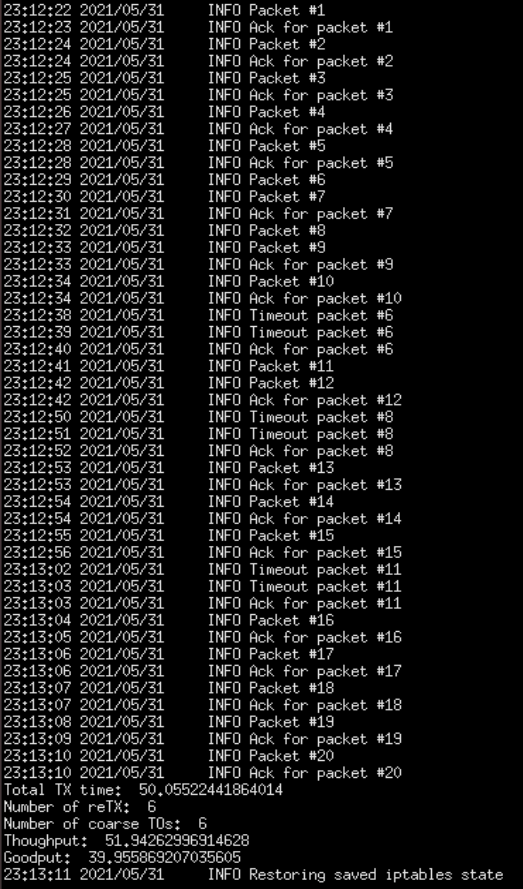
If there is a timeout unack’d packet and it is resent, we shall not send any other packets in this round. Otherwise, we check the sender window and the current sequence number. If the window is not full yet (the number of packets trapped in a traffic jam blocked by one or more unack’d packet is less than the window size), and the current sequence number is no more than the num of packets to be transferred, a packet is constructed, sent and added to the queue. The RHS is then updated to the next number above the current sequence, the sequence is incremented by 1, and length is added to the total throughput. If the sequence is more than num and the send window is empty (in other words, all the packets sent have been successfully ack’d, the blaster should shutdown.

There are also some modifications in the start function:



In each loop the blaster check if over==True, that is to say, all num packets are transferred successfully. It then breaks away from the loop. It then calculates and prints all the information required in the manual and calls shutdown.

1. **总结与感想**
   1. In this lab I still used self-defined classes and queues, like in the previous labs. I also tried to make the logic as simple and efficient as possible. When I went to the TA to test lab 4, he asked me why I put every packet in the wait queue after I receive it, instead of sending the ones than can be matched in forwarding table and save the rest that need ARP. So, this time I send every packet at once upon receiving it, and then put them in a queue for resend.
   2. The entire lab appears more confusing than the ones before it. Perhaps because there are three separate modules that shall cooperate. My suggestion is to read through the manual and find out what the three modules respectively does, how they interact and what packets they each send and receive. Once you have an overview of the whole structure, the rest is quite simple.
   3. 100 packets are hard to follow, so I tried 20. In the screenshot above it seems that each timeout is directly followed by its corresponding ack. I suppose the time is too short so other acks have not arrived yet, so here I tried an extremely large timeout, and it worked. At most five packets in the window at a time.



One thing that confuses me is the number of reTX, with the same parameters it can vary from 2 to 7, perhaps the random function is not that random.