

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	专业（方向）	计算机科学与技术
学号	201220062	姓名	黄子睿
Email	201220062@smail.nju.edu.cn	开始/完成日期	5/6-5/8

1. 实验名称：

Lab5 Reliable Communication

2. 实验目的：

通过 blaster, blastee 与 middlebox 的设计，实现简单的可靠传输（Reliable communication），主要关注其中的滑动窗口机制与重传机制。

3. 实验内容：

Task2 Middlebox

MiddleBox 模拟了简单网络中的路由器，主要功能为转发 blaster 与 blastee 相互之间发送的报文，并以一定概率丢失 blaster 发给 blastee 的报文。

下面给出 MiddleBox 的伪代码：

1. `def` Middlebox:
2. 以一定概率丢失 `blaster` 发送给 `blastee` 的报文
3. 修改报文的 `ethernet` 头，并将 `IPv4` 报头中的 `ttl` 字段减一
4. 将修改后的报文发送给目的地址

Task3 Blastee

在收到 Blaster 发送的报文后，blastee 需要发送 ACK 应答报文。

首先填充 ACK 报文的 ethernet 报头，IPv4 报头与 UDP 报头，前两者由于网

络结构固定，因此直接硬编码就可以了；后者在本次实验中没有直接的用途，只是需要占位防止报错。

同时，ACK 报文需要构造序号 seqnum 与 payload 项，此二者利用 blaster 发送的报文实现。首先从 blaster_pkt 中取出代表序列号的二进制位串 seqbyte 与代表可变装载字段长度的字段 lengthbyte，并通过 struct.unpack 将 lengthbyte 转化为整型数 length。如果 length 小于 8，则将 ACK 中的 payload 部分全赋为 0，否则取 blaster_pkt.payload 得前 8 比特填入 ACK.payload 即可。

下面给出伪代码：

```
1. def blasteer_handle_packet:
2.     get blaster_pkt from middlebox
3.     ackpkt := Ethernet() + IPv4() + UDP()
4.     implement ackpkt[0], ackpkt[1] and ackpkt[2] by hard code
5.     seqbyte := blaster_pkt.seqbyte
6.     lengthbyte := blaster_pkt.lengthbyte
7.     length = lengthbyte to int
8.     payload := None
9.     if length < 8:
10.        fill payload with 0
11.     else:
12.        fill payload with the first 8 bits in blaster_pkt.payload
13.     rpc := seqbyte + payload
14.     ackpkt append with rpc
15.     send packet ackpkt to middlebox
```

Task4 Blaster

对于 Blaster 而言，主要考虑以下三个功能的实现：

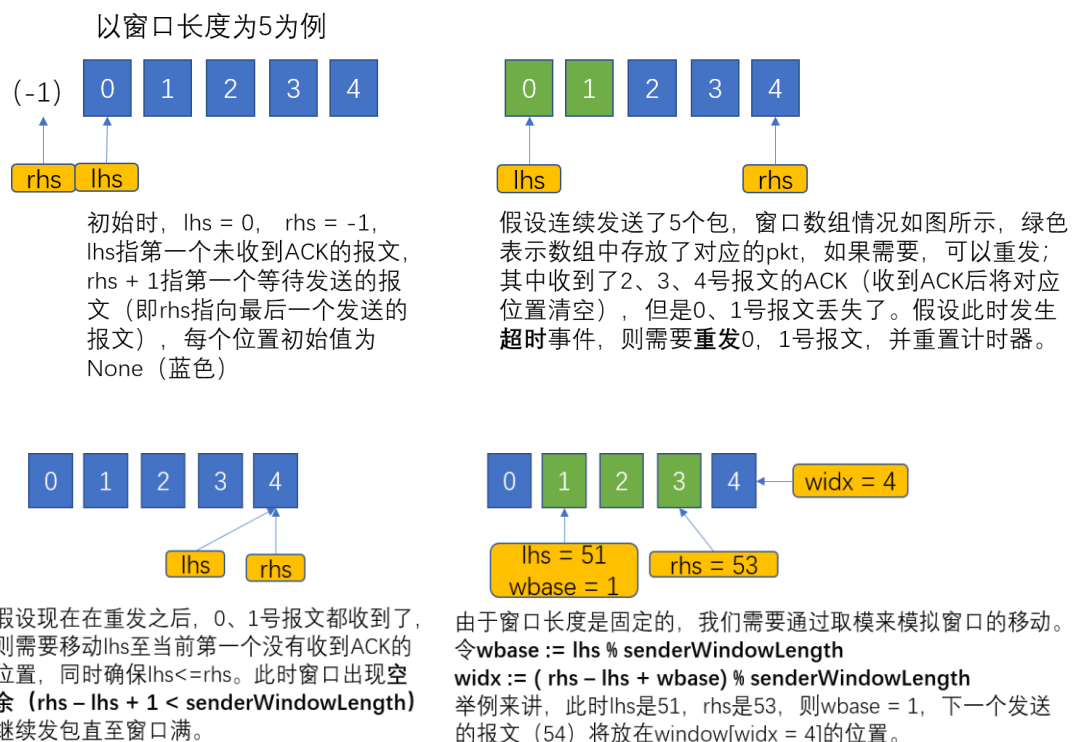
1. 实现滑动窗口
2. 实现超时重传
3. 为了结束时的状态打印，记录运行时的相关数据

首先考虑滑动窗口的实现：

滑动窗口的实现需要增加三个数据成员，分别是滑动窗口的左端，右端以及代表滑动窗口的数组本身。

```
1. # for snederWindow
2. self.lhs = 0
3. self.rhs = -1
4. self.window = [None]*self.senderWindowLenth
```

可以看到自始至终，我们只需要 `self.senderWindowLenth` 长度的滑动窗口，在具体的运行过程中，只需要不断取模就可以实现窗口滑动的效果。具体来讲，固定 `lhs` 后，`rhs` 不大于 `rhs+4`，随着不断发送新的报文与接受到 ACK 报文，`rhs` 与 `lhs` 将以取模加的方式移动，下图为一个例子：



利用 `wbase` 与 `widx` 的取模操作，`lhs` 与 `rhs` 在实际运行时只需要不断增加就可以了，同时利用 `senderWindowLength` 长度的窗口数组即可实现需要的功能。不过其中还有一些细节需要阐述，即每一次程序将丢弃序号小于 `lhs` 或大于 `rhs` 的 ACK；并且假如返回的 `window[ACK.seqNum]`为 `None`，直接忽略该冗余应答，否则将对应位置 `window[ACK.seqNum]`清空为 `None`，表示已经收到应答；同时，

一旦 `window[lhs % sendWindowLength] = None`, 需要向前移动 lhs, 没法送一个新的包, 就像前移动 rhs。

其次考虑**超时重传**的问题, 在没有收到 ACK 的时间段里, 程序需要不断检测计时器, 一旦发生超时(`time.time() - timestamp >= maxTime`), 就立即重传此时窗口中等待 ACK 的所有报文 (根据上述滑动窗口的设计, 所有需要重传的报文应当是 window 数组中非 None 的部分)。在每一次 lhs 向前移动或者重传之后, 就更新时间戳。为了实现这一功能, 需要增加数据成员 `self.roundTime = time.time()`

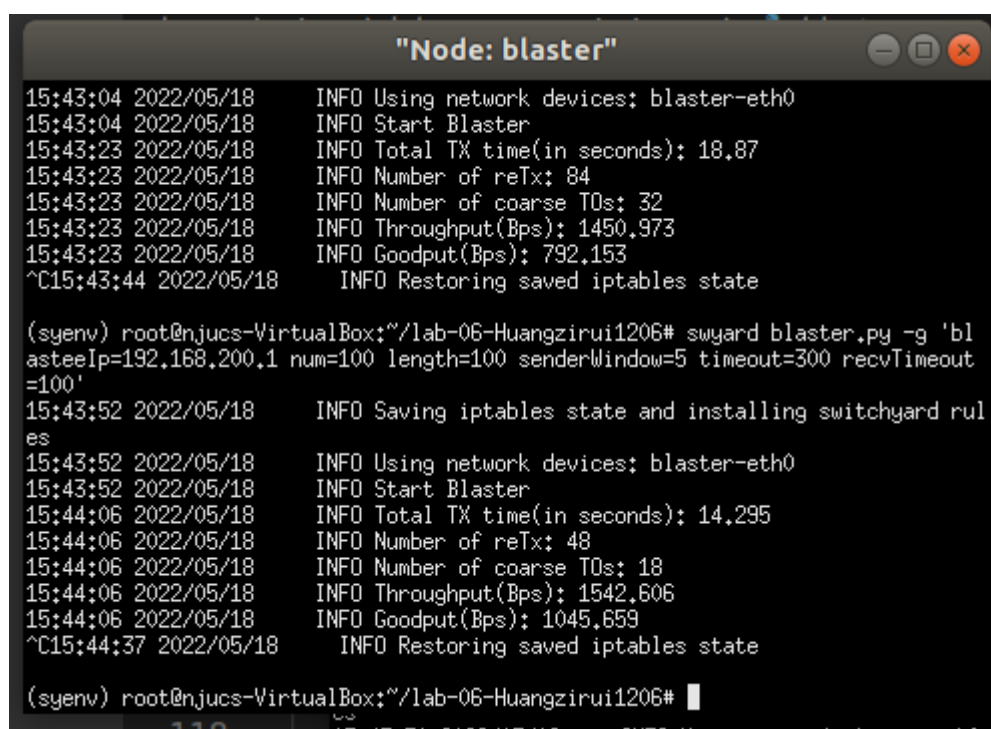
最后考虑程序结束时**状态打印**的问题。首先需要增加相关的数据成员, 如下所示:

```
1. # for printing stats
2. self.initTime = time.time()
3. self.reTXnum = 0
4. self.timeoutnum = 0
5. self.throughput = 0
6. self.goodput = 0
```

每一次发送一个新的包就需要在 goodput 上加 `pkt.size()`(注意到每一个包都是等长的); 而每发送一个包 (无论是否为重发) 就在 throughput 上加 `pkt.size()`; 每发送一个重传的包, 就在 reTXnum 上加 1; 而每次超时就在 timeoutnum 上加 1。最后将各个信息输出即可 (注意到总时间就是输出时刻的时间减去 initTime, 而 throughput 与 goodput 需要在输出时除以总时间)。

下面给出运行的截图：

首先是 xterm 下窗台输出的例子：



```
"Node: blaster"
15:43:04 2022/05/18 INFO Using network devices: blaster-eth0
15:43:04 2022/05/18 INFO Start Blaster
15:43:23 2022/05/18 INFO Total TX time(in seconds): 18.87
15:43:23 2022/05/18 INFO Number of reTx: 84
15:43:23 2022/05/18 INFO Number of coarse T0s: 32
15:43:23 2022/05/18 INFO Throughput(Bps): 1450.973
15:43:23 2022/05/18 INFO Goodput(Bps): 792.153
^C15:43:44 2022/05/18 INFO Restoring saved iptables state

(syenv) root@njucs-VirtualBox:~/lab-06-Huangzirui1206# swyard blaster.py -g 'bl
asteeIp=192.168.200.1 num=100 length=100 senderWindow=5 timeout=300 recvTimeout
=100'
15:43:52 2022/05/18 INFO Saving iptables state and installing switchyard rul
es
15:43:52 2022/05/18 INFO Using network devices: blaster-eth0
15:43:52 2022/05/18 INFO Start Blaster
15:44:06 2022/05/18 INFO Total TX time(in seconds): 14.295
15:44:06 2022/05/18 INFO Number of reTx: 48
15:44:06 2022/05/18 INFO Number of coarse T0s: 18
15:44:06 2022/05/18 INFO Throughput(Bps): 1542.606
15:44:06 2022/05/18 INFO Goodput(Bps): 1045.659
^C15:44:37 2022/05/18 INFO Restoring saved iptables state

(syenv) root@njucs-VirtualBox:~/lab-06-Huangzirui1206#
```

上图包含了两次状态输出（即两次运行的结果），此时输入对的参数即为手册中给出的参数，丢包率是 0.19。可以看到第一次运行中，发生了 32 次超时（即丢失了 32 个包），而第二次运行丢失了 18 个包（超时 18 次），可以明显看到后者的效率（主要是 goodput 一项）比前者高不少。同时重传的报文数基本是丢包数的两到三倍，这是因为当窗口中的第一个报文超时，后面的报文的 ACK 可能还没有及时发到，这会倒是冗余 ACK。

下面不修改 blaster 与 blastee 的参数，仅仅改变 middlebox 的丢包率，调整至 0.30，运行结果如下：

```
"Node: blaster"
15:59:32 2022/05/18 INFO Send a new packet
15:59:32 2022/05/18 INFO Send a new packet
15:59:32 2022/05/18 INFO Send a new packet
15:59:33 2022/05/18 INFO Send a new packet
15:59:33 2022/05/18 INFO Send a new packet
15:59:33 2022/05/18 INFO Send a new packet
15:59:34 2022/05/18 INFO Send a new packet
15:59:34 2022/05/18 INFO Send a new packet
15:59:34 2022/05/18 INFO Send a new packet
15:59:34 2022/05/18 INFO Send a new packet
15:59:34 2022/05/18 INFO Send a new packet
15:59:35 2022/05/18 INFO Send a new packet
15:59:35 2022/05/18 INFO Send a new packet
15:59:35 2022/05/18 INFO Send a new packet
15:59:35 2022/05/18 INFO Send a new packet
15:59:36 2022/05/18 INFO Send a new packet
15:59:36 2022/05/18 INFO Send a new packet
15:59:36 2022/05/18 INFO Send a new packet
15:59:36 2022/05/18 INFO Total TX time(in seconds): 23.57
15:59:36 2022/05/18 INFO Number of reTx: 150
15:59:36 2022/05/18 INFO Number of coarse T0s: 57
15:59:36 2022/05/18 INFO Throughput(Bps): 1588,618
15:59:36 2022/05/18 INFO Goodput(Bps): 646,75
```

比较一下两次不同的参数 (丢包率 0.19 与 0.30), 可以明显看到, throughput 的效率没有多大改变 (这是由 senderWindowLength, recvTimeout 与 timeout 决定的), 而 goodput 明显变小, 发生超时的次数也明显变多了。

下面考虑改变 senderWindowLength, recvTimeout 与 timeout 的值, 使 senderWindowLength = 7, 其余参数与手册上举的例子一致, 则运行结果如下:

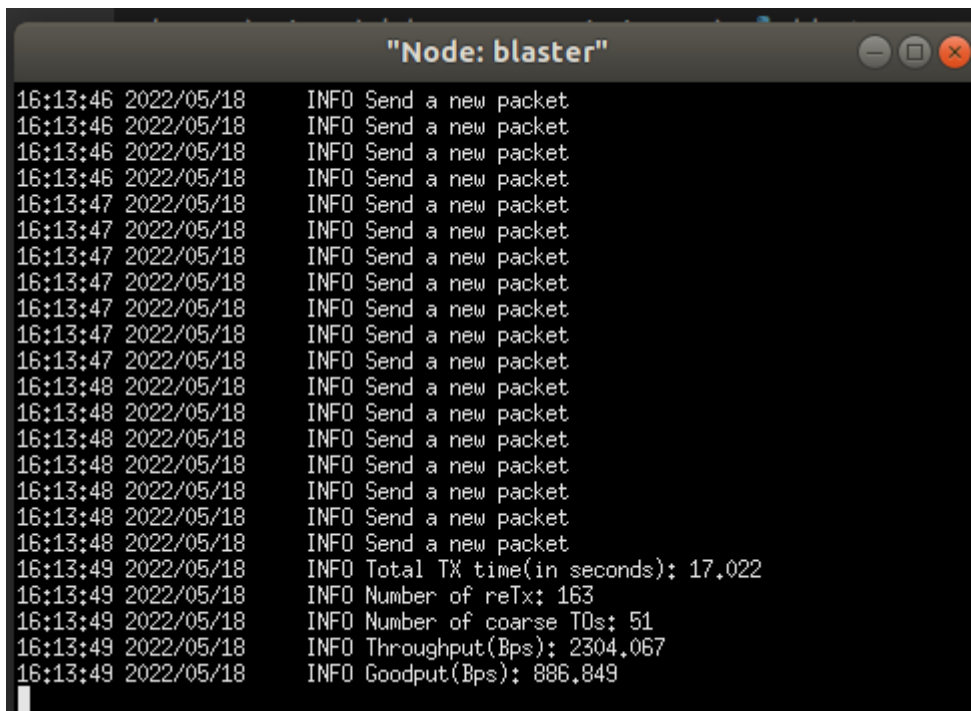
```
"Node: blaster"
16:05:51 2022/05/18 INFO Send a new packet
16:05:51 2022/05/18 INFO Send a new packet
16:05:51 2022/05/18 INFO Send a new packet
16:05:51 2022/05/18 INFO Send a new packet
16:05:52 2022/05/18 INFO Send a new packet
16:05:52 2022/05/18 INFO Send a new packet
16:05:52 2022/05/18 INFO Send a new packet
16:05:52 2022/05/18 INFO Send a new packet
16:05:52 2022/05/18 INFO Send a new packet
16:05:52 2022/05/18 INFO Send a new packet
16:05:52 2022/05/18 INFO Send a new packet
16:05:53 2022/05/18 INFO Send a new packet
16:05:53 2022/05/18 INFO Send a new packet
16:05:53 2022/05/18 INFO Send a new packet
16:05:53 2022/05/18 INFO Send a new packet
16:05:53 2022/05/18 INFO Send a new packet
16:05:53 2022/05/18 INFO Send a new packet
16:05:54 2022/05/18 INFO Send a new packet
16:05:54 2022/05/18 INFO Total TX time(in seconds): 16.877
16:05:54 2022/05/18 INFO Number of reTx: 86
16:05:54 2022/05/18 INFO Number of coarse T0s: 27
16:05:54 2022/05/18 INFO Throughput(Bps): 1657.398
16:05:54 2022/05/18 INFO Goodput(Bps): 903.238
```

与第一张图片里的运行结果相比, 可以看到通过改变 senderWindowLength, 运行时的 throughput 值明显增加了, 两者之间是正相关的关系, 即窗口越长, 总的发包效率就越高, 当然 goodput 的大小除了与此相关, 还与丢包率相关。

recvTimeout 代表函数处理的频率, 自然 recvTimeout 越小, 发包效率就越高, 这里就不额外贴图举例了。

下面将 timeout 的值从原来的 300 改为 150 或 500, 其余参数与手册用例一致, 分别给出两种情况的运行结果:

(timeout = 150)



```
16:13:46 2022/05/18 INFO Send a new packet
16:13:46 2022/05/18 INFO Send a new packet
16:13:46 2022/05/18 INFO Send a new packet
16:13:46 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:47 2022/05/18 INFO Send a new packet
16:13:48 2022/05/18 INFO Send a new packet
16:13:48 2022/05/18 INFO Send a new packet
16:13:48 2022/05/18 INFO Send a new packet
16:13:48 2022/05/18 INFO Send a new packet
16:13:48 2022/05/18 INFO Send a new packet
16:13:48 2022/05/18 INFO Send a new packet
16:13:48 2022/05/18 INFO Send a new packet
16:13:49 2022/05/18 INFO Total TX time(in seconds): 17.022
16:13:49 2022/05/18 INFO Number of reTx: 163
16:13:49 2022/05/18 INFO Number of coarse T0s: 51
16:13:49 2022/05/18 INFO Throughput(Bps): 2304.067
16:13:49 2022/05/18 INFO Goodput(Bps): 886.849
```

通过将上图与第一张运行结果比较, 不难发现, throughput 的值大幅提高了! 但是显然因为此时 timeout 比之 recvTimeout 太小, 出现了不必要的超时重传, 所以实际上 goodput 的值并没有明显的提升, 甚至还下降了。由此可见。需要合理设计 recvTimeout 与 timeout, 使两者处于合理的大小状态, 才能达到效率的最大化。

(timeout = 500)

```
"Node: blaster"
16:15:05 2022/05/18 INFO Send a new packet
16:15:06 2022/05/18 INFO Send a new packet
16:15:06 2022/05/18 INFO Send a new packet
16:15:06 2022/05/18 INFO Send a new packet
16:15:06 2022/05/18 INFO Send a new packet
16:15:07 2022/05/18 INFO Send a new packet
16:15:07 2022/05/18 INFO Send a new packet
16:15:07 2022/05/18 INFO Send a new packet
16:15:07 2022/05/18 INFO Send a new packet
16:15:07 2022/05/18 INFO Send a new packet
16:15:08 2022/05/18 INFO Send a new packet
16:15:08 2022/05/18 INFO Send a new packet
16:15:08 2022/05/18 INFO Send a new packet
16:15:10 2022/05/18 INFO Send a new packet
16:15:10 2022/05/18 INFO Send a new packet
16:15:10 2022/05/18 INFO Send a new packet
16:15:10 2022/05/18 INFO Send a new packet
16:15:10 2022/05/18 INFO Total TX time(in seconds): 21.63
16:15:10 2022/05/18 INFO Number of reTx: 46
16:15:10 2022/05/18 INFO Number of coarse T0s: 17
16:15:10 2022/05/18 INFO Throughput(Bps): 1019,524
16:15:10 2022/05/18 INFO Goodput(Bps): 704,771
```

比较可得，timeout 调整为 500 之后明显传输效率降低了，因为丢包之后，网络将等待过长的时间才会判定需要超时重传。

考虑一个最极端的情况，将丢包率调整为 0，给出这种理想情况下的运行情况（其余参数与手册用例一致）：

```
16:50:51 2022/05/18 INFO Total TX time(in seconds): 13,218
16:50:51 2022/05/18 INFO Number of reTx: 15
16:50:51 2022/05/18 INFO Number of coarse T0s: 5
16:50:51 2022/05/18 INFO Throughput(Bps): 1287,655
16:50:51 2022/05/18 INFO Goodput(Bps): 1119,7
```

可以看到尽管 middlebox 的丢包率是 0，仍然出现了重传的现象，这是因为 blastee 的处理速度太慢（recv = self.net.recv_packet(timeout=1.0)）而导致的超时现象，将 blaster 的 timeout 设置为 2000，再运行可得：

```
16:44:11 2022/05/18 INFO Send a new packet, lhs = 96, rhs = 99
16:44:11 2022/05/18 INFO Total TX time(in seconds): 14,386
16:44:11 2022/05/18 INFO Number of reTx: 0
16:44:11 2022/05/18 INFO Number of coarse T0s: 0
16:44:11 2022/05/18 INFO Throughput(Bps): 1028,788
16:44:11 2022/05/18 INFO Goodput(Bps): 1028,788
```

这样可以看出不再有超时与重传。

下面是 wireshark 对 blastee 的抓包截图:

