实验报告

计算机网络-Lab6

专业：计算机科学与技术

任课教师：田臣

周迅 201220037

目录

一、 实验目的

在 Switchyard 建立一个一个可靠的通信库，基于 3 个 agent，假设只会发生丢包这一种意外（不会有其他问题，例如数据的污染）。

具体地，我们需要实现一个蠢蠢的会丢包的路由器、一个接受包后发送 ACK 的 Blastee（我也不知道该翻译成啥），以及一个发送包、接受 ACK、判断是否发生丢包的 Blaster。

二、 实验结果

- 按照手册要求，完成了 Middlebox、Blastee 和 Blaster 的功能，具体实现思路可能与手册推荐版本略有差异；
- 在实现代码中加入适当的输出内容，便于观察传输过程；
- 使用 Wireshark 进行抓包，并分析包的内容，判断是否符合期望。

三、 实验内容

1. Preparation

```
git clone!
```

2. Middlebox

这个路由器可比实验 3-5 的简单多了。它只需要管好 3 件事：

1) 修改 MAC。根据目的地和发出的端口修改包的以太网头即可。

```
self.modify_packet(
    packet,
    self.net.interface_by_name("middlebox-eth0").ethaddr,
    self.blaster_mac
)
```

2) 随机性丢包。根据给定的丢包率随机丢包。PS：未设保底机制，如果出现奇葩情况，请不要惊讶。

```
def should_drop(self) -> bool:
    return random.random() < self.dropRate
```

3) 当然，正常情况下，还得把包送出去。

```
self.net.send_packet("middlebox-eth0", packet)
```

为了便于后期调试，我们的路由器也能够提取并查看包中的序列号（尽管这已经是越俎代庖）。

```
def get_seq(self, packet: Packet) -> int:
    '''
    Get the sequence number of the packet
    '''
    copy_packet = copy.deepcopy(packet)
    del copy_packet[Ethernet]
    del copy_packet[IPv4]
    del copy_packet[UDP]
    return int.from_bytes(copy_packet[0].to_bytes()[:4], byteorder='big')
```

3. Blastee

Blastee 的实现也并不难。大致思路是，每当收到一个包，提取序列号和负载（的前 8 字节），发送 ACK 确认。

由于双方的序列号均按大端方式排列，所以根据 blaster 发送包的格式直接取出来无需转换就可以用。

```
d headers -----> <----- Your packet header(raw bytes) ------> <-- Payload in raw bytes -
-------------------------------------------------------------------------------------
dr  |  UDP Hdr  | Sequence number(32 bits) | Length(16 bits) |   Variable length payload
-------------------------------------------------------------------------------------
```

```
seq_num = origin_packet[0].to_bytes()[:4]
payload = origin_packet[0].to_bytes()[6:]
if len(payload) < 8:
    payload += (bytes(b'\x23') * (8 - len(payload)))
```

两者统一通过 `RawPacketContents()`塞进 ACK 包里。

```
log_info(f"ACK packet: seq:{int.from_bytes(seq_num, byteorder='big')}")
return eth_header + ip_header + udp_header + RawPacketContents(seq_num + payload)
```

4. Blaster

对于 Blaster，我们略微修改一下手册推荐的定义。LHS 为下一个**期望**收到的 ACK 序号，RHS 为下一个**期望**发出的包的序号。所以，这种情况下，窗口长度为(`RHS - LHS`)。

同时，我们维护一个集合，记录 Blastee 已经收到的包的序号（即已经得到 ACK)。

```
# next packet expected(lhs) and to send(rhs)
self.lhs = self.rhs = 1
# set of seq numbers that have been acked
self.get_acked = set()
self.timer = None
```

对于收到的 ACK，我们从中取出序列号，加入收到的 ACK 集合 (由于集合的性质，重复加入不会导致重复成员)。

```
del packet[Ethernet]
del packet[IPv4]
del packet[UDP]
seq_num = int.from_bytes(packet[0].to_bytes()[:4], byteorder='big')
self.get_acked.add(seq_num)
```

如果需要的话，重置计时器并移动 LHS。

```
# move lhs, if needed
if self.lhs in self.get_acked:
    self.timer = time.time()
while self.lhs in self.get_acked:
    self.lhs += 1
```

对于（`recvtimeout` 超时）没有收到包的情况，首先做几点说明。

按照手册中说的：

- 当超时后，我们需要重新发送每个需要重发的包；
- 每轮 `while` 循环只能发送一个包；
- 整个窗口共享一个计时器。

由此，我们设计一个发送队列，存储等待发送的包。每次在循环中没有包收到时，就构造相应的新包和重发包进队列；每轮循环中，如果队列非空，就从队首取出一个包发出。队列中的成员应包含是否重发的标记，以便于统计。

```
# send a new packet
if self.rhs - self.lhs < self.sender_window and self.rhs <= self.packet_
    self.send_que.append((self.rhs, False))
# resend
if self.timer is not None and time.time() - self.timer > self.timeout:
    for i in range(self.lhs, self.rhs):
        if i not in self.get_acked and i not in self.send_que:
            self.send_que.append((i, True))
```

```
def handle_send_que(self):
    while len(self.send_que) > 0:
        seq, resend_flag = self.send_que.popleft()
        if seq in self.get_acked:
            continue
        if resend_flag:
            self.resend_num += 1
            log_info(f"resend pkg {seq}")
        else:
            self.rhs += 1
            log_info(f"send pkg {seq}")
        self.net.send_packet(
            self.net.interfaces()[0].name,
            self.make_packet(seq)
        )
        break
```

对于统计信息，在这里只说明一点。对于超时部分，由于超时重发不会重置计时器，所以我们记录上次超时的 `timer`（即上一次移动 LHS 的时间），以区分某次超时判断是否已经做过。这里，我们搞点事儿，套个哈希函数。

更新：最新版中，我们重置计时器。

```
if hash(self.timer) != self.timeout_flag:
    self.timeout_num += 1
    self.timeout_flag = hash(self.timer)
```

5. Running your code

首先，定义本次测试参数：包总数 15，负载长度 100，负载内容’Hello,world!Hello,world!He……’，发送窗口长度 3，超时限制 3100ms，收包超时限制 1000ms，路由器丢包率 0.3，IP 和 MAC 设定参照 `start_mininet.py` 的规定。

首先来看路由器的丢包情况。由图，共丢了 4 个包。

```
06:14:55 2022/04/12     INFO Dropping: seq:1
06:14:56 2022/04/12     INFO Sending to blastee: seq:2
06:14:56 2022/04/12     INFO Sending to blaster: seq:2
06:14:57 2022/04/12     INFO Sending to blastee: seq:3
06:14:57 2022/04/12     INFO Sending to blaster: seq:3
06:14:58 2022/04/12     INFO Sending to blastee: seq:1
06:14:58 2022/04/12     INFO Sending to blaster: seq:1
06:14:59 2022/04/12     INFO Sending to blastee: seq:4
06:14:59 2022/04/12     INFO Sending to blaster: seq:4
06:15:00 2022/04/12     INFO Dropping: seq:5
06:15:01 2022/04/12     INFO Sending to blastee: seq:6
06:15:01 2022/04/12     INFO Sending to blaster: seq:6
06:15:02 2022/04/12     INFO Dropping: seq:7
06:15:03 2022/04/12     INFO Sending to blastee: seq:5
06:15:04 2022/04/12     INFO Sending to blaster: seq:5
06:15:04 2022/04/12     INFO Sending to blastee: seq:7
06:15:04 2022/04/12     INFO Sending to blaster: seq:7
06:15:05 2022/04/12     INFO Sending to blastee: seq:8
06:15:05 2022/04/12     INFO Sending to blaster: seq:8
06:15:06 2022/04/12     INFO Sending to blastee: seq:9
06:15:06 2022/04/12     INFO Sending to blaster: seq:9
06:15:08 2022/04/12     INFO Dropping: seq:10
06:15:09 2022/04/12     INFO Sending to blastee: seq:11
06:15:09 2022/04/12     INFO Sending to blaster: seq:11
06:15:10 2022/04/12     INFO Sending to blastee: seq:12
06:15:10 2022/04/12     INFO Sending to blaster: seq:12
06:15:10 2022/04/12     INFO Sending to blastee: seq:10
06:15:10 2022/04/12     INFO Sending to blaster: seq:10
06:15:12 2022/04/12     INFO Sending to blastee: seq:13
06:15:12 2022/04/12     INFO Sending to blaster: seq:13
06:15:13 2022/04/12     INFO Sending to blastee: seq:14
06:15:13 2022/04/12     INFO Sending to blaster: seq:14
06:15:14 2022/04/12     INFO Sending to blastee: seq:15
06:15:15 2022/04/12     INFO Sending to blaster: seq:15
```

接着，看 Blaster 的处理。考虑到丢包重发的步骤都差不多，我们挑丢包 5 和 7 的那部分重点关注。

```
06:14:59 2022/04/12     INFO get ack 4, lhs is 5, rhs is 5
06:15:00 2022/04/12     INFO no pkg, lhs is 5, rhs is 5
06:15:00 2022/04/12     INFO send pkg 5
06:15:01 2022/04/12     INFO no pkg, lhs is 5, rhs is 6
06:15:01 2022/04/12     INFO send pkg 6
06:15:01 2022/04/12     INFO get ack 6, lhs is 5, rhs is 7
06:15:02 2022/04/12     INFO no pkg, lhs is 5, rhs is 7
06:15:02 2022/04/12     INFO send pkg 7
06:15:03 2022/04/12     INFO no pkg, lhs is 5, rhs is 8
06:15:03 2022/04/12     INFO resend pkg 5
06:15:04 2022/04/12     INFO get ack 5, lhs is 7, rhs is 8
06:15:04 2022/04/12     INFO resend pkg 7
06:15:04 2022/04/12     INFO get ack 7, lhs is 8, rhs is 8
```

由以上图，我们可以得出如下结论：

1) 初始状态，LHS = RHS = 5；

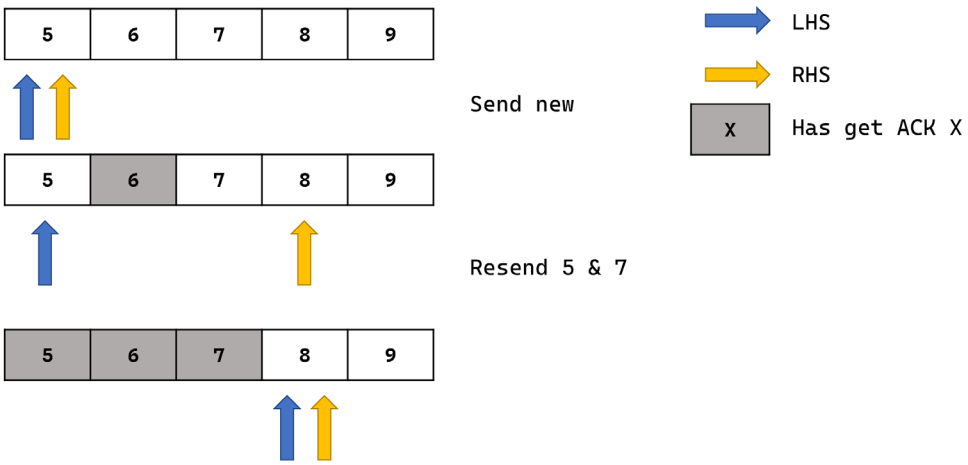
2) 5 丢包后，由于没有超时，继续发送 6；

3) 7 丢包后，首先，窗口长度已经到达 3，不能再有新包进入等待队列，其次，此时已经超时，5 和 7 进入发包等待队列，两者的超时事件按同一个来计；

4) 重发 5 和 7，得到回复；

5) 由于其余的包没有丢包，所以 LHS 移动至 8。

做成生动形象的示意图如下：

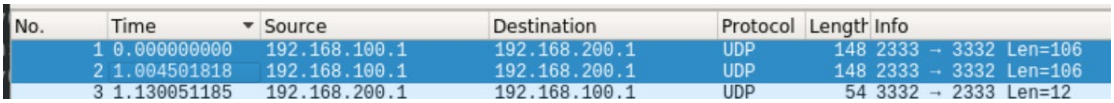


接着，贴一下 Blaster 的感想（统计）。由于上述分析中，我们已经了解到，5 和 7 的丢包属于同一超时事件，故重发次数比超时次数大 1。

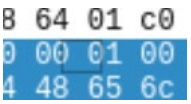
```
Total TX time: 20.591565132141113 seconds
Total resend times: 4
Total timeout times: 3
Throughput: 92.27 Bps
Goodput: 72.85 Bps
```

下面，我们来看一下抓包文件（已经重新加载场景，不是上面的环境）。关注 middle-eth0，即与 Blaster 相连的端口。

关注如下两图，均为 Blaster 发往 Blastee 的包。其中，上面那个包，经分析，序列号为 1。由于丢包，并没有得到回复。

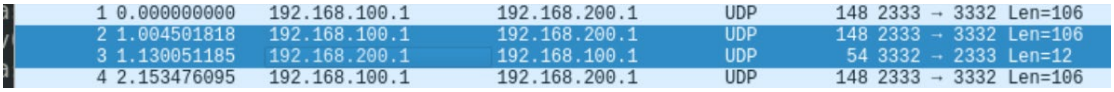


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148	2333 → 3332 Len=106
2	1.004501818	192.168.100.1	192.168.200.1	UDP	148	2333 → 3332 Len=106
3	1.130051185	192.168.200.1	192.168.100.1	UDP	54	3332 → 2333 Len=12



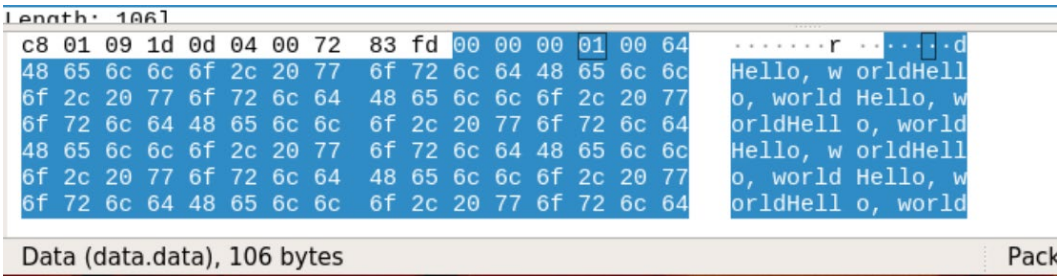
```
8 64 01 c0
0 00 01 00
4 48 65 6c
```

而序列号为 2 的就得到了回复。



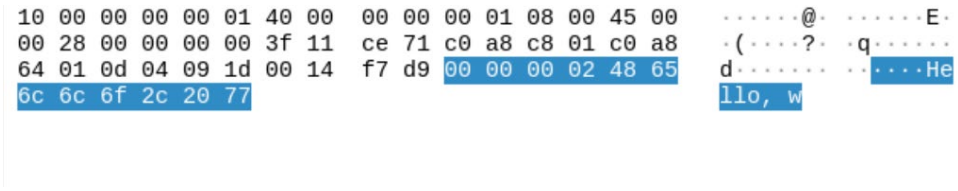
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148	2333 → 3332 Len=106
2	1.004501818	192.168.100.1	192.168.200.1	UDP	148	2333 → 3332 Len=106
3	1.130051185	192.168.200.1	192.168.100.1	UDP	54	3332 → 2333 Len=12
4	2.153476095	192.168.100.1	192.168.200.1	UDP	148	2333 → 3332 Len=106

下面我们检查一下包的内容（除去 3 个头）。仍以 Blaster 发往 Blastee 的序列号为 1 的包为例。前 4 个字节为序列号 1，接着两字节为负载长度 0x64=100。两者均按大端方式存放。后面跟着的就是负载（UTF-8 编码）。这些加起来总长为 106 字节，符合预期。



```
Length: 106]
c8 01 09 1d 0d 04 00 72  83 fd 00 00 00 01 00 64   ·······r ·····d
48 65 6c 6c 6f 2c 20 77  6f 72 6c 64 48 65 6c 6c   Hello, w orldHell
6f 2c 20 77 6f 72 6c 64  48 65 6c 6c 6f 2c 20 77   o, world Hello, w
6f 72 6c 64 48 65 6c 6c  6f 2c 20 77 6f 72 6c 64   orldHell o, world
48 65 6c 6c 6f 2c 20 77  6f 72 6c 64 48 65 6c 6c   Hello, w orldHell
6f 2c 20 77 6f 72 6c 64  48 65 6c 6c 6f 2c 20 77   o, world Hello, w
6f 72 6c 64 48 65 6c 6c  6f 2c 20 77 6f 72 6c 64   orldHell o, world
Data (data.data), 106 bytes                        Pack
```

类似地，ACK 报文由长度和负载组成，总长为 12 字节。



```
10 00 00 00 00 01 40 00  00 00 00 01 08 00 45 00   ·····@· ·····E·
00 28 00 00 00 00 3f 11  ce 71 c0 a8 c8 01 c0 a8   ·(····?· ·q·····
64 01 0d 04 09 1d 00 14  f7 d9 00 00 00 02 48 65   d····· ····He
6c 6c 6f 2c 20 77                                  llo, w
```

四、 核心代码

要不您往上边翻一翻，都有截图；或者，上 Github 看看代码？

五、 总结与感想

- 实验思路清晰，主要难点在于细节的控制，例如 LHS 含义、发包规则等等；
- 实现可靠传输的代价并不小，光是本实验的 ACK 就要维护发送窗口、计时器等等，现实中 TCP 的机制更是复杂；
- 做网络相关工作，需要考虑到现实状态下的最坏情况，提高系统的稳定性。