

#

计网实验报告

姓名	学号
施赢凯	211220025

一.实验名称

Reliable Communication

二.实验目的:

模拟TCP协议构建一个简单的可靠传输模型。通过ACK回复和超时重传机制保证接收方能够收到所有的数据包。

三.实验内容:

1.Middlebox的实现:

1.如果我们接收到的是来自blaster的话,存在着随机丢弃的机制,所以我们采用python的随机函数, `randnum = randint(1, 100)`,再将其除以100判断是否小于等于`dropRate`判断其是否为需要丢弃的包,如果不是,则将其通过middlebox-eth1发出,并将以太网头的src改成middlebox发出的接口的以太网地址, dst为blastee的以太网地址,否则直接丢弃不做处理,代码如下

code:

```
if fromIface == "middlebox-eth0":
    log_debug("Received from blaster")
    ...

    Received data packet
    Should I drop it?
    If not, modify headers & send to blastee
    ...

    randnum = randint(1, 100)
    if(float(randnum)/100<=self.dropRate):
        log_info("drop the packet")
    else:
        for intf in self.interfaces :
            if intf.name=="middlebox-eth1":
                my_port=intf
                packet[Ethernet].src=my_port.ethaddr
                packet[Ethernet].dst="20:00:00:00:00:01"
                self.net.send_packet("middlebox-eth1", packet)
```

2.如果是从blastee发出的，则直接将其发送给blaster，同样的，我们将包的以太网地址src改为"middlebox-eth0",dst改为blaster的以太网地址，代码如下：

code:

```
elif fromIface == "middlebox-eth1":
    log_debug("Received from blastee")
    for intf in self.interfaces :
        if intf.name=="middlebox-eth0":
            my_port=intf
            packet[Ethernet].src= my_port.ethaddr
            packet[Ethernet].dst="10:00:00:00:00:01"
            self.net.send_packet("middlebox-eth0", packet)
```

2.Blastee的实现

blastee在收到来自blaster发出的包后，需要构建回复包，按照实验手册上的逻辑，以太网头的src和dst分别为blastee和middlebox-eth1的以太网地址，IPv4头中的src和dst分别为blastee和给出的blasterIP，然后UDP头内容可以随意添加，之后我们需要的是序列号，即数据包中的自定义头的前四个自己，可以用packet[3].to_bytes()[0:4]获取，由于回复包固定的payload长度为8个字节，所以我们需要判断后续是否满足为8个字节，如果小于等于8个字节，我们需要在后续给他填上0补全，否则直接截取前八个字节，然后从blastee中的接口blastee-eth0发出。代码如下

code:

```
mypkt = Ethernet() + IPv4(protocol=IPProtocol.UDP) + UDP()
mypkt[0].src=self.blasteeEthaddr
mypkt[0].dst="40:00:00:00:00:02"
mypkt[0].ethertype = EtherType.IPv4
mypkt[1].dst=self.blasterIP
mypkt[1].src=self.blasteeIpaddr
mypkt[1].ttl=64
mypkt[1].protocol=IPProtocol.UDP
mypkt[2].src=4444
mypkt[2].dst=5555

#sequence = packet[3].to_bytes()[0:4]
len_pay=int.from_bytes(packet[3].to_bytes()[4:6], 'big')
if len_pay>=8:
    payload=packet[3].to_bytes()[6:14]
else:
    payload=packet[3].to_bytes()[6:]+(0).to_bytes(8-len_pay,"big")

mypkt+= packet[3].to_bytes()[0:4]
mypkt+=payload
```

```
self.net.send_packet( "blastee-eth0",mypkt)
```

Blaster的实现:

为了满足自己后续的实现需要，我定义了这几个参数，其余的都是固定的不展示

```
self.timecheck=time.time()
self.first=time.time()
self.final=time.time()
self.ACKd=[0]*( int(num)+1)
self.send_list=[0]*( int(num)+1)
self.recent=0
self.ackd=0
self.state=0
```

其中 `timecheck` 用于表示最近一次移动LHS的时间，`first`和`final`表示接收到第一个包和处理完最后一个包的时间，用于计算总共处理时间，然后`ACKd`是一个队列，用于判断其是否成功传递以及包的序列号，`send_list`是用于判断该是处于发送还是未发送或者是重传状态，来做相应的处理，`ackd`是用于判断总共成功发出的包序列号，用于判断什么时候结束发送，`state`是用于判断其处于重传还是发送包的时候。`recent`表示为当前重传过程中是从哪个LHS开始的

数据包的构建

我们需要获取当前对应的序列号以及荷载的长度，然后添加payload，这里为了简单，我直接就全部设为0。以太网头的src则是显然`blaster`的mac地址，然后dst显然为`middle-eth0`,然后IPv4头则是`blaster`的IP，dst则为BlasteeIP，代码如下：

```
def make_pkt(self,sequence):
    my_pkt = Ethernet() + IPv4() + UDP()
    my_pkt[1].protocol = IPProtocol.UDP
    my_pkt[1].ttl =64
    my_pkt[0].ethertype = EtherType.IPv4
    my_pkt[0].src = "10:00:00:00:00:01"
    my_pkt[0].dst = "40:00:00:00:00:01"
    my_pkt[1].src = IPv4Address("192.168.100.1")
    my_pkt[1].dst = self.blasteeIPAddr
    my_pkt+=sequence.to_bytes(4,"big")
    my_pkt+=self.Length.to_bytes(2,"big")
    my_pkt+=(0).to_bytes(self.Length,'big')
    return my_pkt
```

处理blastee发送的ACK包

当我们接收到blastee发送过来的包时，我们可以将对应序列的包的`send_list[i]`改为1，此处是我一开始卡住比较久的地方，若无此判断，则会导致重传率比预期大不少因为每次只发送一个包，防止其被重传。我们需要判断当前包对应序列号是否为已经成功接受，不是则设定为接收即`ACKd[i]=1`，并且将`ackd+1`，表示成功发送的包的数量。这时候我们就开始将LHS右移，直到遇到`ACKd[i]=0`，代码如下所示。

```
sequence = int.from_bytes(packet[3].to_bytes()[:4], 'big')
self.send_list[sequence]=1
if self.ACKd[sequence]==0:#repeated packet
    self.ACKd[sequence]=1
    self.ackd+=1
    print("ackd",self.ackd)
tmp4=self.LHS
while tmp4<self.RHS:
    if self.ACKd[tmp4]==1:
        self.LHS+=1
        self.timecheck=time.time()
    elif self.ACKd[tmp4]==0:
        break
    tmp4+=1
log_debug("I got a packet")
```

发送数据包流程

首先，我们知道，发送包有一个固定的发送窗口长度，所以我们一次发送的包不能超过这个长度，按照手册的要求，我们可以使用LHS和RHS来设定，并且不能超过最大长度num：

```
if (self.RHS - self.LHS < self.SW) and (self.RHS<=self.Num):
    self.RHS=min(self.LHS+self.SW,self.Num+1)
```

在这次实验的常见问题中我们可以知道这次每次rcv中最好只进行一次包的发送，所以我们需要用一个参数`state`来判断当前是重传包还是发送包，当`state=1`表示其为重传过程，为0则表明其为发送过程，只需要发包即可 因为超时我们就需要进行重传，所以我们判断当前是否超时，如果超时，则当前的LHS到RHS之间的包如果是已经发送并且未接受到回复的则需要重传，即将 `send_list[i]=2`，并且重传的次数加1，并且将当前的状态设定为重传状态：

```
if(time.time()-self.timecheck>self.timeout) and self.state==0:
    self.Number_of_coarse_TOs+=1
    print("need to recent ",self.Number_of_coarse_TOs)
    self.recent=self.LHS
    self.state=1
```

```

for i in range(self.LHS,self.RHS):
    if self.send_list[i]==1 and self.ACKd[i]==0:
        self.send_list[i]=2

```

判断state, 如果当前处于的是发送状态, 此时我们需要在LHS到RHS之间进行遍历, 找到未发送的包, 即 `send_list[i]=0` 进行发送, 并且将当前Throughput和Goodput加上当前的包的可变长长度length, 并且将其设定为已经发送, 然后我们需要构建包将其发送给blaster, 代码如下:

```

if self.state==0:
    tmp1=self.LHS
    while tmp1<self.RHS:
        if self.send_list[tmp1]==0:
            self.send_list[tmp1]=1
            self.Goodput+=self.Length
            self.Throughput+=self.Length
            mypkt=self.make_pkt(tmp1)
            self.net.send_packet("blaster-eth0", mypkt)
            if tmp1==1:
                self.start=time.time()
            break
        tmp1+=1

```

如果当前处于重传状态, 我们需要在LHS到RHS之间寻找需要重传的包即 `send_list[i]=2`, 然后重新构造包重传, 我们将 `throughput` 增加上当前重传的长度, 然后我们可以将其 `send_list` 中的值改回1, 并且只要找到一个需要重传的包重传结束后直接结束, 这是因为因为每次只能发送一个包, 所以我们需要判断是否所有的重传结束, 在我这里使用 `check point` 这个布尔数表示, 如果完全重传结束, 即 `check point=TRUE` 就将状态从重传改回发送态, 可以将当前重传开始的数字+1使得下次可以直接从当前重传完成的之后开始来减少计算量, 代码如下:

```

elif self.state==1:

    checkpoint=True# make sure all have been resend
    tmp3=self.recent
    while tmp3<self.RHS:
        if self.send_list[tmp3]==2:
            mypkt=self.make_pkt(tmp3)
            self.Number_of_reTX+=1
            self.Throughput+=self.Length
            self.net.send_packet("blaster-eth0", mypkt)
            self.recent=tmp3+1
            self.send_list[tmp3]=1
            checkpoint=False
        break

```

```

        tmp3+=1
    if checkpoint==True:
        self.state=0

```

打印列表以及修改条件

我们将循环的条件改为当前`ackd` 小于`Num`,来判断是否所有的包都被成功发送,之后我们就可以将当前的`recvtimeout`修改,当所有包发送完成我们就可以更新结束的时间然后打印,代码如下。

```

def print_list(self):
    self.Total_TX_time=self.final-self.start

    print("-----printing list-----")
    print("Total_TX_time:", self.Total_TX_time)
    print("Number_of_reTX:", self.Number_of_reTX)
    print("Number_of_coarse_TOs:", self.Number_of_coarse_TOs)
    print("Throughput:", self.Throughput/self.Total_TX_time)
    print("Goodput:", self.Goodput/self.Total_TX_time)
    print("-----")

def start(self):
    '''A running daemon of the blaster.
    Receive packets until the end of time.
    ...
    while self.ackd<self.Num:
        try:
            recv = self.net.recv_packet(timeout=self.recvTimeout)
        except NoPackets:
            self.handle_no_packet()
            continue
        except Shutdown:
            break

        self.handle_packet(recv)
    self.final=time.time()
    self.print_list()
    self.shutdown()

def shutdown(self):
    self.net.shutdown()

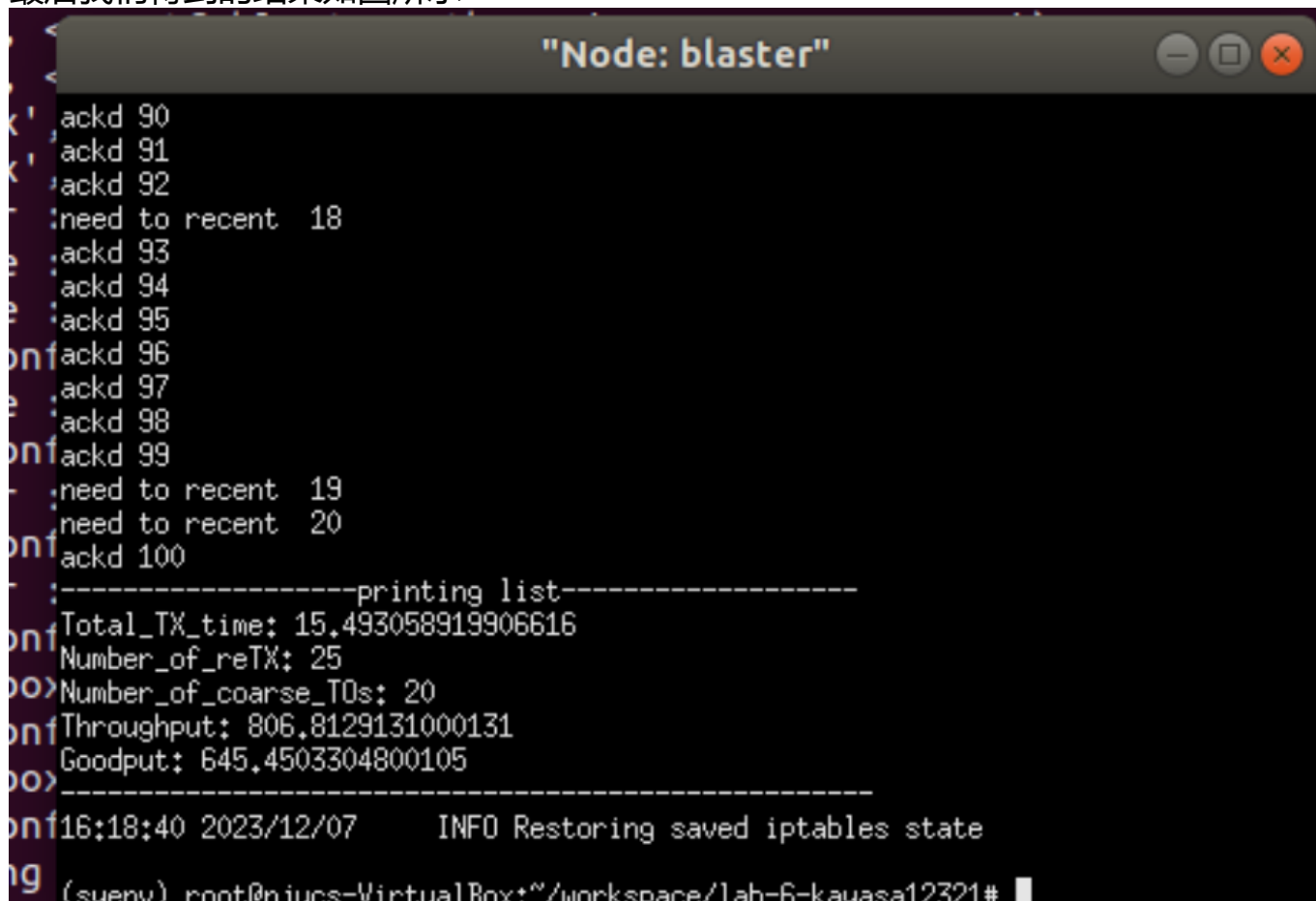
```

对答案的验证

当我们按照结果按照实验手册所给的参数进行测试即

```
middlebox# swyard middlebox.py -g 'dropRate=0.19'
blastee# swyard blastee.py -g 'blasterIp=192.168.100.1 num=100'
blaster# swyard blaster.py -g 'blasteeIp=192.168.200.1 num=100 length=100
senderWindow=5 timeout=300 recvTimeout=100'
```

最后我们得到的结果如图所示



```
Node: blaster
ackd 90
ackd 91
ackd 92
:need to recent 18
ackd 93
ackd 94
ackd 95
ackd 96
ackd 97
ackd 98
ackd 99
:need to recent 19
:need to recent 20
ackd 100
:-----printing list-----
Total_TX_time: 15.493058919906616
Number_of_reTX: 25
Number_of_coarse_T0s: 20
Throughput: 806.8129131000131
Goodput: 645.4503304800105
:-----
16:18:40 2023/12/07 INFO Restoring saved iptables state
(suenv) root@ninux-VirtualBox:~/workspace/lab-6-kauasa12321#
```

最终重传的次数为20，基本上和我们所需要的droprate=0.19相符合

这时候我们可以尝试去修改droprate，不妨令droprate=0，如果最后结果正确，其丢包率应该为0，根据结果来看是正确的

```
"Node: blaster"
ackd 87
ackd 88
ackd 89
ackd 90
ackd 91
ackd 92
ackd 93
ackd 94
ackd 95
ackd 96
ackd 97
ackd 98
ackd 99
ackd 100
-----printing list-----
Total_TX_time: 10.713115453720093
Number_of_reTX: 0
Number_of_coarse_T0s: 0
Throughput: 933.4352871673323
Goodput: 933.4352871673323
-----
16:27:44 2023/12/07      INFO Restoring saved iptables state
(syenv) root@njucs-VirtualBox:~/workspace/lab-6-kayasa12321#
```