Computer Network lab6 实验报告

姓名 曾闻天

学号 231880147

邮箱 2249428377@qq.com

完成日期 2025.4.22

Computer Network lab6 实验报告

1 实验名称

Lab 6: Reliable Communication

2 实验目的

1. 在接收方实现 ACK 机制.
2. 在发送方实现滑动窗口、超时重传和 ACK 处理机制.
3. 通过中间节点(Middlebox)模拟丢包并转发数据包.
4. 理解可靠通信协议的核心原理(如 GBN 协议).

3 实验进行

3.1 Rreparation

网络拓扑结构如下:

Blaster(发送方)

↓

Middlebox(中间节点)

↓

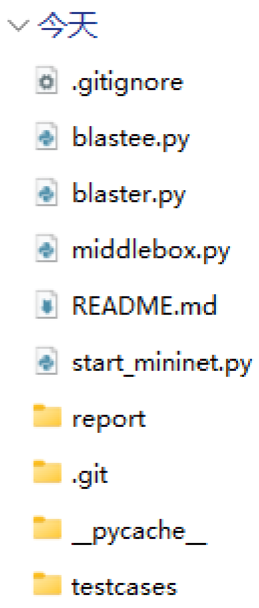
Blastee(接收方)

Blaster: 生成数据包并按滑动窗口发送.

Middlebox: 模拟丢包(仅对数据包),转发 ACK 包.

Blastee: 接收数据包并返回 ACK.

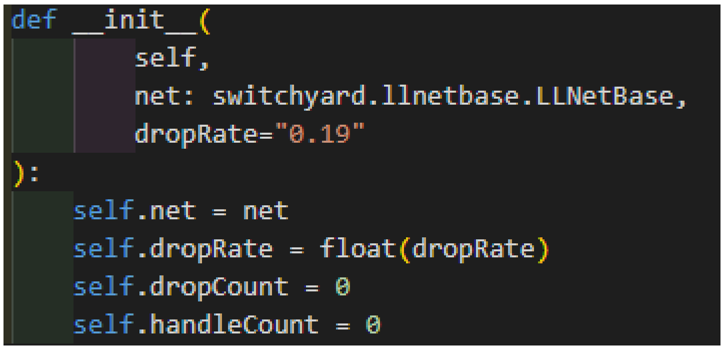
文件结构见下图:



```
今天
  .gitignore
  blastee.py
  blaster.py
  middlebox.py
  README.md
  start_mininet.py
  report
  .git
  __pycache__
  testcases
```

3.2 Middlebox

1.初始化参数:

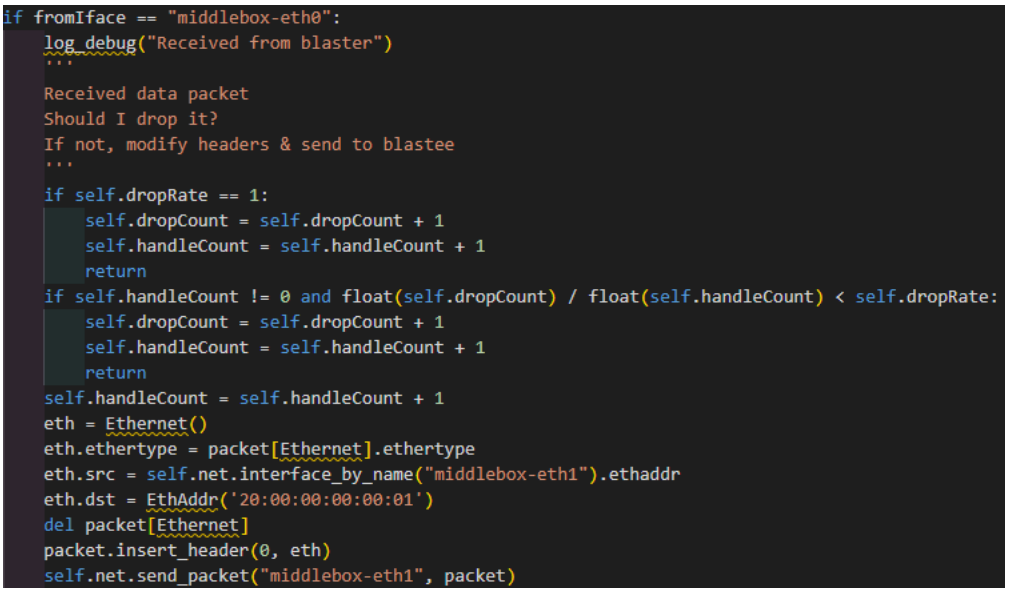


```python
def __init__(
        self,
        net: switchyard.llnetbase.LLNetBase,
        dropRate="0.19"
):
    self.net = net
    self.dropRate = float(dropRate)
    self.dropCount = 0
    self.handleCount = 0
```

2.处理来自 Blaster 的数据包:

根据丢包率随机丢弃数据包(不处理 ACK 包).

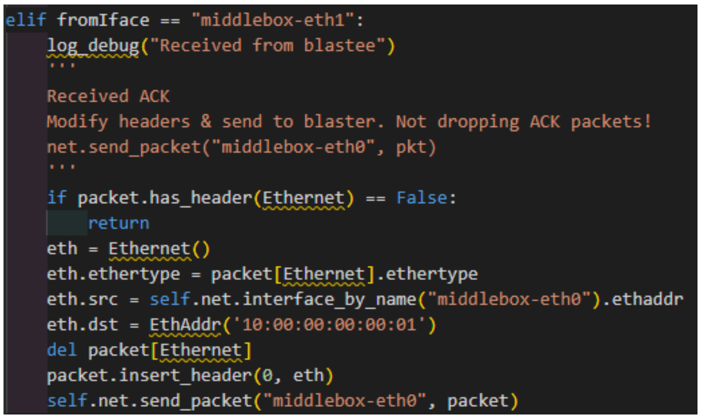
修改 MAC 地址并转发至 Blastee:



```
if fromIface == "middlebox-eth0":
    log_debug("Received from blaster")
    '''
    Received data packet
    Should I drop it?
    If not, modify headers & send to blastee
    '''
    if self.dropRate == 1:
        self.dropCount = self.dropCount + 1
        self.handleCount = self.handleCount + 1
        return
    if self.handleCount != 0 and float(self.dropCount) / float(self.handleCount) < self.dropRate:
        self.dropCount = self.dropCount + 1
        self.handleCount = self.handleCount + 1
        return
    self.handleCount = self.handleCount + 1
    eth = Ethernet()
    eth.ethertype = packet[Ethernet].ethertype
    eth.src = self.net.interface_by_name("middlebox-eth1").ethaddr
    eth.dst = EthAddr('20:00:00:00:00:01')
    del packet[Ethernet]
    packet.insert_header(0, eth)
    self.net.send_packet("middlebox-eth1", packet)
```

3. 处理来自 Blastee 的 ACK 包:

直接修改 MAC 地址并转发至 Blaster:

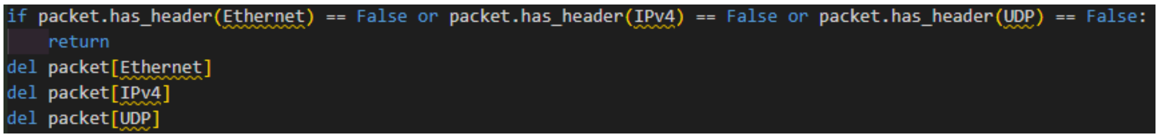


```
elif fromIface == "middlebox-eth1":
    log_debug("Received from blastee")
    '''
    Received ACK
    Modify headers & send to blaster. Not dropping ACK packets!
    net.send_packet("middlebox-eth0", pkt)
    '''
    if packet.has_header(Ethernet) == False:
        return
    eth = Ethernet()
    eth.ethertype = packet[Ethernet].ethertype
    eth.src = self.net.interface_by_name("middlebox-eth0").ethaddr
    eth.dst = EthAddr('10:00:00:00:00:01')
    del packet[Ethernet]
    packet.insert_header(0, eth)
    self.net.send_packet("middlebox-eth0", packet)
```

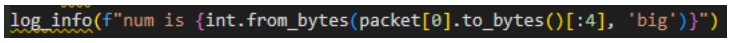
3.3 Blastee

1.解析数据包:

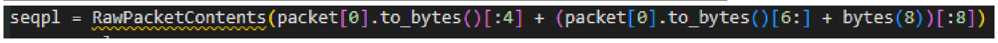
提取序列号(seq)和负载(payload):



```
if packet.has_header(Ethernet) == False or packet.has_header(IPv4) == False or packet.has_header(UDP) == False:
    return
del packet[Ethernet]
del packet[IPv4]
del packet[UDP]
```

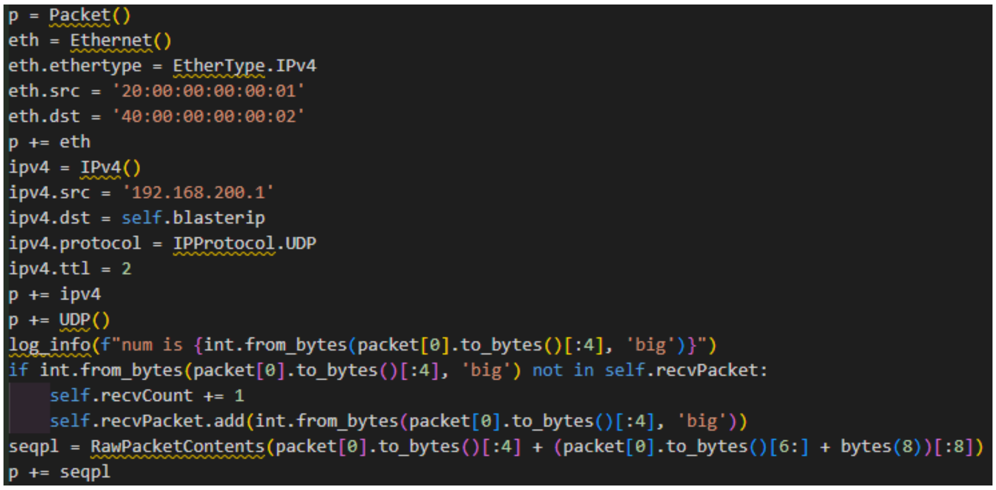


```
log_info(f"num is {int.from_bytes(packet[0].to_bytes()[:4], 'big')}")
```



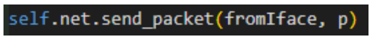
```
seqpl = RawPacketContents(packet[0].to_bytes()[:4] + (packet[0].to_bytes()[6:] + bytes(8))[:8])
```

2.生成 ACK 包:



```python
p = Packet()
eth = Ethernet()
eth.ethertype = EtherType.IPv4
eth.src = '20:00:00:00:00:01'
eth.dst = '40:00:00:00:00:02'
p += eth
ipv4 = IPv4()
ipv4.src = '192.168.200.1'
ipv4.dst = self.blasterip
ipv4.protocol = IPProtocol.UDP
ipv4.ttl = 2
p += ipv4
p += UDP()
log_info(f"num is {int.from_bytes(packet[0].to_bytes()[:4], 'big')}")
if int.from_bytes(packet[0].to_bytes()[:4], 'big') not in self.recvPacket:
    self.recvCount += 1
    self.recvPacket.add(int.from_bytes(packet[0].to_bytes()[:4], 'big'))
seqpl = RawPacketContents(packet[0].to_bytes()[:4] + (packet[0].to_bytes()[6:] + bytes(8))[:8])
p += seqpl
```

3.发送 ACK:

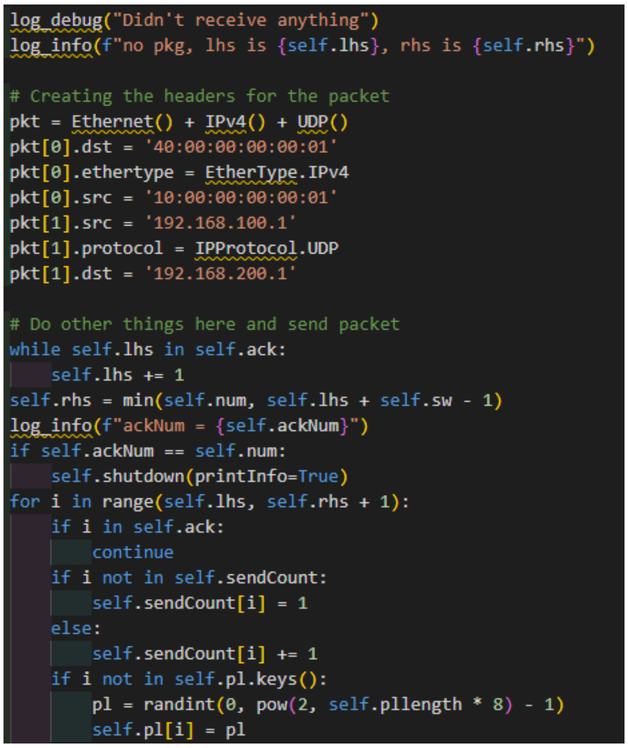


```python
self.net.send_packet(fromIface, p)
```

3.4 Blaster

1.滑动窗口控制:

窗口大小由参数 senderWindow 定义,动态调整 LHS 和 RHS:



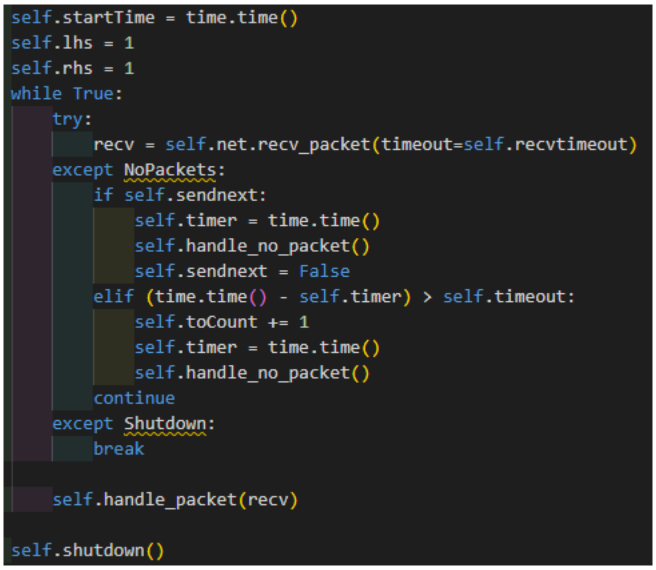
```python
log_debug("Didn't receive anything")
log_info(f"no pkg, lhs is {self.lhs}, rhs is {self.rhs}")

# Creating the headers for the packet
pkt = Ethernet() + IPv4() + UDP()
pkt[0].dst = '40:00:00:00:00:01'
pkt[0].ethertype = EtherType.IPv4
pkt[0].src = '10:00:00:00:00:01'
pkt[1].src = '192.168.100.1'
pkt[1].protocol = IPProtocol.UDP
pkt[1].dst = '192.168.200.1'

# Do other things here and send packet
while self.lhs in self.ack:
    self.lhs += 1
self.rhs = min(self.num, self.lhs + self.sw - 1)
log_info(f"ackNum = {self.ackNum}")
if self.ackNum == self.num:
    self.shutdown(printInfo=True)
for i in range(self.lhs, self.rhs + 1):
    if i in self.ack:
        continue
    if i not in self.sendCount:
        self.sendCount[i] = 1
    else:
        self.sendCount[i] += 1
    if i not in self.pl.keys():
        pl = randint(0, pow(2, self.pllength * 8) - 1)
        self.pl[i] = pl
```

2.超时重传:

若超时未收到 ACK，触发重传:



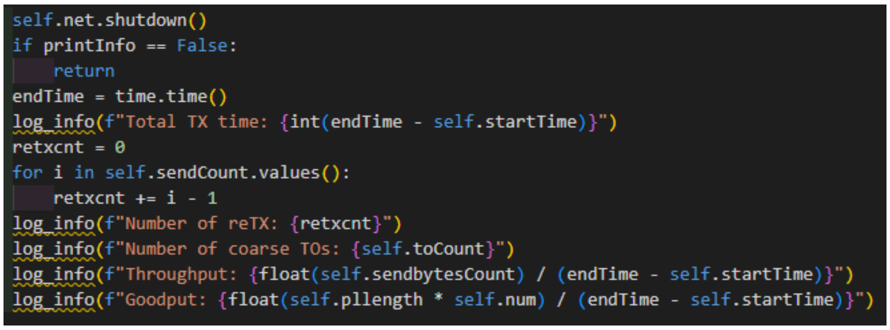
```python
self.startTime = time.time()
self.lhs = 1
self.rhs = 1
while True:
    try:
        recv = self.net.recv_packet(timeout=self.recvtimeout)
    except NoPackets:
        if self.sendnext:
            self.timer = time.time()
            self.handle_no_packet()
            self.sendnext = False
        elif (time.time() - self.timer) > self.timeout:
            self.toCount += 1
            self.timer = time.time()
            self.handle_no_packet()
        continue
    except Shutdown:
        break

    self.handle_packet(recv)

self.shutdown()
```

3.统计指标:

计算吞吐量(Throughput)和有效吞吐量(Goodput):

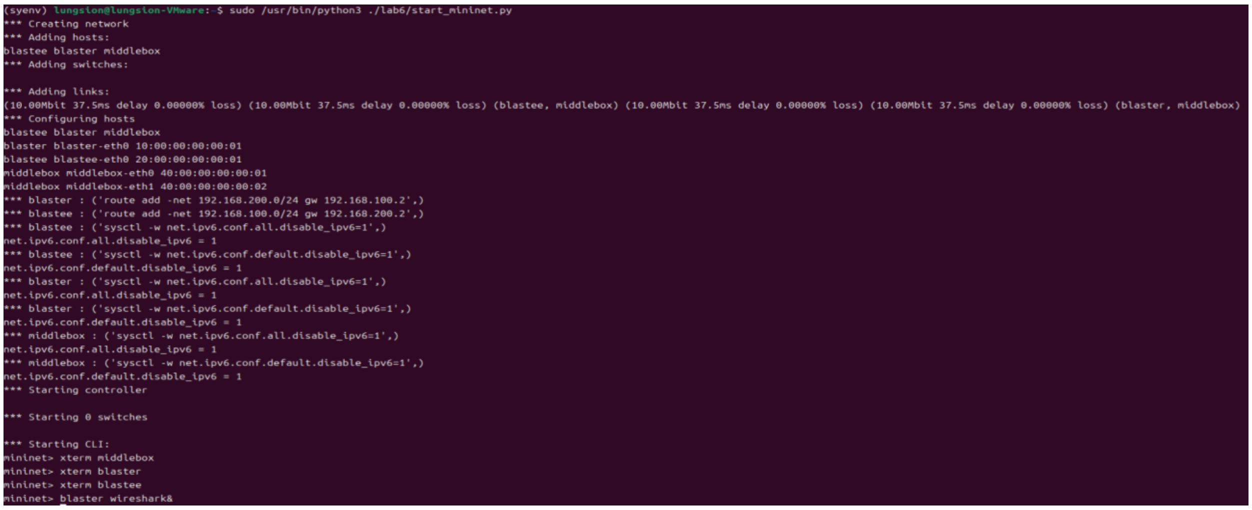


```python
self.net.shutdown()
if printInfo == False:
    return
endTime = time.time()
log_info(f"Total TX time: {int(endTime - self.startTime)}")
retxcnt = 0
for i in self.sendCount.values():
    retxcnt += i - 1
log_info(f"Number of reTX: {retxcnt}")
log_info(f"Number of coarse TOs: {self.toCount}")
log_info(f"Throughput: {float(self.sendbytesCount) / (endTime - self.startTime)}")
log_info(f"Goodput: {float(self.pllength * self.num) / (endTime - self.startTime)}")
```

4 实验结果

4.1 实验环境

Mininet 拓扑:通过 start_mininet.py 搭建三节点网络.



```
(syenv) lungsion@lungsion-VMware:~$ sudo /usr/bin/python3 ./lab6/start_mininet.py
*** Creating network
*** Adding hosts:
blastee blaster middlebox
*** Adding switches:

*** Adding links:
(10.00Mbit 37.5ms delay 0.00000% loss) (10.00Mbit 37.5ms delay 0.00000% loss) (blastee, middlebox) (10.00Mbit 37.5ms delay 0.00000% loss) (10.00Mbit 37.5ms delay 0.00000% loss) (blaster, middlebox)
*** Configuring hosts
blastee blaster middlebox
blaster blaster-eth0 10:00:00:00:00:01
blastee blastee-eth0 20:00:00:00:00:01
middlebox middlebox-eth0 40:00:00:00:00:01
middlebox middlebox-eth1 40:00:00:00:00:02
*** blaster : ('route add -net 192.168.200.0/24 gw 192.168.100.2',)
*** blastee : ('route add -net 192.168.100.0/24 gw 192.168.200.2',)
*** blastee : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** blastee : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** blaster : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** blaster : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** middlebox : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** middlebox : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** Starting controller

*** Starting 0 switches

*** Starting CLI:
mininet> xterm middlebox
mininet> xterm blaster
mininet> xterm blastee
mininet> blaster wireshark&
```

参数配置

# 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

4.2 实验结果

见 lab6.pcapng

5 实验总结

本次实验加深了我对运输层的认识,加深了对滑动窗口、超时重传、ACK 机制的理解.

致谢 在此,我向对本报告的工作给予支持和建议的老师和助教,南京大学南京大学智能软件与工程学院殷亚凤教授及其领导的助教们表示感谢.