# 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	电子科学与工程学院	专业 (方向)	电子信息科学与技术
学号	191180102	姓名	任晟昊
Email	<u>191180102@smail.nju.edu.cn</u>	开始/完成日期	2022.05.16

### 实验名称

Lab6

## 实验目的

学习滑动窗口的原理

### 实验内容

对发送端,中间窗口以及接收端的代码逻辑进行修改

### 实验结果

#### Task2

首先对中间窗口的逻辑尽心修改,按照实验手册中的说明,编写代码如下:

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
   _, fromIface, packet = recv
    if fromIface == self.intf1:
        log_debug("Received from blaster")
        Received data packet
        Should I drop it?
        If not, modify headers & send to blastee
        if random.random() < self.dropRate:</pre>
            log_info("Packet dropped!")
        else:
            packet[Ethernet].dst = EthAddr('20:00:00:00:00:01')
            self.net.send_packet("middlebox-eth1", packet)
    elif fromIface == "middlebox-eth1":
        log_debug("Received from blastee")
        111
        Received ACK
        Modify headers & send to blaster. Not dropping ACK packets!
        net.send_packet("middlebox-eth0", pkt)
        packet[Ethernet].dst = EthAddr('10:00:00:00:00:01')
        self.net.send_packet("middlebox-eth0",packet)
    else:
        log_debug("Oops :))")
```

#### Task3

对接收方进行修改,收到原数据并回复ACK

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    log_debug(f"I got a packet from {fromIface}")
    log_debug(f"Pkt: {packet}")
    packetACK = Ethernet() + IPv4(protocol=IPProtocol.UDP) + UDP()
    seq = packet[3].to_bytes()[0:4]
    payload = packet[3].to_bytes()[4:12]
    packetACK += seq
    packetACK += payload
    packetACK[Ethernet].src=EthAddr('20:00:00:00:00:01')
    packetACK[IPv4].src=IPv4Address('192.168.200.1')
    packetACK[Ethernet].dst=EthAddr('10:00:00:00:00:01')
    packetACK[IPv4].dst=IPv4Address(self.blasterIP)
    self.net.send_packet(self.net.interfaces()[0],packetACK)
```

#### Task4

按照逻辑对发送方进行修改:

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket, queue, tempTime):
   _, fromIface, packet = recv
    log_info("I got a packet")
   seq = packet[3].to_bytes()[0:4]
   seqnum = int.from_bytes(seq, byteorder='big', signed=False)
    for node in queue:
       if node.sequence == seqnum:
            node.ackflag = 1
   while len(queue) > 0:
       if queue[0].ackflag == 1 :
            del(queue[0])
            if self.LHS < self.RHS:
                self.LHS = self.LHS + 1
            tempTime = time.time()
       else:
            break
```

```
def handle_no_packet(self, queue, startTime, tempTime):
    log_info("Didn't receive anything")
   now = time.time()
    if now - tempTime > self.timeout:
        for node in queue:
            if node.ackflag == 0:
                self.net.send_packet(self.net.interfaces()[0],node.packet)
                self.reTX += 1
                self.Tos += 1
   # Do other things here and send packet
   if self.RHS < self.num:</pre>
        if self.RHS - self.LHS + 1 < self.senderWindow:</pre>
            # Creating the headers for the packet
            pkt = Ethernet() + IPv4() + UDP()
            pkt[1].protocol = IPProtocol.UDP
            pkt += self.seq.to_bytes(4,byteorder='big', signed=False)
            pkt += self.length.to_bytes(2,byteorder='big', signed=False)
            pkt += b'Test'
            pkt[Ethernet].dst=EthAddr('20:00:00:00:00:01')
```

```
pkt[IPv4].dst=self.blasteeIp
    pkt[Ethernet].src=EthAddr('10:00:00:00:01')
    pkt[IPv4].src=IPv4Address('192.168.100.1')
    self.RHS = self.seq
    queue.append(Node(pkt,self.seq))
    self.seq = self.seq + 1
    self.net.send_packet(self.net.interfaces()[0],pkt)

elif len(queue) == 0:
    endTime = time.time()
    totalTime = endTime - startTime
    print("Total TX time (in seconds)",totalTime)
    print("Number of reTX",self.reTX)
    print("Number of coarse Tos",self.TOs)
    print("Throughput (Bps)",(self.reTX + self.num) * self.length / totalTime)
    print("Goodput (Bps)",self.num * self.length / totalTime)
```

基本思想是按照实验手册中的两种情形,首先当发送方没有收到ack时,如果此时 RHS · LHS + 1 ≤ SW 时,我们只需要发送我们构造好的包,而当放松方这边计时器超时的时候,我们只需重新发送对应超时的包即可,当我们的处理队列为空且 RHS大于等于我们设定的最大num时,我们只需要打印出计算得到的数据即可。

而当发送端接收到包的时候,也即收到某一个发送包的ack,则将队列中的ackflag置1,将其从队列中移除即可,然后重置 计时器。

### **Deploy**

```
sudo python start_mininet.py

xterm middlebox

xterm blastee

xterm blaster

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'

按照上述部署,可以得到如下结果:
```

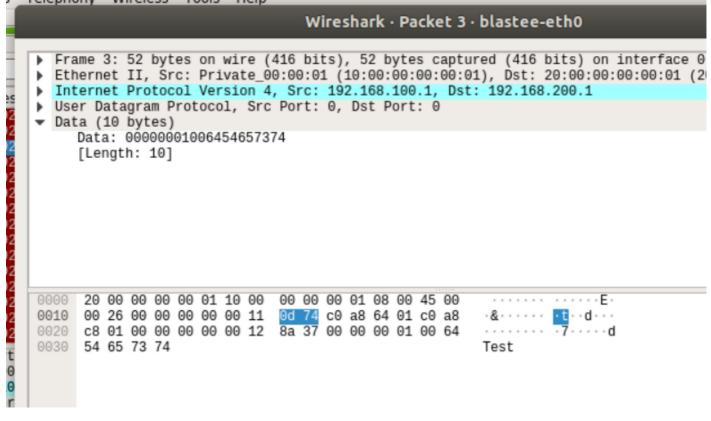
```
"Node: blaster"
                      INFO Didn't receive anything
11:44:02 2022/05/18
97 100
11:44:02 2022/05/18
                       INFO I got a packet
98 100
                       INFO Didn't receive anything
11:44:02 2022/05/18
98 100
                       INFO Didn't receive anything
11:44:02 2022/05/18
98 100
11:44:02 2022/05/18
                       INFO Didn't receive anything
98 100
11:44:02 2022/05/18
                       INFO I got a packet
98 100
11:44:02 2022/05/18
                       INFO I got a packet
99 100
11:44:02 2022/05/18
                       INFO I got a packet
99 100
11:44:02 2022/05/18
                       INFO I got a packet
100 100
11:44:02 2022/05/18
                       INFO Didn't receive anything
Total TX time (in seconds) 14.785858631134033
Number of reTX 184
Number of coarse TOs 184
Throughput (Bps) 1920.7541955121344
Goodput (Bps) 676.3218998282164
```

		"Node: blastee"							
	96 11:44:01 97	2022/05/18	INFO I	[	got	a	packet	from	blastee-eth0
		2022/05/18	INFO I	[	got	a	packet	from	blastee-eth0
	11:44:01 98	2022/05/18	INFO I	[	got	a	packet	from	blastee-eth0
		2022/05/18	INFO I		got	a	packet	from	blastee-eth0
		2022/05/18	INFO I	[	got	a	packet	from	blastee-eth0
i	11:44:02 99	2022/05/18	INFO I	[	got	a	packet	from	blastee-eth0
	11:44:02 100	2022/05/18	INFO I		got	a	packet	from	blastee-eth0
	11:44:02 99	2022/05/18	INFO I	[	got	a	packet	from	blastee-eth0
	11:44:02 100	2022/05/18	INFO I	[	got	a	packet	from	blastee-eth0
	11:44:02 99	2022/05/18	INFO I	I	got	a	packet	from	blastee-eth0
- 44		2022/05/18	INFO I	I	got	a	packet	from	blastee-eth0

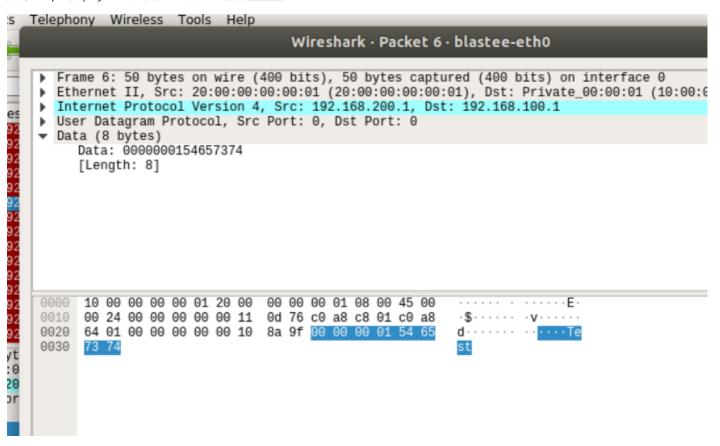
		<u></u>	<u></u>			<u></u>			
			"Node: middlebox"						
S	end pac	97							
1	1:44:01	2022/05/18	INFO	Packet dr	ropped	!			
S	end pac	98							
1	1:44:01	2022/05/18	INFO	Packet di	ropped	!			
_1	1:44:01	2022/05/18	INFO	Received	from	blastee			
1	1:44:02	2022/05/18	INF0	Received	from	blastee			
S	end pac	98							
S	end pac	99							
1	1:44:02	2022/05/18	INFO	Packet di	ropped	!			
	end pac								
	end pac								
a n		2022/05/18							
		2022/05/18	INFO	Received	from	blastee			
e	end pac								
	end pac				_				
L		2022/05/18		Received					
		2022/05/18	INFO	Received	from	blastee			
	end pac								
		2022/05/18		Received					
		2022/05/18		Received					
Ţ	1:44:02	2022/05/18	INFO	Received	Trom	blastee			
1	1:44:02	2022/05/18 2022/05/18	INFO	Received Received	from	blastee			

由上述结果可以看到,基本功能已经实现,具体实现逻辑已经在代码中体现出来。

在wireshark中可以抓到如下包:



其中seq=1, payload是我们的自定义字符 Test



这个是blastee发送回去的ack。

下面改变 recvTimeout 的值,使得 recvTimeout=1000 观察输出结果:

```
"Node: blaster"
12:20:21 2022/05/18
                        INFO I got a packet
98 99
12:20:21 2022/05/18
                        INFO I got a packet
99 99
12:20:22 2022/05/18
                        INFO Didn't receive anything
99 100
12:20:22 2022/05/18
                        INFO I got a packet
100 100
                        INFO Didn't receive anything
12:20:23 2022/05/18
Total TX time (in seconds) 121.63294124603271
Number of reTX 22
Number of coarse TOs 22
Throughput (Bps) 100.30177577735691
Goodput (Bps) 82.21457030930894
```

下面改变 dropRate, 使得 dropRate=0.39 时可见:

File	Luit	Selection	V I C VV	do De	Dug	161111	IIIat	Heth	viastei.p
						"!	Node:	blaste	er"
<sup>1</sup> 12:30:	00	2022/05	/18	INFO	I	got a	pa	cket	
98 98									
(12:30:	01	2022/05	/18	INF0	Dio	dn't	rec	eive	anything
98 99									
12:30:	02	2022/05	/18	INFO	ΙĢ	got a	pa	cket	
299 99									
		2022/05	/18	INFO	Dio	dn't	rec	eive	anything
<sub></sub> 99 100									
		2022/05	/18	INFO	Ιģ	got a	pa	cket	
100 10									
									anything
		time (i		nds) 1	30.2	26318	454	74243	32
		reTX 5		-					
Number of coarse TOs 57									
Throughput (Bps) 120.52522786500873 Goodput (Bps) 76.76766106051511									
Goodpu	τ (	Rbs) /6	./6/66.	100051	DTT				

由于重传变多,所以Throughput变大,但是goodput变小,也正是表明了因为网络的不稳定造成了实际有效传输的减小。

### 实验总结

本次实验首先要理清整体逻辑,进而按照实验手册的逻辑完善自身的实验流程,最后将其部署在网络中得到相应的结果。