计算机网络 lab_7

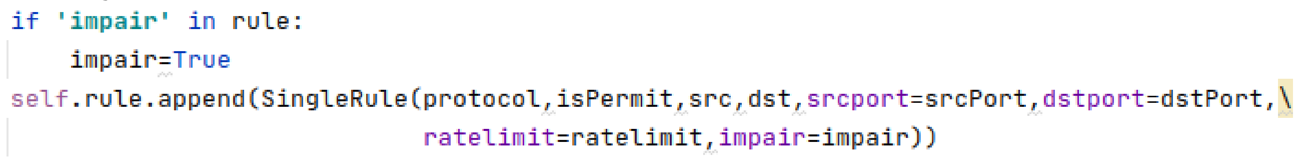
181840135 梁俊凯

Task1

准备工作，复制相应文件到文件夹。

Task2

应用较简单的 firewall_rules，我的做法为先对 firewall_rule.txt 进行解析，过滤掉注释，得到 rule 的纯文本，再对每条文本进行分析，转换成与 switchyard 相关的变量，代码部分截取如下：

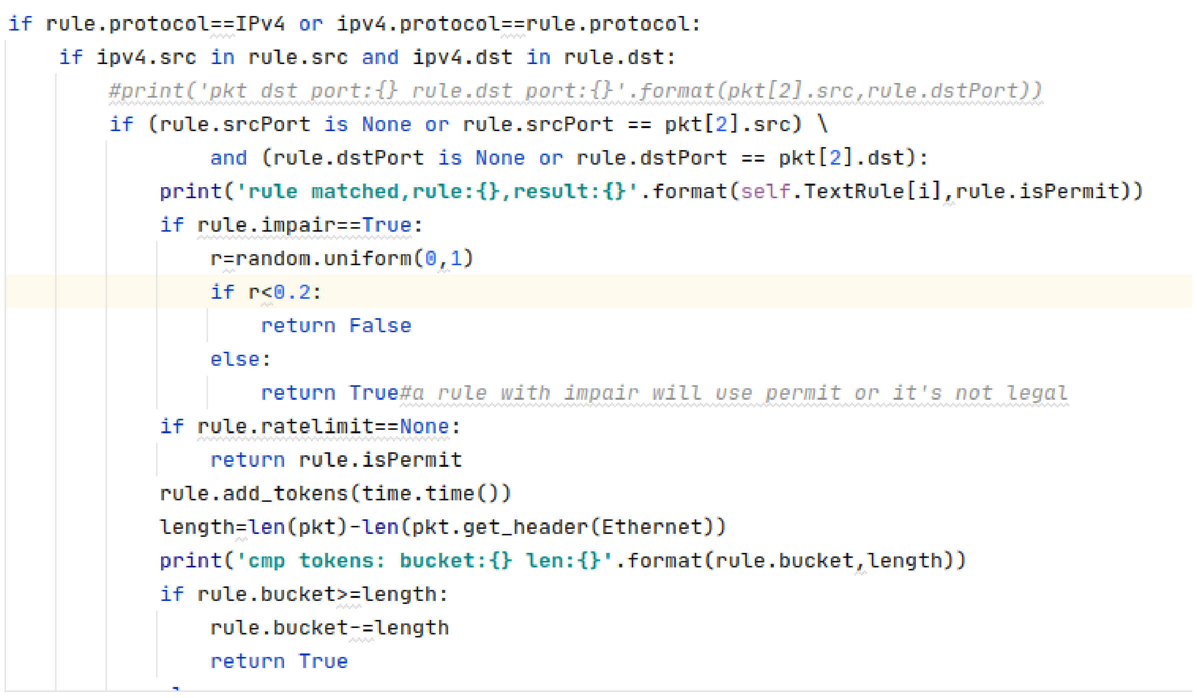


```python
if 'impair' in rule:
    impair=True
self.rule.append(SingleRule(protocol,isPermit,src,dst,srcport=srcPort,dstport=dstPort,\
                                ratelimit=ratelimit,impair=impair))
```

SingleRule 即一个对象名，存储一条规则的信息

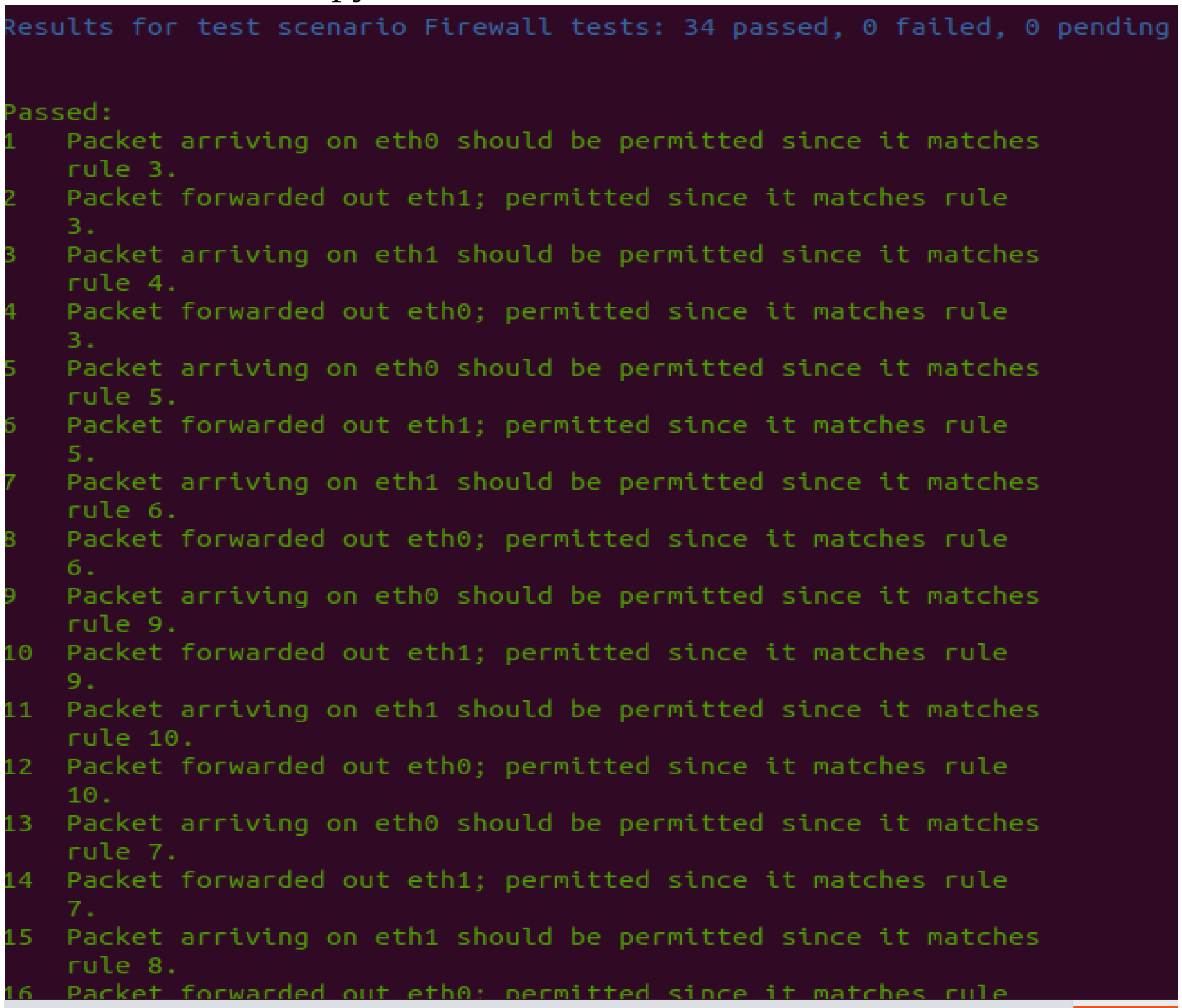
规则分析完成后，定义 judge_permit 函数，判断一个 pkt 包是否可以通过 firewall。judge_permit 即先通过协议，ip 地址，端口号判断是否匹配，若匹配，再根据 ratelimit 或 impair（Task3 Task4）来进行滤过。

部分代码截取如下：

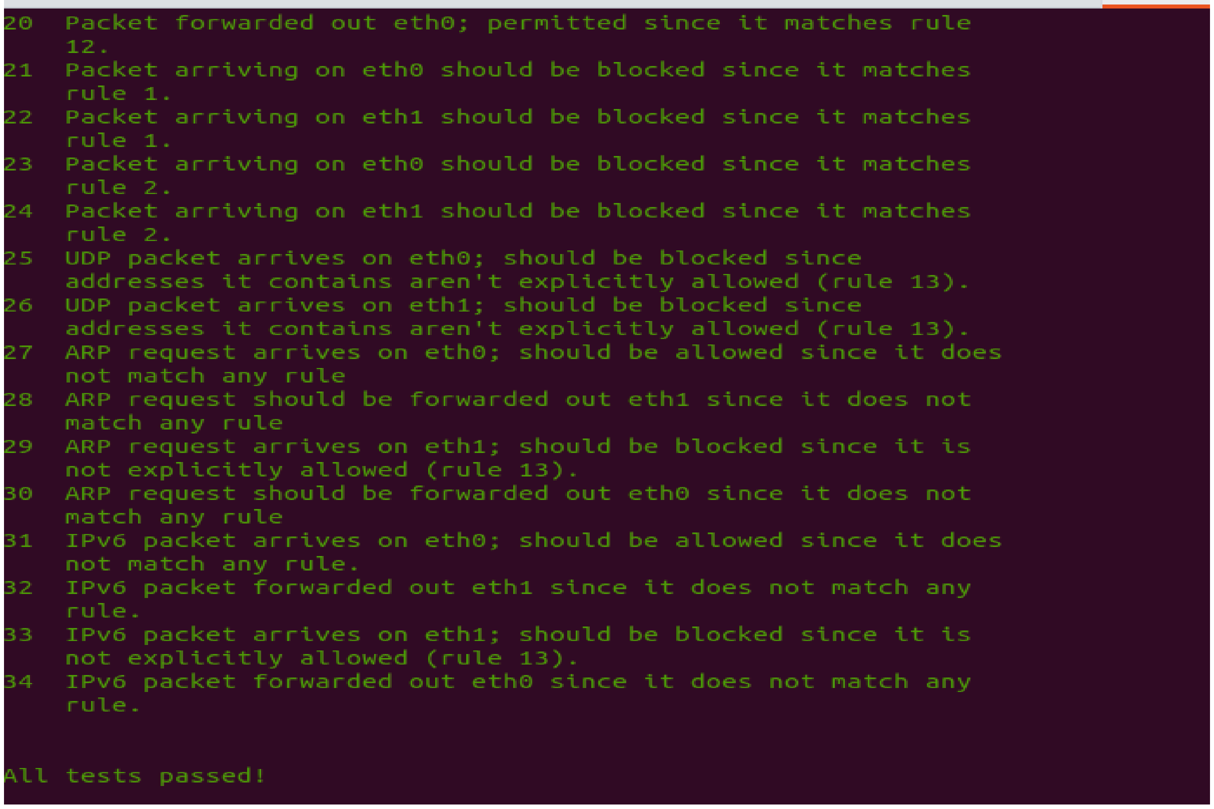


```python
if rule.protocol==IPv4 or ipv4.protocol==rule.protocol:
    if ipv4.src in rule.src and ipv4.dst in rule.dst:
        #print('pkt dst port:{} rule.dst port:{}'.format(pkt[2].src,rule.dstPort))
        if (rule.srcPort is None or rule.srcPort == pkt[2].src) \
                and (rule.dstPort is None or rule.dstPort == pkt[2].dst):
            print('rule matched,rule:{},result:{}'.format(self.TextRule[i],rule.isPermit))
            if rule.impair==True:
                r=random.uniform(0,1)
                if r<0.2:
                    return False
                else:
                    return True#a rule with impair will use permit or it's not legal
            if rule.ratelimit==None:
                return rule.isPermit
            rule.add_tokens(time.time())
            length=len(pkt)-len(pkt.get_header(Ethernet))
            print('cmp tokens: bucket:{} len:{}'.format(rule.bucket,length))
            if rule.bucket>=length:
                rule.bucket-=length
                return True
```

我运行了 firewalltests.py 通过了所有的测试样例。



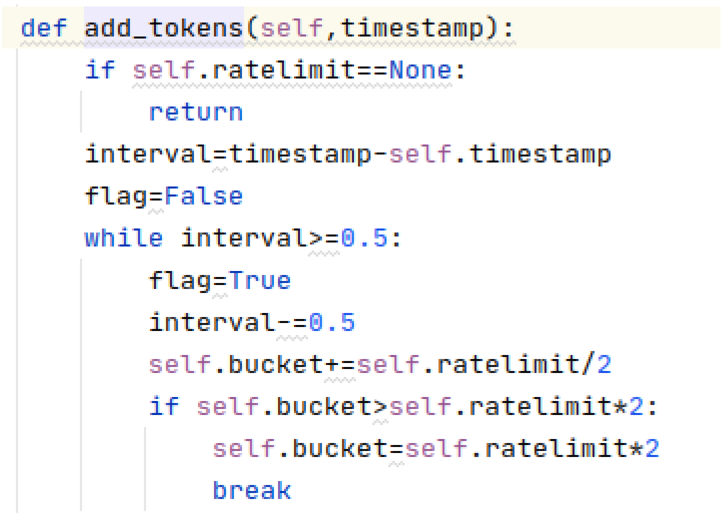
```
Results for test scenario Firewall tests: 34 passed, 0 failed, 0 pending


Passed:
1   Packet arriving on eth0 should be permitted since it matches
    rule 3.
2   Packet forwarded out eth1; permitted since it matches rule
    3.
3   Packet arriving on eth1 should be permitted since it matches
    rule 4.
4   Packet forwarded out eth0; permitted since it matches rule
    3.
5   Packet arriving on eth0 should be permitted since it matches
    rule 5.
6   Packet forwarded out eth1; permitted since it matches rule
    5.
7   Packet arriving on eth1 should be permitted since it matches
    rule 6.
8   Packet forwarded out eth0; permitted since it matches rule
    6.
9   Packet arriving on eth0 should be permitted since it matches
    rule 9.
10  Packet forwarded out eth1; permitted since it matches rule
    9.
11  Packet arriving on eth1 should be permitted since it matches
    rule 10.
12  Packet forwarded out eth0; permitted since it matches rule
    10.
13  Packet arriving on eth0 should be permitted since it matches
    rule 7.
14  Packet forwarded out eth1; permitted since it matches rule
    7.
15  Packet arriving on eth1 should be permitted since it matches
    rule 8.
16  Packet forwarded out eth0; permitted since it matches rule
```



```
20  Packet forwarded out eth0; permitted since it matches rule
    12.
21  Packet arriving on eth0 should be blocked since it matches
    rule 1.
22  Packet arriving on eth1 should be blocked since it matches
    rule 1.
23  Packet arriving on eth0 should be blocked since it matches
    rule 2.
24  Packet arriving on eth1 should be blocked since it matches
    rule 2.
25  UDP packet arrives on eth0; should be blocked since
    addresses it contains aren't explicitly allowed (rule 13).
26  UDP packet arrives on eth1; should be blocked since
    addresses it contains aren't explicitly allowed (rule 13).
27  ARP request arrives on eth0; should be allowed since it does
    not match any rule
28  ARP request should be forwarded out eth1 since it does not
    match any rule
29  ARP request arrives on eth1; should be blocked since it is
    not explicitly allowed (rule 13).
30  ARP request should be forwarded out eth0 since it does not
    match any rule
31  IPv6 packet arrives on eth0; should be allowed since it does
    not match any rule.
32  IPv6 packet forwarded out eth1 since it does not match any
    rule.
33  IPv6 packet arrives on eth1; should be blocked since it is
    not explicitly allowed (rule 13).
34  IPv6 packet forwarded out eth0 since it does not match any
    rule.


All tests passed!
```

Task3

实现 ratelimit。即定义一个 tokens bucket，根据时间增加其中的 tokens 值，并设置上限，tokens 数代表当前该规则匹配下可以转发的比特数，每当转发包时，就减去相应包的大小（不包括 Ethernet）的 tokens 数，若小于 0，则证明超过 ratelimit，drop 该包，tokens 需要随时间而增长，再遇到规则时，调用 add_tokens 函数：

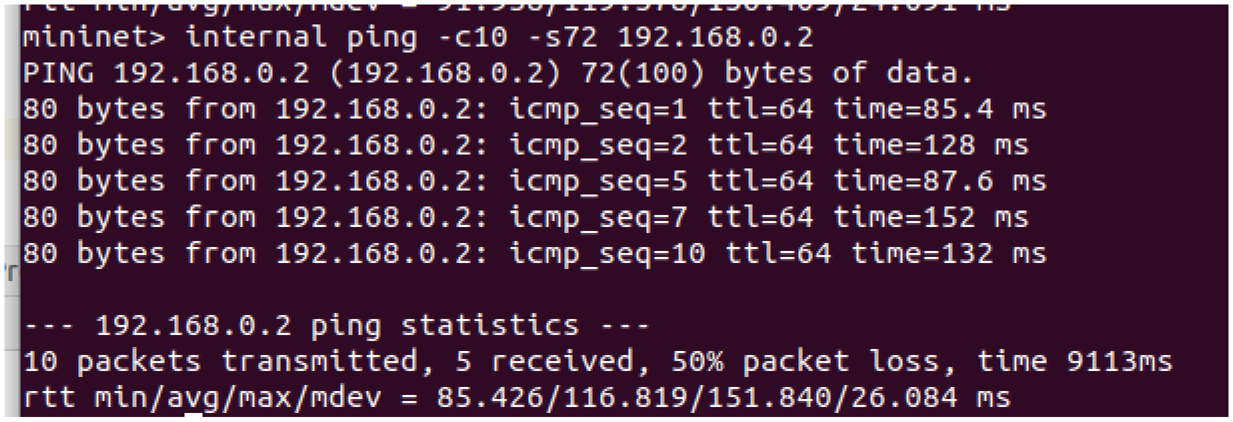


```python
def add_tokens(self,timestamp):
    if self.ratelimit==None:
        return
    interval=timestamp-self.timestamp
    flag=False
    while interval>=0.5:
        flag=True
        interval-=0.5
        self.bucket+=self.ratelimit/2
        if self.bucket>self.ratelimit*2:
            self.bucket=self.ratelimit*2
            break
```

在 judge_permit 中判断 bucket 大小即可，代码在 Task2 中已附出

利用 Task5: Testing 中的测试如下

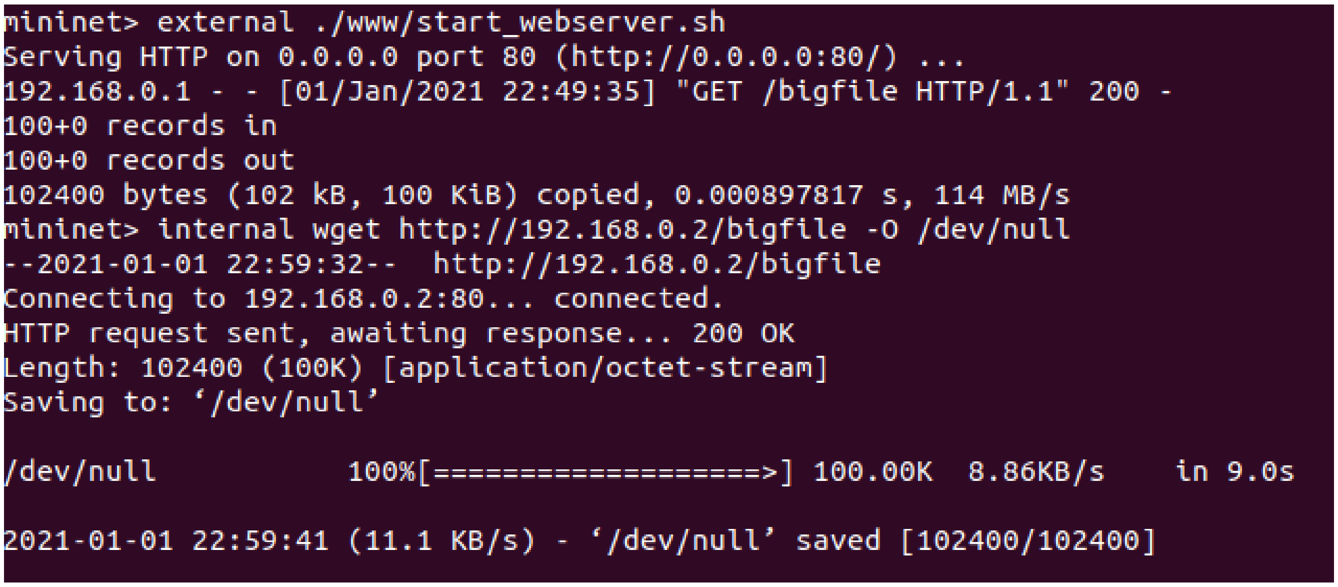
1>利用 ICMP 来测试 ratelimit



```
mininet> internal ping -c10 -s72 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 72(100) bytes of data.
80 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=85.4 ms
80 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=128 ms
80 bytes from 192.168.0.2: icmp_seq=5 ttl=64 time=87.6 ms
80 bytes from 192.168.0.2: icmp_seq=7 ttl=64 time=152 ms
80 bytes from 192.168.0.2: icmp_seq=10 ttl=64 time=132 ms

--- 192.168.0.2 ping statistics ---
10 packets transmitted, 5 received, 50% packet loss, time 9113ms
rtt min/avg/max/mdev = 85.426/116.819/151.840/26.084 ms
```

可以看到，10 个 ICMP 只收到了五个回复，与预期相符。

2>利用 webserver 来进行测试。

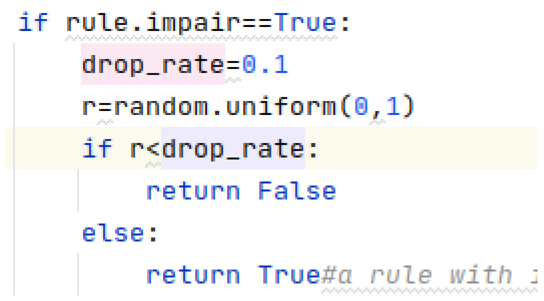


```
mininet> external ./www/start_webserver.sh
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.0.1 - - [01/Jan/2021 22:49:35] "GET /bigfile HTTP/1.1" 200 -
100+0 records in
100+0 records out
102400 bytes (102 kB, 100 KiB) copied, 0.000897817 s, 114 MB/s
mininet> internal wget http://192.168.0.2/bigfile -O /dev/null
--2021-01-01 22:59:32--  http://192.168.0.2/bigfile
Connecting to 192.168.0.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 102400 (100K) [application/octet-stream]
Saving to: '/dev/null'

/dev/null            100%[===================>] 100.00K  8.86KB/s    in 9.0s

2021-01-01 22:59:41 (11.1 KB/s) - '/dev/null' saved [102400/102400]
```

看到速率为 8.68KB/s 而 12500 大约为 12KB/s，满足 ratelimit 的限制。

Task4

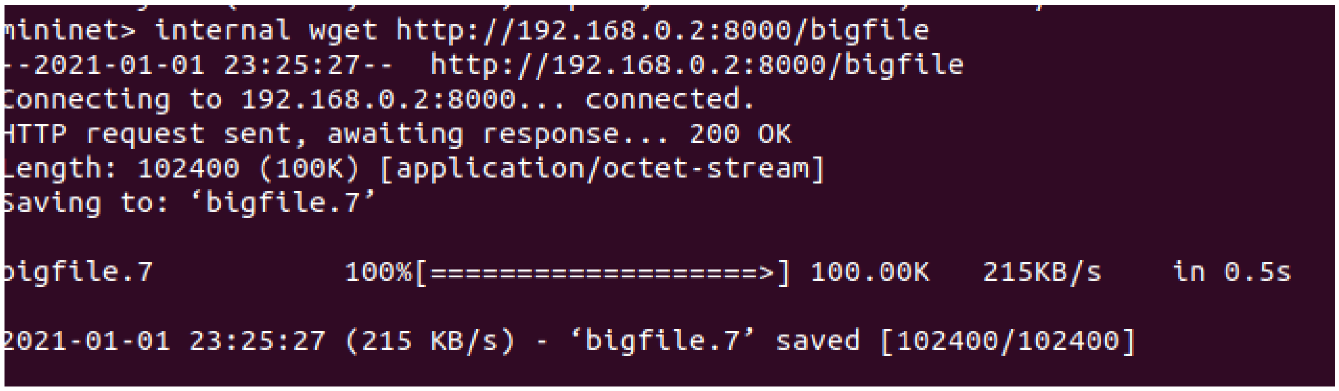
实现 impairment，我采用随机丢包的实现方式，即在 judge_permit 中，若发现 rule.impairment==True，则定义一个（0,1）间的随机数和 drop_rate，若 r<drop_rate，则丢弃之。



```
if rule.impair==True:
    drop_rate=0.1
    r=random.uniform(0,1)
    if r<drop_rate:
        return False
    else:
        return True#a rule with i
```

利用 Test5 中相关的测试来检验 impairment 实现的正确性

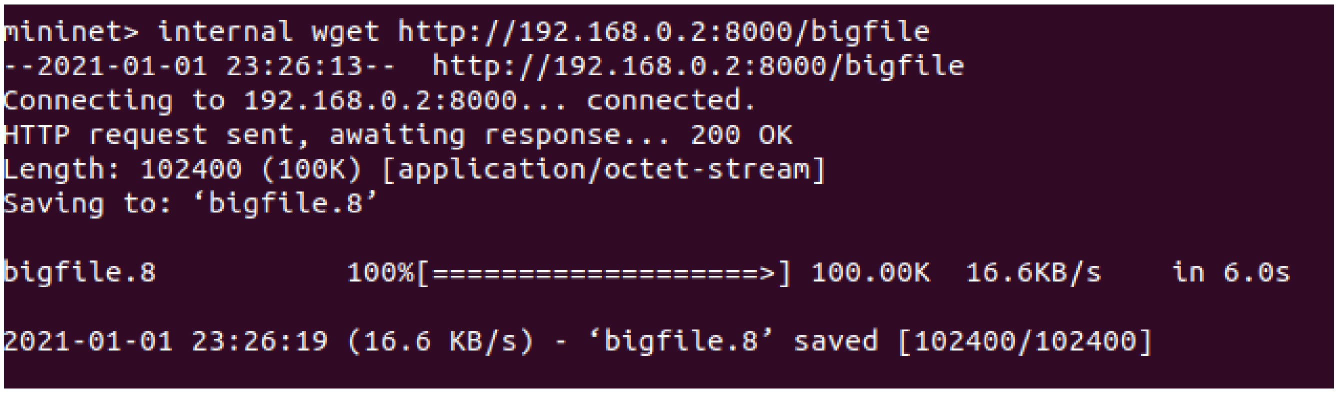
1>drop_rate=0 即不丢包



```
mininet> internal wget http://192.168.0.2:8000/bigfile
--2021-01-01 23:25:27--  http://192.168.0.2:8000/bigfile
Connecting to 192.168.0.2:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 102400 (100K) [application/octet-stream]
Saving to: 'bigfile.7'

bigfile.7            100%[===================>] 100.00K   215KB/s    in 0.5s

2021-01-01 23:25:27 (215 KB/s) - 'bigfile.7' saved [102400/102400]
```

可见此时速率极快，达到了 215KB/s（没有 ratelimit 的限制）

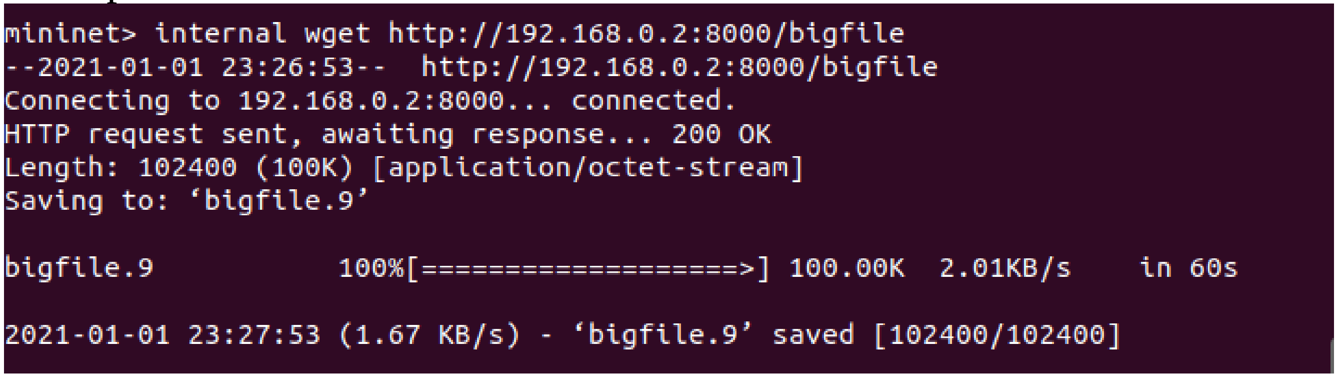
2>drop_rate=0.1



```
mininet> internal wget http://192.168.0.2:8000/bigfile
--2021-01-01 23:26:13--  http://192.168.0.2:8000/bigfile
Connecting to 192.168.0.2:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 102400 (100K) [application/octet-stream]
Saving to: ‘bigfile.8’

bigfile.8           100%[===================>] 100.00K  16.6KB/s    in 6.0s

2021-01-01 23:26:19 (16.6 KB/s) - ‘bigfile.8’ saved [102400/102400]
```

可见此时速率变为 16.6KB/s 约为刚才的 1/15，但是只考虑 10%的包进行重传的话，速率不会下降这么快，可见 TCP 协议的速率与包的丢失情况有很大相关性

3>drop_rate=0.2



```
mininet> internal wget http://192.168.0.2:8000/bigfile
--2021-01-01 23:26:53--  http://192.168.0.2:8000/bigfile
Connecting to 192.168.0.2:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 102400 (100K) [application/octet-stream]
Saving to: ‘bigfile.9’

bigfile.9           100%[===================>] 100.00K  2.01KB/s    in 60s

2021-01-01 23:27:53 (1.67 KB/s) - ‘bigfile.9’ saved [102400/102400]
```

此时速率又呈指数般下降，变为了约 2.01KB/s，证明刚才的结论：丢包的比例是 TCP 协议的速率的很大决定因素。

当 drop_rate 继续增加时，包传递变得更慢，故不再继续进行测试。
以上的结果与预期相符，当 drop_rate 增长时，接受端和发送端总的吞吐率下降，证明 impairment 的实现是正确的

Task5

Task5 中的测试以在上文中全部展示，本次实验的报告内容到此结束，感谢阅读！

总结

本学期的计网实现已经结束，通过这些 lab 我学到了网络各个层次的知识与其具体实现，收益匪浅，也非常感谢认真维护这些实验的助教们，助教辛苦了！期待以后能更深入学习计算机网络的知识！