Computer Networking-Lab-Repot

课程名称：**计算机网络** 任课教师：**田臣/李文中**

学院	Dept. of Computer Science and Technology	专业（方向）	CS
学号	181830044	姓名	董宸郅
Email	pdon#foxmail.com	开始/完成日期	5.27/6.17

实验名称： Lab 7: Firewall

实验目的

- 实现一个简易的防火墙机制，对经过的分组施加过滤、限速、损坏等操作
- 加深对于令牌桶机制的理解

实验内容

理论知识

令牌桶

令牌桶算法可用来**控制**发送到网络上的**数据量**。

大小固定的令牌桶可自行**以恒定的速率源源不断地产生令牌**。如果令牌不被消耗，或者被消耗的速度小于产生的速度，令牌就会不断地增多，直到把桶填满。后面再产生的令牌就会从桶中溢出。最后桶中可以保存的最大令牌数永远不会超过桶的大小。

传送到令牌桶的数据包需要消耗令牌。令牌桶中的**每一个令牌都代表一个字节**。如果令牌桶中存在令牌，则允许发送流量；而如果令牌桶中不存在令牌，则不允许发送流量。因此，如果突发门限被合理地配置并且令牌桶中有足够的令牌，那么流量就可以以峰值速率发送。

实验步骤（含测试结果与关键代码）

`FYI`：代码逻辑细节见注释

Task 2: Implement firewall rules

`Coding`

设计一个类`Rule`用来存放单个规则：

```
class Rule:
    '''
    some useful data related to firewall rules
    '''
    def __init__(self, line):
        self.deny = (line[0] == 'deny')
```

```
        self.impair = (line[-1] == 'impair')
        if line[-2] == 'ratelimit':
            self.rl = int(line[-1])
        else:
            self.rl = -1

        if line[1] == 'ip':
            self.type = 'IPv4'
        elif line[1] == 'icmp':
            self.type = 'ICMP'
        elif line[1] == 'tcp':
            self.type = 'TCP'
        elif line[1] == 'udp':
            self.type = 'UDP'

        if line[1] == 'ip' or line[1] == 'icmp':
            self.src = line[3]
            self.dst = line[5]
        elif line[1] == 'tcp' or line[1] == 'udp':
            self.src = line[3]
            self.srcport = line[5]
            self.dst = line[7]
            self.dstport = line[9]
```

函数`gather_rules`用来读取rule文件以及初始化`rules`列表（以及`buckets`字典）：

```
def gather_rules():
    '''
    gather rules from 'firewall_rules.txt'
    and create a list to include them
    '''
    rules = []
    buckets = {}
    file = open('firewall_rules.txt', 'r')
    for line in file.readlines():
        line = line.split()
        if not len(line) or line[0][0] == '#':
            continue
        new_rule = Rule(line)
        rules.append(new_rule)
        if new_rule.rl != -1:
            bkt = Bucket(new_rule.rl)
            buckets[new_rule] = bkt
    return rules, buckets
```

函数`filter_pkt`用来依据规则过滤分组：

```
def filter_pkt(rules, pkt, net, output_port, buckets):
    '''
    filter the pkt according to the rules
    '''
    if not pkt.has_header(IPv4):
        net.send_packet(output_port, pkt)
        return
    ip_pkt = pkt[IPv4]
    for rule in rules:
```

```
        if not (rule.type == 'IPv4' or rule.type in pkt.headers()):
            continue
        if not is_matchable(rule, pkt):
            continue
        break
    # no match found
    if not rule:
        net.send_packet(output_port, pkt)
        return
    # filter this pkt
    if rule.deny:
        # drop it
        return
    if rule.rl == -1 and (not rule.impair):
        # no additional conditions
        net.send_packet(output_port, pkt)
        return
```

在`gather_rules`中，调用函数`is_matchable`判断当前规则与这个分组是否匹配，`is_matchable`又调用`compare_ip`与`compare_port`这两个小函数：

```
def compare_ip(x, y):
    '''
    compare two IPv4 addrs and see if y belong to x
    '''
    if x == 'any':
        return True
    x1 = int(IPv4Network(x, strict=False).network_address)
    y1 = int(IPv4Network(y, strict=False).network_address)
    return x1 & y1 == x1


def compare_port(x, y):
    '''
    compare two port numbers and see if they match
    '''
    if x == 'any':
        return True
    return int(x) == y


def is_matchable(rule, pkt):
    '''
    see if this rule match the pkt
    '''
    ip_pkt = pkt[IPv4]
    # compare IP addr
    if not (compare_ip(rule.src, ip_pkt.src) and compare_ip(rule.dst,
ip_pkt.dst)):
        return False
    if rule.type == 'IPv4':
        return True
    # compare TCP/UDP port
    if ip_pkt.protocol == IPProtocol.UDP:
        if not (compare_port(rule.srcport, pkt[UDP].src) and
compare_port(rule.dstport, pkt[UDP].dst)):
            return False
```

```
    elif ip_pkt.protocol == IPProtocol.TCP:
        if not (compare_port(rule.srcport, pkt[TCP].src) and
compare_port(rule.dstport, pkt[TCP].dst)):
            return False
    return True
```

Task 3: Implement the token bucket algorithm

设计一个类`Bucket`来存放`ratelimit`与令牌数（初始化工作同样在`gather_rules`中完成）：

```
class Bucket:
    '''
    designed for the token bucket algorithm
    '''
    def __init__(self, rl):
        self.tokens = 2 * rl
        self.rl = rl
```

在`filter_pkt`中，处理`ratelimit`，满足条件才发出：

```
# handle ratelimit
bkt = buckets[rule]
size = len(pkt) - len(pkt[Ethernet])
if size <= bkt.tokens:
    bkt.tokens -= size
    net.send_packet(output_port, pkt)
```

在`main`中的大循环里，每次判断是否要更新`buckets`字典：

（之前把这一步放在了取到`pkt`之后，于是出现了bug）

```
if time.time() - update_time >= 0.25:
    update_time = time.time()
    for bkt in buckets.values():
        bkt.tokens = min(bkt.rl / 4 + bkt.tokens, 2 * bkt.rl)
```

Task 4: Implement some other type of network impairment

在`filter_pkt`中，处理`impair`，选择`TCP advertised window`：

```
# handle impair (TCP advertised window)
if pkt.has_header(TCP):
    sz = pkt[TCP].window
    pkt[TCP].window = change_wndw(sz)
```

通过`change_wndw`函数对window大小进行随机减小：

```
def change_wndw(sz):
    '''
    randomly change the TCP advertised window size
    '''
    return int(random.random() * sz)
```

Task 5: Testing

basic firewall

```
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
    8.
17  Packet arriving on eth0 should be permitted since it matches
    rule 12.
18  Packet forwarded out eth1; permitted since it matches rule
    12.
19  Packet arriving on eth1 should be permitted since it matches
    rule 12.
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!
```

token bucket

- `rule 13`

  经多次测试后发现，`ICMP reply`的丢包率基本在**50%**，且由于`token bucket`的实现机制，常常**每隔一个丢一次包**。

```
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=208 ms
80 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=143 ms
80 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=205 ms
80 bytes from 192.168.0.2: icmp_seq=6 ttl=64 time=186 ms
80 bytes from 192.168.0.2: icmp_seq=8 ttl=64 time=154 ms

--- 192.168.0.2 ping statistics ---
10 packets transmitted, 5 received, 50% packet loss, time 9076ms
rtt min/avg/max/mdev = 143.424/179.727/208.694/26.362 ms
```

- `rule 7&8`

  规定的限速在`12.5 KB/s`，经多次测试，实际传输速度在`8.5~11.5 KB/s`左右。

```
mininet> internal wget http://192.168.0.2/bigfile -O /dev/null
--2020-06-11 18:32:33--  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  10.1KB/s    in 8.8s

2020-06-11 18:32:42 (11.3 KB/s) - ‘/dev/null’ saved [102400/102400]
```

impair

为测试`TCP window`大小的前后改变，我采用直接输出的方式：

```
print("The TCP advertised window size changed from {} to {}!!!\n".format(sz, pkt[TCP].window))
```

```
mininet> external ./www/start_webserver.sh 8000
100+0 records in
100+0 records out
102400 bytes (102 kB, 100 KiB) copied, 0.00058089 s, 176 MB/s
mininet> internal wget http://192.168.0.2:8000/bigfile
--2020-06-11 20:46:36--  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.5’

bigfile.5           100%[===================>] 100.00K   113KB/s    in 0.9s

2020-06-11 20:46:37 (113 KB/s) - ‘bigfile.5’ saved [102400/102400]
```

```
The TCP advertised window size changed from 42190 to 31967!!!

The TCP advertised window size changed from 42190 to 33924!!!

The TCP advertised window size changed from 42190 to 36611!!!

The TCP advertised window size changed from 42137 to 23236!!!

The TCP advertised window size changed from 41808 to 5068!!!

The TCP advertised window size changed from 40736 to 1809!!!

The TCP advertised window size changed from 41808 to 27402!!!

The TCP advertised window size changed from 41808 to 9182!!!

The TCP advertised window size changed from 41808 to 10459!!!

The TCP advertised window size changed from 42190 to 32419!!!

The TCP advertised window size changed from 42190 to 4637!!!

The TCP advertised window size changed from 42190 to 11371!!!
```

总结与感想

不知不觉，竟然已经做完最后一个实验了，还有点小不舍（不是）。

此次实验的难度较小，我遇到的bug比较少，唯一有点问题的是没有考虑到更新`buckets`应该放在取到`pkt`的外部进行，但也很快在群里得到了解答。

计网的课已经全部上完了，虽然还要过至少两个月才会考试，但额外多出来的时间也要充分利用。毕竟实验只涉及最粗浅的知识，教材里的内容感觉不会的还是很多。

骚年，继续加油吧！