Stateful firewall

防火墙可以分为有状态（Stateful）和无状态（Stateless）。这里说的防火墙指的不仅是Firewall，也包括OpenStack中常使用的Security Group，它们只是不同层面的防火墙。

无状态防火墙就是基于静态数值来过滤或阻拦网络数据包。例如基于地址，端口，协议等等。无状态指的就是，防火墙本身不关心当前的网络连接状态。

有状态防火墙能区分出网络连接的状态。例如TCP连接，有状态防火墙可以知道当前是连接的哪个阶段。也就是说，有状态防火墙可以在静态数值之外，再通过连接状态来过滤或阻拦网络数据包。

这两个哪个更好？严格来说没有更好，具体要看使用场景。无状态的防火墙一般更快，在高负载网络流量下性能更好，但是只适用于简单的场景。而有状态的防火墙一般会安全的多，因为它可以定义更加严格的规则。

Connection tracker

Stateful防火墙的特点就是能知道网络连接当前的状态，并根据网络连接的状态来过滤网络流量。网络连接的状态，通常是由connection tracker来实现。可以简称为CT模块。在linux内核中，CT是由conntrack实现，最开始实现是为了Netfilter 框架能了解特定网络连接的状态。其内部实际上是对各个网络连接实现了状态机，这里的状态机区别于TCP连接本身的状态机，毕竟网络连接本身不一定是有状态的，也就是说，CT模块自己有一套状态机。

CT模块可以结合iptables实现有状态防火墙，不过今天说一下如何结合OpenvSwitch和OpenFlow来实现有状态防火墙。

在OpenvSwitch实现的OpenFlow中，CT状态由ct_state提供。ct_state提供如下状态：

- 0x01: new：这是一个连接的开始状态（NEW），这个状态只可能出现在还未commit的connection中。通常这意味着这个包是网络连接中的第一个数据包。
- 0x02: est：这是一个已经存在的连接（ESTABLISHED），这个状态只可能出现在已经commit的connection中。并且当前的数据包是合法的，例如TCP连接中，client发送了SYN，server回复的SYN/ACK。
- 0x04: rel：这是一个与已经存在的连接（est）有关联的连接（RELATED），例如ICMP "destination unreachable" messages 或者FTP 数据连接。这个状态只可能出现在已经commit的connection中。
- 0x08: rpl：匹配回复方向的数据，意味着当前主机并没有初始化连接。这个状态只可能出现在已经commit的connection中。
- 0x10: inv：当前数据包状态是无效的，意味着CT模块不能正确识别当前数据包，例如：L3/L4 protocol handler未加载或者不可用。在linux下，这通常意味着nf_conntrack 模块未加载。L3/L4 protocol handler发现packet包不合法。Packet 长度不符合协议要求。
- 0x20: trk：表明当前数据包经过了CT模块。如果这个标志没有被设置，那么其他所有的标志位都不可能被设置。
- 0x40: snat：数据包经过了SNAT。
- 0x80: dnat：数据包经过了DNAT。

ct_state是从OpenvSwitch 2.5版本才有，而snat和dnat更是从2.6版本才开始有。正是这个原因，早期的OpenStack Neutron的OpenvSwitch实现方案要在虚机port和br-int中间加一个linux bridge来应用iptables以实现Security Group。这实在是个复杂低效的无奈之举，不过后期的SDN实现方案都没有采用这种网络连接方案了。

网络数据包在OpenFlow中有两种状态，untracked和tracked。当网络数据包经过了CT模块，那它就是tracked的，对应的（0x20: trk）会置位。

网络连接在OpenFlow中也有两种状态，uncommitted和committed。CT模块判断connection是否是ESTABLISHED（0x02: est），commit是一个必要条件。

看一个小例子，这个例子在后面还会提到：

```
table=0,priority=100,ip,ct_state=-trk,action=ct(table=1)
table=1,in_port=1,ip,ct_state=+trk+new,action=ct(commit),2table=1,in_port=1,ip,
table=1,in_port=2,ip,ct_state=+trk+new,action=drop
table=1,in_port=2,ip,ct_state=+trk+est,action=1
```

首先，对所有untracked的packets，执行ct(table=1)。这里实际上做了好几件事情：packet被复制，并发送到了CT模块；CT模块处理完之后将packet重新注入到OpenFlow table 1；Table 1收到的packet ct_state已经被设置上相应的值。这里指定的table序号最好大于当前table，以免陷入循环处理。而原packet可以作为untracked packet继续在OpenFlow中处理，只是例子中没有指定其他的操作。

在table 1中，对ESTABLISHED的连接，直接发送到对端端口。

在table 1中，对NEW的连接，也就是还未提交到CT模块的连接，对于来自端口1的连接，执行ct(commit)，再提交到端口2。这样，connection成为了committed，而connection状态变成ESTABLISHED也成为了可能。而对来自端口2的NEW连接，直接drop。

最终的实现效果是，从端口1发起的向端口2的连接可以建立相互通讯，而从端口2发起的连接会被拒绝。这就是一个有状态防火墙的应用，试想一下，无状态防火墙该如何实现类似的功能？

需要注意的是，如果packet本身一个fragment packet，那么在进入CT前，packet会重新聚合，输出时再重新拆分。因为CT模块需要读取数据包的完整数据来判断当前连接的状态。听起来有点傲娇，现实就是这样。很明显，MTU较小时，fragment packet的可能性更大，而性能将受到很大的影响。有关MTU对网络性能的影响，在 解读Mirantis最新的Neutron性能测试也有提到过。

Conntrack Entries

前面说过，CT功能是通过conntrack实现的，哪先看看conntrack有什么输出，具体来说看看conntrack表项（entry），它们可以在通过sudo conntrack –L 命令得到。更多相关的命令在：Conntrack tool页面。

```
tcp     6 117 SYN_SENT src=192.168.1.6 dst=192.168.1.9 sport=32775
        dort=22 [UNREPLIED] src=192.168.1.9 dst=192.168.1.6 sport=22
        dport=32775 [ASSURED] use=2
```

这条entry包含了packet所有的L3信息。首先是TCP packet。6代表了TCP的协议号。117是entry的TTL，117秒之后，如果没有别的数据包来更新这个entry，那它将会被删除。也就是说CT模块认为当前的connection已经关闭。如果有别的数据包更新这个entry，那它将被reset回默认值。SYN_SENT 是TCP协议的状态，接下来的一些没什么好说的。UNREPLIED，表明这个包还没有被回复。之后跟的是期待的回复包的信息，以及收到了回复包之后entry对应的状态。ASSURED的entry，在CT模块达到最大可跟踪的连接数之后，不会被删除，而其他的entry会被删除。最大可跟踪的连接数是可以配置，通常与内存大小有关。

Conntrack entry，是在发现了一个NEW的connection，并提交（commit）到CT之后，才会出现。

接下来看看CT是怎么处理几个协议的状态的。

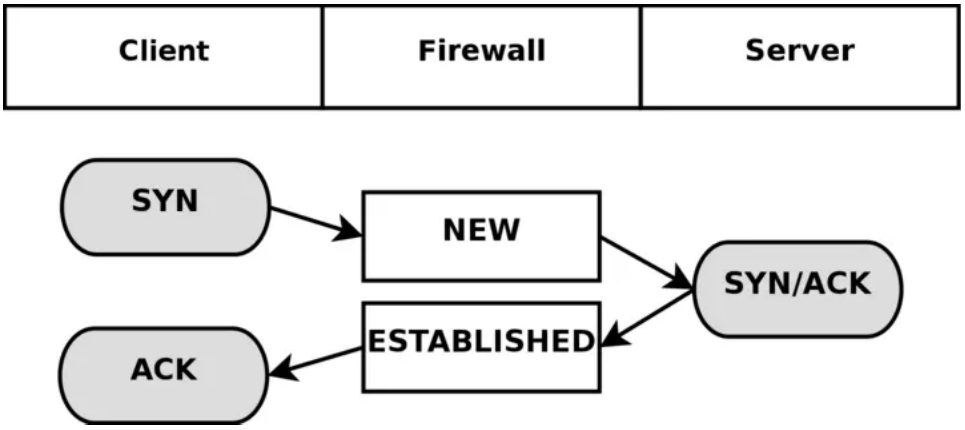
TCP

前面说过CT模块就是实现了connection state的状态机。而TCP本身是一个有状态的协议，有很多状态，如RFC793指出一样。但是CT模块并不会将所有的TCP状态输出。TCP的连接建立需要3-

way握手，之后才可以认为TCP连接的状态时ESTABLISHED。这方面的内容网上很多，知乎上也很多，就不再多说了。当TCP client 发出SYN包，CT模块会认为这是个NEW的connection，一旦TCP server返回了SYN/ACK，CT模块就会这个connection是ESTABLISHED。注意，这里的connection是CT模块概念中的connection，而不是TCP connection。为什么会是这样？先看看不这样会怎样？

假设CT模块一直要到TCP connection是ESTABLISHED 才认为自己管理的connection是ESTABLISHED。那对于TCP server返回的SYN/ACK，CT模块会仍然认为这是一个NEW的connection，回到前面的小例子，端口1是client，端口2是server，也就是说还需要允许端口2的NEW packet传向端口1才能建立TCP connection。那该怎么阻挡由端口2发起的连接呢？

似乎没有什么好办法，所以CT模块不管TCP的状态，收到SYN/ACK就认为connection是ESTABLISHED。这样，再套用上面的小例子，还是能达到应有的控制效果。



对于CT模块来说，一个connection怎样才算是断开？这依赖于前面提到过的TTL，TTL超时了，connection会被删除。之后相关的packet会认为属于新的connection。而这里的TTL是与TCP connection的状态相关的，每个TCP connection状态，都有一个默认的TTL值。

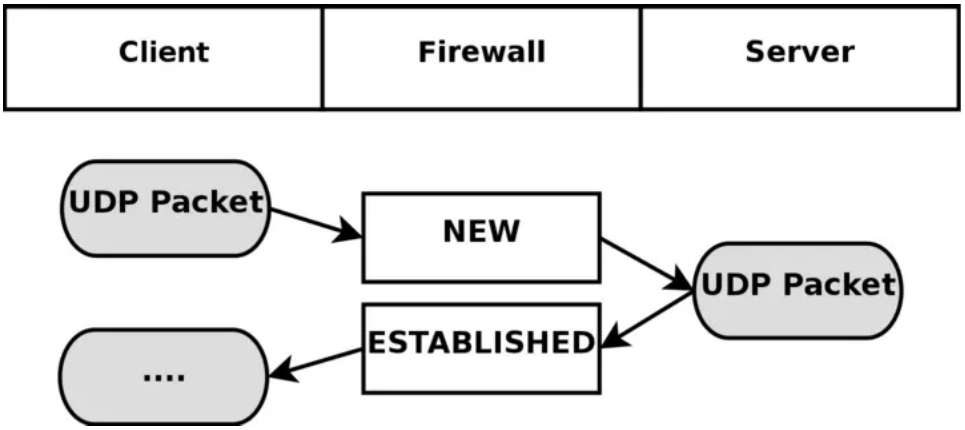
State	Timeout value
NONE	30 minutes
ESTABLISHED	5 days
SYN_SENT	2 minutes
SYN_RECV	60 seconds
FIN_WAIT	2 minutes
TIME_WAIT	2 minutes
CLOSE	10 seconds
CLOSE_WAIT	12 hours

LAST_ACK	30 seconds
LISTEN	2 minutes

UDP

UDP本身是无状态连接，它没有连接建立和断开的过程，它也没有序列号。两个UDP包是无法确定它们之间的先后顺序。但是对于CT模块不会因为UDP本身的无状态而认为UDP对应的connection也是无状态（否则CT模块怎么存活，只为有状态的IP连接服务？CT模块是为所有的网络连接服务的）

CT模块对UDP connection的处理与TCP connection类似，client发出的第一个包，认为connection是NEW状态，server返回第一个包，认为connection是ESTABLISHED。其实到这里，大概就能明白CT模块是如何工作的了吧。只要是第一个packet，就认为connection是NEW，收到了第一个合法的返回，就认为connection是ESTABLISHED。



来看一下UDP connection对应的conntrack entry。

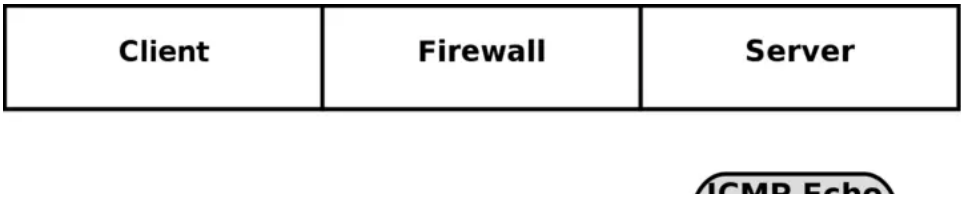
```
udp     17 20 src=192.168.1.2 dst=192.168.1.5 sport=137 dport=1025
        [UNREPLIED] src=192.168.1.5 dst=192.168.1.2 sport=1025
        dport=137 use=1
```

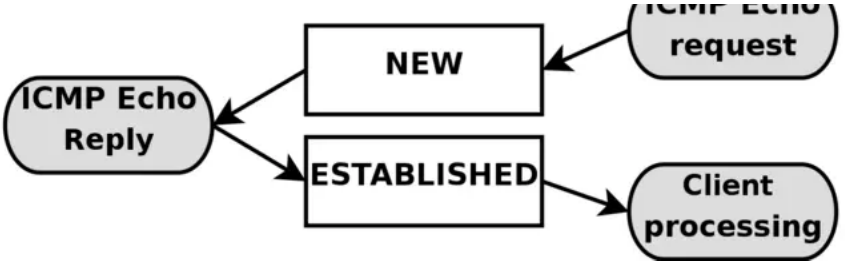
与TCP对应的entry不一样，UDP的entry没有自身的状态信息（例如上面的SYN_SENT），因为UDP本身就没有这些东西。因为UDP状态的简单，CT模块对于UDP的TTL（connection断开的判断机制）默认值是：第一个UDP包创建的NEW状态的UDP connection，TTL是30；当connection变成ESTABLISHED，其TTL默认值是180。

UDP连接本身没有TCP连接复杂，但是对于CT模块来说，两者本质还是一样的。

ICMP

ICMP全称是Internet Control Message Protocol，它主要用来传输控制信息，因此不应该有任何ICMP connection。ICMP有4种数据包可以产生回复，但是目前最常用的只有ICMP echo request。对于有回复的ICMP，CT模块也对connection生成NEW 和 ESTABLISHED状态。





来看看ICMP在conntrack中的表项。

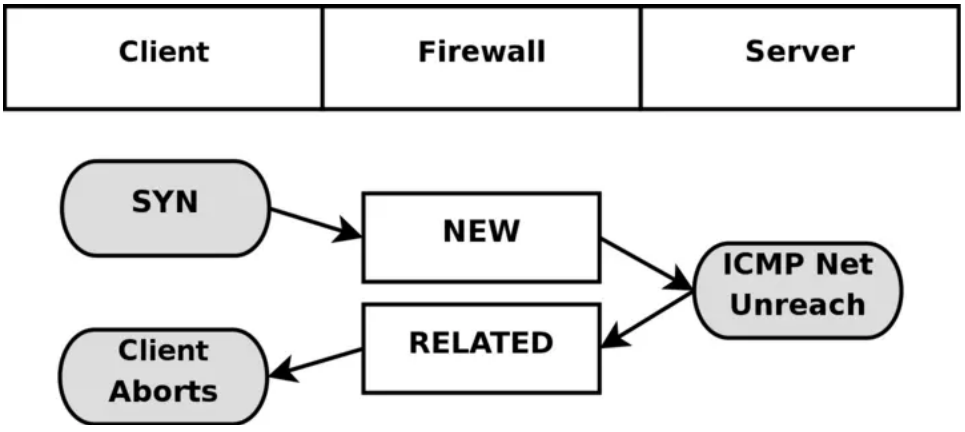
```
icmp     1 25 src=192.168.1.6 dst=192.168.1.10 type=8 code=0
         id=33029 [UNREPLIED] src=192.168.1.10 dst=192.168.1.6
         type=0 code=0 id=33029 use=1
```

这里没有了sport和dport，但是多了type，code和id。这实际上是匹配ICMP echo请求。前面说过，对于CT模块来说，commit connection只是ESTABLISHED状态的必要条件，另一个必要条件就是从对端返回的数据包是正确的。而这里的type，code，id就是用来判断返回的数据包是否是正确的。只有匹配的ICMP echo reply，CT模块才认为connection是ESTABLISHED状态。这样相应的防火墙能进行过滤。但是由于ICMP echo reply之后，肯定不会再有别的相关数据包，所以收到了ICMP echo reply之后，CT模块会立即删除对应的connection，也就是conntrack entry。ICMP echo request默认有30秒的TTL。

除了ICMP echo，ICMP还常用来通知client一些额外的信息，例如ICMP time exceed，或者连接不可达。这个时候，ICMP数据不是由client发出，而是从remote端发出，这里的remote端可能是client的对端，也可能就是中途的某个网络设备。

从client的角度来看，这样的ICMP数据是一个RELATED的连接状态。因为它跟现有的某个连接是有关联的，并且它能告知当前client该如何处理现有的连接。

以TCP connection为例，client向server发出建立连接的请求。但是server没有开放响应的端口，因此server返回ICMP network unreachable。这就是原有TCP connection的一个RELATED连接，而client在收到了这个ICMP信息之后，CT模块不会再傻傻的等到超时，而是直接放弃TCP连接，并且在CT模块中删除响应的connection。



DEFAULT connections

看了前面几种协议，CT模块的工作方式大概也清楚了。

- CT模块输出的状态跟协议关系不大，第一个包认为connection是NEW，收到第一个合法的数据包认为connection是ESTABLISHED。RELATED状态的connection必然跟现有的某一个connection有关联。
- CT模块与协议相关的地方在于TTL的默认值，每个协议的每个状态都对应一个TTL的默认值。

那么对于不能识别的协议，其实只要定义好了默认的TTL，CT模块也能工作。CT模块对于不能识

别的协议，默认的TTL都是600，也就是10分钟。相应的参数在这里：Ipsysctl tutorial

最后

现实中的协议连接，并不是你来我往这么简单，例如CT模块对FTP协议的处理就相对复杂。不过总的来说，CT模块能够识别大部分协议，并且输出正确的状态，结合CT模块，在OpenFlow中也能较为简单的实现有状态的防火墙。使得OpenFlow based SDN又更加有竞争力了。