**实验环境：**

Ubuntu环境：

Python 版本2.7.14

需要scapy，numpy，trafgen

安装scapy：sudo apt-get install python-scapy

安装numpy：sudo apt-get install python-numpy

trafgen是netsniff-ng工具的一个模块。

安装netsniff-ng：

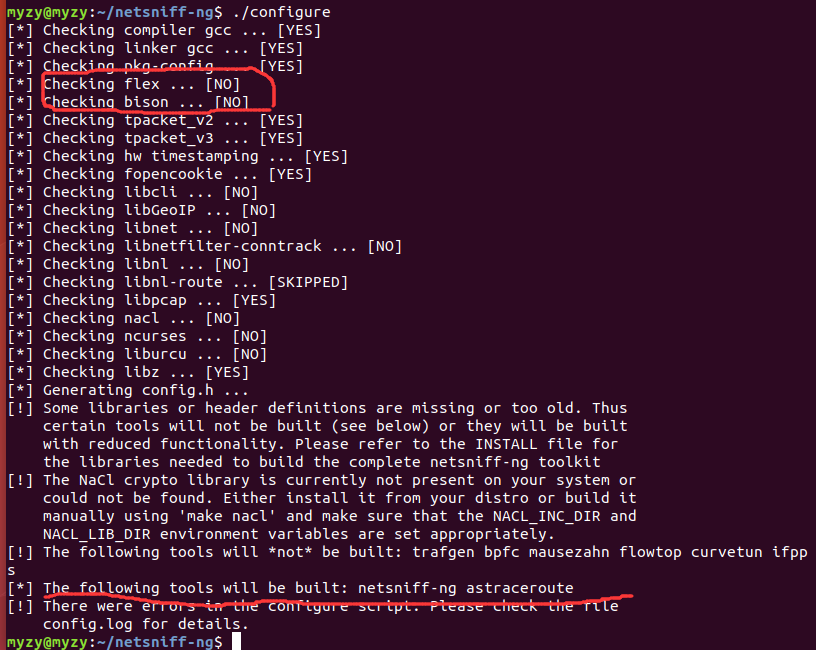
在根目录运行：

git clone <https://github.com/borkmann/netsniff-ng>

复制完毕后，执行一下指令

cd netsniff-ng

./configure



比较如图所划的红线，如果trafgen没有被生成，则运行以下两个指令

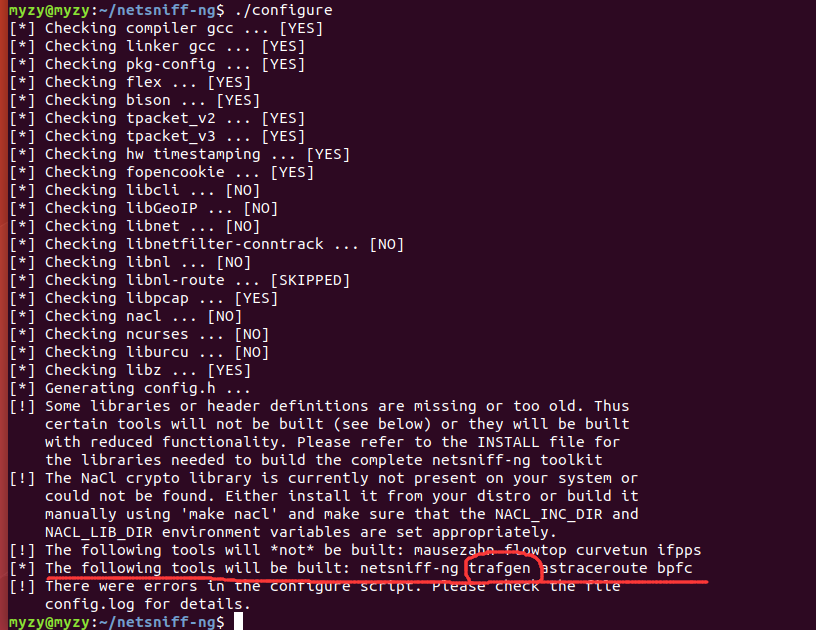
sudo apt-get install flex

sudo apt-get install bison

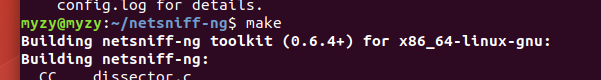
再次执行

./configure

直到出现以下图片内容：



在netsniff-ng目录下，输入make安装模块。



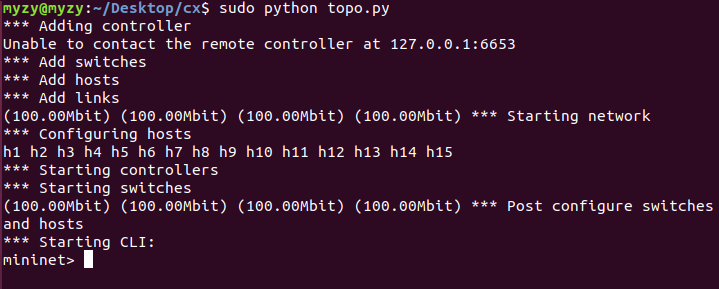
Windows环境：

采用Anaconda3这个python集成环境。其包含了sklearn，numpy和matplotlib库这些本次实验要用到的python库。Python版本3.6.1。

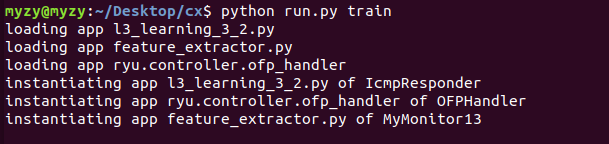
**实验步骤：**

1．训练阶段：

1．1搭建拓扑：sudo python topo.py



1．2运行控制器：python run.py train



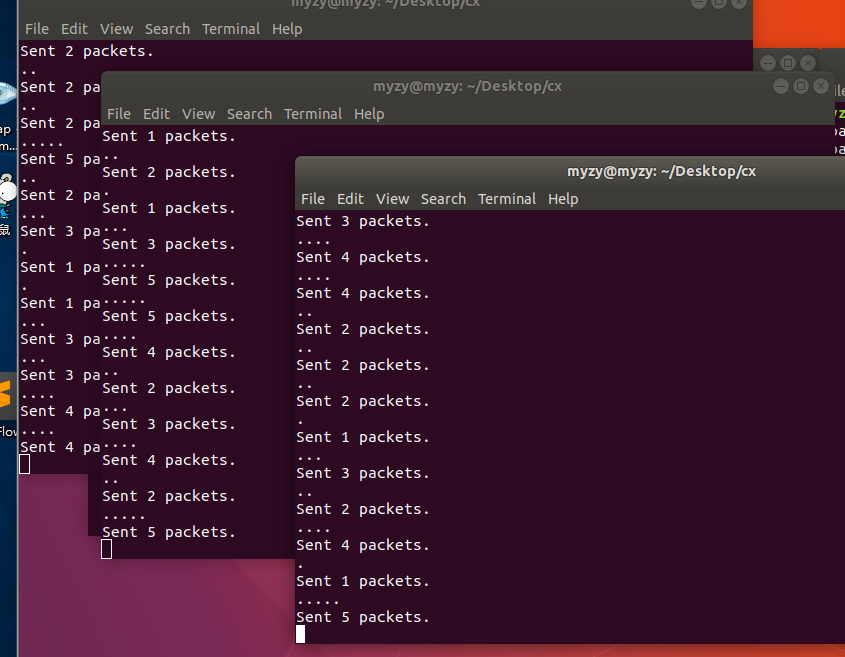
1．3注入流量：

sudo python Rs1.py

sudo python Rs2.py

sudo python Rs3.py

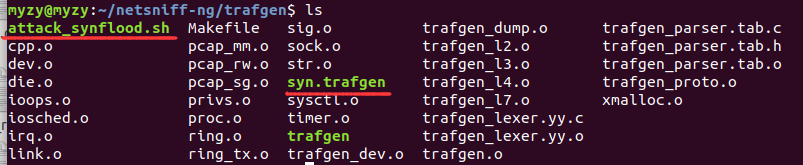
这三个python脚本都是模拟主机间的数据传输，运行后效果如图：



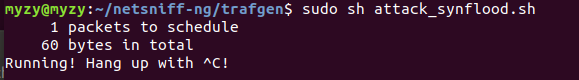
1．4注入攻击流量

在收集一定时间的正常数据后，开始注入DDoS流量。

把attack\_synflood.sh和syn.trafgen文件复制到netsniff-ng/trafgen目录下

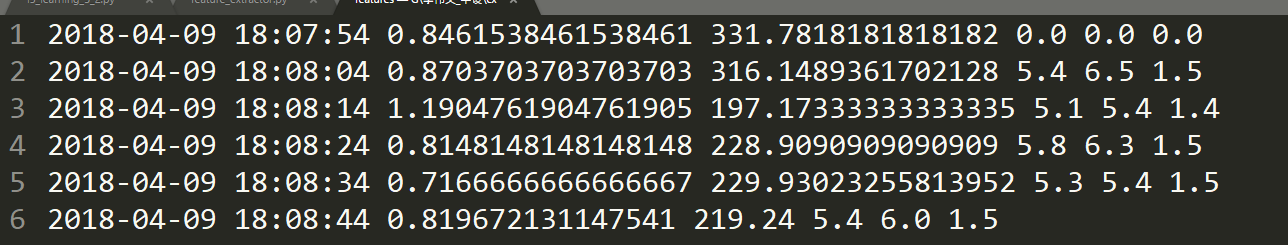


运行 sudo sh attack\_synflood.sh



本次实验DDoS攻击流量输入时间为230s，训练阶段总时间为3311s。

打开features文件可以查看记录：



这里的数据依次是记录的时间，流平均包数，流包平均比特数，端口增速，流增长速率，源ip增速。

1．5数据训练

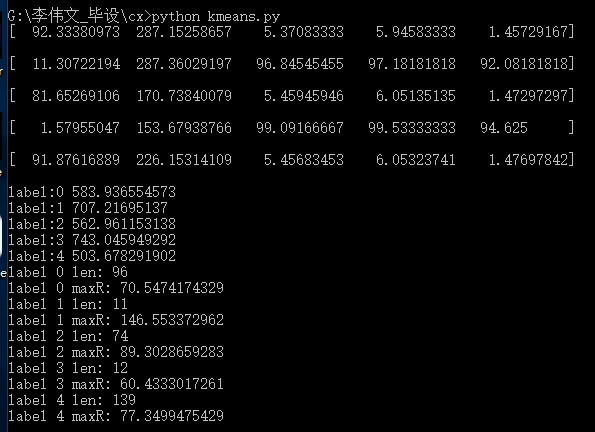
在windows主机中，

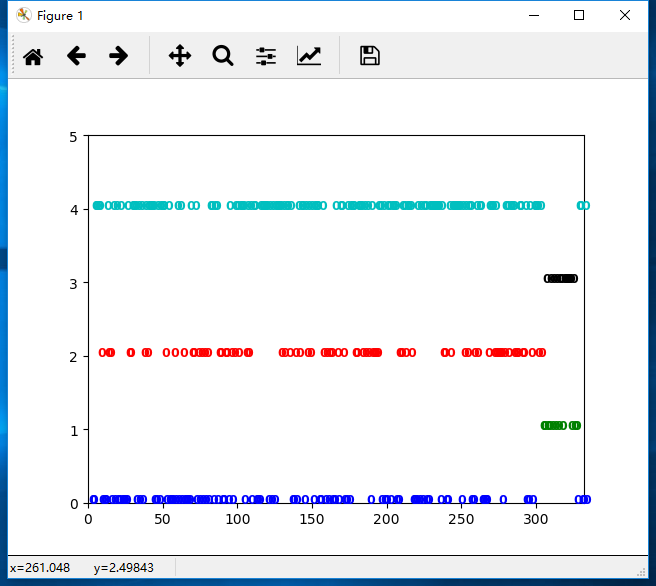
在得到训练结果后，把features文件拷贝到有kmeans.py文件的目录下，进行数据训练。



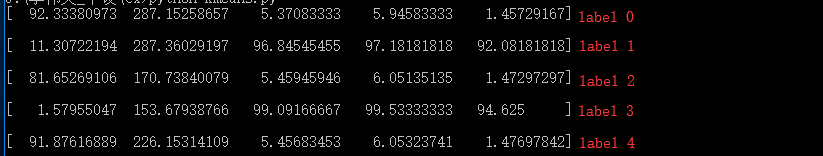
命令行输入python kmeans.py

得到如图结果：

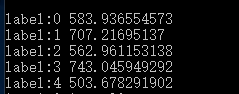




其中，命令行显示的依次是label 0 到label 4的聚类中心：



每个簇的相异度：



具体为：

label:0 583.936554573

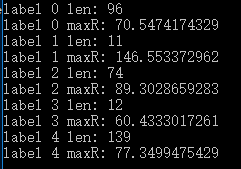
label:1 707.21695137

label:2 562.961153138

label:3 743.045949292

label:4 503.678291902

每个簇的对象个数和它的最大半径：



即

label 0 len: 96

label 0 maxR: 70.5474174329

label 1 len: 11

label 1 maxR: 146.553372962

label 2 len: 74

label 2 maxR: 89.3028659283

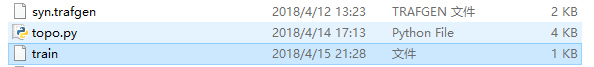
label 3 len: 12

label 3 maxR: 60.4333017261

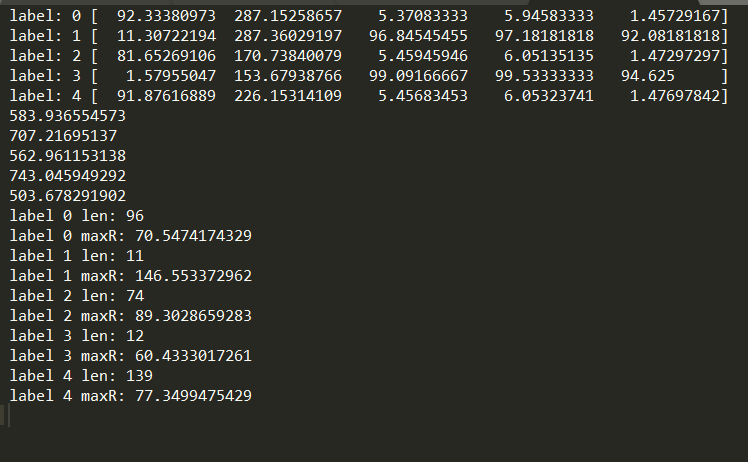
label 4 len: 139

label 4 maxR: 77.3499475429

在数据训练后，会生成train文件以记录训练结果：



打开train文本可以查看以上数据：



从数据收集阶段可以知道正常数据有309个，异常数据有23个，总共332个特征样本。

根据定义，把聚类结果按相异度由小到大排列：

f(label 4) < f(label 2) < f(label 0) < f(label 1) < f(label 3)

计算个数

|label 4| + |label 2| + |label 0|= 139 + 89 + 96 = 309 =< 309

符合定义要求，因此定义簇4，簇2，簇0为正常簇。

提取簇4，簇2，簇0的聚类中心和最大半径作为检测向量。

即













具体数字如下：

label: 4 [ 91.87616889 226.15314109 5.45683453 6.05323741 1.47697842]

label: 2 [ 81.65269106 170.73840079 5.45945946 6.05135135 1.47297297]

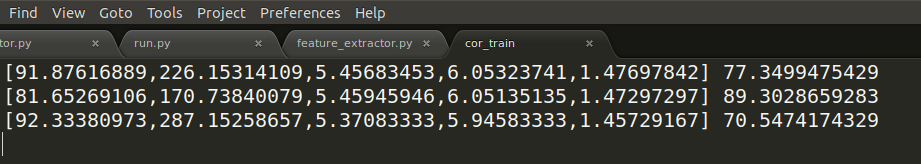
label: 0 [ 92.33380973 287.15258657 5.37083333 5.94583333 1.45729167]

label 4 maxR: 77.3499475429

label 2 maxR: 89.3028659283

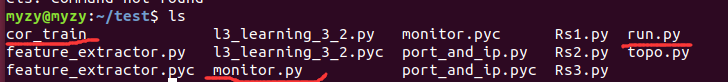
label 0 maxR: 70.5474174329

上述数据记录到cor\_train文件中：



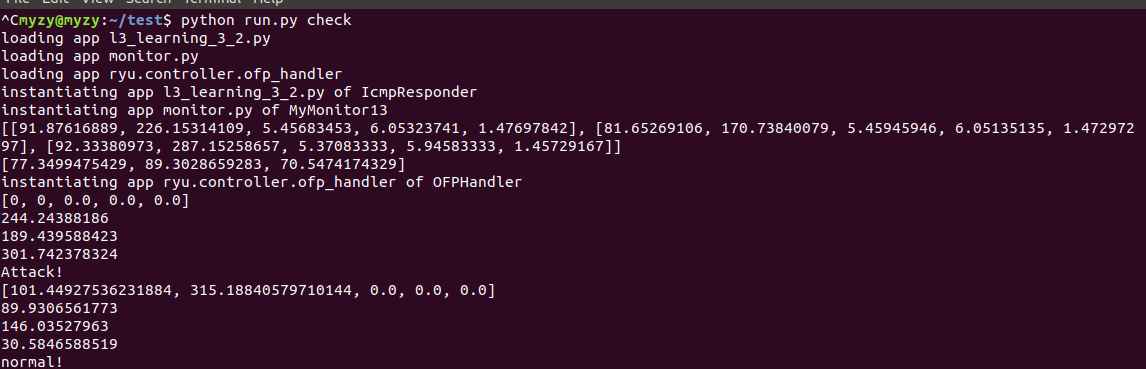
2．检测阶段：

把训练阶段得到cor\_train文件放到运行检测程序的目录下：



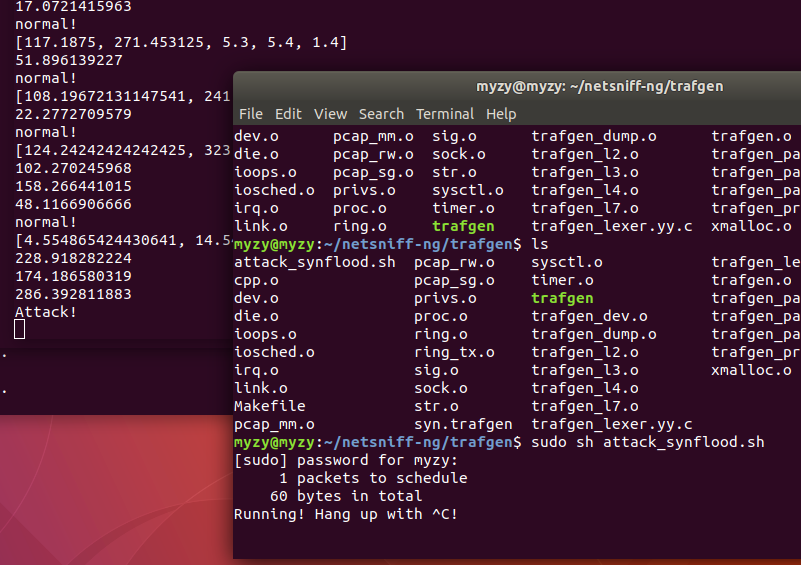
输入python run.py check

检测阶段能看到检测结果

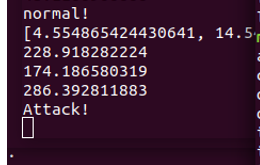


在检测一定时间的正常流量后，注入攻击流量，用于检测。

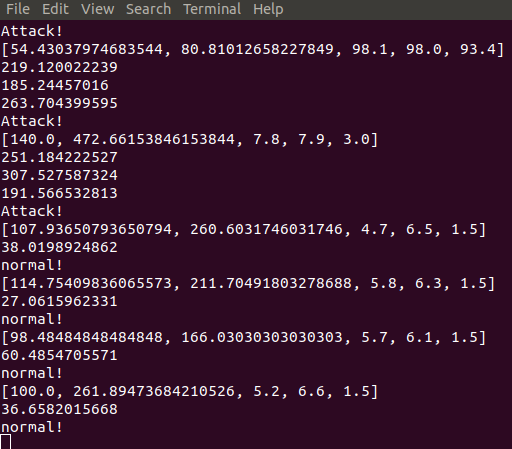
sudo sh attack\_synflood.sh



从命令行窗口能够看到已经发现攻击了

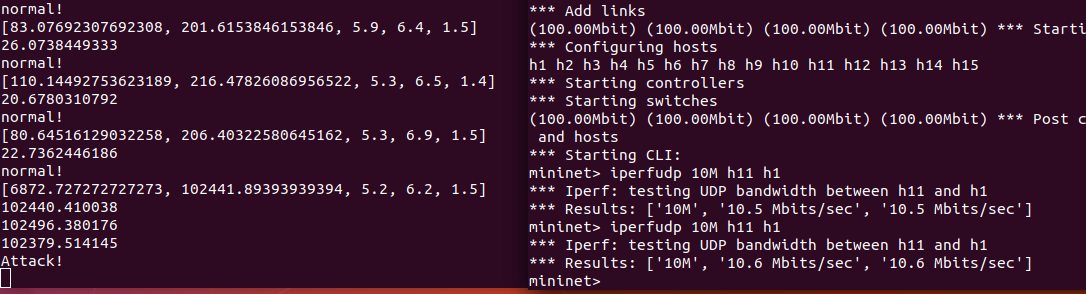


在停止攻击后，能看到恢复正常



进入udpflood攻击检测，

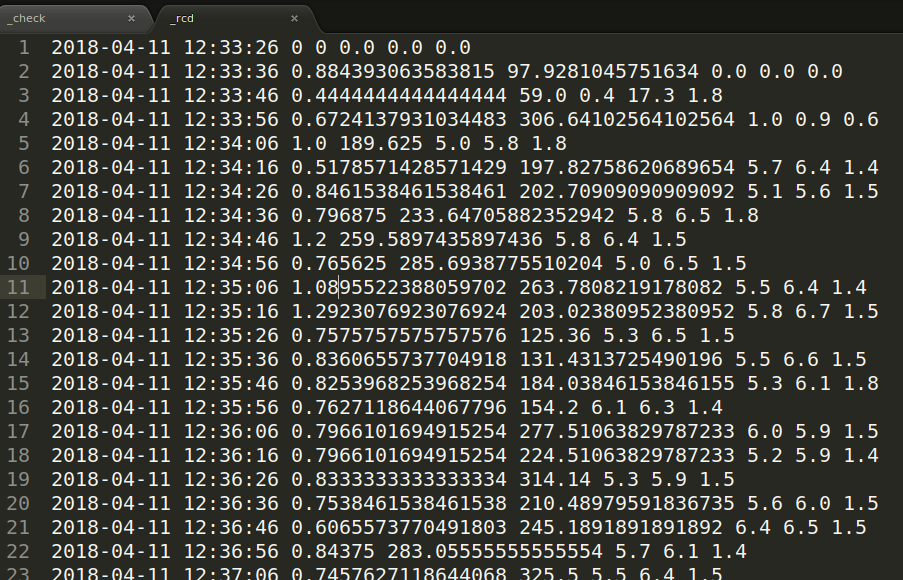
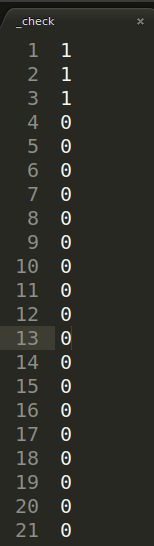
在mininet窗口运行 iperfudp 10M h11 h1（每10s执行两次）



上图是执行iperfudp之后的效果，左边是检测结果，从图中能发现在发生udpfood时候，检测效果更明显。

3结果分析

在进行攻击检测时，会产生两个文件记录，分别是 ‘\_check’ 和 ‘\_rcd’文件



\_check文件记录着每次识别的结果，1表示攻击，0表示正常。

\_rcd文件记录着每次收集的向量和时间。

正常流量从2018-04-11 12：33：26开始到结束，

synflood攻击流量从 2018-04-11 12:50:24开始注入，到2018-04-11 13:07:00结束，

Udpflood攻击流量从2018-04-11 13:18:50开始注入，到2018-04-11 13:24:23结束。

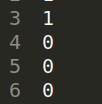
根据时间，统计出正常流量的数据和synflood的数据：

正常流量数据从 记录3到记录102 共100条

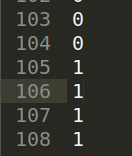
synflood流量数据从 记录103 到记录 202 共100条

其对应的\_check文件记录着对应的数据是否攻击流量：

\_check文件里记录3到102中，有1个被识别出是攻击，其余都是正常，即99正常，1攻击。



而记录103到202中，有两个被识别为正常



检测率 = 攻击数据被正确检测出的个数/总攻击数据的个数

= 98 / 100 = 98%

误报率 = 正常数据被错误检测出的个数/总正常数据的个数

= 1/100 = 1%

错误率 = 被错误识别的个数/总数据的个数

= （1 + 2）/ 200 = 1.5%

正常流量数据从 记录223到272，共50条

Udpflood流量数据从 记录273 到记录292，共20条

\_check文件记录223到272中，50个正常

记录273到292中有1个正常，19个攻击

检测率 = 攻击数据被正确检测出的个数/总攻击数据的个数

= 19/20 = 95%

误报率 = 正常数据被错误检测出的个数/总正常数据的个数

= 0/100 = 0%

错误率 = 被错误识别的个数/总数据的个数

= 1/70 = 1.4%