软件学报ISSN 1000-9825, CODEN RUXUEW E-mail: jos@iscas.ac.cn

*Journal of Software*, [doi: 10.13328/j.cnki.jos.000000] http://www.jos.org.cn

©中国科学院软件研究所版权所有. Tel: +86-10-62562563

使用机器学习以及卷积神经网络对以太坊流量进行识别分析[[1]](#footnote-1)\*

舒卓卓1

1(东南大学 网络空间与安全学院,江苏 南京 210000)

摘 要: 近年来以太坊快速发展的同时伴随了各种安全问题的出现,当前大部分的研究主要针对以太坊的探索,例如以太坊的攻击模式,以太坊的设计漏洞,针对以太坊的流量的研究还是一个较大的空白.本文针对以太坊流量在正常流量的识别,提出了两类方法进行实践研究：机器学习方法和卷积神经网络方法.首先搜集以太坊流量以VPN-nonVPN数据集作为背景流量,然后进行预处理,根据两种方法构建数据集.接着在数据集上进行方法的验证和评估.最后我们发现随机森林算法拥有更好的综合性能,能够快速准确识别正常流量中的以太坊流量.最后我们提出研究存在的一定问题并且提出了改进意见.

关键词: 以太坊;流量识别;机器学习;神经网络

中图法分类号: TP311

中文引用格式: 舒卓卓.使用机器学习以及卷积神经网络对以太坊流量进行识别分析.软件学报,2021,32(7).

英文引用格式: Shu ZZ. Identifying and analyzing ethereum traffic using machine learning and convolutional neural networks. Ruan Jian Xue Bao/Journal of Software, 2021 (in Chinese).

Identifying and analyzing ethereum traffic using machine learning and convolutional neural networks

SHU Zhuo-Zhuo1

1(School of Cyber Science and Engineering, Southeast University, Nanjing 210000, China)

**Abstract**: In recent years, the rapid development of ethereum is accompanied by various security problems, most of the current research is mainly focused on the exploration of ethereum, such as the attack pattern of ethereum, the design vulnerability of ethereum, and the research on the traffic of ethereum is still a big gap. In this paper, we propose two types of methods for the identification of ethereum traffic in normal traffic for practical research: machine learning methods and convolutional neural network methods. Firstly, we collect Ethereum traffic with VPN-nonVPN dataset as background traffic, and then preprocess it to construct the dataset according to the two methods. Then we validate and evaluate the methods on the dataset. And we found that the random forest algorithm has better comprehensive performance and can identify the ethereum traffic among normal traffic quickly and accurately. Finally, we present some problems of the study and suggest improvements.

**Key words**: ethereum; traffic recognition; machine learning; neural networks

区块链是一种由多方共同维护的记账技术,使用密码学来保证传输和访问的安全.它能够实现一致的数据存储,使其具有被篡改和防止抵赖的挑战性,也被称为分布式账本技术.区块链技术为进一步提高互联网的信任,安全和效率提供了一个新的解决方案.

区块链技术最早由中本聪提出,随后以加密数字货币为首的各类区块链产业迅速发展. 2020年中国区块链产业规模已达48.5亿元,增长率为48.5％.随着区块链的快速发展,区块链中潜在的监管问题也逐渐显现出来.首先,区块链数字货币为洗钱,勒索等犯罪活动提供了安全,稳定的资金渠道.其次,区块链数字货币使得跨国转移资金更加容易.最后,由于区块链上的数据存在不可更改的特性,一旦不法分子将敏感信息和虚假信息上传到脸上,会对国家安全和社会稳定造成重大影响.如果不对区块链进行有效地监管,放纵区块链的野蛮生长,将严重危害国家安全和稳定,给网络安全监管带来巨大威胁和挑战.

作为区块链的代表性应用,比特币[1]通过脚本引擎实现区块链应用开发,受限于脚本语言的表达能力,在支持复杂合约开发方面存在一定的困难.以太坊基于以太坊虚拟机来实现交易状态机[2],智能合约[3]的使用使得以太坊具有更多的开发潜力.同时以太坊还克服了比特币非图灵完备的困难,其使用图灵完备的编程语言,实现了对记录任意信息和执行任意函数的支持.与比特币等其他区块链实现方式相比,以太坊可以更好地支持区块链分布式应用开发,研究价值更高.

虽然区块链的发展迅速,但是对区块链的监管的研究却存在一定的落后.现有对区块链安全问题的研究主要是针对区块链技术的探索,如区块链攻击模式,区块链设计漏洞,区块链应用方向等.当前区块链使用最为广泛的依然是挖取加密数字货币,挖取加密数字货币造成了大量的电力资源的浪费,然而由于区块链的去中心化的设计,很难直接去追查相关用户.如果能够有效测量和分析以太坊网络流量,探索以太坊安全监管方案,不但能够减少电力资源的损耗,同时还能促进区块链的良性发展.

本文将通过采集以太坊流量,构建以太坊流量数据集,使用机器学习以及卷积神经网络的方法对以太坊流量和正常流量进行识别区分.

本文第1节介绍以太坊的相关工作.第2节介绍以太坊流量搜集以及实验数据集构造.第3节介绍使用的实验方法.第4节通过对比实验验证了使用方法的有效性.第5节总结全文,提出实验有待改进的部分.

# 以太坊相关工作

现有的区块链网络流量研究大致可分为区块链网络拓扑结构分析,去中心化应用流量分类[4],结合网络流量研究比特币去匿名化方法.

区块链网络拓扑结构的分析主要针对区块链底层网络拓扑结构,一般采用被动或主动的方法来检测区块链节点.被动法在中间网络节点收集区块链流量,通过流量源-主机IP地址信息构建区块链网络拓扑.Neudecker[5]通过分析比特币泛滥的p2p网络信息接收的时间间隔序列,推测出可能是比特币节点的网络节点.Li[6]通过从ISP提供的NetFlow流量数据中收集区块链流量,构建区块链网络拓扑.然后他识别以太坊流量,获得以太坊节点信息,并在此基础上构建以太坊网络拓扑.主动式方法通过主动加入区块链网络,检测区块链节点信息,构建区块链拓扑.文献[7]通过比特币地址可以请求相应IP地址的漏洞,主动探测比特币节点信息.Kim[8]设计了节点检测器NodeFinder,通过修改以太坊客户端来探测以太坊节点,并利用检测结果来构建以太坊拓扑结构.

去中心化应用的流量分类问题主要是针对部署在以太坊上的DApps的识别方法.DApps认为以太坊是一个分布式服务器,并通过TLS协议与这个分布式服务器进行通信.DApp流量的流量分类问题本质上是对TLS加密流量的研究.沈蒙老师使用核函数来降低DApp流量特征的维度[9],并得到降低后的特征向量来比较SVM和KNN等机器学习方法的分类效果.他还使用图神经网络对DApp流量进行分类,获得了比机器学习方法更好的分类结果.

比特币去匿名化的问题主要是对比特币链和比特币网络流量的信息分析.比特币节点地址是通过椭圆曲线加密得到的[10],只有私钥所有者才能解锁,但节点地址的交易对链上所有节点都是公开的.由于比特币网络层使用明文传输数据,对于任何交易流量,比特币的去匿名化可以通过找到交易的发起节点来实现.Biryukov[11]通过在目标节点部署相邻的节点,将所有相邻节点的第一笔交易视为目标节点发出的交易,将交易的比特币地址与IP绑定,实现去匿名化.然而,以太坊选择在传输前对数据进行加密,这使得识别流量是否包含交易行为具有挑战性.更不可能从流量中获得交易的具体内容.因此,用同样的方法来定位以太坊中交易开始的节点是不现实的.

综上所述,目前对以太坊网络安全流量的分析研究工作还处于起步阶段.对以太坊行为的研究可以为以太坊溯源和以太坊定点监管提供实际帮助.同时,现有的研究工作忽略了以太坊行为流量分类的重要研究价值.

# 以太坊流量搜集及数据集构造

## 以太坊流量搜集

由于以太坊流量分析研究仍处于起步阶段.尚且缺乏公开可用的以太坊流量数据集.因此.我们需要自己搜集以太坊流量.在这里我们参考了Kim设计了节点检测器NodeFinder.Kim[8]通过研究以太坊底层协议簇Devp2p[12],深入理解了以太坊连接建立以及节点交互的过程.Devp2p协议簇主要分为三类:一,是用于节点发现的NDP（Node Discovery Protocol）协议.二,是用于建立加密通信的RLPx协议.三,是定义内容传输阶段各种数据格式的ETH（Ethereumeum Wire Protocol）协议.同时作者就以太坊官方端Geth,非官方的客户端Parity进行了研究.作者有了以下发现:1,Geth和Parity在几分钟内就达到了它们默认的peer最大数量限制,收敛后连接peer数量存在波动,网络扫描器提供短暂的机会窗口,使其能够在一段时间内连接到它们.2,一旦一个节点与区块链同步,TRANSACTIONS消息就会占网络I/O的主要.但Geth向网络发送的消息比例高于Parity.因为Geth向其所有peer广播交易,而Parity只将交易发送到个对等体.



图1 以太坊Devp2p协议簇

基于以上的各种发现,作者修改了Geth客户端设置出了NodeFinder.NoderFinder秉持了以下几个设计原则: 1,NodeFinder忽略了最大的peer连接限制,以便持续执行发现并尽量减少peer断开连接.2,NodeFinder一旦通过GET\_BLOCK\_HEADERS检查了DAO分叉区块,就会与peer断开连接.这一阶段经历了DEVp2p握手,Ethereumeum握手和DAO分叉块验证后立即进行.3,为了定期地重新连接到之前发现的peer,NodeFinder将所有的地址和它们最后拨出的时间戳存储到本地数据库中.4,当peer发送HELLO,DISCONNECT和STATUS消息时,NodeFinder将每个消息的解码内容记录在日志中.peer连接产生的每条日志信息都有以下内容:时间戳,peer的节点ID,IP地址,端口,连接类型(动态拨号,静态拨号),连接延迟和连接时间.

基于以上内容,NoderFinder能够有效和各种以太坊节点进行连接交互,但由于其本意在发现以太坊节点的可达性,其没有与以太坊节点进行长时间的连接.因此修改了用于探测以太坊节点的工具NodeFinder,修改了其与以太坊节点保持连接的时间,用于搜集以太坊流量.我们在主机上运行修改后的NoderFinder以与被探测到的以太坊节点进行通信.同时在中间路由器上捕获以太坊流量.并采用公开数据集VPN-nonVPN[47]中各种应用流量作为背景流量.为了方便后续研究,我们将搜集的以太坊流量分为了TCP流量以及UDP流量,流量搜集配置以及搜集的部分流量情况如图所示.



图2 实验环节

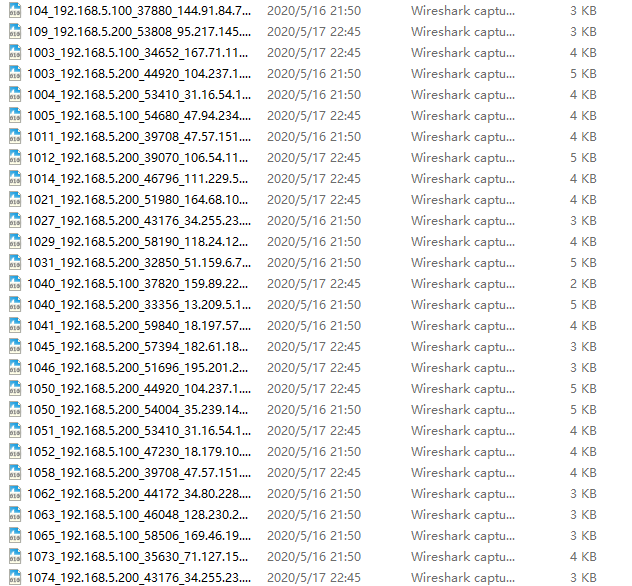


图3 搜集的部分以太坊流量

## 以太坊数据预处理及数据集构造

由于我们使用机器学习以及神经网络两种方法进行以太坊流量的识别工作,因此我们需要对原始的PCAP数据包进行两次处理,分别生成机器学习所使用的的数据集以及神经网络所使用的数据集.

在机器学习数据集的生成上,我们使用了CICFlowMeter4.0[13]工具来完成.CICFlowMeter是一个开源的工具,可以从pcap文件中生成双向流,并从这些流量中提取特征,将特征写入CSV文件中.CICFlowMeter对流的切分是分为TCP以及UDP两类,其中TCP流通常在连接断开时切分（通过FIN包来判断）,而UDP流则通过流超时来切分.默认设置时CICFlowMeter统计一个流中的统计信息作为提取的特征.且统计的特征都分前后向,规定由源地址到目的地址为正向,目的地址到源地址为反向,为每个流构建一个标志叫Flow ID:192.168.31.100-183.232.231.174-46927-443-6,由源地址、目的地址、协议号组成.CICFlowMeter默认设置共计生成了76维度的特征,所选部分特征如下图所示:

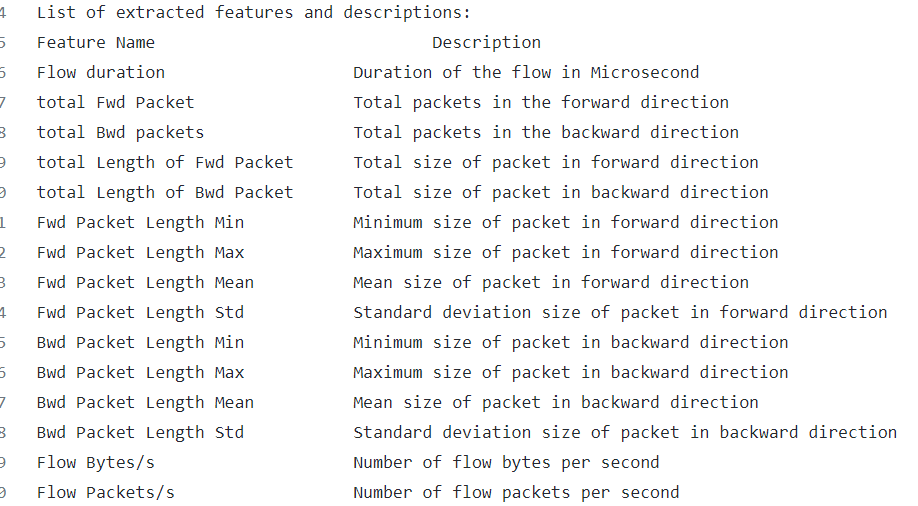


图4 部分选取的特征

同时CICFlowMeter提供了批处理命令,我们可以一次性处理多个PCAP文件生成对应的CSV文件.我们将生成的CSV文件进行合并,删除了诸如源宿IP,时间戳等信息,最后根据TCP和UDP分别生成了两个CSV文件,用作实验所需的数据集.数据集情况如下表所示:

表1 机器学习数据集

|  |  |  |
| --- | --- | --- |
| 序号 | 流量类型 | 条数 |
| 1 | 以太坊TCP流量 | 9900条 |
| 2 | 非以太坊TCP流量 | 10000条 |
| 3 | 以太坊UDP流量 | 21000条 |
| 4 | 非以太坊UDP流量 | 9000条 |

除此之外,我们还需要构建用于神经网络的数据集.我们的目的是将原始pcap格式的流量数据转换为 CNN输入的IDX的数据格式的过程,在此我们使用USTC-TFC2016[14]所述的工具集来完成.整个流程主要包括四个步骤:其分别是流量分割、流量清除、图像生成和 IDX转换.如图所示:



图5 神经网络数据集流程

第一步:流量分割,该步将连续的原始流量分割成多个离散的流量,输入的数据为pcap文件.如果表示类型是 Flow + All 或 Session + All,输出数据格式是 pcap.如果表示类型是 Flow + L7 或 Session + L7,输出数据为 bin文件.

第二步:流量筛选,该步骤首先进行流量的匿名化,在数据链路层和IP层中分别随机化MAC地址和IP地址.在流量完成匿名化操作之后,我们执行文件清理操作.在这一流程中,我们会处理没有应用层的数据包以及重复的数据包文件.

第三步图像生成,这一步首先将所有文件修剪成统一长度.如果文件大小大于784字节,它将被裁剪到到784字节.如果文件大小短于784字节,我们则在最后使用0填充将其补充到784字节.然后,具有相同大小的文件将被转换为灰度图.原始文件的每个字节代表一个像素,例如0x00是黑色,0xff是白色.

第4步IDX转换,这一步将图像转换为IDX 格式的文件.一个IDX文件包含一组图像的所有像素和统计信息.IDX格式是机器学习领域的一种常见文件格式.

表2 神经网络数据集

|  |  |  |
| --- | --- | --- |
| 序号 | 流量类型 | 条数 |
| 1 | TCP流量 | 16952条 |
| 2 | UDP流量 | 32065条 |

至此我们分别得到了用于机器学习的数据集,其格式为CSV,和用于神经网络的IDX文件.

# 实验方法设计

我们设计的实验方法分为机器学习以及神经网络两部分.

## 机器学习方法设计

由于我们使用了CICFlowMeter这一工具生成了基础数据集,数据集包含了每一条流中的包括源端口、宿端口、数据包时间间隔等在内的76种特征.显然,直接提取76种特征相当消耗时间,预处理时间过长不利于在实际情况中使用.同时如果将所提取的76种流量特征直接用于训练以太坊流量识别模型,可能会出现过拟合,训练时间过长,训练生成的模型过大等问题.因此,需要选择这76种特征中合适的特征作为输入训练模型的输入向量,在保证识别准确率的情况下还能兼顾训练时间较短的情况.

在此我们使用特征相关性来来进行特征的选择,特征相关性可以描述各个特征在用于分类时对系统的贡献程度.特征相关性可以用信息熵、信息增益和互信息来表示.信息熵可以描述随机变量的不确定性,而信息增益通过信息熵描述变量对分类系统提供的有效信息的多少,信息增益大的特征具有更强的分类能力,互信息则基于信息熵和信息增益提供了特征选择的依据.

我们计算由特征向量表示的流其每个特征的互信息.计算完毕后对以太坊UDP流量和TCP流量各选择其中互信息值最高的前12个特征,作为以太坊流量识别的特征集合.在提取了UDP流或TCP流的特征,构成流特征向量后,我们选择了四种主流的机器学习分类方法分别训练了以太坊流量识别模型,其分别是:

随机森林算法（RF）:随机森林算法是一种用于分类,回归和其他任务的整体学习方法,通过在训练时构造大量决策树并输出来完成分类、预测、回归等任务.随机决策森林改变了决策树过拟合的情况,训练的准确率较高.但是当存在较多的决策树时,随机森林算法占用的空间会变大,训练时间也会随之增加.随机森林经常被用作企业中的“黑匣子”模型,因为它们可以在很少配置的情况下针对各种数据生成合理的预测.

支持向量机算法（SVM）:支持向量机算法本质就是计算出分类边界的超屏幕方程的最优化问题.其优势在于可以利用设置核函数解决高维空间的分类问题,同时泛化能力优秀.但是其在大规模的分类问题上处理能力不佳,耗费时间远比其他算法长,而且算法本事分类效果绝大程度取决于参数的选择和核函数的设置.

K近邻算法（KNN）: K近邻算法本质为计算出样本与其他样本距离,通过选择与自己最近的k个样本对应的种类来判断自身所处的分类.其原理简单易懂,容易实现,可以处理多分类问题,不但可以在聚类上应用,也可以实现解决问题.但是算法依赖数据集的分布,在面对多维度问题的时候会产生“维度灾难”,导致识别准确率急速下降.

逻辑回归算法（LR）:逻辑回归算法本质为计算出预测函数的对应参数,通过输入自变量观察预测函数的输出值来进行分类的判断.逻辑回归算法计算简单,训练速度快,占用空间少,以概率形式输出.但是逻辑回归分类器是线性分类器,无法解决非线性问题,同时算法要求各类别样本数较为平均,检测准确率欠佳.

## 神经网络方法设计

在经过USTC-TFC2016工具集的处理后,我们将流量转换为了28\*28的图像.在此我们选取每类各9张图像,将四类图像:以太坊TCP流量,非以太坊TCP流量,以太坊UDP流量,非以太坊UDP流量进行展示.如下图所示,从根据流量前784字节生成的图像来看,以太坊TCP流量与非以太坊TCP流量之间存在明显的差异,而以太坊UDP流量与非以太坊UDP流量存在添补区域较大的情况,差异没有以太坊TCP流量与非以太坊TCP流量明显.





图6 以太坊TCP,UDP流量与非以太坊TCP,UDP流量可视化

我们使用CNN神经网络来进行以太坊流量和非以太坊流量的识别,最后我们得到的数据集为IDX文件的内容,其中的流量图像为28\*28\*1的情况.因此我们的神经网络总体处理流程为:CNN首先从IDX文件中读取28\*28\*1的流量图像.这些图像的像素从[0,255]被归一化到[0,1].第一个卷积层C1执行一个卷积运算,有32个大小为5\*5的内核.C1层的结果为32个特征图,大小为28\*28.在Ｃ1层之后,在P1层中有一个2\*2的max池操作,结果是32个特征图,大小为14\*14.第二个卷积层C2的内核大小也是5\*5,但有64个通道.结果是64个尺寸14\*14的特征图.在第二个2\*2的max池层P2之后,生成了64个大小为7\*7的feature map.最后两个层是完全连接的,输出大小分别为50和10,并使用softmax函数用来输出每个类的概率.此外,dropout是用来缓解过度拟合的.

# 实验分析

## 实验环境

实验机器为联想拯救者R720,CPU为Intel i5-7300HQ,基准频率为2.5Ghz,内存为18G,显卡为移动端1050ti.

两个实验中我们均使用Python语言,在机器学习中我们使用sklearn库来进行不同机器学习算法的调用,在神经网络中我们使用Pytorch架构来完成神经网络的搭建.

## 评价指标

我们使用准确率、精确率、召回率、F1分数来评价分类器对于以太坊和非以太坊流量的识别性能.在计算准确率、精确率、召回率、F1分数四个分数的时候我们需要先行计算真阳性(TP),假阳性(FP),假阴性(FN),真阴性(TN).其中真阳性指的是实际为正常的样本被分类为异常样本的数目.假阳性指实际为异常的样本被分类为正常样本的数目.假阴性指的是实际为正常的样本被分类为异常样本的数目.真阴性指实际为异常的样本被分类为异常样本的数目.准确率、精确率、召回率、F1分数的计算如下所示:

 (1)

 (2)

 (3)

 (4)

.

## 实验结果与分析

首先我们就机器学习的结果进行分析,我们将流量划分为TCP流量和UDP流量两类流量分别进行以太坊流量的识别.如表3所示,我们可以看到,随机森林算法在四类指标中都领先于其他算法均超过了99.9%这一界限,表现十分出色.而KNN算法与随机森林算法的各类性能指标较为接近,也超过了99%,LR算法各性能指标位于第三,性能指标评价位于96%附近.但是出乎意料的是SVM算法相较于其他三类算法而言,其各类指标都不太理想,都在75%上下波动,分析其原因,可能是因为我们对所有算法都使用默认设置,而SVM算法要想达到一个较好的识别情况,需要对参数进行反复调试.

表3 以太坊TCP流量识别各方法性能指数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 机器学习方法 | 准确率 | 精确率 | 召回率 | F1指数 |
| **RF** | **0.9991** | **0.9986** | **0.9987** | **0.9990** |
| SVM | 0.7501 | 0.8210 | 0.7439 | 0.7322 |
| LR | 0.9672 | 0.9671 | 0.9673 | 0.9672 |
| KNN | 0.9967 | 0.9967 | 0.9967 | 0.9967 |

接着我们就UDP流量分析结果分析,如表4所示,我们可以看到,随机森林算法在UDP流量的分类效果更好,四类指标都达到了99.9%左右,分类效果优于其他几类算法.而KNN算法与随机森林算法的各类性能指标机器接近,性能只相差了大概0.1%,四类指标也超过了99.8%,LR算法各性能指标位于第三,性能指标评价位于99%附近.在这里SVM算法与其他三类算法差距没有在TCP流量分类上那么悬殊,也达到了95%.

表4 以太坊UDP流量识别各方法性能指数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 机器学习方法 | 准确率 | 精确率 | 召回率 | F1指数 |
| RF | **0.9993** | **0.9991** | **0.9989** | **0.9992** |
| SVM | 0.9508 | 0.9618 | 0.9280 | 0.9423 |
| LR | 0.9912 | 0.9930 | 0.9924 | 0.9937 |
| KNN | 0.9983 | 0.9982 | 0.9979 | 0.9981 |

综上,在机器学习的算法选择上,我们使用随机森林算法作为分类算法能够得到一个较好的识别结果.

其次我们就神经网络的结果进行分析,我们使用的是CNN神经网络,迭代轮数设置为100轮,学习率0.01.基于以太坊TCP流量的识别结果为:平均损失率为0.0299,准确率为98.8201%,我们将四类性能指数作图得到如下图像:

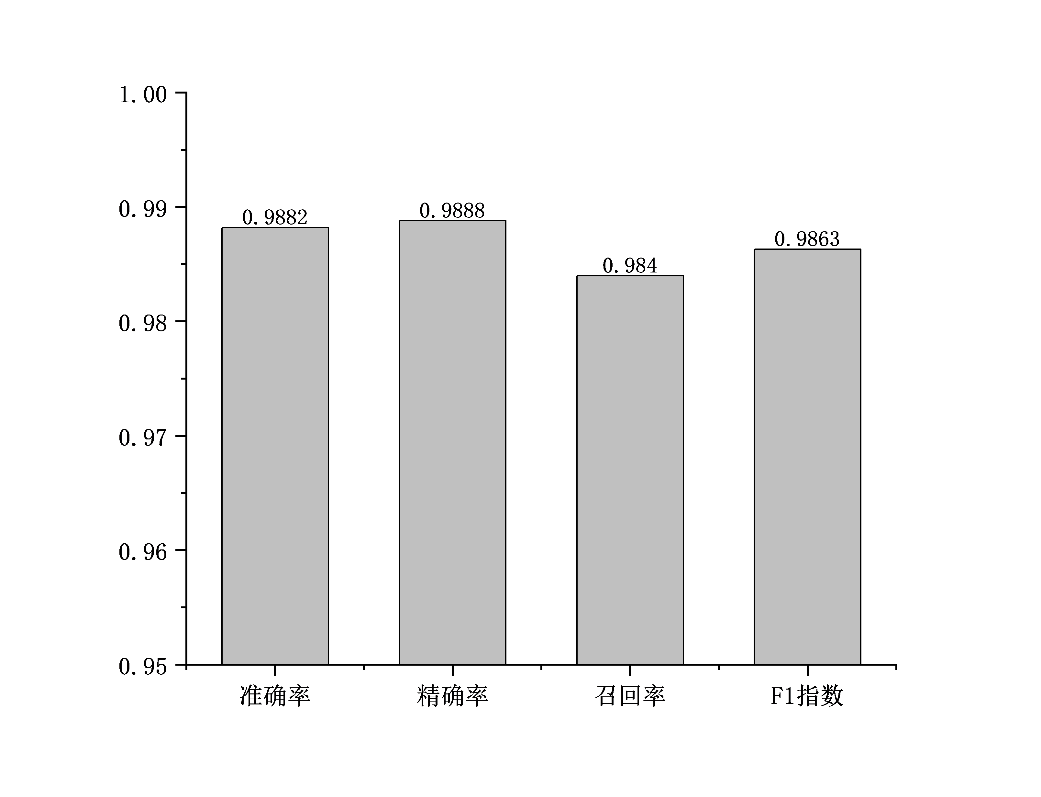


图7 CNN神经网络在以太坊TCP流量识别结果

从图中我们可以看到,CNN神经网络针对以太坊TCP流量的识别准确率达到了98.82%,精确率,召回率,F1指数也均超过了98%的情况.由于我们只进行了100轮的学习,模型识别效果还达不到最佳,对于CNN参数的设置也有待修改.因为实验设备性能的原因,我们只能设置学习轮数为100轮,如果学习轮数能够再增加,那识别准确率一定会比现在的情况更加优秀.

而基于以太坊UDP流量的识别结果为:平均损失率为0.0031,准确率为99.9376%,我们同样对四类性能指标进行作图,获得如下图像:

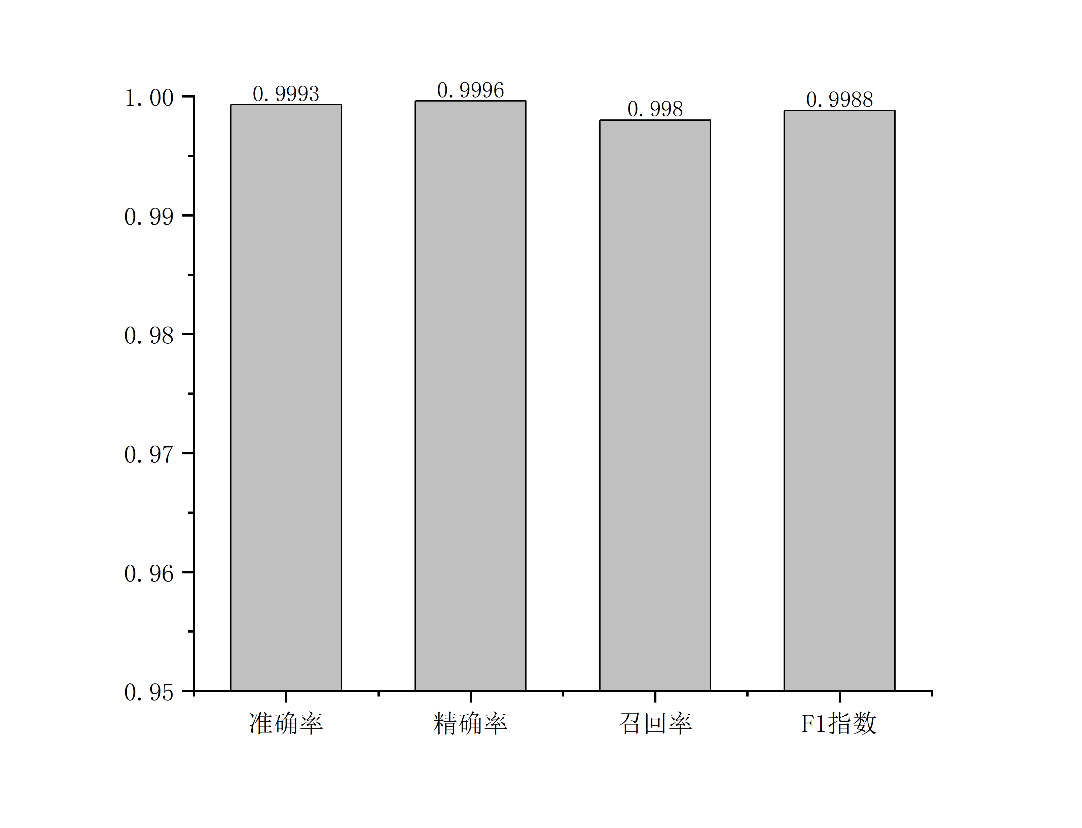


图8 CNN神经网络在以太坊UDP流量识别结果

由图像我们可以看到CNN神经网络针对以太坊UDP流量的识别效果更好,识别准确率达到了99.9376%,其他三类性能指标也都超过了99%.同样的方法在TCP与UDP流量上识别结果存在一定的差异,可能因为TCP与UDP流量本来就存在一定的差异,TCP为有连接而UDP为无连接,在流的分割上也采取了不同的分割方式:TCP按照FIN标志来切分而UDP使用超时来判断切分,而我们使用流量生成图像时也可以看到,大多数UDP流量需要使用0填充,有些流量甚至需要填充一半及以上,而TCP流量则几乎不需要填充,反而需要去掉多余的流量.

对比机器学习四类算法和CNN神经网络,综合各类情况,我们可以发现随机森林算法是所有方法中最为优秀的.不论从识别准确率,训练花费时间,判断花费时间来说,其都优秀于CNN神经网络,分析这一现象原因可以做出一些合理推测.首先,随机森林算法使用的数据集处理与CNN神经网络数据集不一样,随机森林算法采用了整条流的特征,信息完整,更能体现出流量特征.而CNN神经网络数据集使用的是前 784bits的数据,对TCP流量而言,这远远达不到一条流的长度,因此有很多流量信息没有获取到,其识别准确率比随机森林算法低也是应当的.从实际的情况而言,采用整个流的特征进行以太坊流量识别其实是不太合理的,这会耽误大量的实际,如果能够获取流量前n个报文或者前n个bits的信息就能识别的话,无疑会有效增加识别的效率.

# 总 结

本文就机器学习方法和神经网络方法对以太坊流量的识别进行了讨论分析,在预处理部分将流量分成TCP流量以及UDP流量,并且分别生成了机器学习和神经网络的数据集.两类方法都在识别结果上取得了良好的性能,随机森林算法的识别准确率更是达到了99.91%以及99.93%.然而实验依然存在一些不足,数据集可能存在不均衡的问题,以太坊流量和背景流量的占比也没有模仿真实世界中以太坊流量占网络总流量的比例.同时随机森林算法使用的流量为全流,如果能够只依靠前n个报文就进行以太坊流量的识别无疑是一个巨大的改进,这些问题有待后续研究进行完善改进.

References:

1. Nakamoto S, Bitcoin A. A peer-to-peer electronic cash system[J]. Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf, 2008, 4.
2. Wood G. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum project yellow paper, 2014, 151(2014): 1-32.
3. von Haller Gronbaek M. Blockchain 2.0, smart contracts and challenges[J]. Comput. Law, SCL Mag, 2016: 1-5.
4. Taş R, Tanrıöver Ö Ö. Building a decentralized application on the Ethereum blockchain[C]//2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT). IEEE, 2019: 1-4.
5. Neudecker T, Andelfinger P, Hartenstein H. Timing analysis for inferring the topology of the bitcoin peer-to-peer network[C]//2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld). IEEE, 2016: 358-367.
6. Li Z, Hou J, Wang H, et al. Ethereum Behavior Analysis with NetFlow Data[C]//2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2019: 1-6.
7. Deshpande V, Badis H, George L. BTCmap: mapping bitcoin peer-to-peer network topology[C]//2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN). IEEE, 2018: 1-6..
8. Kim S K, Ma Z, Murali S, et al. Measuring ethereum network peers[C]//Proceedings of the Internet Measurement Conference 2018. 2018: 91-104.
9. Shen M, Zhang J, Zhu L, Xu K, Du X. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. IEEE Transactions on Information Forensics and Security. 2021 Jan 11;16:2367-80.
10. Seo Y. Practical Implementations of ECC in the Blockchain[J]. ANALYSIS OF APPLIED MATHEMATICS, 2017: 43.
11. Biryukov A, Khovratovich D, Pustogarov I. Deanonymisation of clients in Bitcoin P2P network[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014: 15-29.
12. go-ethereum. 2021. Go Ethereum. Retrieved October 12, 2021 from https:// github.com/ethereum/go-ethereum.
13. CICFlowMeter. 2021. Arash Habibi Lashkari. Retrieved December 21, 2021 from https://github.com/ahlashkari/CICFlowMeter.
14. Wang W, Zhu M, Zeng X, et al. Malware traffic classification using convolutional neural network for representation learning[C]//2017 International Conference on Information Networking (ICOIN). IEEE, 2017: 712-717.

1. [↑](#footnote-ref-1)