2023 年《软件定义网络》大作业

题目　**基于 MVSketch 和深度学习算法的网络流量重流分析与攻击检测**

成员 1	**222100405** 陈知菲
成员 2	**102101219** 邱思源
成员 3	**102101221** 张一凡
成员 4	**102101311** 黄安妮
成员 5	**102101517** 杨宇晗

完成时间：**2023** 年 **11** 月 **20** 日

目录

1. 目标问题与意义价值

1.1

目标问题：

在校园网络中，网络速度不稳定是一个常见的问题。本次实验的目标是通过 SDN 架构和开源控制器实现校园 SDN 网络的管理和测量分析，以排查出影响网络速度的根本问题。

1.2

意义价值：

揭示校园网络速度不稳定问题的具体原因，并提供相应的解决方案，帮助解决校园网络速度不稳定的问题，能够提升师生的网络使用体验。

2. 设计思路与方案

2.1 总体设计

根据实验要求我们得出以下几个关键步骤：

(1) 搭建拓扑结构

(2) 数据包构造

(3) 流量统计

(4) 真实流量的测量分析

(5) 安全预警与防御

(6) 系统集成

我们根据以上五个关键步骤划分模块如下：

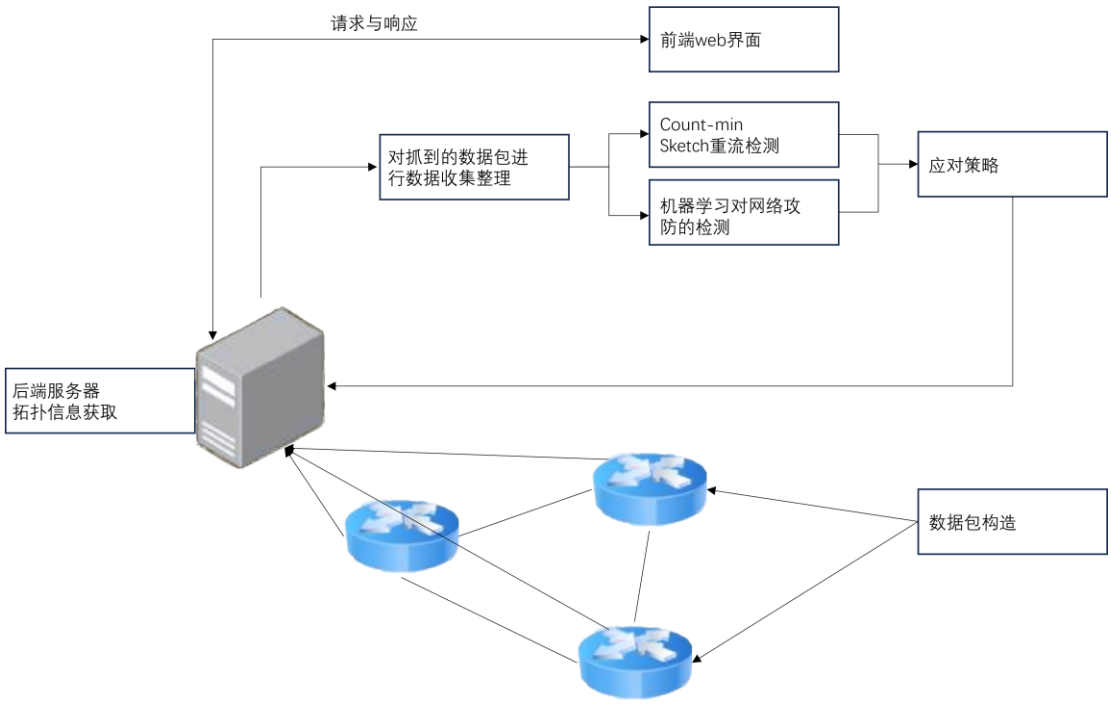
(1) 搭建拓扑结构：根据附件给定的拓扑 3 搭建拓扑。

(2) 数据包构造与抓包：使用 Scapy 工具自定义数据包；对抓包数据进行整理。

(3) 流量统计与分析：利用 MV Sketch 算法对构造的网络流量包进行统计，估计每个流的出现次数；判断是否存在重流，获取 k 条重流。

(4) 安全预警与防御：深度学习实现网络攻击检测；拦截检测到攻击的流量。

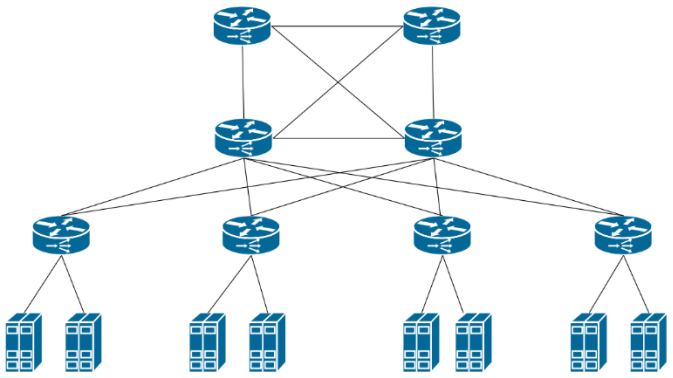
(5) 系统集成：前端设计；采用 vue 实现前端界面，动态生成拓扑，提供网络监测功能。



2.2 详细设计

1. 搭建拓扑结构：

(1) 根据附件给定的拓扑 3 搭建拓扑



2. 数据包构造与抓包：

(1) 使用 Scapy 工具自定义数据包并对抓包数据进行整理

我们组选择的拓扑有八台主机，每台主机都需要 xterm 来发送数据包，所以需要针对每台主机都写一个构建脚本。抓交换机的流表只需要抓和八台主机连接的四个交换机的对应接口即可。首先编写 UDP、TCP、ICMP 的构造函数，然后设定其他随机主机地址为目的 ip、进行无限循环发送随机的 UDP or TCP or ICMP 数据包。

读取 scapy 抓取到得的.pcap 文件，对于每一条数据流，按 eth 层、ip 层的顺序进行解析，得到五元组特征并返回。

3. 流量统计与分析：

(1) 利用 MV Sketch 算法对构造的网络流量包进行统计，估计每个流的出现次数。

(2) 判断是否存在重流，获取 k 条重流

按数据规模为 MV-Sketch 所需要的空间分配大小，初始化每一个桶，然后将之前解析的每一条数据流的五元组的地址当作键值，将键值放入哈希函数，根据得到的值更新相应的桶内容；所有数据插入之后，根据传入的参数选择返回重流或返回所有流，将所需信息按 json 格式返回。

MV-Sketch 实现如下：

函数简介：

1) 构造函数 MVSketch::MVSketch(int depth, int width, int lgn)

depth：散列函数个数

width：每个散列函数对应的桶个数（宽度）

lgn：键值长度

其中使用 AwareHash()生成用于进行散列的种子

```
unsigned long seed = AwareHash((unsigned char*)name, strlen(name), 13091204281, 228204732751, 6620830889);
for (int i = 0; i < depth; i++) {
    mv_.hardner[i] = GenHashSeed(seed++);
}
```

2) 更新函数 void MVSketch::Update(unsigned char* key, val_tp val)

key：用于散列的键值（用于溯源的键值）

val：用于票选的值（重量）

对于每一个键值，需要根据所有散列函数的返回值将其插入桶中。若桶规模为 depth*width，则对于一个键值，要根据 depth 个散列函数将其插入至 depth 个桶中，其中一个 depth 函数对应一行数量为 width 桶。

对于一次循环，首先使用初始化 MVSketch 时生成的种子进行散列，根据返回值得到本次要更新的桶的下标。若桶为空，则标识该桶的键值更新为 key 并且该桶的重量自增 val；若该桶的键值与 key 相同，则该桶的重量自增 val；若该桶的键值与 key 不同，将该桶的重量减去 val，此时若该桶的 val 小于等于 0 则将该桶的键值更新为 key。如此重复 depth 次。

```
for (int i = 0; i < mv_.depth; i++) {
    bucket = MurmurHash64A(key, keylen, mv_.hardner[i]) % mv_.width;
    int index = i*mv_.width+bucket;
    MVSketch::SBucket *sbucket = mv_.counts[index];
    sbucket->sum += val;
    if (sbucket->key[0] == '\0') {
        memcpy(sbucket->key, key, keylen);
          sbucket->count = val;
```

```
        } else if(memcmp(key, sbucket->key, keylen) == 0) {
            sbucket->count += val;
        } else {
            sbucket->count -= val;
          if (mv_likely(sbucket->count < 0)) {
              memcpy(sbucket->key, key, keylen);
              sbucket->count = -sbucket->count;
          }
      }
}
```

3) 查询函数 void MVSketch::Query(val_tp thresh, std::vector<std::pair<key_tp, val_tp> >&results)

thresh：用于确定重流的因子，当一个桶的重量大于 thresh 时它才有可能成为重流。

results：用于存放查询结果的一个 vector，其中 key_tp 为键值、val_tp 为重量。

对于每一个候选结果，查询它在 depth 行中的最小重量，若最小重量仍大于 thresh，则将该候选插入结果。

```
    for (auto it = res.begin(); it != res.end(); it++) {
        val_tp resval = 0;
        for (int j = 0; j < mv_.depth; j++) {
            unsigned long bucket = MurmurHash64A((*it).key, mv_.lgn/8, mv_.hardner[j]) % mv_.width;
            unsigned long index = j*mv_.width+bucket;
            val_tp tempval = 0;
            if (memcmp(mv_.counts[index]->key, (*it).key, mv_.lgn/8) == 0) {
                tempval = (mv_.counts[index]->sum - mv_.counts[index]->count)/2 + mv_.counts[index]->count;
            } else {
                tempval = (mv_.counts[index]->sum - mv_.counts[index]->count)/2;
            }
            if (j == 0) resval = tempval;
            else resval = std::min(tempval, resval);
        }
        if (resval > thresh ) {
            key_tp key;
            memcpy(key.key, (*it).key, mv_.lgn/8);
            std::pair<key_tp, val_tp> node;
            node.first = key;
            node.second = resval;
```

```
                results.push_back(node);
            }
        }
```

如何估计流量是否准确？

由于散列函数的特殊性，两个大小不同的键值经过同一个哈希函数散列后得到的值可能相同，因此估计出的流量大小可能偏大。

如何如何提高准确率？

若散列函数的数量设置的足够多或者根据数据内容将宽度进行调整，可以减少散列得到的值重复的可能性，但是过多的散列函数将导致空间占用过大，宽度过长也将导致退化问题。

4. 安全预警与防御：

(1) 深度学习实现网络攻击检测

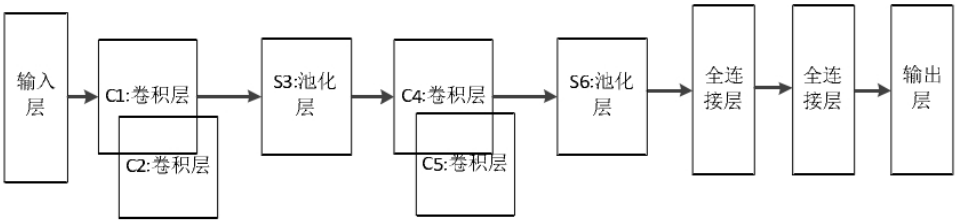
数据集 CIC-IDS-2017：

大多数数据集（比如经典的 KDDCUP99，NSLKDD 等）已经过时且不可靠。其中一些数据集缺乏流量多样性和容量，一些数据集没有涵盖各种已知的攻击，这不能反映当前的趋势。

而 CIC-IDS-2017 数据集提取出多达 75 个特征，并分为 15 种网络攻击类型，它包含良性和最新的常见攻击，类似真实世界数据，可用 PCAP 文件实现特征提取，处理起来比较方便快捷，因此，我们选择 CIC-IDS-2017 作为我们的训练数据集。

检测模型：

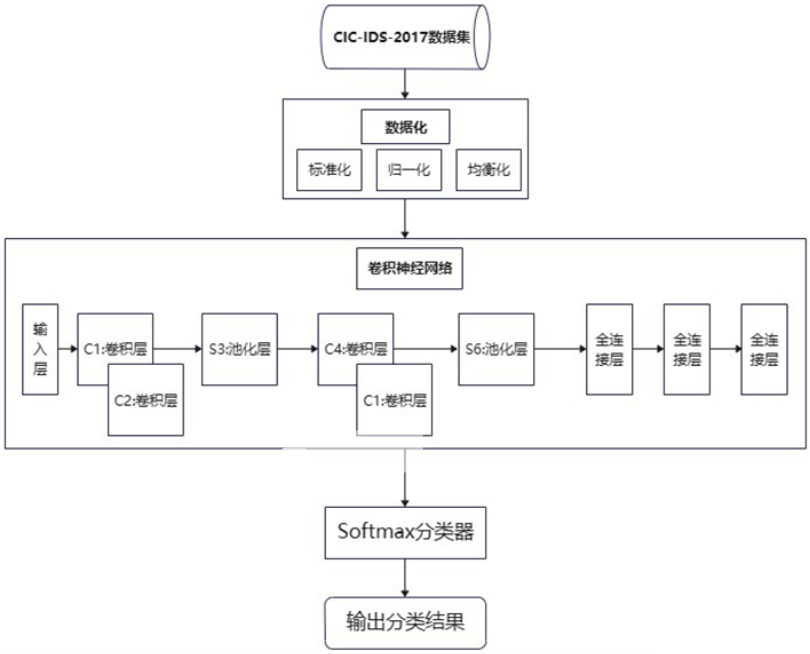
采用改进的 LeNet-5 的入侵检测模型



将 LeNet-5 中单个卷积层接一个池化操作改为两个卷积层后接一个池化操作，避免池化操作过于密集，两个卷积层的直接堆叠也使模型对入侵数据的特征提取更加准确。

最后使用了两个全连接层和 Softmax 函数，分为 15 类攻击类型。

具体流程：



1) 训练模型：

版本：python: 3.7，pytorch: 1.8.0

模型构建如上述

2) SCPAY 抓包,将数据包以.pcap 文件的形式存储

读取.pcap 文件，使用 cicflowmeter（流量特征提取工具，输入 pcap 文件，输出 pcap 文件中包含的数据包的特征信息，共 80 多维，由于实验需要，丢弃其图形化界面，改为命令行执行）从 pcap 文件中提取出 CIC-IDS-2017 数据集的 75 个特征，以.csv 文件存储

3) 数据预处理：将.csv 文件转化为 CIC-IDS-2017 的数据形式，并将数据标准化，归一化，送入之前训练的模型

4) 提取出分类结果中检测到网络攻击的数据,将其的五元组(目的 IP,源 IP，目的端口，源端口，协议）以 JSON 格式返回给前端。

(2) 拦截流量

对于流量攻击检测的检测出攻击的结果，将五元组返回给前端，由用户决定是否进行拦截

5. 系统集成：

(1) 前端设计

采用墨刀作为原型设计工具。

(2) 采用 vue 实现前端界面，动态生成拓扑，提供网络监测功能

调用 ryu 接口，通过 Antv G6 图可视化引擎在前端页面上动态生成拓扑信息，根据原型图对前端页面布局进行代码实现，在前端的不同功能模块的页面中，调用后端接口并对返回数据进行处理，将处理后的数据在前端页面中静态/动态展现。

前端基于 Vue9.6.7，Mininet，Ryu 控制器，设计了拓扑查看、流量统计、流量分析、安全预警、安全防御页面。

a. 拓扑查看页面

拓扑查看页面基于 Mininet，RYU 控制器实现，通过主机调用 Ubuntu20.04 虚拟机上的 Ryu REST api，在 Vue 中通过阿里的 Antv G6 图可视化引擎动态生成拓扑主机、交换机以及主机与交换机之间的连接，每个主机节点和交换机节点包含名称，编号，Ip，状态等相关信息，当鼠标移动到主机/交换机节点上时，能够查看对应主机/交换机的信息。主机状态分为”attacked”和”safe”，主机状态由调用 http://101.37.13.1:8888/get_flow_count 接口，请求网络异常信息后整理判断得到，不同状态下主机节点的图标有着不同的样式以区分。页面中的拓扑每隔 20 秒自动刷新一次，也就是每隔 20 秒就会再次请求拓扑数据和受到网络攻击的主机数据，并更新拓扑以及主机节点的状态。

b. 流量统计页面

流量统计页面由调用 http://101.37.13.1:8888/get_flow_count 接口，整理并在前端页面上分块展示总流量次数和每条流的信息，包括源 IP，源端口，目的 IP，目标端口，出现次数，页面每隔 20 秒自动获取并更新流量统计信息。

c. 流量分析页面：

用户在输入框中输入要查看的出现次数最多的 k 条流量（Top-k）的 k 值，点击获取按钮后，页面调用 http://101.37.13.1:8888/get_hitter 接口，请求并分块展示出现次数最多的 k 条流，Top-k 中的每条流的源 IP，源端口，目标 IP，目标端口，出现次数。并显示当前展示的重流的条数，出现总次数以及重流的总条数。

d. 安全预警页面：

安全预警页面由调用 http://101.37.13.1:8888/read_json 接口，整理并在前端页面上展示异常网络流数量以及分块展示各个异常流的信息，异常流信息包括源 IP，源端口，目标 IP，目标端口，流类型，页面每隔 20 秒自动获取并更新异常网络信息。

e. 安全防御页面

安全防御页面分为两个模块，异常网络信息模块和修改流表防御模块，页面调用 http://101.37.13.1:8888/read_json 接口在页面左半部分容器显示异常网络信息，中间部分为开始防御按钮，用户点击该按钮后，右半部分容器根据请求异常网络得到的流信息，作为 json 参数，请求安全防御模块的接口 http://101.37.13.1:8889/intercept_flow，防御过程中动态生成添加的流表信息以及是否添加成功，若添加成功，则相应的异常网络信息消失，拓扑中主机节点的状态发生改变。

6. REST API

流量统计——返回出现各条流的次数

url:

http://101.37.13.1:8888/get_flow_count

方法：

post

返回结果示例：

return:

```
{
    "all_message": {
        "all_net_packets": [],
        "net_packets": [
```

```
                {
                    "appear_time": 2,
                    "dstip": "43.237.97.250",
                    "dstport": 9339,
                    "protocol": 6,
                    "srcip": "43.139.98.136",
                    "srcport": 44069
                },
                {
                    "appear_time": 4,
                    "dstip": "221.46.220.117",
                    "dstport": 443,
                    "protocol": 6,
                    "srcip": "194.22.221.170",
                    "srcport": 52435
                }
            ],
            "total_packets_num": xxx
        }
}
```


流量分析——获取 k 条重流

url:

http://101.37.13.1:8888/get_hitter

方法：

Post

传入参数：

Content-type: application/json

参数名称	参数类型	参数描述
k	integer	获取重流条数

返回结果示例：


```
{
    "message": {
        "Error": 790.7215775854728,
        "Precision": 1.0,
        "Recall": 0.02510864708988262,
        "net_packets": [
            {
                "dstip": "69.45.5.161",
                "dstport": 40757,
                "protocol": 3,
                "resval": 16367,
                "srcip": "69.48.219.0",
                "srcport": 54945
```

```
            },
            {
                "dstip": "69.45.5.161",
                "dstport": 40757,
                "protocol": 3,
                "resval": 15236,
                "srcip": "69.48.219.5",
                "srcport": 54945
            },
            {
                "dstip": "69.48.219.5",
                "dstport": 17709,
                "protocol": 13729,
                "resval": 2210,
                "srcip": "200.0.6.1",
                "srcport": 1441
            },
            {
                "dstip": "69.48.219.0",
                "dstport": 17709,
                "protocol": 13729,
                "resval": 2002,
                "srcip": "40.0.6.1",
                "srcport": 1441
            },
        ],
        "res_size": xxxx
    }
}
```


安全防御——拦截流量

url:

http://101.37.13.1:8888/intercept_flow

方法：

Post

传入参数：

Content-type: application/json

参数名称	参数类型	参数描述
dpid	integer	交换机编号
ip_src	string	源 IP
ip_dst	string	目的 ip
tp_src	integer	源端口
tp_dst	integer	目的端口

proto	integer	协议编号

返回结果示例：

成功：流表项添加成功！

失败：流表项添加失败！

安全预警——网络攻击检测

url:

http://101.37.13.1:8888/read_json

方法：

Post

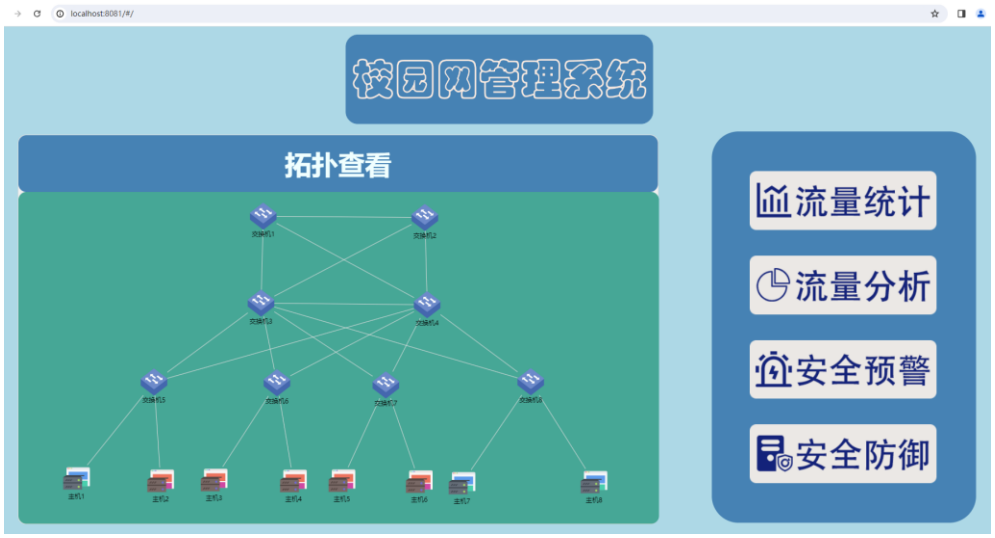
返回结果示例：

return:


```
[
    {
        "dst_ip": "137.182.81.175",
        "dst_port": "443",
        "flow_id": "130.22.37.224-137.182.81.175-57164-443-6",
        "protocol": "6",
        "src_ip": "130.22.37.224",
        "src_port": "57164",
        "type": "DoS Hulk"
    },
    {
        "dst_ip": "3.252.157.39",
        "dst_port": "80",
        "flow_id": "130.77.108.225-3.252.157.39-40392-80-6",
        "protocol": "6",
        "src_ip": "130.77.108.225",
        "src_port": "40392",
        "type": "DoS Hulk"
    },
    {
        "dst_ip": "1.146.90.204",
        "dst_port": "443",
        "flow_id": "111.37.248.214-1.146.90.204-12041-443-6",
        "protocol": "6",
        "src_ip": "111.37.248.214",
        "src_port": "12041",
        "type": "DoS Hulk"
    }
]
```

3. 作品实现

拓扑查看页面：



流量统计页面：

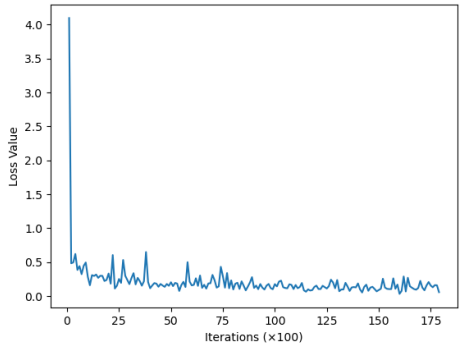
流量分析页面：

Ⅳ安全预警页面：

Ⅴ安全防御页面：

网络攻击检测：

训练结果




```
BENIGN                          2260360
DoS Hulk                         229198
PortScan                         157703
DDoS                             127082
DoS GoldenEye                     10289
Infiltration                       8704
FTP-Patator                        7894
Bot                                7772
Web Attack • Brute Force           5988
SSH-Patator                        5861
DoS slowloris                      5771
Heartbleed                         5632
DoS Slowhttptest                   5485
Web Attack • Sql Injection         5376
Web Attack • XSS                   5184
Name: 78, dtype: int64
Accuracy: 0.9602675279991574
Average Loss: 0.11548131394698644
```

4. 创新与特色

(1) 我们小组采用深度学习实现对流量的攻击检测,与传统的机器学习方法相比，基于深度学习的技术的泛化能力更好，能够利用大数据和高算力达到学习的高准确率。

(2) 把整个项目部署在云服务器上，便于多人协作完成项目任务。

(3) 在流量统计部分，我们选择运用运用 MVSketch 算法的实现。

5. 小组总结

222100405 陈知菲

1. 所做工作

(1) 后端流量统计、安全预警、流量分析、安全防御接口的编写及接口文档的撰写;（10h）

(2) 深度学习模型的训练及其与后端接口的整合（包括特征提取、csv 文件的转换）;（30h）

(3) 拓扑的搭建和 ryu 控制器的选择修改（3h）

2. 课程的收获和体会

在 SDN 课程中，我了解了软件定义网络的基本概念与思想，了解了 SDN 网络中的集中式控制平面和分布式转发平面的关系。学会了如何使用 mininet、ryu、odl 等工具搭建一个 sdn 网络，并向控制器下发命令对网络进行管理和流量控制、以及 SDN 网络中的安全问题和解决方法。在课外，我了解到一些将 SDN 和深度学习结合，以实现对网络的流量进行检测和分类的拓展知识，极大地丰富了我的知识量，令我受益匪浅。

SDN 课程理论与实践结合的模式，使我能够很快地将理论知识运用到实践上，从而巩固理论，提高动手能力，而在最后的大作业中，我不但增强了自己的团队协作能力，更在合作的过程中将 SDN 运用于完整的一个项目上，提出了将 SDN 和深度学习结合实现攻击检测的创新。在训练模型、调试框架、整合代码的过程中，和队友们一起发现问题、解决问题，填补了理论知识的空缺。

102101219 邱思源

1. 所做工作

(1) 读取 scapy 抓取到得的.pcap 文件，利用 libpcap 库提供的方法，解析.pcap 文件，得到每一条数据流的五元组特征（包括源 ip、目的 ip、源端口、目的端口、协议类型）。

(2) 使用 MV-Sketch 算法对先前解包得到的所有数据流进行统计与分析。

(3) 编写方法调用 ryu 控制器对异常流量进行控制，完成安全防御功能。

(4) 根据传入的参数，返回 Top-K 条重流信息或者所有数据流信息，返回信息使用 nlohmann/json 库按 json 格式返回。(综上，总共花费 42h)

2. 课程的收获和体会

在此次大作业中我主要负责分析数据流以及对异常流量的控制，学习 libpcap 和 nlohmann/json 的使用、分析不同协议下数据帧报文格式、构建 sketch 并使用 MV-Sketch 进行数据流量分析。在解包部分，根据网上教程对数据流进行分层解析，重点学习 TCP 与 UDP 报文格式。在使用 MV-Sketch 算法的过程中，体会散列思想在大规模数据分析中的应用效果；在对桶进行更新过程中，对比 Count Min-Sketch 算法，学习如何利用键值进行散列并为数据溯源做准备。本次大作业让我学会对大规模数据进行简单统计和通过控制器完成安全控制，在实现过程中代码部分出现过许多漏洞，包括指针异常等问题，在与组内成员讨论交流下完成对程序的修改。学习和实现功能的过程是曲折和痛苦的，但是成功修复漏洞和运用新学知识为团队献出一份微薄贡献是令人喜悦的。

102101221 张一凡

1. 所做工作

(1) 自学 scapy，编写构造各主机向其他主机发送的数据包脚本，编写交换机抓包脚本，.pcap 文件获取，分析数据包。(32h)

2. 课程的收获和体会

单掌握了通过 scapy 来构造 TCP、UDP、ICMP 数据包，重点了解了 send（）与 sendp（）两种函数的区别并在后续工作中不断改进代码，实现要求。由于网上没有成型的 scapy 教学，自学 scapy 的过程十分艰难，本人与隔壁组与我相同工作的同学花了三天空闲时间不断改代码，重复试错，尝试在不同的终端运行代码，最终探索出正确的构建抓包的代码与方式。这次自学的过程对本人的自学能力是强大的挑战，本次实验经历使我获得了很多的知识与乐趣。

102101311 黄安妮

1. 所做工作

(1) 选择并学习原型设计工具，对前端界面进行原型设计，并进行合理的调整.(3h)

(2) 基于任务完成过程中的需求，需要解决 python 调用程序并返回正确值，尝试通过 CTypes 调用和使用 python 标准库中的 subprocess 包调用，并选择合适的方法。(5h)

(3) 整理与撰写报告文档。(5h)

(4) 学习剪辑与视频剪辑。(3h)

2. 课程的收获和体会

软件定义网络课程理论与实践结合的模式,让我学会使用 Mininet 的可视化拓扑，了解了分析 openflow 协议以及在这个协议下交换机和控制器间进行交流分享数据的方式，又在后续的实验中学习了如何向控制器下发命令对网络进行管理和流量控制。在完成大作业的过程中，我将知识运用到实践中的能力得到了提升，在此期间我与队友相互配合解决问题，提高了彼此团队协作能力。

102101517 杨宇晗

1. 所做工作

(1) 对项目进行初步了解和生成设计文档。(2h)

(2) 选择合适的开发工具、选择合适的图可视化引擎，学习 npm 组件库（axios，Antv G6 等）和其他学习过程。(8h)

(3) 具体设计和具体编码过程，包括处理各类异常和错误。(18h)

(4) 代码复审、测试过程。(2h)

2. 课程的收获和体会

在本次课程中，我收获颇丰，学习了 Mininet 的可视化拓扑，通过自己的布置主机以及交换机等，就可以让他们互相连通起来，以及编写 python 脚本来生成复杂的数据中心网络拓扑，也在 51OpenLab 平台，学习了很多关于构造拓扑网络的知识，令我对何为软件定义网络有了更加直观且全面的认识，还了解如何分析 openflow 协议，了解了在这个协议之下交换机以及控制器进行交流分享数据的方式，对数据交互有了更深的理解，以及如何使用 wireshark 抓包，之后完成了完成了 OpenDaylight 控制器的安装配置和 UI 界面的生成，直观地体现了网络拓扑信息，以及如何使用 Postman 下发流表。我在 SDN 学习中对 POX 控制器

的工作原理和功能有了更深刻的影响印象，通过实践验证了 POX 的 Hub 和 L2_learning 模块的功能，深刻理解了 RYU 控制器实现软件定义的集线器原理和交换机原理，以及调用 rest api 得到拓扑信息。

在这次团队项目中，中途遇到了不少的困难，例如一开始在虚拟机上部署 Vue 项目时就遇到的 npm，nodejs 之间的版本不对应,npm 和组件库 axios 之间的版本不对应问题，以及在后续出现的 CORS 跨域请求问题，在尝试设置代理等多种方法后仍没有解决，后来不得不将 Vue 项目重新部署到 Windows 主机上，通过设置浏览器启动状态，这个问题得到了解决。跨域请求解决后，出现了主机如何连接虚拟机的问题，后续也遇到了分析和加工 rest api 中返回数据得到隐含信息，如何在 SDN 中使用图引擎更直观的描述出网络拓扑情况等等，经过不断地翻阅资料和请教团队队友，遇到的困难基本都得到了解决，在解决困难中我更深刻了解了搭建 SDN 的流程，也学习巩固了开发工具的使用，掌握了流表，网络攻击，重流等一系列概念。通过实践，我对 SDN 的理解和使用有了更进一步的了解，并对 SDN 产生了浓厚兴趣。

6. 参考资料

Yu, L., Dong, J., Chen, L., et al. (2021). PBCNN: Packet Bytes-based Convolutional Neural Network for Network Intrusion Detection. Computer Networks, 194, 108-117.

Tang, L., Huang, Q., & Lee, P. P. C. (2019). MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In 2019 IEEE Conference on Computer Communications (INFOCOM) (pp. 1311-1319). IEEE.