



基于深度学习的软件定义网络应用策略冲突检测方法

李传煌¹, 程成¹, 袁小雍², 岑利杰¹, 王伟明¹

(1. 浙江工商大学信息与电子工程学院, 浙江 杭州 310018;

2. 美国佛罗里达州大学大规模智能系统实验室, 美国 佛罗里达州 盖恩斯维尔 32611)

摘要: 在基于 OpenFlow 的软件定义网络 (SDN) 中, 应用被部署时, 相应的流表策略将被下发到 OpenFlow 交换机中, 不同应用的流表项之间如果产生冲突, 将会影响交换机的实际转发行为, 进而扰乱特定应用的正确部署以及 SDN 的安全。随着 SDN 规模的扩大以及需要部署应用的数量剧增, 交换机中的流表数量呈现爆炸式增长。此时若采用传统的流表冲突检测算法, 交换机将会耗费大量的系统计算时间。结合深度学习, 首次提出了一种适合 SDN 中超大规模应用部署的智能流表冲突检测方法。实验结果表明, 第一级深度学习模型的 AUC 达到 97.04%, 第二级模型的 AUC 达到 99.97%, 同时冲突检测时间与流表规模呈现线性增长关系。

关键词: 流表冲突检测; 深度学习; 异常检测; 软件定义网络; OpenFlow

中图分类号: TP393

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2017305

Policy conflict detection in software defined network by using deep learning

LI Chuanhuang¹, CHENG Cheng¹, YUAN Xiaoyong², CEN Lijie¹, WANG Weiming¹

1. School of Information and Electrical Engineering, Zhejiang Gongshang University, Hangzhou 310018, China

2. LiLAB, University of Florida, Gainesville, Florida 32611, USA

Abstract: In OpenFlow-based SDN (software defined network), applications can be deployed through dispatching the flow policies to the switches by the application orchestrator or controller. Policy conflict between multiple applications will affect the actual forwarding behavior and the security of the SDN. With the expansion of network scale of SDN and the increase of application number, the number of flow entries will increase explosively. In this case, traditional algorithms of conflict detection will consume huge system resources in computing. An intelligent conflict detection approach based on deep learning was proposed which proved to be efficient in flow entries' conflict detection. The experimental results show that the AUC (area under the curve) of the first level deep learning model can reach 97.04%, and the AUC of the second level model can reach 99.97%. Meanwhile, the time of conflict detection and the scale of the flow table have a linear growth relationship.

Key words: policy conflict detection, deep learning, anomaly detection, SDN, OpenFlow

收稿日期: 2017-06-29; 修回日期: 2017-11-10

基金项目: 国家高技术研究发展计划 (“863”计划) 基金资助项目 (No.2015AA011901); 国家自然科学基金资助项目 (No.61402408, No.61379120); 浙江省自然科学基金资助项目 (No.LY18F010006); 浙江省重点研发计划基金资助项目 (No.2017C03058)

Foundation Items: The National High Technology Research and Development Program (863 Program) (No.2015AA011901), The National Natural Science Foundation of China (No.61402408, No.61379120), Zhejiang Provincial Natural Science Foundation of China (No.LY18F010006), Zhejiang's Key Project of Research and Development Plan (No.2017C03058)



1 引言

随着互联网业务和服务需求的不断增长,云计算网络、大型数据中心网络等已经成为承载各种大型计算和存储业务的重要网络基础设施,现有网络架构已经不能有效地满足这些新型网络平台的需求。作为下一代网络技术的重要代表,软件定义网络 (software defined network, SDN) 技术受到了学术界和产业界的极大关注,而其中基于 OpenFlow 的 SDN 技术更是得到了大家的青睐^[1,2],它已应用于如 Google 等互联网企业的大型数据中心网络中。

SDN 网络架构的重要特征是控制平面与转发平面的分离,在基于 OpenFlow 的 SDN 架构中,用户通过分离的控制器对网络中的 OpenFlow 交换机下发各种类型的转发策略,实现用户特定业务的部署,如防火墙、负载均衡、流量工程等。

SDN 应用/编排层根据某业务特征生成特定流表项时,所生成的策略可能与系统内已有的其他业务策略间产生冲突。如表 1 所示,策略 1 和 4、策略 2 和 3 以及策略 5 和 6 之间都存在重叠内容,且不同策略的行为互相抵触,这些策略间都存在冲突问题。

当对一个大型的基于 OpenFlow 的 SDN 中的所有业务进行自动部署和控制时,如果不对 OpenFlow 交换机中的流表进行冲突检测,一旦 OpenFlow 控制器对 OpenFlow 交换机添加的某些策略与之前已有的策略间存在冲突,将使得一些已存在的流表项被更改或覆盖而失效,最终对特

定业务产生致命的影响,如,当前策略冲突中有防火墙应用,则会使某些本被防火墙阻止的数据分组顺利通过,甚至给基于 OpenFlow 的 SDN 带来安全隐患,也不利于大型网络中 SDN 业务的自动部署。

图 1 展示了一个策略冲突引起防火墙功能失效的例子,该例子的拓扑图中包含了 4 台 PC 机、1 台 OpenFlow 交换机和 1 台 OpenFlow 控制器。PC 机 A 的 IP 地址为 192.168.1.10, PC 机 B 的 IP 地址为 192.168.2.2, PC 机 C 的 IP 地址为 192.168.3.7, PC 机 D 的 IP 地址为 192.168.6.5, 4 台 PC 机分别与 OpenFlow 交换机相连组成一个局域网。OpenFlow 交换机中的 Flow Table 0 中存在一条防火墙规则,该防火墙的规则是屏蔽源 IP 地址为 192.168.1.0/24、目的 IP 地址为 192.168.6.0/24 的数据分组。此时,OpenFlow 控制器即将下发一条策略,该策略是将目的 IP 地址和源 IP 地址都为 192.168.0.0/16 的所有数据分组设置一个队列 ID。如果把该策略下发到 OpenFlow 交换机中,所有源 IP 地址为 192.168.1.0/24 和目的 IP 地址为 192.168.6.0/24 的数据分组的动作行为将会冲突,从而导致原有防火墙规则的失效。

OpenFlow 相关协议本身并未提供流表冲突检测的处理机制,当前某些 OpenFlow 交换机具有简单的流表冲突检测功能,其流表冲突检测功能采用了基于传统的算法查找 OpenFlow 交换机中冲突的流表项的方法。如果 OpenFlow 交换机中的流表数目非常庞大,使用传统的流表冲突检测算法会增加 OpenFlow 交换机的运行负担、延缓流表的部署,影响数据分组正常的转发。

表 1 存在冲突的流表策略

规则	源 MAC 地址	目的 MAC 地址	源 IP 地址	目的 IP 地址	动作
1	*	*	10.20.0.1/8	10.20.5.100	drop
2	*	*	*	10.20.1.1/8	limit 8 Mbit/s
3	*	*	10.20.100.1/8	*	limit 1 Mbit/s
4	*	*	10.20.0.100	*	output port 10
5	14:10:9f:ea:7a:fd	*	*	*	*
6	*	14:10:9f:ea:50:3e	*	*	*

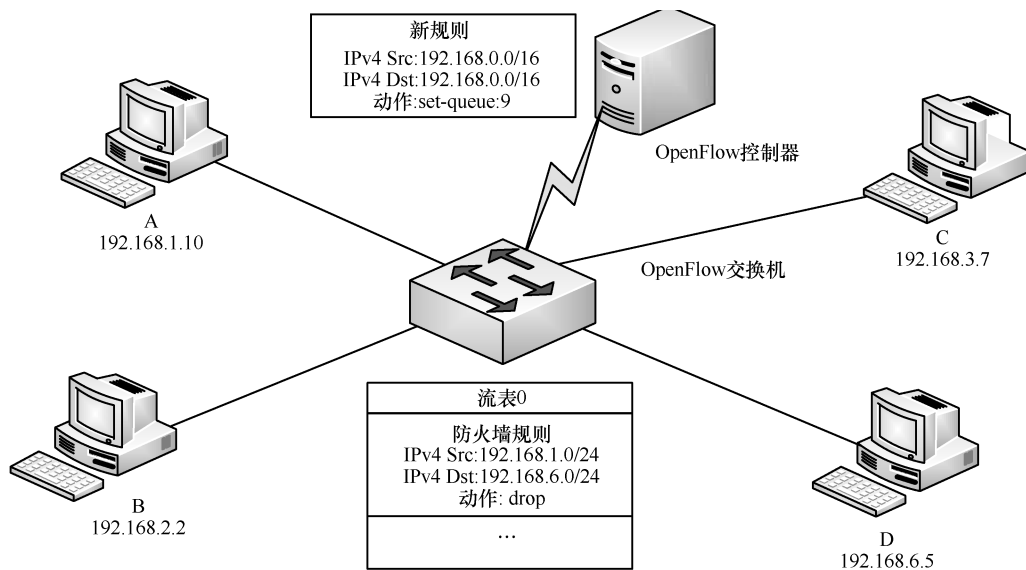


图1 策略冲突引起防火墙功能失效

当某个应用在 SDN 中进行具体部署时,若此应用需同时向多个 OpenFlow 交换机下发不同的策略,当其中一个交换机中检测出有流表冲突时,为避免对已有应用产生影响,SDN 控制器需要撤销当前应用下发的流表操作,并将对其他交换机已完成的操作进行回滚,然后将处理结果通知应用/编排层。显然因为冲突检测被放置在交换机内,给部署操作带来了冗余,增加了部署时间,也给整个部署过程带来了更多的不确定性,所以在控制器下发策略之前进行检测,预防冲突策略显得十分必要。

本文首次将 SDN 与深度学习结合,提出一种部署于控制器上的基于深度学习的 SDN 流表冲突检测方法。该方法利用深度学习抽象化高层数据、自动学习的特点,相比于传统的冲突查找算法,在大规模应用部署时能更快速地对超大规模的流表项做出是否冲突的检测。

2 相关工作

OpenFlow 协议^[3]中没有给出流表冲突的具体解决策略,仅有如下描述:如果对流表中 OFPFF_CHECK_OVERLAP 位进行了设置,

OpenFlow 交换机必须首先检测新下发的流表与交换机中已经存在的流表是否存在重复。当前对 OpenFlow 流表冲突检测的研究,都是基于传统的查找算法。

参考文献[4]提出了通过定义策略简化网络的管理,同时提出了相关算法,可用在传统网络的冲突策略和不可达策略的检测中;参考文献[5,6]提出了一种基于“first-order”逻辑来定义可能冲突的方法来检测冲突,并且在 OpenFlow 控制器下发流表前部署了“推理引擎”来检测流表冲突。

参考文献[7]介绍了一种通过使用 FlowVisor 技术来检测 OpenFlow 网络中潜在的冲突问题,FlowVisor 是一种有着特殊用处的 OpenFlow 控制器,它可以在 OpenFlow 交换机和多台 OpenFlow 控制器之间充当一个代理,他们提出的冲突检测算法是:基于散列字典树(hash-trie based conflict detection, HTCD)算法及基于冲突检测的本体论(ontology based conflict detection, OCD)。

参考文献[8]为 SDN 提出了一种减少比特向量(RBV reduced bit vector)算法对流表进行冲突检测。RBV 算法用到了比特矢量的概念并且采用



了按组分类的算法,将相同前缀长度的流表分到一个组中,从而减少了多余的比特矢量。

参考文献[9]提出了安全策略执行内核(FortNOX)作为对开放源 NOX OpenFlow 控制的一个扩展, FortNOX 采用了一种基于别名检测的方法来确定每一个应用是否都被安全授权,强制优先级最低的策略进入仲裁。

参考文献[10]提出了称为“FlowGuard”的综合框架,应用在动态的 OpenFlow 网络中检测和解决与防火墙相冲突的策略, FlowGuard 在网络状态更新时会对网络中的流表进行检测是否与防火墙策略相冲突。

参考文献[11]提出全网内所有转发规则在进行插入、更新和删除操作时,使用 VeriFlow 动态地检测违反规则的流表,他们在传统分组分类算法的基础上提出了多维前缀树,一棵多维前缀树是一种用来存储组合数组的有序的树状数据结构,这棵树与符合流表规则的数据分组相关联;参考文献[10]提出了一种形式化的模型来检测内部功能是否有异常,这里异常的定义是在同一个网络中,是否有两个或更多的相同功能被部署。

参考文献[13]提出了使用 ADRS (anomaly detecting and resolving for SDN) 解决策略和规则中存在的异常,采用了间隔树模型快速扫描流表,然后建立一个共享模型来分配网络的优先级,通过这两个模型,提出了一种自动化的算法来检测和解决

SDN 模型中存在的流表异常问题。

参考文献[14]提出了一种基于 FlowPath 的实时动态策略冲突检测及解决办法,通过获取实时的 SDN 状态检测防火墙策略的直接和间接冲突,基于 FlowPath 进行自动化和细粒度的解决冲突。

在以上所有相关研究中,未有对超大规模流表项进行冲突检测的研究及相关实验结果,也无人提出使用与机器学相关的方法进行检测,本文首次提出一种基于深度学习的适应于超大规模流表项冲突检测的方法。

3 基于深度学习的冲突检测

3.1 冲突检测架构

当应用/编排层部署新的应用时,需检测下发到某一交换机内的策略是否与之前已存在的流表项存在冲突,使用深度学习进行检测时,以需要检测的所有流表项的域作为模型特征。由于 SDN 中的交换机内的流表项数目将随着应用的增长而不断增长,深度学习难以对非固定大小的输入进行检测,采用基于固定窗口的循环检测方法,检测架构如图 2 所示。

通常应用/编排层(或控制器)中维护着 SDN 中所有交换机的流表信息,假设当前应用产生了某条策略,需要下发到某一 OpenFlow 交换机,此时将该交换机内获取的所有流表信息按每 X 条流表项来分割,当有一条新的流表项需要下发时,将这条流表项与已分割的 X 条流表项组成一个

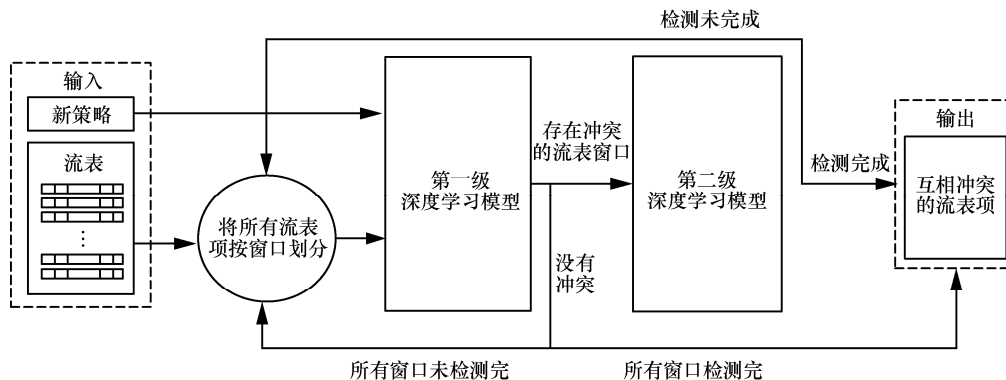


图 2 流表冲突检测架构

$X+1$ 条流表项的窗口, 作为第一级深度学习模型的输入数据。其中, 第一级深度学习模型的输入数据是 $X+1$ 条流表项的所有信息特征。如果 OpenFlow 控制器中已存在的流表项数目不足 X 条, 则用无效的填充信息将窗口填充到 X 条流表项。如果分割后的片段中的流表项数目不足 X 条, 则用其他窗口中的流表项将此窗口的流表项数目填充到 X 条。分别将所有的窗口作为第一级深度学习模型的输入来判断这个窗口中的所有流表项是否存在冲突。如果存在冲突, 则将该窗口的每条流表项与即将下发的流表项组成一个窗口, 作为第二级深度学习模型的输入数据来定位具体的两条冲突的流表项。其中, 第二级深度学习模型是以两条流表项的所有特征信息作为该模型的输入数据。

3.2 深度学习模型

根据 OpenFlow1.5 协议, OpenFlow 匹配域共包括 45 个字段, 取其中的 ETH_DST、ETH_SRC、ETH_TYPE、IPv4_SRC、IPv4_DST、IPv4_SRC_MASK 和 IPv4_DST_MASK 作为深度学习模型的输入特征。其中, ETH_DST 为目的 MAC 地址, ETH_SRC 为源 MAC 地址, ETH_TYPE 为以太网协议类型, IPv4_SRC 为源 IP 地址, IPv4_DST 为目的 IP 地址, IPv4_SRC_MASK 为源 IP 地址的子网掩码, IPv4_DST_MASK 为目的 IP 地址的子网掩码。最终, 这些字段的值将会被转换成二进制值作为深度学习模型的输入特征。

第一级深度学习模型的输入层为 CNN 层, 隐藏层由 M 层 CNN 层和 N 层全连接层组合, 输出层为只有一个神经元的全连接层。输入层和隐藏层使用的激活函数为 ReLU (rectified linear unit) 函数, 该函数为非线性函数, 函数为:

$$f(x) = \max(0, x) \quad (1)$$

输出层的激活函数为非线性激活函数 Sigmoid 函数, 为:

$$f(x) = 1 / (1 + e^{-x}) \quad (2)$$

为了简化第二级深度学习模型, 并取效果最好的一个模型, 将第二级深度学习模型总共分为两类: 一类模型包括 CNN 层和全连接层; 另一类模型只有全连接层。前者的输入层为 CNN 层, 隐藏层都为全连接层, 输出层为只有一个神经元的全连接层。输入层和隐藏层的激活函数都为 ReLU 函数, 输出层的激活函数为 Sigmoid 函数。后者的输入层、隐藏层和输出层都为全连接层, 激活函数的分布情况与前者相同。

第一级深度学习模型和第二级深度学习模型的总体结构分别如图 3、图 4 所示。

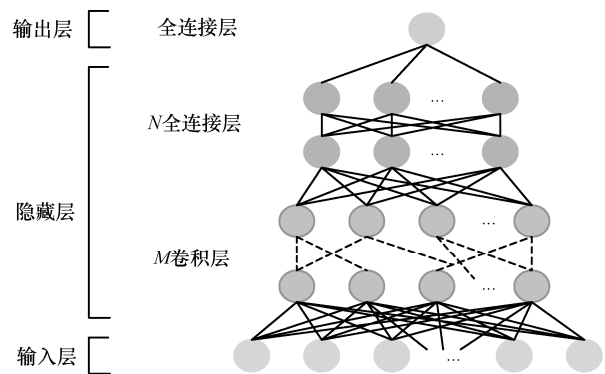


图3 第一级深度学习模型总体结构

本文将使用的深度学习模型结构用如下方式描述:

$$Model = (I, (K_0, K_1, \dots, K_i, \dots, K_n), [T, O]) \quad (3)$$

其中, $K_i = [T, L, n_0, n_1, \dots, n_i, \dots, n_L]$ 。“ I ”表示输入层神经元的个数; “ T ”表示层的类型, 卷积层用“C”表示, 全连接层用“F”表示; “ L ”表示层的数量; “ n_i ”表示隐藏层中每一层的神经元个数; “ O ”表示输出层神经元的个数。

模型训练的特征个数越多, 达到高精度模型所需的样本数量就越多。由于第一级深度学习模型具有的输入特征个数有 24 000 个, 特征数较多, 所以训练第一级深度学习模型时采用 1 200 000 个样本的数据集。为了获取最佳的 SDN 策略冲突检测的第一级深度学习模型, 本文构建了多个深度学习模型进行比较。

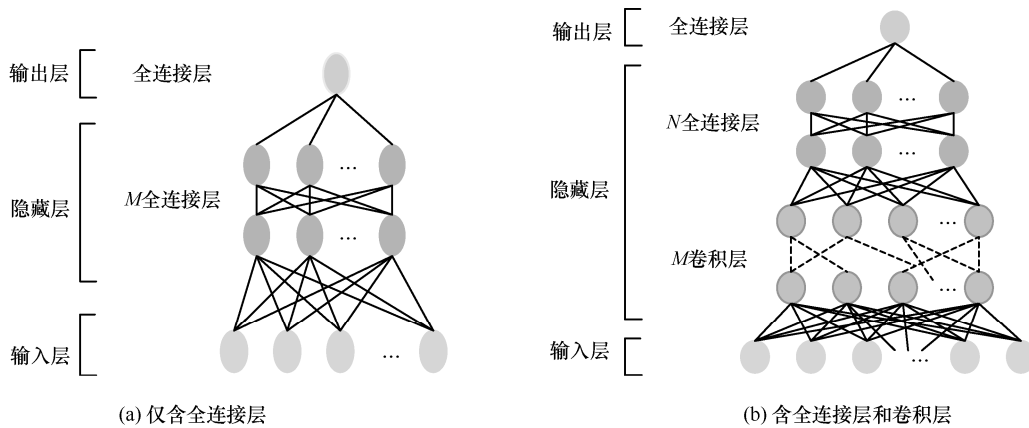


图4 第二级深度学习模型结构

本文进行训练时所使用的的第一级深度学习模型如下：

1>C3F4_1=(24 000,["C",3,512,256,128],["F",3,128,128, 128],["F",1]);

2>C3F4_2=(24 000,["C",3,128,128,64],["F",3,64,64,64],["F",1]);

3>C3F4_3=(24 000,["C",3,128,128,64],["F",3,64,32,16],["F",1]);

4>C3F3=(24 000,["C",3,128,64,32],["F",2,32,16],["F",1]);

5>C3F2=(24 000,["C",3,80,80,64],["F",1,64],["F",1]);

6>C3F5=(24 000,["C",3,128,128,128],["F",4,128,64,32,16],["F",1]);

7>C4F4=(24 000,["C",4,128,128,64,64],["F",3,64,32,16],["F",1]);

8>C5F2=(24 000,["C",5,128,64,32,16,8],["F",1,8],["F",1])。

由于第二级深度学习模型具有的输入特征个数为 480 个，特征数较少，因此训练第二级深度学习模型时采用了 400 000 个样本的数据集，且为了获取最佳的第二级深度学习模型，本文构建了两种不同类型的多个深度学习模型进行比较：一种类型是最简单的模型，模型中所有层都为全连接层；另一种类型是稍微复杂的模型，模型中包括卷积层和全连接层。

根据模型的描述方式，本文进行训练的不同类型的多个第二级深度学习模型描述形式如下：

1>F3=(24 000,(["F",2,256,128]),["F",1]);

2>F4=(24 000,(["F",3,256,128,64]),["F",1]);

3>C1F4=(24 000,(["C",1,512],["F",3,512,256,128]),["F",1]);

4>C1F3=(24 000,(["C",1,256],["F",2,256,128]),["F",1])。

4 实验实现与评估

4.1 数据集产生和预处理

当前使用流表项中的 ETH_DST、ETH_SRC、IP_PROTO、IPv4_SRC、IPv4_DST、IPv4_SRC_MASK 和 IPv4_DST_MASK 字段作为每一条流表项的特征。第一级深度学习模型输入的是 100 条流表项的特征信息，所以对于第一级深度学习模型的单位模型输入特征包括了 100 条流表项的所有特征信息。第一级深度学习模型的数据集包括了两类数据，一类为具有冲突流表项的数据，另一类为没有冲突流表项的数据。第一级深度学习模型的数据集产生算法如下。

(1) 产生冲突类型的流表项数据

flow_entry_1 = GenerateRandomFlowEntry()

flow_entry_2 = GenerateConflictFlowEntry
(flow_entry1)

```

while true:
    flow_entry_i = GenerateRandomFlowEntry()
    if isConflict(flow_entry_i, conflict_
flow_entries) == False THEN
        conflict_flow_entries.append(flow_
entry_i)
        break
    end if
number of conflict_flow_entries: m
if m < 100 then
    go to 3
end if
Shuffle (conflict_flow_entries)
    conflict_sample.append(conflict_
flow_entries)
number of conflict_sample: n
if n < 600 000 then
    go to 1
end if

```

上述算法中一个冲突样本的生成方法：先随机产生一条流表项，后根据冲突规则生成一条与之冲突的流表项，再生成其余与上两条流表项均不冲突的 98 条流表项，最后将上述 100 个流表项顺序打乱。按此方法生成 600 000 个冲突样本。

(2) 产生非冲突类型的流表项

```

flow_entry_1 = GenerateRandomFlowEntry()
while true:
    flow_entry_i = GenerateRandomFlowEntry()
    if isConflict(flow_entry_i, non_ con-
flict_flow_entries) == false then
        non_conflict_flow_entries.append(flow_entry_i)
        break
    end if
number of non_conflict_flow_entries: m
if m < 100 then
    go to 2

```

```

end if
Shuffle(non_conflict_flow_entries)
    non_conflict_sample.append(flow_entries)
number of non_conflict_sample: n
if n < 600000 then
    go to 1
end if

```

一个非冲突流表项的生成的过程与冲突样本生成的过程类似，但生成的规则不同，按此方法生成 600 000 个非冲突样本。

在对使用上述算法产生的流表项进行预处理时，分别将冲突类型和非冲突类型的流表项数据转换成二进制形式，并将每个冲突类型的样本打上标签 1，将每个非冲突类型的样本打上标签 0。最后将冲突类型和非冲突类型的流表项数据合并，并且打乱顺序，最后的数据集含 1 200 000 个样本，冲突和非冲突占比为 1:1。在第 4.2 节所述的实验环境下，生成第一级模型的训练样本时间约为 5 h。

第二级深度学习模型输入的是 2 条流表项的特征信息，所以对于第二级深度学习模型的单位模型输入特征包括了 2 条流表项的特征信息。第二级深度学习模型的数据集同样包括两类数据，即具有冲突流表项的数据和没有冲突流表项的数据。第二级深度学习模型的数据集产生算法与第一级深度学习模型的数据集产生流程相似。使用了用该算法产生的 400 000 个样本对第二级模型进行训练和测试，其中冲突和非冲突占比为 1:1。第二级模型的训练样本生成时间约为 15 min。

4.2 实验结果

本文涉及的实验环境如下：两块 Nvidia K80 的显卡，128 GB 的内存，操作系统为 Ubuntu14.04，使用了基于 Keras 的深度学习框架。

针对表 2 不同的第一级深度学习模型，使用含 1 200 000 个样本的数据集进行训练，最后的测



表 2 第一级深度学习模型的测试精度

参数名	C3F4_1	C3F4_2	C3F4_3	C3F3	C3F2	C4F4	C5F2
filter_length1	3	3	3	2	5	5	2
filter_length2	2	2	2	2	3	3	2
filter_length3	2	2	2	2	3	3	2
filter_length4	/	/	/	/	/	2	2
filter_length5	/	/	/	/	/	/	2
训练精度	90.06%	90.03%	91.23%	90.82%	90.68%	88.90%	89.38%
测试精度	90.14%	91.20%	91.46%	90.84%	90.24%	89.60%	89.24%
AUC	96.45%	96.91%	97.04%	97.00%	96.61%	96.10%	96.39%

表 3 第二级深度学习模型的测试精度

参数名	F3	F4	C1F4	C1F3
filter_length	/	/	1	1
训练精度	98.53%	98.21%	98.50%	97.77%
测试精度	99.35%	98.35%	98.59%	98.60%
AUC	99.97%	99.96%	99.80%	99.93%

试精度见表 2。通过表 2 可以看出, C3F4_3 模型是测试精确度最高的模型, 因此本文采取的第一级深度学习模型是 C3F4_3 模型。

针对表 3 不同的第二级深度学习模型, 使用含 400 000 个样本的数据集进行训练, 最后的测试精度见表 3。通过表 3 可以看出 F3 模型是测试精度最高的模型, 因此本文采取的第二级深度学习模型是 F3 模型。

4.3 实验评估

本文对流表冲突检测的深度学习模型与机器学习中的 RandomForest 分类器^[13]进行了比较。在与第一级冲突检测相同的硬件环境下, 基于 Ubuntu 14.04 及 scikit-learn 机器学习框架, 使用相同的数据集, 使用 RandomForest (随机森林) 分类器进行了实验, 与深度学习的结果比较见表 4。

从该结果可以看出, 训练相同的样本数量, 深度学习方法的测试精度比其他机器学习方法的

表 4 深度学习方法和机器学习方法的比较

样本个数/个	深度学习(C3F4)	随机森林
400 000	89.56%	62.7%
1 200 000	91.46%	内存溢出

测试精度高。其中, 随机森林分类器在训练 1 200 000 个样本数据时, 由于内存溢出而无法获得具体的精度。

同时对基于深度学习的检测方法的检测时间进行了测试。在实验过程中, 在不同大小的原始流表项作为检测对象时, 第一级深度学习模型的检测时间见表 5。

第二级深度学习模型定位出具体冲突流表项 (即检测 100 个第二级深度学习模型的窗口) 的时间为 0.206 s。冲突检测总时间为第一级深度学习模型和第二级深度学习模型检测时间之和。当 OpenFlow 控制器中所有的流表项与即将下发的流表项没有冲突流表项时, 则无需使用第二级深

表 5 第一级深度学习模型的检测时间

流表项个数/个	1 000	10 000	25 000	50 000	100 000	200 000
检测时间/s	0.341	0.607	0.957	1.677	2.906	5.631

表 6 first-order logic 方法下不同数量流表项冲突检测时间

流表项个数/个	1 000	10 000	25 000	50 000	100 000
检测时间/s	0.010	0.172	0.817	9.079	257.675

深度学习模型进行检测,该级时间为 0;当 OpenFlow 交换机中的某一条流表项与即将下发的流表项有一个冲突时,第二级深度学习模型冲突检测时间为 0.206 s。当与 n 个第一级深度学习模型窗口内的流表项有冲突时,第二级检测时间为 $n \times 0.206$ s,总时间随着流表项的大小呈线性增长。

参考文献[5,6]中使用的 first-order logic 算法对流表冲突检测时的检测时间见表 6。

使用两种检测方法所耗费的总时间比较如图 5 所示。从图 5 中可以看出,在对小规模数量的流表项进行冲突检测时,参考文献[5]使用的基于 first-order logic 方法,与基于深度学习方法相比,检测时间没有太大区别;而在对大规模数量的流表项进行流表冲突检测时,参考文献[3]所使用的 first-order logic 算法,检测时间呈指数级增长,而深度学习所需时间则呈线性增长,远小于传统算法的检测时间。

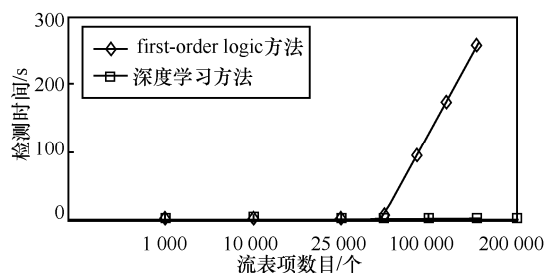


图 5 深度学习方法与 first-order logic 方法冲突检测时间比较

参考文献[6]提出使用 HTCD 算法及 OCD 算法对冲突进行检测。前者的平均算法时间复杂度为 $O(nd)$, 后者为 $O(n)$ 。其中, n 为流表项总数, d 为数据分组头检测域的个数。而本文提出的基于

深度学习的冲突检测方法,如采用窗口分割方法将所有 n 条流表项分割成每个窗口含 k ($k > 100$) 条流表项的 m 个窗口,则在任何情况下时间复杂度均为 $O(n/k)$ 。

参考文献[14]中提出的基于 FlowPath 的 SDN 安全策略动态冲突检测方法。在一个有 h 台主机、1 台交换机、平均每台交换机中的流表项数为 n 的情况下。找出全网的所有冲突的算法查找次数为:

最坏情况: $h \times n \times 1$ 次比较 (全部搜索);

平均情况: $h \times (n/2) \times (1/2)$ 次比较;

最佳情况: h 次比较 (比较时一次命中)。

本文采用两级深度学习的冲突检测算法,要完成对所有流表项的冲突检测,第一级只需完成对这 m 个窗口的基于深度学习模型的检测,检测次数为 $m = \lceil n/k \rceil + 1$,第二级检测只有在定量检测,并检测到某个窗口存在冲突时才会被启动,如果仅定性检测某流表是否与系统内流表存在冲突,则 $m = \lceil n/k \rceil + 1$ 次即最坏情况,该值远小于参考文献[14]中提到的平均检测次数及最坏次数。

5 结束语

本文通过将深度学习与基于 OpenFlow 的 SDN 相结合,提出了一种新的用来检测基于 OpenFlow 的 SDN 中的流表冲突的方法。该方法采用第一级深度学习模型定性判断在所有流表中是否存在冲突,并采用第二级深度学习模型定位出具体冲突的流表项。通过实验数据可以看出,当 OpenFlow 网络中存在大量的流表时,本文提出



的方法在检测冲突流表项所花费的时间上远小于传统的流表冲突检测方法。因此, 这种新方法适合于对超大规模流表进行冲突检测。

但是, 任何算法都不能保证达到 100% 的检测精确度, 基于深度学习的算法也存在同样的问题, 该方法的精确度与各种因素有关, 如训练模型时所使用的数据集的大小、数据集的特征选择、深度学习模型的层数及每层所使用的模型等, 在后续的研究过程中, 将朝此方向努力, 使该方法的精度能得到进一步的提高。

参考文献:

- [1] 韦乐平. SDN 的战略性思考[J]. 电信科学, 2015, 31(1): 7-12.
WEI L P. Strategic Thinking on SDN [J]. Telecommunications Science, 2015, 31(1): 7-12.
- [2] CLOUGHERTY M M, WHITE C A, VISWANATHAN H, et al. SDN 在 IP 网络演进中的作用[J]. 电信科学, 2014, 30(5): 1-13.
CLOUGHERTY M M, WHITE C A, VISWANATHAN H, et al. Role of SDN in IP network evolution[J]. Telecommunications Science, 2014, 30(5): 1-13.
- [3] SPECIFICATION O F S. Version 1.5. 1(wire protocol 0x01)[EB]. 2015.
- [4] VERMA D C. Simplify network administration using policy-based management[J]. Network IEEE, 2002, 16(2): 20-26.
- [5] LOPES A B B, LIMA DE C G A, FERNANDEZ M P. Flow-based conflict detection in OpenFlow networks using first-order logic[C]//2014 IEEE Symposium on Computers and Communication (ISCC), April 27-May 2, 2014, Toronto, Japan. New Jersey: IEEE Press, 2014: 1-6.
- [6] BATISTA B L A, DE CAMPOS G A L, FERNANDEZ M P. A proposal of policy based OpenFlow network management[C]//2013 20th International Conference on Telecommunications (ICT), Aug 14-16, 2013, Guilin, China. New Jersey: IEEE Press, 2013: 1-5.
- [7] NATARAJAN S, HUANG X, WOLF T. Efficient conflict detection in flow-based virtualized networks[C]//2012 International Conference on Computing, Networking and Communications (ICNC), Oct 8-10, 2012, Maui, Hawaii. New Jersey: IEEE Press, 2012: 690-696.
- [8] LO C C, WU P Y, KUO Y H. Flow entry conflict detection scheme for software-defined network[C]//2015 International Telecommunication Networks and Applications Conference(ITNAC), November 18, 2015, Sydney, Australia. New Jersey: IEEE Press, 2015: 220-225.
- [9] PORRAS P, SHIN S, YEGNESWARAN V, et al. A security enforcement kernel for OpenFlow networks[C]//The First Workshop on Hot Topics in Software Defined Networks, August 13, 2012, Helsinki, Finland. New York: ACM Press, 2012: 121-126.
- [10] HU H, HAN W, AHN G J, et al. FLOWGUARD: building robust firewalls for software-defined networks[C]//The Third Workshop on Hot Topics in Software Defined Networking, August 22, 2014, Chicago, Illinois, USA. New York: ACM Press, 2014: 97-102.
- [11] KHURSHID A, ZOU X, ZHOU W, et al. VerIFlow: verifying network-wide invariants in real time[C]//10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), Apr 3-5, 2013, Lombard, IL, USA. [S.l.:s.n.], 2013: 15-27.
- [12] BASILE C, CANAVESE D, LIOY A, et al. Inter - function anomaly analysis for correct SDN/NFV deployment[J]. International Journal of Network Management, 2016, 26(1): 25-43.
- [13] WANG P, HUANG L, XU H, et al. Rule anomalies detecting and resolving for software defined networks[C]//2015 IEEE Global Communications Conference (GLOBECOM), Dec 6-10, 2015, San Diego, USA. New Jersey: IEEE Press, 2015: 1-6.
- [14] 王娟, 王江, 焦虹阳, 等. 一种基于 OpenFlow 的 SDN 访问控制策略实时冲突检测与解决办法[J]. 计算机学报, 2015, 38(4): 872-883.
WANG J, WANG J, JIAO H Y, et al. A method of OpenFlow-based real-time conflict detection and resolution for SDN access control policies[J]. Chinese Journal of Computers, 2015, 38(4): 872-883.
- [15] LIAW A, WIENER M. Classification and regression by randomForest[J]. R News, 2002, 2(3): 18-22.

[作者简介]



李传煌 (1980—), 男, 博士, 浙江工商大学信息与电气工程学院副教授, 2016 年美国佛罗里达大学访问学者, 主要研究方向为软件定义网络、深度学习、开放可编程网络、系统性能预测和分析模型, 发表 EI/SCI 检索论文 40 余篇, 申请专利 15 项。

程成 (1993—), 男, 浙江工商大学信息与电气工程学院硕士生, 主要研究方向为软件定义网络、深度学习。

袁小雍 (1990—), 男, 美国佛罗里达大学博士生, 主要研究方向为网络安全、深度学习、云计算和分布式系统。

岑利杰 (1992—), 男, 浙江工商大学信息与电气工程学院硕士生, 主要研究方向为网络安全、深度学习、软件定义网络。

王伟明 (1964—), 男, 博士, 浙江工商大学信息与电子工程学院教授, 主要研究方向为新一代网络架构、开放可编程网络, 特别是 IETF ForCES、SDN 及可重构网络等方面的协议、模型和算法。