细粒度网络流量分类架构及其优化

李勋[1,2]，唐亚哲[1]
（1. 西安交通大学电信学部，710049，西安；2. 通信网信息传输与分发技术重点实验室，050081，石家庄）

摘要：针对现有网络流量指纹自动生成难度大、粒度粗及匹配阶段内存消耗大等问题，提出了细粒度网络流量分类架构及其优化。在线下，根据特定字符片段在对应流量中保持不变，且代表流量功能的有效字符片段比随机噪声片段出现的频率高这一特性，寻找流量中字符片段出现频率和长度达到一定阈值的有效片段，并将其作为备选指纹规则，通过交并、合并、指纹提纯操作获取该流量对应的指纹。在线上字符串匹配时，根据 k 均值分类思想重新定义距离，并利用异构位分割状态机的启发式算法对指纹中的字符串进行重新组织，对内存使用进行优化。实验结果表明：所提算法能够在未知网络流量协议格式的情况下自动生成细粒度的流量指纹，平均识别准确率为 93.65%，对噪声不敏感；在匹配时若将原所有指纹字符片段重新优化组织，当指纹规则数量在4 000以上时，可节约近 50%的内存需求。

关键词：细粒度网络流量；指纹自动生成；位分割状态机；启发式算法；字符串匹配
中图分类号：TP393.0 **文献标志码**：A
DOI：10.7652/xjtuxb202011015 **文章编号**：0253-987X(2020)11-0121-08

OSID 码

Fine-Grained Network Traffic Classification Architecture and Optimization

LI Xun[1,2], TANG Yazhe[1]
(1. Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China; 2. Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory, Shijiazhuang 050081, China)

Abstract: To solve the problems of coarse-grained network traffic fingerprints and large memory consumption in the matching phase, we proposed a fine-grained network traffic classification architecture to achieve automated signature generation and optimize memory. As some portion of the data payload in a function is invariant and the signature reflecting the application function is atypical in internet traffic, the signature generation is mapped into the problem of obtaining the frequently occurring substrings and their corresponding occurrence frequency. In the case of online matching, according to the idea of k-means classification, the distance is redefined and the rules are reorganized using the heuristic heterogeneous bit-split deterministic finite automaton method to achieve the purpose of optimizing memory usage. The experimental results show that without knowing the format of the network traffic protocol, the method can automatically generate fine-grained traffic fingerprints with an average recognition accuracy of 93.65%, and is not sensitive to noise. In addition, if all fingerprint character fragments are re-optimized during matching, when the number of fingerprint rules is above 4 000, it can save nearly 50% of memory requirements compared with the previous bit-split string matching methods.

Keywords: fine-grained traffic; automated signature generation; bit-split deterministic finite automata; heuristic algorithm; string matching

收稿日期：2020-02-19。 作者简介：李勋(1987—)，男，博士生；唐亚哲(通信作者)，男，副教授。 基金项目：通信网信息传输与分发技术重点实验室开放基金资助项目(SXX18641X024)。
网络出版时间：2020-07-10 网络出版地址：http://kns.cnki.net/kcms/detail/61.1069.T.20200708.1804.002.html

网络流量分类是增强网络可控性的基础技术，有助于网络管理者对各种业务流进行区分并进行实时的监控与管理。传统流量分类是通过识别已知的从链路层到传输层、甚至到常规的 HTTP、FTP 等应用层的协议头来进行的。对于一般的未知的应用层流量分类，则存在较大的挑战。例如，对于常规的 C/S 或者 B/S 架构的应用软件，不同的模块与服务器通信时，其流量从链路层到应用层的已知头部部分是完全相同的。此时，若需要区分不同模块所产生的流量，就不能使用传统流量分类算法。本文将这种同一个程序中不同模块运行所产生的流量分类问题称为细粒度流量分类。显然，细粒度流量分类有两个关键挑战：①在链路层到应用层的所有协议头都一样的情况下，如何从应用层负载中自动获取细粒度流量的指纹规则；②完成指纹规则提取之后，在匹配阶段如何在有限的存储硬件条件下进行快速高效匹配。

目前，细粒度流量识别国内外研究还比较少，人工制作细粒度指纹的算法费时费力，且不一定准确。细粒度流量识别多采用流量统计信息[1-3]进行识别，例如将采集包长、包间隔、比特率等作为特征[4]。这种算法受网络状况影响较大、对噪声敏感，精度很难保证。同时，人工制作出来的流量对应指纹很难将细粒度的相似流量进行有效区分。另外，在进行离线指纹获取时，现有算法对噪声比较敏感，需要捕获更加干净的流量来提升准确率[5]。实际中，当一个特征频繁发生时，可以把它看作是这个事物的一个表征。在流量识别中，例如一个细粒度的登陆操作所对应的负载中有很高的概率频繁出现 login 相关的字符片段，完全就可把这样的片段和出现的最少频率组合作为细粒度的登录操作的指纹之一。在其他领域，也有类似的问题和类似的处理算法。例如在计算机安全中识别蠕虫问题，就是通过识别出现频率较高的特定片段来识别蠕虫病毒。在生物领域，挖掘生物序列往往也是挖掘频繁片段。首先，在生物基因序列中相同或者相似的序列往往有着相同或者相似的生物外在表现。其次，通常大部分 DNA 序列都被认为是噪声，起关键作用的 DNA 序列往往只是很小的几个片段。

通常情况下，得到的细粒度指纹中包含的规则（通常为字符串）数非常大，且这些字符串长短不一，会导致匹配过程中对内存的大量消耗。以往的匹配算法[6-8]当字符串数量很大时，都存在内存消耗过大的问题。为了克服这些问题，目前已经提出了基于平均长度的算法[9]、基于字典序的算法[10]以及基于确定性有限自动机（DFA）的字符串匹配算法[11]。尽管研究者在 FPGA[12-14]和 TCAM[15]方面做了很多降低内存需求的工作，但状态机的稀疏矩阵问题依旧存在，导致大量的内存浪费。因此，如何提高内存使用效率也是细粒度网络流量识别的重要研究问题。

针对现有网络流量指纹自动生成难度大、匹配阶段内存消耗大这两个关键挑战，本文给出了细粒度网络流量识别算法。针对第一个挑战的解决算法是：对网络功能对应流量进行多次采集，然后进行交集处理，去除网络噪声流量对制定指纹规则准确性的影响；在指纹生成环节将底层处理好的流量按照长字符串进行对待，计算并记录频繁出现的字符片段和对应出现的数量；通过交并、合并、指纹提纯操作最终获取对应的细粒度指纹规则。针对第二个挑战，利用启发式算法进行优化解决：将每一个细粒度功能对应的指纹中的字符串划分为指定数目的集合，根据每个集合中的内容，按需申请内存，构建位分割状态机。实验结果表明：本文算法能自动生成细粒度的应用模块指纹规则，识别准确率平均为 93.65%，且对噪声不敏感；当指纹规则数在 4 000 以上时，与传统位分割字符串匹配算法相比，本文算法可以节省近 50% 的内存消耗。

1 细粒度的指纹自动生成

为了能够自动获取流量的细粒度指纹规则，首先要考虑本文算法要寻找的细粒度指纹规则有什么样的特征。线下自动获取细粒度指纹的目标是找到长度大于给定长度且出现频率高于给定频率的所有片段。这一思路主要基于两个事实：①特定细粒度功能对应的负载中有一部分特定的片段是固定不变的，并且这部分字符串片段在网络流量中很少出现；②这些有效片段比随机噪声出现得更频繁。

此问题的数学描述如下。假设有流量 $\sigma=\langle a_1,\cdots,a_n\rangle$，片段 $\sigma'=\langle a_i,\cdots,a_j\rangle$，其中，每个 a_n 是字符，$i,j\in(1,\cdots,n)$ 且 $i<j$，则细粒度指纹提取任务就是：如果存在 σ'，且 $F_{\text{size}}\geqslant f_{\text{size}}$ 且 $F_N\geqslant m'$，那么就把键值对 (σ',F_N) 插入候选指纹集合，最终指纹将从指纹候选集合中进行提取，其中，$F_{\text{size}}=j-i+1$ 为片段 σ' 长度，f_{size} 为片段长度最小阈值，F_N 为片段 σ' 在流量 σ 中出现的数量，m' 为最小出现数量阈值。

由此，本文提出自动生成细粒度网络程序功能流量指纹的算法（STAFF），流程如图 1 所示。

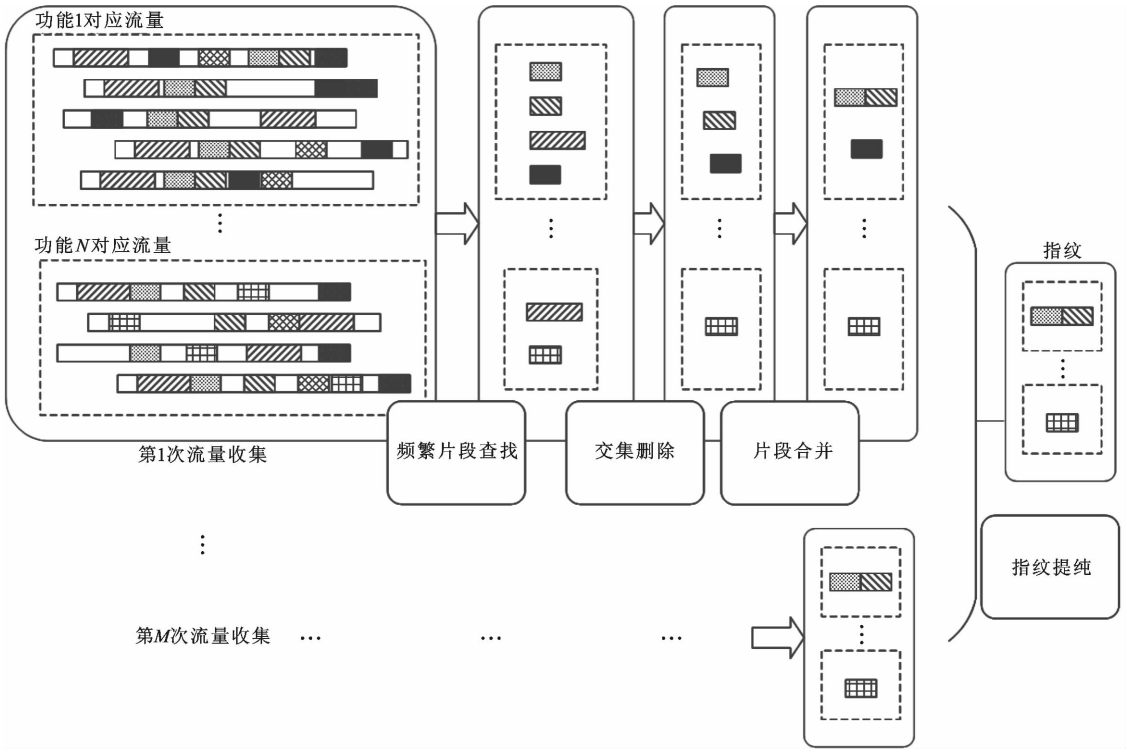


图 1　细粒度网络程序功能指纹自动生成算法流程

STAFF 具体包括以下步骤。

步骤 1:按照程序功能的不同,采集对应的流量数据包。

步骤 2:将不同功能的流量数据包按照长字符串对待,并将其分成固定长度的片段。如果一个片段频繁在数据包中出现,出现的频率大于等于预先设置好的阈值,则将此片段将被选中作为频繁片段。

步骤 3:得到频繁片段后,找出频繁片段之间的交集集合,在不同功能对应的频繁片段中删除相同片段。

步骤 4:将每一个程序功能对应的频繁片段集合中的片段进行合并,保留合并后片段及其出现数量,删除合并之前的两个短字符片段及其出现数量,循环该步骤直到不能合并为止,最终频繁片段集合为初步指纹。

步骤 5:循环步骤 1 到步骤 4 多次,对相同的功能制定的多次初步指纹进行指纹提纯求交集,最后生成程序各功能的最终指纹。

STAFF 算法可以识别同一应用程序中的不同功能并且对流量噪声不敏感,所以不要求输入流量完全来自相同的应用程序。

2　启发式减少内存匹配算法

按照第 1 节的算法可以自动获取细粒度网络流量指纹,但在当前高速发展的网络背景下,指纹中字符串的数量会急剧增加,而字符串的长度也可能长短不一。所以,本文利用启发式算法来克服不同模式长度带来的状态机稀疏矩阵问题。

2.1　异构有限状态机体系结构

为了减少存储状态转换的内存需求,对文献[16]中提出的基于位分割 DFA 的字符串匹配算法和匹配架构进行优化。通过将一个或多个 ASCII 字符码拆分为几个位组,可以为每个输入位构造多个 DFA,从而减少每个状态的状态转换总数。为了在状态中识别匹配的目标模式,每个状态都应该包含自己的匹配向量和一组部分匹配向量(PMV),PMV 每个位表示相关的目标模式在状态中是否匹配。在这种算法中,存储匹配向量的内存需求可能非常大。同构的有限状态机(FSM)的大小由 FSM 中最大的规则状态节点数决定,必然会有空间浪费。异构意味着每个 FSM 可以有不同的内存大小,分配给每个有限状态机的内存大小是根据每个状态机自身需求确定的。要能发挥异构 FSM 的优势,关键是要找到合适的指纹字符串划分算法,使具有尽量多公共状态的字符串片段尽量分到一起。

为此,本文提出了一种基于模式分类的异构位分割状态机多模匹配算法(HARD),并从两方面对内存使用进行了优化:①利用文献[11]提出的模式

的唯一性，将规则集合分为唯一模式字符串集合和非唯一模式字符串集合两类，减少字符串的完全匹配向量空间，节省内存空间；②考虑到同构的 FSM 的大小是由最大的规则状态节点数决定的，提出通过异构 FSM 来减少内存需求。根据文献[17]已有架构，提出本文异构 FSM 的体系结构，如图 2 所示。规则块可以在每个循环中接受 lb W_{PMV} 的字节作为输入，输出为每个块匹配向量逻辑和的最终结果。

从图 2 所示的架构中可以估计出内存消耗。第 m 块的第 n 个 FSM 的最小内存需求为

$$M_{\mathrm{FSM}_{mn}}=(2^b\lceil \mathrm{lb}S_{\mathrm{num}_n}\rceil+W_{\mathrm{PMV}_m})S_{\mathrm{num}_n} \quad (1)$$

式中：b 为 FSM 中每次处理的比特数；S_{num_n} 为第 n 个 FSM 的总状态数；W_{PMV} 为 PMV 的长度；$\lceil\cdot\rceil$ 为向上取整。最终总的空间消耗为

$$T_{\mathrm{mem}}=\sum_{m=1}^{G}\sum_{n=1}^{8/b}M_{\mathrm{FSM}_{mn}} \quad (2)$$

式中 G 为分组的数量。

2.2 NP 问题

2.1 节体系结构带来了一个新的问题：如何将指纹规则分成不同的类别来减少位分割 DFA 的状态节点数？可以将此问题定义为不同类分配(ARCP)问题。此问题具体描述如下：对于一个确定的细粒度指纹规则，寻求一种算法将指纹规则分成不同的类别，以此来减少自动状态机的状态节点，最终达到减少存储空间的效果。假设域 i 经过计算后得到一个新的顺序 $\psi(i)$，$i=1,2,\cdots,N$，其空间占用为 $D_{\psi(i)}$，T_N 为所有的全排列，则 ARCP 问题就是为了寻找一个优化的 $\psi(i)$，$i=1,2,\cdots,N$，使得空间开销最小，即

$$\min_{\psi\in T_N}\sum_{i=1}^{N}D_{\psi(i)} \quad (3)$$

实际上，ARCP 问题可以由二次分配问题(QAP)[18]在多项式时间内规约而来。由于 QAP 是 NP 难问题，所以 ARCP 也是 NP 难问题。

利用蛮力法计算 ARCP 会导致不可接受的计算时间。因此，本文提出了一种新的启发式字符串分组算法，该算法针对异构的位分割字符串匹配体系结构，使用特定的测量值来估计字符串之间的相

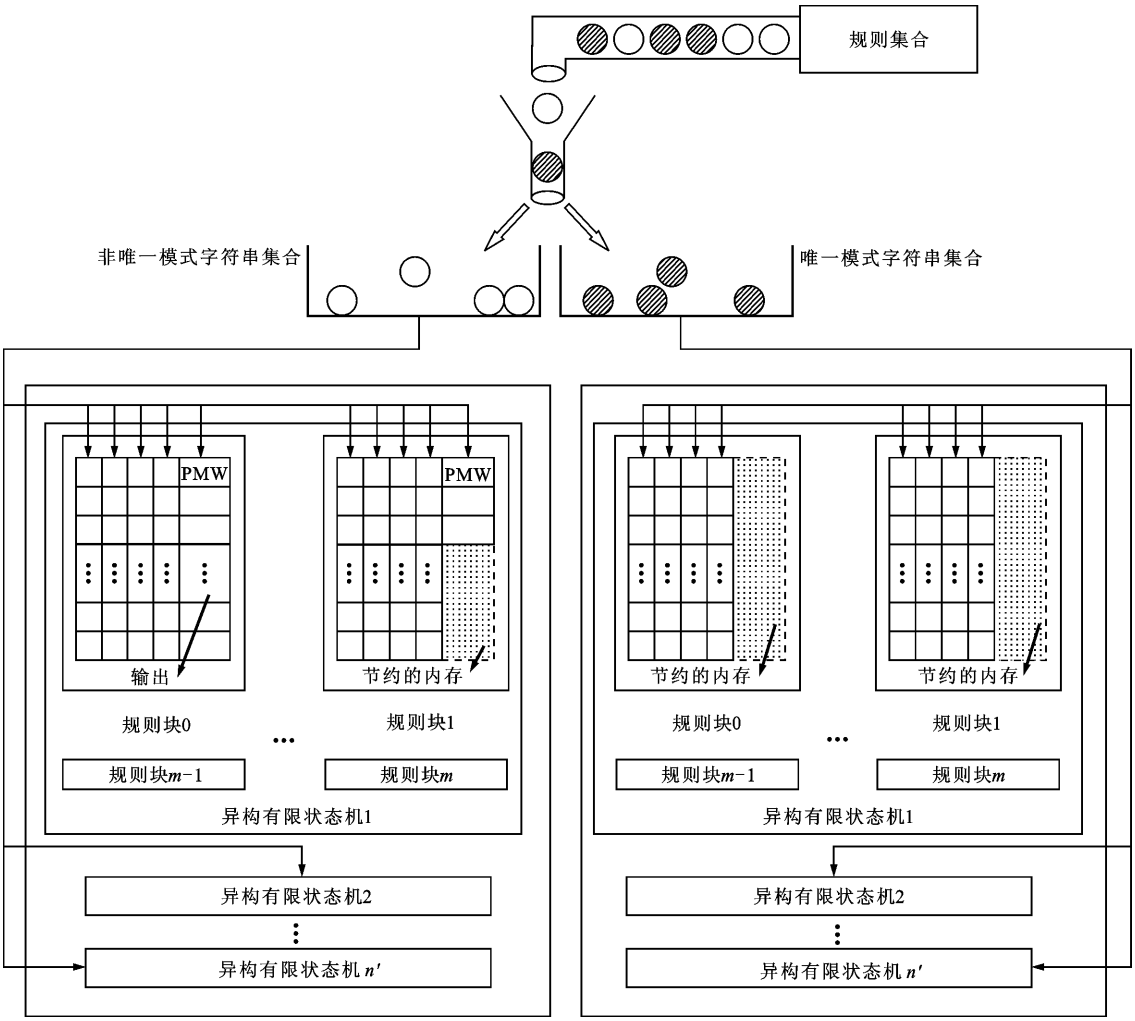


图 2 本文异构 FSM 的体系结构

关性。该测量值在字符串映射到组上起关键作用,可以更有效地分配资源,进一步减少内存使用。

2.3 启发式算法

与 k 均值算法相似,本文算法将目标字符串集划分为指定的子集。算法主要分为两个步骤:选取子集样本点;根据子集样本点划分指纹规则集。

在第一步骤,两个目标字符串之间的通用字符前缀可以减少 FSM 中的状态总数。将具有最长公共前缀的字符串映射到相同的组可以减少内存消耗。定义一个新的距离来度量两个字符串之间的关系。d_{ij} 为将第 i 个和第 j 个字符串划分到同一集合中的得分,公式为

$$d_{ij}=\frac{s'_{\mathrm{num}}(i_s,j_s)}{s_{\mathrm{num}}(i_s)+s_{\mathrm{num}}(j_s)} \tag{4}$$

式中:$s_{\mathrm{num}}(i_s)$ 表示字符串 i_s 在 FSM 上需要的状态节点数;$s'_{\mathrm{num}}(i_s,j_s)$ 表示字符串 i_s 和字符串 j_s 分为一组后 FSM 上所需的状态节点数。d_{ij} 越小,则最终的距离越小。首先,第一个字符串可以分配给第一组,然后计算出这个字符串和剩余所有字符串的距离。如果需要将字符串分为两类,那么将和第一类相关度最小的字符串分配给第二组。这个过程可以一直持续到所有分组的第一个字符串确定后为止。

在第二步骤,以迭代的方式将剩余的字符串映射到组上。在每个字符串增长迭代中,选择一个未分组的字符串并将其分配给一个组。因为选择顺序对结果有很大的影响,所以首先需要处理那些最近分组距离与其他分组距离差异最大的指纹。同时,与扩大后的组一起的所有剩余元素的距离都被更新。这个过程一直持续到所有的字符串都被分组。元素 j_s 与组 i_g 之间的距离为

$$D_{ij}=\frac{s'''_{\mathrm{num}}(i_g,j_s)}{s''_{\mathrm{num}}(i_g)+s_{\mathrm{num}}(j_s)} \tag{5}$$

式中:$s''_{\mathrm{num}}(i_g)$ 表示分组 i_g 中所有字符串所需的状态总数;$s'''_{\mathrm{num}}(i_g,j_s)$ 表示将字符串 j_s 分到分组 i_g 后,所有字符串所需的状态总数。

3 实验与测试

使用 Python 仿真对本文算法的性能进行了评估。实验使用的平台主频为 3.1 GHz Intel Core i5-2400 2 CPU、内存为 8 GB 的计算机。离线测试数据从某企业在线财务系统中捕获。使用不同的用户名和不同的浏览器(Chrome 和 Mozilla)进行 4 次捕获。流量由 21 个功能流组成,共 7.47 MB,每个功能流量由约 900 个数据包组成。在线匹配阶段数据集合如表 1 所示。

表 1　实验选择的数据集合属性

数据集	数据量/条	数据大小/B	数据长度平均值	数据长度标准差
Snort v2.8.spyware	2 299	26 103	11.4	8.1
Snort v2.8.web-client[19]	1 657	67 527	40.8	22.8
ClamAV[20]	28 786	1 921 305	63.2	40.8
Suricata[21]	2 974	91 216	33.4	19.6

3.1 离线阶段细粒度指纹生成有效性实验

利用识别精度评价本文算法。测试集里有 M' 个正例,若能判断覆盖 N' 个,则识别精度为 N'/M'。识别正确是这样确认的:将一个正例对应的底层流量和网络中随机抓取的大量流量进行混合,然后利用制定出来的指纹识别此带标签的正例,若能够识别出来则为识别正确。

第 1 节提到的细粒度指纹生成算法中有两个重要的参数,分别是片段最小长度 f_{size} 和字符串最小出现数量阈值 m'。选取 f_{size} 的取值范围为 3～8,这是因为英文单词的平均长度为 7.6,并且 80% 的单词长度在 4～10 个字母之间。m' 实验取值集合为 {3,6,7,8,9,10,15,20,25,50}。

图 3 是参数不同时识别精度的变化,可以大致看出:f_{size} 越小,检测精度相对越高;当 $f_{\mathrm{size}}=3$ 时,精度不稳定,摆动幅度很大。作为权衡,$f_{\mathrm{size}}=4$ 是最佳值,后续实验中使用 $f_{\mathrm{size}}=4$。

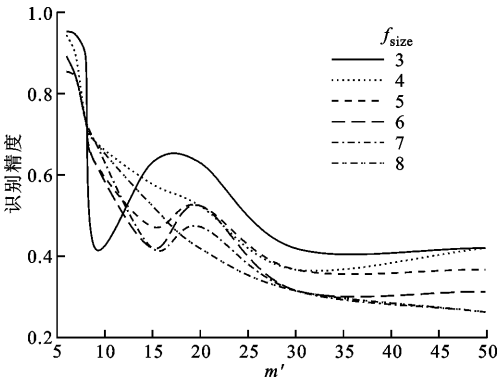


图 3　参数不同时识别精度的变化

图 4 是参数不同时时间开销的变化,可以看出:当 m' 大于 10 时,随着 f_{size} 的增大,时间开销增大,结合图 3 来看,此时识别精度实际上总体在下降;当 m' 在 5～8 时,f_{size} 越大,则计算成本越高,计算结果越准确。作为权衡,m' 建议取为 6。原则上 f_{size} 可

以取任意值，但是，如果选择的 f_{size} 太小，则几乎所有的流量中都出现了目标片段；如果选择的 f_{size} 太大，则很可能无法得到频繁片段。通过实验分析，f_{size} 建议取为 4。

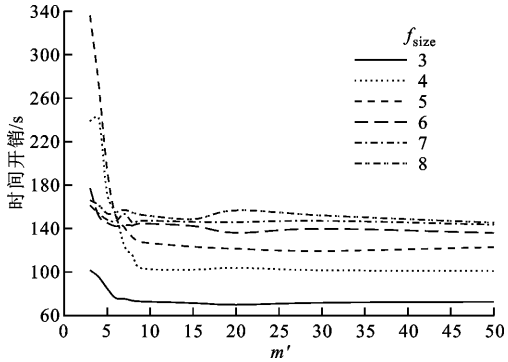


图 4 参数不同时时间开销的变化

为了进一步验证本文算法生成指纹的有效性，选取细粒度网络流量分类的典型算法，即 LASER[22] 和 BS[5] 算法作为对比实验，并分别在微博(Weibo)、财务系统(Financial)、BBS、今日头条(News)、BT 下载、爱奇艺(Iqiyi)6 个系统上捕获流量进行测试。图 5 是对比算法在不同应用流量数据上进行细粒度测试的精度比较，可以看出：STAFF 可以有效区分细粒度的功能，平均识别精度可达 93.65%；LASER 在功能流量数据中寻找最长公共子序列，对协议格式的提取很有效，但当用 LASER 对每个功能进行生成指纹时，所有指纹几乎是相同的，所以不能有效生成细粒度应用功能指纹；BS 只识别在多个流量的前几个数据包中出现的具有多种可选流量特征的唯一行为模式，由于这些特征不够稳定，所以精度波动较大。

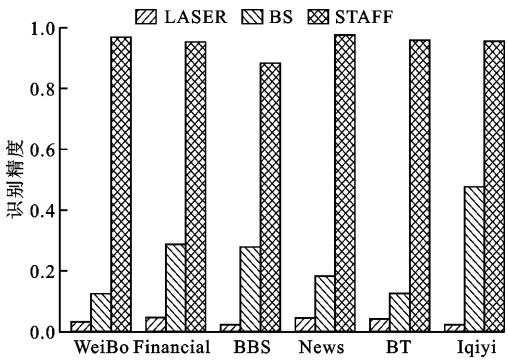
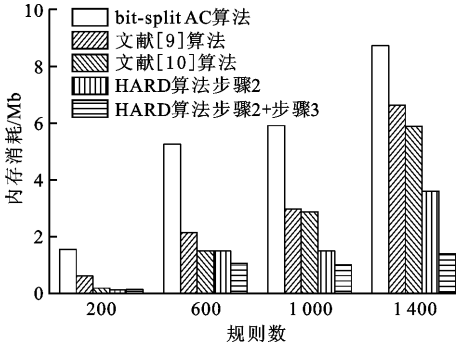


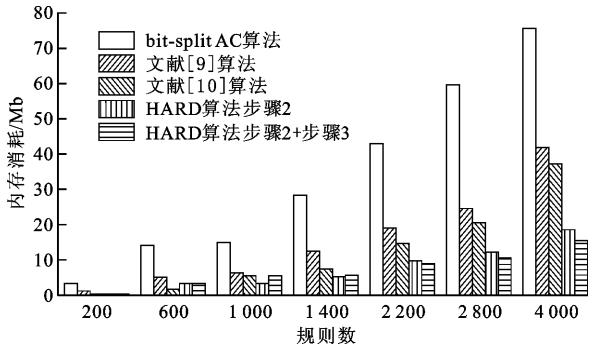
图 5 不同算法在不同应用流量数据上进行细粒度测试的识别精度比较

3.2 匹配阶段启发式算法空间节约实验

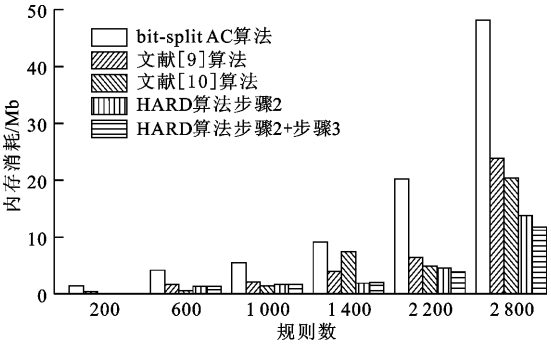
在表 1 的 4 个不同的规则数据集上，将本文 HARD 算法与原始的 bit-split AC 算法、文献[9]、文献[10]算法的性能进行了比较，实验结果如图 6 所示。为了减少相同数据集中规则数量对算法测试结果的影响，随机在同一数据集中选取 200、600、



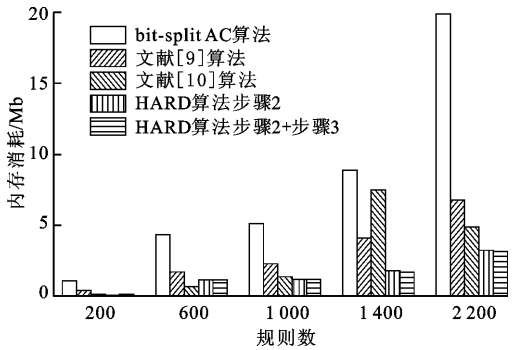
(a)Snort v2.8 spyware



(b)ClamAV



(c)Suricata



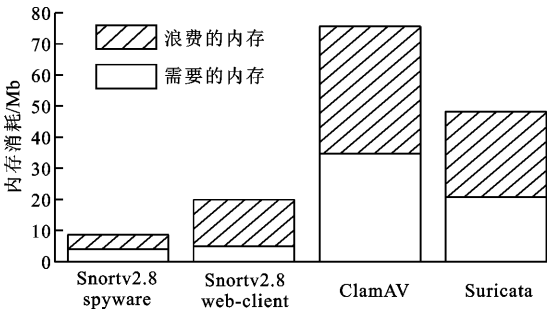
(d)Snort v2.8.web-client

图 6 不同数据集上的内存消耗情况

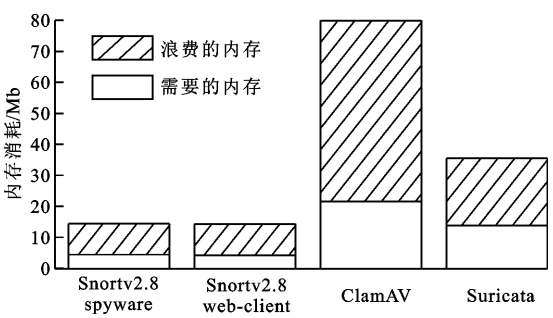
1 000、1 400、2 200、2 800、4 000条规则进行测试,G 设置为 50。

从图 6 可以看出:HARD 算法可以有效节约内存;bit-split AC 算法在所有测试用例中占用的内存最多,这是因为其每一个状态机都使用了统一的内存大小,浪费了大量的内存空间;对比 HARD 算法的步骤 2 和步骤 1+步骤 2 的结果对可知,步骤 1 对算法的整体性能起着重要的作用。值得注意的是,图 6d 中的步骤 1 没有显示显著的内存节省,这是因为数据集中的几乎所有字符串都属于非唯一模式。其次,随着字符串数的增加,HARD 算法的性能更加稳定。文献[9]、文献[10]算法主要是减少 FSM 上的状态数,而不是减少 PMV 的空间使用。从图 6b 也可以看出,在 ClamAV 上,HARD 算法的性能比其他算法的好得多。这是因为 ClamAV 中字符串的长度变化很大。从每个数据集的字符串长度标准差可以看出,HARD 算法在标准差较大的数据集上表现更好。规则集中的字符串数越多,HARD 算法的效果越明显。当指纹中字符串数在 4 000以上时,与其他算法相比,HARD 算法可以节省近 50%的内存消耗。

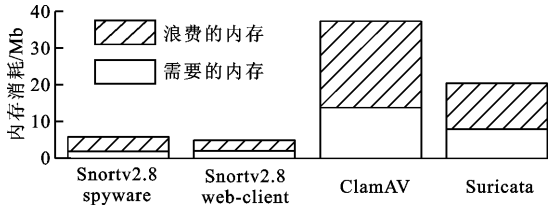
图 7 是不同算法的实际内存消耗和由于结构原因造成的内存浪费情况。可以看出,不同算法的内存需求并没有太大的不同,这是因为它是由指纹中字符串大小本身决定的。然而,不同算法浪费的内存不同,这是因为不同算法使用的数据组织算法和



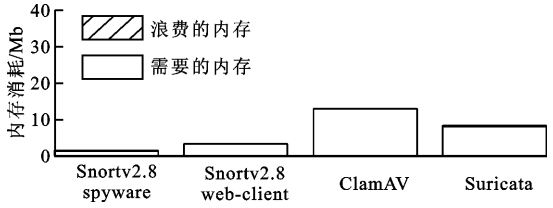
(a)bit-split AC 算法



(b)文献[9]算法



(c)文献[10]算法



(d)本文 HARD 算法

图 7　不同算法的实际内存消耗和由于结构原因造成的空间浪费

数据结构不同。图 7d 中的 HARD 算法按需申请内存空间,几乎没有空间浪费,而其他算法处理不同长度的字符串时,只能根据最长的字符串申请内存,造成了一定的空间浪费。虽然文献[9]算法对公共子序列的组织进行了优化,但每个组中状态机的大小是由状态数最大的指纹决定的。由于所有状态机使用相同大小的存储空间,因此不可避免地会导致更大的内存浪费。虽然文献[10]算法尽量将每个字符串放到平均长度相似的组中,但是不可能所有指纹中字符串的长度都相同,因此仍然存在一定的空间浪费。HARD 算法从两个方面减少了空间消耗:①组内尽量减少状态机的状态数,减少了实际的内存消耗;②每个块中的位分割状态机大小允许不同,有效地减少了浪费的内存。实验结果表明,增加组的数量可以节省更多内存,这在指纹中字符串数量较大的情况下更为显著。

4　结　论

本文提出了一种细粒度网络流量识别算法。分为离线自动细粒度指纹规则生成和在线匹配识别两大部分。离线部分提出的解决算法完全基于原始网络流量数据包,不需要任何协议格式,并且对流量噪声不敏感。在线识别阶段,重新定义距离,利用启发式算法对规则进行重新组织分类,降低状态机状态数。本文主要结论如下。

(1)当一个特征频繁发生时候,就可以将其视为这个事物的一个表征。在获取细粒度流量指纹时,选择长度特定并且出现次数满足一定数量的片段,

将这样的片段作为指纹的方案是逻辑实现简单并且有效的。

(2)利用启发式算法能很好地组织指纹片段,分配给每个有限状态机的内存大小是根据每个状态机自身需求确定的,可节约匹配空间。

细粒度网络流量识别为软件优化、用户画像以及业务关联分析等提供了可能性。加密流量的细粒度分类是未来工作关注的重点。

参考文献:

[1] ERMAN J, MAHANTI A, ARLITT M. QRP05-4: internet traffic identification using machine learning [C]// Proceedings of the IEEE 2016 Global Telecommunications Conference. Piscataway, NJ, USA: IEEE, 2006: 1-6.

[2] WILLIAMS N, ZANDER S, ARMITAGE G. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification [J]. ACM SIGCOMM Computer Communication Review, 2006, 36(5): 5-16.

[3] 徐斌. 细粒度用户级的网络流量分析与应用研究 [D]. 上海: 上海交通大学, 2013: 19-25.

[4] YOON S H, PARK J S, KIM M S. Behavior signature for fine-grained traffic identification [J]. Applied Mathematics & Information Sciences, 2015, 9(2): 523-534.

[5] 熊刚, 孟姣, 曹自刚, 等. 网络流量分类研究进展与展望 [J]. 集成技术, 2012(1): 32-42.
XIONG Gang, MENG Jiao, CAO Zigang, et al. Research progress and prospects of network traffic classification [J]. Journal of Integration Technology, 2012 (1): 32-42.

[6] BOYER R S, MOORE J S. A fast string searching algorithm [J]. Communications of the ACM, 1977, 20 (10): 762-772.

[7] ABDULLAH A, NAZIR A, SENAPAN M, et al. Fast multi-keyword range search using GPGPU [J]. GPU Computing and Applications, 2015: 259-274.

[8] SOURDIS I, PNEVMATIKATOS D N, VASSILIADIS S. Scalable multigigabit pattern matching for packet inspection [J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008, 16 (2): 156-166.

[9] XU Kefu, QI Deyu, QIAN Zhengping, et al. Fast dynamic pattern matching for deep packet inspection [C] // Proceedings of the 2008 IEEE International Conference on Networking, Sensing and Control. Piscataway, NJ, USA: IEEE, 2008: 802-807.

[10] KIM H, KANG S. A pattern group partitioning for parallel string matching using a pattern grouping metric [J]. IEEE Communications Letters, 2010, 14(9): 878-880.

[11] KIM H, CHOI K I, CHOI S I. A memory-efficient deterministic finite automaton-based bit-split string matching scheme using pattern uniqueness in deep packet inspection [J]. PLoS One, 2015, 10(5): e0126517.

[12] AHO A V, CORASICK M J. Efficient string matching: an aid to bibliographic search [J]. Communications of the ACM, 1975, 18(6): 333-340.

[13] LE H, PRASANNA V K. A memory-efficient and modular approach for large-scale string pattern matching [J]. IEEE Transactions on Computers, 2013, 62 (5): 844-857.

[14] HUA N, SONG H, LAKSHMAN T V. Variable-stride multi-pattern matching for scalable deep packet inspection [C]// Proceedings of the 28th IEEE Conference on Computer Communications. Piscataway, NJ, USA: IEEE, 2009: 415-423.

[15] PAO D, LIN W, LIU B. A memory-efficient pipelined implementation of the Aho-Corasick string-matching algorithm [J]. ACM Transactions on Architecture and Code Optimization, 2010, 7(2): 1-27.

[16] YUN S. An efficient TCAM-based implementation of multipattern matching using covered state encoding [J]. IEEE Transactions on Computers, 2012, 61(2): 213-221.

[17] TAN L, BROTHERTON B, SHERWOOD T. Bit-split string-matching engines for intrusion detection and prevention [J]. ACM Transactions on Architecture and Code Optimization, 2006, 3(1): 3-34.

[18] AHO A V. Compilers: principles techniques and tools [M]. 2nd ed. New York, USA: Pearson Education Inc., 2006: 82-101.

[19] Snort. Snort v2. 8. web-client dataset [DS/OL]. [2020-01-10]. https: // www. snort. org/downloads/ # ruledownloads.

[20] ClamAV. ClamAV dataset [DS/OL]. [2020-01-10]. http: // www. clamav. net/downloads.

[21] Suricata. Suricata dataset [DS/OL]. [2020-01-10]. https: // suricata-ids. org/download/.

[22] PARK B, HONG J W K, WON Y J. Toward fine-grained traffic classification [J]. IEEE Communications Magazine, 2011, 49(7): 104-111.

(编辑 陶晴)