计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(2):207−212 [doi: 10.15888/j.cnki.csa.007785]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于文件分时索引的大规模流量实时 IoT 终端识别算法①

徐彭娜[1], 彭行雄[2]

[1](福州职业技术学院 阿里巴巴大数据学院, 福州 350108)
[2](福建师范大学 网络与数据中心, 福州 350117)
通讯作者: 彭行雄, E-mail: pengxingxiong@163.com

摘　要: 随着 5G 时代的来临, 诸如工业区, 校园网等开放性园区网络中存在大量的物联网 (Internet of Things, IoT) 终端, IoT 终端由于其数据流量巨大, 伪造 IoT 终端进行网络攻击的问题日益严重. 现有 IoT 终端识别技术在面对海量数据时计算资源的成本逐渐提高. 针对以上问题, 提出了基于文件分时索引的大规模流量实时 IoT 终端识别算法. 首先, 建立内存分时索引元数据; 其次, 使用文件的分时索引来存储构建会话的中间数据; 最后, 控制内存分时索引元数据触发从少量文件中提取特征并进行 IoT 终端识别. 实验中, 在不损失 IoT 终端识别算法精度条件下, 仅消耗少量磁盘, 可将内存消耗降低 92%. 实验结果表明, 该技术能够用于实时 IoT 终端识别框架中.

关键词: 物联网 (IoT); IoT 终端识别; 异常检测; 网络流量

引用格式: 徐彭娜,彭行雄.基于文件分时索引的大规模流量实时 IoT 终端识别算法.计算机系统应用,2021,30(2):207–212. http://www.c-s-a.org.cn/1003-3254/7785.html

Real-Time IoT Terminal Identification Algorithm for Large-Scale Flow Based on Time-Sharing Index of Files

XU Peng-Na[1], PENG Xing-Xiong[2]

[1](Alibaba Big Data Institute, Fuzhou Polytechnic, Fuzhou 350108, China)
[2](Network and Data Center, Fujian Normal University, Fuzhou 350117, China)

Abstract: With the advent of the 5G era, there exist a large number of Internet of Things (IoT) terminals in the open campus network such as industrial area and campus network. Due to the huge data flow of IoT terminals, the problem of counterfeiting IoT terminals for network attack becomes increasingly serious, and the cost of computing resources of the existing IoT terminals identification technologies in the face of massive data increases gradually. To solve these problems, we propose a real-time IoT terminals identification algorithm for large-scale flow based on the time-sharing index of files. Firstly, the metadata for the time-sharing index of memory is established. Secondly, the time-sharing index of files is used to store the intermediate data of the construction session. Thirdly, the metadata trigger for the time-sharing index of memory is controlled to extract features from a small number of files and perform IoT terminals identification. In the experiment, on the premise of maintaining the accuracy of the IoT terminals identification algorithm, only a little disk space is occupied and the memory consumption is reduced by 92%. These results show that the proposed algorithm can be used in the framework of real-time IoT terminals identification.

Key words: Internet of Thing (IoT); IoT terminals identification; anomaly detection; network flow

① 基金项目: 福建省高校产学研合作科技计划重大项目 (2016H6007)
Foundation item: Major Project of Higher Education Industry-University-Research Cooperated Science and Technology Plan of Fujian Province (2016H6007)
收稿时间: 2020-06-24; 修改时间: 2020-07-21; 采用时间: 2020-07-27; csa 在线出版时间: 2021-01-27

近年来, 随着智能手机以及摄像头、打印机等 IoT 终端的快速发展, 人们逐渐倾向于连接或使用 IoT 终端进行工作和学习, 个人的身份信息和行为信息都作为物联网服务中重要的数据, 园区网络管理机构对各个 IoT 终端进行管理和认证. 而伴随着 5G 时代的来临, IoT 终端的数据流量呈现出爆炸式增长, 伪装成 IoT 终端访问园区数据以及窃取个人隐私数据使得园区网络安全面临巨大挑战[1]. 为了加强网络安全, 对网络空间中的 IoT 终端的研究是当前的热点[2].

利用异常检测算法的 IoT 终端识别技术保证大数据平台安全性是一种有效的解决方式, 但仍然存在许多困难. 第一, 在服务器端汇聚的 IoT 终端的网络流量主要是以 TCP、UDP 等为主的报文数据, 其海量性以及时效性可能导致现有 IoT 终端识别技术无法在短时间内迅速检测出异常; 第二, 由于每个终端设备在出厂后都有指纹信息以及通信行为数据, 在应用层进行的基于 Web 流量中用户行为的 IoT 终端识别, 主要通过用户行为数据来反映访问习惯, 根据攻击者与正常访问者的行为逻辑不同来进行识别. 但对于摄像头、打印机等哑终端发送报文数据方面的研究不多. IoT 终端中存在大量的哑终端, 主要通过 TCP 或 UDP 报文和服务器进行通信[3], 难以直接从 Web 流量角度来识别[2]. 第三, 5G 时代使得 IoT 终端不得不面临大数据挑战, 需要依托于大数据平台完成 IoT 终端识别算法处理, 给服务器带来性能和成本等多方面压力. 因此, 如何高效快速的利用报文数据在少量内存中识别 IoT 终端具有重要意义.

针对以上问题, 本文提出基于文件分时索引的 IoT 终端识别算法 (a real-time IoT terminal recognition algorithm for huge data based on File Time-Sharing Index, IoT-FTSI) 大幅度降低实时 IoT 终端识别中的内存消耗. 具体方法是: 首先, 利用哈希桶多进程按批次分发数据, 提高数据吞吐量, 并建立内存分时索引元数据; 其次, 使用文件的分时索引来存储构建会话的中间数据; 最后, 控制内存分时索引元数据触发从少量文件中提取特征并进行 IoT终端识别, 而达到在少量磁盘消耗的代价下, 将内存消耗降低 92% 后并保证 IoT 终端识别算法精度.

1 相关工作

随着物联网在智慧城市、智慧园区等各个领域的广泛应用, 物联网的安全问题也层出不穷. 当前, 物联网安全问题主要通过 IoT 终端识别技术[2,4] 来解决. IoT 终端识别的核心是通过指纹生成技术, 将探测到的 IoT 终端数据经过特征提取并转换为相应的指纹, 从而根据分类模型识别 IoT 终端.

一般情况下 IoT 终端的数据是成对出现的, 例如请求和响应包. 数据可来自传输层、网络层和应用层协议, 特别是传输层的 TCP/UDP 的数据报文头部包含了大量可用信息, 例如分段时长、延时、请求字节数等丰富的可用于特征提取的数据[4]. Beverly[5] 使用半链接 TCP 探测及交互时间延迟, 完成全网拓扑结构识别. 但仅支持 IPV4 协议的网段. Liu 等[6] 通过发送四组报文到防火墙后而得到的报文时间差异作为特征, 然后通过统计学习方式识别 IoT 终端类型. 但报文特征有限, 识别率不高. Shamsi 等[7] 提出的 Heshel 方法利用 TCP 数据包发送时间与重传时间的差异性作为特征, 并使用 EM 算法降低网络抖动、丢包等因素的影响. 但是主要数据是设备操作系统信息, 有效的设备行为数据包不多.

随着 5G 技术的兴起, 例如智能终端与服务器存在大量的交互, 势必其 TCP/UDP 等数据流量大幅度增加. Bezawada 等[8] 通过从网络流量中提取设备行为的近似特征, 用于训练设备类型的机器学习模型. Yang 等[9] 利用神经网络对监控设备的海量数据报文进行解析, 提高了 IoT 终端的识别率. 贾煜璇[2] 通过建立一个 IoT 终端识别系统对海量的 TCP 数据进行提取和分类, 具有较高的精确率. 宋金珂[3] 实现了大规模实时在线监控设备的隐私检测, 但每次检测的资源成本较大. 使用大规模流量来描绘行为轮廓是当前的研究热点[2,10], 引入机器学习或者深度学习的方法对网络流量进行分析, 虽然能够提高 IoT 终端识别准确率并提高系统的自动化能力, 但使得承载 IoT 终端识别技术的系统尤其是对于实时性要求高的系统或服务器带来了资源成本的大幅度提升. 尹方鸣等[11] 提出一种基于内存受限的 RFID 复杂事件处理优化算法, 使用文件分时索引方法进行海量数据中的复杂事件处理, 降低了资源消耗成本. 因此, 降低大规模流量下 IoT 终端识别的资源消耗是 IoT 终端识别技术中的关键问题.

基于以上工作, 受到文献 [11] 的启发, 本文建立一个新的 IoT 终端识别算法模型, 该模型并不针对 IoT 终端识别准确率的提升, 而是借助于文件分时索引技

术, 以极小的磁盘消耗代价达到大幅度降低实时 IoT 终端识别中的内存消耗的目标.

2 理论基础

2.1 会话约束条件

生成会话的目标是为 IoT 终端识别提供数据支持, 本文使用数据特征较为丰富的 TCP/UDP 数据[12], 并使用 FlowBroker[13] 抓取报文数据 Packet, 最后利用机器学习算法对报文组成会话后的特征向量进行分析. 在构建流的特征向量前, 对报文生成会话的过程约束如下:

(1) 构建会话序列 S: 根据四元组 k 收集在时间 T_s 内的报文数据组成会话, T_s 指从第一次收集到某 k 的报文的时间开始计算. 使用非自然时间的超时机制, 对各 k 维护 T_s 时间内的缓存并能触发生成会话, 而 k 相同的报文在 T_s 内将构成会话序列 S. 对于哑终端而言, 其报文数据中的 k 会维持长时间不变. 对于非哑终端而言, 其报文数据最终符合用户的习惯, 文献 [14] 表明网络流量的访问概率服从齐普夫分布, 即非哑终端访问服务器时, 其访问次数和访问概率成反比. 因此可以使用较少的报文数量 n_p 来控制报文数量, 以防止在 T_s 内出现了热点问题.

(2) 提取会话流特征向量 SF: 收集 n_s 条 k 相同的会话后, 统计并提取会话特征 SF. 其中 n_s 需要根据不同的场景调整.

2.2 文件分时索引

基于内存的会话生成算法在接收到报文时, 根据 k 存放到集合 KP 中, 如果该 k 是第一次出现, 则记录该 k 的时间作为初始时间; 否则, 则将当前时间和初始时间对比, 超过设定的 Ts 则形成会话, 并从 KP 移出, 否则加入 KP. 由于每个 k 都有一个需要维护的 T_s, 其难点包括以下两个方面: 一方面, 当该时间段内数据量巨大时, 内存占用量极大, 需要设计良好的反压机制. 另一方面, KP 到达 T_s 时需要快速老化, 否则会影响后续进入系统的数据. 该问题属于复杂事件处理范畴, 文献 [11] 表明, 在内存受限的情况下, 使用 B±树分时优化索引 (BIOT) 的复杂事件处理算法将数据流按时序进行分割且用 B±树进行区间分块索引. 受此启发, 本文使用文件的分时索引来存储构建会话的中间数据, 达到减少内存消耗的目标. 图 1 是使用文件的分时索引存储会话中间数据的描述.

相似地, 横坐标为时间线, 表示在 T_i 到 T_{i+1} 时间内的文件索引情况. 竖直方向上, A 表示哈希桶的集合, 共有 b 个 Bucket, 通过对数据包 p 的字符串进行哈希运算分配到 Bucket 中, 其中 b 的值影响算法的并行性能. B 表示 Bucket 中 IP(终端) 的集合, 共有 n 个 IP, 表示通过哈希运算分配到该 Bucket 中的 IP 对应的数据包 p. C 表示 IP 对应的 KP, 共有 m 个 k. D 表示 k 对应的数据包 p 的集合 P, 共有 z 个 p. 在文件分时索引算法的文件存储阶段中, A 和 B 都表示为索引的集合, C 表示文件的集合, D 则是文件中内容的集合. 在内存索引阶段中, 使用 TR 集合来存储以上 A 和 B 构成的二级索引元数据, TR 在内存中用于加速文件的索引过程, TR 在算法启动时从数据库中加载状态.

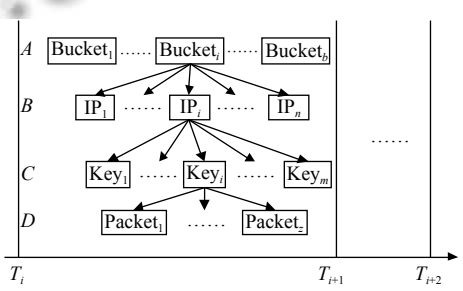

图 1 使用文件的分时索引存储会话中间数据

3 IoT-FTSI 算法框架

3.1 算法描述

ks 表示 Bucket_i 中的四元组集合. IP 表示 ks 集合中某个 k 解析后的源 IP 地址. TR_{IP} 表示 TR 中 IP 的集合, TR_{Key} 表示 TR 中 k 的集合. fl 表示 IP 和 k 索引到的会话文件的最后更新时间和文件长度. $active_{\text{IP}}$ 表示 IP 的激活状态, 当值为 true 时表示处于激活状态, 允许和 IP 相关的新的报文进入, 否则不允许进入. $active_k$ 表示某会话的激活状态, 当值为 true 时表示处于激活状态, 允许与 k 相关的报文进入, 否则不允许进入. lk 表示从消息队列消费的当前批次中 k 对应的报文数量, m 表示 IP 对应的会话数量. δt 表示当前时间和的 ft 差值. n_p 表示每个会话中的报文数量上限, δl 表示 n_p 和当前会话文件中报文数量的差值. n_s 表示每个 IP 允许的会话数量上限. IoT-FTSI 算法实现如算法 1 所示.

算法 1. IoT-FTSI

1) for k in ks:
2) if ip in TR_{IP} && $active_{\text{IP}}$:
3) if key in TR_k && $active_k$:
4) if $\delta t<T_s$ && $fl< n_p$:

```
5)            wf()
6)          else: even(RF_FE)
7)       else:
8)          if m <n_s:
9)             if lk<n_p: even(RF_FE)
10)            else: even(AW)
11)         else: even(RF_FE)
12)   else: even(AW)
```

IoT-FTSI 算法的步骤描述如下: 步骤 1) 遍历 *ks*. 步骤 2) 判断源自 *k* 中的 IP 是否存在于内存的二级索引 *TR* 的第一级索引中, 并且当前 *ip* 处于激活状态, 如果是则进入步骤 3), 否则进入步骤 12). 步骤 3) 判断 *k* 是否存在于内存的二级索引 *TR* 的第二级索引中, 并且当前 Key 处于激活状态, 如果是则进入步骤 4), 否则进入步骤 7). 步骤 4) 判断 δt 是否超过了 T_s, 并且当前会话文件的文件长度 *fl* 是否超过了报文最大数量 n_p, 如果是则进入步骤 5), 否则进入步骤 6). 步骤 5) 中的 *wf*() 函数功能为写入文件, *wf* 函数的算法描述如算法 2 所示.

算法 2. *wf*

```
1) del wf():
2) if δl>=lk: even(MW)
3) else: even(AW)
4) update(ft,fl)
```

算法 *wf* 主要功能是判断待写入的报文是否超过了会话文件允许的最大报文数量上限, 如果超过 (包含等于) 则触发事件 *even*(MV), 即将待写入的报文数据切割出 δl 长度, 再将会话文件中的报文全部提取出来, 进行合并后进行特征提取; 如果未超过, 则触发事件 *even*(AW), 即将所有待写入报文数据全部写入文件中. 这种写入方式主要是减少和磁盘的读写交互. 而且 *wf*() 函数采用文件的顺序读写, 对磁盘的开销也比较小. 最后在以读写文件为主的事件结束后, 需要对会话文件的信息进行更新, 即更新 *ft* 和 *fl*.

IoT-FTSI 算法的步骤 6) 触发事件 *even*(RF_FE), 其算法描述如算法 3 所示.

算法 3. *even*(RF_FE)

```
1) fe(ip)
2) k.clean
3) remove_file(k)
4) m+=1
5) if m>=n_s: active_IP= active_k=False
```

事件 *even*(RF_FE) 的功能前置条件是会话已经达到了 T_s 并且会话文件长度已经达到了 n_p, 那么此时不需要再将报文数据写入会话文件, 并且直接读取 IP 下的 n_s 个会话文件, 使用 *fe*() 函数对二级索引 *TR* 中 IP 索引下的所有 *k* 的会话文件进行特征提取, 并将 $active_{IP}$ 设置为冻结状态. 同时清空内存中的 *k* 以及删除该 IP 的所有会话文件.

3.2 算法分析

在磁盘占用方面, IP 体现为文件夹名, *k* 体现为 csv 文件, 文件中的每行是报文, 并删除了四元组信息, 可将原有处于内存中的 JSON 格式的 600 Byte 的报文对象压缩为 50 Byte 以内的字符串. 实际内存最大使用量=*b*×*n*×报文 CSV 字符串字节数, 默认情况下我们使用 4 进程, 即 *b*=4, 每个 IP 桶最大允许存放报文数量 *n*=20 000 个, 并且默认设置反压阈值 (back pressure threshold, *bpt*) 为 0.8, 即当报文数量达到最大允许报文数的反压阈值时则会进行休眠操作. 默认值=4×20 000×50 Byte≈4 MB. 另外, 与磁盘的交互主要表现为文件追加和删除文件的顺序批量操作. 经过测试, 实验用机械硬盘的每秒进行读写操作的次数 (input/output operations per second, iops) 为 138 MB/s, IoT-FTSI 算法最差情况为将全部数据顺序写入磁盘, iops 为 4 MB/s, 因此对磁盘影响不大.

在时间消耗方面, IoT-FTSI 算法从消息队列每次拉取的报文数量为 *batch*=*b*×*n*×*bpt*, 每个报文数据通过不同的进程分配到不同桶中, 在桶的内部, 报文数据按照 *k* 进行分组. IoT-FTSI 算法遍历 *ks* 中的 *k*, 最后将 *k* 拉取到的所有数据写入磁盘. 最差情况下, 将 n_p 数量的报文从磁盘中读取出来进行特征提取. 因此 IoT-FTSI 算法的时间消耗主要为磁盘的读写耗时.

4 实验分析

4.1 实验数据集

本实验目标在于验证 IoT-FTSI 算法的内存消耗, 数据来源于通过位于高校服务器上 FlowBroker 两次采集到的 IoT 报文数据, 分别为 Da 和 Db 两个数据集, 包含了 337 个终端. 采集时间为工作日的流量高峰时段, 每次采集为 1.5 小时, 数据量分别为 21 305 332、22 700 220 条. Da 数据集采集时间为 9:00~10:30, Db 数据集采集时间为 14:00~15:30. 考虑到 IoT-FTSI 算法可能存在的计算延迟, 需要将 FlowBroker 的数据缓存到

消息队列中, 其结构包含了除了四元组外的包大小、时间延迟等特征数据[2].

4.2 结果分析

根据文献 [10] 中提到使用特征加权聚类算法 (Feature Selection algorithm based on Feature Weighted Clustering, FS-FWC) 作为 IoT 识别算法基础. 实验采用在预测问题中常用的准确率作为评测标准. 将 FS-FWC 算法和使用 IoT-FTSI 算法进行对比, 在相同的准确率下, 比较内存消耗量 (memery used, *mu*) 和数据处理完的时间消耗量 (time used, *tu*). 为了减少实验中因系统环境导致的不一致性以及模拟真实运行环境. 将实验部署在相同 docker 虚拟出的 Ubuntu 系统环境中, 在 7200 r/min 的机械硬盘上限制虚拟环境最大使用 4 核 CPU 和 4GB 内存的硬件资源. 实验结果为在数据集 Da 和 Db 上运行 10 次并取平均值.

首先, 在 Da 和 Db 上使用 FS-FWC 算法进行预测, 为了分析准确率和 n_p 的关系, 观察不断增大 n_p 后准确率的变化情况. 实验结果如图 2 所示.

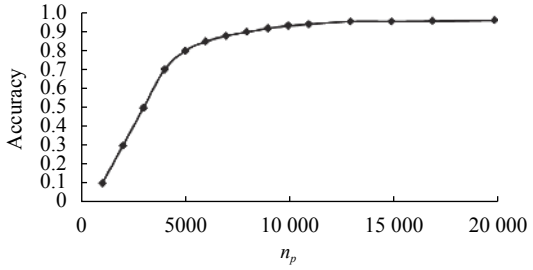

图 2 增大 n_p 后算法准确率变化情况

图 2 中横坐标代表 n_p 值, 最小为 1000, 最大为 20000. 纵坐标代表使用系统 FS-FWC 算法时在 Da 和 Db 的准确率的平均值. 每个报文占用内存大小约为 600 字节. 那么对于 337 个终端而言, 内存使用量理论值已经达到了 3.77 GB, 因此试验中将 n_p 最大值设定为 20000. 从图 2 中可以看出当 n_p 达到 10000 时, 准确率已经达到 0.93, 随着 n_p 的增加, 准确率增加非常缓慢. 当 n_p 达到 18000 时, 准确率达到 0.96, 继续增加 n_p 导致的准确率极小, 因此将 n_p 设定为 18000. 那么内存使用量理论值为 3.38 GB.

其次, 在 docker 虚拟出的 Ubuntu 系统中使用 free 命令观察 FS-FWC 算法和 IoT-FTSI 算法在处理数据集时的内存总体消耗情况与时间的关系, 实验结果如图 3 所示.

在图 3 和图 4 中前 3 分钟为空载时间, 可以看出 IoT-FTSI 使用的内存是超过 FS-FWC 的, 这是因为 IoT-FTSI 算法的元数据消耗了部分内存.

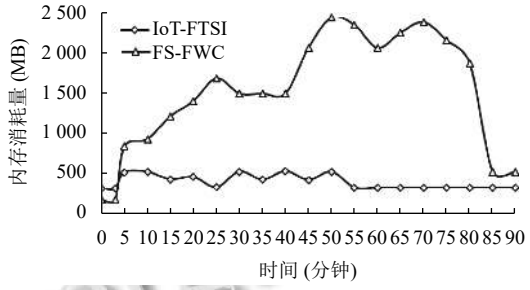

图 3 Da 数据集中算法内存使用量随时间变化情况

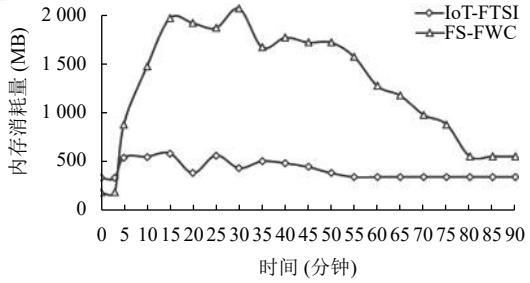

图 4 Db 数据集中算法内存使用量随时间变化情况

从第 3 分钟开始, IoT-FTSI 和 FS-FWC 系统都从消息队列中拉取同样的数据, 两者的 *mu* 都突发性的增长, 但对比发现 FS-FWC 的 *mu* 持续递增. 另外, FS-FWC 的 *mu* 对突发的热点更加敏感, 反映了 IoT-FTSI 更加稳定, 面临更小的内存耗尽风险. 在图 3 中, IoT-FTSI 的 *mu* 在前 55 分钟有明显的波动现象, 因为每个会话在 T_s 间隔就会触发特征提取与预测, 使得 *mu* 突然增加; 而且 *mu* 较小, 原因是其最大理论 *mu* 更小. IoT-FTSI 在 55 分钟后的 *mu* 趋于平缓且接近空载时的 *mu*, 原因是前期已处理的报文被过滤掉. 反观 FS-FWC, 虽然 *mu* 在 25 分钟后出现下降, 但是由于其报文都是存放在内存中, 使得 *mu* 居高不下并且在 55 分钟时达到最高点. 在图 4 中, IoT-FTSI 的 *mu* 在前 25 分钟有明显波动现象, 但是其波动的 *mu* 仍然较小, 虽然在前 25 分钟时数据量突发增长, 但是由于最大报文数量 n_p 的限定和达到反压阈值 *bpt* 时的反压功能使得其具有更好的稳定性.

最后, 通过磁盘读写性能来对比 A 和 B 系统对磁盘的影响. 实验结果如表 1 所示.

表 1 算法在不同数据集上的磁盘性能对比情况

数据集算法	读(kb/s)		写(kb/s)	
	Da	Db	Da	Db
IoT-FTSI	60.42	58.78	49.13	47.31
FS-FWC	0.06	0.04	0.16	0.12

在表 1 中可以明显发现 IoT-FTSI 对磁盘的影响更大, 这主要是其将用于产生会话的临时报文数据按批次写, 同时伴随着到期删除文件、特征提取时全量读取文件内容等读过程. 而 FS-FWC 则对磁盘影响较小, 原因是其产生会话、特征提取等都在内存中完成, 最后只有少量的预测结果写入磁盘. 故 IoT-FTSI 对磁盘的影响更大, 但对于顺序读写性能达到 138 MB/s 的磁盘而言, 其影响程度不大.

5 结论与展望

本文针对实时 IoT 终端识别算法对服务器资源需求高的问题, 结合文件分时索引的方法将原存储于内存中的用于产生会话的大量临时报文数据使用文件分时索引存储在磁盘中, 提出了 IoT-FTSI 算法, 使用少量的磁盘读写代价替换了大量内存的资源耗用, 同时降低了热点问题导致的内存高占用风险. 实验结果表明, 该算法利用文件分时索引方法生成会话, 能够更合理的使用服务器资源, 降低成本, 可以有效地应用于 IoT 终端识别. 然而, 文中提到依赖于哈希算法将报文数据负载到桶中, 哈希算法的性能影响了算法的负载均衡能力. 因此, 如果更好地提高 IoT-FTSI 算法负载均衡性能是下一步要做的工作.

参考文献

1 方滨兴. 定义网络空间安全. 网络与信息安全学报, 2018, 4(1): 1–5. [doi: 10.11959/j.issn.2096-109x.2018002]

2 贾煜璇. 大规模物联网设备组织信息的发现与提取 [硕士学位论文]. 北京: 北京交通大学, 2019.

3 宋金珂. 大规模在线监控设备的隐私及安全检测研究 [硕士学位论文]. 北京: 北京交通大学, 2017.

4 邹宇驰, 刘松, 于楠, 等. 基于搜索的物联网设备识别框架. 信息安全学报, 2018, 3(4): 25–40. [doi: 10.19363/J.cnki.cn10-1380/tn.2018.07.03]

5 Beverly R. Yarrp'ing the Internet: Randomized high-speed active topology discovery. Proceedings of 2016 Internet Measurement Conference. New York, NY, USA. 2016. 413–420. [doi: 10.1145/2987443.2987479]

6 Liu AX, Khakpour AR, Hulst JW, *et al*. Firewall fingerprinting and denial of firewalling attacks. IEEE Transactions on Information Forensics and Security, 2017, 12(7): 1699–1712. [doi: 10.1109/TIFS.2017.2668602]

7 Shamsi Z, Nandwani A, Leonard D, *et al*. Hershel: Single-packet os fingerprinting. ACM SIGMETRICS Performance Evaluation Review, 2014, 42(1): 195–206. [doi: 10.1145/2637364.2591972]

8 Bezawada B, Bachani M, Peterson J, *et al*. Behavioral fingerprinting of IoT devices. Proceedings of 2018 Workshop on Attacks and Solutions in Hardware Security. New York, NY, USA. 2018. 41–50. [doi: 10.1145/3266444.3266452]

9 Yang K, Li Q, Sun LM. Towards automatic fingerprinting of IoT devices in the cyberspace. Computer Networks, 2019, 148: 318–327. [doi: 10.1016/j.comnet.2018.11.013]

10 周丹. 多方法融合的智能终端检测及应用识别 [硕士学位论文]. 重庆: 重庆邮电大学, 2019.

11 尹方鸣, 康慕宁, 陈群, 等. 基于内存受限的 RFID 复杂事件处理优化算法. 计算机应用研究, 2009, 26(8): 2864–2867. [doi: 10.3969/j.issn.1001-3695.2009.08.017]

12 Meidan Y, Bohadana M, Shabtai A, *et al*. ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis. Proceedings of Symposium on Applied Computing. New York, NY, USA. 2017. 506–509. [doi: 10.1145/3019612.3019878]

13 Marconett D, Liu L, Yoo SJB. Optical FlowBroker: Load-balancing in software-defined multi-domain optical networks. Proceedings of Optical Fiber Communication Conference 2014. San Francisco, CA, USA. 2014. W2A. 44. [doi: 10.1364/OFC.2014.M3H.3]

14 彭行雄, 肖如良. 基于稳态过程的多重分形 Web 日志仿真生成算法. 计算机应用, 2017, 37(2): 587–592. [doi: 10.11772/j.issn.1001-9081.2017.02.0587]