**通过加密网络流量分析识别强大的智能手机应用程序**

**摘要：**

安装在智能手机上的应用程序可以显示有关用户的大量信息，例如他们的医疗状况，性取向或宗教信仰。此外，智能手机上是否存在特定应用程序可以通知攻击设备的对手。在本文中，我们展示了被动窃听者可以通过指纹识别他们发送的网络流量来可行地识别智能手机应用程序。虽然SSL / TLS隐藏了数据包的有效负载，但是通道数据（例如数据包大小和方向）仍然会从加密连接中泄漏。我们使用机器学习技术从这种旁道数据中识别智能手机应用程序。除了仅仅指纹识别和识别智能手机应用程序外，我们还会调查应用指纹如何随着时间的推移，跨设备以及不同版本的应用程序的变化。此外，我们引入了一些策略，使我们的应用程序分类系统能够识别和减轻模糊流量的影响，即应用程序中常见的流量，例如广告流量。我们完全实施了一个指纹应用程序框架，并运行了一组完整的实验来评估其性能。我们在Google Play商店中对110种最受欢迎​​的应用进行了指纹识别，并且能够在六个月后识别它们，准确率高达96％。此外，我们还显示应用指纹在不同设备和应用版本中的持续存在。我们完全实施了一个指纹应用程序框架，并运行了一组完整的实验来评估其性能。我们在Google Play商店中对110个最受欢迎的应用进行了指纹识别，并且能够在六个月后识别它们，准确率高达96％。此外，我们还显示应用指纹在不同设备和应用版本中的持续存在。我们完全实施了一个指纹应用程序框架，并运行了一组完整的实验来评估其性能。我们在Google Play商店中对110个最受欢迎的应用进行了指纹识别，并且能够在六个月后识别它们，准确率高达96％。此外，我们还显示应用指纹在不同设备和应用版本中的持续存在。

**发表于：** [IEEE信息取证与安全交易](https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=10206) （ 卷：13 ，[问题：1](https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=8114524)，2018年1月）

**页数：** 63 - 78

**出版日期：** 2017年8月9日

**ISSN信息：**

**INSPEC登录号：** 17364017

**DOI：** [10.1109 / TIFS.2017.2737970](https://doi.org/10.1109/TIFS.2017.2737970)

**出版商：** IEEE

**赞助商：** [IEEE信号处理协会](http://www.signalprocessingsociety.org/) 

**第一节**

介绍

随着设备变得更加经济实惠且功能丰富，智能手机的使 应用程序是这种增长的主要驱动因素，因为它们提供了对增强功能的便捷访问。许多应用程序通常利用Internet访问来提供此额外功能，因此许多应用程序会生成网络流量。应用程序使用量的增加加上应用程序生成的网络流量，使智能手机成为任何想要发现人们的应用程序使用习惯的人的有吸引力的目标。

智能手机用户通常安装和使用符合他们兴趣的应用程序。应用程序提供广泛的功能，包括医疗，金融，娱乐和生活方式服务。因此，安装在典型智能手机上的应用可能会泄露有关用户的敏感信息[1]。这可能包括用户的医疗条件，爱好和性/宗教偏好。对手还可以推断出用户所在的银行，他们通常乘坐哪家航空公司以及哪家公司为他们提供保险。此信息在“鱼叉式网络钓鱼”攻击中可能特别有用。除了发现上述敏感信息之外，攻击者还可以使用应用识别来枚举和利用可能存在漏洞的应用，以尝试获取智能手机的权限。

网络流量指纹识别不是一个新的研究领域，实际上文献例证了传统计算机上的网络流量分类技术 [2]。然而，在智能手机上，应用指纹识别和识别在几个方面受挫。基于端口的指纹识别失败，因为应用程序主要使用HTTP / HTTPS提供数据。典型的网页指纹识别失败，因为应用程序通常使用XML和JSON等文本格式来回发送数据，从而删除有助于网页分类的丰富信息（例如文件数和文件大小）。此外，许多应用程序使用内容传送网络（CDN）和第三方服务，从而消除域名解析或IP地址查找作为可行的策略。由于使用CDN，观察域名解析或TLS握手也证明不太有用。此外，由于使用客户端缓存，或者仅仅由于移动性质，可能根本无法观察到DNS和TLS交换（即，

在本文中，我们将重点介绍智能手机应用程序的指纹识别程度，并通过分析来自它们的加密网络流量进行识别。我们利用这样的事实：虽然SSL / TLS保护网络连接的有效载荷，但它无法隐藏其他粗略信息，例如数据包长度和方向。此外，我们通过测量不同设备，不同应用版本或仅仅时间流逝的影响来评估应用指纹识别框架的稳健性。

应用指纹识别在各种场景中都很有用：

1. 拥有特定应用程序漏洞利用的对手可以使用应用程序指纹识别来识别网络上的这些易受攻击的应用程序，以缩小其目标设备列表。
2. 与受害者在同一Wi-Fi网络上的对手可以偷偷地监视受害者的网络流量，以识别设备上为了勒索而安装的应用程序。
3. 在当前自带设备（BYOD）时代的应用指纹识别可以提供关于组织内这些应用的应用类型和使用模式的有价值数据。
4. 应用指纹识别可以帮助市场研究，因为应用程序的使用可以在目标人群中进行测量。

在本文中，我们扩展了AppScanner，首先由Taylor *等人* 提出*。*[3]，沿着几个重要方面。AppScanner是一个高度可扩展且可扩展的框架，用于通过网络流量对应用程序进行指纹识别和识别。该框架与加密无关，只分析旁道数据，因此无论网络流量是否加密，它都能很好地运行。除原始论文外，我们还做出以下额外贡献：

1. 分析不同设备和应用程序版本的应用程序指纹识别的稳健性。我们还通过测量六个月前生成的指纹尝试识别应用程序时性能受影响的方式来分析应用指纹的时间不变性。
2. 设计和完整实施1的应用程序分类系统，其中包含新颖的机器学习策略，以识别模糊的网络流量，即跨应用程序类似的流量。
3. 证据表明应用指纹是时间，应用版本和设备不变的程度。这有助于支持应用程序分类在实际环境中有用的想法。

本文的其余部分安排如下：第二部分对相关工作进行了调查; 第三部分概述了我们的系统如何在高层工作并解释了关键术语; 第四节概述了我们识别模糊交通的方法; 第五节概述了收集的数据集; 第六节全面评估系统性能; 第七节讨论了使用后处理策略提高分类器准确度的方法; 第八节讨论我们的结果; 最后，第九节总结了论文。

**第二节**

相关工作

在分析来自工作站和Web浏览器的流量方面已经做了大量工作[4]。乍一看，指纹识别智能手机应用程序似乎是对现有工作的简单翻译。虽然存在一些相似之处，但智能手机发送的流量类型以及发送方式的细微差别使得智能手机上的流量分析与传统工作站的流量分析不同[5] - [6] [7] [8 ] ]。我们首先列举了工作站的流量分析方法（第II-A节），然后重点关注智能手机的流量分析（第II-B节），概述了相关工作。

**A.工作站的传统流量分析**

传统的分析方法依赖于HTTP协议的伪像来使指纹识别更容易。例如，在请求网页时，浏览器通常会获取HTML文档以及HTML代码标识的所有相应资源，例如图像，JavaScript和样式表。这简化了对网页进行指纹识别的任务，因为攻击者具有关于附加到单个文档的各种资源的信息语料库（IP地址，文件大小，文件数量）。如果流量未加密，则深度包检测（DPI）可用于流量分类。Bujlow *等。* [9]对利用深度包检测（DPI）的流量分类工具进行独立比较。作者发现，商业工具可以提供最佳性能，尽管开源工具也具有很高的准确性。但是，DPI在加密时失败，在对加密流量进行分类时需要采取不同的方法。

许多应用程序为了可扩展性，在诸如Akamai或Amazon AWS等CDN之上构建其API [10]。这会减少（平均）应用与之通信的端点数量。在过去，查看某些流量的目标IP地址并推断发送流量的应用可能很有用。随着CDN和标准Web服务API的出现，越来越多的应用程序将其流量发送到类似的端点，这使得仅基于目标IP地址指纹应用流量的尝试受挫。

一些作品被认为是强大的对手（例如政府），可以利用流量分析来监控用户活动。这些攻击者能够捕获流经通信链路的网络流量[11]。Liberatore和Levine在[12]中的开创性工作展示了通过加密HTTP流量分析进行网页识别的可行性。随后，Hermann *等人。* [13]通过提出一种依赖于在可观察的IP包大小的归一化频率分布上执行的通用文本挖掘技术的方法，优于Liberatore和Levine [12]的上述工作。此方法正确分类了大约97％的HTTP请求。Panchenko *等。* [14]提出了一种基于支持向量机（SVM）的系统，尽管使用了洋葱路由匿名化（例如，Tor），但该系统正确地识别了网页。最近，蔡*等人。* [15]和Dyer *等人。* [16]提出了网页指纹攻击，并讨论了其有效性，尽管流量分析对策（如HTTPOS）。尤其是Dyer *等人。* [16]旨在避免基于数据包填充的流量分析对策，提出一种依赖于每个方向带宽和流量持续时间的技术。

最近，Panchenko *等人。* [17]提出了一个针对Tor的网站指纹识别攻击，其攻击性能优于相关工作（包括他们自己早先的提议），同时需要较少的计算资源。使用SVM作为分类器，他们提出了一种特征提取技术，该技术从网页的网络流的累积表示开始对特征进行采样。Muehlstein *等人。* [18]表明HTTPS流量可用于识别操作系统，浏览器和应用程序。与我们的工作类似，作者使用加密网络流量概念生成功能。米勒*等人。* [19]使用流量分析来识别网站内的各个网页，准确率约为90％。然而，作者做出两个假设：（i）它们依赖于使用整个网络突发（多个流）的特征提取，假设同一突发中的所有流属于同一网页; （ii）他们还可以依靠多次突发通过隐马尔可夫模型（HMM）构建网站图。这些假设虽然合理用于网页指纹识别，但却无法用于移动应用程序指纹识别。在智能手机上，多个应用可以同时发送网络流量，因此单个突发可能包含不同应用生成的流。在分类器训练期间，可以使用我们后面描述的技术（第III-B节）识别哪个应用程序生成了网络流（即使在并发应用程序场景中）），但在真实世界的攻击中，这种流标签是不可能的。

**B.智能手机的流量分析**

在早期关于这个主题的工作中，戴*等人。* [20]提出了NetworkProfiler，这是一种使用动态方法分析和识别Android应用程序的自动化方法。他们使用用户界面模糊测试（UI模糊测试）自动探索应用程序中的不同活动和功能，同时捕获和记录生成的网络流量。作者在他们的分析中检查HTTP有效载荷，因此这种技术仅适用于未加密的流量。作者没有完全了解他们正在分析的交通痕迹，因此很难系统地量化NetworkProfiler在精确度，召回率和总体准确度方面的准确程度。

Qazi *等人。* [21]介绍Atlas，一个使用网络流识别应用程序的框架。Atlas使用众包来获取基本事实。作者在40个Android应用程序上测试了他们的系统，平均性能达到了94％。然而，仍然不清楚Atlas是否保持良好的性能，因为要分类的应用程序数量增加。Le *等人。* [22]提出了AntMonitor，这个系统也使用众包，但是用于从Android设备中精细收集网络数据。相比之下，AppScanner并不利用众包方法。实际上，AppScanner能够获得完美的基础事实，并使用UI模糊测试以可扩展的方式实现（第III-B节）。

Stöber *等。* [23]提出了一种使用其产生的特征流量模式识别移动设备的方案。作者认为可以逼真地截获和解调3G传输，以获得诸如数据量和定时信息之类的辅助信道信息。作者利用网络突发来提取功能，因为他们无法直接分析TCP有效负载。通过使用监督学习，作者构建了来自设备的后台流量模型。使用他们的系统，使用大约15分钟的捕获流量可以使分类准确率超过90％。这项工作的一个主要缺点是系统需要6小时的培训和15分钟的监控才能实现可靠的指纹匹配。

Mongkolluksamee *等。* [24]，[25]使用数据包大小分布和通信模式来识别移动应用流量。作者的F分数约为95％。不幸的是，他们只考虑了五个应用程序，因此目前尚不清楚他们的方法有多可扩展 作者也未能收集到完美的基础事实，因为他们的方法要求在设备上一次运行一个应用程序以降低噪声而不是更强大的方法（参见第III-B节）。艾伦和考尔 [26]使用TCP / IP标头来识别应用程序。作者使用应用程序启动时生成的前64个数据包中的数据包大小信息，确定应用程序的准确率高达88％。作者发现，当训练和测试设备不同时，性能会下降。他们还发现，在培训和测试之间经过几天后，性能仅略有下降。作为补充，我们使用来自应用的所有网络流量调查问题。通过在六个月而不是几天内收集数据，我们表明流量分类受时间影响更严重，需要采用其他策略来提高性能。

王*等人。* [27]提出了一种从加密的802.11帧中识别智能手机应用程序的系统。他们通过动态运行数据从目标应用程序收集数据，并使用第2层帧的功能训练分类器。作者测试了来自八个不同应用商店类别的13个任意选择的应用，并收集网络跟踪五分钟。通过考虑更多的应用程序，我们发现增加应用程序的数量会对分类器的准确性产生负面影响。王*等人。* [27]也未能收集到完美的基础事实。实际上，我们的方法通过一次运行一个应用程序来最小化噪音，我们仍然必须过滤13％的流量，因为它是来自其他应用程序的后台流量。AppScanner通过使用来自更广泛类别的更多应用程序样本并在更长时间内收集网络流量来解决上述问题。

Conti *等。* [28]和Saltaformaggio *等人。* [29]确定用户在其智能手机应用程序中执行的特定操作。由于相似性，我们简要描述了Conti等人的方法。作者通过流分类和监督机器学习确定具体行动。他们的系统在加密连接的情况下工作，因为作者只利用粗流信息，如数据包方向和大小。AppScanner还利用数据包的方向和大小，但是从原始数据包数据中获得了更强大的统计功能，如第III-B节所示。Conti *等。*大多数考虑的行动都达到了95％以上的准确率。然而，他们的工作在识别离散行为方面具有特殊性。通过在一组有限的应用程序中选择特定的操作，Conti *等人。*可能会受益于产生的更独特的流量。他们的系统也不能很好地扩展，因为在选择和指纹识别操作时采用了手动方法。实际上，作者在这些应用程序中选择了一小组应用程序和一小部分操作来进行分析。

我们以前的工作[3]改善了上述系统的弱点。首先，通过仅利用旁道信息，我们能够在加密的网络流量面前对应用进行分类。此外，我们的系统经过110个应用程序的培训和测试，每个应用程序收集的流量持续30分钟。由于我们框架的性质，应用程序也可以自动进行培训，无需人工干预。然而，我们以前的工作仅限于处理模糊的流量。不明确的流量，即多个应用程序中常见的流量，会对我们以前的系统产生负面影响并导致性能下降。我们以前的工作也没有提供对app指纹的可变性和寿命的理解。在本文中，我们测量不同的设备，应用程序版本或时间的流逝如何影响应用程序指纹识别。

**第三节**

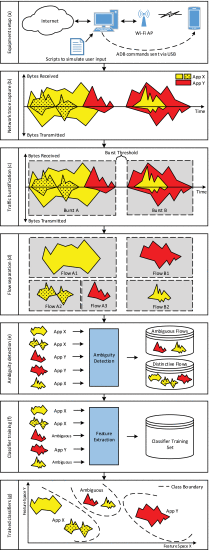
系统总览

作为概述，AppScanner通过使用机器学习来了解智能手机应用，以了解他们生成的网络流量。应用程序生成的流量中的模式（稍后看到）用于识别应用程序。

不幸的是，应用程序有时会有共同的流量模式，因为它们共享库，例如广告库，这些库可以跨不同的应用生成类似的流量2这可能会阻碍使用流量分析进行应用分类的尝试，因为它可能会产生误报。因此，需要一种策略来首先识别在应用程序之间共享的流量，以便在传递给分类器之前对其进行适当标记。我们将应用程序之间的*流量*称为*模糊流量*和剩余流量*特征流量*。

在下文中，我们介绍了*爆发*和*流动*的概念，这些概念对我们的指纹识别方法至关重要：

*突发：*突发是一起出现的所有网络分组（不管源或目的地地址）的组，其满足最近分组在先前分组的时间阈值（*突发阈值*）内发生的条件。换句话说，分组在时间上被分组，并且仅当在设置为突发阈值的时间量内没有新分组到达时才创建新组。这在图1（c）中被视觉描绘为*交通突发*，其中我们可以看到突发A和突发B被*突发阈值*分开。我们使用突发的概念在逻辑上将网络流量划分为离散的，可管理的部分，然后可以进一步处理。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo1abcdefg-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo1abcdefg-2737970-large.gif)

**图。1。**

分类器训练的高级表示，以及网络流量中的突发和流的可视化。（a）设备设置。（b）网络跟踪捕获。（c）交通分流。（d）流动分离。（e）歧义检测。（f）分类器培训。（g）训练有素的分类器。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

*流：*流是具有相同远程IP地址的数据包序列（在突发内）。也就是说，在一个流程中，所有数据包将要去往（或来自）相同的远程IP地址。不要将流与TCP会话混淆。流在突发结束时结束，而TCP会话可以跨越多个突发。因此，流程通常持续几秒钟，而TCP会话可以无限期地继续。AppScanner利用流而不是TCP会话来实现实时/接近实时的分类。从*流动分离*在图1（d）图中，可以看出，脉冲串可能包含一个或多个流。如果单个应用程序*App X*快速连续启动TCP会话或其他应用程序（例如，*应用程序Y）*），碰巧与*App X*同时启动TCP会话。

我们的应用程序识别框架首先从应用程序中引出网络流量，从该流量生成功能，使用这些功能训练分类器，最后在分类器随后呈现未知流量时识别应用程序。

**A.设备设置**

用于收集从应用网络迹线的设置被描述为*设备安装*在图1（a）中。工作站配置为在Wi-Fi接入点（AP）和Internet之间转发流量。为了生成捕获我们的训练/测试数据的流量，我们使用Android调试桥（ADB）通过USB与目标智能手机通信的脚本。这些脚本用于模拟应用程序内的用户操作，从而从应用程序中获取网络流。这种技术称为UI模糊测试。

智能手机生成的流量被捕获并导出为包含捕获数据包详细信息的网络流量转储。我们收集了数据包详细信息，如时间，源地址，目标地址，源端口，目标端口，数据包大小，协议和TCP / IP标志。还收集了每个数据包的有效负载，但未用于提供功能，因为它可能加密也可能不加密。尽管物理硬件用于网络流量生成和捕获，但通过在虚拟机上的Android模拟器中运行应用程序，可以大规模自动化并并行化此过程。

**B.指纹制作**

指纹制作过程分为以下几个阶段：

**1）网络跟踪捕获：**

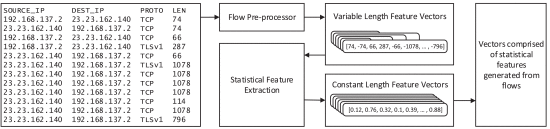
使用UI模糊测试自动引发应用程序的网络流量。UI模糊测试涉及使用工作站上的脚本通过Android调试桥（ADB）与目标设备进行交互。用户界面事件（例如按钮按下，滑动和按键）以自动方式发送到应用程序。这些触摸屏输入事件导致应用程序内的逻辑执行，从而产生网络流量。

We performed UI fuzzing on one app at a time to minimise ‘noise’ (i.e., traffic generated simultaneously by other apps) in the network traces. Traffic from other apps or the Android operating system itself could interfere with and taint the fingerprint making process. To combat the problem of noise, the Network Log tool [30] was used to identify the app responsible for each network flow. Using data from Network Log combined with a ‘demultiplexing’ script, all traffic that did not originate from the target app was removed from the traffic dump for that app. In this way, and in contrast to related work, we obtained perfect ground truth of what flows came from what app.

数据收集后，网络流量转储被过滤，仅包含无错误的TCP流量。例如，我们过滤以删除由于网络错误而导致的数据包重新传输。

**2）交通分流和流动分离：**

下一步是解析网络转储以获取网络流量突发。流量首先被离散化为突发，以获得短暂的网络流量块，可以立即发送到AppScanner的下一阶段进行处理。这使我们能够满足网络流量的实时或近实时分类的设计目标。法拉基*等人。* [31]观察到智能手机上95％的数据包“在前一个数据包的4.5秒内被接收或发送”。在我们的测试中，我们观察到将突发阈值设置为1秒而不是4.5秒仅略微增加了网络跟踪中看到的突发数量。这表明自从最初的研究以来，网络性能（在带宽和延迟方面）已经有所改善。出于这个原因，我们选择使用一秒的突发阈值来支持更多的整体突发和更接近实时的性能。如图2所示）使用远程IP地址将突发分成单独的流（如本节开头所定义并描述）。我们强制执行系统将考虑的最大流量长度。这只是为了确保在现实世界中可以安全地忽略异常流量。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo2-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo2-2737970-large.gif)

**图2。**

从分类器训练的流生成特征。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

值得注意的是，虽然远程IP地址用于流分离，但它们并未用于协助识别应用程序。我们还选择不使用从DNS查询收集的信息或使用未加密的有效负载的流。我们采取此设计决策，以避免依赖于经常更改的特定领域知识，从而使我们的框架在长期内有用。

**3）歧义检测：**

如本节开头所述，许多应用程序都有第三方库（特别是广告库），这些库本身会产生网络流量。不幸的是，不可能以可扩展的方式区分来自库的流量（而不是嵌入库的应用程序），即没有诸如逆向工程或修改应用程序之类的侵入式方法。实际上，就操作系统而言，应用程序及其捆绑的库是同一进程中的一个实体。由于跨应用程序的公共库生成的网络流量相似，这将使指纹识别过程受挫，因为分类器将被给予相互矛盾的训练示例。这个*模糊流动的*问题对天真的机器学习方法提出了挑战。为了减轻负面影响，我们引入*歧义检测*，详见第IV节。模糊检测使用简单的强化学习技术来识别来自不同应用的类似流。在训练阶段，检测到模糊流并将其重新标记为属于“模糊”类，以便系统稍后能够正确地识别和处理它们。

**4）分类训练：**

统计特征由流生成并用于训练分类器。统计特征提取涉及从每个流导出54个统计特征，如图2所示。对于每个流，考虑三个向量：仅传入分组的大小，仅传出分组的大小，以及传入和传出分组的大小。对于每个向量（总共3个），计算以下值：最小值，最大值，平均值，中位数绝对偏差，标准偏差，方差，偏斜，峰度，百分位数（从10％到90％），以及元素数量该系列（总共18个）。这些统计特征是使用Python pandas计算的[32]图书馆。因此，任意长度的流被转换为长度为54的特征向量。这些特征向量及其相应的基础事实被用作训练示例。

**C.应用识别**

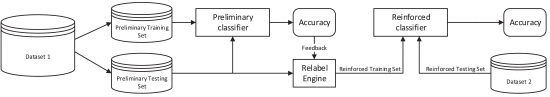
未知流被传递给训练有素的分类器。由于分类器经过训练以理解模糊流，因此识别歧义流并将其标记为这样。未被分类器标记为不明确的流程将按照第VII-B节中的描述进行*分类验证*。分类验证阶段对于一个主要原因至关重要。机器学习算法将始终尝试将未标记的示例放入最接近类似的类中，即使匹配不是很好。鉴于我们的分类器永远不会受到来自应用程序的流量的训练，因此会有一些流向AppScanner的流程，这些流程只是未知或从未见过。如果不加以控制，这可能导致假阳性（FP）率的不期望的增加。

To counteract these problems, we leverage the prediction probability metric (available in many classifiers) to understand how certain the classifier is about each of its classifications. For example, if the classifier labelled an unknown sample as *com.facebook.katana*, we would check its prediction probability value for that classification to determine the classifier’s confidence. If this value is below the classification validation threshold, AppScanner will not make a pronouncement. However, if this value exceeds the threshold, AppScanner would report it as a match for that particular app. In Section VII，我们将讨论如何改变此阈值会影响AppScanner的精确度，召回率和总体准确性，以及这会如何影响分类器足够自信地进行分类的总流量百分比。

**第四节**

模糊检测

歧义检测阶段旨在识别和重新标记模糊流程。这一阶段涉及在分类器培训期间利用的强化学习策略。如图3所示，分类器训练分为两个阶段：*初步分类器*阶段和*强化分类器*阶段。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo3-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo3-2737970-large.gif)

**图3。**

使用强化学习来获得对模糊流的鲁棒性。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

在分析中考虑的主要训练集首先被随机洗牌并分成两半：*初步训练集*和*初步测试集*。初步训练集用于训练初步分类器。初步测试集用于测量初步分类器的准确度，并作为生成增强分类器的训练集的基础。通过这种方式，我们可以首先确定哪些流被初步分类器错误地分类。我们验证了这些错误标记的流程在很大程度上是库流量，正如预期的那样。

在*重新标记引擎*，充分利用了初步分类的准确度反馈，以确定暧昧流动。初步测试集中未正确分类的流程被Relabel引擎重新标记为“模糊”。另一方面，由初步分类器正确分类的流保持其原始标签（即，生成它们的app）。此重新标记的数据集现在用作增强训练集，并传递给强化分类器。因此，增强的分类器被配备用于识别模糊流，因为它是用模糊流的示例训练的。

我们向读者强调，在强化训练集中没有使用初步训练集的流量。初步分类器和初步训练集仅用作识别模糊流的手段，以便可以向增强分类器提供额外的知识。

**第五节**

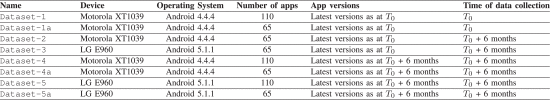
数据集集

为了测试AppScanner的性能，我们考虑了Google Play商店列出的200个最受欢迎的免费应用中的110个。我们选择了最受欢迎的应用程序，因为它们构成了全球应用程序安装基础的很大一部分。此外，我们选择了免费应用，因为免费应用往往受广告支持，因此使用广告库。有一小部分主要广告库，因此广告库往往在应用之间共享。这表明免费应用程序比付费应用程序更容易产生模糊的流量。因此，能够正确地指纹和识别免费应用程序意味着AppScanner足够强大，可以处理付费应用程序。

我们的测试平台中的智能手机通过Linksys E1700 Wi-Fi路由器/ AP连接到互联网，该路由器的AP通过工作站进行了网络连接。使用Android SDK中的MonkeyRunner工具在每个应用上执行UI模糊测试30分钟。UI模糊测试通过调用UI事件（如触摸，滑动和按钮按下）来模拟用户操作。这些UI事件是随机生成的，并发送给应用程序。值得注意的是，一些应用程序在首次启动时会显示登录屏幕。在这种情况下，我们首先在登录之前为这些应用程序手动创建帐户。我们这样做是为了确保使用UI模糊测试的流量生成不会受到登录屏幕的阻碍。理论上可以通过使用由Dynodroid等框架提供的高级UI模糊技术来获得应用程序中所有网络流的更大覆盖率[33]，或通过招募人类参与者。但是，我们认为这些方法超出了我们的研究范围。

这项工作的主要贡献是了解应用程序指纹识别如何受时间，使用的设备，应用程序版本以及这些变量的组合的影响。出于这个原因，我们收集了表I中列出的几个数据集。在下文中，我们将详细描述这些数据集。

**表I所** 使用的每个数据集的设备，操作系统，应用程序数量，应用程序版本和数据收集时间的描述

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t1-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t1-2737970-large.gif)

我们认为是基线的数据集是Dataset-1，它是使用*Device-A*收集的，这是运行Android 4.4.4版本的Motorola XT1039。此数据集包含110个应用程序的网络流量，这些应用程序在初始数据收集时使用每个应用程序的最新版本。我们将此时的初始数据收集称为Ť0 。所有其他主要数据集（数据集-2至数据集-5）在六个月后收集Ť0 ，即在时间 Ť0+ 6 个月。

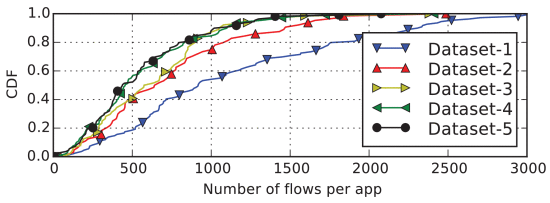
数据集-2与数据集-1的区别仅在于数据收集时。数据集-2仅包含来自65个应用程序（而不是110个）的数据，因为其余45个应用程序拒绝运行而未进行更新。我们在下文中引用数据集-2中的65个应用程序，这些应用程序在未更新为无*运行更新*子集*的情况下运行*。

使用运行Android版本5.1.1的LG E960 *Device-B*收集数据集-3 。Dataset-3还使用了*run-without-update*子集。

数据集-4和数据集-5是通过收集原始110个应用程序的最新版本（在初始数据收集后六个月的数据收集时）的网络流量获得的，并分别使用*Device-A*和*Device-B*收集。

此外，我们考虑具有110个应用程序的数据集的变体。这些变体仅考虑*run-without-update*子集中的应用程序。我们将这些数据集变体分别表示为Dataset-1a，Dataset-4a和Dataset-5a for Dataset-1，Dataset-4和Dataset-5。生成这些数据集是为了在存在少于110个应用程序的数据集（即数据集-2和数据集-3）的情况下提供平衡分析。

图4示出了五个主要数据集中每个数据集中每个应用程序的流数的累积分布函数（CDF）。以数据集-1为例，在UI模糊测试期间，每个应用程序平均收集了1132个流。在同一数据集中，大约80％的应用程序拥有500个或更多流量。其他数据集平均每个应用程序包含的流量更少。尽管如此，请记住，AppScanner可识别个人流量。因此，即使一个流程足以成功识别应用程序，如果该流程对于该应用程序而言是独特的。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo4-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo4-2737970-large.gif)

**图4。**

CDF图显示每个数据集中每个应用程序的流量。

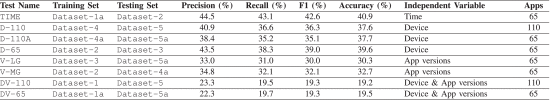
[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

**第六节**

评估

在评估我们的系统时，我们遵循两步程序。首先，我们使用从单个数据集派生的训练和测试集报告系统性能的基线评估结果。其次，为了获得更具代表性的系统性能测量，我们使用完全独立的培训和测试集进行了一整套测试（如表III所示）。进行了测量以了解时间，设备（包括操作系统），应用程序版本以及设备和应用程序版本的组合等因素如何影响性能。

**表III** 用于测量应用程序分类系统性能的综合测试套件摘要。所有训练和测试装置完全相互独立。确定每个测试的独立变量

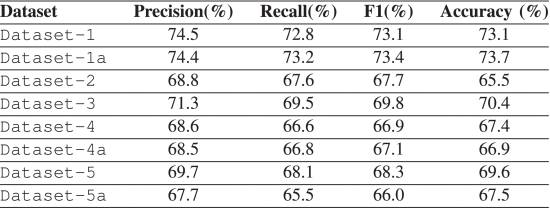
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t3-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t3-2737970-large.gif)

我们利用*scikit-learn*[34]机器学习库在我们的框架中实现分类器。所有分类器都设置为使用默认参数。选择随机森林分类器是因为它们在我们之前的工作中提供了优于支持向量机（SVM）的优越性能。随机森林分类器使用聚合决策树，这反过来又减少了偏差。此外，这些分类器本质上是多类分类器，使其适用于应用程序分类等任务。此外，随机森林分类器本身给出了属于一个类的概率，这是我们在分类验证中使用的一个特征（第VII-B节））。虽然SVM可以处理多类问题并输出属于类的概率，但这些功能对于SVM来说并不是原生的。

我们向读者强调，本节中报告的任何结果都应被视为系统性能的下限。实际上，本节中的结果显示了在应用任何性能增强器之前系统的性能，例如模糊度检测和分类验证（第VII节）。本节中执行的测试仅用于在应用后处理之前评估默认系统性能。

对于我们的基线结果，我们将每个数据集分成训练集（75％的流量）和一个测试集（25％的流量），并用它们训练分类器，详见第III节。每个测试用随机洗牌的数据集运行50次，并将结果取平均值。我们在表II中报告了系统的性能。数据集内的准确率介于65.5％和73.7％之间。这些结果相当不错，但可能高估了系统的性能。这是因为每种情况下的训练和测试集都是从一个原始数据集生成的。

**表II** ：未应用任何后处理技术的每个数据集的应用程序分类的基线性能

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t2-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t2-2737970-large.gif)

在下文中，我们通过使用完全独立的数据集进行训练和测试来进行更强大的测量，以对系统性能进行更真实的评估。每个测试用随机洗牌的数据集运行50次，并将结果取平均值。每种情况的结果总结在表III中。

**A.时间的影响**

为了测量时间对分类性能的影响，我们使用Dataset-1a训练了分类器并使用Dataset-2进行了测试。这种训练和测试集的组合评估了保持设备和应用程序版本不变的效果，但是在训练和测试的数据收集之间导致了六个月的时间。该测试的总体准确度称为TIME测试，为40.9％，是我们使用完全独立的培训和测试集的测试的最高性能。

由于应用程序版本和设备（包括操作系统版本）不变，因此提供最高性能的TIME测试并不令人惊讶。即使经过一段时间（在这种情况下为六个月），生成流量的逻辑（应用程序和操作系统）似乎也会产生相同的流量。由于底层逻辑没有改变，因此期望app指纹也保持不变是合理的。

**B.不同设备的影响**

为了评估不同设备对应用程序分类的影响，我们进行了三项测试：D-110，D-110A和D-65。D-110使用Dataset-4作为训练集，使用Dataset-5作为测试集。也就是说，我们在一台设备上使用110个应用程序进行了培训，并在不同设备上使用相同的110个应用程序进 D-110的总体准确度为37.6％。D-110A使用了*无需运行的运行*D-110中使用的数据集子集，总体准确度为37.7％。使用数据集-2的训练集和数据集-3的测试集进行D-65。也就是说，我们在一台设备上使用65个应用程序进行了培训，并在另一台设备上使用65个应 该测试的总体准确度为39.6％。我们注意到，这个测试有65个应用程序，性能可与TIME测试相媲美，TIME测试也有65个应用程序。这种见解表明，设备模型和操作系统版本对app指纹识别性能没有显着影响。

**C.不同应用版本的影响**

我们进行了两项测试，以了解不同应用程序版本对应用程序指纹识别的影响。V-LG涉及使用Dataset-3进行培训并使用Dataset-5a进行测试。对于此测试，使用了相同的设备，但使用相同应用程序的不同版本。该测试的总体准确度为30.3％。V-MG使用了数据集-2的训练集和数据集-4a的测试集。该测试的总体准确度为32.7％。我们注意到这两种测试的准确度非常相似，但明显低于TIME，D-110或D-110A或D-65测试。这种见解表明应用程序版本的变化会影响应用程序指纹识别的可靠性。我们认为这种现象可能是由于应用程序代码或逻辑的变化导致应用程序生成网络流的方式产生直接影响。

**D.不同设备和不同应用程序版本的影响**

最后两项测试用于衡量更改设备和应用程序版本的影响。第一个测试DV-110使用了数据集-1的训练集和数据集-5的测试集，即使用了总共​​110个应用程序。第二个测试DV-65使用了Dataset-1a训练集和Dataset-5a测试集。这些测试的总体精度分别为19.2％和19.5％。正如我们之前测试的结果所预期的那样，将设备和应用程序版本一起更改会更严重地影响分类性能。然而，值得注意的是，在这些不利条件下，培训和测试集中的应用程序数量似乎并未以负面方式影响整体分类准确性。该结果支持了应用指纹识别可以是可扩展的努力的想法。

**第七节**

提高准确性

到目前为止，我们的结果显示了AppScanner的性能，没有应用任何后处理。此外，这些结果模拟了实验室条件，因为它们来自已过滤噪声的数据集（使用NetworkLog，如第III-B节所述）。在本节中，我们将介绍两种已经证明有效提高系统准确性的后处理策略：模糊检测和分类验证。模糊检测在第IV / VII-A 节中详述，分类验证在第VII-B节中讨论。通常，这两种策略都旨在识别对于app指纹识别不可靠的网络流。

同样在本节中，我们分析了噪声对分类准确性的影响。在我们之前的测试中过滤了噪音。虽然攻击者可以在训练期间轻松过滤噪音，但他们无法在测试过程中过滤噪音。因此，我们使用三个（一个实验室和两个真实世界）实验设置来检查噪声的影响：

1. 该*噪音过滤*设置删除使用NetworkLog训练和测试集的所有噪音。这给出了系统的性能，假设设备没有从非应用程序源（例如操​​作系统）生成任何网络流量。
2. 该*噪声忽略*设置从训练集去除噪声，但在测试组离开的噪音。这给出了在实际攻击场景中可能预期的性能的实际估计。这是因为攻击者可以从训练集中移除噪声，但在攻击期间无法消除噪声。
3. 所述*噪声管理*设置通过实际识别和在训练标记噪声和测试集进一步进了一步。这允许分类器了解来自Android操作系统本身的网络流量。因此，分类器能够更好地识别现实世界中的噪声，这进一步提高了准确性。

除非另有说明，否则使用*噪声管理的*实验设置生成后续结果。

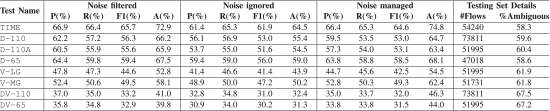
**A.模糊检测**

如第IV部分所述，许多应用程序都有共同的流量，如果处理不当，这会阻碍应用程序分类。我们的强化学习方法识别并重新标记模糊流，以便分类器具有识别它们的模型。当使用模糊度检测来测量性能时，在分类器性能的计算中省略标记为模糊的未知流。也就是说，识别并忽略模糊流程，因此不会影响我们系统性能的测量。

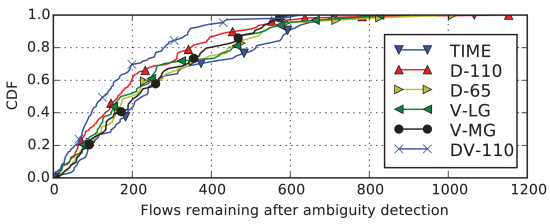
在下文中，我们报告了使用我们的强化学习方法识别模糊交通流量可以带来的改进。表IV显示了通过应用模糊度检测获得的性能改进，如图3所示。此外，该表还显示了每个测试集中的流数以及被认为不明确的流的百分比。每个测试都使用表III中描述的训练和测试集，但具有不同的噪声处理，以便为每个*噪声滤波*，*噪声忽略*和*噪声管理*提供结果实验设置。对于这些测试，使用强化分类器以便可以管理模糊的流量。模糊检测应用于这些增强分类器的训练集，详见第IV节。每个测试用随机改组的数据运行50次，并将结果取平均值。

强化分类器在整体精度方面获得了大约两倍的提升。最具挑战性的测试，DV-110和DV-65（使用不同的物理设备，Android版本以及训练和测试集之间的应用程序版本），在使用强化分类器时性能提升最大，并且准确度提高了一倍以上。例如，在DV-110中，使用模糊检测，准确度从19.2％增加到41.0-46.3％。使用强化分类器提高性能突出了应用流量中模糊流的普遍存在，并重申需要可以解决它们的系统。

**表IV** 强化学习策略如何改进每个测试的分类器性能

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t4-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t4-2737970-large.gif)

模糊度检测以系统分类的流的数量为代价提高了准确度。在最坏的情况下，许多应用程序将所有流程视为模糊，因此如果使用模糊度检测，则系统无法对其进行分类。图5显示了被认为不是模糊流的每个应用程序的流数。为清楚起见，我们仅显示该图中*噪声管理*实验设置的结果。其他设置给出了类似的图。为了更加清晰，我们省略了D-110A和DV-65，因为它们分别是D-110和DV-110的缩小版本。对于除DV-110之外的所有测试，所有应用程序都有一些明确的流程，即在模糊检测后，每个应用程序都有一些流量剩余。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo5-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo5-2737970-large.gif)

**图5。**

CDF图显示了应用模糊检测后每次测试剩余的流量数。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

对于测试DV-110，有一个应用程序没有明确的流程。为了识别该应用程序，必须放弃模糊检测（以系统准确性为代价），或者必须重新指纹应用程序以试图获得可能是明确的附加流程。还有可能AppScanner无法识别某些应用程序，因为它们根本不会生成明确的网络流。这是使用网络流量分析进行应用识别的限制。

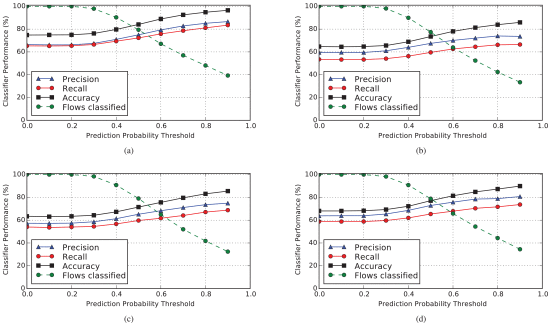
**B.分类验证**

Classification validation is another effective strategy that can be leveraged to improve app classification performance. Classifiers can be made to output their confidence when labelling and unknown example. In simple terms, a classifier may be very confident about a classification if the class boundaries within its models are distinct, i.e., with sufficient separation between classes. In other cases, this distinction may be less clear.

通过评估分类器报告其分类的置信度，可以判断该分类是否将被系统视为有效。我们将被认为是有效分类的截止值称为预测概率阈值（PPT）。较高的PPT将导致更保守的预测，从而导致更高的准确性，代价是被认为有效的分类流的数量。另一方面，较低的PPT降低了准确性，但最大化了被认为是有效的分类的流的数量。对于与准确识别应用程序有关的系统，误报通常是不合需要的，因此较高的PPT可能是合适的。

分类验证减少了被认为是“正确”分类的流的数量，但重要的是要注意，没有固有的要求来标记所有未知流。应用程序在使用时通常每分钟发送数十或数百个流量，因此仍有很大机会从更独特的流程中识别应用程序。因此，分类验证可以是一种有效的技术，用于改善应用程序分类性能，同时产生可忽略的缺点。在下文中，我们报告了通过对我们之前描述的强化分类器应用分类验证所提供的改进。

图6显示了TIME，D-110，D-110A和D-65测试的分类验证所提供的改进。我们通过考虑PPT为0.9来突出一些结果。图6a显示TIME测试具有74.8％的初步准确度，使用分类验证将其提高至96.5％。D-110和D110-A测试的结果分别显示在图6b和6c中。测试D-110的总体准确度从64.7％提高到85.9％，D-110A的总体准确度从63.4％提高到85.5％。在使用分类验证时，D-65组的最终测试结果显示准确度从68.1％上升到89.9％。

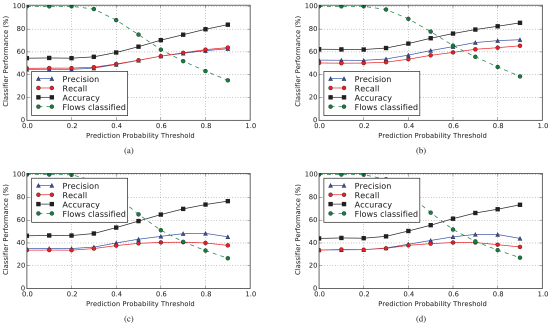
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo6abcd-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo6abcd-2737970-large.gif)

**图6。**

增强型分级机在TIME，D-110，D-110A和D-65测试中的性能。（a）TIME测试的表现。（b）D-110测试的表现。（c）D-110A测试的表现。（d）D-65测试的表现。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

图7显示了V-LG，V-MG，DV-110和DV-65测试的分类验证所提供的改进。考虑到PPT为0.9，我们再次报告我们的结果。图7a显示分类验证将V-LG测试的准确度从54.5％提高到83.9％。图7b显示了V-MG测试的结果，其类似于V-LG但具有不同的设备。在这种情况下，分类验证将准确度从62.4％提高到85.5％。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo7abcd-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo7abcd-2737970-large.gif)

**图7。**

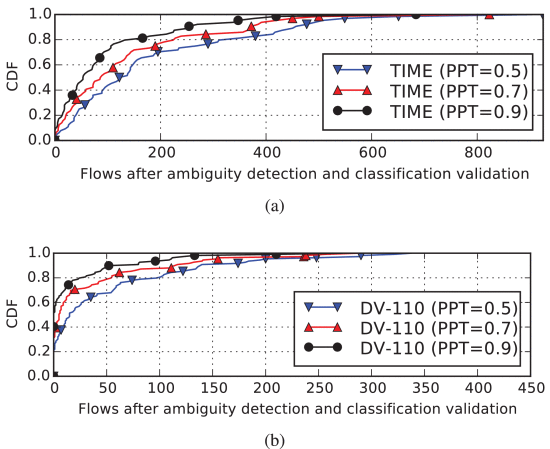
增强型分类器在V-LG，V-MG，DV-110和DV-65测试中的性能。（a）V-LG测试的表现。（b）V-MG测试的性能。（c）DV-110测试的性能。（d）DV-65测试的性能。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

图7c和7d显示了我们最具挑战性的测试结果：DV-110和DV-65。分类验证能够将DV-110的准确度从46.3％提高到76.7％。同样，对于测试DV-65，准确度从44.0％增加到73.5％。这表明分类验证可以成为在困难条件下提高系统性能的有用工具。

**C.参数调整的注意事项**

鉴于分类验证忽略了一些分类器预测，如果分类器不够自信，则将PPT设置得太高可能导致系统不再能够对来自特定应用程序的流进行分类。我们测量分类验证和PPT对系统能够分类的应用程序数量的影响。为简洁起见，我们分别显示了具有最佳和最差基线性能（即TIME和DV-110）的测试结果。此外，我们展示了0.5,0.7和0.9的不同PPT。这些结果总结在图8的CDF图中。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo8ab-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo8ab-2737970-large.gif)

**图8。**

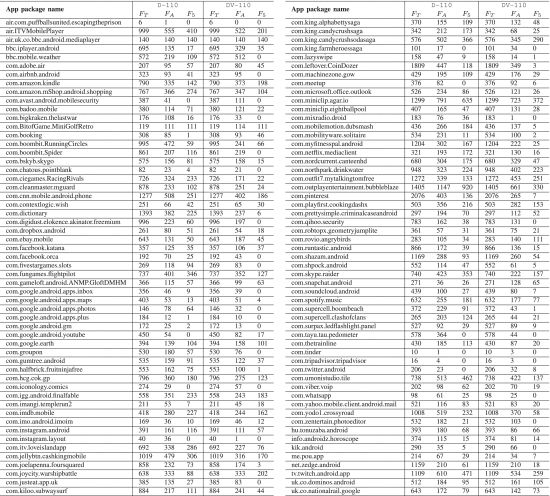
CDF图显示了模糊检测和分类验证后每个应用程序正确分类的流数。（a）时间测试。（b）DV-110测试。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

通常，PPT的较高阈值减少了分类验证后剩余（并且被正确分类）的流量。实际上，将PPT设置得太高会导致系统无法正确识别某些应用程序。在极端情况下，DV-110测试将PPT设置为0.9会导致系统无法对大约一半的应用进行分类。然而，我们提醒读者，DV-110并不代表真实的攻击情形，因为测试使用过时的应用程序签名。该测试更能代表真实场景，TIME，无法识别PPT为0.9的6个应用，而PPT为0.5时仅识别2个应用。因此，分类验证虽然有用，但必须在了解系统如何针对特定情况执行后才进行调整*感兴趣的应用程序*。

作为进一步的细分，附录中的表V概述了在D-110和DV-110测试的每个分类阶段剩余（并且正确分类）的流量。我们展示的是D-110（而不是我们迄今为止所做的那样），因为它拥有像DV-110这样的所有110个应用程序，并且提供与TIME类似的性能。从表中可以看出，某些应用程序（如air.com.puffballsunited.escapingtheprison）对我们的系统分类提出了挑战，而其他应用程序如air.uk.co.bbc.android.mediaplayer则可以轻松准确地识别。

**表V** D-110和DV-110测试的应用分类期间各点的流量细节。 FŤ= 每个应用程序的测试集中的流量，F一个= 模糊检测后的流量， F五= 在PPT为0.5时正确分类的流数。为了简洁，我们显示PPT为0.5，因为它处于PPT范围的中间点

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t5-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo.t5-2737970-large.gif)

在系统能够分类的应用程序数量与系统可以识别应用程序的整体准确性之间进行权衡。应根据AppScanner的使用场景调整实验设置。可以以较低的准确度检测更多应用，或者可以以更高的准确度检测更少的应用。攻击者还可以使用她*感兴趣的应用程序*在AppScanner系统中的行为方式来了解她的系统。

**第八节**

讨论

Smartphone app fingerprinting is challenging because of a variety of variables that are likely to change between fingerprint building and final deployment. Such variables include device, operating system version, app version, and time. Any mismatch between variables during app fingerprinting and app identification has the potential to reduce the performance of our app classification system. To this end, we assessed how the aforementioned variables affected system performance. Apps were fingerprinted and later re-identified under a thorough suite of experimental settings.

在表II中，我们报告了从同一数据集生成培训和测试集时的应用程序分类性能。在其他测试中，我们使用完全独立的数据集进行培训和测试。使用独立组时的系统性能显着低于基线实验。如果想要更准确地估计应用指纹识别系统的性能，这突出了对完全独立的训练和测试集的需求。

在收集培训和测试数据之间的六个月内，使用特定应用程序版本和设备进行培训具有最高的基线准确性。这表明时间（在六个月时间刻度上）引入了app指纹中的最小方差。这种见解表明，尽管应用程序服务器返回的内容可能已经发生变化，但我们的模型对这些变化具有相当的弹性，并且仍能提供良好的性能。我们对使用不同设备（和操作系统版本）收集的数据集的分析使得性能略低于先前的测试。这表明不同设备的设备或操作系统特性可能会在一定程度上引入一些影响分类性能的额外噪声。预计会出现这种性能下降，因为已知应用程序会根据运行它们的Android操作系统版本来更改其行为。另外，操作系统本身的差异也可能产生影响分类器性能的额外噪声。

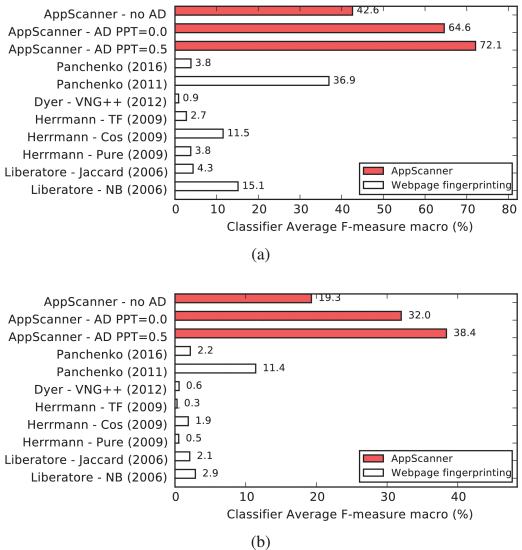
对一组应用程序进行指纹识别并识别相同应用程序的新版本会进一步降低性能。这种现象并不出乎意料，因为应用程序在更新期间会定期收到逻辑更改[35]，这可能会导致其网络流量的变化。但是，我们的分类系统表明它能够在一定程度上应对这些变化。然而，这促使需要在应用程序更新时重新指纹，但建议旧指纹可能有用，尽管可能不如应用程序接收更多更新。更改设备和应用程序版本（以及时间）为我们的分类系统提供了最大的性能损失。这是预期的惩罚，因为时间，设备，操作系统版本和应用程序版本在培训和测试之间都发生了变化。即使在这些最严重的约束条件下，我们的分类器也能够实现20倍于纯随机猜测的基线性能。

大多数性能损失似乎来自所谓的模糊流程。这些流量是跨应用程序类似的流量，通常来自应用程序中常见的第三方库。这种模糊的流量阻碍了天真的机器学习方法，因为分类器被有效地给予具有不同标签的相同训练示例。使用具有强化学习的新颖的两阶段分类策略，我们能够使我们的分类器的基线性能大约翻倍。使用分类验证的附加后处理技术，可以从系统中提取进一步的准确性，但是以分类器能够给出足够自信的预测的流的数量为代价。

**A.与相关工作的比较**

AppScanner适用于智能手机的指纹识别网络流量领域，因此无法直接与网站指纹进行比较。但是，为了激发为什么在指纹应用流量时需要采用AppScanner等不同方法，我们会展示网站指纹识别的现有工作在应用流量上运行时的准确性降低了。为了进行全面比较，我们使用了性能最佳和性能最差的数据集：TIME和DV-110。

通常，所有方法在TIME测试（图9a）中比在DV-110测试中表现得更好（图9b）。Panchenko *等。* [14]表现最佳，F指数分别为36.9％和11.4％。没有歧义检测的AppScanner的表现分别优于42.6％和19.2％。添加歧义检测和分类验证进一步提高了AppScanner相关工作的性能。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo9ab-2737970-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/10206/8114524/8006282/taylo9ab-2737970-large.gif)

**图9。**

与TIME和DV-110测试的相关工作进行比较。AD =模糊度检测和PPT =预测概率阈值。（a）TIME测试的表现。（b）DV-110测试的性能。

[查看全部](https://ieeexplore.ieee.org/document/8006282/all-figures)

**B.限制**

应用程序识别受到许多因素的影响，例如“流量覆盖”，更改应用程序行为和模糊流程。流覆盖率是指在UI模糊测试期间实际触发的数量流量与应用程序可以生成的流量总数的比例。实际上，UI模糊测试可能无法从应用程序中获取所有流。获得完整的代码覆盖是一项具有挑战性的任务，甚至人类参与者也只能通过手动交互在应用程序中获得60％的代码覆盖率[33]。

如果应用程序在不同时间运行，它们也可能具有不同的行为。这可能是因为应用程序本身已经更新，现在在运行时从服务器更改逻辑或应用程序下载（动态）配置参数。这些可能性中的任何一种都可能导致应用程序在培训和测试之间有不同的行为 为了缓解这种情况，需要对应用程序进行重复和连续的分析。幸运的是，可以使用虚拟设备和UI模糊来自动进行性能分析，从而无需物理硬件或手动干预。

模糊的流量也会给应用识别带来问题。它会降低分类器的性能，因为分类器可能会受到冲突数据的训练。此外，由于应用程序可以生成有限数量的流，因此可以想象多个应用程序将生成类似的流。因此，模糊检测方案对于识别来自应用程序的非模糊，独特的流程是至关重要的。也可能是某些应用程序根本不生成非模糊流程的情况。在这种情况下，需要采取其他方法来识别这些应用程序，因为这是使用网络流量分析进行应用程序分类的基本限制。

**C.对策**

通过流量分析减轻应用识别是一项复杂的任务。AppScanner使用来自网络流量的粗略侧通道数据进行特征生成。因此，可行的对策可能涉及充分填充网络流以使得一个app不再可以与另一个app区分开。从理论上讲，这种方法对于抑制应用程序指纹识别非常有用。然而，在实践中，正如Dyer *等人* 所论述的那样*。*[16]，带宽有效的通用缓解策略不太可能提供必要的保护。此外，在电池电量，带宽和数据使用是瓶颈的智能手机（和其他移动设备）上，这个问题很复杂。实际上，这些增加的智能手机瓶颈使得高效的流量分析对策成为一个开放的研究问题。

**第九节**

结论

在本文中，我们扩展了AppScanner，这是一个强大且可扩展的框架，用于从网络流量中识别智能手机应用程序。我们从几个方面全面评估了指纹识别智能手机应用程序的可行性。我们使用不同的设备（和Android操作系统）和不同的应用程序版本，在不同的时间（相隔六个月）收集了应用程序生成的流量的几个数据集。我们证明了时间的流逝是影响app指纹识别的最小变量。我们还发现应用程序指纹不会受安装该应用程序的设备的影响。我们的结果显示，对应用的更新会降低指纹的准确性。这并不奇怪，因为新的应用程序版本可能会有其他功能，这可能会影响指纹识别过程。我们展示了即使在特定设备上生成应用指纹，也可以在运行不同版本的相同应用的不同设备上六六个月后识别它们，其基准精度比随机猜测好20倍。使用模糊检测和分类验证技术，我们获得了系统性能的显着提高。我们能够指纹并在以后重新识别应用程序，在最佳情况下准确率高达96％，在最坏的情况下高达73％的准确率。这些结果表明，应用指纹识别和识别在现实世界中确实是可行的。应用程序指纹识别解决了与用户安全性和隐私相关的各种新挑战。通过在这一领域继续研究，我们的目标是更好地了解这些挑战，