**AppScanner：从加密网络流量自动指纹识别智能手机应用程序**

**摘要：**

智能手机应用程序的自动指纹识别和识别正成为对手，网络管理员，调查员和营销机构非常有吸引力的数据收集技术。实际上，设备上安装的应用程序列表可用于识别攻击者利用的易受攻击的应用程序，发现受害者对敏感应用程序的使用，协助网络规划以及辅助营销。但是，app指纹很复杂，因为可以下载大量应用程序，可以安装各种各样的设备，以及使用HTTPS / TLS等有效负载加密协议。在本文中，我们提出了一种新的方法和实现它的框架，称为AppScanner，用于从加密的网络流量中自动指纹识别和Android应用程序的实时识别。要构建应用指纹，我们在物理设备上自动运行应用程序以收集其网络跟踪。在提取用于训练我们的监督学习算法的特征之前，我们对这些网络轨迹应用各种处理策略。我们的指纹生成方法具有高度可扩展性，不依赖于检查数据包有效负载，因此即使采用HTTPS / TLS，我们的框架也能正常工作。我们构建并部署了这个轻量级框架，并运行了一组完整的实验来评估其性能。我们会自动分析Google Play商店中110个最受欢迎的应用，之后能够以超过99％的准确率重新识别这些应用。在提取用于训练我们的监督学习算法的特征之前，我们对这些网络轨迹应用各种处理策略。我们的指纹生成方法具有高度可扩展性，不依赖于检查数据包有效负载，因此即使采用HTTPS / TLS，我们的框架也能正常工作。我们构建并部署了这个轻量级框架，并运行了一组完整的实验来评估其性能。我们会自动分析Google Play商店中110个最受欢迎的应用，之后能够以超过99％的准确率重新识别这些应用。在提取用于训练我们的监督学习算法的特征之前，我们对这些网络轨迹应用各种处理策略。我们的指纹生成方法具有高度可扩展性，不依赖于检查数据包有效负载，因此即使采用HTTPS / TLS，我们的框架也能正常工作。我们构建并部署了这个轻量级框架，并运行了一组完整的实验来评估其性能。我们会自动分析Google Play商店中110个最受欢迎的应用，之后能够以超过99％的准确率重新识别这些应用。我们构建并部署了这个轻量级框架，并运行了一组完整的实验来评估其性能。我们会自动分析Google Play商店中110个最受欢迎的应用，之后能够以超过99％的准确率重新识别这些应用。我们构建并部署了这个轻量级框架，并运行了一组完整的实验来评估其性能。我们会自动分析Google Play商店中110个最受欢迎的应用，之后能够以超过99％的准确率重新识别这些应用。

**发表于：** [2016 IEEE欧洲安全与隐私研讨会（EuroS＆P）](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7467055)

**会议日期：** 2016年3月21日至24日

**IEEE *Xplore*日期：** 2016年5月12日

**ISBN信息：**

**INSPEC登录号：** 15987975

**DOI：** [10.1109 / EuroSP.2016.40](https://doi.org/10.1109/EuroSP.2016.40)

**出版商：** IEEE

**会议地点：** 德国萨尔布吕肯

**第1节**

介绍

随着设备功能越来越强大，功能越来越丰富且价格越来越便宜，智能手机和移动设备的使用率也在不断提高。Gartner报告称，仅2014年一年，智能手机对消费者的销量就超过10亿台，比2013年增长28.4％[1]。此外，他们还报告说，全球销售的手机中有三分之二是智能手机。移动分析公司Flurry报告称2014年整体应用使用率增长了76％[2]。智能手机开箱即用，但用户经常下载并安装称为应用程序的附加应用程序，以引入其他功能。对智能手机的强烈需求以及应用程序使用量的快速增长使得移动平台成为任何个人或组织的主要目标，这些个人或组织希望识别用户智能手机上特定应用程序的存在，无论是出于仁慈还是恶意的原因。

在个人计算机上，许多技术已用于识别网络流量的类型，以及生成此流量的应用程序。Nguyen和Armitage [3]调查了用于互联网流量分类的机器学习技术。传统上，*TCP / IP*可以通过端口号来识别流量，因为应用程序通常使用为每种类型的服务保留的“众所周知的”目的地端口号。在从使用相同端口号的服务（例如Web浏览）中识别多个流量源的情况下，有时可以依靠HTTP主机头或目标IP地址来识别通信的接收者。但是，在移动领域，由于许多应用程序通过使用*HTTP / HTTPS*发送和接收数据与其服务器进行通信，因此流量指纹识别变得复杂。在开发人员选择使用HTTPS的情况下，有效负载是加密的，因此无法检查以帮助识别流量源自的应用程序。此外，出于可扩展性原因，许多开发人员和广告网络使用内容分发网络（CDN）来提供内容并为其应用提供API。CDN和API的使用意味着多个应用程序可以向相同的IP地址或IP地址范围发送（和接收）数据，因此令人沮丧的依赖于IP地址的应用程序识别尝试。

用户通常根据他们的兴趣安装应用程序。因此，仅知道用户在其设备上安装了哪些应用程序可以提供有关用户的有价值的分析信息[4]。此分析信息对于意图侵入该个人隐私的广告商，政府或流氓个人非常有用。另一方面，用户在其设备上安装的应用列表对于与网络规划，安全性或流量工程有关的网络管理员可能非常有用。我们认为一个演员能够被动地监视网络流量或者能够获得网络踪迹。我们通过概述应用指纹识别和识别对这样一个演员有用的四个具体场景来激励我们的工作。

**针对特定应用的攻击者**。对特定应用程序拥有漏洞攻击（可能是零日攻击）的对手可以使用应用程序指纹识别来识别网络上的易受攻击的应用程序。攻击者可以“离线”构建易受攻击的应用程序（或应用程序的易受攻击版本）的指纹，然后使用它来识别这些应用程序。一旦识别出易受攻击的应用程序，攻击者就可以利用这些漏洞来获取自己的利益。特别令人担忧的是，在用户的设备上考虑对手指纹识别和扫描易受攻击的移动银行应用程序。通过执行应用程序指纹识别，攻击者可以提高攻击目标受害者的准确性，并且在攻击时变得更加谨慎，不需要将攻击“广播”给不易受攻击的用户。

**针对特定用户的攻击者**。App指纹识别也可用于存在特定目标的情况。通过加入受害者的网络（或仅仅保持在无线范围内而不与网络相关联），攻击者可以偷偷地监视并指纹受害者的流量，以识别受害者正在使用或已经安装在他/她的设备上的应用程序。对于备受瞩目的客户来说，这可能是非常不受欢迎的，因为仅知道受害者在智能手机上使用哪些应用程序可能非常重 例如，竞争对手可能会认为一般公众知道已婚政客在他/她的设备上使用约会/调情应用程序会很有趣。当人们考虑高级人员成为目标的高级持续威胁（APT）背景时，这个问题的严重性就会突显出来。一旦确定了应用列表，攻击者就可以继续获取相关漏洞以试图控制受害者的设备或数据。在这种情况下，app指纹识别用于通过快速轻松地枚举受害者使用的服务来降低利用受害者的潜在成本（就时间和金钱而言）。据推测，对手将使用最具成本效益的途径来攻击受害者。

**网络管理**。应用程序指纹识别提供有关组织内这些应用程序的应用程序类型和使用模式的有价值数据。在当前的自带设备（BYOD）时代，这些信息对于希望优化其网络的网络管理员来说是非常宝贵的。例如，了解最受欢迎的应用程序及其吞吐量和延迟要求以获得良好的用户体验，然后管理员可以配置其网络，以便更有效地执行特定应用程序。此外，app指纹识别可用于确定是否在企业网络上使用不允许的应用程序。然后，管理员可以针对违法者采取适当的行动。

**广告和市场研究**。应用指纹识别可以为市场研究提供有价值的帮助。假设分析公司想知道应用在特定位置或特定事件期间（例如在足球比赛期间）的流行度。该公司可能会指纹应用程序，然后进入他们感兴趣的位置，以从一群用户中识别应用程序使用情况。通过在目标人群中指纹识别应用程序使用，广告商可以更好地构建其目标市场的简档，并因此更有效地将广告定向到用户。

**1.1。捐款**

在本文中，我们介绍了AppScanner，这是一个实现强大且可扩展的方法的框架，用于从其网络流量中自动指纹识别和Android应用程序的实时识别，无论此流量是加密还是未加密。我们已经使用Android设备和应用构建并测试了App-Scanner。但是，由于其模块化设计，AppScanner可以轻松移植到指纹和识别其他平台上的应用程序，如iOS / Windows / Blackberry。我们的主要贡献如下：

* 枚举网络流量预处理策略，可以准确提取可以可靠地用于重新识别应用程序的功能。
* 展望使用新颖的多路分解策略获得应用程序负责每个网络传输的完美基础事实的方法。
* 提供高度可扩展的监督学习框架，可用于准确建模并随后识别来自应用的流量。
* 概述了利用实时数据包捕获对拦截的Wi-Fi流量进行实时分类的方法。
* 比较用于从加密网络流量识别智能手机应用程序的各种分类策略的性能。

本文的其余部分安排如下：第2节讨论与流量分析和指纹相关的工作; 第3节概述了AppScanner的设计以及不同组件如何协同工作来指纹应用程序; 第4节讨论了测试的分类策略; 第5节评估AppScanner的性能; 第6节讨论了AppScanner的局限性，并重点介绍了未来工作的各个方面; 最后，第7节总结了论文。

**第2节**

相关工作

流量分析和指纹识别决不是一个新的研究领域，事实上，在分析来自工作站和Web浏览器的流量方面已做了大量工作[5]。乍一看，似乎智能手机应用程序的流量分析和指纹识别是现有工作的简单翻译。虽然存在一些相似之处，例如使用IP地址/端口的端到端通信，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领域的流量分析与传统工作站的流量分析[6] [7] [8]。在本节的其余部分，我们将考虑工作站上的传统流量分析方法（第2.1节）），然后我们看看智能手机上的流量分析（第2.2节）。

**2.1。传统的工作站流量分析**

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）。

许多应用程序开发人员为了可扩展性，在Akamai或Amazon AWS等内容交付网络（CDN）之上构建应用程序API [9]。这会减少（平均）应用与之通信的端点空间。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。目前，对*graph.facebook.com的*请求可能来自Facebook应用程序，但它们也可能来自查询Facebook Graph API的各种应用程序。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。

在文献中，一些作品被认为是可以利用流量分析的强大对手（例如政府）。这些攻击者能够捕获流经通信链路的网络流量[10]。Liberatore等。[11]显示了旨在通过加密HTTP流量分析识别网页的提案的有效性。随后，赫尔曼等人。[12]优于Liberatore等人。通过提出一种依赖于常见文本挖掘技术的方法来实现可观察IP分组大小的归一化频率分布。此方法正确分类了大约97％的HTTP请求。Panchenko等人提出了类似的工作。[13]。他们的提议正确地识别了网页，尽管使用了洋葱路由技术，如Tor。最近，蔡等人。[14]提出了一个网页指纹攻击，并显示其有效性，尽管流量分析对策（如HTTPOS）。不幸的是，这些工作不是为智能手机流量分析而设计的。实际上，作者专注于识别传统PC上的网页，并利用HTTP流量可以非常独特的事实，具体取决于网页的设计方式。在智能手机上，虽然应用程序使用HTTP进行通信，但它们通常通过基于文本的API进行通信，从而消除了典型HTTP网页流量中存在的丰富流量功能。

**2.2。智能手机的流量分析**

许多作者提出了不同的方案，用于从智能手机流量中识别智能手机应用和智能手机。这些方案依赖于检查IP地址和数据包有效负载等。我们在本文中提出的方法和框架仅使用IP地址进行流分离（即，不用于特征生成，如第3节中所述），并且不利用数据包有效负载中包含的任何信息。

戴等人。[15]提出NetworkProfiler，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（UI）模糊测试技术在应用程序中自动尝试不同的执行路径，同时监控网络跟踪。他们检查HTTP有效负载，因此这种技术的缺点在于它只适用于未加密的流量。戴等人。他们没有完全掌握他们正在分析的交通痕迹，因此很难系统地量化NetworkProfiler在精度，召回率和总体准确度方面的准确程度。

可能是最直接相关的工作，Wang等人。[16]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框并使用此数据的功能训练分类器，从目标应用程序收集数据框。这项工作显示出前景，但是受到以下事实的影响：作者只测试了来自8个不同应用商店类别的13个任意选择的应用，并且仅收集网络跟踪仅5分钟。实际上，作者发现，在使用他们的系统对某些应用进行分类时，较长的训练时间会对准确性产生不利影响。此外，作者使用不足的样本量（即仅13个应用程序）来验证其结果。通过考虑大量应用程序，在第5节（具体而言图5），我们展示了如何增加应用程序数量对分类器准确性产生负面影响。此外，尚不知道是否有人等。选择了特定的应用程序集，因为它提供了良好的分类性能，或者统计上合适的集合大小是否会产生类似的良好性能。作者也没有提供精确/召回测量，因此很难判断他们的系统性能。最后，一般来说，量化它们的结果是有问题的，因为作者无法收集准确的基础事实，即没有来自其他应用程序的噪音的标记数据集。实际上，我们的方法要求一次运行一个应用程序以减少噪音，我们仍然必须过滤掉13％的原始数据集，因为它是噪音。AppScanner通过进一步系统地研究这个重要主题来解决上述问题。我们使用来自26个不同类别的110个随机选择的应用程序（来自Google Play商店中最受欢迎的应用程序），每个应用程序收集网络跟踪75分钟。我们使用新的解复用技术预处理这些网络迹线，以获得完美的基础事实。我们研究了两种分类算法，两种特征生成方法和三种整体分类策略。最后，我们确定并验证流量错误分类的原因并提出缓解策略。我们使用新的解复用技术预处理这些网络迹线，以获得完美的基础事实。我们研究了两种分类算法，两种特征生成方法和三种整体分类策略。最后，我们确定并验证流量错误分类的原因并提出缓解策略。我们使用新的解复用技术预处理这些网络迹线，以获得完美的基础事实。我们研究了两种分类算法，两种特征生成方法和三种整体分类策略。最后，我们确定并验证流量错误分类的原因并提出缓解策略。

Conti等。[17]识别用户在其智能手机应用程序中执行的特定操作。他们通过流程分类和监督机器学习来实现这一目标。与AppScanner一样，他们的系统也可以在加密连接的情况下工作，因为它们只利用粗流信息，如数据包方向和大小。作者对大多数考虑的行为的准确率达到了95％以上。这项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，作者必须选择应用程序的子集和这些应用程序中的一部分操作来训练他们的分类器。

Stöber等。[18]提出了一种通过识别特定于设备的流量模式来指纹识别整个设备的方案。他们认为，70％的智能手机流量属于设备上发生的后台活动，并且可以利用它来创建指纹。作者认为可以逼真地拦截和解调3G传输，以从诸如传输数据量和定时之类的传输中获得侧信道信息。他们利用“突发”数据生成功能，因为他们无法直接分析TCP有效负载。使用监督学习算法，作者构建了他们想要指纹的流量模型。然后，该模型能够在以后识别类似的数据突发。作者得出结论，使用大约15分钟的捕获流量可以使分类准确率超过90％。这项工作与AppScanner类似，因为它们都利用突发流量来生成指纹。但是，AppScanner与众不同，因为我们利用突发一次识别单个特定应用，并且无法利用利用多个交错流量突发时获得的丰富信息来获得更独特的指纹。此外，Stöber等人。并且无法利用利用多个交错流量突发时获得的丰富信息来获得更独特的指纹。此外，Stöber等人。并且无法利用利用多个交错流量突发时获得的丰富信息来获得更独特的指纹。此外，Stöber等人。[18]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，而AppScanner每个应用程序使用75分钟的捕获流量进行培训（可以在攻击者自己的设备上完成），然后可以对真实的未知流量进行分类-时间。

**第3节**

系统设计

支持AppScanner的主要思想是关注来自可用于识别该应用的应用的流量。来自应用的流量可以是交互式的或非交互式的; 也就是说，它们可以在有或没有用户交互的情况下生成。如果新闻阅读器应用程序在后台轮询服务器以获取最新新闻，则它可能会生成非交互式流量。交互式流量流由用户操作生成，例如启动应用程序或导航应用程序的用户界面。对于我们的指纹识别方法，我们主要关注交互式应用流量。我们的主要设计目标是：

* 开发可高度扩展的框架，可用于指纹识别智能手机应用程序。
* 确保可以以自动方式构建新应用程序或更新应用程序的模型，并将其添加到系统中。
* 确保模型是可移植的，即，它们可以在新的部署中构建和重用，而不会受到再培训的惩罚。
* 提供一个系统，可以在网络上观察流量的实时（或接近实时）分类。

**3.1。定义**

在进一步讨论之前，我们定义了本文后面使用的一些术语，并解释了AppScanner框架的核心其他关键概念。

**爆**

突发是一起出现的所有网络分组（不相关或源或目的地地址）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。也就是说，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的*Traffic Burstification*部分中可视地描绘，其中我们可以看到Burst A和Burst 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。Stöber等人先前使用过爆发的概念。[18]并在此类似地使用。

**流**

流是具有相同目的地**IP**地址和端口号的一系列分组（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）相同的目标**IP**地址/端口。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。来自图1的*流动分离*部分，我们可以看到一个爆发可能包含一个或多个流。流可能在突发重叠，如果一个单一的应用程序，*应用程序X*，开始快速连续或者如果另一个应用程序，TCP会话*应用Ÿ*，发生在同一时间发起TCP会话*X程序*。我们将在3.3节中解释如何准确地将流程归因于其原始应用程序。Conti等人先前使用过流动的概念。[17]并在这里类似地使用。

我们使用有监督的机器学习来对流进行模式识别。在AppScanner中，监督学习算法提供了来自每个app的流的标记示例（或从这些流中提取的统计特征），然后用于构建模型。然后可以使用这些模型对未标记的流进行分类。模型需要是轻量级的，因为我们需要AppScanner即使在处理/内存资源有限的环境中也可以部署，并且仍然可以实时或接近实时地执行应用程序分类。

**3.2。设备安装**

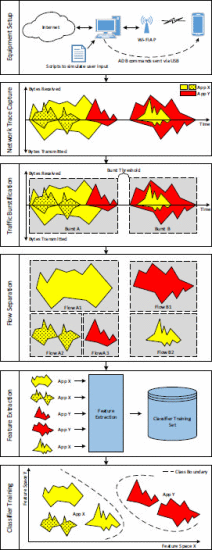
用于从应用程序收集网络跟踪的设置显示在图1的*设备设置*部分中。工作站配置为在Wi-Fi接入点（AP）和Internet之间转发流量。为了生成捕获我们的训练数据的流量，我们使用Android调试桥（ADB）通过USB与目标智能手机通信的脚本。这些脚本用于模拟测试设备上的用户操作，从而从应用程序中获取网络流。捕获流经工作站的流量并将其导出为逗号分隔值（CSV）文件，每行包含捕获的数据包的详细信息。我们收集了数据包详细信息，如时间，源地址，目标地址，端口，数据包大小，协议和TCP / IP标志。还收集了每个数据包的有效负载，但未用于提供功能，因为它可能加密也可能不加密。我们的目标是让AppScanner能够识别应用程序的流量是加密的还是未加密的。虽然物理硬件用于网络流量生成和捕获，但是通过在虚拟机上的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。

**3.3。指纹制作**

指纹制作过程由我们在图1中概述的许多步骤组成，并在下面描述。

**网络跟踪捕获**

网络跟踪捕获过程需要在硬件设置上运行用户仿真脚本。这些脚本生成了应用程序启动，触摸和按钮按钮，以从应用程序中获取交互式流量，然后使用数据包嗅探器收集这些流量。我们一次只运行一个应用程序，以最大限度地减少网络跟踪中的“噪音”。我们观察到测试集中的所有应用程序在启动时都会生成网络流。用户模拟可以通过各种方式完成，但我们利用标准的Android SDK UI练习器工具，*猴子*和*monkeyrunner*。AppScanner不会利用可能在人工生成的应用流量中强调的流模式或其他此类人工制品。因此，我们可以使用UI练习工具自动化培训过程（用于高可伸缩性）。UI自动化的使用可能导致我们无法获得应用程序的所有独特流程，但如果人类用户使用它，则获得的流量仍然是来自应用程序的真实流量流量。例如，应用程序仍将与其服务器签入，发送标准API查询，加载在线资源和其他类似任务。通常，应用程序建立的连接越多样化，当用于特征生成和随后的分类器训练时，流量将更加明显。[19]，或通过招募人类参与者。但是，我们认为这些方法超出了本文的范围。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-1-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-1-source-large.gif)

**图1：**

可视化tcp / ip流量内的突发和流，以及appscanner执行的分类器训练步骤的高级表示。

[查看全部](https://ieeexplore.ieee.org/document/7467370/all-figures)

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。但是，这些转储可能包含来自智能手机上运行（在后台）的其他Android应用程序的流量，这些应用程序可能会干扰并玷污指纹制作过程。除了目标应用程序，另一个开源应用程序，网络日志[20]，已在目标设备上安装并启动。网络日志用于识别负责来自测试设备的每个网络流的应用程序。通过这种方式，我们获得了来自什么应用程序的流量的完美基础事实。使用来自网络日志的记录数据与“解复用”脚本相结合，从该应用程序的流量转储中删除了所有未来自目标应用程序的流量。此时，每个网络转储仅包含来自目标应用程序的无错TCP流量。

**交通分流与流动分离**

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。[21]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。这向我们表明，自原始研究以来，网络性能（在带宽和延迟方面）已得到改善。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。这些爆发被分成单独的流（如3.1节所定义，如图1所示））使用目标IP地址/端口信息。我们强制实施AppScanner考虑的最小流量长度和最大流量长度。这只是为了确保AppScanner在现实世界中部署时安全地忽略异常流量。

值得注意的是，虽然目标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。这些都是经过深思熟虑的设计决策，用于了解AppScanner在最坏情况下的表现，以及避免依赖经常更改的特定领域知识。具体而言，由于以下原因，这些额外的数据来源可能被认为是不合适的：

* **IP地址 -**如果使用基于DNS的负载平衡/高，则应用程序联系的目标IP地址可能会更改。此外，许多应用程序联系类似的IP地址，因为它们使用相同的CDN或属于同一个开发人员。
* **DNS查询 -**由于使用客户端DNS缓存，因此并不总是发送/观察DNS查询。此外，多个应用可以发送相同的DNS查询，例如，以解析广告服务器域名。
* **数据包有效负载 -**许多应用程序开发人员越来越注重隐私，并选择使用HTTPS / TLS来加密数据包负载。因此，从TCP有效载荷中提取的特征将随着时间的推

虽然上述数据源可能会被仔细用于协助识别应用程序，但我们认为它们的使用超出了本文的范围，并将此类分析留待将来的工作中使用。

**特征提取与分类器训练**

一旦我们获得落在规定的流量长度阈值内的单个流量，就从它们生成特征并用于训练分类器。来自流的原始包长度被用作特征，以及这些流的统计特性。我们在第4节详细介绍了特征生成和分类方法/策略。

**3.4。指纹匹配**

指纹匹配阶段遵循与指纹制作阶段类似的步骤，直到特征提取步骤结束。此时，这些特征将被传递给要分类的预建模型，接下来是我们称之为“分类验证”阶段的结果。在指纹匹配期间，网络流量捕获阶段也有些不同，因为我们可以在实时网络捕获期间或在保存的网络跟踪上执行指纹匹配。

**网络流量捕获**

AppScanner可以在线和离线模式下工作，以捕获和处理网络流量。

* *在线模式* - 来自目标智能手机的网络流量使用*tshark*（Wireshark的终端版本）或类似工具直接从实时网络上的空中嗅探，并通过*tshark*包装传递到AppScanner的流量捕获模块图书馆。的*交通点播送分组*缓冲器收集进入网络分组，并且将它们传递到*流动分离*模块每当被看到的时间的经过，没有新的数据包的突发阈值量的突发。因此，AppScanner可以实时或接近实时地执行应用识别。
* *离线模式* - 可以将预先收集的网络跟踪输入AppScanner进行“批处理”。网络跟踪被解析为突发并传递到*流分离*模块，就像在线模式一样。

**Tcp预处理和流分类**

的*流动分离*模块，在接收到的网络流量突发，使用源和目的地IP地址将其分离成流。在将流传递到AppScanner中的下一个阶段之前，如果它们包含任何TCP重新传输或其他错误，或者它们超出流长度阈值，则会将其丢弃。丢弃包含TCP重传或其他错误的流，因为它们会将噪声引入不应存在的流中。如前所述，设置流长度阈值以确保非常冗长（并且最可能是异常）的流不会进入系统。我们讨论第5节中使用的实际流量长度阈值。特征生成（见图2对于如何完成此操作的概述，对这些无错误，经过验证的流执行，并且每个流的特征被传递到分类器以进行识别。然后将该分类的结果传递到最后阶段，称为分类验证。

**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。此外，正如我们在第5节中讨论的那样，来自不同应用程序的一些流量非常相似，如果不加以检查，这也会导致AppScanner的FP率出现不合需要的增加。

为了解决这些问题，我们利用分类器中可用的预测概率特征来了解分类器对每个分类的确定性。预测概率是由分类器报告的度量，其给出分类器对其将特定标签分配给未知样本的信心的指示。例如，如果分类器将未知样本标记为*com.facebook.katana*，我们将检查其分类的预测概率值以确定分类器的置信度。如果此值低于分类验证阈值，AppScanner将不会发表声明。但是，如果此值超过阈值，AppScanner会将其报告为该特定应用的匹配项。在第5节，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4节**

分类器设计

由于AppScanner是模块化的，因此可以使用不同的机器学习算法，只需进行修改即可轻松完成。我们设计并彻底测试了六种分类方法，如表1所示。每种方法都使用支持向量分类器（SVC）或随机森林分类器。选择这两个分类器是因为它们在使用我们从网络流中提取的功能进行训练时特别适合预测类（在我们的示例中为应用程序）。

支持向量分类器将训练示例建模为空间中的点，然后使用超平面划分空间以在类之间实现最佳分离。在非线性可分问题的情况下，支持向量分类器可以依靠内核函数将数据投影到高维特征空间中以使其可线性分离。随机森林分类器是一种集合方法，它使用多个较弱的学习者来构建更强大的学习者。该分类器在训练期间构造多个决策树，然后选择由各个树输出的类的模式。它还能够对其选择使用的功能的重要性进行排名（如表3所示）。

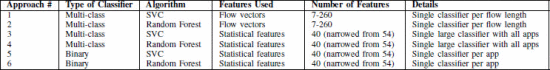
在表2中，我们概述了六种分类方法中每种方法的其他特征，这些方法说明了为什么人们倾向于采用特定方法而不是另 分类器在训练速度，分类器大小，每个分类的平均置信度，是否可以测量真实负数以及它们是否对无序数据包具有鲁棒性方面进行了比较。通常，*每流长度分类器*因为它们具有较小的训练集（仅具有一定长度的流动），所以训练更小且更快。我们使用数据集训练时包括方法的平均速度和大小。只有二​​元分类器能够理解真正的否定。只有使用统计特征的分类器对于无序数据包是健壮的，因为其他分类器在呈现交换数据包时会错误地分配特征。

用于训练分类器的特征是原始分组长度的实际流向量或从这些流向量导出的统计特征。图2广泛地示出了使用流向量或统计特征的两种方法。从图中可以看出，流预处理器只是改变了输入数据包的符号（即，使其为负）。然后，流预处理器的输出作为（可变长度）流向量传递给使用分组长度作为特征的分类器，或者传递给用于其他分类策略的统计特征提取功能。

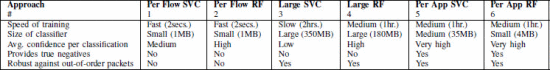
统计特征提取涉及从每个流中导出54个统计特征（无论流长度如何）。对于每个流，考虑三个分组序列：仅传入分组，仅传出分组，以及双向流量（即传入和传出分组）。对于每个系列（总共3个），计算以下值：最小值，最大值，平均值，中位数绝对偏差，标准偏差，方差，偏斜，峰度，百分位数（从10％到90％）和元素数量系列（共18个）。这些统计特征是使用Python pandas库计算的[22]。

然后，这些特征通过特征缩放器功能，该功能是最小 - 最大缩放器（即，训练集中特定特征的最小值和最大值分别对应于0和1）。为了避免维度的诅咒，使用特征选择功能来选择最佳特征。特征选择功能利用在训练集[23]上运行的随机森林分类器使用的基尼重要性度量。该度量依赖于在估计器构建期间计算的Gini杂质指数。在训练结束时，分类器根据其重要性对每个特征给出分数。此时，我们仅选择分数高于1％的那些特征，原始54的总共40个特征。在表3中，我们报告每个前40个功能的得分。

**表1：**在appscanner中测试的六种不同的分类方法。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-1-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-1-source-large.gif)

**表2：**六种分类方法的附加特征，有助于确定哪种方法更适合其特定部署。RF意味着随机森林分类器。大SVC和大RF参考方法3和4，每个方法具有单个大分类器。平均。每个分类的置信度是根据我们广泛测试的结果确定的。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-2-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-2-source-large.gif)

**方法1-2-使用a的多类分类**

**每个流程长度的分类器**

这些方法涉及训练多类支持向量分类器和随机森林分类器，其特征是来自每个流的分组大小的向量。对于支持向量分类器，我们使用了带参数*gamma*的*rbf*内核 = 0.0001，C = 10000，对于随机森林分类，我们使用的参数*搜索标准* = g我ñ 我*max\_features* = N.Ø ñ Ë ñ- Ë 小号吨我中号一吨Ô ř 小号= 150对一组广泛的超参数（具有5倍交叉验证）的详尽搜索被用于优化这些参数。来自流的特征数组的长度等于流中的分组数量，因此等于流长度的分类器ñ 将接受培训 ñ每个训练示例的功能。每个流长度只有一个分类器是可能的（因为分类器中的每个训练示例需要具有相同数量的特征），因此我们具有*最大流*分类量的分类器。

**使用单个大型分类器的方法3-4-多类分类**

这些方法涉及训练多类支持向量分类器和随机森林分类器，其特征是从每个流的分组大小的向量导出的统计特征。在这些方法中，每个分类器都非常大，并且包含测试应用程序集中的所有应用程序。支持向量分类器的参数是内核：=升我ÑË一个řÇ= 100，对于随机森林分类器，我们使用参数*标准* = g我ñ 我 *maX\_features* = sqr t n- Ë 小号吨我中号一吨Ô ř 小号= 150。对一组广泛的超参数（具有5倍交叉验证）的详尽搜索被用于优化这些参数。

**方法5-6-每个应用程序使用单个分类器进行二进制分类**

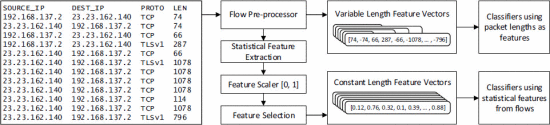
这些方法涉及训练二进制支持向量分类器和二进制随机森林分类器，其特征是从每个流的分组大小的向量导出的统计特征。对于支持向量分类器，我们使用带有*gamma*的*rbf*内核 = 0.001，C = 100。对于随机森林分类器，我们使用参数ñ- Ë 小号吨我中号一吨Ô ř 小号= 10。在这些方法中，每个分类器都是二元分类器，并且经过训练以识别一个应用程序。由于分类器具有二元性质，因此未标记的流程在被分类时并行地传递给110个分类器中的每一个。

在训练分类器之后，使用序列化将模型保存到持久状态。通过序列化训练好的分类器，它们可以在下次使用时几乎立即加载而不会受到再训练的惩罚。

**第5节**

系统评估

在本节中，我们将介绍实验设置以及我们在AppScanner上执行的测试结果。为了构建和测试我们的框架，我们使用了运行Android 4.4.4（KitKat）的Motorola XT1039（Moto G）智能手机。智能手机通过Linksys E1700 Wi-Fi *路由器/ AP*连接到互联网，该*路由器*的互联网连接通过带有两个网络接口卡的Dell Optiplex 9020工作站进行路由。每个应用程序在150轮中自动运行（使用3.2节中概述的程序），持续75分钟，并使用Wireshark收集最终的网络流量。我们使用scikit-learn机器学习库在Python中构建了分类器[24]。在训练过程结束时，分类器在称为酸洗的过程中被序列化。Pickling是Python提供的一项功能，它允许将分类器数据结构和对象状态转换为文件。

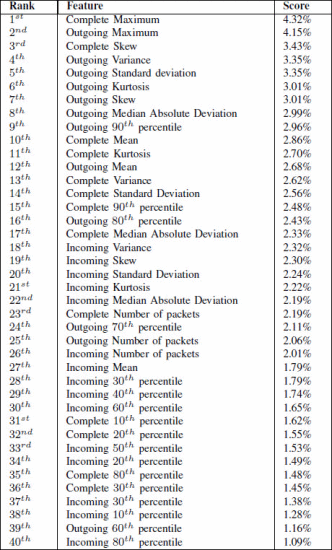
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-2-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-2-source-large.gif)

**图2：**

从流中提取特征 - appscanner用于从分类器训练的流生成特征的两种主要方法。

[查看全部](https://ieeexplore.ieee.org/document/7467370/all-figures)

**表3：**表格显示超过阈值1％的40个统计特征的百分比分数。

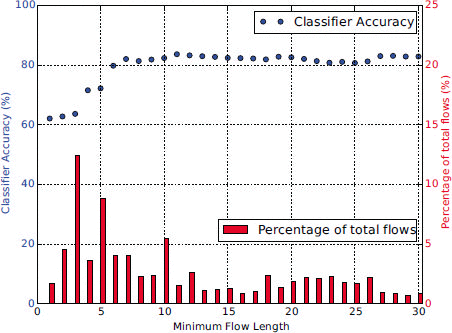
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-3-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-3-source-large.gif)

该实验的目的是找出从网络捕获的交互式流量中指纹和重新识别应用程序的准确程度。对于这些测试，AppScanner接受了来自110个应用程序的交互式流量培训。这些应用程序是从2015年7月Google Play商店中列出的150个顶级免费应用程序中随机选择的。（请参阅附录中的表9，了解用于测试AppScanner的应用程序列表。）

我们选择了最受欢迎的应用，因为我们认为这些应用代表了全球应用总安装量的非常大的横截面。如果AppScanner在这些应用程序上表现良好，则表明AppScanner作为在全球范围内识别应用程序的框架的有用性。此外，我们使用免费应用程序，因为免费应用程序往往包含比付费应用程序更多的广告，因此会产生更多的广告流量。由于特定广告网络提供的广告流量趋于相似，因此AppScanner在将广告流分类为属于来自使用相同广告网络的应用组的一个应用时将具有更困难的任务。的确，我们的结果证实了这一点 出于这个原因，我们认为我们从AppScanner获得的免费应用程序测试结果是最差的性能数据。

**5.1。测量Appscanner的性能**

在训练分类器之前，我们需要为要考虑的最小流量长度选择合适的值。对于该测试，我们选择了我们的*每流量随机森林分类器*（方法2）并改变最小流量长度阈值，同时保持最大流量长度阈值恒定在无穷远处。图3显示了改变最小流量长度对分类精度的影响。分类器精度从一个包的流长度急剧增加到七个包的流长度，并且之后保持恒定（或多或少）。这是可以理解的，因为较短的流携带较少的信息，因此，我们期望分类器在分类较短的流时产生更多的错误。流量长度为7是最小流量长度的良好选择，因为它是最短“完整”流量的长度; 即，包含TCP握手（三个数据包）的流程，后跟HTTP请求，响应和确认（四个数据包）。请注意，我们不考虑最短完整流长度的TCP会话终止数据包。这是因为*突发阈值*通常会在TCP会话终止数据包之前发生，因此它们永远不会成为流的一部分。由于这些原因，最小流量长度为7用于其余测试。当然，这可以根据任何其他特定需求轻松调整。其他分类方法产生的图表具有类似的分类器精度与最小流量长度的行为，为简洁起见省略。

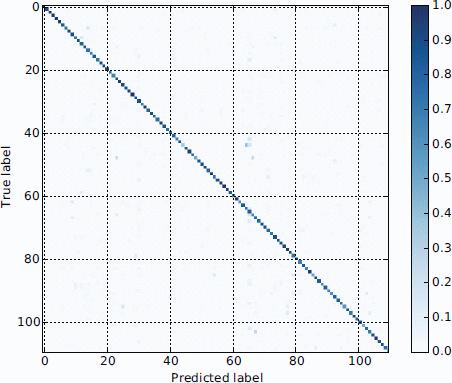
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-3-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-3-source-large.gif)

**图3：**

最小流长对每流随机森林分类器分类器精度的影响。

使用最小流量长度7和任意最大流量长度260，我们的交互式流量包含131,736个流量，分别为训练/测试集的75％125％。我们使用的最大流量长度为260，因为这是我们的训练数据中观察到的最长流量的长度。可以根据典型使用场景中预期的最大流量长度轻松调整该值。我们使用训练集中的特征训练分类器，并通过将它们的预测与测试集中的基本事实进行比较来测量它们的准确性。对于第一轮测试，未使用分类验证。这是为了帮助我们理解分类器如何在没有任何额外后处理的情况下执行。图4显示了我们的混淆矩阵*每流随机森林分类器*（方法2）。为简洁起见，我们只显示一个混淆矩阵，因为其他分类方法产生了类似的图。此外，在混淆矩阵本身中，110个应用程序中的每一个都在轴上分配了唯一的数字（0-109），而不是显示应用程序名称。y轴显示负责流程的真实应用程序，而x轴显示从分类器输出的预测应用程序。混淆矩阵中的单元显示每个流的分类情况，较暗的颜色表示更准确的分类。

接下来，我们计算了六种分类方法的精确度，召回率和准确度。其中*TP*指的是真阳性的数量，*FP*指的是假阳性的数量，*FN*指的是假阴性的数量，*TN*指的是真阴性的数量：精确度是使用公式计算的ŤP/（TP+ F.P），召回是使用公式计算的 ŤP/（TP+ F.ñ）。对于方法1-4（多类分类器），精确度计算为正确分类的总数除以分类总数。方法5-6涉及二元分类器，因此精度计算为（T.P+ T.ñ）/（TP+ F.P+ T.ñ+ F.ñ）。结果报告在表4中。在不使用分类验证的情况下，AppScanner在针对来自网络流的统计特征进行训练的*Per App随机森林分类器*方面具有最佳的整体性能（方法6）。对于110个应用程序的测试集，这些分类器的总体精度为96.0％，召回率为82.5％，准确率为99.8％。我们的*Per App支持向量分类器*（方法5）具有可比较的精度和准确度，但较低的召回率为64.8％。鉴于没有对表5中显示的结果使用分类验证，这些是AppScanner使用这些分类方法可以预期的最差情况性能数据。

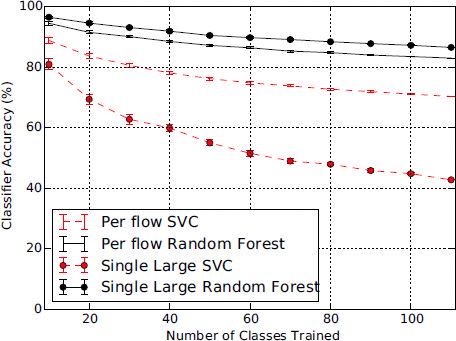
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-4-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-4-source-large.gif)

**图4：**

规范化混淆矩阵，显示每流随机森林分类器的实际类别与预测类别。

[查看全部](https://ieeexplore.ieee.org/document/7467370/all-figures)

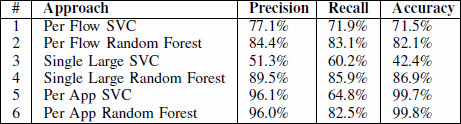
对于下一轮测试，我们想要测量增加类数对我们的多类分类器的分类精度的影响（方法1-4）。我们从一组10个应用程序开始，这些应用程序是从110个应用程序的测试集中随机选择的。测量分类性能。将该测试重复50次（使用相同组大小的随机组）并将结果取平均值。重复整个过程，每次将应用程序集大小增加10，直到我们的最大设置大小为110.这些测试的结果如图5所示。。从图中我们可以看出，增加分类器中的应用程序数量会导致分类的精确度，召回率和总体准确度降低。这并不意外，因为多类分类器的准确性是未知输入可以匹配的类数的函数。然而，重要的是要注意，随着类的数量增加，分类器性能的降低速率降低。因此，当类的数量显着增加时，我们期望分类器性能最终趋于平稳并保持不变。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-5-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-5-source-large.gif)

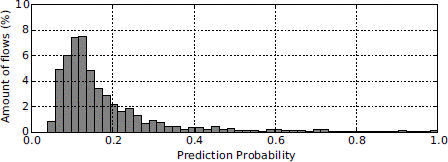
**图5：**

分类器中训练的应用程序数量对四个多类分类器的分类器性能的影响。误差线显示平均值的95％CI。

**表4：**显示六种分类方法的分类器性能的表：每流SVC，每流随机森林分类器，单大SVC，单大随机森林分类器，每应用SVC和每app随机森林分类器。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-4-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-4-source-large.gif)

如果想要指纹应用程序的范围，他们将使用*每个流长度分类*策略（第4节中的 Ap-proach 1-2））。这将确保没有单个分类器包含大量应用程序，因为并非所有应用程序都为每个流长度生成流。出于这个原因，我们认为扩展AppScanner以识别应用程序的范围是可行的。然而，这种大规模的“appscanning”并不常见，因为我们认为典型的应用场景是使用AppScanner来定位某个应用程序子集。例如，通过对前10,000个应用程序进行指纹识别，可以对给定网络上可见的所有应用程序（具有非平凡的可能性）进行实际覆盖。在其他情况下，我们希望部署AppScanner仅识别非常特定的应用程序（例如公司政策允许/禁止的应用程序），在这种情况下，*每个应用程序*的*单一分类器*策略（方法5-6 in将使用第4节）。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-6-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-6-source-large.gif)

**图6：**

由单个大svc输出的每个分类的预测概率的直方图。

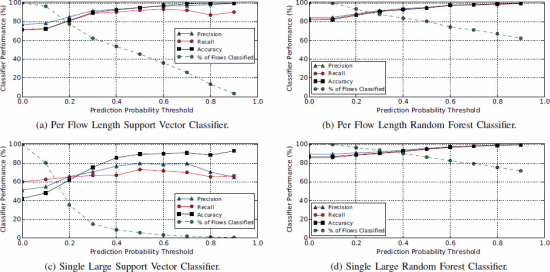
**5.2。使用分类验证来提高性能**

为了理解分类验证后处理函数的效用（详见3.4节），我们研究了我们的多类分类器对每个分类报告的置信度。图6示出由我们的表现最差的分类器所报告的预测概率的直方图*（单一大型SVC）*为≈ 33 ，500测试集中的流程。预测概率具有均值μ = 0.18 有标准差 σ= 0.14。对于绝大多数的分类，分类器对其决定的确定性不到20％。实际上，我们从图中可以看出，分类器对大量选择的信心仅为10-12％左右。

在多个应用程序具有类似流程的情况下，例如广告/分析流量或查询类似的API，分类器对其分类不会非常有信心是可以理解的。之所以这样，是因为类边界不会像所有应用程序都具有完全独特的流量那样明显。这表明分类验证可以是提高分类性能的有用策略，因为我们可以为分类器接受的“最小标准”设置为自信分类。通过使用分类验证，我们可以让AppScanner免于对分类器真正非常模糊的流程做出决策。

表5总结了我们通过使用分类验证获得的分类性能的（有时）显着改进。通常，随机森林分类器的性能优于我们数据集的支持向量分类器，无论是使用*分类器每流长度*还是*单个大分类*器。随机森林分类器使用聚合决策树，这反过来又减少了偏差。而且，它们能够更好地处理噪声，因为它们是一种集成学习方法。支持向量分类器对它们的预测不是很有信心，实际上可以看出，随着预测概率阈值的增加，它们足够自信地分类的流量百分比急剧下降。总冠军是*单一大型随机森林分类器*（方法4）使用从流量导出的统计特征。我们现在详细介绍这些分类方法的性能。

图7a显示了我们的*每流长度支持向量分类*器的分类器性能（方法1）。在没有使用分类验证的情况下，精度，召回率和准确度分别为77.1％，71.9％和71.5％，分类器对所有未标记流量进行判断。通过将预测概率阈值设置为适度的0.5，精度，召回率和准确度分别增加到95.1％，92.4％和95.0％，分类器对未标记流量的不到一半（45.5％）进行判断。从图中可以看出，通过将预测概率阈值设置为0.7，可以实现超过99％的准确度（准确度为99.1％，精度为97.2％，召回率为92.3％）。然而，在0.7的门槛下，AppScanner只有足够的信心对大约四分之一（26.0％）的流量做出判断。在更高的阈值，

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-7-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-7-source-large.gif)

**图7：**

预测概率阈值对分类器性能的影响。

图7b显示了我们的*每流长度随机森林分类*器的分类器性能（方法2）。由于未使用分类验证，精确度，召回率和准确度分别为84.4％，83.1％和82.1％。在0.5的阈值处，准确度跃升至94.7％，并且在0.7的阈值处我们可以超过98％的准确度，同时仍然对超过71％的流量进行分类。尽管*每流量支持向量分类器*具有更高的峰值精度，但是它们可以在更高阈值处分类的流量百分比使得它们仅在非常特定的情况下有用。

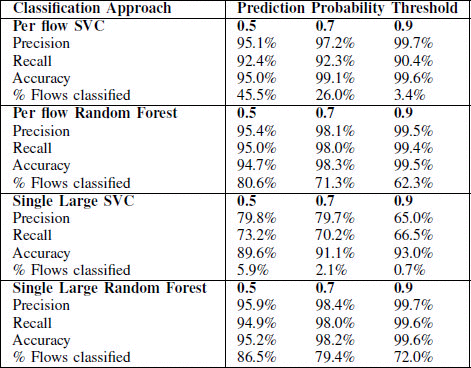
图7c显示了我们的*单个大型支持向量分类器*（方法3）的分类器性能。这是我们表现最差的分类器。由于未使用分类验证，精度，召回率和准确度分别为51.3％，60.2％和42.4％。通过调整预测概率阈值，可以从该分类器中挤出额外的性能，但这会损害分类的流的百分比。分类的流量比在*每流量长度分类器*方法（方法1）中使用的相同类型的分类器更加急剧下降。在0.5的阈值处，准确度小于90％并且为了实现这一点，分类器仅能够对5.9％的流进行分类。

图7d显示了我们的*单一大型随机森林分类*器的分类器性能（方法4）。这是我们表现最好的分类器。由于未使用分类验证，精确度，召回率和准确度分别为89.5％，85.9％和86.9％。在0.7的阈值处，所有三个精度，召回率和准确度均超过98％，并且在0.9的阈值下，精度，召回率和准确度均超过99.5％。这种近乎完美的精度得以实现，同时仍然可以对大约四分之三的所有流量进行分类。

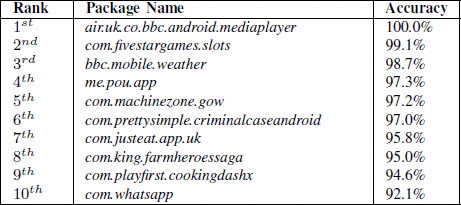
**5.3。了解分类错误**

在精确度/召回率/准确度以及使用分类验证时分类的流量百分比方面，一些分类方法的表现优于其他方法。通过AppScanner进行分类时，某些应用程序本身也比其他应用程序表现更好。我们希望性能更好的应用程序是那些流量与其他应用程序流量截然不同的应用程序。为了验证这个假设，我们分析了我们的*流量随机森林分类器*（方法2），这个分类器来自中间某个地方（不是最好而不是最差）的整体表现。表6显示由此分类器最准确分类的应用程序（不使用分类验证）。我们删除了此步骤的分类验证，以更全面地了解被错误分类的流的类型。*airuk.co.bbc.android.mediaplayer*软件包在所有情况下都被完美地分类，另外15个应用程序的分类准确度超过了90％。其他应用程序执行得更糟，如表7所示。*com.google.android.apps.plus*包中没有任何流量不使用分类验证时，AppScanner正确分类。我们测试的110个应用程序中的另外13个应用程序使用此设置执行低于50％的标记。这些应用程序似乎生成流程更难分类，从而产生更多的误报和误报结果。为了理解是否是这种情况，我们对错误分类的流程进行了深入分析，以获得更多洞察力。随着分类验证仍然被删除，我们执行了另一组测试，其中AppScanner将最佳地猜测流属于哪个应用程序。我们使用分类方法1进行了这种分析; 性能处于中间的另一种方法（不是最好的而不是最差的）。我们收集了≈ 10 ，000 流动的 ≈ 33 ，500 ）被错误分类（使用这种分类方法）并按目的地**IP**地址对它们进行手动/半自动分析。

**表5：**使用分类验证时总结多类分类器性能的表。

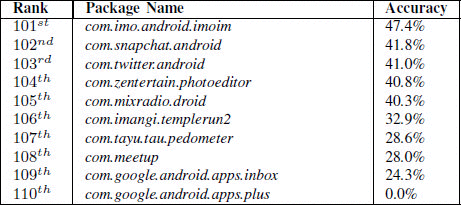
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-5-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-5-source-large.gif)

**表6：** appscanner分类的最佳10个应用程序。

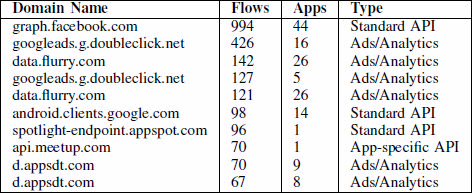
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-6-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-6-source-large.gif)

该 ≈ 10 ，000错误分类的流量将达到约1,467个唯一目标**IP**地址。值得注意的是，这些**IP**地址中的前25个占错误分类流量的30％以上。为简洁起见，我们在表8中报告了前10个目的地（针对错误分类的流量）。该表显示了域名，流向每个域的流标记错误的数量，向每个域发送流的应用程序数以及在该域上运行的服务类型。请注意，在域中运行的服务类型是通过以WHOIS查询的形式手动对域进行研究，访问域，分析子域等来推断的，因此在所有情况下可能都不完全准确。

**表7：**通过appscanner进行分类的最差10个应用程序。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-7-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-7-source-large.gif)

**表8：**标记不正确的流量的前10个目的地，流向这些目的地的流量，发送这些流量的不同应用程序的数量以及每个目的地的服务类型。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-8-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-table-8-source-large.gif)

根据表8，除了两个域之外，我们排名前十的“最差分类域”列表中的所有域都从多个应用程序中获得流量。这些“多应用”域的其余部分提供标准开发人员API或广告/分析资源。这支持了我们的假设，即由于他们联系相同的标准Web服务，类似的流确实是由多个应用程序发送的。在这种情况下，没有太多可以帮助AppScanner区分这些网络流的确切来源而不利用其他功能。但是，如果允许AppScanner更通用并将流标记为广告/分析/标准流量而不是命名特定应用，则可以提高分类准确性。

**第6节**

讨论和未来的工作

AppScanner是一种新颖的方法的实现，它利用机器学习和流量分析来自动指纹识别智能手机应用程序。智能手机领域为流量分析提供了独特的挑战，例如可用性较低的功能以及对自动化和高可扩展性的需求。我们基于流程的分类框架提供了新的见解。我们探索了三种一般分类策略（即每个流长度的分类器，所有应用程序的单个分类器和每个应用程序的单个分类器），并根据训练分类器所花费的时间，所得分类器的大小，探索和列举每个策略的权衡，和分类性能。

我们的框架能够非常准确地从他们的网络流量中识别应用程序，但它也有一些限制。我们在6.1节讨论这些限制，我们将AppScanner与6.2节中的现有技术进行比较，并在6.3节讨论未来的工作。

**6.1。限制**

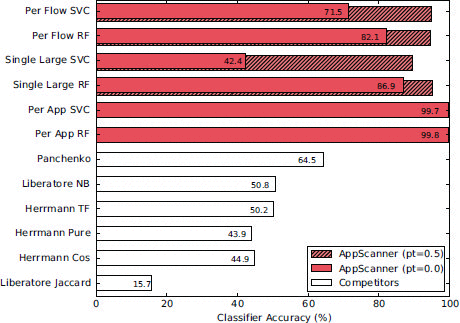
表7显示AppScanner在识别一些应用程序（如Temple Run，计步器，MeetUp，Gmail收件箱和Google+）方面更糟糕。我们认为，这是由于这些应用程序具有非常通用的流量流量。表8支持这一假设，我们发现标记最不正确的流程来自多个应用程序到达类似目的地。简单的事实是，模糊流程难以分类，AppScanner（或任何其他系统）无法在不利用其他功能的情况下可靠地区分这些流程。

AppScanner使用单个设备构建，可生成培训和测试流程。应用可能会在不同设备或不同版本的Android上表现不同。如果使用来自不同设备的网络跟踪训练它们，则不同设备上的不同风格的TCP也可能导致我们的分类器错误分类。我们计划通过使用在虚拟机上运行的各种Android模拟器从应用程序生成流来测试这一点。

**6.2。与网站指纹识别方法的比较**

由于本文的范围是智能手机应用程序指纹识别，我们在第2节中确定的最接近的相关工作主要集中在网站指纹识别，因此无法直接比较AppScanner与相关工作。但是，我们通过展示文献中现有的网站流量分析技术（存在基本事实）在从我们的数据集中对智能手机应用流量进行分类时的表现低于标准，来验证AppScanner的必要性和实用性。我们的比较结果如图8所示。

考虑进行比较的第一组方法是Liberatore等人提出的方法。在[11]中：利用Jaccard相似性度量（即*Liberatore Jaccard）*和另一个利用Naive Bayes分类器（即*Liberatore NB）的*分类器。在这两个分类器中，*Liberatore NB*以50.8％达到最佳精度。第二组方法由Herrmann等人提出。在[12]。与该组中的提议的不同之处在于转换应用于数据集：无转换（即*Herrmann Pure）*，术语频率转换（即*Herrmann TF）*和在TF转化后应用的余弦标准化（即*Herrmann Cos）*。最佳性能是50.2％的准确度，通过没有余弦归一化的TF变换，即*Herrmann TF实现*。最后，Panchenko等人提出的方法。在[13]中表现最好，准确率为64.5％。正如我们从图8中可以看到的那样，这种方法的性能与我们的相近。然而，我们的六个分类器中有五个优于它，我们的两个最佳方法的准确率超过35％。当使用分类验证（具有0.5的适度预测概率阈值）时，我们最糟糕的四种方法优于Panchenko等人。精度为25％-30％。

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-8-source-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/7467055/7467331/7467370/7467370-fig-8-source-large.gif)

**图8：**

Appscanner与文献中现有方法的准确性相比。我们使用pt来表示用于分类验证的预测概率阈值。RF意味着随机森林分类器。

### 6.3。未来的工作

对于未来的工作，我们将研究分组流的方法，以更可靠地确定原始应用程序。例如，当单独分析时，三个流可能是不明确的，但是当作为一组进行评估时，它们可能匹配总是将这些流中的三个一起发送的特定应用。我们还将介绍识别应用程序的其他方法，例如主动网络探测，这些方法可用于从应用程序中进一步识别网络流量。我们打算使用不同的建模工具，如隐马尔可夫模型和有限状态机，进行应用程序分类。我们还计划通过识别和使用流中的其他功能（例如数据包到达时间）来提高分类准确性。还可以利用其他容易获得的信息，例如流是否总是在具有多个流的突发内发生，或者是否包含HTTPS / TLS分组，以提高准确性。最后，我们计划在存在MAC层加密的情况下检查在MAC层可以进行app指纹识别的程度。

**第7节**

## 结论

在本文中，我们介绍了AppScanner，这是一个实现一种新方法的框架，用于从加密网络流量中自动指纹识别和实时识别智能手机应用程序。我们的评估表明，即使存在加密流量流（如HTTPS / TLS），也可以确定应用程序的准确率超过99％。我们验证了多类分类器可用于在单个分类器中指纹识别各种应用程序。我们还表明，在仅关注某些应用程序的情况下，二进制分类器也可用于获得非常高的精度和总体准确性。毫无疑问，随着应用开发者继续向消费者提供新应用以满足他们无法满足的胃口，智能手机的使用量将继续增加。结果是，越来越多的演员会因为仁慈和恶意的原因而对指纹识别和识别这些应用程序感兴趣。通过继续在该领域的研究，我们希望更好地了解最终用户目前面临的隐私和安全风险。通过这种方式，我们可以继续走现在，帮助保护现在和未来的隐私和安全。