2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems

TrafficVision: A Case for Pushing Software Defined Networks to Wireless Edges

Mostafa Uddin
Department Of Computer Science
Old Dominion University, Norfolk, USA
Email: muddin@cs.odu.edu

Tamer Nadeem
Department Of Computer Science
Old Dominion University, Norfolk, USA
Email: nadeem@cs.odu.edu

***Abstract*—In wireless network edges, knowing the network flow types and applications can enable various policy-driven network managements (i.e. traffic offloading, BYOD, E2E QoS etc.). However, applying network policies at wireless links between the mobile devices and access points (APs) requires greater visibility and control on generated traffic generated from mobile devices. The recent advent of Software-Defined Networking (SDN) could enable fine-grained network management at the edge. However, in existing solutions, SDN uses external Deep-Packet Inspection (DPI) engine that requires additional and potentially heavy loaded computational resources to perform the packet analysis. Moreover, mobile applications become more dynamic (rapid install/update), diverse and complex (individual applications generate multiple traffic types) in which scalability and granularity requirements challenging current DPI solutions. In addition, DPI is unreliable in classifying application's encrypted packets. Therefore, in this paper we present the design and the development of *TrafficVision* that extends the SDN's layer architecture to have fine-grained and real-time policy making at wireless network edges. More specifically, we carefully extends the SDN framework to develop tools that allow to have scalable, efficient and flexible way to classify the network traffic flows at fine-grained fashion using Machine-Learning (ML) based technique. We evaluate our system using the performance of CPU utilization, network overhead and network throughput metrics. Finally, as a proof of concept, we develop a simple case study of traffic management application that exploits *TrafficVision*.**

***Keywords*-Traffic Classification; Software Defined Network; Wireless Network;**

I. Introduction

We are approaching a fundamental shift in the computational era as the penetration of smart devices (e.g., smartphones and tablets) is expected to reach 60% of the global population in 2019 [33]. Given the advancements in microprocessors and the development of new types of connected smart devices with different capabilities such as smart watches, smart glasses, smart meters, and connected vehicles, we are seeing the next phase of the Internet populated with traffic primarily from these devices forming what we call Internet of Things (IoT). Cisco predicts the number of connected IoT devices to reach 50 billion by 2020 [49], [16].

This explosion in number of smart devices, as well as various applications and services, results in a significant growth of network traffic as well as new traffic types. According to Nielsens survey on mobile consumers [4], smart devices become a significant source of data with multimedia streaming (radio, music, video) dominating the data demand. Therefore, number of researchers have proposed a new research paradigm referred to as *"edge computing"*, where services and applications runs at *wireless network edges* [8], [43] and closer to client devices for low latency, high bandwidth, and privacy [51], [52], [53]. In this paper, wireless network edges are used to refer to both *end devices* (e.g., smartphone, tablets, smart watch etc.) and *wireless access devices*[1] (e.g., Wi-Fi access points (APs), base stations, edge routers etc.).

With this context of pushing computation and storage resources closer to client devices, it is essential to have flexible and efficient network management at wireless network edges to support complex network management and configuration tasks such as end-to-end QoS for various network traffic, different traffic engineering schemes with various policies, and efficient load balancing [27], [45], [50], [15], [30]. Recently, network community have embraced Software Defined Networking (SDN) [24], [31] and Network Function Virtualization (NFV) [42] in network core components as well as access devices (i.e., cellular/Wi-Fi backhaul network) to ease many aspect of "edge computing" such as resource allocation, VM migration, traffic monitoring, application-aware control and programmable interfaces [52], [46]. In addition, SDN solution ease the implementation of network management on switches/routers/APs by providing enough flexibility at access devices.

The need for a smart network management supporting various policies for wireless network edges traffic urges to have light-weight, real-time fine-grained application and flow type awareness. Existing SDN solutions for application and flow-type awareness depends on the centralize deep packet inspection (DPI) engine [6], [20], [5] for traffic classification. Unfortunately, unlike edge data center machines, many devices of the wireless network edges are not computationally powerful for supporting DPI based solution [38], [6]. In addition, now-a-days DPI techniques have limitations in providing fine grained traffic classification due to payload encryption, privacy issues, and tunneling transfer [54], [38]. Therefore, SDN based

Mostafa Uddin is now with Bell Labs (Nokia) and could be reached at mostafa.uddin@nokia.com

[1] In the rest of this paper, we refer to them as *access devices*.

2155-6814/16 $31.00 © 2016 IEEE
DOI 10.1109/MASS.2016.52

centralized solutions of traffic awareness are inefficient and impose significant overhead (e.g., delay) in order to control the traffic of the wireless network edges [19]. Furthermore, these centralize solutions are incapable to take into consideration the end-device's context for smarter network management.

In this paper, we present our design, development and evaluation of a light weight and flexible application and flow-type awareness framework, called *TrafficVision*, for wireless network edges. *TrafficVision* is the first of its kind that provides on-fly fine-grained visibility and control over the network traffic generated by different applications and corresponding various flow-types running on wireless network edges. In *TrafficVision*, we push the SDN-like paradigm all the way to end-devices, by extending and deploying Open vSwitch [3] and OpenFlow protocol [34]) on both *end devices* and *access devices*, to extract new flow statistics such as packet sizes, directions, sequences, and timestamps, where a 'flow' is identified by 5-tuple fields. These extracted statistics are passed to the control layer using our extended *OpenFlow* protocol. In *TrafficVision*, we develop a network service in the control layer, *TV Engine* that provides scalable, efficient and real-time solutions for classifying the network traffic flows based on Machine-Learning (ML) techniques. *TV Engine* uses new features that extracts high-order frequency and temporal information from the packet sizes and arrival timestamps, improving the flow-type detection accuracies from 75-89% (using the state-of-the-art [35], [47], [11]) to 85-98%. *TV Engine* is also the core network service that provides a set of API to the upper Application layer in order to develop sophisticated network management applications. We also implemented *TrafficVision* in real test-bed in order to evaluate the efficiency and the overhead of the framework. Finally, as a proof of concept, we develop two case studies of our *TrafficVision* that allow us to automatically apply simple traffic management policies on the wireless network edges.

II. Background and Related Work

SDN has three layers network architecture; *data layer*, *control layer*, and *application layer*. In SDN, control layer is the core of the entire network system that maintains a global view of the network for network application layer, and fully controls the behavior of the network devices (e.g., router, switches, Wi-Fi AP etc.) thru data layer. In SDN data layer, Open vSwitch [3] (OVS) is a multilayer OpenFlow [34] supported software virtual switch that runs in the network devices. Note that OVS has both user-space and kernel-space components. The kernel space component is called Datapath.

SDN *data layer* (i.e. OpenFlow switches) and *control layer* only analyze Layers 2-4 packet header fields, SDN cannot recognize packet's application or flow type information. Note that traditional schemes that use port number and IP address for application/flow classification is not reliable in many cases since now-a-days applications start to use non pre-defined or dynamic port numbers [55], [14]. Therefore, existing applications/flow identification schemes require direct user/application inputs [27], [15], [18] or custom integration with application servers [45]

Although solutions using Deep Packet Inspection (DPI) can be more accurate, they incur high computation cost and limitation in differentiating between the various type of network flows generated from the same applications[40]. Moreover, DPI tools require complex reverse engineering or manual process of building or updating the packet signatures for new or updated mobile applications. Therefore, the exponential growth in diverse mobile applications and the requirement of fine-grained application detection makes DPI-based solution impractical. Moreover, DPI schemes fail to classify encrypted packets. As more mobile applications are using encrypted content and data, more concerns are being raised on the future of Deep Packet Inspection [20], [5]. To overcome this issue, current DPI schemes uses heuristic classification techniques for encrypted traffic, which is mostly unreliable and slow. Despite such difficulties, DPI is a popular solution with the SDN framework. In many commercial solutions, DPI is deployed with the SDN control layer that is referred as DPI engine. However, several earlier studies [9], [23], [38], [54] have already claimed that DPI engines are computationally expensive, which make such DPI based solution inapplicable in wireless edges.

Machine Learning (ML)-based approaches [26], [48], [39], [35], [10], [17], [54], [38], [37], have much lower computational cost than DPI-based solutions [48], and can correctly identify encrypted traffic in many scenarios. There are wide range of work on ML based traffic classification in the past, for example [25], [28], [29]. However, ML approaches have so far been restricted to traditional internet applications with coarse-grained classifications such as web, P2P and VoIP [26]. Also, recent works in mobile application detection [13], [32], [44] have only targeted coarse-grained application classification such as p2p, email, web browsing, and game application. Therefore, existing coarse-grained solutions can not differentiate between video flows generated by different multimedia applications. In fact, this deep level of traffic awareness is crucial for delivering an optimal QoE for end users. For example, Netflix and Livestream are two applications that generate similar video traffic as both use the same streaming technology, HTTP adaptive streaming over TCP. Since Netflix app comes with larger streaming buffer, as it is designed for on-demand videos, it is more robust to the fluctuation in the bandwidth than Livestream. On the other hand, Livestream app is more subject to playback stalls and instability in the viewing quality even with short term drop in the available bandwidth. Therefore, when these two applications start competing over the bandwidth, it will be more efficient to assign more network resources or priority to the video streaming flow generated by Livestream than Netfilx. However, existing coarse-grained application aware solutions are unable to differentiate the video streaming flows of Netflix and Livestream. Therefore, they can not apply network resource policies for addressing the competition over the bandwidth, which often results in delivering unsatisfactory level of QoE.

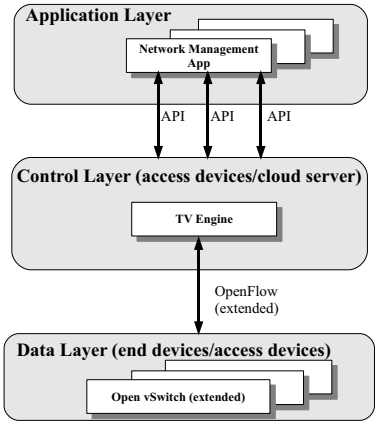


Fig. 1: High-level system architecture of *TrafficVision* within SDN framework.

Unlike traditional applications, mobile applications (e.g., facebook, skype, tango, fringe etc.) often generate various types of flows (e.g., video chat, voice chat, video stream, etc.). For example, Skype application can do voice, video, screen sharing, file sharing, and Instant Messaging (IM) flows as well as the background traffic for signaling, analytics, advertisement, etc. Therefore, for addressing mobile applications we require fine-grained mobile traffic identification, where we need to identify both the application name (i.e., youtube, vimeo, skype etc.) and the flow types (i.e., video stream, audio stream, video chat etc.)

In [40], researchers have proposed SDN solution that uses Machine Learning (ML) technique to classify mobile application's traffic. However, the proposed schemes are coarse-grained and incapable of providing fine-grained application name and flow-type classification. Note that, different flow types of different applications have different network loads, QoS requirements, security/resource policies, etc. Thus, it is very critical to be able to accurately and efficiently recognize individual mobile applications and its various traffic flows in real-time. In *TrafficVision*, for the first time, we propose to extend and use OpenFlow [34] protocol with Open vSwitch (OVS) [3] to collecting flow level features for traffic classification. Note that OpenFlow provides flexible network control and fine-grained network monitoring capability. Previously, researchers have used Netflow [22] for collecting flow level features for real-time traffic classification[12], [23], [41].

III. TrafficVision System Overview

Figure 1 shows the overall architecture of *TrafficVision*, where at the bottom we have the data layer that consists of *end devices* and *access devices*. In *TrafficVision*, these devices runs Open vSwitch that, unlike the conventional routing table, maintains a *flow table* to make forwarding decision when a packet arrives at the switch. A *flow table* consists of flow-entries with an action associated with the flow. In addition, existing flow table maintains entries of simple statistic about each flow-entry, such as total byte count and total packet count. In *TrafficVision*, we extend Open VSwitch (OVS) to collect additional statistic for each flow table entry, such as the packet sizes and the packet arrival timestamps of recently receiving packets. We also extended OpenFlow protocol to allow the OVS to communicate the additional flow statistics of packet sizes and packet arrival timestamps with the control layer. In section IV, we describe more details about the extension of the OVS and OpenFlow protocol.

In *TrafficVision*, at control Layer, we develop (section V-C) and evaluate (section VI) the network service component, *TrafficVision Engine* (*TV Engine*), which is the core part of our traffic-aware framework for the network edge. *TV Engine* uses the extended OpenFlow protocol to collect the additional statistics of the packet sizes and the packet arrival timestamps information from the OVS periodically. Furthermore, *TV Engine* aggregates the additional statistics extracted from the OVS and keeps a local record of the packet arrival timestamps and the packet sizes information for each flow. It also makes sure no packet is counted twice in record collection. Then, *TV Engine* extracts features from the collected record of packet arrival timestamps and packet sizes over the recent time window duration(e.g. window size 2000ms). All records of packet sizes and packet arrival timestamps before the time window are removed. Finally, *TV Engine* uses the extracted features to identify the application and the flow type using classifier; which is trained on the collected ground truth data. Section V-A describes more details about extracting the features and building the classifier, and section V-C describes more details about identifying the applications and the flow types.

TV Engine provides on-fly information about active applications and flow types to the upper Application Layer using specified API (Figure 1). We use a standard event model for implementing the API, where number of event listener objects (i.e. control applications) keep listening to all kind of events (i.e. Identification of different types of traffic or application flows) from the event source object of the *TV Engine*. In developing a "network management" service, the service needs to bind to these event listener object to get notify about identifying flow of certain type/application. There are number of popular "network management" services such as policy enforcement and service differentiation, which leverage the traffic-aware and application-aware flow information to apply proper action commands (e.g. packet dropping, traffic throttling, modifying QoS etc.) on different network flows. Typically "network management" send the action commands to the control layer, which is later converted to the flow rules and forwarded to the appropriate network devices through OpenFlow protocol. In section VII, we develop two prototypes of "network management" services, using our *TrafficVision* framework at wireless network edges.

IV. TrafficVision Data Layer: Extending OpenFlow Switch

In extending the *TrafficVision* data layer, we add two new statistics (packet sizes and packet arrival timestamps) to collect from the OVS per flow. Note that, OVS has two forwarding components, one at the kernel space (`datapath`) that is the *fast path*, and another at the user space (`ovs-vswitchd`) that is the *slow path*. The *fast path* maintains a "cache" of the flow entries (i.e. called micro-flow) in the hash-table that recently has been observed by the `datapath`. Therefore, when a packet is first received at the *fast path*, we look for corresponding entries at the hash-table. If the packet's header matches with a flow entry in hash, the packet is forwarded according to the action of the matched flow entry. Otherwise, the packet is sent to the user space (i.e *upcall*) for the *slow path* forwarding. In *slow path*, once the packet is delivered to the user space, its header is matched with "flow table" entries to make the forwarding decision, and then send it to the *fast path*. Then, the *fast path* forwards the packet according to *slow path* decision, and adds a new hash entry corresponding to the matching flow entry of the packet.

In *fast path*, when a packet of a flow matches with a micro-flow entry in hash-table, we update the statistics of `sw_flow` structure with the new packet size and the packet arrival timestamp information in function `ovs_flow_used`. Note that we added two arrays of 80 entries to `sw_flow` structure to store packet arrival timestamps (`arrival_time`) and packet sizes (`packet_size`) respectively. We chose the array size to be 80 entries since this is the maximum amount of information (total packet count, total byte count, last 80 packet sizes and packet arrival timestamps) can be transferred in one packet using OpenFlow protocol. In *fast path*, `sw_flow` structure keeps record of packet sizes and arrival timestamps since the entry is created in the hash-table. The two arrays are maintained in queue form, where the oldest element is removed when the queue is full. Now, if an entry gets removed, for not recently receiving any packets correspond to that flow entry or any periodic request is sent from the user-space, the two arrays get copied from the `sw_flow` structure to `ovs_flow_stats` structure in along with the packet and byte count statistics. Once statistics are copied, values of `sw_flow` structure get reseted. Later, `ovs_flow_stats` structure is sent to the user space to get merged with other `subfacet`, `facet`, `rule_dpif`, and `rule` structures for updating the flow table statistics.

V. TrafficVision Control Layer: *TV Engine*

In *TrafficVision* control layer, we develop a network service, *TV Engine*, which has three major tasks: (1) Collecting, storing and extracting higher-order flow statistics and temporal features from packet sizes and packet arrival timestamps. (2) Building a classifier from the collected ground-truth training data. (3) Finally, applying the classifier to identify application and flow types. The *TV Engine* is also the core network service that provide API to the upper application layer for running policies to control applications in the network. Figure 2 shows an overview of *TV Engine*.

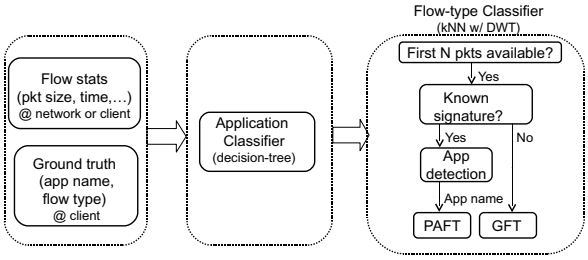


Fig. 2: *TV Engine* workflow.

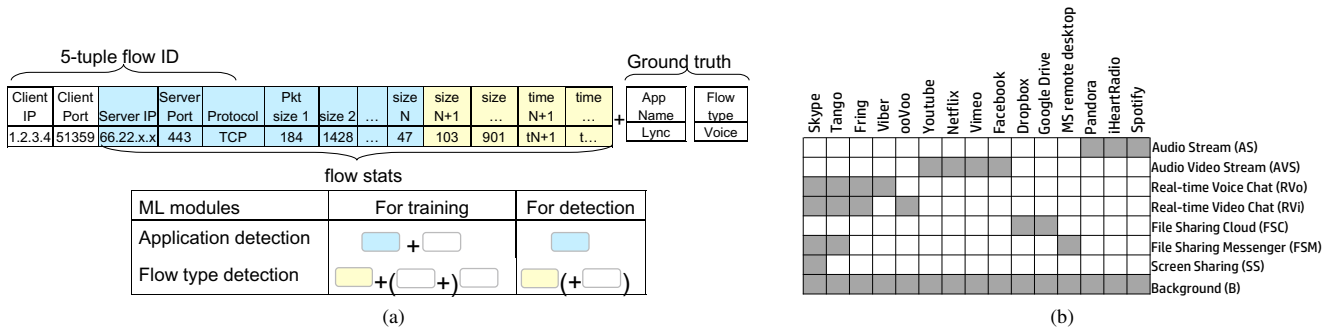
A. Flow Feature Extraction

Any network flow has two directions of packet flows, incoming and outgoing, with respect to the device. We merge the packets from both directions while maintaining the relative order to extract two sets of flow features for each flow.

The first set of features corresponds to the signaling portion of the flow that is consisting of the first 'N' packet sizes as well as server port, server IP, protocol information. These packets typically correspond to session initialization and signaling, and are unique to each application. For example, the sizes of the first 11 packets of Twitter flows are all unique compared to flows of other applications. There could be several unique signatures corresponding to a given application. Hence, we use these features to identify the application corresponding to the flow. However, existing in-network measurement APIs (e.g., OpenFlow protocol, sFlow) do not provide packet size & time information 'per-flow'. Thus, previous ML-based solutions have to use software or hardware packet capture tools (e.g., `pcap`). However, this incurs unnecessary overhead for traffic mirroring, capturing and flow processing; and also inadequate for real-time in-switch/AP processing. In this paper, we extend Open vSwitch and OpenFlow statistic APIs to provide per-flow packet size (and arrival timestamp, for the second set) with only marginal memory overhead to *TrafficVision*.

The second set of features corresponds to the non-signaling (data) portion of the flow, consisting of packet sizes and inter-packet times (IPT) observed after the first N packets. Most existing approaches extract flow statistics either over the entire flow, or over certain number of packets for unlimited amount of time. We collect flow stats over a fixed time window, as short as 200 msec, and determine the flow-type corresponding to each window of the flow. This real-time per-window detection of flow-type is useful for conferencing apps such as Skype that can switch the type of a flow between voice and video (+voice) over time.

In addition to the lower order flow-level statistics (i.e., number of packets, number of bytes, protocol, and mean, median, minimum, maximum, and variance of the packet sizes and IPTs), researchers have also utilized higher order statistics such as the Discrete Fourier Transform (DFT), to further improve the detection accuracy. However, DFT has two main



(a) (b)

Fig. 3: (a) Data requirement for various *TV Engine* modules. (b) Applications and the corresponding flow types in Dataset 2.

time window	low-order features f	f_{DFT}	f_{DWT}
200ms	**75.5 ± 2.6**	**81.5 ± 3.1**	**89 ± 3.8**
2000ms	**83.1 ± 2.3**	**88.0 ± 2.7**	**94.8 ± 2.7**

TABLE I: Flow-type detection accuracy (% F-measure) of 3 feature sets. [mean ± stdev] over 15 apps with 8 flow types.

limitations: 1) It assumes that signals are stationary, i.e., the periodic patterns do not change over time. However, most real world time series signals including the packet size and IPT signals are not stationary. 2) DFT only captures global features thereby loosing information on local properties of the signal. Considering these limitations, in this paper, we introduce Discrete Wavelet Transform (DWT) [7] that is more robust in capturing both the global and the local variations of the time-series data. DWT ($O(N)$) is also faster than DFT ($O(N\log N)$), where N = # of data points in the sequence.

In [35], authors have summarized a list of flow features that have been used for traffic classification. We use these features to create two features sets; f that includes the list of basic features without the DFT coefficients calculated for packet size and IPT sequences, and f_{DFT} that combine f with the DFT coefficients of packet size and IPT sequences. In addition to these two feature sets, we develop another feature set, f_{DWT} that combine f with the DWT coefficients of packet size and IPT sequences. Note that, none of the previous traffic classification techniques have used DWT coefficients as features. Table I compares the flow-type detection accuracy using these three feature sets. Clearly, inclusion of DWT coefficients as features improves the overall accuracy of traffic classification.

B. Ground-Truth Data Collection

Fined-grained, accurate classification is only possible with fine-grained and reliable ground truth data. Existing solutions mostly relies on DPI and/or manual inspections [26], [48], [39] for collecting ground-truth data. However, recent reports [36] shows only 50% detection accuracy using OpenDPI and other opensource tools. Furthermore, Commercial DPI engines still misclassify significant portion of encrypted and HTTPS traffic. DPI requires manual effort to build the application signature, which makes it not scalable with the large growth of mobile applications. Therefore, in this paper, we took the approach of running an "agent" software in user's mobile devices that directly communicate with the SDN controller (e.g. Floodlight) to provide the fine-grained application information. We leverage this agent software to collect standard `netstat` logs, which provides the mapping of every active network socket (TCP, UDP, SSL-encrypted) to the application owning that socket. We use this mapping to label each flow statistics collected from the wireless edges with the corresponding application name. In many enterprises, employees require to deploy device management agent software on their mobile devices. This motivated us to run a software "agent" on selected employee devices, volunteers or dedicated testing device to collect ground-truth data with flow features to train and build classifier at network controller. This crowd-source based scheme allows us to build the classifier in an automatic and efficient way. Note that if a mobile app gets upgraded, it's ML-based classification model also needs to be redone. In that case frequent update of mobile apps make ML-based solution unscalable and challenging. Despite that, our automatic and crowd-sourcing approach simplify the process of updating the app's traffic classification model.

Obtaining flow-type ground truth is more challenging as there is no standard API, similar to `netstat`, that provides clear information of flow type. Therefore, we optionally instrument the software agent to automatically monitor activities of media devices such as microphone, speaker and camera. In addition to this automation, users can also voluntarily provide information about their current activity (i.e. playing a video in YouTube, Skype video chat etc.) using the agent software to the SDN controller. Correlating this activity information with network flow start/stop information from `netstat` is used to infer the ground truth of these flow types.

C. ML Classifier

In *TV Engine*, the classifier has two modules for: 1) identifying application name (i.e., Skype, Tango etc.), and 2) identifying the flow types. In following subsection, we describe each of these modules.

1) Application Detection: We use a supervised learning approach for application detection, where a classification rule

is learned using the labeled training data. Specifically, we use a C5.0 decision tree classifier [2] due to its overall performance (accuracy and speed) [26]. In this decision tree, an internal (non-leaf) node denotes a test on a flow feature, a branch represents the outcome of a test, and a leaf node holds a class label (i.e., application name). The training phase is performed by using the device-crowd-sourcing approach to collect the ground truth in a scalable and accurate manner. The training can be done periodically to automatically update or improvise the classifier. This trained classifier is used to identify the application names of network flows in real-time.

2) Flow-type detection: Figure 2 shows the flow-type detection module consists of two different scenarios (branches) and consequently, requires two different flow-type classifiers models. In this paper, as an example, we use the same k-NN algorithm on the same feature set, different combinations of ground truth and training sets for building the two flow-type classifier models.

In the first scenario, the first N packets of the test flow are captured but is not identified as belonging to a known application. For this case, *TV Engine* uses **aGgregated Flow-Type (GFT)** classifier that identifies only the flow-type. The training data samples for GFT classifier are only labeled with flow types. For example, assuming the training data has no flows corresponding to Tango application, a Tango video chat flow during real-time classification will be detected as a flow corresponding to a new/unseen application and, consequently, be forwarded to the GFT classifier to identify the corresponding flow type (in this case, 'real-time video').

Per-Application Flow-Type (PAFT) classifier, the third classifier, is used in scenarios where the application detector successfully identifies the application name of a flow. In this case, the flow will be forwarded to the PAFT classifier to further identify the flow-type. For each known application, we build a PAFT classifier using training data only from that specific application. As an example, for Skype application that has five flow types (as described later), we build a PAFT classifier with five classes using the training data flows only from the Skype application. During real-time classification, when a flow identified as Skype using the application detector, Skype PAFT classifier is used to identify the corresponding flow-type. Our evaluation shows that PAFT yields higher accuracy over the two other classifiers.

VI. Experimental Evaluation of *TraffficVission*

We evaluated *TraffficVision* using two datasets, described in more detail below.

Dataset #1 was collected from a production enterprise WLAN and used for application detection experiments. The agent has been deployed on eight volunteer Android phones, as well as two dedicated testing phones for manual collection. The manual collection was needed to collect a reasonable sample size for the applications of interest. Over 100K flow samples from 89 different applications were collected, along with the corresponding application names, over 4 weeks period. Out of these set of applications, 37 applications are ranked as the most popular applications in Google Play Store.

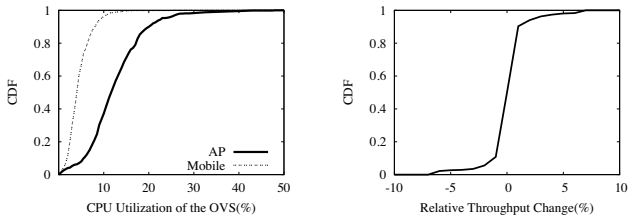


Fig. 4: System Evaluation a) CPU overhead for collecting statistics from OVS for both APs and mobile devices. b) Relative throughput changes while using *TraffficVision*.

Dataset #2 was collected from Old Dominion University campus network and used for flow-type detection module. In terms of user interests [1], we choose 45 applications that are the most popular applications from different application categories (e.g. video stream, video/audio chat, audio stream, social networking etc.). Flows from these applications were categorized into seven major flow types as shown in Figure 3b. For any other flow type that does not fit into any of these seven flow types, we categorize it as a *Background* flow. Each of these applications has at least two different flow types including Background. In total, up to 3 million samples (windows of minimum 200 msec) from 1200 flows with a total duration of over $10,000$ minutes was collected for this data set.

A. System Evaluation

In this subsection, we use *Dataset #2* to evaluate the CPU overhead of both APs and mobile devices as well as the network overhead of *TraffficVision* while collecting flow statistics. Note that, we use Open vSwitch version 1.11 in our implementation, which only uses the micro-flows for caching in the kernel space. In order to evaluate the CPU overhead of the AP, we use one smartphone that runs different types of applications and is associated with our monitoring AP (Linksys E3000) running *TraffficVision* that collect flow statistics. Figure 4a shows the distribution of the CPU overhead for both the AP and the mobile device (Nexus 4 Android) with the average overhead of 12% and %4 respectively. The figure shows that mobile devices has much less CPU overhead compared to APs, though the used AP has very slow processing speed (450MHz) compare to the Nexus smartphone (1.5GHz quad-core Snapdragon S4). Despite the minor increases in CPU overhead, *TraffficVision* has no impact on the network performances as discussed next.

Figure 4b shows the relative change in network throughput while running the *TraffficVision*. The figure shows that our system has no noticeable changes in throughput. Typically, OVS is designed to operate on very high line-speed such as 10Gbps, which is much higher then what we see at the wireless

network edges. Moreover, the flow-statistics collection process in *TrafficVision* runs as a separate thread from the main thread of handling a incoming packet in the kernel space. Therefore, our additional task to store timestamps and packet sizes in the micro-flow hash table of OVS at the kernel space is negligible with no impacts on network throughput. Similar to throughput, we observed that there is no change in RTT values. In future, we are interested in evaluating application QoE for *TrafficVision*.

B. TV Engine - Classifier Models

In this section, we evaluate the accuracy of different classification models of TV Engine using both "Dataset #1" and "Dataset #2".

1) Application detection: Figure 5a shows the application detection accuracy for the most 37 popular applications in our "Datasets #1". This figure shows the average precision and recall values for the 10-fold cross-validation, where applications are ordered in a decreasing order of their size. The figures shows an accuracy over 90% for most applications, with an overall average accuracy of 95.5%. In addition, eight very popular applications (including Microsoft Exchange service, Facebook, and Google+) achieved an accuracy of 100%. These eight applications constituted around 40% of the flows in our datasets. Moreover, 60% of the flows in our datasets were over HTTPS, which demonstrates the advantage of the ML based approach over DPI-based classifiers. Note that the average application detection accuracy does not vary significantly beyond the first 7 packets of the flow.

2) Flow-type detection: We evaluate the two classifiers defined within flow-type detection module described in §V-C2 using Dataset #2 for both f_{DFT} and f_{DWT} feature sets. We use the *weka* [21] machine learning tool to build GFT and PAFT supervised *K-NN* classifiers with $k = 3$, which we found to be optimal.

GFT detection: The *aGgregated Flow-type (GFT)* classifier is used to identify the flow-type corresponding to a flow from a new unknown application, as identified by the application detection module. To evaluate the GFT classifier, for each of the 45 applications, we build a classifier using the training data that contains all flows except for that application. The GFT classifier has eight classes corresponding to the tested flow types. Due to space restriction, we present the confusion matrix for only a subset of the applications as shown in Table II, using the f_{DWT} feature set and a window size of 200 msec. The accuracy of the GFT classifier is high (more than 90%) for most media flow types, except for the 'file sharing' and 'screen sharing' flow types (from Skype and Tango). Interestingly, the file sharing and screen sharing flows are misclassified with audio-video stream flows. We relate this low accuracy to the very similar continuous TCP transfers of these three flow types as well as to the low number of samples in our training data (e.g., among the 15 applications, only Skype and Microsoft Remote Desktop applications have screen sharing flow type). This accuracy increases to 98% when using window size of 2000 msec (not shown in the table).

PAFT detection: The *Per-Application Flow-Type* (PAFT) classifier is used when the application name is already identified using app-detection of Section VI-B1. In this case, we build and use a per-application classifier to identify the flow-type. We build a PAFT classifier for each of the 45 applications using only the flows corresponding to that application. Once again, we present the confusion matrix of PAFT classifier for the subset of applications using the f_{DWT} feature set and a window size of 200 msec in Table II. The table clearly shows that the PAFT classifier outperforms the two other classifiers. PAFT yields more than 90% accuracy for most media flow types, except for the file sharing and screen sharing types that exhibit similar flow patterns (continuous TCP transfers). However, the accuracy increases to 98% using a window size of 2000 msec (not shown).

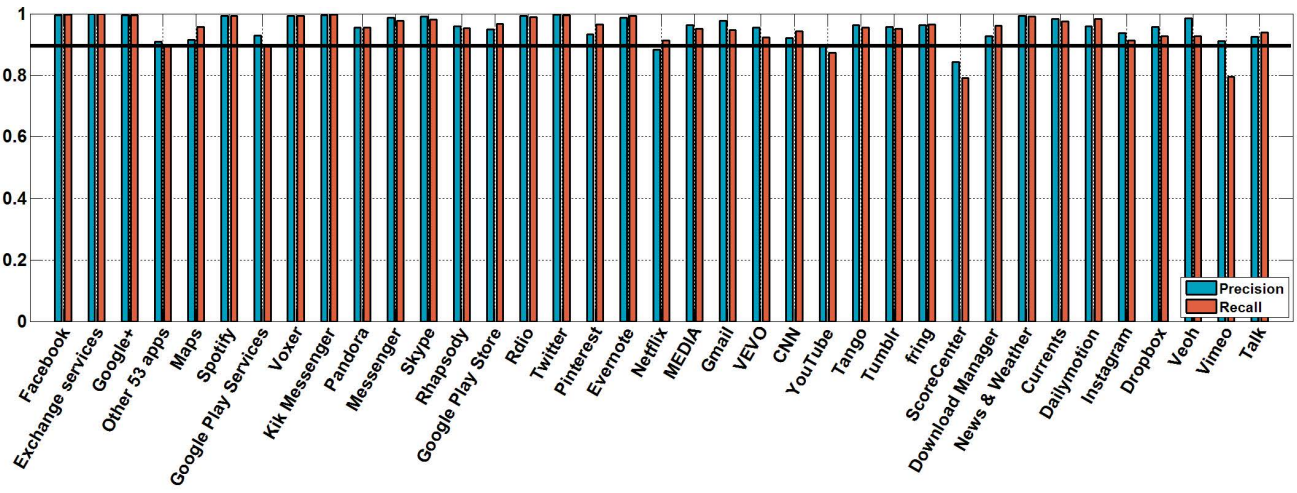
3) Flow detection under adverse network conditions: Given that the network condition can have a serious impact on the values of the extracted features, we evaluate the robustness and the consistency of our classifiers under various adverse network conditions. More specifically, we evaluated our classifiers when the network flows experience packet loss and out-of-order packets.We emulate packet loss and out-of-order packets by dropping packets randomly and buffering packets randomly over short durations. The classifiers are trained with the ground truth collected under normal network conditions, and tested under adverse network conditions. Due to the space restriction, we only discuss the results corresponding to the GFT classifier using a time window of 2000 msec. The average classification accuracy (over all flow types) of the GFT classifier using f_{DWT} as features is 90.3%, 87.8%, 84.5% and 81.9% when 10%, 15%, 20% and 25% of packets are dropped respectively. Similarly, the average accuracy of the GFT classifier using f_{DWT} as features is 91.7%, 88.5%, 85.7% and 83.1% when the percentage of the out-of-order packets are 10%, 15%, 20% and 25%, respectively. We find that these accuracies are dropped by 10-12% when using f_{DFT} as features. We observe similar patterns for other classifiers, which clearly demonstrates the robustness and consistency of *TV Engine* classifiers to realistic network conditions.

VII. A Case Study of Traffic Management

In this section, as a proof of concept, we describe two simple application-aware traffic management policy services that leverage *TrafficVision* framework.

A. Use Case #1

In the first use case, we develop a "network management" service that takes advantage of the deployment of *TrafficVision* for home wireless networks. Assume members of a house uses several smartphones and a tablet to stream movies and other videos from Netflix, YouTube, and other popular video providers. Since the tablet has bigger screen, it is expected that this tablet should utilize higher bandwidth than any individual smartphone in order to maintain a good video quality. Having this requirement, we develop a policy control service to allocate higher bandwidth to the video flows streamed by the



(a)

Fig. 5: Application Detection Accuracy Using the first 7 packet sizes in the flow features

		Pandora		Skype					Youtube		Tango				MRD	
		AS	B	RVo	RVi	FSM	SS	B	AVS	B	Rvo	RVi	FSM	B	SS	B
Audio Stream	GFT	**86.9**	2.6	0	0	5	4.7	0	5.5	0	0	0	0	0	0	0
(AS)	PAFT	**97.3**	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0
Audio Video Stream	GFT	8.7	0	0	0	0	24.1	0	**89.1**	0	0	0	30.0	0	0	0
(AVS)	PAFT	0	0	0	0	0	0	0	**95.6**	1.3	0	0	0	0	0	0
Real-time Voice Chat	GFT	0	0	**94**	0	3	3.4	0	0	0	**95.4**	3.9	0	0	0	0
(RVo)	PAFT	0	0	**94.6**	0	0	0	0	0	0	**93.5**	2	0	0	0	0
Real-time Video Chat	GFT	0	0	0	**90.3**	15	0	0	0	0	2.6	**92.1**	0	0	0	0
(RVi)	PAFT	0	0	0	**95.7**	4	3	0	0	0	3.5	**94.8**	11.3	0	0	0
File Sharing - Cloud	GFT	2.4	5.8	0	5.8	0	0	7.9	0	6	0	0	38.7	10.3	7.5	1.8
(FSC)	PAFT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
File Sharing - Messenger	GFT	0	0	0	3.9	**37.5**	0	2	0	4	2	5	**27.8**	3.2	5.2	8.2
(FSM)	PAFT	0	0	0	0	**70**	4.3	4.2	0	0	0	2	**78.7**	11.8	0	0
Screen Sharing	GFT	0	0	3.5	0	39.5	**67.8**	0	5.4	0	0	0	3.5	0	**87.3**	0
(SS)	PAFT	0	0	2.3	4.3	26	**92.7**	3.5	0	0	0	0	0	0	**98.5**	1.4
Background	GFT	2	**90.6**	2.5	0	15	0	**90.1**	0	**94.0**	0	0	0	**86.5**	0	**90**
(B)	PAFT	2.3	**99.1**	3.1	0	0	0	**92.3**	4.4	**98.7**	3	1.2	10	**88.2**	1.5	**98.6**

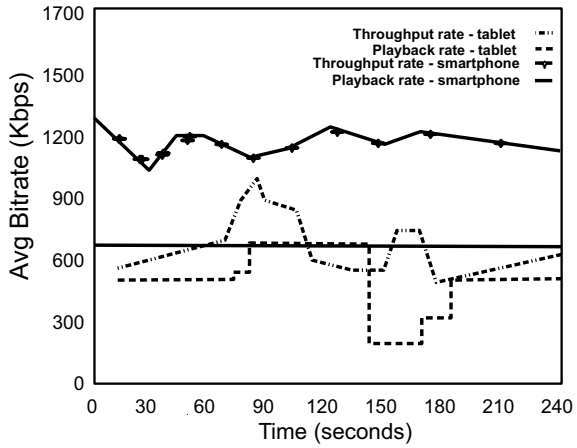
TABLE II: GFT and PAFT classifier accuracy using f_{DWT} as the feature set and a window size of 200 msec.

tablet to guarantee its quality. This service allocates higher bandwidth to the tablet by throttling the traffic of the other devices (smartphones). Consequently, the video player on the tablet would experience a higher throughput and thus would request a higher quality profile that suites the tablet screen. Intuitively, this device-based traffic throttling policy is only activated when there is a video stream initiated by the tablet and automatically deactivated as soon as the tablet video player is turned off or when it downloads all the video chunks.

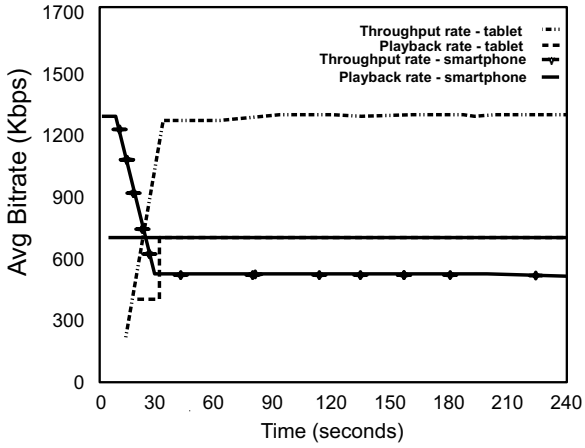
In the prototype setup, we used one android phone and one tablet running YouTube app on both devices to stream and watch the same video. The selected video is encoded with different five bit rates from 245kbps to 730kbps corresponding to the resolutions of 144p to 480p respectively. Initially, we start streaming the video on the smartphone first and then after a couple of seconds, we start playing the video on the tablet. We measure the throughput and the playback rates on both devices when *TrafficVision* is both active and not active. As shown in Figure 6a when *TrafficVision* is not active, the tablet video player not only suffers from poor viewing quality, but also a significant instability in video quality. Figure 6b, on the other hand, shows the performance when *TrafficVision* is active and applying the policy control. It is obvious from the figure that the video player on the tablet achieves better viewing quality (730kbps/480p) in addition to better stability due to the bandwidth reallocation enforced by *TrafficVision*. This significant improvement indicates the necessity of deploying *TrafficVision* at the wireless network edges to enforce network fine-grained policy control.

B. Use Case #2

In this use case, we consider an enterprise WLAN setting scenario of a company in which it prohibits the use of any video chat application except its proprietary video chat application during the working hours. Hence, traffic of any commercial video chat application such as Skype video chat needs to avoid the enterprise WLAN and gets offloaded to cellular during the working hours. Given this scenario, we develop a "network management" service that enforce employee devices to run Skype Video chat traffic over cellular interface during working hours. On the other hand, off the working hours, the management service permits Skype Video



(a) *TrafficVision* is not active



(b) *TrafficVision* is active

Fig. 6: Throughputs and video rates of a smartphone and a tablet when: a) *TrafficVision* is not active, and b) *TrafficVision* is active.

chat traffic to run over the company's enterprise WLAN. Note that this policy of offloading is only applied on Skype video chat flows but not Skype voice flows. Hence, Skype voice chat flows are allowed to use the company's WLAN at any time of the day. In this enterprise scenario, we assume that employee devices are running company's agent software that receives the policy command directly/indirectly from our traffic management policy service that is running either on WLAN access points or on WLAN controller.

In the prototype setup, we used 8 android phones and one laptop playing an access point role. We run our traffic management policy service and *TrafficVision* on the laptop. On the phones, we run an "agent" software that receives direct command from the "management service". In the evaluation, we run Skype video chat on one android smartphone and Skype voice chat on another smartphone. The rest of the smartphones generate background traffic using `iperf`. On the laptop, the "management service" sets a policy to offload the

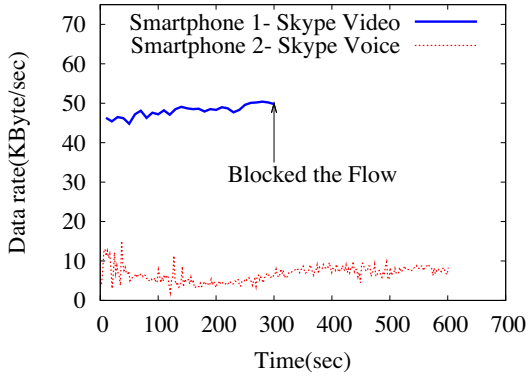


Fig. 7: Traffic Management for Skype flows.

Skype video chat flows to cellular interface during specific time frame (working hours). Figure 7 shows how a Skype video chat traffic flow has been offloaded to the cellular interface when the policy is activated after about 5 minutes, while the Skype voice chat traffic running on the another smartphone remains using the Wi-Fi interface. In addition, the figure shows how the data rate drops when Skype video chat flow is offloaded to cellular, which is HSPA+. These two prototype services demonstrate the potentiality of the real-time, and more fine-grained policy making capability of *TrafficVision* framework.

VIII. Conclusion and Future Work

In this paper, we push and extend the SDN framework all the way to wireless network edges to have fine-grained and real-time traffic classification solution, called *TrafficVision*. In *TrafficVision*, we use ML-based approach for classifying both mobile applications and their flow types in real-time. In this paper, we also describe multiple use-case scenarios based on *TrafficVision*. Later, in the paper, we evaluate our ML-classifier by accurately classifying the 40 top most popular mobile applications. We observe that in working places, most user use a limited number of applications and most enterprises care the most of the "top" applications running on their network. Therefore, for a specific enterprise, our system can be easily configured and used to provide better fine-grained classification performance for their top N most common applications. Though we showed higher applications detection accuracy for as many (or more) applications as compared to previous work [26], [39], [35], [10], [17], we will continue this research and cover wider range of applications and devices/OS platforms.

References

[1] 25 most popular mobile apps. http://tinyurl.com/kds8y7t.
[2] C5.0 open source tool. http://www.rulequest.com/see5-info.html.
[3] Open vSwitch. http://openvswitch.org/.
[4] Android leads in u.s. smartphone market share and data usage., May 2011. http://blog.nielsen.com/nielsenwire/consumer/android-leads-u-s-in-smartphone-marketshare-and-data-usage/.
[5] Mike Coward. Encryption: will it be the death of DPI? http://tinyurl.com/oeuklk5.

[6] Safa Alkateb. 5 Things You Need to Know About Deep Packet Inspection (DPI), 2011. http://tinyurl.com/kqpc4uh.

[7] I. Batal and M. Hauskrecht. A Supervised Time Series Feature Extraction Technique Using DCT and DWT. IEEE ICMLA, 2009.

[8] M. T. Beck, M. Werner, S. Feld, and S. Schimper. Mobile edge computing: A taxonomy. In *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014.

[9] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral. Deep packet inspection as a service. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 271–282. ACM, 2014.

[10] T. Bujlow et al. Classification of HTTP traffic based on C5.0 machine learning algorithm. IEEE ISCC, 2012.

[11] A. Callado et al. A Survey on Internet Traffic Identification. *IEEE Commun. Surveys Tuts.*, 2009.

[12] V. Carela-Espanol, P. Barlet-Ros, and J. Solé-Pareta. Traffic classification with sampled netflow. *traffic*, 33:34, 2009.

[13] Y. Choi, J. Y. Chung, B. Park, and J.-K. Hong. Automated classifier generation for application-level mobile traffic identification. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 1075–1081. IEEE, 2012.

[14] A. Dainotti et al. Issues and future directions in traffic classification. *Netwrk. Mag. of Global Internetwkg.*

[15] S. Deng, A. Sivaraman, and H. Balakrishnan. All Your Network Are Belong to Us: A Transport Framework for Mobile Network Selection. In *ACM HotMobile*, 2014.

[16] C. Directive Outsource IT Chase. The internet of things as the next big thing., June 2013. http://www.directive.com/blog/item/the-internet-of-things-as-the-next-big-thing.html.

[17] J. Erman et al. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 2007.

[18] A. D. Ferguson et al. Participatory networking: An api for application control of sdns. SIGCOMM '13. ACM, 2013.

[19] G. Finnie. Dpi and traffic analysis in networks based on nfv and sdn. 2014.

[20] Graham Finnie. DPI and Traffic Analysis in Networks Based on NFV and SDN, 2014. http://tinyurl.com/q2btt5w.

[21] M. Hall et al. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explor. Newsl.*, 2009.

[22] C. IOS. Netflow white papers., 2006. http://www.cisco.com/en/US/products/ps6601/prodwhitepaperslist.html.

[23] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang. Lightweight application classification for network management. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, pages 299–304. ACM, 2007.

[24] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 163–174. ACM, 2013.

[25] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[26] H. Kim et al. Internet traffic classification demystified: myths, caveats, and the best practices. ACM CoNEXT, 2008.

[27] W. Kim et al. Automated and scalable QoS control for network convergence. USENIX INM/WREN, 2010.

[28] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[29] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

[30] J. Lee et al. meSDN: Mobile Extension of SDN. ACM MCS (in conjunction with MobiSys), 2014.

[31] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem. mesdn: mobile extension of sdn. In *Proceedings of the fifth international workshop on Mobile cloud computing & services*, pages 7–14. ACM, 2014.

[32] Y. Liao, S. Miskovic, G. Lee, and M. Baldi. Appprint: Automatic fingerprinting of mobile apps in network traffic. 2015.

[33] J. R. Ltd. Wi-fi calling operators., June 2015. http://www.juniperresearch.com/documentlibrary/white-papers/wifi-calling-operators?utm_source=gorkanapr&utm_medium=email&utm_campaign=dataoffload15pr2.

[34] N. McKeown et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*

[35] T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 2008.

[36] B. Park et al. Fine-grained trafc classication based on functional separation. *Int. J. Network Mgmt*, 2013.

[37] B. Park, J. W.-K. Hong, and Y. J. Won. Toward fine-grained traffic classification. *Communications Magazine, IEEE*, 49(7):104–111, 2011.

[38] B. Park, Y. Won, J. Chung, M.-s. Kim, and J. W.-K. Hong. Fine-grained traffic classification based on functional separation. *International Journal of Network Management*, 23(5):350–381, 2013.

[39] G. Piraisoody et al. Classification of applications in HTTP tunnels. IEEE CloudNet, 2013.

[40] Z. Qazi et al. Application-awareness in SDN. ACM SIGCOMM, 2013.

[41] D. Rossi and S. Valenti. Fine-grained traffic classification with netflow data. In *Proceedings of the 6th international wireless communications and mobile computing conference*, pages 479–483. ACM, 2010.

[42] K. Sabnani. Future communication clouds. In *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*, pages 32–32. ACM, 2015.

[43] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab. Edge computing enabling the internet of things. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 603–608. IEEE, 2015.

[44] A. Tongaonkar, R. Keralapura, and A. Nucci. Santaclass: A self adaptive network traffic classification system. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[45] UCI Forum. UC SDN Use Case – Automating QoS. http://tinyurl.com/lch8qy7.

[46] L. M. Vaquero and L. Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[47] Y. Wang et al. Machine Learned Real-Time Traffic Classifiers. *IEEE Intelligent Information Technology Applications*, 2008.

[48] N. Williams and S. Zander. Real time traffic classification and prioritisation on a home router using DIFFUSE. Technical report, Swinburne University of Technology, Australia, 2011.

[49] S. WIRED Amyx. Why the internet of things will disrupt everything., July 2014. http://innovationinsights.wired.com/insights/2014/07/internet-things-will-disrupt-everything/.

[50] K.-K. Yap et al. Making use of all the networks around us: a case study in android. ACM CellNet, 2012.

[51] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog computing: Platform and applications. In *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*, pages 73–78. IEEE, 2015.

[52] S. Yi, C. Li, and Q. Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM, 2015.

[53] S. Yi, Z. Qin, and Q. Li. Security and privacy issues of fog computing: A survey. In *Wireless Algorithms, Systems, and Applications*, pages 685–695. Springer, 2015.

[54] S.-H. Yoon, J.-S. Park, and M.-S. Kim. Behavior signature for fine-grained traffic identification. *Appl. Math*, 9(2L):523–534, 2015.

[55] M. Zhang et al. State of the art in traffic classification: A research review. In *PAM '09: 10th International Conference on Passive and Active Measurement, Student Workshop*, 2009.