iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification

Hassan Habibi Gharakheili, Minzhao Lyu, Yu Wang, Himal Kumar, and Vijay Sivaraman

***Abstract*—Video continues to dominate network traffic, yet operators today have poor visibility into the number, duration, and resolutions of the video streams traversing their domain. Current monitoring approaches are inaccurate, expensive, or unscalable, as they rely on statistical sampling, middle-box hardware, or packet inspection software. We present *iTelescope*, the first intelligent, inexpensive, and scalable softwarized network middle-box solution for identifying and classifying video flows in real-time. Our solution is novel in combining dynamic flow rules with telemetry and machine learning, and is built on commodity OpenFlow switches and open-source software. We develop a fully functional system, train it in the lab using multiple machine learning algorithms, and validate its performance to show over 95% accuracy in identifying and classifying video streams from many providers including Youtube and Netflix. Lastly, we conduct tests to demonstrate its scalability to tens of thousands of concurrent streams, and deploy it live on a campus network serving several hundred real users. Our traffic monitoring system gives unprecedented fine-grained real-time visibility of video streaming performance to operators of enterprise and carrier networks at very low cost.**

***Index Terms*—Softwarized Networks, Telemetry, Classification, Video Traffic.**

I. Introduction

VIDEO constitutes a majority of Internet traffic today, and is slated to increase even further in the near future, as higher resolutions (1440p and 4K) become more prevalent, and augmented/virtual reality (AR/VR) starts to take off [8]. In order to manage this video traffic (for quality and cost reasons), enterprises and carriers need better visibility into the video streams in their network. Operators can today infer macroscopic attributes (such as aggregate volume of video traffic on their peering link with a video content provider like Netflix), but they have near-zero visibility into the micro-scopic aspects, such as how many video streams are concurrently active at a time, what their durations are, what resolutions they operate at, and how often they adapt their rate. Visibility into these attributes can allow them to better understand both video content characteristics and video viewing patterns, so they can tune their network to meet content-provider expectations as well as enhance user experience.

Several existing methods can be used for visibility into video streams, but they come with disadvantages: SNMP [27] can be used to retrieve traffic counts from switches, but these counters are at the interface-level, and may represent an aggregate of many video and non-video flows. NetFlow [16] enables a switch to aggregate IP flow information in a local cache and export this information periodically – this requires the switch hardware to be capable of decoding, collating and caching entries, and can also entail a penalty in switch CPU utilization in the range of 7-22% [1]. sFlow [42] reduces this overhead by statistically sampling traffic; however, lower sampling rates inevitably lead to reduced accuracy in traffic characterization [43] (explained in §III-F). Specialized traffic monitoring solutions can provide both accuracy and performance – for example deep packet inspection "middle-boxes" (*e.g.,* Sandvine) can inspect data packets at high rates; however, such solutions cost hundreds of thousands of dollars that is prohibitive for many network operators.

The general problem of classifying network traffic has been studied by many prior research works [9], [46], [14], [41], [37], [38], [21], using various methods ranging from inspecting a few bytes in the payload, to processing headers or characterizing the signatures of packets streams, which may become unreliable with the adoption of tunnel and payload encryption. Our paper focuses more narrowly on streaming video flows, and the general methods developed earlier do not directly apply, as they are either reliant on tunnel or packet payloads being visible (video traffic is increasingly encrypted) or require long trains of packets to be analyzed in software (limiting scalability). Further, they do not determine aspects specific to video streams, such as rates and resolutions. We believe that the Software Defined Networking (SDN) paradigm is ideally suited to the task of identifying and classifying video traffic, since Openflow by its nature provides flow-level isolation and visibility in a low-cost and scalable manner.

Video streaming over the Internet has evolved over the past two decades [31]. Currently, dynamic adaptive streaming over HTTP (DASH) is the dominant technique of video delivery, used by major providers such as Youtube and Netflix, that breaks the video into a sequence of small chunks, each chunk containing a short interval of playback time of video. The video content is made available at a variety of different resolutions (*i.e.,* bit rates) – the higher the video resolution, the larger the chunk size (in Bytes). While the video is being played back, the client automatically selects from the alternatives of next chunk to download and plays based on current network conditions [32], providing high quality playback with minimal stalls [25].

While SDN-based flow-level monitoring for video streams

Email addresses: H. Habibi Gharakheili, M. Lyu, H. Kumar, and V. Sivaraman are with the School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, NSW 2052, Australia (e-mails: h.habibi@unsw.edu.au, minzhao.lyu@unsw.edu.au, yu.wang1@unswalumni.com, himal.kumar@unsw.edu.au, vijay@unsw.edu.au).

This article is an extended and improved version of our paper presented at the EWSDN 2016 conference [48].

may seem conceptually simple, there are several challenges to be overcome: *correctness* of the solution requires dealing with an arbitrary set of content providers and dynamic video end-points; carrier-grade *performance* requires the controller to be protected against packet overload and the solution to be resilient to controller failures; and high *scalability* requires minimizing the software inspection of packets as well as flow-modifications on the switch hardware. In this paper we develop, deploy, and evaluate our SDN-based solution called *iTeleScope* that meets these challenges to provide fine-grained visibility into streaming video flows. Our **first** contribution is to develop a system architecture, comprising selective packet inspection, dynamic flow-table management, and flow traffic profile analytics, that is intelligently able to identify and classify long video streams. We show how our design meets our goals of low cost (by using commodity Openflow switches), scalability (by filtering packets to minimize software processing), and high accuracy (via machine learning methods that use key attributes related to flow traffic profiles). Our **second** contribution develops a fully-functional prototype based on the above architecture using commodity hardware and software: a NoviFlow Openflow switch, the Ryu SDN controller, the Bro packet inspection engine, and the Weka machine learning suite. We show how we train and tune our classifier in the lab, and validate its performance to obtain over 95% accuracy in identifying video streams and deducing their resolution, typically within 60-90 seconds, even when such traffic comes from shared server pools that serve many types of content (such as done by Google). **Finally**, we demonstrate the scalability of our system to tens of thousands of concurrent streams generated from a hardware tester, and do a field-deployment in a University dorm network serving hundreds of students, yielding new insights into video viewing patterns and quality measures for the University residence network.

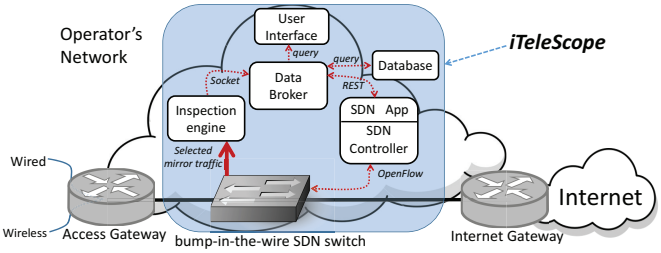
The rest of this paper is organized as follows: §II describes prior work on network monitoring solutions, and §III describes our solution approach that captures and evaluates flow-level information. In §IV we describe our prototype implementation used to validate our solution, while in §V we evaluate the scalability and efficacy of our system. The paper is concluded in §VI.

II. Related Work

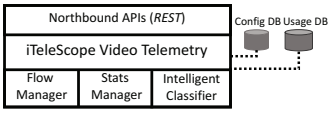
Traffic classification: This has been a broad-ranging area of interest to the research community for well over a decade, aiming to distinguish mice from elephants, peer-to-peer from downloads, and over-the-top voice/video from streaming applications. Several surveys of this area have been conducted [9], [46], [14] and reveal existing classification techniques to have different trade-offs in terms of their *accuracy, computing cost, and scalability*. Among widely used approaches: (a) TCP/UDP port-based classifiers have become less reliable since modern sophisticated applications use non-standard or random port numbers; (b) payload inspectors come at a high cost of processing and are increasingly being defeated due to encrypted content [13]; (c) statistical and behavioral classifiers are attractive as they are fairly light-weight, employing flow-level information and machine learning algorithms to identify various traffic types. It has been shown [22] that flow-level analysis (*i.e.,* based on NetFlow and IPFIX) is more scalable than packet-level analysis, and combination of packet inspection and flow monitoring achieves more accurate results. Indeed, traffic modeling has been studied extensively over the past two decades [20], [23], [39], and many researchers have worked to develop models ranging from fractal-based [40], [52] to structural models [36], [33] for describing the behavior of Internet traffic in short timescales. Inspired by prior work [28], we use flow attributes computed at multiple timescales as input to our classifiers (§III-E). Also, there exist commercial middle-boxes which are either purely software-based (i.e., Virtual Network Functions) [5], [11], [18] or custom-hardware-based [4], [6]. Both types of existing middle-boxes employ deep packet inspection, but their effectiveness may reduce as more Internet traffic becomes end-to-end encrypted. We note that existing appliances generally perform classifications for a broader range of traffic types [14], [45], and hence may not accurately characterize video flows. Further, we note that fully virtual (software-based) solutions are difficult to scale while custom-built hardware solutions are very expensive.

Authors of [29] proposed a sampling technique for classifying TCP flows that only zero-payload packets are extracted by network functions to address the scalability concern of software-based inspection. Work in [24] developed an algorithm to enhance the performance of rule matching inside SDN hardware switches using multi-dimensional lookups. Moreover, there exist a body of work on custom sketch-based algorithms [49], [51] for monitoring network flows. However, these methods require specialized data structure and hardware design which hinders the practical adoption of such approaches. Additionally, sketching algorithms typically demand additional compute resources on network switches [34] (concern of scalability). Some [43] have proposed to measure network activity of "top-k" heavy hitters (only a limited number of elephant flows) by running a customized algorithm in data-plane. Our objective, instead, is to monitor all potential elephant flows (videos and non-videos) with generic flow-level telemetry given limited resources available on network switches. Our iTelescope system leverages the best of both worlds, by keeping the state management and intelligent processing in the software (at low load) while pushing repetitive tasks for long flows into the SDN white-box hardware (at high speeds). This dynamic split lets our solution achieve unprecedented scale. Unlike much of the prior work that tries to classify traffic by application type, our focus in this paper is specifically on streaming video for reasons stated earlier. As we will see soon, our solution uses a combination of packet filtering based on header information, flow-level telemetry, and behavioral pattern recognition to detect and classify streaming video flows in real-time.

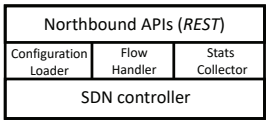
SDN-based monitoring: Several proposals for flow-based measurement and monitoring for SDN have been developed in the literature [17], [51], [15], [50], [47]. OpenSketch [51] proposes a clean slate redesign of the data-plane to support monitoring; unfortunately it requires an upgrade to the data-plane, which can be a barrier to uptake. DevoFlow [17]



(a) iTeleScope system



(b) Data Broker



(c) SDN application

Fig. 1: iTeleScope (a) system, (b) data broker, and (c) SDN application, architecture.

highlights the performance limits of OpenFlow when scaling to extract counters for all flow entries. We partially avoid these problems by inserting reactive entries for only a small fraction of flows, namely flows that transfer a large volume of traffic. Examples of standard OpenFlow based approaches for traffic monitoring include PayLess [15], FlowSense [50], Sample&Pick [12] and OpenNetMon [47], which insert rules and collect per-flow counters in response to standard `PacketIn` and `FlowRemoved` messages respectively. These approaches try to balance the accuracy of flow-level statistics against the cost of control-plane overhead, by adjusting the time-out attribute of rules for example based on their counters (a large byte-count shortens the time-out). However, we believe that these reactive interactions between the controller and the switch (*i.e.,* `PacketIn` and `FlowRemoved`) can impose a heavy processing load on the controller, and suffer from disruptions in data-plane operation if there is a control-plane failure. As we will explain later, our architecture does not use any `PacketIn` messages, minimizing the use of controller resources, while also being robust to failures.

The work in [26] proposes an interactive user interface that uses SDN to monitor and visualize the network state, including traffic rates and rules within each switch. The network administrator can adjust the time-outs of rules as well as the frequency of statistics collection. Our solution also provides an intuitive web-based user interface, though it is specifically for visualizing video flows, and it manages flow-table entries automatically without operator involvement. The work in [30] conducts empirical studies to show that flow-level counters in OpenFlow switches may have significant inaccuracies; however, we have confirmed in our experimental work (§V-A) that our NoviFlow switches are accurate to within 1.7% in terms of their flow byte-counts.

The current work builds upon our earlier proposal for flow-based video telemetry [48]. However, our earlier work relied only on average bitrate, with threshold detectors statically configured for selected video content providers. The current work extends it by extracting a richer set of attributes (such as idle-fraction and burstiness at various time-scales) and using machine learning to automatically identify and classify video flows. Further, we expand our system to incorporate DNS inspection to identify a larger set of content providers, and validated its performance in a live network serving several hundred users.

III. System Design and Architecture

In this section we describe our *iTelescope* solution, including the major architectural decisions (§III-A), the functional blocks (§III-B), flow-table management (§III-C), packet inspection (§III-D), telemetry collection (§III-E), and the classification algorithm (§III-F).

A. Architectural Decisions

Our solution is designed to be a "bump-in-the-wire" on the link at which video classification is desired (an alternative approach is to feed our system a mirror of all organizational traffic, though this precludes active traffic management at a later date). Our system is therefore transparent to the network, and does not modify packets in any way. Further, our SDN switch does not send any data packets to the controller; instead, packets that need to be inspected in software are sent as copies on a separate interface of the switch, to which a software inspection engine is attached. This protects the controller from overload from the data-plane, allowing it to scale to high rates and to service other SDN applications. Moreover, since incoming data packets are sent onwards by the switch immediately, the data-plane benefits in having minimal latency overhead, and is protected from controller failures.

Our second architectural decision is in the judicious combination of packet-level and flow-level monitoring. In essence, we use the Openflow switch as a hardware filter to limit the fraction of traffic (to the first few MB of each flow) that is mirrored for software inspection; heavier (elephant) flows are suppressed from software mirroring by inserting reactive flow-table entries, and are monitored by periodically polling their flow-level counters. This approach is tuned to balance the load between the software (for packet inspection) and the hardware (for flow table size, modification rate, and counter polling). We undertook an experimental investigation of this trade-off using a 24-hour feed of our University campus traffic (our empirical results are outlined in [35]), and found that a volume threshold of 4 MB is suitable for declaring a flow as an elephant and creating a reactive flow-table entry for it. With this threshold, only a small fraction of flows were elephants. We required less than 5000 entries in the hardware flow-table at any time and less than 20 flow-mods per-second. Also, no more than 25-30% of packet traffic (corresponding to mice flows) was sent to the software inspection engine. This is because according to our empirical study in [35], 70-75% of the traffic was carried in elephant flows and thus bypasses software inspection. As we will demonstrate later, this balance between hardware and software processing reduces cost and increases scalability, while enabling extraction of attributes for machine learning-based classification with high accuracy.

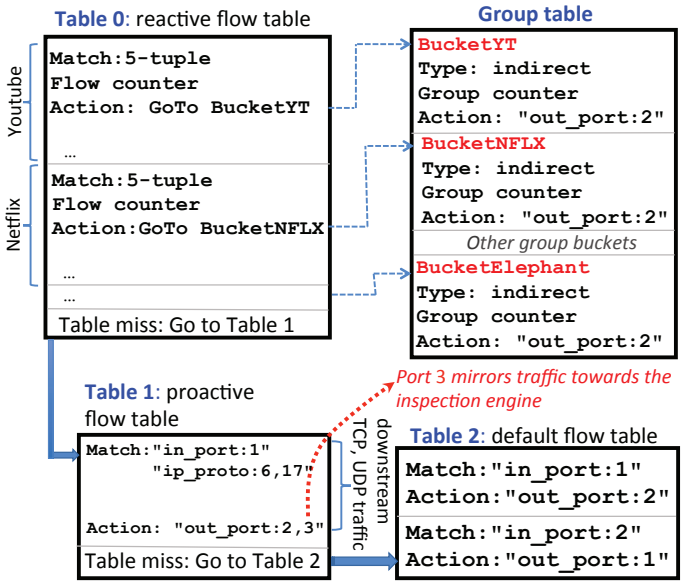


Fig. 2: Flow table structure.

B. Functional Blocks

Fig. 1(a) shows the functional blocks in the iTeleScope architecture applied to a typical carrier or enterprise network. End-users are on the left, and can be on an access network using wired (DSL, Ethernet, Fiber) and/or wireless (*e.g.,* 3G/4G, WiFi) technology. The video content providers are on the right, connected to the carrier/enterprise network through an Internet gateway. Our iTelescope solution can be applied on any desired link as a bump-in-the-wire. It comprises an SDN switch whose flow-table rules will be managed dynamically (explained in §III-C), a packet inspection engine (described in §III-D), and a data broker in conjunction with our App on the SDN controller (internal modules shown in Figures 1(b) and 1(c)) that collect telemetry (§III-E) and run the classification algorithms (§III-F).

The operational flow of events is as follows: assume that video traffic enters (from the content provider) on port-1 and exits (towards the consumer) on port-2; the switch is initially configured to mirror all traffic to the inspection engine on port-3. The inspection engine keeps track of flow volume, and once it exceeds the threshold of 4 MB, notifies the data broker. The broker then instructs our SDN App to insert a reactive flow-entry for the specific stream, which stops the mirroring of packets for this stream. Thereafter, the data broker polls the counters (via SDN App) periodically and develops a traffic profile for the stream, which is fed to the machine learning algorithm for classification. Flow entries for a stream are automatically aged out upon inactivity. Further, our inspection engine has a specific event handler that captures DNS A-type replies, and extracts the server name and IP address so the content provider for the video stream can be identified. In what follows we describe each of the components in more detail.

C. Flow Table Management

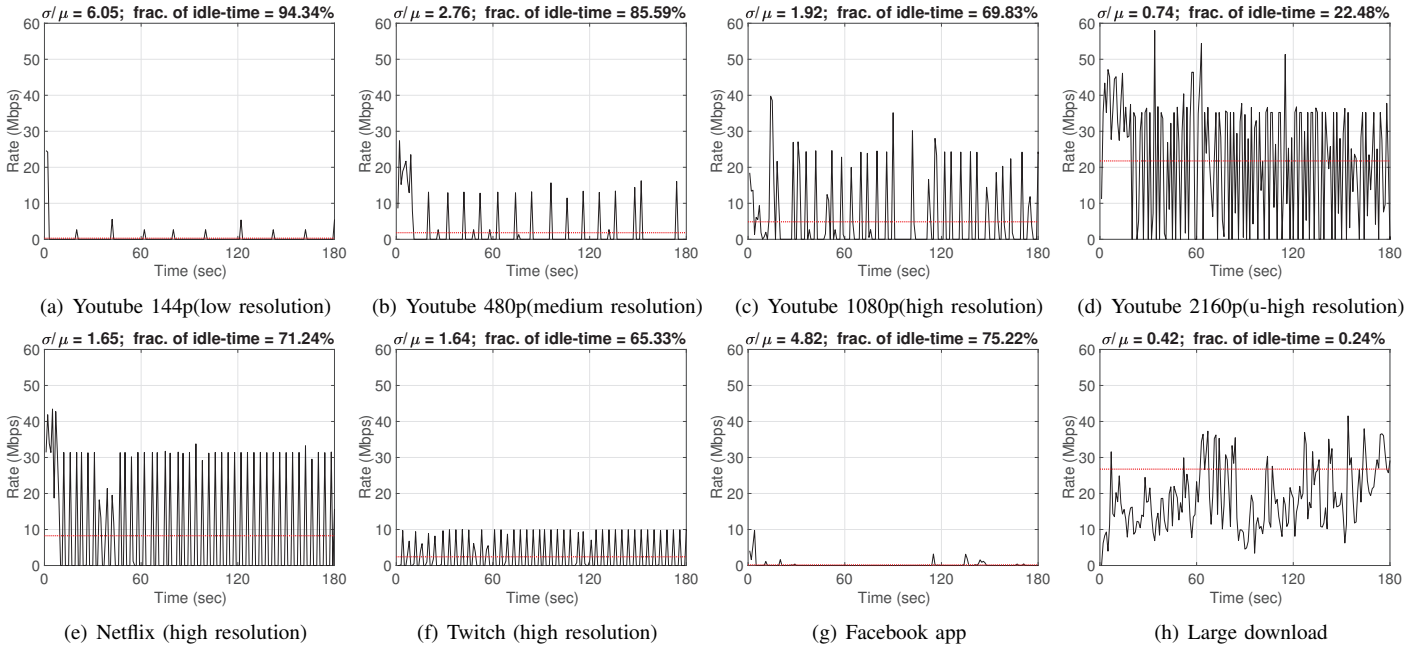
We use a combination of proactive and reactive entries in a multi-table pipeline of the SDN switch, as shown in Fig. 2. Reactive rules match on the 5-tuple, are of highest priority (Table 0), and are installed as a consequence of elephant flows detected by the packet inspection engine. They automatically time out upon a minute of inactivity, so as to reduce TCAM usage. The reactive flow entries achieve two objectives: to stop mirroring of long-flow packets to the software inspection engine, and to provide flow-level telemetry for the individual (potentially video) long-flows. The action corresponding to a match in the reactive table sends the flow to its appropriate entry in the *group table*, which identifies the content provider (Youtube, Netflix, etc.). The content provider for the flow is identified by searching for the server IP address in the most recent captured DNS suffixes (*e.g.,* `googlevideo.com` or `nflxvideo.com`) that are stored in a time-series database by the software inspection engine. We note that if a video stream from a new DNS suffix is detected (*e.g.,* `ttnvw.net`) then a new group entry (for Twitch in this example) will be created dynamically. Our design not only makes the system adaptive to new video content providers, but also allows us to track aggregate video volumes for each video content provider.

Proactive (Table 1) entries are statically pushed by the controller so that all TCP (proto=6) and UDP (proto=17) packets received from the content provider, that have not already matched an elephant flow (Table 0), are forwarded (on port 2) and mirrored (on port 3) to the software inspection engine. Note that this includes DNS reply packets that contain the domain name of the video content provider and the video server IP address. All other packets are sent to Table 2, where the default action is to cross-connect the input and output ports (without any mirroring). We again emphasize that no data packets are sent to the controller, minimizing controller load, reducing packet-forwarding latency, and immunizing against controller failures.

D. Packet Inspection Engine

The packet inspection engine keeps track of new flows, including 5-tuple information, duration, and volume, using efficient hash-table based data structure with 5-tuple information as the hash key. The engine maintains states in our optimized hash-table ordered in time. Our hash table data structure is a combination of key, value pairs where values are doubly linked (using pointers) and arranged in time to ease timeout and deletion. If a flow is active for more than a threshold volume, it is deemed as a elephant flow, and the engine informs the Broker which then makes a RESTful API call to the SDN controller to insert the reactive flow-table entry into the switch. This suppresses data-plane traffic for this flow from being mirrored to the inspection engine (as described in §III-C), and also triggers telemetry for that flow, as described in §III-E.

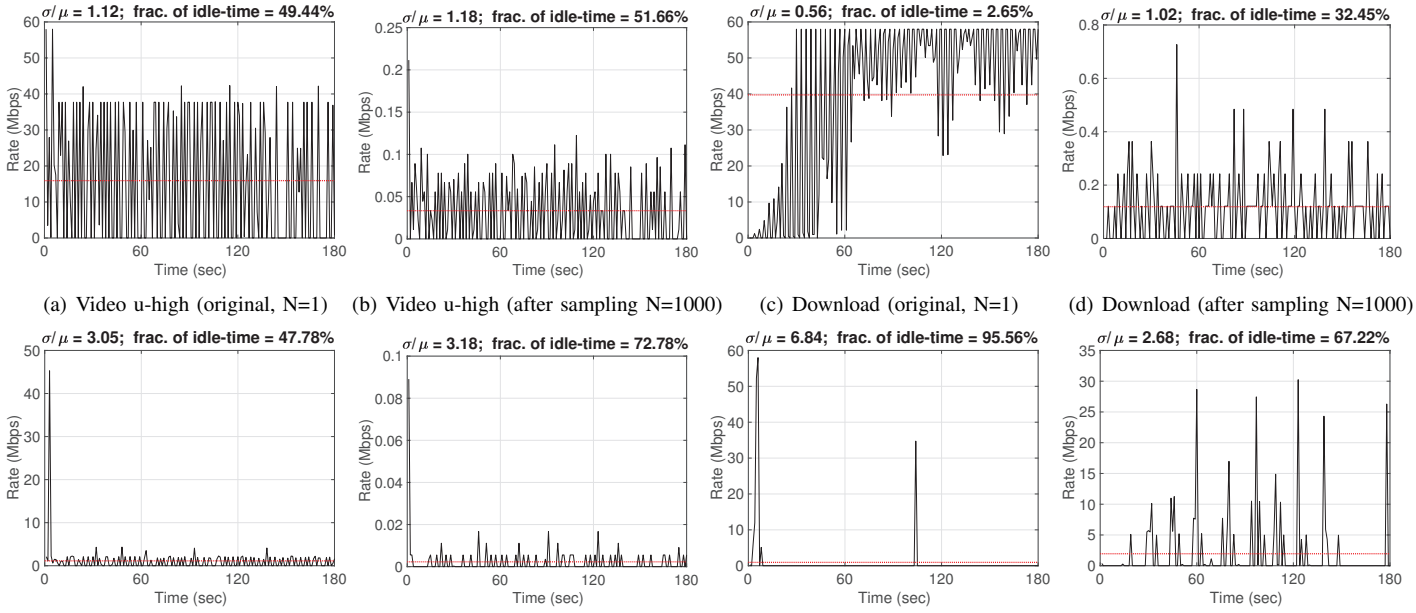
The other responsibility of the packet inspection engine is detection of DNS A-type replies, upon which it extracts the domain name and server IP addresses, and sends these via JSON to the broker, which writes it into a time-series DNS database. This database is used to associate a video stream to its content provider. Note that we use unencrypted DNS replies since our empirical results from the university campus traffic feed show that only a negligible fraction (less than 0.01%) of



(a) Youtube 144p(low resolution) (b) Youtube 480p(medium resolution) (c) Youtube 1080p(high resolution) (d) Youtube 2160p(u-high resolution)

(e) Netflix (high resolution) (f) Twitch (high resolution) (g) Facebook app (h) Large download

Fig. 3: Traffic profiles.



(a) Video u-high (original, N=1) (b) Video u-high (after sampling N=1000) (c) Download (original, N=1) (d) Download (after sampling N=1000)

(e) Video medium (original, N=1) (f) Video medium (after sampling N=1000) (g) Missed video 1 (original, N=1) (h) Missed video 2 (original, N=1)

Fig. 4: Traffic profiles under sampling.

daily DNS messages is encrypted. In future, with growth of encrypted DNS traffic (*i.e.,* DNS-over-TLS protocol), one may choose to perform reverse DNS lookups [19] or query existing `WHOIS` databases [7] (mapping IP address to AS/organization) to label video flows with their providers.

E. Telemetry Algorithm

Our data broker queries per-flow statistics (counters and timers), stores them in a time-series database, and exposes them to the user interface (which provides a real-time visualization of statistics for video streams and their resolutions) via appropriate RESTful APIs. The telemetry collects per-flow (fine grain) and per-group (coarse grain) usage statistics using the Stats collector module of our SDN application.

Algorithm: Recollect that our packet inspection engine identifies all elephant flows, which may include a mixture of video streams and elephant downloads, and suppresses their packets from being mirrored. We now develop an algorithm to distinguish video streams (from elephant transfers), and identify their content providers and resolutions. At a high level, the algorithm: (a) computes attributes of a given flow,

TABLE I: Number of flows captured with sampling.

Sampling N	**1**	**100**	**200**	**500**	**1000**	**2000**
Total num. of flows	16983	3251	1964	873	544	312
Num. of elephant flows	50	48	47	43	42	38
Num. of mice flows	16933	3203	1921	832	503	274

which are then fed into an intelligent classifier (discussed in §III-F) to distinguish video streams from elephant transfers, (b) queries DNS database using the flow's client/server IP address to associate the video stream with its content provider and (c) estimates the resolution of the video stream (*i.e.,* Low, Medium, High, Ultra-high).

Usage Collection and Storage: We collect flow counters per content provider (group table) and per video stream (reactive flow table). While the number of entries in the group table is generally small and fixed, the number of reactive flow entries can vary significantly with time. Polling the latter when the number of entries is large can result in a multi-part reply – for example the Noviflow switch breaks the response into chunks of 2500 flows each – putting considerable strain on the agent in the switch and affecting timeliness of the results. Prior work such as [15] has explored the trade-off between accuracy, timeliness, and network overhead of polling switch entries, though their work is evaluated only in mininet emulation; in this work we take a relatively simplistic approach, whereby we tune the polling frequency depending on the number of entries. Consequently, when the number of reactive flows is less than 2500, we poll the counters every second, and the frequency reduces to once every 4 seconds when the number of entries increases to 10,000. The flow/group-level counters are stored in a time-series Flow DB, as shown in Fig. 1(b), and are exposed periodically using a JSON-formatted message to the machine learning algorithm described next.

F. Classification using Machine Learning

We develop a machine learning technique to determine if traffic pertaining to an elephant flow is streaming video or not (the "video identifier"), and if so, to determine the stream resolutions (the "resolution classifier"). Our objective is to achieve video identification and classification in real-time with accuracy comparable to or better than computationally expensive techniques that require inspection of all traffic.

1) Attributes: Attributes selection is of paramount importance for training of classifiers, given that these should be predictive to correctly identify/classify video streams. To motivate our attributes selection and have an insight into behaviour of various flows, we plot in Fig. 3 the traffic pattern we have observed for various video streams including Youtube, Netflix and Twitch (at different resolutions: low, medium, high and ultra-high definition) and other elephant flows including Facebook application and large download (*i.e.,* a representative of bulk transfer or GoogleDrive/dropbox cloud storage sychronization) during the first three minutes of their activity. It can be seen that due to buffering that accompanies video streaming, the *idle-time* characteristic (*i.e.,* fraction of time that no data is exchanged) of video flows in Figs. 3(a)-3(f) is quite distinctive compared to the large download flow in Fig. 3(h)). We also note that the *average rate* (shown by dotted red lines) of the Youtube 2160p (4k ultra-high definition video) in Fig. 3(d) is much higher than that of other video resolutions (shown in Figs. 3(a)-3(c) and 3(e)-3(f)) but comparable to the large download in Fig. 3(h). In addition to idle-time and average rate, the *burstiness* characteristic of each flow is also distinctive – the low resolution video and the large download exhibit the most and the least bursty patterns respectively, among these representative profiles shown in Fig. 3. Based on these visual observations, we believe that idle-time, average rate and burstiness are collectively needed to identify and classify video flows. For example, the Facebook application flow shown in Fig. 3(g) exhibits similar characteristics of video streams (shown in Figures 3(b)-3(c)) in terms of idle-time and burstiness, but its rate is far below those of video streams.

The average rate and fraction of idle-time for a flow can be computed over a moving window (of say one minute). Burstiness of flow traffic can be computed in various ways [28], and it has been noted (particularly in the characterization of long-range dependent traffic) that it should be measured at multiple time-scales. We therefore compute the coefficient of variance (*i.e.,* the ratio of the standard deviation to the mean, CV = σ/μ) of our streams at time-granularities of 1-, 2-, 4-, 8- and 16-seconds, giving us σ_1/μ, σ_2/μ, σ_4/μ, σ_8/μ, and σ_{16}/μ. These burstiness measures, in addition to idle-time and average rate μ of each flow, are input as attributes to our classifiers. Note that for a new flow, we may have only a subset of burstiness attributes at the begininng, as computing σ_{16} would require collection of data for at least a minute. A flow that commenced only 20 seconds ago would only be able to yield σ_1/μ, σ_2/μ and σ_4/μ since we have less than 4 data points at time scales of 8-second and 16-second. It is important to note that we identify and classify elephant flows only. Therefore, attributes are computed by the SDN application by collecting flow-level telemetry for the reactive flow-rules inside the SDN switch.

Impact of packet sampling on attributes: We note that existing sampling methods for monitoring flows make it hard to achieve reasonable accuracy at acceptable overheads [43] – common practice of sampling is 1 in 1000 packets [2]. We undertook an experimental investigation of packet sampling from a real traffic mix of mice and elephant flows. We collected a 1-hour PCAP trace (of size 3.6 GB comprising 3,392,018 packets) from our University campus testbed. Our trace contained 16,983 flows of which 50 were elephant flows (ground truth known from the PCAP) including video and large downloads. We then applied a simple packet sampling (*i.e.,* randomly selecting 1 in N packets) at varying rates to our traffic trace. Table I shows the impact of sampling periodicity N on the count of flows captured. For each N we ran our sampling 10 times and computed the average values. We observe that sampling completely misses some flows, which gets worse as the sampling periodicity N increases. For example, with $N = 1000$, 97% of mice flows are missed, and more worryingly, 16% of elephant flows (8 out of 50) are totally missed, all of which happened to be video flows – this indicates that sampling can miss a significant number of long streams.

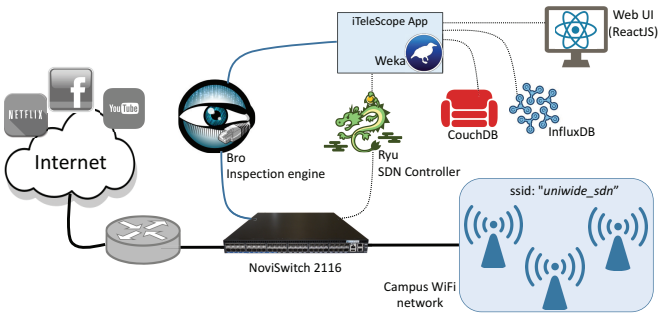


Fig. 5: iTeleScope prototype.

Another significant observation is that packet sampling does not preserve the profile of elephant flows, for video streams as well as large downloads. Considering the two main attributes needed for classification, namely burstiness and idle-time, we show in Fig. 4 how the traffic profile and the resulting attributes change due to sampling. Fig. 4(a) and Fig. 4(b) show the profile before and after sampling for ultra-high resolution video, Fig. 4(e) and Fig. 4(f) for medium resolution video, and Fig. 4(c) and 4(d) for a download stream. Worryingly, the profile of the download stream becomes similar to that of video, since its burstiness almost doubles and idle-time fraction increases significantly from 2.65% to 32.45% – this greatly increases the risk of misclassification of the download stream as a video stream. We also show in Fig. 4(g) and Fig. 4(h) that a low-resolution and a medium-resolution video stream are entirely missed by the random sampling process, and will hence not be classified at all.

To summarize, random sampling of packets causes two problems: (a) it risks entirely missing a non-negligible fraction of elephant flows, specifically video flows of low and medium resolutions; and (b) it distorts the traffic profile and hence risks misclassifying download streams as video flows. By contrast, our SDN-based telemetry method is more intelligent in how it samples only the initial packets of a flow and thereafter obtains per-flow telemetry on elephant flows from the switch. This flexibility provides us with a fine-grained visibility into the profile of elephant flows (and their attributes), enabling accurate video identification and classification without missing any flows.

2) Identification/Classification: As mentioned earlier, our iTeleScope data broker employs two classifiers namely the video identifier (to indicate if the flow is a streaming video or not) and the resolution classifier (to determine the resolution of video during playback). The identifier and classifier are invoked every 16 seconds to dynamically capture profile changes (*e.g.,* video stream rate adaptation) – initial invocation may have access to only five attributes (idle-time, μ, σ_1/μ, σ_2/μ, and σ_4/μ), and subsequent invocations that have access to more (burstiness-related) attributes may change the classification, improving accuracy and/or identifying resolution changes. Note that our attributes are extracted from flows activity, without packet-level inspection. This means that our classifiers are trained only by behavioral attributes of flows, and hence become agnostic to traffic encryption – payload contents are not used. The training of the classifiers will be described in the next section.

Recent Flows

Tag	Server IP	Client IP	Total Bytes (MB)	Estimated Rate (Mbps)	Quality	Duration (s)
netflix	138.44.10.30	Anonymous IP	9.21	1.04	Medium	126
youtube	203.5.76.207	Anonymous IP	5.58	0.15	Low	218
netflix	138.44.10.30	Anonymous IP	209.16	7.69	High	249
youtube	203.5.76.206	Anonymous IP	888.57	17.00	Ultra-high	421
facebook	157.240.8.23	Anonymous IP	128.23	4.70	High	525

Fig. 6: Recent video flows.

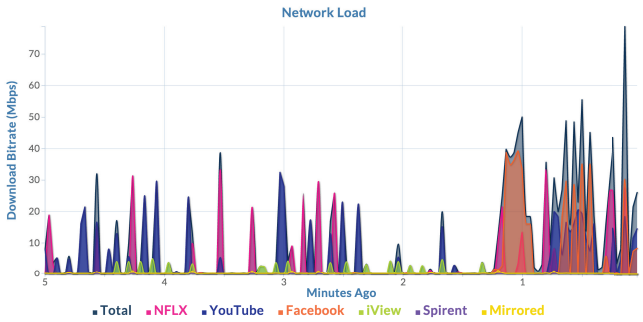


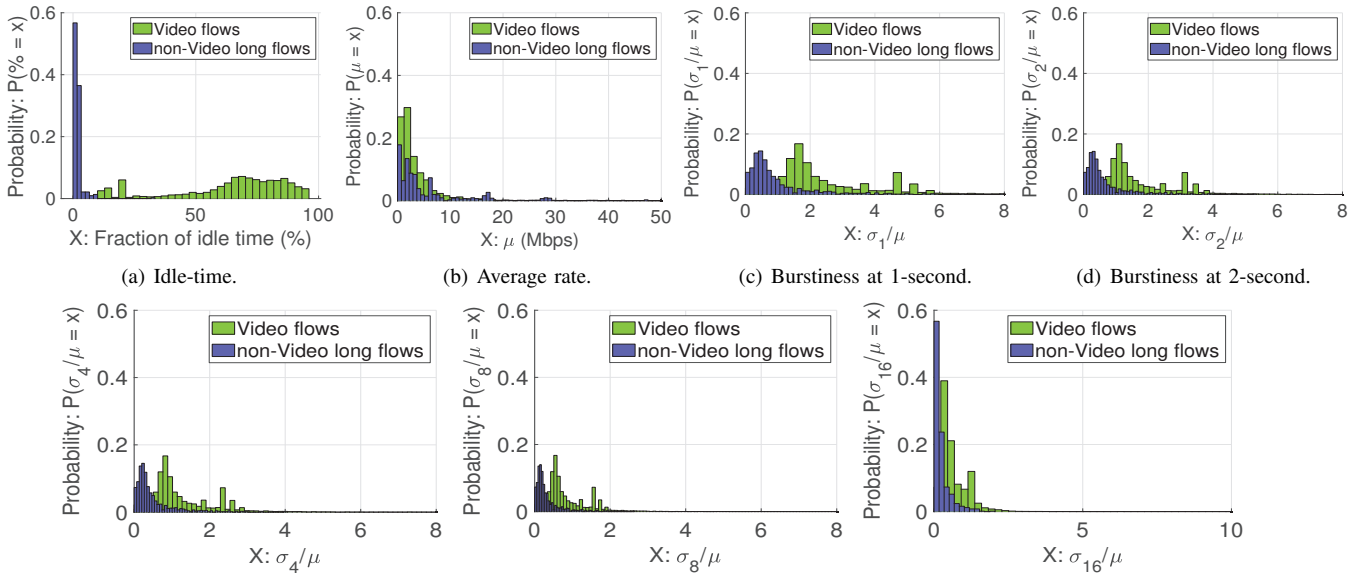
Fig. 7: Load of the network and video traffic.

IV. Prototype Implementation and Machine Training

A. Prototype

We have implemented a fully functional near-production-grade iTeleScope system that identifies and classifies video streams in real time at line rate of up to 10 Gbps. For our system we have implemented an application on top of the Ryu SDN controller, augmented the Bro packet inspection engine for flow state management and event triggering, implemented various databases including InfluxDB, PostgreSQL, and CouchDB, and a web-GUI (in ReactJS) for interaction with our tool. Further, each of these components operates on a separate docker container or virtual machine in our cloud environment powered by the VMware Esxi 6.0 hypervisor. All VMs run Ubuntu server 14.04 LTS and are allocated to a four-core CPU, with 8 GB of memory and 32 GB disk space. Our system is currently managing three environments: (a) an SDN-enabled experimental lab network connected via WiFi access points (used for machine training in §IV-B), (b) a point-to-point link over which an industrial scale Spirent traffic generator feeds traffic into our setup, and (c) a live campus dorm network link operating at 10 Gbps and serving several hundred real users. Our implemented design is depicted in Fig. 5, and the interested reader can see the interface live via our publicly accessible website at: `https://telescope-core.sdn.unsw.edu.au/`.

SDN switch: Our SDN switch is a fully Openflow 1.3 compliant NoviSwitch 2116, as shown in Fig. 5. It provides 160 Gbps of throughput, tens of thousands of TCAM flow entries, and millions of exact-match flow-entries in DRAM, and we found it to amply cater to the requirements of this project.



(a) Idle-time. (b) Average rate. (c) Burstiness at 1-second. (d) Burstiness at 2-second.

(e) Burstiness at 4-second. (f) Burstiness at 8-second. (g) Burstiness at 16-second.

Fig. 8: Histogram of idle-time, average rate and burstiness at various time scales (video vs. non-video).

Packet Inspection Engine: We use the Bro (v2.4.1) [45] open-source tool for inspection of the mirror traffic. We wrote event-handlers in Bro that keep track of the flow duration and volume, and to trigger an API call to the data broker when an elephant flow is detected. Similarly, DNS replies are also parsed and the information is passed to the data broker for recording into the time-series database.

Data broker: We used python to implement our data broker that receives the 5-tuple of elephant flows and DNS information from the Bro inspection engine, inserts/modifies flow/group entries, and collects statistical data from our SDN application via RESTful API. Flow and group stats collected from the SDN application are written into a time series InfluxDB. Flow level information is queried from InfluxDB periodically for processing by the intelligent classifier trained by the Weka tool [3] using Weka's Python library wrapper interface (v0.3.9). The intelligent classifier identifies video flows, queries the DNS database to label video flows, calls RESTful APIs to modify flow entries' output group, and identifies video stream resolutions, as described in §III-C.

SDN controller and application: We used the Ryu (v4.0) Openflow controller for operating our system, and developed a Python based SDN application exposing northbound RESTful APIs to the data broker for inserting or modifying network rules and polling flow statistics. Successful RESTful API calls result in appropriate actions (*e.g.,* network rules insertion, modification and counters collection) at the SDN switch serving the data-plane.

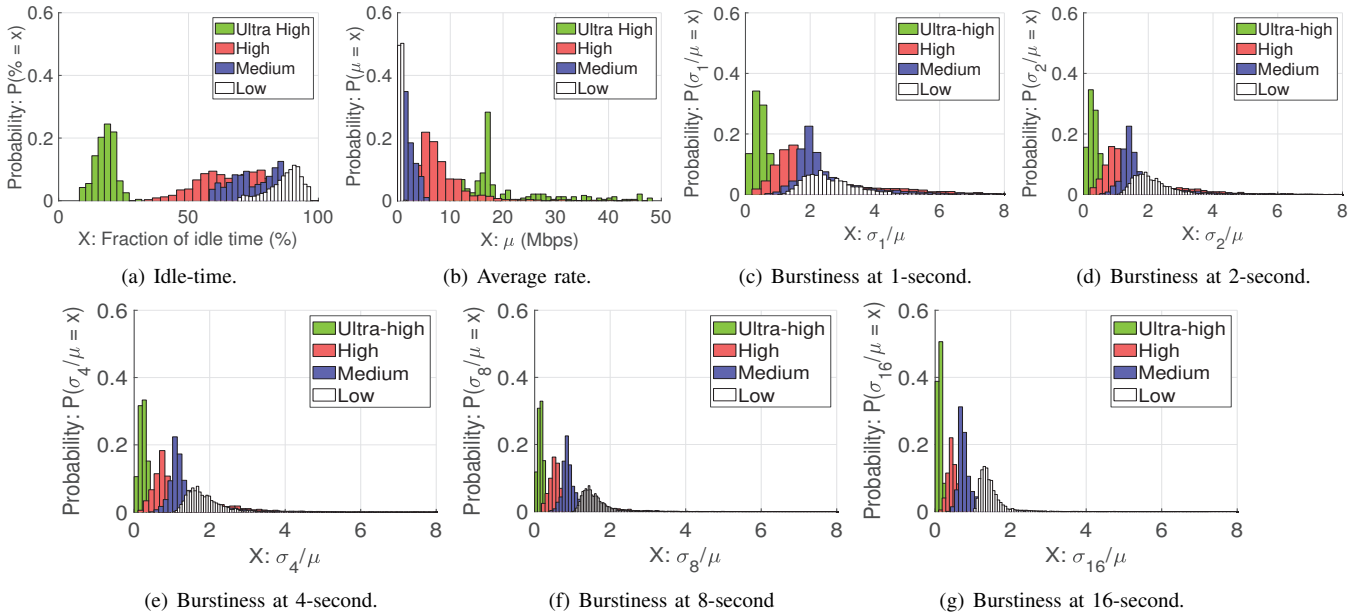
DataBases: We employ three databases in our system to store flows usage statistics, DNS information, and system configurations. We use time-series InfluxDB (v1.0.0) to store periodic flow/group statistics as mentioned in §III-E. In the same InfluxDB we also store information of DNS A-type replies including the domain name and client/server IP addresses. An object relational database PostgreSQL (v9.6.3) is used to store the mapping between domain IP addresses, domain name suffixes and provider names. Lastly, we use a NoSQL CouchDB (v2.0.0) document-oriented database to store configurations of the SDN switch such as DPID and multi-table configs.

Web Interface: We provide a front-end for network operators to visualize video streams in their network, implemented in ReactJS using Rubix template and D3 library. Snapshots are shown in Fig. 6 and 7. The reader can see the live interface via a public accessible website at: `https://telescope-core.sdn.unsw.edu.au/`. The interface shows aggregated video consumption statistics by different content providers over the last one hour, one day and one week, the total number of elephant flows, and the most recent video streams.

B. Machine Training

We now train our classifiers with datasets collected in our lab over our prototype. In order to have the ground truth for the training, we scripted the streaming of video from various providers (*i.e.,* Youtube, Netflix, Youku, Facebook, Tencent) at various resolutions. The automation was done using APIs where possible, such as the Youtube Player API that allows videos to be streamed at specified resolution (*i.e.,* low: 144p, 240p, 360p; medium: 480p, 720p; high: 1080p, 1440p; and ultra-high: 4K), and by launching a browser URL otherwise. We also scripted the generation of elephant flow traffic including large ISO file downloads and Google-Drive sync operations, and mice flows from dynamic webpage loads (*e.g.,* Office 365, Facebook homepage, WhatsApp).

For the purpose of training, our scripts limit all flows (video and non-video) to 128 seconds (*i.e.,* about two minutes), even though all chosen videos have total length in excess of 20 minutes. At the end of each two-minute activity, the script queries the InfluxDB to extract the flow profile (byte counts at 1-second time interval) to calculate attributes (as explained in



(a) Idle-time. (b) Average rate. (c) Burstiness at 1-second. (d) Burstiness at 2-second.

(e) Burstiness at 4-second. (f) Burstiness at 8-second (g) Burstiness at 16-second.

Fig. 9: Histogram of idle-time, average rate and burstiness at various time scales (various resolutions).

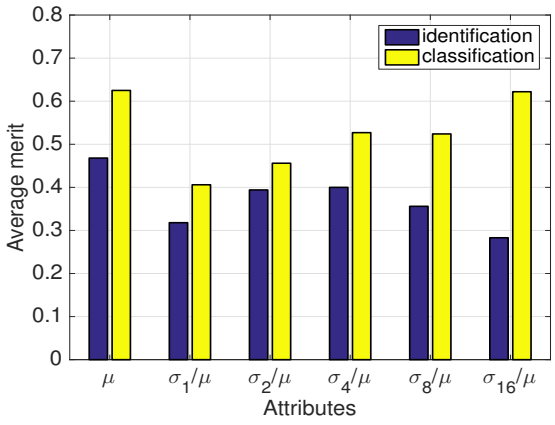


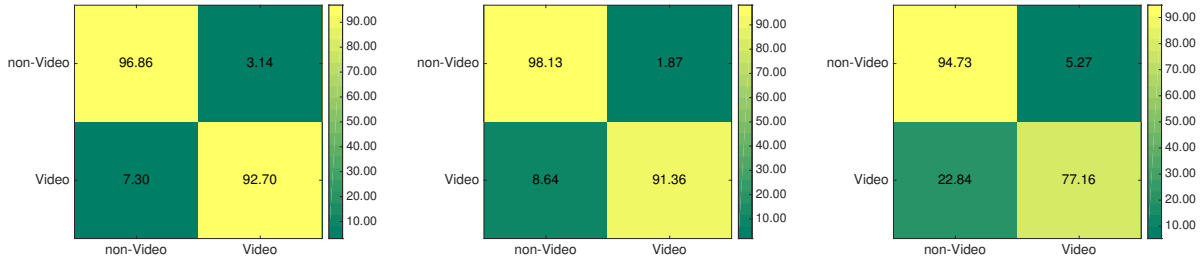
Fig. 10: Merit of attribute.

§III-F). We then split the 128-second traffic profile into 8 sub-profiles (*i.e.,* time intervals of [1,16]s, [1,32]s, [1,48]s, [1,64]s, [17, 80]s, [33, 96]s, [48, 112]s, and [65, 128]s). The script lastly computes the attributes for each of the sub-profiles. We note that short sub-profiles (*e.g.,* [1,16]s) will have incomplete attributes such as σ_8/μ and σ_{16}/μ. We have run our script for two weeks and collected a total of 28,543 labeled training instances for elephant flows (video and non-video) of which 10,416 instances were labeled for various video resolutions.

1) Attribute Profiles: We briefly describe the profiles of the attributes (idle-time, average rate, and burstiness at various time-scales) collected from our dataset. Fig. 8 shows the histogram of these attributes used by the video identifier machine. The difference between video and non-video long flows is visually apparent: for example, Fig. 8(a) shows that non-video flows have very low idle time fraction (centered at about 1% with minor deviations), whereas video traffic idle-time fraction is widely spread between 20% and 95%. The video and non-video streams are not so distinct in their distribution of average rate (Fig. 8(b)); however, they do have different burstiness behaviors at various time-scales, as seen in Figs. 8(c)-8(g).

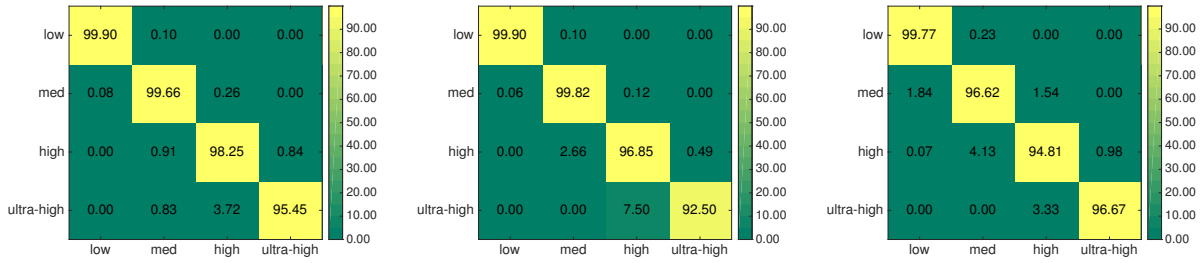
The distribution of the attributes used by the resolution classifier is shown in Fig. 9. The distinctions between the resolutions are again visually apparent: Fig. 9(a) shows that as video resolution increases, the idle-time fraction distribution (predictably) shifts to the left, whereas the average rate distribution shifts to the right (Fig. 9(b)). The video stream burstiness for the various resolutions is also distinct at the various time-scales, as shown in Figs. 9(c)-9(g).

It is evident from the above that the different attributes will have different importance for the classification engines that identify video streams and their resolutions. In machine learning, there exist selection techniques such as principal component analysis (PCA) and information gain that help simplify (optimize) models and rank attributes by importance. PCA [10] is a statistical procedure, typically used to reduce a large set of attributes to a small set while keeping most of the information in the original set. Since we have identified only a handful of traffic attributes (§III-F) which are not computationally expensive, reduction of attributes space (via PCA) is not employed for this work. Instead, in order to quantify the importance of attributes, we used the `InfoGain` tool that is part of the Weka Machine Learning package, that determines an average merit score of each attribute based on the measure of information gain from that attribute – the larger the reduction of entropy (uncertainty), the higher the merit of an attribute. Fig. 10 shows the merit of each attribute for the two machines. The blue bars, corresponding to the video identification machine, show that all attributes are nearly equally important, with idle-time being slightly more dominant, and average rate slightly less. By contrast, the video resolution classifier (yellow bars) relies heavily upon the average rate, followed by the idle-time. The burstiness at the various time-scales are roughly equally important to both machines. These merit scores confirm our initial intuition that the selected attributes constitute reasonable inputs to the video classification engines.



(a) J48. (b) Random forest. (c) MLP.

Fig. 11: Confusion matrix of video identifier.



(a) J48. (b) Random forest. (c) MLP.

Fig. 12: Confusion matrix of video resolution classifier.

TABLE II: Tuning Random-Forest (video identifier).

		Number of attributes					
		1	2	3	4	5	6
Depth of tree	1	80.7978	89.2830	91.5748	88.0770	85.7852	85.7634
	2	85.5997	91.7440	92.9281	93.4083	93.7138	93.7084
	3	90.7454	94.0849	94.4996	94.576	94.5542	94.5051
	4	93.7193	94.4505	94.6033	94.7015	94.7288	94.6306
	5	94.1777	94.6797	94.8434	94.898	94.7725	94.5869
	6	94.5762	94.9089	94.9907	94.7179	94.5978	94.6742
	7	95.0071	95.0999	94.7506	94.7397	94.7233	94.7179
	8	95.1162	95.0453	94.7670	94.7124	94.6197	94.6361
	9	95.2908	94.8981	94.7670	94.7615	94.6306	94.6579
	10	95.1108	94.8052	94.7070	94.7452	94.7288	94.6906
	11	94.9471	94.7831	94.6977	94.7124	94.7233	94.6561
	12	94.7834	94.7179	94.6488	94.6397	94.7233	94.6343

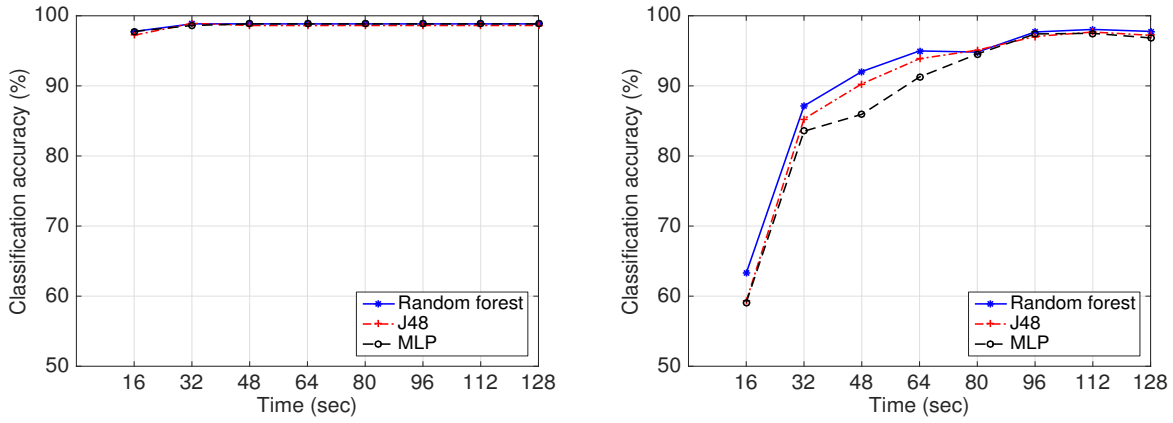
2) Tuning Machine Parameters: We employ three popular classification algorithms: J48, Random Forest, and MLP, from the Weka machine learning library. We tune the parameters of these machine learning algorithms to maximize their performance for the chosen attributes. For example, the Random Forest algorithm has two parameters we can tune – the depth of the tree and the number of selected attributes for each tree. For each combination of parameters, we evaluated its efficacy via 10-fold cross-validation, whereby the dataset (collected in our lab) is randomly split into training (90% of total instances) and testing (the remaining 10% of total instances) sets, and accuracy is averaged over 10 runs to produce a single performance metric.

For the video identification machine, the overall accuracy of Random Forest is shown in Table II for the various parameter combinations. The highest accuracy (of 95.29%) was achieved by setting the tree-depth to 9 and the number of attributes in the tree to 1. Increasing tree-depth or number of attributes per-tree beyond these numbers reduces accuracy due to overfitting. We similarly tuned the J48 algorithm by adjusting the number of instances on each leaf (optimal = 4), and the MLP algorithm by varying the number of hidden layers (optimal = 8), in order to maximize performance, yielding accuracy of 95.12% and 87.36% respectively.

For the video resolution machine, we used a similar method to tune the parameters. The Random Forest model reaches its highest accuracy of 99.4411% when the tree-depth equals to 5 and the number of attributes per-tree is 3. For optimal settings of other algorithms: J48 uses 4 instances per-leaf for the best accuracy of 99.4521%, and MLP uses 5 hidden layers for the accuracy of 97.3577%.

3) Off-Line Accuracy: Having tuned the various algorithms to maximize their performance, we now conduct an off-line evaluation of their accuracy on our lab dataset (for which the ground truth is known). A ten-fold cross-validation is performed over the entire dataset, and results are depicted in the form of a confusion matrix, in which rows denote the ground truth and columns the machine output. Fig. 11 shows the accuracy of the video identification machine from the three machine learning algorithms. J48 and Random Forest correctly identify video flows over 90% of the time, while MLP has a poor true positive rate of 77%. We believe this is because the geometry of our training instances is more suitable for decision-tree-based classifiers (J48 and Random forest) than for neural-network-based classifiers (MLP). The identification of non-video flows has higher accuracy with all the methods, with J48 and Random Forest being above 96% and MLP nearly 95%. We believe the higher false-positive rate for video flows (than non-video flows) is because they can sometimes change their profile (due to network conditions and rebuffering), especially at higher resolutions, making them look closer to downloads.

Fig. 12 shows the confusion matrix for the resolution classifier. All three machine learning models yield a fairly high accuracy, being over 99% at low resolutions, and drop-



(a) Video identification.

(b) Resolution classification.

Fig. 13: Real-time accuracy of (a) video identification, and (b) video resolution classification.

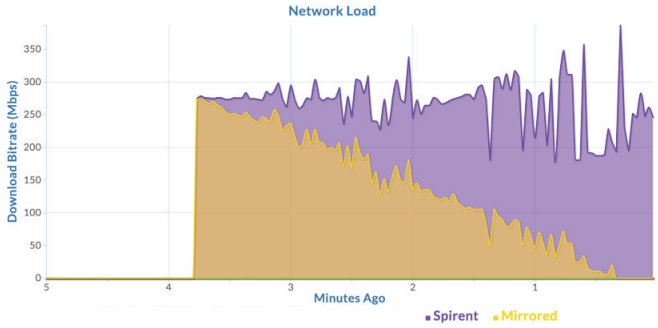


Fig. 14: Network load (31920 flows arrive at the rate of 280 flows-per-second).

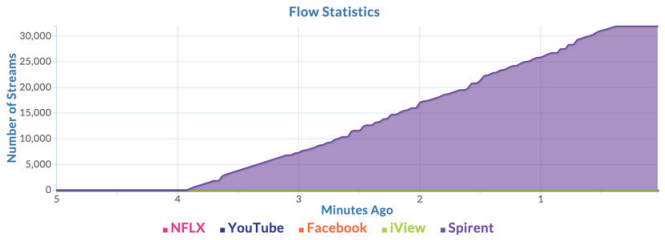


Fig. 15: Flow statistics (31920 flows arrive at the rate of 280 flows-per-second).

ping somewhat at higher resolutions. J48 and Random Forest outperform MLP at higher resolutions, though they all tend to sometimes classify ultra-high resolution videos as high resolution, and medium resolution videos as high resolution. These errors (of about a few percent) are not surprising, since the attributes of the high resolution videos overlap with those of medium and ultra-high resolution flows, as depicted in Fig. 9 – specifically, the two most important attributes, idle-time and average rate, have significant overlaps.

4) Real-Time Accuracy: Recall that certain attributes (such as burstiness at time-scales of 8 and 16 seconds) become available only after the flow has been active for a certain duration. We now evaluate the accuracy of our classification methods when they operate in real-time, namely as and when flow "sub-profiles" become available from the first 16 seconds to the past one minute over their two-minute lifespan. Fig. 13(a) shows the time evolution of real-time classification accuracy – video streams are identified with an accuracy of about 60% if only the first 16 seconds worth of their profile is available to the classifier. This is also because video flows tend to buffer in the beginning, which makes them less distinguishable from downloads in the initial few seconds. As the length of sub-profiles increases, so does the accuracy – after 48 seconds, 80% accuracy is achieved, and this rises to 95% for J48 and Random Forest at about a minute-and-a-half.

Similarly, Fig. 13(b) shows that the accuracy of the resolution classifier increases rapidly with the length of sub-profile. This is not surprising, as various attributes computed during the first 16 seconds do not perfectly identify/classify video flows due to their initial buffering. For example, an ultra-high resolution video (Fig. 3(d)) is very similar to a large download if one considers the idle-time, average rate and burstiness for only the initial 16 or 32 seconds of the profile. The attributes σ_8/μ and σ_{16}/μ become available respectively only after 32 and 64 seconds of stream activity, and are fairly important for the classification, leading to a very rapid rise in accuracy at around the minute mark.

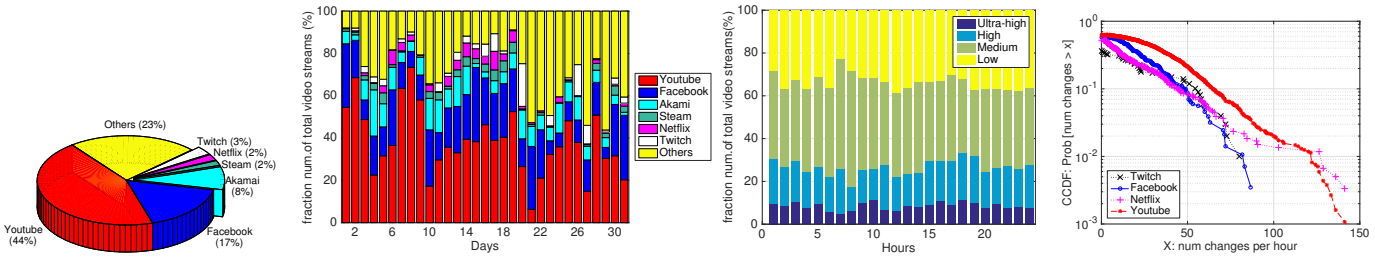
Summary: Our system uses a judicious combination of software and hardware to isolate elephant flows and monitor their individual behavior. Flow attributes such as idle-time fraction, average rate, and burstiness at various time-scales are extracted without packet-level inspection, and fed to a machine learning model. Video flows can be identified by our machine with 70% certainty within 30 seconds, rising to over 90% within two minutes, while video resolution can be correctly deduced with 80% accuracy in 30 seconds, rising to over 95% within two minutes. In what follows we evaluate the scalability of our system and discuss insights obtained from a real deployment.

V. Evaluation and Deployment

We briefly evaluate the scalability of our system to large flow numbers and high arrival rates (§V-A), and describe insights obtained from a trial deployment (§V-B).

A. Scalability Test

We subject our system to stress-testing with emulated elephant flows from a traffic generator. We connect a Spirent [44]



(a) Total num. of video streams. (b) Daily num. of video streams. (c) Hourly num. of video resolution. (d) CCDF of resolution change.

Fig. 16: Distribution of Dorm video consumption.

TestCenter chassis SPT-11U (firmware v4.24.1026), which is a high-precision commercial-grade hardware traffic generator equipped with a 12-port GE HyperMetric test module, to our NoviSwitch on two ports. One port of the Spirent generates traffic streams representative of video servers, while the other port receives traffic back from our system to represent end-user clients. We wrote TCL scripts to automate the process of traffic emulation: 14 pairs of transmitter/receiver were created, each allocated a distinct /28 public-IP address, and each pair establishes 20 parallel stream blocks each of a separate layer-4 (TCP) port number, thereby generating 280 concurrent flows per second. Further, the port number of each stream block kept circulating each second over a range of 114, resulting in a total of 31920 flows. Each flow sends traffic at a constant rate uniformly distributed between [0.8, 1.2] Mbps (representative of a 360p video). The emulation was run for 300 sec.

Fig. 14 depicts the link load (purple line) and mirror load (brown line) at 1s intervals. The measured loads (deduced from OpenFlow counters) corroborate very well with the actual loads (reported by Spirent), being within 1.7% of each other, confirming that our system measures flow rates accurately. Further, the mirror traffic load (sent for software processing) is initially at 100% of offered load, but gradually drops to zero (over a period of 210 seconds) as the reactive flow entries are pushed into the OpenFlow switch to stop the mirroring of elephant flows. Fig. 15 shows the ramp-up in the number of reactive flow-table entries, being pushed at the rate of around 280 flow-mods per-second. The stress-test is meant to ensure that our system is scalable to large number of active elephant flows (31920 in this case), and to handle high rate of new flows in the switch hardware (280 new flows per-second), ensuring proper operation in real networks, as described next.

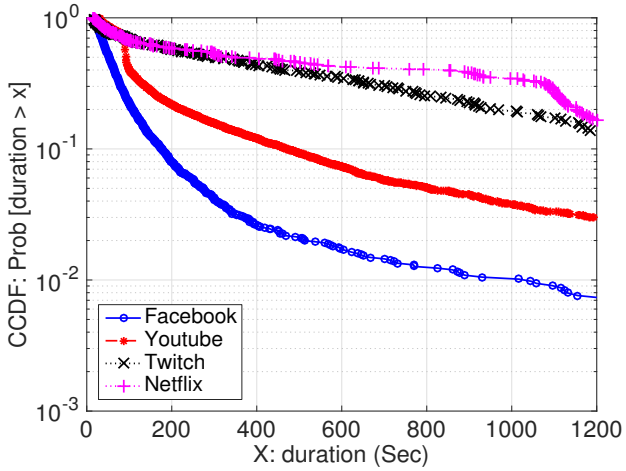
B. Campus Deployment

Our iTeleScope system has been deployed and operational for several months in a university network serving hundreds of students resident in the on-campus dorm. The University IT department provisioned a full feed of the dorm traffic to our system, and we obtained ethics clearance (UNSW Human Research Ethics Advisory Panel approval number HC16712) from our organization in order to conduct this trial, since it gives us access to all network traffic.

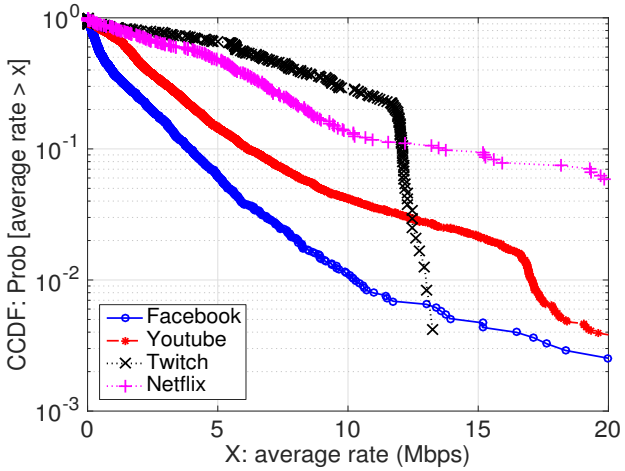
Our system not only displays video flow information (end-point, provider, duration, rate, resolution) in real-time at: `https://telescope-core.sdn.unsw.edu.au/`, but also records video flow information into an InfluxDB that can be analysed post-facto. In what follows we highlight some of the insights we obtained from our system's flow database over a one month period during the academic term. In Fig. 16(a) we show a pie-chart of the fraction of streams from the most popular video content providers – not surprisingly, Youtube and Facebook video streams dominante at 44% and 17% respectively. The gaming video platform Twitch contributes 3% of streams, more than Netflix (2%), most likely because students tend to prefer free over paid content. Around 8% of video flows are sourced from Akamai media servers (*i.e.,* `akamai.net` and `akamaiedge.net`). Lastly, our system allowed identification of many other video providers such as Tencent, Youku, Amazon, Yahoo, Instagram, Fastly, Alibaba, Baidu, Huya, Battlenet, HLtv, OurDvs, and Dailymotion (grouped under "Others" in Fig. 16(a)) that collectively constitute 23% of video streams during the month. This break-down of video streams by provider elicited much interest from the IT department, who had no prior visibility into video viewing patterns (especially for less popular video providers) in the campus dorm.

The day-by-day video consumption pattern over the month is shown in Fig. 16(b). It is seen that there is substantial fluctuation in the relative proportion of video providers from day to day, and interestingly, the dorm residents tend to watch Twitch gaming videos more on weekends than weekdays. In Fig. 16(c) we plot the fraction of video streams at different resolutions on an hourly basis (averaged over the selected month). Surprisingly, a majority of videos are playing at medium resolution and only a small fraction of videos are at ultra-high resolution, though the campus network has abundant bandwidth and rarely experiences congestion. We believe that this is because most of the free content on Youtube and Facebook is only available at medium or lower resolution (*i.e.,* 144p, 240p, 360p, 480p and 720p).

Our system also lets us analyze mid-stream resolution changes – in Fig. 16(d) we plot the CCDF of resolution changes (normalized to a per-hour basis) in video streams from Youtube, Facebook, Netflix and Twitch. Unsurprisingly, Youtube videos are the most aggressive in adapting their resolution, with 20% of streams adapting their resolution at least 20 times per hour (*i.e.,* once every 3 minutes on average), and 1% of streams adapting once per minute. Facebook videos are generally shorter (more on this next), yet 11% adapt their resolution on average every 3 minutes. Netflix videos tend to adapt their resolution less frequently, with only 7% of videos



(a) Video duration



(b) Video rate

Fig. 17: CCDF of Dorm video characteristics.

changing their resolution every 3 minutes on average. Twitch videos show the least adaptation, with 85% of them never changing their resolution during their entire playback (which can be reasonably long).

Further insights into the dorm video viewing patterns are shown in Fig. 17, depicting the CCDFs of playback duration and average bit-rates for the 4 popular content providers (Facebook, Youtube, Twitch, and Netflix) over the selected month. Fig. 17(a) confirms that Netflix and Twitch videos are watched for reasonably long durations (nearly 40% of streams last longer than 10 minutes), followed by Youtube and Facebook for which 7% and 2% of videos are respectively watched for longer than 10 minutes by dorm residents. The average bit-rates shown in Fig. 17(b) also confirm that Twitch and Netflix videos are more bandwidth intensive than Youtube and Facebook videos – Twitch and Netflix use on average 6.6 Mbps while this measure is 2.8 Mbps and 1.5 Mbps for Youtube and Facebook respectively.

Our system is in commercial pilot with a Tier-1 carrier, and is providing real-time video traffic visibility as well as off-line reporting of video consumption patterns; the outputs are comparable to those from a commercial DPI appliance, but at lower cost by virtue of white-box hardware. Confidentiality requirements unfortunately prevent us from disclosing any findings from the trial.

VI. Conclusion

Video traffic dominates enterprise and carrier network traffic, yet operators have limited visibility into the number, duration, and quality of video flows traversing their network. Existing solutions are either hardware-based and expensive, or software-based and unscalable. Our solution, iTelescope, judiciously combines software packet-level inspection with hardware flow-level telemetry to isolate elephant flows and extract flow attributes, that are used in conjunction with machine learning to identify and classify video flows in real-time at low-cost. We have built our solution using off-the-shelf SDN hardware and open-source software. We have trained and validated the accuracy of our machine learning algorithms in the lab, demonstrated our system scalability to tens of thousands of concurrent streams, and deployed it in a live network serving hundreds of real users. Our solution provides unprecedented visibility to network operators, and has the potential to become a platform for actively managing video delivery quality on a per-stream basis in the near future.

References

[1] Network performance analysis. Tech. rep., Cisco Systems, 2005.
[2] SFlow sampling rates. https://blog.sflow.com/2009/06/sampling-rates.html, 2009.
[3] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/, 2016.
[4] Allot ClearSee Network Analytics. https://www.allot.com/products-enterprise/clearsee-network-analytics/, 2018.
[5] Qosmos Detailed Traffic Visibility Up To Layer 7. https://www.qosmos.com/products/deep-packet-inspection-engine/, 2018.
[6] Sandvine Network Analytics. https://www.sandvine.com/products/network-analytics, 2018.
[7] Whois Lookup. http://whois.domaintools.com, 2019.
[8] Cisco visual networking index: Forecast and methodology, 2015-2020. Tech. rep., Cisco Systems Inc., June 2016.
[9] A. Dainotti et al. Issues and future directions in traffic classification. *IEEE Network 26*, 1 (January 2012), 35–40.
[10] Abdi, H., and Williams, L. J. Principal component analysis. *WIREs Comput. Stat. 2*, 4 (July 2010), 433–459.
[11] Aceto, G., Dainotti, A., de Donato, W., and Pescape, A. PortLoad: Taking the Best of Two Worlds in Traffic Classification. In *Proc. IEEE INFOCOM* (March 2010).
[12] Afek, Y., et al. Sampling and Large Flow Detection in SDN. In *Proc. ACM SIGCOMM* (London, UK, August 2015).
[13] Bujlow, T., Carela-Español, V., and Barlet-Ros, P. Independent Comparison of Popular DPI Tools for Traffic Classification. *Comput. Netw. 76*, C (Jan. 2015), 75–89.
[14] Bujlow, T., et al. Independent comparison of popular DPI tools for traffic classification. *Computer Networks 76* (2015), 75 – 89.
[15] Chowdhury, S. R., et al. PayLess: A low cost network monitoring framework for Software Defined Networks. In *Proc. IEEE NOMS* (May 2014).
[16] Claise, B. Cisco systems netflow services export version 9. RFC 3954, Octoer 2004.
[17] Curtis, A. R., et al. DevoFlow: Scaling Flow Management for High-Performance Networks. In *Proc. ACM SIGCOMM* (Toronto, Ontario, Canada, Aug 2011).
[18] Deri, L., Martinelli, M., Bujlow, T., and Cardigliano, A. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)* (Aug 2014).

[19] FUKUDA, K., HEIDEMANN, J., AND QADEER, A. Detecting Malicious Activity With DNS Backscatter Over Time. *IEEE/ACM Transactions on Networking 25*, 5 (Oct 2017), 3203–3218.

[20] GILBERT, A., AND KARLOFF, H. On the Fractal Behavior of TCP. In *Proc. ACM Symposium on Theory of Computing* (San Diego, CA, USA, Jun 2003), STOC '03.

[21] HE, L., XU, C., AND LUO, Y. vtc: Machine learning based traffic classification as a virtual network function. In *Proc. ACM SDN-NFVSec* (New York, NY, USA, Mar. 2016).

[22] HOFSTEDE, R., ET AL. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys & Toturials 16* (2014), 2034 – 2064.

[23] HOHN, N., VEITCH, D., AND ABRY, P. Does Fractal Scaling at the IP Level Depend on TCP Flow Arrival Processes? In *Proc. ACM SIGCOMM Workshop on Internet Measurment* (Marseille, France, Nov 2002).

[24] HSIEH, C., WENG, N., AND WEI, W. Scalable Many-Field Packet Classification for Traffic Steering in SDN Switches. *IEEE Transactions on Network and Service Management 16*, 1 (March 2019), 348–361.

[25] HUANG, T.-Y., ET AL. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proc. ACM SIGCOMM* (Chicago, Illinois, USA, Aug 2014).

[26] ISOLANI, P. H., ET AL. Interactive monitoring, visualization, and configuration of OpenFlow-based SDN. In *Proc. IFIP/IEEE IM* (May 2015).

[27] J. CASE ET AL. Simple Network Management Protocol (SNMP). RFC 1157, May 1990.

[28] JIANG, H., ET AL. Why is the Internet Traffic Bursty in Short Time Scales? In *Proc. of ACM SIGMETRICS* (Banff, Alberta, Canada, 2005).

[29] KAMPEAS, J., COHEN, A., AND GUREWITZ, O. Traffic classification based on zero-length packets. *IEEE Transactions on Network and Service Management 15*, 3 (Sep. 2018), 1049–1062.

[30] L. HENDRIKS ET AL. Assessing the quality of flow measurements from openflow devices. In *Proc. Traffic Monitoring and Analysis (TMA)* (Louvain La Neuve, Belgium, April 2016).

[31] LI, B., WANG, Z., LIU, J., AND ZHU, W. Two Decades of Internet Video Streaming: A Retrospective View. *ACM Trans. Multimedia Comput. Commun. Appl. 9*, 1s (Oct. 2013), 33:1–33:20.

[32] LIU, C., ET AL. Rate Adaptation for Adaptive HTTP Streaming. In *Proc. ACM MMSys* (San Jose, CA, USA, 2011).

[33] LIU, N. X., AND BARAS, J. S. Modeling Multi-Scale TCP Traffic With A Structural Model. http://nelsonliu.info/archive/InternetTraff.pdf, 2003.

[34] LIU, Z., MANOUSIS, A., VORSANGER, G., SEKAR, V., AND BRAVERMAN, V. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proc. ACM SIGCOMM* (Florianopolis, Brazil, Aug 2016).

[35] MADANAPALLI, S., LYU, M., KUMAR, H., HABIBI GHARAKHEILI, H., AND SIVARAMAN, V. Real-time detection, isolation and monitoring of elephant flows using commodity SDN system. In *Proc. IEEE/IFIP NOMS* (Taipei, Taiwan, April 2018).

[36] MISRA, V., AND WEI-BO GONG. A hierarchical model for teletraffic. In *Proc. IEEE Decision and Control* (Dec 1998).

[37] NAMDEV, N., SANJAY, S., AND SHIKHA, A. Recent Advancement in Machine Learning based Internet Traffic Classification. *Procedia Computer Science 60* (2015), 784 – 791.

[38] NG, B., HAYES, M., AND SEAH, W. Developing a Traffic Classification Platform for Enterprise Networks with SDN: Experiences & Lessons Learned. In *Proc. IFIP Networking* (Toulouse, France, May 2015).

[39] RIEDI, R., AND LÉVY VÉHEL, J. Multifractal Properties of TCP Traffic: a Numerical Study. Research Report RR-3129, INRIA, 1997. Projet FRACTALES.

[40] RIEDI, R. H., CROUSE, M. S., RIBEIRO, V. J., AND BARANIUK, R. G. A multifractal wavelet model with application to network traffic. *IEEE Transactions on Information Theory 45*, 3 (April 1999), 992–1018.

[41] SEDDIKI, M. S., SHAHBAZ, M., DONOVAN, S., GROVER, S., PARK, M., FEAMSTER, N., AND SONG, Y.-Q. FlowQoS: QoS for the Rest of Us. In *Proc. ACM HotSDN* (Chicago, Illinois, USA, Aug. 2014).

[42] SFLOW.ORG. sFlow consortium. http://sflow.org/, 2017.

[43] SIVARAMAN, V., ET AL. Heavy-Hitter Detection Entirely in the Data Plane. In *Proc. ACM SoSR* (Santa Clara, CA, USA, April 2017).

[44] SPIRENT COMMUNICATIONS. Spirent traffic generator. http://www.spirent.com/, 2017.

[45] TEAM, B. P. Bro project. https://www.bro.org/, 2017.

[46] VALENTI, S., ET AL. Datatraffic monitoring and analysis. Springer-Verlag, Berlin, Heidelberg, 2013, ch. Reviewing Traffic Classification, pp. 123–147.

[47] VAN ADRICHEM, N. L. M., DOERR, C., AND KUIPERS, F. A. Opennetmon: Network monitoring in openflow software-defined networks. In *Proc. IEEE NOMS* (Krakow, Poland, May 2014).

[48] WANG, Y., ET AL. TeleScope: Flow-Level Video Telemetry using SDN. In *Proc. EWSDN* (The Hague, Netherlands, Oct 2016).

[49] YANG, T., JIANG, J., LIU, P., HUANG, Q., GONG, J., ZHOU, Y., MIAO, R., LI, X., AND UHLIG, S. Elastic Sketch: Adaptive and Fast Network-wide Measurements. In *Proc. ACM SIGCOMM* (Aug 2018).

[50] YU, C., ET AL. FlowSense: Monitoring network utilization with zero measurement cost. In *Proc. Springer PAM* (March 2013).

[51] YU, M., ET AL. Software Defined Traffic Measurement with OpenSketch. In *Proc. USENIX NSDI* (Berkeley, CA, USA, April 2013).

[52] ZHANG, Z.-L., RIBEIRO, V. J., MOON, S. B., AND DIOT, C. Small-time scaling beahviors of internet backbone traffic: An empirical study. In *Proc. IEEE INFOCOM* (San Francisco, CA, USA, Mar 2003).

Hassan Habibi Gharakheili received his B.Sc. and M.Sc. degrees of Electrical Engineering from the Sharif University of Technology in Tehran, Iran in 2001 and 2004 respectively, and his Ph.D. in Electrical Engineering and Telecommunications from the University of New South Wales (UNSW) in Sydney, Australia in 2015. He is currently a lecturer at UNSW Sydney. His research interests include programmable networks, learning-based networked systems, and data analytics in computer systems.

Minzhao Lyu received his B.Eng. degree with honours class 1 from the University of New South Wales, Sydney, Australia in 2017. He is currently pursuing Ph.D degree in the area of computer networks from the University of New South Wales and CSIRO's Data61. His research interests include software defined networking, network security and applied machine learning.

Yu Wang received his Bachelor's and Master's degrees in Electrical Engineering and Telecommunications from the University of New South Wales in Sydney, Australia in 2013 and 2017 respectively. His research interests include automated network technologies and software defined networking.

Himal Kumar received his B. Tech. from the Indian Institute of Technology in Patna, India, in 2013, his M.Phil. in Electrical Engineering and Telecommunications from UNSW in Sydney, Australia in 2017. He has worked at HPE as a student Intern. His research interests include Software Defined Networking, Internet frameworks, and network security.

Vijay Sivaraman received his B. Tech. from the Indian Institute of Technology in Delhi, India, in 1994, his M.S. from North Carolina State University in 1996, and his Ph.D. from the University of California at Los Angeles in 2000. He has worked at Bell-Labs as a student Fellow, in a silicon valley start-up manufacturing optical switch-routers, and as a Senior Research Engineer at the CSIRO in Australia. He is now a Professor at the University of New South Wales in Sydney, Australia. His research interests include Software Defined Networking, network architectures, and cyber-security particularly for IoT networks.