Context-driven Encrypted Multimedia Traffic Classification on Mobile Devices

Hoque, Mohammad Ashraful

IEEE
2022-04-27

Hoque , M A , Finley , B J , Rao , A , Kumar , A , Hui , P , Ammar , M & Tarkoma , S 2022 , Context-driven Encrypted Multimedia Traffic Classification on Mobile Devices . in 2022 IEEE International Conference on Pervasive Computing and Communications (PerCom) . IEEE International Conference on Pervasive Computing and Communications , IEEE , The 20th International Conference on Pervasive Computing and Communications , Pisa , Italy , 21/03/2022 . https://doi.org/10.1109/PerCom53586.2022.9762389

http://hdl.handle.net/10138/345971
https://doi.org/10.1109/PerCom53586.2022.9762389

acceptedVersion

Context-driven Encrypted Multimedia Traffic Classification on Mobile Devices

Mohammad A. Hoque*, Benjamin Finley*, Ashwin Rao*, Abhishek Kumar*, Mostafa Ammar†, Pan Hui‡*, and Sasu Tarkoma*§

* University of Helsinki, Finland. Email: firstname.lastname@cs.helsinki.fi

†Georgia Institute of Technology, Atlanta, USA. Email: ammar@cc.gatech.edu

‡Hong Kong University of Science and Technology, Hong Kong. Email: panhui@cse.ust.hk

§University of Oulu. Email: sasu.tarkoma@oulu.fi

***Abstract*—The Internet has been experiencing immense growth in multimedia traffic from mobile devices. The increase in traffic presents many challenges to user-centric networks, network operators, and service providers. Foremost among these challenges is the inability of networks to determine the types of encrypted traffic and thus the level of network service the traffic needs for maintaining an acceptable quality of experience. Therefore, end devices are a natural fit for performing traffic classification since end devices have more contextual information about the device usage and traffic. This paper proposes a novel approach that classifies multimedia traffic types produced and consumed on mobile devices. The technique relies on a mobile device's detection of its *multimedia context* characterized by its utilization of different media input/output components, e.g., camera, microphone, and speaker. We develop an algorithm, MediaSense, which senses the states of multiple I/O components and identifies the specific multimedia context of a mobile device in real-time. We demonstrate that MediaSense classifies encrypted multimedia traffic in real-time as accurately as deep learning approaches and with even better generalizability.**

***Index Terms*—multimedia context, deep learning, traffic classification, VoIP, streaming, broadcast.**

I. Introduction

Mobile devices have been generating 60% of all Internet traffic, and the most significant proportion of this traffic comes from multimedia applications that involve streaming and interactive video and audio. Along with the content from popular content providers, user-generated content is also on the rise due to online lectures from academic institutions and additional remote collaborations worldwide [1]. The Cisco Visual Network index predicts that traffic from various services such as video broadcast, live streaming, AR/VR applications would grow by five-to-seven fold by 2022 [2]. As the Corona pandemic continues, perhaps we have already reached such levels because millions of people are engaged in remote work, interactive online education, and consuming entertainment at significantly higher levels than before.

While ubiquitous devices enable a diverse set of multimedia activities [3], users generally have no control over their resulting traffic after such traffic leaves their device. Fortunately, a new set of personalized or user-centric networking applications are emerging, such as Personal Virtual Network (PVN) [4] and Middle Box Zero (MBZ) [5]. These virtual networks perform traffic inspection [6], and monitor network performance [7] on mobile devices. While these networks promise users more control over their traffic, they lack a privacy-aware traffic classification mechanism to assist in performing networking activities, such as performance monitoring, protecting privacy, and requesting QoS.

Existing approaches rely on root certificates to perform deep packet inspection on encrypted packets [8], [6] or certificate pinning to perform similar inspection [9], [10]. Naturally, these approaches have significant privacy concerns. Relatedly, deep learning algorithms that learn very application-specific features, such as Deep Packet [11] leverages application signatures from the initial SSL/TLS packets, raises privacy concerns (since the usage of a specific application can be considered private information). Furthermore, they require large training datasets and significant energy and hence are not optimal for mobile devices. Besides, an application can be responsible for different types of traffic.

In contrast, we propose an approach that classifies multimedia traffic into specific multimedia activity categories (such as streaming, broadcasting, and conversation) using a set of general (non app-specific) features. Our approach relies on a mechanism to identify such multimedia activities, which we call *multimedia contexts*. PVNs or MBZs can employ our approach to detect multimedia traffic types and then perform various optimizations (such as network selection, performance monitoring for different traffic types, traffic padding for preserving privacy, or route optimization for improved QoS) in a more privacy-preserving fashion.

A device's multimedia context describes whether the device user is producing content, consuming content, or conversing (thus both producing and consuming content) on a mobile device. We present a unique sensing algorithm, MediaSense, to accurately detect *eleven* such multimedia contexts of a device. MediaSense can be used to identify various multimedia traffic scenarios in real-time on mobile devices. MediaSense relies on the answers to the following questions: (i) what are the content types users interact with?, (ii) how do users interact with each type of content?, (iii) which I/O components are utilized during such interactions on smart devices, and (iv) what are the states of these I/O components while interacting with different content types?

Our key contributions are the following.

(1) Context Definition and MediaSense. To the best of our knowledge, we are the first to define multimedia contexts

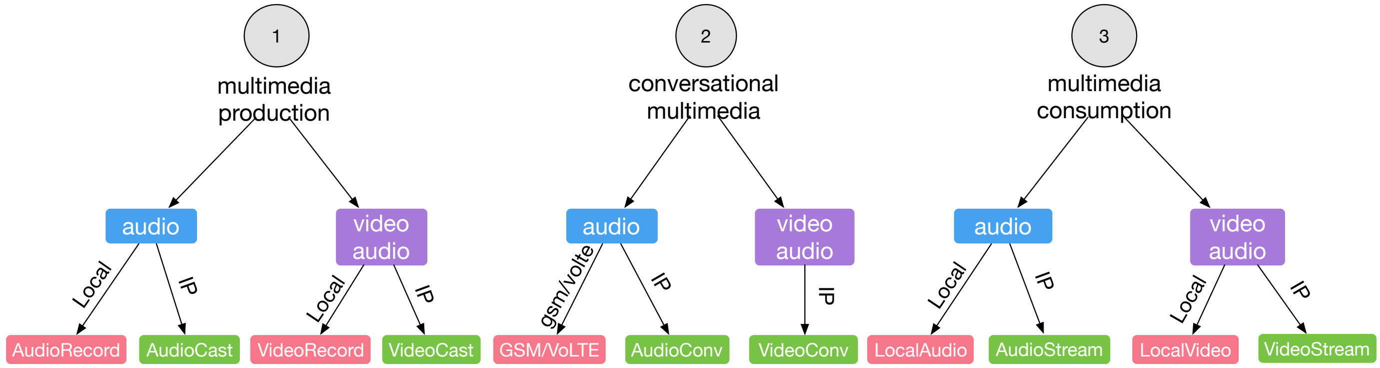


Fig. 1: Multimedia interaction types on mobile devices. *The second layer denotes the typical content types produced, consumed, or exchanged during the interaction. The third layer represents the context according to the multimedia type and their medium of use; 'local' implies media production and consumption without any IP communication.*

and propose a method to detect such contexts on mobile devices. We study *sixty-two* popular multimedia applications on Android and iOS devices and classify them according to how users interact with different multimedia contents using these applications (Section 2). From the investigation, we initially define *eleven* multimedia contexts. These contexts can be abstracted into three high-level multimedia contexts: (i) multimedia production, (ii) multimedia consumption, and (iii) conversational multimedia, as demonstrated in Figure 1.

Next, we explore how these contexts use several media I/O hardware components on mobile devices and present a multimedia context sensing algorithm called MediaSense. Through an extensive evaluation using over *52* applications, we demonstrate that with flow-level information MediaSense identifies the correct multimedia contexts with 97-100% accuracy. Furthermore, it identifies the corresponding voice/video over IP, live broadcast, and multimedia streaming network flows in real-time with an accuracy higher than 93% (Section IV) with negligible energy.

(2) Comparison with state-of-the-art approaches We further evaluate the performance of state-of-the-art deep learning approaches, such as 1D/2D Convolutional Neural Networks (CNNs) [12], [13], [11], for encrypted multimedia traffic classification (Section V). We capture traffic of the target multimedia applications and label them according to six IP-based multimedia contexts presented in Figure 1 and train the CNNs. Our evaluation shows that these approaches perform poorly or have inadequate generalization performance (e.g., to new applications in a multimedia context). In contrast, MediaSense is generic across different multimedia types and for new apps, and very energy efficient.

The rest of the article is organized as follows. We detail the contexts of various multimedia applications and their usage of I/O components in Section II. MediaSense is presented and evaluated in Sections III and IV respectively. Section V investigates the performance of CNNs for encrypted multimedia traffic classification and compares with MediaSense. Section VI highlights the potential use cases of MediaSense, and the related works are discussed in Section VII. The paper concludes in Section VIII.

TABLE I: Multimedia production contexts and the corresponding 19 Applications for Android and iOS devices. The *Periscope* service is discontinued from March 2021.

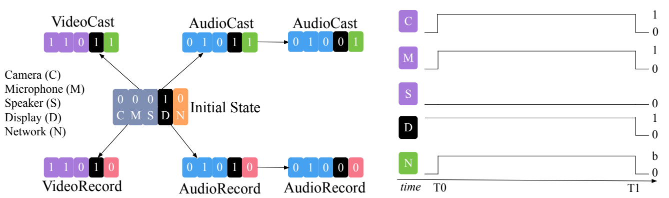
AudioRecord: Voice Memes(i), Voice Recorder(a), Dolby On(a), Hi-Q Recorder(a), RecForge II(a), ASR Recorder(a), Wear Recorder(a).
AudioCast: Mixlr(a/i), Spreaker(a/i).
VideoRecord: Open Camera(a), Camera(i), Dolby On(a), HD Camera(a), Camera MX(a), Camera360(a).
VideoCast: Periscope(a/i), StreamLab(a/i), BroadcastMe(a/i), Facebook Live(a/i).

II. Multimedia Applications and Contexts

This section develops a foundation for describing multimedia contexts determined by how applications use I/O components on mobile devices. We investigate the I/O component utilization of *sixty two* multimedia applications (see Tables I, II, and III) on Android and iOS devices, such as Nexus 6, LG G5, and iPhone 6/6s. Thirty-five applications are available on both Android and iOS devices, 17 are only available for Android devices, and ten are only available for iOS devices. All the required I/O components are typically initialized simultaneously depending on the application's characteristics. Since the user needs to launch an app via the touch screen, it is intuitive that the screen is busy. Therefore, the initial states of the I/O components for all the multimedia applications are the same. In Figures 2-4, we represent the status of the I/O components with '1' and '0', where the bits represent the busy and free status of the corresponding I/O components. In practice, a free I/O component can be physically off, e.g., as with a display. In contrast, the network I/O status does not represent the network interface status. Rather the status depends on network activities, such as the bit rates of the applications in terms of sending, receiving, or both.

A. Multimedia Production Contexts

A mobile device is in a production context when an application records audio/video on local storage or broadcasts live to some remote consumers. Voice Memos and Camera are the default audio and video production applications on iOS



(a) Media Production Contexts and I/O States (b) I/O states for Periscope live video broadcast

Fig. 2: The media production contexts and the corresponding I/O states. *(a) AudioRecord and VideoRecord refer to local (on-device) recording of audio and video, whereas AudioCast and VideoCast refer to live audio and video broadcast from a device using microphone and camera. (b) States of the I/O components while broadcasting live with Periscope.*

TABLE II: 32 applications/services of four multimedia consumption contexts for Android (a) and iOS (i) devices.

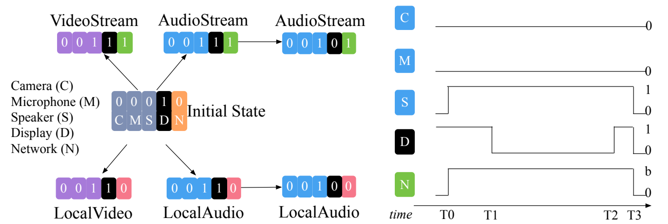
LocalAudio: Vox(i), Flacbox(i), Radsone(i), jetAudio(i), Stezza(i), Music Player Go(a), Poweramp(a), Omnia(a), Pulsar(a), VLC(a), AIMP(a).
AudioStream: Spotify(a/i), TuneIn(a/i), Tidal(a/i), qobuz(a/i), Idagio(a/i), ShoutCast(a/i), Soundcloud(a/i).
LocalVideo: VLC(a/i), MX Player(a/i), PlayerXtreme(a/i), KMPlayer (a/i), OPlayer Lite(i), 8Player (i).
VideoStream: YouTube (a/i), Vimeo(a/i), Dailymotion(a/i), HBO(a/i), Netflix(a/i), Twitch(a/i), Prime Video (a/i), Periscope(a/i).

devices. Similarly, Android devices have built-in microphone and camera applications. In addition to these applications, Mixlr, Periscope, and StreamLab are popular live audio and video broadcasting applications as presented in Table I. By investigating these applications, we derive four media production contexts emerging from the initial state as shown in Figure 2(a). Since video recording typically requires users' attention, the application must ideally be in the foreground while recording. If the user switches to another app or turns off the display, the camera and microphone become free. In contrast, audio recordings and live audio broadcasts can continue in the background. Thus, the states of I/O components for the recording applications differ from those of the live broadcasting applications only by the network (as they do not transmit).

Figure 2 (b) shows the states of the I/O components during a live video broadcasting session of Periscope. We observe that the live broadcast begins at $T0$, and the application initializes the camera and microphones. The output component, the display, is also used, and data transmission begins. The broadcasting terminates at $T1$. Periscope also initiates TCP connections. We observed that the uplink bit rates of Periscope and Mixlr are 459 kbps and 128 kbps for broadcasting live video and audio, respectively.

B. Multimedia Consumption Contexts

A mobile device is in a consumption context, when an end-user plays multimedia content from local storage or streams from a remote service provider.



(a) Media consumption Contexts and I/O states (b) I/O states for TuneIn Audio Streaming

Fig. 3: The media consumption contexts and the states of the I/O components. *(a) LocalAudio and LocalVideo refer to audio and video playback from the local (on-device) storage, whereas AudioStream and Video Stream refer to streaming audio and video from remote services. (b) States of the I/O components while streaming audio with TuneIn.*

Both Android and iOS devices have default applications for audio/video playback from local storage. In addition, we investigated popular streaming applications, such as YouTube, TuneIn, Periscope live streaming, and many others presented in Table II. Through this exploration, we derive four media consumption contexts, as shown in Figure 3(a). For watching videos from local storage or video streaming, display and speaker are mandatory; i.e., the playback stops when the user switches to another application. In contrast, audio applications can still play audio while in the background. Furthermore, streaming applications download content from a remote server and thus require IP connectivity. Despite the diverse set of multimedia streaming applications, such as on-demand streaming and live/pseudo live streaming, we observe that audio and video consumption exhibit the same I/O states. Overall the media consumption applications vary significantly: some apps play with negligible initial playback delay, whereas some continue to cache and depend on the user's input.

Figure 3(b) shows that only speaker and network activities begin when a TuneIn audio streaming session starts at $T0$. The display is turned off at $T1$ and turned on again $T2$. Finally, the user terminates the streaming session at $T3$. TuneIn initiates multiple TCP flows as soon as the playback begins. The bitrates of the TuneIn audio streams vary from 24 kbps to 320 kbps. We also observe that the downlink bit rate of Periscope, i.e., the bit rate when viewing a Periscope live stream of other Periscope users, is similar to the uplink bit rate. Along with the application-specific bit rates, we also observed different ON/OFF patterns in the network traffic [14].

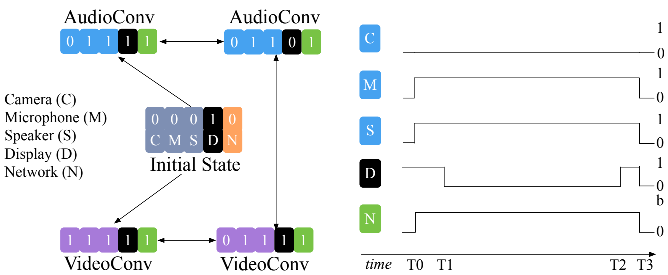
C. Conversational Multimedia Contexts

A device is in a conversation context when a user engages in an audiovisual conversation with another remote user using a conversational application.

In addition to GSM and VoLTE calls, we also experiment with the popular VoIP applications presented in Table III. Figure 4 (a) shows that the states of the I/O components change according to the conversation types. An audio conversation

TABLE III: Conversational contexts and 11 applications on Android (a) and iOS (i) devices. Only VoLTE/GSM calls are responsible for non-IP AudioConv context.

AudioConv: WhatsApp(a/i), IMO(a/i), Viber(a/i), Kakao Talk(a/i), Line(a/i), Skype(a/i), Messenger(a/i), Duo(a/i), VoLTE/GSM(a/i), Snapchat (a/i).	
VideoConv: WhatsApp(a/i), IMO(a/i), Viber(a/i), Kakao Talk(a/i), Line(a/i), Skype(a/i), Messenger(a/i), Duo(a/i), Snapchat (a/i), FaceTime (i).	



(a) Conversation contexts and I/O states (b) I/O states for WhatsApp audio call

Fig. 4: The conversation contexts and the states of the corresponding I/O components. *(a) AudioConv and VideoConv represent Audio and Video call contexts respectively. (b) States of the I/O components during a WhatsApp audio conversation.*

does not need the display to be active during the call. In contrast, a video call uses all the media I/O components.

Conversational applications have two media contexts, i.e., audio or video conversations. Figure 4(b) demonstrates that a WhatsApp VoIP call begins at $T0$, and all the I/O components become busy, except the camera. The display turns off at $T1$ and turns on again at $T2$. Finally, the call terminates at $T3$. WhatsApp initiates both TCP and UDP flows as soon as the call begins. The TCP flows are mostly used for signaling, and UDP flows carry the media. The bitrate of the audio flow in each direction ranges from 17-22 kbps, and the bitrate increases to a few hundred kbps during video conversations.

A video conversation can also proceed without an active camera on either the caller's or callee's device. When the caller initiates a video call, all the media sensors are activated on the callee's device. If the caller turns off the camera after the call is established, the user still needs to keep the display active, as the caller device receives video from the other end. When both users turn off their cameras, the media context changes to an audio call (Figure 4).

D. Mixed Multimedia Contexts

Multiple applications cannot utilize some I/O components simultaneously. We also explore these limitations. Table IV demonstrates our findings when using applications of two media contexts together on single screen devices, iPhone 6s and Nexus 6. The table shows that a user cannot use two media production applications simultaneously on a single screen device. The user also cannot record audio and video content simultaneously on two separate applications. We also tried to

TABLE IV: Mixed contexts on Android and iOS devices.

Multimedia context	**Media Production**	**Conversation**	**Media Consumption**
Media Production	No	No	No
Conversation	No	No	Yes
Media Consumption	No	Yes	Yes

Algorithm 1: MediaSense

▷ **Comment 1:** Pre-computed features.;
mediaFeatures = Map(mediaContext, bitrate);
while *true* **do**
  trafficstat = getTXRXbytes();
  mic = sampleMicrophone() $\in \{1,0\}$;
  speaker = sampleSpeaker() $\in \{1,0\}$;
  camera = sampleCamera() $\in \{1,0\}$;
  display = sampleDisplay()$\in \{1,0\}$;
  ▷ **Comment 2:** Audio/Video media contexts.
  mediaContext = camera|mic|speaker|display;
  media = camera|mic|speaker;
  T_{media} = gettimeoftheday();
  ▷ **Comment 3:** IP-based media contexts
  if $(mediaContext(!network))$ **then**
    mediaVec = computeFeatures(trafficstat, mediaContext);
    mediaFet = getFeatures(mediaContext, mediaFeatures);
    if *(mediaVec $\approx$ mediaFet)* **then**
      mediaContext = mediaContext|network;
    end
  end
  ▷ **Comment 4:** Updating Video to Audio consumption.
  if $((mediaContext == VideoStream)\&\&(!display))$ **then**
    mediaContext = mediaContext|(!display);
  end
  ▷ **Comment 5:** Voice/Video call state changes.
  if $((mediaContext == VideoConf)\&\&(!camera))$ **then**
    mediaContext = mediaContext|(!camera);
  end
  if *(media==0)* **then**
    ▷ **Comment 6:** MediaContext duration.
    $Media_{session} = gettimeoftheday() - T_{media}$
  end
end

record audio via the default voice recording application, and the recorder stops as soon as Periscope starts broadcasting.

Figures 2 & 3 show that media production and consumption applications do not necessarily overlap in terms of requiring the camera, microphone or speaker. However, the Nexus 6 does not allow recording or broadcasting media when another streaming application is running. For example, running an audio recording application in the background and streaming via YouTube or TuneIn in the foreground is impossible. In this case, the audio recording terminates as soon as the player initializes the required sensors. Similarly, the user cannot use recording applications during a VoIP call. Nevertheless, we were able to stream audio in the presence of a VoIP call on a single screen.

TABLE V: Android APIs for detecting media contexts and utilization of Media I/Os.

Android API	I/O component	User Permission
AudioManger.getMode()	Micrphone, Speaker	No
AudioManager.getMode(), CameraManager. registerAvailabilityCallBack()	Camera, Microphone, Speaker	No
AudioManager.isMusicActive()	Speaker	No
CameraManager. registerAvailabilityCallBack()	Camera, Microphone	No
MediaRecorder.record()	Microphone	Yes

E. Summary

In this section, we have investigated many multimedia applications for mobile devices. We have shown that the utilization of the I/O components by multimedia applications can be generalized across both Android and iOS devices. This requirement of I/O components also allows us to extend the generalization to an arbitrary number of applications for media consumption, production, or conversation.

At the first level (Figure 1), the distinct usage of the microphone and speaker components differentiates the production, conversational, and consumption contexts. While at the second level, the camera separates video from audio contexts for production and conversational media, whereas the display separates video from audio for media consumption. All the contexts are separated at the third level according to network activities, i.e., transmitting, receiving, or exchanging traffic.

Note that augmenting a phone with a headset via Bluetooth or cable does not affect the state of the I/O components; it only changes the route of the audio signal. Thus, being outdoors, indoors, or mobile does not change the need for these I/O components. This also applies to adjusting brightness, camera focus, or adjusting volume. However, loss of signal or poor signal may disrupt a VoIP call, streaming session, or a live broadcast and may terminate the media context.

III. MediaSense

Given that a user interacts with an arbitrary multimedia application, we devise MediaSense (Algorithm 1) that scans the states of five I/O components to infer the resulting multimedia context. We implement MediaSense as a user-level service for Android devices. The service runs as a background service and looks for multimedia contexts periodically at 1Hz. Whenever one or more I/O components changes states, MediaSense initiates a new multimedia context. The algorithm first checks whether the media context is audio or video-related with the camera and display (Comment 2 in Algorithm 1). Then, the algorithm checks the traffic flow statistics in the uplink and downlink to separate IP-based contexts from local media (Comment 3 in Algorithm 1). We describe these steps in detail below.

A. Separating Audio/Video Contexts

The algorithm periodically scans the states of the I/O components using on-device system APIs to determine the media context.

(1) Conversational Multimedia Context. Fortunately, Android provides APIs for applications to indicate their modes of operation to the AudioManager [15], thus allowing other applications to query the status of the AudioManager via getMode() API. AudioManager operates in one of three modes; IN_CALL, IN_COMMUNICATION, and RINGTONE. These modes indirectly indicate that an ongoing context is conversational and that both the microphone and speaker are busy. TelephonyManager has getCallState() to indicate a GSM/VoLTE call [16] and thus expresses the states of the microphone and speaker.

Table V summarizes the mapping between Android APIs and the corresponding media I/O components. MediaSense characterizes a context as a video conversation if the AudioManager is in one of the modes and one of the cameras is initialized at the same time. MediaSense implements registerAvailabilityCallback() from CameraManager [17] to poll camera status, i.e., available/unavailable, exactly when the audio mode changes.

(2) Multimedia Consumption Contexts. The isMusicActive() API from AudioManager helps to differentiate music playback contexts from VoIP/GSM calls or other media production contexts on the device. This API provides the speaker information. However, the API does not differentiate whether the playback is audio or video. In other words, it could be an audio-only application or a video application that uses the speaker for the audio track. We also could not find APIs hinting about streaming. In Figure 3, we notice that distinguishing between audio and video consumption contexts is not straightforward, given just the statuses of the I/O components. The reason is that audio applications require only speakers and can work while keeping the display active or inactive. Therefore, the algorithm first assumes a media consumption context is video type. Then when the display turns off, the media context is changed to audio type as the media session continues.

(3) Multimedia Production Contexts. MediaSense uses APIs which do not require explicit user permission to detect the earlier described media contexts. Similarly, the algorithm uses registerAvailabilityCallback() from CameraManager to detect the video production contexts from the Camera or Periscope like applications. This API initializes the camera and microphone together. However, detecting the state of the mic is not possible without explicit user permission. MediaSense implements MediaRecorder APIs with user permission to detect the audio production contexts due to the applications presented in Table I.

B. Separating Local/IP-based media contexts

MediaSense distinguishes IP-based contexts from the local media context by identifying the traffic flows from a set of live flows using flow-level information, as presented in Table VI.

TABLE VI: Features considered for the identifying the media contexts. "faststartbyte" denotes the amount of data downloaded during the first 10 seconds of streaming.

uplink bitrate	downlink bitrate	Features ? Context
✓	X	if (bitrate$\geq$8 kbps, microphone) ? AudioCast : AudioRecord
✓	X	if (bitrate $\geq$8 kbps, microphone, camera) ? VideoCast : VideoRecord
✓	✓	if (bitrate $\geq$8 kbps, microphone, speaker) ? AudioConv : GSM/LTE
✓	✓	(bitrate $\geq$8 kbps, microphone, speaker, camera) ? VideoConv
X	✓	(faststartbytes $\geq$100 KB, bitrate>300kbps,speaker, display) ? VideoStream : LocalVideo
X	✓	(faststartbytes $\geq$100 KB, bitrate<300kbps, speaker, !display) ? AudioStream : LocalAudio

It relies on the Android VPN API to gain access to such information from live flows. There is no other way to access network traffic information on mobile devices. This API was also used by Lumen [18], Mopeye [7], and AntMonitor [19]. Unlike these, MediaSense neither installs root certificates nor performs deep packet inspection. The algorithm uses five tuples as the flow identifier. MediaSense does not compute all the features in the table for a media context. It rather computes media context-specific features. These features are derived from our observations in Section II with the following reasoning.

(1) Conversational Multimedia Contexts. Unlike the other applications, conversational traffic carries voice or video data in both directions. Voice traffic has a minimum bit rate of 14 kbps. However, the bitrates of these applications depend on the underlying codec [20], which can be as low as 8 kbps in one direction [21]. A GSM/VoLTE call be identified using the getMode() API and by noting that there is no uplink and downlink traffic (see Table VI).

(2) Multimedia Consumption Contexts. On-demand streaming applications, e.g., YouTube, Spotify, begin streaming with a fast start phase and download 10-40 seconds of equivalent playback content. Spotify streams audio via persistent HTTP connections over TCP, regardless of the device type [22]. The audio streams are encoded at 96-360 kbps, and the selection depends on the subscription type. The size of the first segment is 139.53 KB [22]. YouTube downloads more than one megabyte during the fast start phase. Periscope downloads content at a constant bit rate after the fast start. Therefore, MediaSense relies on isMusic() API and the flow features presented in Table VI to separate streaming contexts from local music playback.

(3) Multimedia Production Contexts. Periscope's outgoing bitrate is 459 kbps. A 64 kbps outgoing bitrate is very common for live audio broadcasting. Mixlr supports 32-128 kbps. We consider a lower bound of 8 kbps as the context feature. Overall MediaSense uses the bitrate feature along with the CameraManager & MediaRecorder APIs to separate the IP-based based contexts from the local recording contexts.

IV. Performance Evaluation

We installed MediaSense on an LG G5 (Android 8.0) with an LTE connection for evaluation. The data plan allowed a maximum of 45 Mbps and 20 Mbps speed for downlink and uplink, respectively. The device had 101 applications installed, including 52 multimedia apps. However, we interacted with only one multimedia application at a time, and the session duration remained between 30-60 seconds. We denote this as a media session or the duration of a media context. The device was fully charged during the experiments to avoid any system-assisted performance degradation [23].

We evaluate MediaSenseon how accurately it can differentiate local versus IP-based media contexts in the media sessions and how accurately it can identify the corresponding network flows. We estimate flow detection accuracy as:

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \quad (1)$$

We define the proportions in the equation as follows. True positive (TP) denotes the number of cases correctly identified as IP-based contexts. False positive (FP) denotes the number of cases incorrectly identified as IP-based contexts. True negative (TN) denotes the number of cases correctly identified as local contexts. And False negative (FN) denotes the number of cases incorrectly identified as local contexts.

The accuracy in identifying an IP-based context is expressed by the precision, which estimates the proportion of true positives in the corresponding sessions. This can be stated as:

$$Precision = TP/(TP + FN) \quad (2)$$

In contrast, the recall expresses the local context accuracy, which estimates the proportion of true negatives in the IP-based context that resemble local media contexts. This can be stated as:

$$Recall = TN/(TN + FP) \quad (3)$$

For network flow identification or traffic classification performance, we only use the *accuracy* measure.

TABLE VII: MediaSense correctly identified 100% (all true positives) of multimedia sessions and network flows for audio/video broadcasts during the first 10 seconds of broadcasts.

VideoCast				AudioCast			
App	TP/ #sess	TP/ #flow	bitrate-mbps	App	TP/ #sess	TP/ #flow	bitrate-kbps
Periscope	53/53	53/53	0.4-0.7	Mixlr	76/76	76/76	16-128
Streamlabs	39/39	39/39	0.5-1.5	Spreaker	20/20	20/20	16-128
BroadcastMe	26/26	26/26	0.2-0.7	-	-	-	-
FacebookLive	29/29	29/29	0.6-1.4	-	-	-	-

A. Multimedia Production Contexts.

We first experiment with the multimedia production applications presented in Table I.

Local/non-IP production context identification. There are 220 sessions for 11 local media production applications. MediaSense did not identify any media network flows during these local media production contexts. In other words, MediaSense correctly identified all the AudioRecord and VideoRecord sessions without any FNs, and therefore, the recall is 100%.

IP-based production context identification. We experimented with six applications. The bitrates of the audio broadcasts varied between 16 and 128 kbps. Periscope and BroadcastMe had the average bitrates for medium quality videos, whereas Streamlabs and Facebook Live transmitted high definition videos. Network traffic should be generated by the broadcasting applications when the media context is initiated or later on based on user interaction with the application. MediaSense correctly identified all 243 media contexts as presented in Table VII. Consequently, there were no FPs while detecting the broadcast contexts, and the precision is 100% for both AudioCast and VideoCast.

Traffic Classification. The very high precision in detecting IP-based media contexts (see Table VII) also indicates that MediaSense detects the relevant flows. However, it can generate TPs (detecting the actual flow), FPs (detecting a wrong flow), or FNs (not detecting a flow). Note that TNs are not possible for flow detection. Therefore, we use eq. (1) to determine the flow detection accuracy. Table VII shows that MediaSense identified *243* audio/video live broadcast flows during *243* sessions of Mixlr, Spreaker, Periscope, Streamlabs, BroadcastMe, and Facebook Live. There were no FP or FN, and MediaSense is 100% accurate in identifying the broadcasting flows.

TABLE VIII: MediaSense's performance in identifying AudioStream contexts and network flows real-time (display-off).

AudioStream						
	Sessions		Flows			
App	TP	FN	TP	FP	FN	bitrate-mbps
TuneIn	52	0	47	5	0	0.03-0.44
ShoutCast	43	0	38	5	0	0.03-0.44
Qobuz	50	0	49	1	0	0.6-4.1
Idago	39	0	37	2	0	0.5-5.0
Tidal	33	0	31	2	0	1.2-5.2
Spotify	29	0	26	3	0	2.5-7.5
SounCloud	23	0	22	1	0	0.8-4.5

B. Multimedia Consumption Contexts.

Media consumption contexts relate to live streaming, pseudo-live streaming, or on-demand streaming applications and other local playback applications (see Table II). In the streaming cases, since the content is first downloaded and then played, there is an initial playback delay of 3-5 seconds [14]. Therefore MediaSense considers this absolute time difference, i.e., $|t_{media} - t_{flow}|$, in filtering the flows. In addition to display status, we also consider 16-300 kbps bitrates to indicate audio streams and higher bitrates to indicate video streams. The absence of network flows with such features indicates a local media consumption context.

Local/non-IP context identification. We did not observe FNs for the local audio and video playbacks from the ten applications, as MediaSense associates the flow initiation time with context beginning and considers the flow bitrates. Therefore, it identifies 200 local audio/video playback contexts from 10 applications with 100% recall.

IP-based video consumption context and Traffic Classification. Table IX shows that MediaSense correctly identified 97% of 357 VideoStream contexts from eight applications. Three YouTube, four Periscope, and three Dailymotion sessions were identified as AudioStreams, as the bitrates were below 300 kbps. Likewise, MediaSense correctly detected 95% of the multimedia flows during VideoStream contexts as shown in Table IX. However, MediaSense incorrectly detected additional flows in ten sessions (FPs), including during Dailymotion and Periscope contexts. MediaSense also revealed that Twitch, Vimeo, and Prime Video use multiple flows to download content. More than 70% of the 48 Twitch sessions and all the 47 Vimeo sessions had two flows, and 36 Prime sessions had four flows per session. Table IX shows that MediaSense identified their network flows with 97% accuracy.

IP-based audio consumption context and Traffic Classification. Table VIII shows that MediaSense identified 269 AudioStream contexts from seven applications with 100% accuracy as there were no FNs. However, it identified some of the corresponding network flows as FPs. The FPs occurred due to unrelated background flows during streaming. Likewise, MediaSense detected the corresponding network flows with 93% accuracy due to some false positives resulting from background flows.

TABLE IX: MediaSense's performance in identifying VideoStream contexts and network flows real-time.

VideoStream						
	Sessions		Flows			
App	TP	FN	TP	FP	FN	bitrate-mbps
YouTubeLive	49	3	52	0	0	0.35-4.7
Periscope	48	4	47	5	0	0.2-3.01
Twitch	53	0	53	0	0	0.6-6.3
Vimeo	47	0	47	0	0	0.4-6.2
Dailymotion	42	3	38	5	0	0.35-11
Netflix	43	0	43	0	0	3.5-29
Prime Video	36	0	36	0	0	0.35-4.7
HBO Nordic	29	0	29	0	0	0.5-9.1

TABLE X: MediaSense identified audio and video conversation contexts as 100% true positives.

Apps	**VideoConv**			**AudioConv**		
	TP/ #sess	TP/ #flow	bitrate-mbps	TP/ #sess	TP/ #flow	bitrate-kbps
WhatsApp	50/50	50/50	0.3-0.51	50/50	50/50	10-23
Skype	50/50	50/50	0.4-0.8	50/50	50/50	43-73
Viber	50/50	50/50	0.9-2.2	50/50	50/50	10-18
IMO	50/50	50/50	0.3-0.7	50/50	50/50	14-18
Kakao	28/28	28/28	0.3-0.5	22/22	22/22	10-24
Duo	26/26	26/26	0.35-1.8	26/26	26/26	40-55
Messenger	25/25	25/25	0.4-1.1	23/23	23/23	8-23
Snapchat	25/25	25/25	0.3-0.7	25/25	25/25	14-18
Line	25/25	25/25	0.3-0.7	25/25	25/25	10-40

C. Conversational Multimedia Contexts.

We investigated the performance of MediaSense with all the conversational applications in Table III. Since the delay requirement of conversation multimedia is very strict, MediaSense considers the flows initiated within three seconds of the media contexts, i.e., $|t_{media} - t_{flow}| \leq 3$s.

Local/non-IP context identification. Conversational contexts have data exchange in both directions. However, GSM and VoLTE data do not follow the same path as normal application data. Therefore, the absence of flows with at least eight kbps bitrates in both directions indicates an ongoing GSM/VoLTE call. On LG-G5, the algorithm identified 20 conversational sessions due to 20 GSM calls without any FN; thus, the recall is 100%.

IP-based context identification and Traffic Classification. MediaSense identified 329 AudioConv and 329 VideoConv contexts with similar features with 100% precision on LG G5, as shown in Table X. There were no FPs. Although there could be rate control by these applications [24], [25], an 8 kbps bitrate (and an active camera) are sufficient for detecting video calls and flows. A conversational context can switch to a hybrid mode when the camera is off/on during a conversation. Consequently, MediaSense also accurately identifies such hybrid contexts via camera status and flow bitrate features. Similarly to the audio/video broadcasting flows, MediaSense identified 658 conversational flows with 100% accuracy without any FP or FN (see Table X).

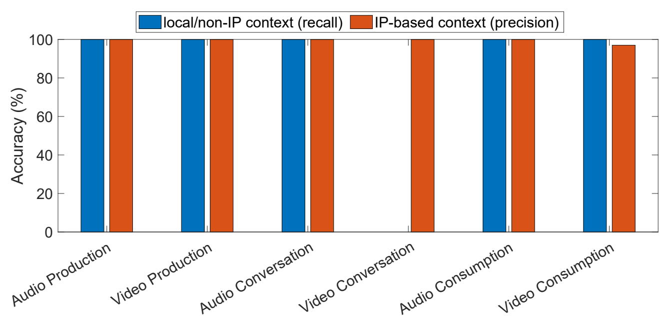


Fig. 5: MediaSense performance in identifying the 11 multimedia contexts from Figure 1. Note that the local/non-IP context for audio conversation represents GSM and VoIP calls.

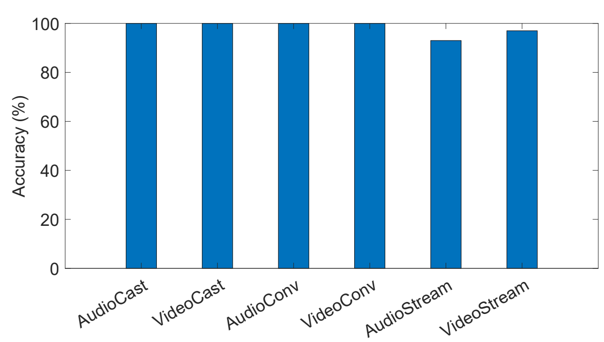


Fig. 6: Real-time flow classification performance of MediaSense for different IP-based multimedia contexts.

D. Discussions and Energy Consumption

We further measured the electric current consumption of MediaSense on the Nexus 6. Although Power Monitor [26] would provide better estimates, modern mobile devices come with difficult-to-access batteries, and thus, instrumenting these devices is very challenging. We instrumented MediaSense with Android API to sample the run time current consumption at 1 Hz and tested on the Nexus 6. The device had 101 applications installed, including the 52 multimedia applications connected to a WiFi access point. The device was fully charged during the measurements. MediaSense consumes 70 mA on average while the device is idle. Table XI compares the average current consumption of the Nexus 6 for nine applications with eleven media contexts in the absence (baseline) and presence of MediaSense. During the video contexts, the display was ON, and the front camera was used for the VideoConv and VideoCast contexts. The display was off during the audio contexts. MediaSense computes flow statistics, tracks media contexts, and associates contexts with the flows. We notice MediaSense does not consume considerably more energy compared to the baseline.

TABLE XI: The average current consumption of Nexus 6 over the periods 60 seconds for the multimedia applications.

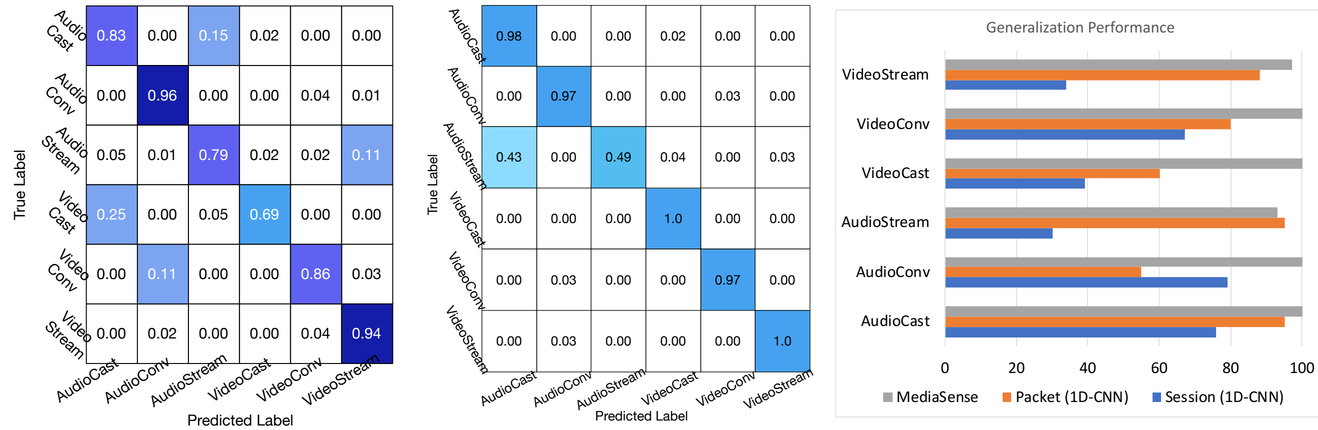
Application	Context	Basline (mA)	MediaSense (mA)
Voice Recorder	AudioRecord	110	119
Camera	VideoRecord	700	800
MixIr	AudioCast	140	139
BroadcastMe	VideoCast	439	440
TuneIn	AudioStream	140	200
YouTubeLive	VideoStream	525	540
WhatsApp	AudioConv	222	231
Phone	AudioConv(GSM)	135	155
WhatsApp	VideoConv	743	700
VLC	LocalVideo	545	579
VLC	LocalAudio	130	122

V. Performance Comparison

In this section, we compare the performance of MediaSense with state-of-the-art deep learning methods for traffic classification [12], [13], [11].

A. Deep Learning Approaches

We evaluate the performance of session and flow-level 1D and 2D Convolution Neural Networks (CNN)s and a packet-level 1D-CNN to classify encrypted multimedia traffic. The session and flow-level 1D-CNN models have two 1D convolution layers with 32 and 64 filters, respectively. Each convolution layer is followed by a 1D max-pooling and terminated by two fully connected layers. The session and flow-level 2D-CNN model is constructed by replacing 1D convolution and pooling layers with the corresponding 2D layers. Further details of the models can be found in [12], [13]. In contrast, the packet-level 1D-CNN has two 1D convolution layers also each followed by 1D max-pooling and a final three fully connected layers. Further details of this model can be found in [11]. These deep models train on actual byte data (as we discuss further), thus flow, or packet-level feature engineering is not required.



(a) 1D-CNN (Flow)

(b) 1D-CNN (Packet)

(c) Generalization Performance

Fig. 7: Confusion Matrices for Session and Packet Level 1D-CNN Models and Comparison with MediaSense.

B. Dataset & Training the Networks

The total size of the training dataset is 6.7 GB in *pcap* format. It contains the traces for the subsets of applications of six IP-based classes (contexts) discussed in the previous sections. A traffic session is defined by a 5-tuple (source IP, source port, destination IP, destination port, and transport protocol). A flow is similar and considers traffic direction (so the IP and port are not reversible).

We use 80% of total flows or sessions for the flow and session-level models to train the 1D and 2D-CNN models and the remainder for testing. Each flow or session is represented (to the model) by the first 784 bytes of the PCAP file containing only that session or flow (as in the original model [12]) and thus includes the first few packets of the flow or session along with packet metadata (which is part of the PCAP file format) such as their capture timestamps.

Whereas for the packet-level model, we also use 80% of the packets for training and the remainder for testing. Each packet is represented (to the model) by the first 1500 bytes of the packet (with zero-padding if necessary). Additionally, we need to consider the high-class imbalance due to the significant differences in packet volumes for different multimedia classes (e.g., video vs. audio). Therefore, we perform random undersampling of the training data to equalize the class frequencies. In contrast, the testing dataset is not undersampled to maintain a realistic evaluation. Finally, we provide the complete confusion matrix results, as only accuracy values can be misleading [27].

For all models, we additionally test model generalization concerning new apps in categories by testing with different apps in the training data and the testing data (e.g., for video streaming, we have Netflix and Vimeo in the training data and YouTube in the testing data). In this additional test, for the session-level model, we also mask the IP address due to concerns that this can unfairly imply or leak app identity [13].

We train the CNN models using Tensorflow [28] for the flow and session-level models and PyTorch for the packet level model on a Linux server with an Nvidia Tesla K80 GPU. In terms of metrics, we train the flow and session-level models for about 30 epochs until the performance levels, and similarly, we train the packet-level model for five epochs.

C. Evaluation

Figures 7a, and 7b illustrate the test confusion matrices for the 1D-CNN session and packet-level model respectively. We omit the flow-level and 2D-CNN model results because their performance is similar to those presented in the figure. The results show that all the deep learning approaches achieve reasonable accuracy (whether on a flow or packet level). However, the session-level methods have difficulty distinguishing VideoCast from AudioCast sessions as both contexts resemble constant bitrate traffic and have similar packet header attributes. In contrast, the packet-level approach faces difficulty distinguishing AudioStream sessions from AudioCast sessions.

Furthermore, in Figure 7c, we find that the generalization performance for both deep learning models is significantly worse compared to the base evaluations (from Figures 7a and 7b) and MediaSense. Specifically, we find that the accuracies of 1D-CNN session-level model for categories are 76%, 79%, 30%, 39%, 67%, and 34% (compared to 89%, 96%, 79%, 56%, 89%, 95% for the base evaluation) for a new application. Likewise, the 1D-CNN packet-level model the accuracies are 95%, 55%, 95%, 60%, 80%, and 88% (compared to 98%, 97%, 49%, 100%, 97%, 100% for the base evaluation). This suggests that these models are learning app specific features, for example from the first few bytes of the SSL/TLS exchange, rather than the more general category specific features of multimedia traffic. Ref. [13] also notes similar findings for the packet-level model.

Overall, by determining and associating contexts, MediaSense identifies such traffic with similar or higher accuracy and better generalization (for new apps) on mobile devices without any training as demonstrated in Figure 6, and with negligible energy cost.

VI. Discussion

There are many potential uses of information about a device's multimedia context. In this work, we demonstrated how multimedia context information could be used to classify encrypted multimedia traffic in real-time on mobile devices. Desktop computers, laptops, and other handheld devices have similar I/O components. Therefore, MediaSense can be generalized across such devices as well.

Mobile multimedia contexts further can be used to judicially select the radio network interface (WiFi/LTE) on multi-homing devices. Specifically, on-device traffic conditioning according to signal strength and multimedia context can improve the quality of experience. The CPU governor can plug/unplug the cores and scale CPU frequency according to the context. Similarly, the devices can schedule tasks according to the media context. Any user application can use such contexts for energy-aware scheduling of background traffic [29], [30].

Different stakeholders in the networks also can benefit from MediaSense. In 5G, the access networks classify traffic and send the QoS policies to mobile devices for applying [31]. The policies include dropping packets, routing packets according to IP/MAC address, and marking flows with Differentiated Service Code Points (DSCPs). MediaSense is a practical approach that can be used to extend DiffServ on mobile devices to complement the QoS architecture in 5G. As a result, the network operators do not have to perform multimedia traffic classification, as the traffic can be marked even before arriving at the access network. Furthermore, the operators can map the DSCP marked traffic to core or radio network slices [32]. The network service providers can further use such information for billing, network planning/provisioning, and security [33].

VII. Related Work

Identifying multimedia traffic is essential. Significant works have been done that are most suitable for network service providers to manage their networks. Different traffic classification methods can be divided into three categories.

Deep Packet Inspection. The MIMIC system [34] looks into HTTP logs of the adaptive streaming requests to estimate the average bitrates, bitrate switch, and the playback buffer status. Similarly, BUFFEST [35] investigates the HTTP logs. Lumen [8], and VPNGuard [6] investigate the encrypted packet payloads with the help of a VPN service and using custom root certificates. Several approaches investigated the codec formats of encrypted VoIP packets from Skype [36]. However, modern multimedia services, such as Periscope, and Netflix, communicate over HTTPS [37]. Therefore, the traditional port-based classification techniques do not work, and deep packet inspection [38] is difficult given the encryption.

Statistical Methods. Statistical methods rely on flow features, such as packet size distribution, packet gap, burstiness, and packet headers [39]. These features can be used to understand VoIP applications' traffic patterns, such as Skype [40]. Bonfiglio *et al.* [36] first looked into the statistical properties of message content and then matched with the Skype voice traffic sources by using Naive Bayesian techniques. Do and Branch used inter-packet gaps to classify VoIP traffic in real-time [41]. However, the flow features can vary with speech codecs and ambient noise [42].

Machine Learning. We have already evaluated two deep learning approaches in Section V using packet [11] and flow [12], [13] level features. Some recent studies identified YouTube videos of different qualities from the encrypted traffic by modeling the relationship between burstiness, i.e., chunk size and gaps, and videos' quality [43]. The basic flow features can be used by machine learning algorithms, such as K-NN clustering, to classify encrypted multimedia traffic [44]. Kim et al. [45] showed that Support Vector Machine achieves more than 98% accuracy with less training data than other machine learning algorithms.

Summary. The deep packet inspection methods are difficult on encrypted traffic. The various machine learning approaches require retraining the models, large training data, and energy consumption as the traffic pattern changes. In contrast, MediaSense is specifically for user-centric networks on end-devices. It performs much better with the help of media contexts. MediaSense is as accurate as of the existing deep learning approaches as demonstrated in Section V.

VIII. Conclusions

This article introduces the multimedia context concept and presents a novel algorithm to identify the corresponding encrypted multimedia traffic on mobile devices in real-time. MediaSense computes and leverages the media context-specific flow features for finding and identifying the corresponding multimedia flows. The approach is energy efficient and can generalize across multimedia applications without training. MediaSense is also privacy-preserving, as it neither infers the application nor examines the actual packet data nor leaks the contexts. MediaSense opens the door for context-aware system and traffic optimization on mobile devices.

Acknowledgment

The work has been supported by the Academy of Finland IDEA-MILL project (Grant Number 335934), the Academy of Finland 5GEAR project (Grant Number 319669), and the Academy of Finland FIT project (Decision No. 325570). In addition, the authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources. Finally, Mostafa Ammar's work was partially supported by NSF grant NETS: 1909040.

References

[1] Anja Feldmann and Others. The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 1–18, New York, NY, USA, 2020. ACM.

[2] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2016−−2021. Technical report, 2017.

[3] Mohammad A. Hoque, Ashwin Rao, Abhishek Kumar, Mostafa Ammar, Pan Hui, and Sasu Tarkoma. Sensing multimedia contexts on mobile devices. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '20, page 40–46, New York, NY, USA, 2020. ACM.

[4] David Choffnes. A case for personal virtual networks. HotNets '16, page 8–14, New York, NY, USA, 2016. ACM.

[5] James Newman, Abbas Razaghpanah, Narseo Vallina-Rodriguez, Fabian E. Bustamante, Mark Allman, Diego Perino, and Alessandro Finamore. Back in control – an extensible middle-box on your phone. *CoRR*, 2020.

[6] Yihang Song and Urs Hengartner. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15, pages 15–26, New York, NY, USA, 2015. ACM.

[7] Daoyuan Wu, Rocky K. C. Chang, Weichao Li, Eric K. T. Cheng, and Debin Gao. MopEye: Opportunistic Monitoring of Per-app Mobile Network Performance. In *Proceedings of USENIX ATC '17*, pages 445–457. USENIX Association, 2017.

[8] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. Haystack: In Situ Mobile Traffic Analysis in User Space. *CoRR*, 2015.

[9] Study on encrypted traffic detection and verification. Technical Specification (TS) 23.787, 3rd Generation Partnership Project (3GPP), 03 2018. Version 16.0.0.

[10] Certificate Pinning. https://www.symantec.com/content/dam/symantec/docs/white-papers/certificate-pinning-en.pdf, 2017.

[11] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.

[12] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Transactions on Network and Service Management*, 16(2):445–458, 2019.

[13] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48, 2017.

[14] Mohammad Ashraful Hoque, Matti Siekkinen, Jukka K. Nurminen, Mika Aalto, and Sasu Tarkoma. Mobile multimedia streaming techniques: Qoe and energy saving perspective. *Pervasive and Mobile Computing*, 16:96 – 114, 2015.

[15] Android AudioManager. https://developer.android.com/reference/android/media/AudioManager, 2020.

[16] Android TelephonyManager. https://developer.android.com/reference/android/telephony/TelephonyManager, 2020.

[17] Android CameraManager. https://developer.android.com/reference/android/hardware/camera2/CameraManager, 2020.

[18] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. Studying tls usage in android apps. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 350–362, New York, NY, USA, 2017. ACM.

[19] Anh Le, Janus Varmarken, Simon Langhoff, Anastasia Shuba, Minas Gjoka, and Athina Markopoulou. AntMonitor: A System for Monitoring from Mobile Devices. In *Proceedings of C2B(1)D '15*, pages 15–20. ACM, 2015.

[20] Anssi Rämö and Henri Toukomaa. Voice Quality Characterization of IETF Opus Codec. In *INTERSPEECH*, 2011.

[21] Voice Over IP - Per Call Bandwidth Consumption. https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/7934-bwidth-consume.html, 2016.

[22] Anika Schwind, Florian Wamser, Thomas Gensler, Phuoc Tran-Gia, Michael Seufert, and Pedro Casas. Streaming characteristics of spotify sessions. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, May 2018.

[23] Mohammad A. Hoque, Ashwin Rao, and Sasu Tarkoma. Network and application performance measurement challenges on android devices. *SIGMETRICS Perform. Eval. Rev.*, 48(3):6–11, mar 2021.

[24] Xiaoqing Zhu and Rong Pan. NADA: A Unified Congestion Control Scheme for Low-Latency Interactive Video. In *2013 20th International Packet Video Workshop*, pages 1–8, 2013.

[25] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[26] Monsoon Power Monitor. https://www.msoon.com, 2020.

[27] Paula Branco, Luís Torgo, and Rita P. Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Comput. Surv.*, 49(2):31:1–31:50, 8 2016.

[28] Martín Abadi and Others. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[29] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: A practical approach to energy-aware cellular data scheduling. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, pages 85–96, New York, NY, USA, 2010. ACM.

[30] Fengyuan Xu, Yunxin Liu, Thomas Moscibroda, Ranveer Chandra, Long Jin, Yongguang Zhang, and Qun Li. Optimizing Background Email Sync on Smartphones. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 55–68, New York, NY, USA, 2013. ACM.

[31] 3GPP. System Architecture for 5G. Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 12 2017. Version 15.0.0.

[32] Ramon Ferrus, Oriol Sallent, Jordi Perez-Romero, and Ramon Agusti. On 5G Radio Access Network Slicing: Radio Interface Protocol Features and Configuration. *IEEE Communications Magazine*, 56(5):184–192, May 2018.

[33] Alok Tongaonkar, Ram Keralapura, and Antonio Nucci. Challenges in network application identification. In *Presented as part of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats*, San Jose, CA, 2012. USENIX.

[34] Tarun Mangla, Emir Halepovic, Mostafa Ammar, and Eellen Zegura. Mimic: Using passive network measurements to estimate http-based adaptive video qoe metrics. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–6, June 2017.

[35] Vengatanathan Krishnamoorthi, Niklas Carlsson, Emir Halepovic, and Eric Petajan. Buffest: Predicting buffer conditions and real-time requirements of http(s) adaptive streaming clients. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, pages 76–87, New York, NY, USA, 2017. ACM.

[36] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing Skype Traffic: When Randomness Plays with You. *SIGCOMM Comput. Commun. Rev.*, 37(4):37–48, August 2007.

[37] Andrew Reed and Michael Kranch. Identifying HTTPS-Protected Netflix Videos in Real-Time. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 361–368, New York, NY, USA, 2017. ACM.

[38] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. *SIGCOMM Comput. Commun. Rev.*, 36(4):339–350, August 2006.

[39] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: A statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, page 135–148, New York, NY, USA, 2004. ACM.

[40] Kyoungwon Suh, Daniel R. Figueiredo, Jim Kurose, and Don Towsley. Characterizing and detecting relayed traffic: A case study using Skype. *IEEE Infocom*, 2006.

[41] Philip Branch and Lam Hoang Do. Real time voip traffic classification. Technical report, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 2009.

[42] Mohammad A. Hoque, Petteri Nurmi, Matti Siekkinen, Pan Hui, and Sasu Tarkoma. The bits of silence: Redundant traffic in voip. MMSys '20, New York, NY, USA, 2020. ACM.

[43] Feng Li, Jae Won Chung, and Mark Claypool. Silhouette: Identifying youtube video flows from encrypted traffic. In *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '18, pages 19–24, New York, NY, USA, 2018. ACM.

[44] Weirong Jiang and Maya Gokhale. Real-time classification of multimedia traffic using fpga. In *2010 International Conference on Field Programmable Logic and Applications*, pages 56–63, Aug 2010.

[45] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 11:1–11:12, New York, NY, USA, 2008. ACM.