Sensing Multimedia Contexts on Mobile Devices

Mohammad A. Hoque
University of Helsinki
mohammad.a.hoque@helsinki.fi

Ashwin Rao
University of Helsinki
ashwin.rao@helsinki.fi

Abhishek Kumar
University of Helsinki
abhishek.kumar@helsinki.fi

Mostafa Ammar
Georgia Institute of Technology
ammar@cc.gatech.edu

Pan Hui
University of Helsinki, HKUST
pan.hui@helsinki.fi

Sasu Tarkoma
University of Helsinki
sasu.tarkoma@helsinki.fi

ABSTRACT

We use various multimedia applications on smart devices to consume multimedia content, to communicate with our peers, and to broadcast our events live. This paper investigates the utilization of different media input/output devices, e.g., camera, microphone, and speaker, by different types of multimedia applications, and introduces the notion of *multimedia context*. Our measurements lead to a sensing algorithm called MediaSense, which senses the states of multiple I/O devices and identifies *eleven* multimedia contexts of a mobile device in real time. The algorithm distinguishes stored content playback from streaming, live broadcasting from local recording, and conversational multimedia sessions from GSM/VoLTE calls on mobile devices.

CCS CONCEPTS

• **Information systems** → **Multimedia information systems**; • **Networks** → **Network algorithms**; **Network performance evaluation**; • **Computer systems organization** → **Embedded systems**; • **Hardware** → *Sensor applications and deployments*;

KEYWORDS

Multimedia context, Live Streaming, Voice over IP, Traffic Classification, Network Management, Sensor Fusion, Virtual Reality.

1 INTRODUCTION

Mobile devices have been generating 60% of all Internet traffic, and the Cisco Visual Network index predicted that traffic from various services such as video broadcast, live streaming, AR/VR applications would grow by a five-to-seven fold by 2022 [17]. With the recent pandemic [11], perhaps we have already reached such growth. The obvious reason is that millions of peoples are engaged in remote work, interactive online education, and entertainment, which are mostly video over IP, voice over IP (VoIP), live broadcast, and streaming traffic. YouTube/Netflix are already downgrading QoS by throttling the streaming bitrates to accommodate essential services [13].

The network service providers have to verify the QoS requirements before applying any policies. In most cases, they employ their traffic identification methods. Traffic identification also helps to manage their networks [30]. Along with the content from popular content providers, user-generated content is also on the rise due to online lectures from the academic institutions and additional remote collaborations worldwide [12]. At the same time, with numerous privacy leakage incidents [5], mobile operating systems have been enforcing strict on-device data access policies [31] and the use of secure protocols to keep the application traffic secure in transit [15]. Therefore, verifying QoS requirements and providing better QoS while abiding users' privacy continues to be non-trivial for the network operators and Internet service providers. At the same time, they are struggling to manage their networks.

In these circumstances, mobile devices can assist the networks. Since the users interact with various types of multimedia content using different multimedia applications on their smart devices, it is essential to understand the ongoing multimedia activity on a device. We call this *multimedia context. A multimedia context defines whether a user is producing or consuming content or engaged in a conversation on a mobile device.* In this article, we present a unique sensing algorithm, MediaSense, to accurately detect the multimedia contexts of a device. MediaSense can be used to identify various multimedia traffic real time on mobile devices. Thus multimedia traffic classification by the service providers or mobile operators can be offloaded to mobile devices. Besides, different energy-aware optimization techniques for mobile devices, e.g., traffic aggregation [25], computation offloading [18], and background traffic scheduling [32], can take advantage of MediaSense.

MediaSense relies on the answers to the following questions. *(i) What are the content types various multimedia applications offer to the users? (ii) How do the users interact with each of the contents? (iii) Which I/O devices are utilized during such interactions on smart devices. (iv) What are the states of these I/O devices when users interact with the multimedia applications?*

We first classify the multimedia application according to their purposes and define three multimedia contexts; (i) multimedia production, (ii) multimedia consumption, and (iii) conversational multimedia. Next, we explore their utilization of several media I/O hardware components and present a multimedia context sensing algorithm, called MediaSense. The algorithm initially identifies the above media contexts by inferring the state of microphone and speaker. With the help of camera and display, it further separates the video contexts from audio. Finally, network flow information, such as bitrate and bytes exchanged, assists in classifying them as local or IP-based contexts, as presented in Figure 1. MediaSense distinguishes stored content playback from streaming, live broadcasting from local recording, and conversational multimedia from GSM or voice over LTE (VoLTE) calls.

2 MULTIMEDIA APPS AND CONTEXTS

Multimedia streaming applications received significant attention for their diverse streaming techniques [21], optimizing content

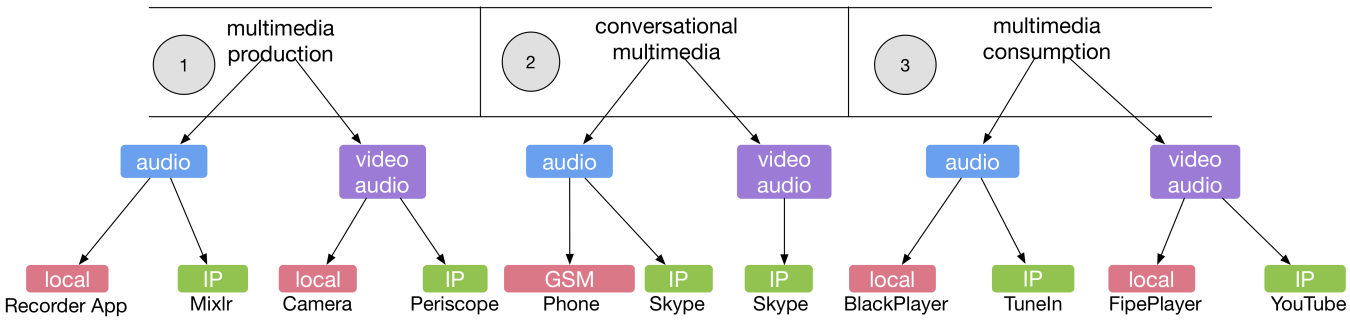

Figure 1: Multimedia Applications and their types according to their purposes. 'local' implies media production and consumption on mobile devices without any IP communication. The example applications are at the bottom.

delivery [28], QoE prediction and optimization [14, 22], and traffic classification [24]. modern multimedia services communicate over HTTPS [26]. Several approaches decrypt VoIP traffic from Skype [16], WhatsApp [23], and IMO [29] to investigate their codec.

In contrast, we define multimedia context, and MediaSense addresses the multimedia context identification problem by sensing the status of multiple I/O devices. A classification of multimedia contexts and the corresponding application types, and example applications are illustrated in Figure 1. In this section, we study those multimedia applications on two Android devices; Nexus 6 (Android 7.1.1) and LG G5 (Android 8.0). We explore how different I/O devices are utilized by such multimedia applications and their corresponding multimedia contexts. We capture traffic on Nexus 6 with tcpdump. The smartphones were fully charged during the measurements to avoid the consequence of any system optimization due to the lower battery level [20].

2.1 Mobile Multimedia Contexts

The various combinations of different I/O devices can reveal the multimedia contexts of a device while using multiple different applications. A multimedia context hints whether a device is (i) producing media content, (ii) consuming media content, or (iii) engaged in a conversation, (iv) the content type, i.e., audio or video, and (v) whether the media context requires IP to exchange the content.

Depending on the characteristics of the application, all the required I/Os are initialized at the same time. Since a user needs to launch an app with a screen touch, it is intuitive that the display is busy. Therefore, the initial states of the I/Os for all the multimedia applications are the same. We represent the status of the I/O devices with '1' and '0', where the bits represent the busy and free status of the corresponding I/O devices. However, the network I/O status does not represent the network interface status. It rather depends on network activities, such as bitrates of the corresponding applications for the sending, receiving, and exchanging media.

2.2 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.

Both the microphone and camera are two necessary input devices and initiated together when recording a video. Figure 2 (Left) shows

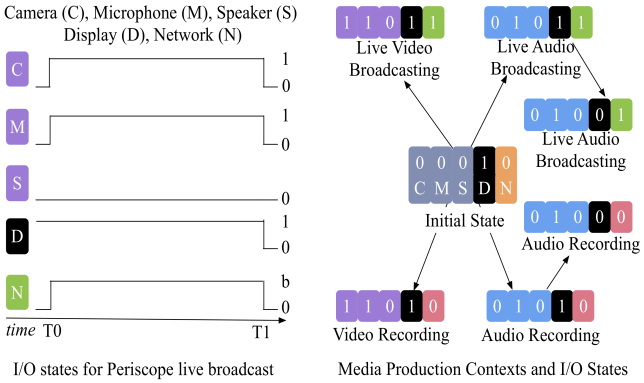

Figure 2: States of the I/O devices while broadcasting live with Periscope (Left). The media production contexts and the corresponding I/O states (Right).

the states of the I/O devices during a live video broadcasting session of Periscope. The live broadcast beings at $T0$, and the application initializes camera and microphones. The output device, display, is also used, and the data transmission begins. The broadcasting terminates at $T1$. Periscope initiates 2-8 TCP connections as shown in Table 1. We observed the uplink bitrate of 459 kbps and 128 kbps bitrates for broadcasting live video and audio respectively.

Figure 2 (Right) illustrates the meaningful media production contexts emerge from the initial state. Since video recording requires users' attention, the application must be at 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 recording and audio live broadcast can continue in the background once the required I/O devices are initialized. The states of I/O devices for the recording applications differ from those of the live broadcasting applications only by the network.

2.3 Conversational Multimedia Contexts

Conferencing applications have two media contexts, i.e., audio or video conversations. Figure 3 (Left) demonstrates that a WhatsApp VoIP call begins at $T0$, and all the I/O devices become busy, except the camera. The display turns off at $T1$ and turns on again at $T2$.

Application	Media Context	Protocol	uplink	downlink
Periscope [7]	Media Production (LiveVideoCast)	TCP	459 kbps	421 kbps
Mixlr [6]	Media Production (LiveAudioCast)	TCP	21-128 kbps	
TuneIn [8]	Media Consumption (Streaming)	TCP		32-320 kbps
YouTube [10]	Media Consumption (Streaming)	UDP (QUIC)		120-3455 kbps
WhatsApp [9]	Conversational Media (AudioConf)	TCP, UDP	22 kbps	17 kbps
WhatsApp	Conversational Media (VideoConf)	TCP, UDP	378 kbps	411 kbps

Table 1: Basic traffic properties of the network dependent, i.e., IP-based media contexts and the applications.

Finally, the call terminates at $T3$. WhatsApp initiates both TCP and UDP flows, as soon as the call begins, as shown in Table 1. 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. GSM/VoLTE calls also require a microphone and speaker together.

Figure 3 (Right) shows that the states of the I/O devices change according to the conversation type. An audio conference requires the microphone and speaker; however, it does not require the display to be active once the call is initiated. In contrast, a video call initializes all the media I/Os. .

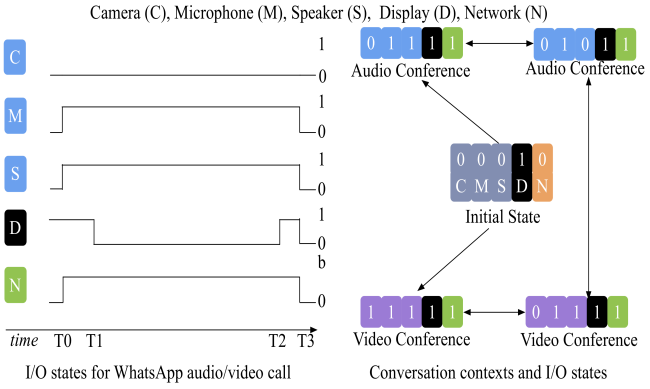

Figure 3: States of the I/O devices during a WhatsApp call (Left). The conversation contexts and the states of the corresponding I/O devices (Right).

The video conferencing calls have similar bitrates to a live video broadcasting application. Table 1 shows that Periscope's bitrate is very close to the bitrates of a WhatsApp video call. However, they differ on the transport protocol. A video conversation also can proceed even without a camera is active either on the caller's or on the collee device. For example, when the caller initiates a video call, all the media sensors are activated on callee's device as well. Then 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. The media state changes to an audio call when both users turn off their cameras (Figure 3).

2.4 Multimedia Consumption Contexts

When an end-user plays multimedia content from local storage or streams from a remote service provider, the device is in a media consumption context.

Figure 4 (Left) 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 $T2$. The user terminates the streaming session at $T3$. TuneIn initiates multiple TCP flows as soon as the playback begins. The bitrates of the Shoutcast/TuneIn audio streams vary 24-320 kbps, as shown in Table 1. Similar to the live broadcast, Periscope live streaming (from Periscope users) uses multiple TCP connections and similar bitrates. Some streaming applications can have ON/OFF pattern in network traffic [21].

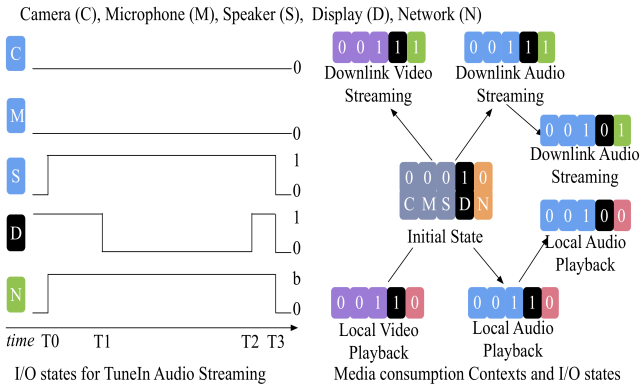

Figure 4: States of the I/O devices while streaming audio music with TuneIn (Left). The media consumption contexts and the states of the I/O devices (Right).

There are four media consumption contexts, as shown in Figure 4. For watching videos from local storage or video streaming, the use of display and speaker are mandatory. If the user switches to another application, the playback stops. On the contrary, the audio applications can be in the background once the playback starts, and the corresponding state change is depicted in the figure. The streaming applications download content from the remote server for playback and thus depend on IP activities. Both audio and video playbacks exhibit the same I/O states. Given the diverse set of multimedia streaming applications, on-demand streaming, and live/pseudo live streaming. Some application starts playing content with negligible initial playback delay, whereas some apps continue to cache and depend on input from the user for the playback. In the latter scenario, the initial playback delay can be significant.

2.5 Mixed Multimedia Contexts

In principle, the states of some I/O devices are important for an application, as it is necessary to check whether another application is already using a particular I/O device or not.

Media context	Media Production	Conversation	Media Consumption
Media Production	No	No	No
Conversation	No	No	Yes
Media Consumption	No	Yes	Yes

Table 2: Mixed media contexts on Nexus 6.

Algorithm 1: MediaSense

```
▷ Comment 1: Pre-computed features;
mediaFeatures = Map(mediaContext, bitrate);
while true do
    trafficstat = getTXRXbytes();
    mic = sampleMicrophone() ∈ {1,0};
    speaker = sampleSpeaker() ∈ {1,0};
    camera = sampleCamera() ∈ {1,0};
    display = sampleDisplay()∈ {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 ≈ 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: Conference 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
```

Table 2 demonstrates our findings when using applications of two media contexts together on a single screen device, Nexus 6. The table shows that two media production applications cannot be played together on a single screen device. Audio and video content cannot be recorded at the same time by two separate applications. We also tried to record audio via the default voice recording application, and the recorder stops as soon as Periscope starts broadcasting.

Figures 2 & 4 show that media production and consumption applications do not share camera, microphone or speaker. However, Nexus 6 does not allow to record or broadcast media when another streaming application is running. For example, it is not possible to run an audio recording application in the background and streaming via YouTube or TuneIn in the foreground. In this case, the audio recording terminates as soon as the player initializes the required sensors. Similarly, recording applications cannot be played during a VoIP call. Nevertheless, we were able to stream audio in the presence of a VoIP call in a single screen.

Android API	I/O device	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

Table 3: Android APIs for detecting media contexts, utilization of Media I/Os, and their permission requirements.

We could not play two multimedia applications requiring an input media I/O device, i.e., a camera or microphone, at the same time. The applications can share only the speaker concurrently. For example, we played a streaming application during an audio conference on an Android device. Two media consumption applications also can continue playing together, such as TuneIn and YouTube.

2.6 Summary

The media context tree in Figure 1 also summarizes the dependency of the applications and the corresponding contexts on I/O devices. For example, at the first level, the use of only a microphone separates the production context from the others. The conversational contexts use both the microphone and speaker together. The use of only the speaker separates the consumption context. At the second level, the camera separates the video from audio contexts for both production and conversational media, whereas the display separates the video from audio for media consumption. At the third level, all the contexts are separated according to the required network activities, i.e., transmit, receive, or exchange traffic.

3 MEDIASENSE

MediaSense (Algorithm 1) scans the states of five I/O devices and fuses them to infer a media context. We implement MediaSense algorithm as a user-level service for Android devices. It runs as a background service and looks for multimedia contexts periodically at 2Hz. Whenever one or more of media I/O devices change states, MediaSense initiates a new multimedia context. It first checks whether the media context is audio or video-related with the help of the camera and display (Comment 2 in Algorithm 1). Then, it finds a particular feature from the traffic stat in uplink/downlink or both directions to separate IP-based contexts from the local media context (Comment 3 in Algorithm 1).

3.1 Separating Audio/Video Contexts

The algorithm periodically infers the states of the I/O devices and fuse them together to determine the media context.

(1) Conversational Context. Fortunately, Android provides APIs for the applications to indicate their modes of operation to the AudioManager [2], which allow other applications to know the status of AudioManager via getMode() API. AudioManager operates in one of the three modes; IN_CALL, IN_COMMUNICATION, and RINGTONE. These modes indirectly indicate that an ongoing context is conversational, and both the microphone and speaker are busy. TelephonyManager has getCallState() to characterize a

Media context	uplink	downlink	Features
LiveAudioBroadCast or AudioRecord	✓	X	bitrate (≥8 kbps), microphone
LiveVideoBroadCast or VideoRecord	✓	X	bitrate (≥8 kbps), microphone, camera
AudioConference or GSM/VoLTE	✓	✓	bitrate (≥8 kbps), microphone, speaker
VideoConference	✓	✓	bitrate (≥8 kbps), microphone, speaker, camera
VideoStreaming or LocalVideoPlayback	X	✓	faststartbytes (≥100 KB), speaker, display

Table 4: Features considered for the identifying the media contexts. "faststartbyte" denotes the amount of data downloaded during the first 15 seconds of streaming.

GSM/VoLTE call [3] and thus expresses the states of microphone and speaker.

Table 3 summarizes the mapping between Android APIs and the corresponding media I/O devices. MediaSense characterizes a context as a video conference, 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 [4] to poll camera status exactly when the audio mode changes.

(2) Media 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. It does not differentiate whether the playback is audio or video. We also could not find APIs hinting about streaming. In Figure 4, we notice that it is not straightforward to distinguish between audio and video consumption contexts, given the status of the I/O devices. The reason is that audio applications require only speakers and can be played by keeping the display either active or inactive. Therefore, the algorithm first decides a media consumption context as the video. When the display is off, the media context is changed to audio type, as the media session continues.

(3) Media Production Contexts. MediaSense uses the APIs which do not require 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 user permission. MediaSense implements MediaRecorder APIs with user permission to detect the audio production contexts due to the Voice Recorder or Mixlr like applications.

3.2 Separating Local/IP-based Contexts

In order to distinguish IP-based and local media contexts, MediaSense tracks very high-level traffic stats of the devices, e.g., total incoming and outgoing bytes, periodically. In this case MediaSense relies on TrafficStats() APIs. From these stats collected over a period of time, MediaSense determines whether there is a change in traffic patterns due to a particular media context or not. It considers the bitrates and other derived features, as presented in Table 4. Note that 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 2, and the reasonings are the following.

(1) Conversational Multimedia Contexts. Unlike the other applications, conversational traffic carry voice or video data in both directions. In Table 1, we notice that the voice traffic has a minimum bit rate of 14 kbps. However, the applications may have bitrates even lower to 8 kbps in one direction [1]. A GSM/VoLTE call be identified using the getMode() API and given that there is no traffic associated with the context.

(2) Multimedia Consumption Contexts. On-demand streaming applications, e.g., YouTube, Spotify, begin with a fast start. In this phase, these applications download 10-40 seconds equivalent playback content. Spotify streams audio via persistent HTTP connections over TCP, regardless of the device type [27]. 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 [27]. YouTube downloads more than one Megabytes during the fast start phase. Periscope downloads content at a constant bit rate after the fast start. The encoding rates of the audio/video streams are presented in Table 1. Therefore, MediaSense relies on isMusic() API and the traffic feature presented in Table 4 to separate streaming contexts from the local music playback.

(3) Multimedia Production Contexts. In Table 1, we notice that the Periscope's outgoing rate is 459 kbps. A 64 kbps outgoing bitrate is very common for live audio broadcasting. Mixlr supports 32-128 kbps. We consider the minimum bound of 8 kbps as the context feature. MediaSense uses bitrate feature along along with the CameraManager & MediaRecorder APIs to separate the IP-based based media contexts from the local recording contexts.

3.3 Performance Evaluation

We installed MediaSense on LG G5 (Android 8). We evaluated the performance of the algorithm in separating the media contexts in real time of various multimedia applications presented in Figure 5. For all 12 applications in the figure, there were 280 media sessions; 80 media consumption sessions, 120 conversation contexts, and 80 media consumption contexts. The duration of each session was 60-90 seconds. The media sessions for the same application and media type were executed after every 1 minute.

Figure 5 shows that MediaSense separates the top level three media contexts accurately and so the second level media contexts.

At third-level, MediaSense also identifies the media productions and conversation contexts very accurately. It classified the VoLTE calls as GSM, as VoLTE traffic do not follow the standard TCP/IP stack. Note that the MediaSense first detects the media consumption contexts as video, since the display is active initially in either

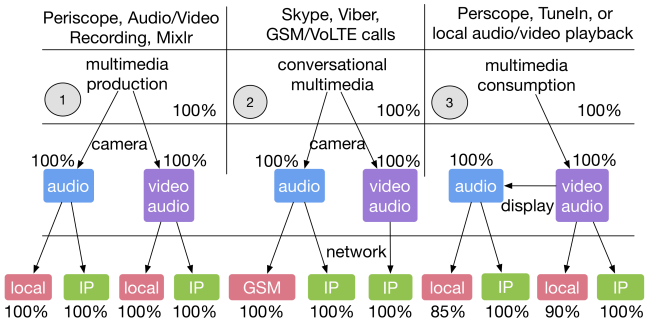

Figure 5: Performance of MediaSense in separating different multimedia contexts. MediaSense separates three high-level media contexts at the first level. It separates five second level media contexts and 11 media contexts at the bottom.

scenario. Therefore, the audio consumption contexts are initially detected as video contexts. After 15 seconds of playback, we turned off the display, and MediaSense accurately detected the audio contexts.

MediaSense suffered from error in the presence of local media consumption applications. Among 20 sessions, the algorithm classified three sessions as audio streaming. Similarly, two local video playback sessions were also classified as video streaming, as the smartphone updated some applications in the background during playback, which satisfied the streaming condition presented in Table 4. MediaSense also measured the duration of 280 media sessions accurately (Comment 6 in Algorithm 1).

4 DISCUSSIONS

MediaSense effectively identifies the media contexts on mobile devices. These contexts can assist in various optimizations and network management. The accurate usage time of these contexts can be helpful for users to understand their engagement with different types of multimedia applications. In this section, we discuss the implication of using media I/O devices on mobile devices towards privacy and different applications. We also discuss our future work.

Media Context and Privacy. On mobile devices, it is common to have multimedia notifications, which are a particular kind of local music playback. The prime examples are alarms or notifications for other calendar applications. The duration of alarm playback depends on the configured alarm tones and user response. Our many trials with the Android AudioManager APIs were failed to track such notifications.

Nevertheless, on the latest Android devices (>8.0), it is possible to run media production and consumption applications at the same time. Although such actions do not produce any sensible user experience, this raises privacy concerns as we were able to record streaming content on Mi8. Therefore, media recording should not be allowed in the presence of media consumption or conversational contexts due to privacy reasons and possible violation of rules for DRM protected contents.

Media Context and Extended Reality Applications. Recent advances in sensor technologies have enabled a new breed of applications that operate in three-dimensional space: augmented reality (AR), virtual reality (VR), mixed reality (MR), and 360-degree video. MediaSense can aid in optimized resource allocation for these applications. For example, in a VR-based collaborative gaming application, players communicate over VoIP, thereby making the use of speaker and microphone consumes network bandwidth. Besides, the player's hand and foot gestures change frequently. Thus the camera and depth sensors consume more battery power [33].

In the future, meetings are likely to be done with holographic techniques, where the participants are present through their holographs, which gives the impression that everyone is present in the room. In such applications, people take their turn to speak, which implies frequent changes in contexts. Only the active speaker needs the most of resources, while others can free their resources. Consequently, MediaSense can also assist in resource allocation for holographic applications that are associated with MR [19].

Media Context and Network Management. A mobile device can perform multimedia context-aware communication. With flow level access, it would be possible to find the exact flows for particular IP-based media contexts. This implies that MediaSense enables flow classification and marking the flows according to the media context. This would also intuitively mean that the end devices are classifying multimedia traffic and alleviating the limitations of the service providers in classifying encrypted traffic. The service providers can use the marked flows to manage the networks.

We have presented how different multimedia contexts can be detected with reasonable accuracy. However, the algorithm suffers from error (false negative) in the presence of competitive downlink traffic. Although downlink traffic is dominant on user devices, similar scenarios are also susceptible to happen with the local media production contexts in the presence of other uplink traffic. Nevertheless, it would be possible to remove such false negatives by extending the features to the flow level.

As of our future work, we aim to implement the MediaSense as part of the mobile system and use media context-specific traffic classification and traffic termination from mobile devices. In this case, a user device receives media context-specific traffic profiles from the network and re-connects to the corresponding network for a particular media context. This mechanism also enables multimedia context-specific network slice allocation where each profile logically may represent a slice.

ACKNOWLEDGEMENTS

The research is supported by the Academy of Finland, grant no 1319017.

REFERENCES

[1] Voice Over IP - Per Call Bandwidth Consumption. https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/7934-bwidth-consume.html, 2016. Last Accessed: November 24, 2020.
[2] Android AudioManager. https://developer.android.com/reference/android/media/AudioManager, 2018. Last Accessed: November 24, 2020.
[3] Android AudioManager. https://developer.android.com/reference/android/telephony/TelephonyManager, 2018. Last Accessed: November 24, 2020.
[4] Android CameraManager. https://developer.android.com/reference/android/hardware/camera2/CameraManager, 2018. Last Accessed: November 24, 2020.
[5] Facebook Cambridge Analytica data scandal. https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal, 2018. Last Accessed: November 24, 2020.
[6] Mixlr - Social Live Audio. https://play.google.com/store/apps/details?id=com.mixlr.android, 2018.

[7] Perscope LIVE. https://www.periscope.tv, 2018.
[8] TuneIn. https://tunein.com, 2018.
[9] WhatsApp Messenger. https://www.whatsapp.com/android/, 2018.
[10] YouTube. https://www.youtube.com, 2018.
[11] 2019–20 Coronavirus Pandemic. https://en.wikipedia.org/wiki/2019%E2%80%9320_coronavirus_pandemic, 2019. Last Accessed: November 24, 2020.
[12] Nokia reveals impact of COVID-19 on network traffic. https://telecoms.com/503494/nokia-reveals-impact-of-covid-19-on-network-traffic/, 2020. Last Accessed: November 24, 2020.
[13] YouTube throttling streaming quality globally as coronavirus forces people indoors. https://abcnews.go.com/Technology/netflix-youtube-throttle-streaming-quality-europe-coronavirus-forces/story?id=69754458, 2020. Last Accessed: November 24, 2020.
[14] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a Predictive Model of Quality of Experience for Internet Video. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 339–350, New York, NY, USA, 2013. ACM.
[15] Lucia Ballard and Simon Cooper. What's New in Security? Apple wwdc16 session 706, 2016.
[16] 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.
[17] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2016–˜2021. Technical report, 2017.
[18] Bo Han, Pan Hui, V.S. Anil Kumar, Madhav V. Marathe, Guanhong Pei, and Aravind Srinivasan. Cellular Traffic Offloading through Opportunistic Communications: A Case Study. In *Proceedings of the 5th ACM Workshop on Challenged Networks*, CHANTS '10, page 31–38, New York, NY, USA, 2010. Association for Computing Machinery.
[19] Dongbiao He, Cédric Westphal, and J. J. Garcia-Luna-Aceves. Network support for AR/VR and immersive video application: A survey. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications, ICETE 2018 - Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS, Porto, Portugal, July 26-28, 2018*, pages 525–535. SciTePress, 2018.
[20] Mohammad A. Hoque, Ashwin Rao, and Sasu Tarkoma. In Situ Network and Application Performance Measurement on Android Devices and the Imperfections, 2020. arXiv:2003.05208.
[21] Mohammad A. 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.
[22] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 393–406, Boston, MA, 2017. USENIX Association.
[23] F. Karpisek, I. Baggili, and F. Breitinger. WhatsApp network forensics: Decrypting and understanding the WhatsApp call signaling messages. *Digital Investigation*, 15:110 – 118, 2015. Special Issue: Big Data and Intelligent Data Analysis.
[24] 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.
[25] Swadhin Pradhan, Sourav Kumar Dandapat, Niloy Ganguly, Bivas Mitra, and Pradipta De. Aggregating inter-app traffic to optimize cellular radio energy consumption on smartphones. In *7th International IEEE Conference on Communication Systems and Networks (COMSNETS), 2015*, pages 1–8. IEEE, 2015.
[26] 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.
[27] A. Schwind, F. Wamser, T. Gensler, P. Tran-Gia, M. Seufert, and P. Casas. Streaming Characteristics of Spotify Sessions. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, May 2018.
[28] M. Siekkinen, M. A. Hoque, and J. K. Nurminen. Using Viewing Statistics to Control Energy and Traffic Overhead in Mobile Video Streaming. *IEEE/ACM Transactions on Networking*, 24(3):1489–1503, 2016.
[29] M.A.K. Sudozai, Shahzad Saleem, William J. Buchanan, Nisar Habib, and Haleemah Zia. Forensics study of IMO call and chat app. *Digital Investigation*, 25:5 – 23, 2018.
[30] 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.
[31] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. Android Permissions Remystified: A Field Study on Contextual Integrity. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 499–514, Washington, D.C., 2015. USENIX Association.
[32] 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*, pages 55–68. ACM, 2013.
[33] Wenxiao Zhang, Bo Han, and Pan Hui. On the networking challenges of mobile augmented reality. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 24–29, 2017.