2018 Third ACM/IEEE Symposium on Edge Computing

QoE Inference and Improvement Without End-Host Control

Ashkan Nikravesh
University of Michigan
ashnik@umich.edu

Qi Alfred Chen
University of California, Irvine
alfchen@uci.edu

Scott Haseley
University of Illinois at Urbana-Champaign
haseley2@illinois.edu

Xiao Zhu
University of Michigan
shawnzhu@umich.edu

Geoffrey Challen
University of Illinois at Urbana-Champaign
challen@illinois.edu

Z. Morley Mao
University of Michigan
zmao@umich.edu

***Abstract*—Network quality-of-service (QoS) does not always translate to user quality-of-experience (QoE). Consequently, knowledge of user QoE is desirable in several scenarios that have traditionally operated on QoS information. Examples include traffic management by ISPs and resource allocation by the operating system. But today these systems lack ways to measure user QoE.**

To help address this problem, we propose *offline* generation of per-app models mapping app-*independent* QoS metrics to app-*specific* QoE metrics. This enables any entity that can observe an app's network traffic—including ISPs and access points—to *infer* the app's QoE. We describe how to generate such models for many diverse apps with significantly different QoE metrics. We generate models for common user interactions of 60 popular apps. We then demonstrate the utility of these models by implementing a QoE-aware traffic management framework and evaluate it on a WiFi access point.

Our approach successfully improves QoE metrics that reflect user-perceived performance. First, we demonstrate that prioritizing traffic for latency-sensitive apps can improve *responsiveness* and *video frame rate*, by 46% and 115%, respectively. Second, we show that a novel QoE-aware bandwidth allocation scheme for bandwidth-intensive apps can improve average video bitrate for multiple users by up to 23%.

***Keywords*-Measurement; Application Performance; Quality of Service (QoS); Quality of Experience (QoE)**

I. Introduction

For network-based applications (apps) like video, Voice over IP (VoIP), and web browsing, knowledge of end users' QoE is valuable in various ways. When dealing with congestion, any ISP can shape traffic in a manner sensitive to the perceived QoE of its users, *e.g.,* throttling every flow only to the extent that does not significantly degrade QoE for the corresponding users. An app's servers can leverage the knowledge of users' QoE to appropriately adapt its traffic delivery to its users. For example, a video service can reduce the video bitrate to eliminate rebuffering delays incurred at a higher bitrate. Furthermore, if the operating system (OS) on a user's end device can detect when the user is suffering from poor QoE, it can attempt to diagnose the problem.

However, today, all of these useful QoE-aware mechanisms for traffic management, app delivery adaptation, and user experience problem diagnosis are stymied by a basic limitation: determining a user's QoE for a particular app requires software on the user's device that is capable of measuring QoE metrics for that app, and reports information to the entity (OS, ISP, or application server) implementing the QoE-aware mechanism. This limitation stems from several reasons.

- **App-specific QoE metrics.** The metrics that capture user QoE vary significantly across apps, *e.g.,* rebuffering delays for video, mean opinion score for VoIP, and page load times for the Web. This makes it challenging, if not impossible, to write one software, which if installed on a user's device, can measure the user's QoE for any arbitrary app.
- **A lack of API to communicate QoE.** In cases where the user interacts with an app via client software offered by that app's provider, that client is able to measure the user's QoE and relay such information to the app's servers. However, there typically does not exist an interface via which an app's client software can relay measured QoE information to other entities that can make use of this information, such as the user's OS or ISP.
- **Third-party clients.** It can also be challenging for an app's own servers to discover user-perceived QoE because users often access apps via client software not developed by the app provider, *e.g.,* YouTube accessed on Internet Explorer, or a messaging service accessed via a third-party client that has support for several messaging services.

As a result of these limitations, we are currently at an impasse. There is growing recognition that dealing with network traffic based on traditional QoS metrics, (*e.g.,* allocating an equal share of the bottleneck link's bandwidth to all flows, irrespective of which apps those flows correspond to) does not accurately account for users' QoE. Yet, all of the wonderful QoE-aware optimizations detailed above are infeasible to implement today due to the lack of software on end devices which can measure and report QoE to the entity implementing the optimization.

To chart a way forward out of the current impasse, we

978-1-5386-9445-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SEC.2018.00011

argue that it is indeed feasible for an entity that has access to a user's network traffic to *infer* the user's QoE, despite not having direct access to app-level QoE measurements from the user's device. Our proposed approach for inferring QoE corresponding to a traffic flow is to rely on models that can map the flow's QoS metrics (such as latency, bandwidth, and loss rate) to the corresponding app's QoE metrics. While a generic QoS-to-QoE model is impractical, our key observation is that such models are indeed feasible on a *per-app* basis.

Here, we use *objective metrics* for quantifying QoE, as subjective tests are time consuming and human subjects must be involved in the assessment process; this does not scale for a large number of apps. While the objective QoE metrics may not reflect the actual user experience for many applications, they have a direct relationship with subjective QoE metrics like user satisfaction and engagement [37], [53], [55], [39], [22]. In fact, according to the recommendations by International Telecommunication Union (ITU), objective QoE metrics for video streaming can be used to estimate and model the subjective (perceived) QoE [7]. In addition, applications themselves rely on the mappings between network QoS metrics and objective QoE metrics to adapt to the changing network condition and improve the end-user perceived performance.

In this paper, we first describe how app-specific models that map QoS metrics to corresponding QoE metrics can be generated. Apps often provide multiple features, and each app *usage* may have a different corresponding QoE metric. To find the most common usages from which to build our models, we perform an app usage measurement study, collecting app usage data from 99 real users interacting with 531 apps over 10 days. We then develop *UsageReplayer*, an app-independent tool to replay these user interactions across different apps in testbeds. By varying QoS metrics as we replay user traces, we are able to understand the effect QoS has on the QoE of individual app usages and build our QoS-to-QoE models.

We present results for three types of apps: video conferencing (AppRTC and Skype), on-demand video streaming (playing three state-of-art adaptive streaming schemes), and 55 interactive apps. For video streaming and video conferencing apps, we find significant non-linearities between QoS and QoE, validating the need for our models. For interactive apps, we find that the QoE of these apps are highly sensitive to the changes in end-to-end delay.

Once equipped with per-app QoS-to-QoE models, we present the design and implementation of QOEBOX, a QoE-centric traffic management framework. QOEBOX is a proxy solution and is transparent to both user-facing apps and backend servers. It relies on the QoS-to-QoE models to infer apps' QoE. Deploying QOEBOX at the edge of the network can be an ideal strategy for cellular networks and ISPs, as being able to measure the QoS metrics in the vicinity of end-users and incorporate the generated models into the scheduling and resource allocation components within the edge of the these networks can further improve the accuracy of the system. Therefore, we implement and evaluate QOEBOX on a WiFi access point.

Finally, we present how the per-app QoS-to-QoE models, once generated, can be utilized for the purpose of traffic management. We showcase direct applications of the models by implementing two QOEBOX modules: classification and prioritization of apps traffic, and an optimal fair bandwidth allocation scheme. In the former, we show that identifying and prioritizing the traffic of various usages can improve app responsiveness by up to 964%, which translates into 6 seconds of extra delay (§V-A). In the latter application, we demonstrate that we can leverage the models to optimally allocate bandwidth to multiple users and improve average video bitrate by 23%, without hurting QoE of any of the users (§V-B).

Our contributions can be summarized as follows:

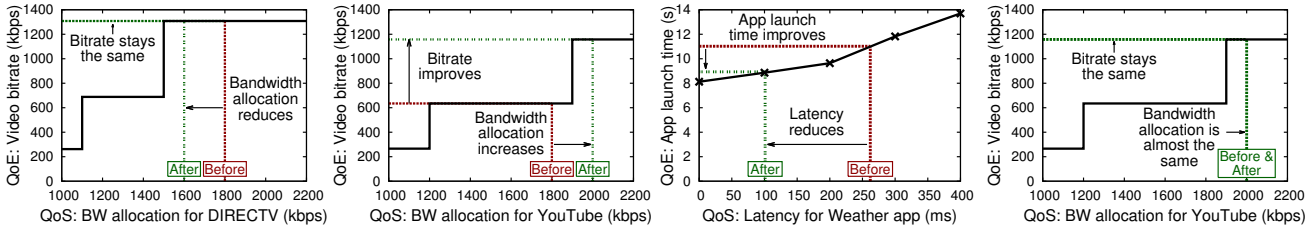
• We propose offline generation of per-app models mapping app-independent QoS metrics to corresponding app-specific QoE metrics. We generate the models by replaying real users' common interactions in popular interactive apps, two popular video conferencing apps, and three video streaming apps, with significantly different QoE metrics. We identify important combination of QoS values that causes change in QoE of these apps using a new adaptive sampling technique.

• We design and implement QOEBOX, a proxy-based QoE-centric traffic management framework. QOEBOX maps the traffic belonging to an app to its corresponding QoE model, and invokes custom modules to apply traffic shaping policies.

• We showcase how the models can be utilized for the purpose of traffic management by designing, implementing, and evaluating two QoE-aware traffic managements schemes on WiFi access points: prioritizing latency sensitive traffic and an optimal fair bandwidth allocation for bandwidth intensive apps. We show that for both schemes, we can significantly improve various QoE metrics, including frame rate, app responsiveness, and video bitrate, without modifying the end-host or requiring a very precise model.

II. Motivating Examples

Equipping OSes, ISPs, cellular carriers, and cloud providers with the knowledge of app QoE has several benefits. First, it enables these entities to efficiently allocate resources in a way that maximizes their users' QoE. Second, it provides a feed-back control loop in which these entities can continuously monitor users' QoE, detect and diagnose performance degradation problems, and react by allocating resources appropriately. Third, the knowledge of users' QoE provides an opportunity for application servers to move their adaptation logic from end-user device to cloudlets or base-



(a) QoE-aware bandwidth allocation for DIRECTV. (b) QoE-aware bandwidth allocation for YouTube. (c) QoE-aware traffic prioritization for Weather app. (d) QoE-aware traffic prioritization for YouTube.

Figure 1: **Problem of resource management for two scenarios, in which two users using different apps at the same time.**

stations, where they have a more accurate estimate of end-user QoS (*e.g.,* allocated bandwidth).

An important case that can benefit from per-app QoS-to-QoE models is resource management in home WiFi routers. In this section, we use two simple motivating examples of QoE-based traffic management schemes to illustrate the usefulness of QoS-to-QoE models in improving app QoE. In both examples, we consider a scenario where two users are connected to the Internet through a WiFi access point (AP).

Example 1. QoE-aware bandwidth allocation. Suppose two users, user A and B, are watching videos using the YouTube and DIRECTV apps, respectively. The amount of bandwidth each app receives ends up depending on the transport and application protocols. In our example, DIRECTV—which uses Apple HLS—and YouTube both use a single TCP connection to download video chunks, so both get 1800 Kbps from of total link bandwidth of 3600Kbps. Since YouTube and Apple HLS use adaptive bitrate streaming and different encoding schemes, the video bitrates for the YouTube and DIRECTV apps are different—620 Kbps for YouTube and 1277 Kbps for DIRECTV. However, as we see in Figure 1(a), the QoS-to-QoE model of Apple HLS indicates that its user can achieve a bitrate of 1277 Kbps with *less* bandwidth. In fact, the extra bandwidth is wasted, in the sense that it is spent on downloading the chunks faster. And as we see from the QoS-to-QoE model for YouTube in Figure 1(b), if we were to allocate this extra 200 Kbps bandwidth to YouTube, it can switch to a higher bitrate.

From this example, we see that a change in bandwidth does not necessarily lead to a change in video quality, and with per-app QoS-to-QoE models, an AP can allocate bandwidth in such a way to improve the overall video quality of the system. Note that QoE is also affected by rebuffering frequency and bitrate switches, which are not illustrated in this example. In §III, we show how QoS-to-QoE models are generated, and in §V-B we formulate a bandwidth allocation problem as an optimization problem, which utilizes these models to maximize overall QoE.

Example 2. QoE-aware traffic prioritization. Suppose now that user A is launching the Weather app while user B is watching a YouTube video. The YouTube app aggressively tries to consume all available bandwidth, which results in an increase in queuing delay in the router's buffer. The QoS-to-QoE model of the Weather app shows that its QoE is sensitive to changes in latency (Figure 1(c)). Due to extra queuing delay caused by YouTube traffic, the Weather app's launch time increases to 11s. To reduce end-to-end latency of the Weather app, one solution pointed out by previous works [14], [29] is to prioritize its traffic. As depicted in Figure 1(d), by prioritizing the Weather app's traffic, we can improve its launch time to 9s, without affecting the QoE of YouTube.

We see from this example that applying traffic prioritization policies can also improve the overall QoE of the system, given per-app QoS-to-QoE models. In §V-A, we design and implement a traffic classification and prioritization module that can control the interaction between different types of traffic with different QoS requirements.

III. Generating QoS-to-QoE models

QoE is measured in different ways by different apps: latency and frame rate for video conferencing, video quality for video streaming, and page load time for web browsing. Consequently, a single mapping from QoS to QoE for all apps does not exist. Because of the differences in the protocols used by different apps, even generating separate QoS-to-QoE models for every *app type* is infeasible. For instance, Skype and Google Hangouts use different techniques to deal with packet loss [53]. Even with the same underlying network packet loss rate, users may experience different QoE when using the two apps. Furthermore, apps often provide multiple activities, services, and features to users, each with possibly its own QoS requirement. For example, searching for a video in YouTube vs. playing it, and posting a picture on Facebook vs. updating a status. As a consequence, we focus on generating QoS-to-QoE models on a per-app basis, and at the *per-usage* level within the app.

We define a *usage* to be a particular interaction with an app, generally with a particular UI component. Scrolling the news feed in Facebook and clicking the play button in YouTube are examples of usages. The set of usages maps to the set of features than an app exposes to users, though some features are not exposed this way (*e.g.,* notifications), and some usages might map to the same feature, depending how the app is designed. We use the expression *usage type*

to categorize the usage, with *scroll*, *click*, and *app launch* being examples of usage types.

To generate per-app models, we need to identify the various usages of an app, find the relevant QoE metric for the usage, and understand how various QoS metrics affect that QoE metric. We now describe in detail how we accomplish this.

A. Recording and Identifying App Usages

There are multiple approaches to identifying app usages. We could rely on app developer support, use an automated state exploration technique, or collect app traces from automated or actual interactive sessions. We chose to collect traces from actual interactive sessions for two reasons. First, collecting traces provides us with the opportunity to replay those traces later, something we rely on when building our QoS-to-QoE models. Second, collecting traces from actual users allows us to understand which apps are commonly used, and which usages are most common. While a monkey testing approach could generate traces for a large number of apps as well, we would not be able to discern the important usages. Understanding which usages are most common allows us to focus on building the models that will provide the largest benefit.

1) Recording App Usage Traces

We record app usage traces using a modified version of the Android framework. We define each touch event as a single *interaction*. To capture interactions, we instrument the `onTouchEvent` function in the `View` class to record two actions: click and scroll. The `View` class is the parent class of all Android UI components, including widgets—buttons, text fields, etc.—and layouts. So `onTouchEvent` will be called when any of `View`'s child classes is touched. This way, we can capture touch actions with arbitrary apps without instrumenting app source code.

To uniquely identify the touched UI component and replay the action later, we fingerprint and record each UI component that is interacted with. The fingerprint consists of multiple attributes of the UI component. Depending on what is available, it can include: (1) the app name, (2) the `Activity` class name, (3) its parent class name, (4) the name of the `View` class, (5) its relative location in the layout, (6) its resource ID, and (7) a hash of the text and content description.

To protect users' privacy and avoid capturing sensitive data, we hash the UI components text and content description. We also do not record any text entered by users.

2) Crowdsourcing App Usage Collection

Any Android device running our modified Android framework would be capable of collecting interactive user traces. But to do this at a large scale, we crowdsourced the task, deploying our recording framework on the PhoneLab [45] testbed as part of an IRB approved study. 99 users using

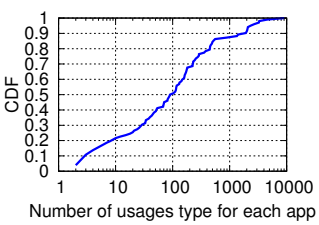


Figure 2: **Number of distinct usages for top 100 apps.**

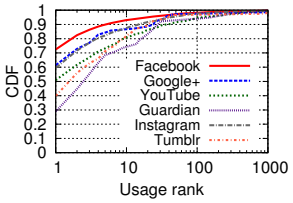


Figure 3: **Distribution of the frequency of interactions for each usage for six popular apps**

Nexus 6 smartphones running Android 6.0.1 (CyanogenMod 13.0) participated in the experiment. We recorded how they interacted with different apps for 10 days between Nov. 1st and Nov. 10th, 2016. Importantly, most users of the testbed use their testbed device as their primary smartphone, which enables us to identify which apps are most popular and which usages are most common.

3) Identifying Common App Usages

From our PhoneLab experiment, we collected input event data from 99 users interacting with 531 apps. For this analysis, we chose the top 100 apps in terms of the total number of interactions in our dataset. As illustrated in Figure 2, the number of distinct usages varies significantly across different apps. The app with largest number of usages is Facebook with 9082 different UI fingerprints that are touched by users 300K times in total.

Almost all the apps with a relatively large number of interactions but small number of usages are games and web-based apps. For the gaming apps, to display animation content and to be consistent across various app activities, all content is displayed in a custom-built `View`. For instance, for Candy Crush Saga, we only captured two usages in which a single custom View (`GameView`) is either scrolled or clicked by users. For web-based apps, all the content is displayed in a `WebView`. Since native Android widgets are not used by these two app types, we could not distinguish between different inputs, which limits our ability to replay users' interactions and create models based on usage.

Examining the frequency of the top interactions for each app allows us to identify common usages. Figure 3 shows the distribution of the frequency of interactions for each usage for six popular apps. As shown, 75% of total interactions for each app correspond to only 10 usages. This shows that we can capture how the app is being used and identify common user interactions by only considering a small number of usages. By crowdsourcing our usage data collection, we were indeed able to identify which apps are commonly used and which app usages are most common.

B. Replaying App Usage Traces

Before we can measure QoE for the usages of interest and build a QoS-to-QoE model, we need a way to replay the app

usage traces that were previously collected. To accomplish this, we built a new tool called *UsageReplayer*, which takes each individual trace as input, and replays the actions step by step to reach the particular usage of interest. The input to *UsageReplayer* is a `json` file, which contains the steps to reach the usage that have been extracted from each user input trace. Figure 4(b) shows an example `json` file to replay "scrolling on the search results of a keyword" in the YouTube app.

The `json` file consists of an array of actions. For each action, the fingerprint of its corresponding UI component is specified by the list of attributes that uniquely identify it. *UsageReplayer* measures the QoE metric for each action of interest if the value of `measure` attribute is `true`. *UsageReplayer* is an app-independent replay tool that is implemented using UIAutomator, which is an Android testing support library. This makes our approach suitable for black box testing—we do not need access to app source code.

We have published the source for *UsageReplayer* in Github [2]. Our sources include common usages with the 55 top apps collected from 99 users. This tool can be used to generate realistic smartphone application traffic by replaying and emulating how different users interact with different apps.

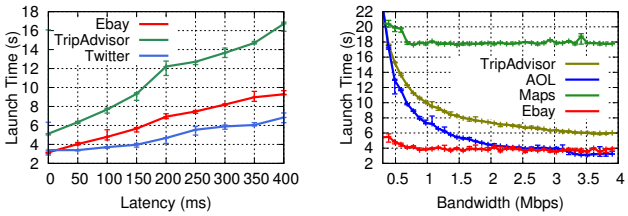
C. QoE Measurement

Our goal is to build the QoS-to-QoE model for each usage by replaying it in a testbed where we can control the network conditions experienced by the app. But first, we must be able to measure QoE for each usage, which we do using the three techniques described below.

App-independent Android instrumentation. To capture how quickly apps respond to each user input, we instrument the `onDraw` method in the `View` class. This method is invoked whenever any of the `View`'s child classes (*e.g.,* `TextView`, `ImageView`, and other UI components used by arbitrary apps) update their content. By passively monitoring the `onDraw` for a given user input, we can infer how long it takes for the app to respond and update the screen. To measure app responsiveness, we measure the time from user input to the last screen update via the `onDraw` event (we exclude UI update from Ads-related Views). This is analogous to the onload event for page load time in web browsing.

App responsiveness is a critical QoE metric [8], [49]. To the best of our knowledge, this is the first system that measures responsiveness for a large number of apps by passively monitoring changes to the screen. Differing from AppInsight [48], we instrument the Android framework as opposed to app binary. Therefore, our system does not need access to app binary and we do not need to instrument each individual app.

App source code instrumentation. In cases where the app is open source—such as AppRTC and ExoPlayer—we can instrument the source code directly. This allows us to record app statistics, debugging information, and QoE measures.



(a) Launch time monotonically increases with adding more latency.

(b) Launch time is not affected by increasing bandwidth when bandwidth is higher than a threshold.

Figure 5: **Mapping (a) latency and (b) bandwidth to launch time.**

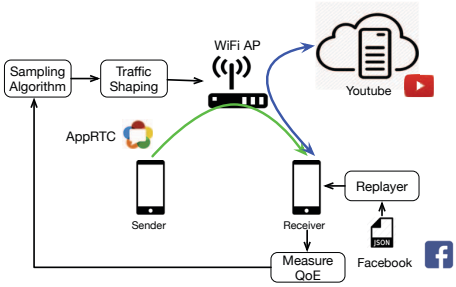
App-specific Android instrumentation. Different apps use different UI components and may have their own QoE metrics. To capture app-specific QoE metrics for these apps, we need to further instrument the Android framework in a way required by the particular app. For instance, YouTube uses Android's *throbber* widget when a video stalls. So to capture video stall events, we instrument the `ProgressBar` class. Another example is Skype. It exposes certain debugging statistics for developers using a `TextView`. Statistics shown include instantaneous QoS and QoE metrics. QoE Doctor [20] periodically records the UI tree to capture changes in app-specific UI components. In contrast, we directly instrument the corresponding classes of these components in the framework layer to monitor such changes. Our method is able to avoid the UI tree processing overhead and more accurately measure QoE metrics.

D. QoS-to-QoE Mapping

To construct the mapping from individual QoS metrics to the corresponding QoE value, we vary one QoS metric at a time, keeping the other metrics fixed.

As depicted in Figure 4a, we emulate different network settings for latency, bandwidth, and loss rate through traffic shaping using `tc` at the WiFi access point. To emulate variable latency, we add extra delay to downlink packets using `netem`. To emulate bursty packet loss, we also use `netem`, which allows us to specify the percentage of packets to be randomly dropped, and how much dropping a packet should depend on its previous packet [11]. We run every app so that it can communicate with its own app server. Where necessary (*e.g.,* for collaborative apps such as those that offer video conferencing), we run multiple clients to mimic the operation of the app. We then measure its corresponding QoE value at the client using the techniques described above. For each network setting, we wait until QoE stabilizes, as there might be a delay during which the app tries to adapt to new network conditions.

The mapping has to be generated again when new features are added to the apps. Given the infrequent updates of the



(a) Traffic shaping at WiFi AP applied to different types of apps including video conferencing, VOD, and interactive apps.

```
{"app": "YouTube",
"package": "com.google.android.youtube",
"name": "scroll_search_results",
"actions": [
  {"action": "click", "findby": {"id": "com.
      google.android.youtube:id/menu_search",
       "desc": "Search"}},
  {"action": "search", "findby": {"id": "com.
      google.android.youtube:id/
      search_edit_text"}, "text": "election"}
      ,
  {"action": "scroll", "measure":true, "
      findby": {"id": "com.google.android.
      youtube:id/results"}}
 ]
}
```

(b) An example `json` file to replay "scrolling on the search results of a keyword" in the YouTube app

Figure 4: **Our experimental setup for generating QoS-to-QoE mappings**

popular apps[1] and the fact that model generation is done automatically and in an offline fashion, updating the model for each usage requires minimal user involvement and does not need to be repeated frequently. We discuss how we can distinguish between the traffic of different versions of an app in §IV-A.

We now describe how we generate this mapping for the usages collected from our user traces. We exclude YouTube, Hangouts, and Skype and generate their models separately, as their QoE metric is different from the rest of the usages.

1) Interactive apps

To create the model for the most common usages of each app, we picked the top three usages to replay for each app, which covers 74% of total user interactions. We excluded gaming apps and interactions for which we could not identify a UI component during replay—usually because a fingerprint failed to uniquely identify a UI component. We created replayable `json` files for 56 apps and 186 usages, which includes 83 scrolls, 47 clicks, and 56 launches.

For these 56 apps, we generate the model for three usage types: scroll, click, and cold start launch. For these usage types, app responsiveness—the time it takes for the app to update the screen—is the key QoE metric [8], [49]. To capture responsiveness, we use `onDraw` events as previously explained. We generate the model by replaying these usages under various QoS values. To construct the mapping, we vary one of the QoS metrics (*i.e.,* bandwidth, loss, and latency) at a time, while keeping the other metrics fixed.

To determine whether a usage is directly affected by latency, we compute Spearman's rank correlation coefficient between latency and app response delay. As shown in Figure 5a, we increase the downlink latency at 50ms granularity and measure its corresponding QoE value. Here, a high correlation coefficient indicates the monotonicity of the increase in app response delay when increasing end-to-end network delay. We find that for 49% of clicks and 85% of launches, the correlation coefficient between app response delay and latency is higher than 0.9. This indicates that most launches are latency sensitive. However, only 32% of the scrolling usages are latency sensitive. We find that for most of the scrolling usages with a low correlation coefficient, the app does not download the data while scrolling. Instead, it fetches the content when initially loading the `Activity`. However, latency-sensitive scrolls do lazy-loading and fetch content as the user scrolls.

To see how app response delay is affected by bandwidth, we increase the bandwidth, starting from 300Kbps, and measure its corresponding app response delay value. As shown in Figure 5b, we observe that QoE is not affected by increasing the bandwidth when bandwidth is higher than a specific value. We find that these values—which specify the bandwidth requirement of each usage—are in fact small (median bandwidth is around 2.2Mbps for launch and 1.5Mbps for click). This is attributed to the fact that most of mobile apps' uplink and downlink transfers are small (less than 100KB [30]), and under high bandwidth conditions, latency plays a more important role in determining short-lived flows' performance than bandwidth does.

As we have shown, the QoS requirement of each usage can be derived from its QoS-to-QoE model. In §V-A, we will show how leveraging this information can help ISPs improve QoE of latency sensitive usages.

2) Video streaming and video conferencing apps

Video conferencing and on-demand video streaming are two popular types of apps with different QoE metrics

[1]According to a study by McIlroy *et al.* [43], only 14% of popular apps are updated bi-weekly or more frequently and only 35% of updates add new features to the apps.

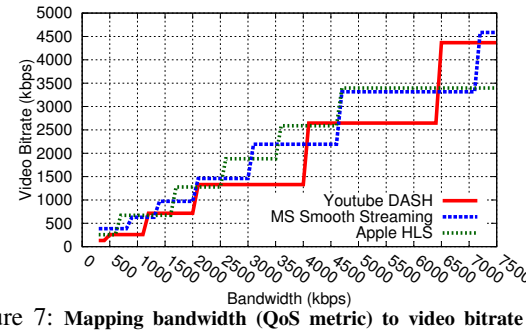


Figure 7: **Mapping bandwidth (QoS metric) to video bitrate (QoE metric) for Youtube, Microsoft Smooth Streaming, and Apple HLS**

and requirements. For video conferencing, frame rate and video/audio delay are the key QoE metrics and have direct relationships with user satisfaction [53], [55]. For video streaming, video bit-rate and rebuffering frequency are the key QoE metrics [33], [40], [15], [52]. For video conferencing, we generate models for two popular apps: AppRTC [12] and Skype. AppRTC is a video conferencing app developed by Google. It uses Chrome's native WebRTC implementation and shares the same WebRTC code base as Google Hangouts. For on-demand video streaming, we use the ExoPlayer library [4]. It provides a pre-built video player for Android using DASH and is currently used by YouTube and Google Play Movies [5]. Using ExoPlayer, we can play three state-of-art HTTP-based adaptive streaming schemes: YouTube DASH, Apple HLS, and Microsoft Smooth Streaming.

To minimize disruption, both apps adapt their QoE to variations in QoS. AppRTC and Skype adapt their encoded and decoded frame rate to respond to available bandwidth. As shown in Figure 6, various QoS metrics affect QoE differently. Since both Skype and AppRTC use Forward Error Correction (FEC) techniques [53] they can tolerate some amount of packet loss—up to 5%. However, both are highly sensitive to bandwidth variations.

For all three metrics, we observe that changes in QoS do not necessarily lead to changes in QoE. For example, video frame rate changes only at certain transitions in QoS. We made the same observation for all the video steaming schemes shown in Figure 7. Here the relationship between QoS and the QoE metric (video bitrate) is even more discrete[2]. This is particularly important to consider when performing traffic management, since the impact of changing bandwidth on app QoE is important for network operators. In §V-B we show how ISPs can tune QoS to control QoE by using the QoS-to-QoE mapping for the corresponding app.

For on-demand video streaming, rebuffering ratio and bitrate switches are the other key QoE metrics. We observe that the duration and frequency of rebuffering events depend on the degree to which the bandwidth is changed and current

[2]These mappings are consistent across different videos, as video streaming services usually transcode uploaded videos into a specific set of bitrates.

Algorithm 1 Adaptive sampling of QoS metric space

```
1:  procedure SAMPLE(n-dim space R)        ▷ n QoS metrics with arbitrary range r
2:      NewSubSpaces ← {...}
3:      for each r_i do
4:          if r_i ≤ Thresh(i) then
5:              R_i1, R_i2 ← divide r_i by 2
6:              NewSubSpaces.append(R_i1, R_i2)
7:      if len(NewSubSpaces) = 0 then
8:          return
9:      else
10:         for each R_i in NewSubSpaces do
11:             BadQoESamples ← 0
12:             for each Edge e_j do        ▷ Each Space R has 2^n edges
13:                 if QoE(e_j)=Bad then
14:                     BadQoESamples ← BadQoESamples +1
15:             if 0 < BadQoESamples/2^n < 1 then
16:                 SAMPLE(R_i)
```

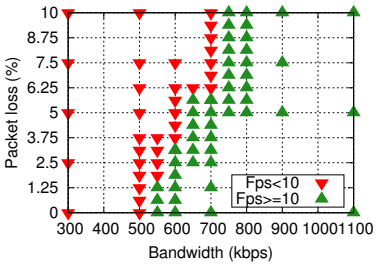


Figure 8: **Sampled QoS values based on Algorithm 1**

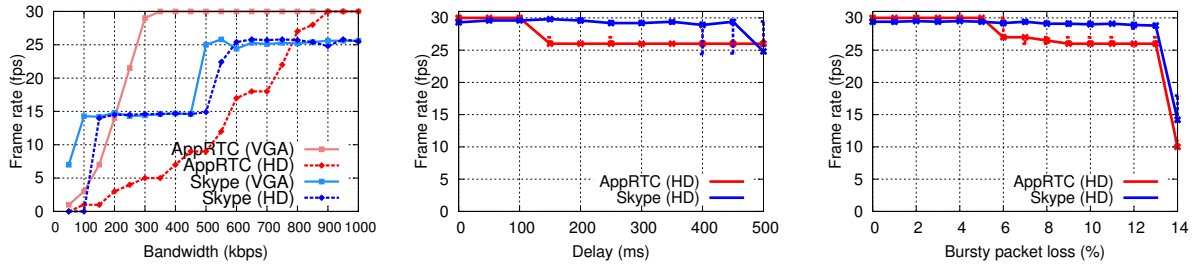
buffer occupancy. Due to lack of visibility into the current buffer occupancy, our model cannot be used to precisely estimate the rebuffering ratio. In our experiment, since the bandwidth is gradually increased by 100Kbps for each sample, we did not observe any rebuffering events. Inferring the number of bitrate switches requires keeping track of the bitrate of each user, which would be unscalable to do so. Thus, our proposed approach can only be applied to infer video bitrate.

E. Adaptive Sampling of the QoS Metric Space

While we have shown how to map from individual QoS metrics to the corresponding QoE value, constructing a precise model

$$QoE = f(bw, delay, loss\_rate)$$

requires emulating all combinations of QoS values. However, as QoS metrics are continuous variables, experimenting with all possible combinations is impractical. We propose a sampling technique to find important combinations of QoS values. We argue that we can map QoE values to a limited set of QoE classes. In fact, customer satisfaction models typically follow a threshold-based approach to distinguish between various levels of customer satisfaction and dissatisfaction. For example, if frame rate is above a threshold, users may not notice any further improvement. Then, we can selectively increase our sampling of the QoS metric space close to the borders of different QoE classes.



(a) Mapping bandwidth to frame rate for VGA and HD video quality.

(b) Mapping delay (added by AP) to frame rate for VGA and HD video quality.

(c) Mapping packet loss to frame rate for HD video quality.

Figure 6: **Mapping various QoS metrics to frame rate (QoE) for AppRTC and Skype**

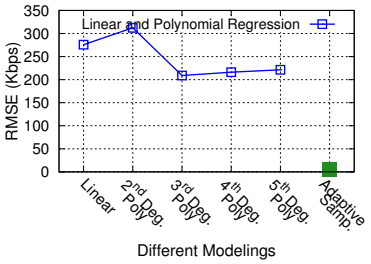


Figure 9: **Compare accuracy of adaptive sampling with regression-based modelings.**

We describe our algorithm in Algorithm 1. In this algorithm, to identify the boundary between different classes of QoE, the algorithm reduces the search space by half and if it observes samples from different QoE classes in each sub-space, it recursively calls the algorithm on that sub-space. For simplicity, we present the version of our algorithm for the case where we have a *thresholded* model for QoE [19], [13] with two classes —*Good* or acceptable and *Bad* or unacceptable—and n QoS metrics. But the algorithm is easily extensible to more than two QoE classes. We demonstrate the result of sampling for AppRTC in Figure 8. For simplicity, two QoS metrics are sampled: bandwidth and packet loss. As shown, our sampling algorithm is able to clearly identify the boundary between the two classes of QoE.

F. Evaluation

We evaluate the accuracy of the models generated by the adaptive sampling technique by comparing it with mappings that use linaer and polynomial regression. Specifically, we compare our technique with Prometheus [13], which uses linear regression (*i.e.,* LASSO regression) to map network traffic features to the binary classification of QoE. We consider the model we generated for a Microsoft Smooth Streaming video (Figure 7) to compare the accuracy of our adaptive sampling technique with linear and 2nd to 5th degree polynomial regression. To train the regression-based models, we generate the same number of randomly selected bandwidth and video bitrate samples. To measure the accuracy in terms of root mean squared error (RMSE), we used another set of random samples (20% of the size of the samples we used for training). Figure 9, shows that our model provides 98% and 97% higher accuracy for predicting the video bitrate than linear (*e.g.,* Prometheus) and 2nd degree polynomial regression, respectively. This is attributed to the fact that due to the complex interaction between app protocol and network conditions, we may not be able to generate the QoS-to-QoE mapping using linear or polynomial models.

G. Impact of System-level QoS Metrics

While we study and model the impact of network-level QoS metrics on app QoE, system-level QoS metrics (*e.g.,* CPU load, GPU, RAM) may also affect QoE. For instance, poor GPS signal can increase launch time of an activity tracker app. To understand how robust the models are to common system-level QoS variations and device hardware differences, we generated the models in two scenarios: (1) device hardware differences, for which we experimented on Galaxy S3 (2012) and Galaxy S7 (2016), and (2) resource contention from background apps, for which we used a synthetic background program to increased CPU load by 20% (the maximum increase from background apps [21]). We performed experiments for both video streaming and video conferencing, but did not find any change in the generated model. This suggests that the models generated for the apps considered in this paper are relatively robust to common system-level QoS metric variations.

We do admit that for certain apps, e.g., some gaming apps, the QoE can be affected by system-level QoS metrics such as GPU and CPU. Such models that consider both system-level and network-level QoS metrics can be utilized in OSes (*e.g.,* OS-level scheduler) and are a part of our future work.

IV. QoEBox: QoE-Aware Traffic Management

To optimize the use of their network, network operators may utilize various traffic shaping techniques. For example, an ISP may throttle flows traversing a congested link. Or, it may attempt to apply limits to certain traffic categories, such as streaming video [26].

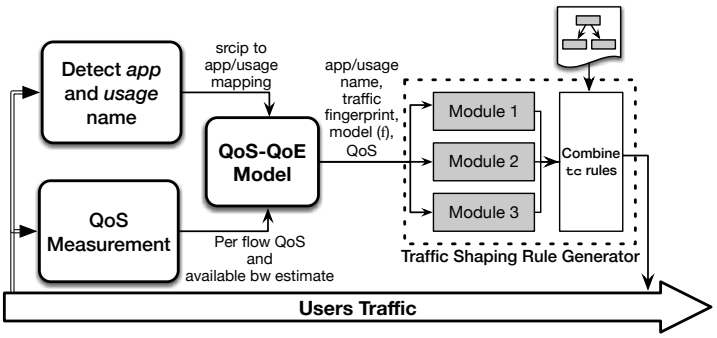


Figure 10: **QoEBox architecture.**

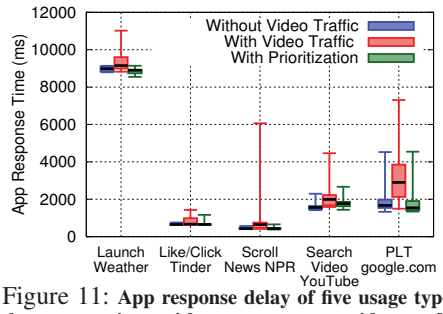


Figure 11: **App response delay of five usage types under three scenarios: without concurrent video traffic, with video traffic, and with prioritization.**

To respond to different app demands, we argue that network operators cannot treat all traffic equally. For example, in the face of congestion they should attempt to allocate network resources to minimize the degradation in QoE experienced by users. As shown by previous studies [39], [44], when QoE drops below a certain threshold, users become frustrated. They may respond by quitting the app or abandoning the service.

In this section, we demonstrate the practical utility of per-app QoS-to-QoE models for addressing these problems. We design, implement, and evaluate QoEBox, a QoE-centric traffic management framework. QoEBox is a proxy solution that can be installed on any Linux-based middlebox located on a network path carrying all traffic for one or more users. QoEBox is transparent to both user-facing apps and back-end servers. It does not need to communicate with end-hosts to get instantaneous user-perceived QoE. Instead, it relies on per-app QoS-to-QoE models, which are generated offline, to applies various traffic shaping techniques to optimize the use of network and improve end-user QoE.

As illustrated in Figure 10, QoEBox consists of three main components: app name and usage detection, QoS measurement, and traffic shaping rule generator.

A. App Name and Usage Detection

As traffic from one or more devices is routed through QoEBox, the first step is to detect app name and usage for each flow. This allows the system to find its corresponding model and apply the appropriate QoE-aware traffic shaping policies.

To map flows to apps, network operators can use existing deep packet inspection tools such as nDPI [10] or other app traffic classification techniques [54], [51]. However, we face two unique challenges not addressed by existing techniques. First, we need to detect usage as well as app name, as different usages within the same app may have a different QoS-to-QoE models. For instance, searching for videos in YouTube is latency sensitive, while watching videos is bandwidth sensitive. Second, we need to detect the app name and usage as early as possible. This is particularly important for latency-sensitive traffic, which tends to be short-lived. This gives QoEBox only a short amount of time to identify the usage and apply the proper traffic shaping policy.

Our implementation of QoEBox uses a DNS-based technique to infer the app name and usage. Our technique is based on the observation that for each usage, the set of domain names used by uplink traffic is unique to that usage. For example, when an app is launched, it may send data to various tracking and advertisement services, whereas for scrolling, it may only fetch the content that is going to be shown to user. These DNS-based fingerprints can be generated automatically by replaying usages on different devices and times. Then, the fingerprint for each usage will be the intersection of the sets of domains observed across different runs. Fingerprints for different versions of an app can be generated in a similar way. These fingerprints can be created based on other attributes, including IP address and port. Specifically, in enterprise networks, network admins may already know the fingerprints or network signature of their "managed applications" [9], [1] and these fingerprints can be provided to QoEBox as input. Using these fingerprints, QoEBox can map a set of IPs to their corresponding app/usage name.

B. QoS Measurement

To infer end-user QoE for a given app, we must be able to measure the QoS of its flows in order to apply its QoS-to-QoE model. This includes an estimate of bandwidth consumed by app, packet loss, latency, and an estimate of available link bandwidth.

Since not all network operators have visibility at the network edge, the ability to infer QoS depends on the metric and protocol specification. For example, some QoS metrics such as UDP packet loss can be measured in the core network only if the protocol exposes some information such as sequence number.

To overcome these challenges, we leverage existing tools and techniques to measure bandwidth, packet loss, and delay from passive analysis of user traffic [30]. For instance, to infer packet loss and delay from TCP traffic, we keep track of sequence/ack numbers and TCP handshake RTT. For UDP traffic, if the protocol includes timestamp and sequence

number, it is possible to estimate delay and measure packet loss. For instance, the Real-time Transport Protocol (RTP)—a popular protocol for real-time applications and used by WebRTC [12]—includes both timestamp and sequence number in the header. Even if the payload is encrypted we can still measure TCP and UDP throughput.

While measuring bandwidth, loss, and latency for TCP connections and UDP streams, QoEBox also measures available link bandwidth and reports it to the traffic shaping rule generator module. Available bandwidth is needed when multiple bandwidth-intensive usages are detected and competing for network resources. In that case, QoEBox can leverage the model to optimally allocate the available bandwidth to competing flows. In §V-B, we describe the design, implementation, and evaluation of a QoE-aware bandwidth allocation scheme that leverages this information. To estimate available bandwidth, we adopt the technique described in [28]. By passively monitoring the apps' traffic and measuring aggregated throughput of *all* TCP flows that have transferred enough bytes to exit slow-start, QoEBox can estimate the achievable bandwidth when the link is saturated.

C. Traffic Shaping Rule Generation

QoEBox allows network operators to implement various QoE-aware traffic management schemes and include their implementation as a *module*. As input, each module takes the model, as a function, per-flow QoS information, app and usage names, and their corresponding traffic fingerprint. As output, the module generates a set of `tc` rules for a *class* of traffic. In the generated `tc` rules, `filters` specify the traffic that should be processed by each class. Here, the traffic fingerprint of each usage can be included in the filters.

Modules can classify traffic based on various attributes, including app name and app type (*e.g.,* video streaming), and one module can apply its shaping on a class of traffic that is classified by another module. Thus, we have a notion of ordering between classes. We will showcase an example of this ordering in §V, in which a module first classifies the traffic as latency sensitive or bandwidth intensive, then another module applies QoE-aware bandwidth allocation only on traffic belonging to the bandwidth intensive class. To enforce ordering among different modules, QoEBox takes the relation between the modules as input and uses Hierarchical Token Bucket (HTB) qdisc to combine the generated rules from different modules.

QoEBox is designed and implemented as a middlebox that can be installed on any Linux-based machine. Depending on where QoEBox is installed (*e.g.,* cellular base-station or home WiFi access point), it can measure and control different QoS metrics. Thus, as a result of the modular design, network operators can implement and include different modules to control QoS and enforce various QoE-aware traffic management schemes on different platforms and networks.

V. Case Studies

We design, implement, and evaluate two modules for WiFi access points, as case study: a traffic classification and prioritization module, and QoE-aware bandwidth allocation module. Both make a direct use of the models generated for various types of apps. For evaluation, we cross-compile QoEBox and these modules as a package with OpenWrt SDK for Chaos Calmer (15.05.1) release. We evaluate QoEBox on a TP-Link Archer C7 WiFi router with OpenWrt 15.05.1 and we consider home network scenarios where the number of users typically ranges from 1 to 10.

A. Classifying and Prioritizing Different Traffic Types

Our goal is to properly allocate resources and satisfy usages' various QoS requirements by applying traffic shaping policies while managing competition between traffic from different usages. To do this, this module first classifies the traffic based on the app and usage's model. Traffic from different user interactions may behave differently under different network conditions. For instance, normally video streaming is delay tolerant. But at low bandwidth, as the video chunk sizes become smaller, it may also become sensitive to increases in latency. Thus, this module takes the current QoS values as an input to classify the flows. The model generated for each flow is used to determine if changing the current value of each individual QoS metric (*i.e.,* bandwidth and latency) will affect QoE.

As a result of classification, apps and usages with the same requirements will be assigned to the same class. This separation and control is necessary, as traffic from different classes can interact poorly and adversely affect each other's QoE. The poor interaction between different classes of traffic can be caused by the fact that each usage has its own QoS requirements. As a result, they tend to leverage different protocols that satisfy their needs. For instance, video streaming apps require high bandwidth to provide high quality video to users. To achieve this goal, video streaming apps use TCP, which tries to aggressively consume all the available bandwidth. However, in the presence of other classes of traffic, this strategy can adversely affect the QoE of other apps.

QoEBox classifies flows into three classes:

1) **Latency sensitive traffic** that includes various usage types, such as click, launch, and scroll in interactive apps;
2) **Bandwidth and latency sensitive traffic** that represents video conferencing apps; and
3) **Bandwidth intensive traffic** such as video streaming, foreground app installation, and file downloads.

and uses HTB qdisc in `tc` to generate the classes.

We first focus on characterizing the interaction between these three classes of users' traffic. To characterize how

bandwidth intensive traffic affects the other two traffic classes, we conduct the following experiment. First we measure the QoE of 5 devices replaying different types of latency sensitive usages. We repeat this 20 times for each usage. Then we add three more devices that play a YouTube video, representing bandwidth intensive usage. We compare the app response delay for latency sensitive usages with and without the presence of video streaming traffic. We repeat this experiment for video conferencing with AppRTC which represents bandwidth and latency sensitive usage. We then compare video quality of video conferencing with and without the presence of video streaming traffic.

Since YouTube uses TCP, it can consume all available bandwidth. We observe that this behavior affects video conferencing and interactive latency sensitive apps in two different ways. Figure 11 shows that when video streaming traffic co-exists with different latency-sensitive usages, the median, 75th, and 95th percentile of the response time for all 5 usage types increases. For example, median PLT of `google.com` rises by 73%, while 95th percentile of app response time for scrolling in NPR increases by 964%. This translates into between 1 and 6 s of extra delay. As explored in previous work [46], this increase in app delay is caused by bursty video traffic that increases queuing delay at the router buffer and corresponding end-to-end latency.

For video conferencing apps, Figure 12 shows that when other users are watching YouTube, median frame rate drops from 24 fps and 27 fps to 2 fps and 7 fps for AppRTC and Skype, respectively. For AppRTC, although the SCTP protocol aggressively tries to obtain more bandwidth, it loses the competition with TCP and cannot deliver all frames successfully. In addition, additional delay caused by queuing in the router leads to high frame drops at the receiver. Because video conferencing apps cannot tolerate delay, late packets are dropped.

Previous studies have addressed poor interaction between different traffic types by prioritizing traffic from latency-sensitive apps. QoEBox prioritizes traffic classes as follows: (1) Latency-sensitive traffic, (2) Latency and bandwidth sensitive traffic, (3) Bandwidth-intensive traffic, and (4) Background traffic, which is both latency and bandwidth tolerant. We used `prio` qdisc in `tc` to enforce these priorities.

The effects of traffic prioritization are shown in Figures 11 and 12. Figure 11 shows that latency-sensitive usages recover the performance they achieve without concurrent video traffic. Figure 12 shows that video frame rate for the video conferencing app also improve to their values without concurrent video traffic. By prioritizing the traffic of latency sensitive usages, these usages can automatically receive the amount of bandwidth they need, thus satisfying their bandwidth requirement, as explained in §III-D.

For flows with unknown models—which is caused by inaccurate app name and usage detection—we ensure that their performance will not be affected by QoEBox. To achieve this goal, QoEBox first tries to detect whether a flow with an unknown model is short-lived or long lived. If it is short-lived, it will be assigned the same priority as latency and bandwidth sensitive traffic, otherwise, it will be assigned the lowest priority. In terms of bandwidth, it will be allocated an equal share of the available link's bandwidth.

Although prioritizing the traffic has been extensively explored in previous studies, we are the first to demonstrate its effectiveness on improving QoE of various types of apps and usages.

B. QoE-Aware Bandwidth Allocation

One observation arising from the model we built for video streaming and conferencing apps is that a change to QoS does not necessarily lead to a change in QoE. For all the three adaptive video streaming schemes in Figure 7, the video bitrate is discrete. One immediate implication of the model is that allocating more bandwidth to video streaming apps does not necessarily improve the video bitrate. Moreover, applications do not seem to control the amount of bandwidth they consume. This is due to use of TCP at the transport layer, which may try to download chunks as fast as possible and consume a large amount of bandwidth, and that amount of bandwidth might be more than enough for the selected bitrate.

To understand this effect better, we did an experiment with YouTube. Based on the YouTube model in Figure 7, 2.5Mbps and 3.5Mbps will lead to the same bitrate. But the app does in fact consume its total allocated bandwidth. YouTube consumes the allocated bandwidth to download video chunks faster, but the extra bandwidth does not result in a higher bitrate. Thus, allocating the optimal amount of bandwidth to each user is necessary.

To motivate the need for limiting the bandwidth of each user, we run an experiment with 10 devices simultaneously streaming a 10 min HLS video. We first limit the total bandwidth to 20Mbps and compare the mean video bitrate and rebuffering duration with the case where each device is individually limited to 2Mbps. We find that when limiting the bandwidth of each user separately, the mean video bitrate drops by 10%. However, the mean rebuffering time improves from 18 s to no rebuffering. We attribute this to the fact that due to the buffering, downloading consecutive video chunks exhibits an ON-OFF pattern. When multiple capacity-based ABR clients are streaming videos simultaneously, the overlap between these ON-OFF patterns may cause devices to overestimate the available bandwidth. This causes them to switch to a too-high bitrate, which eventually leads to rebuffering events.

These observations lead us to derive a new bandwidth allocation scheme to improve overall QoE of video streaming apps. To allocate the bandwidth efficiently across different apps with different models, we formulate the problem as

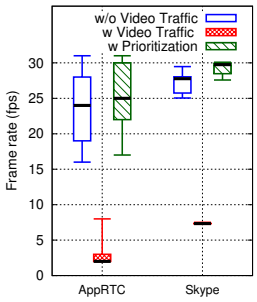


Figure 12: **Frame rate of AppRTC and Skype w/o and with video traffic, and with prioritization.**

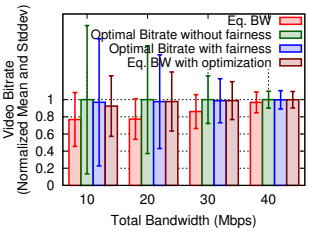


Figure 13: **Video bitrate of 10 users streaming video at random times and duration.**

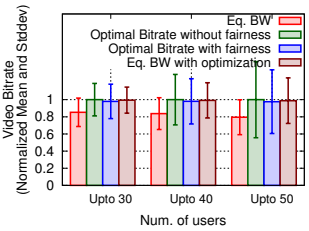


Figure 14: **Simulation results of different number of users streaming video under 100Mbps bandwidth.**

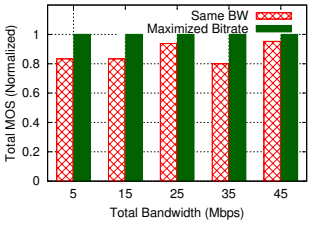


Figure 15: **Total MOS of 10 users streaming video and downloading a file at the same time.**

an optimization problem. We first use the model to find the *minimum bandwidth required* for each QoE class (for video streaming, QoE classes correspond to discrete bitrate values). To account for fluctuations in network bandwidth or any inaccuracy in the model, we add extra bandwidth ϵ[3] to the minimum value. We then form the function f to map discrete bandwidth values b to their corresponding bitrate value v. Then, for each user, we need to choose from available discrete bandwidth values to maximize the overall bitrate with the constraint that the total allocated bandwidth must be less than available bandwidth.

This problem is equivalent to the *multiple-choice knapsack problem (MCKP)* [36]. The bitrate v_{ij} corresponds to the value of bandwidth j for app i and we need to choose exactly one bandwidth value for each device. We can also incorporate additional constraints reflecting common ISP policies—limiting the total bandwidth of a particular app, or capping users' bandwidth based on their service plan.

One issue with MCKP is that it does not consider fairness, so the users of the same app may be assigned different bandwidth values. To enforce fairness across the users of the same app, we modified the MCKP formulation as follow to make the users of the same app receive equal bandwidth:

$$\text{maximize} \sum_{i=0}^{n} \sum_{j \in S_i} V_{ij} x_{ij}$$

$$\text{subject to} \sum_{i=0}^{n} \sum_{j \in S_i} B_{ij} x_{ij} \leq \text{Total}BW,$$

$$\sum_{j \in S_i} x_{ij} = 1, i = 1, \ldots, n \quad x_{ij} \in \{0,1\}, i = 1, \ldots, n, j \in S_i$$

Where:

- n is the number of apps—instead of users in MCKP;
- S_i is the set of bandwidth values available to choose for app i;
- $V_{ij} = k_i \times v_{ij}$ where k_i is the user count for app i;
- $B_{ij} = k_i \times b_{ij}$ is the total bandwidth allocated to the users of app i.

Various algorithms are available to solve the MCKP. We used a dynamic programming formulation explained in [36] to solve MCKP. Solving MCKP with dynamic programming requires finding optimal bandwidth allocations (*i.e.,* B_i for app i) for a *range* of input values, *i.e.,* available bandwidth values and apps. Using these precomputed solutions, we can update the apps' bandwidth value in response to the changes in available bandwidth, without extra computational overhead. We use HTB qdisc to limit the bandwidth of each app to its corresponding precomputed bandwidth B_i.

To evaluate how much our optimization algorithm can improve overall video bitrate, we did an experiment with 10 devices. Each device plays a video from a random app at random times and for a random duration. We perform this experiment for 1 hour. Then we compare mean video bitrates of devices using three different bandwidth allocation approaches: (1) same bandwidth for each device, (2) optimal bandwidth without fairness, and (3) optimal bandwidth with fairness. We show the results under four different bandwidth values in Figure 13. Optimal bandwidth without fairness improves mean bitrate by up to 30%, and with fairness by up to 26%. However, as can be seen, there is a high standard deviation in both approaches, meaning that MCKP may allocate high bandwidth to some users, while other users may be allocated a very low bandwidth.

To reduce this disparity, we modify the optimization formulation in the following way. First, we equally divide and allocate the available bandwidth to all the users ($\bar{b}$). Then for each user, based on its model, we find $\bar{b}$'s corresponding bitrate value v_i and then the minimum bandwidth required for v_i (b_i). Here if $b_i < \bar{b}$, we can potentially use all the extra bandwidth of $B_{\text{extra}} = \sum_i \bar{b} - b_i$ to improve video quality. To do so, we formulate the problem as a MCKP with B_{extra} as the total available bandwidth and optimally allocate this extra bandwidth to users that can use it to switch to higher bitrates. As can be seen in Figure 13, compared with

[3]for our experiments, we choose 100Kbps as ϵ

optimal bandwidth with and without fairness, this approach can achieve the lowest standard deviation in exchange for 1 to 7% lower bitrate.

We also performed a simulation of public WiFi networks, where the number of users can be higher (up to 50). As in our home WiFi network experiment, users can start watching videos at random times and for a random duration. As shown in Figure 14, with 100Mbps bandwidth, we can achieve up to 25%, 22%, and 24% higher average bitrate for the three proposed bandwidth allocation schemes.

Here we formulate the problem for video streaming apps and we maximize a single QoE metric (*i.e.,* video bitrate). In case there are multiple bandwidth intensive apps with *different* QoE metrics (*e.g.,* downloading a file in foreground and video streaming), first we need to map all various QoE metrics to a *single* metric. To do so, we can utilize existing models and subjective quality evaluation methods that provide the mapping between different app-specific objective QoE metrics to app-independent MOS (Mean Opinion Score). To evaluate our proposed bandwidth allocation scheme in such scenario, we did an experiment with 10 devices: 8 devices streaming YouTube videos and 2 devices are downloading a 10MB file[4]. We use the models that map video bitrate to MOS from [38] and download time to MOS from [24]. Figure 15 shows the results under five different bandwidth values. As can be seen, our proposed scheme improves the overall MOS by 25%.

VI. Related Work

Our work builds upon and complements a series of related work on QoE models and traffic management. In this section, we discuss some of the prior work in these areas and highlight the limitations that we address.

QoE Predictive Models. There is a rich body of work that leverages predictive models to estimate video QoE within the network [50], [18], [35], [41]. Schatz *et al.* [50] presented methods to estimate the number of stalling events and their duration for YouTube using network level measurements. Casas *et al.* [18] presented YOUQMON, which can detect stalling events in YouTube video stream by analyzing the traffic collected in 3G core network, and then map it to MOS (a subjective QoE metric). Mangla *et al.* [41] presented VideoNOC, a platform for video QoE monitoring in cellular networks. VideoNOC analyzes HTTP header information to infer various objective QoE metrics for video streaming services. Balachandran *et al.* [16] presented a data-driven approach to develop a predictive model of user engagement in video streaming services. The developed model maps quality metrics (*i.e.,* objective QoE metrics) to user engagement metrics, including viewing time and number of visits. Compared to these video streaming specific methods, our work is more broadly applicable to a wide range of apps and QoE metrics. The closest work to ours is Prometheus [13], which estimates app QoE using passive network measurement, and then uses linear regression to map network traffic features to the binary classification of QoE. In contrast, we argue that the QoS-to-QoE mapping may *not* be linear, due to (1) the complex interaction between app protocol and network conditions, and (2) the non-linear relationship between QoS metrics (*e.g.,* bandwidth) and user satisfaction [37]. Moreover, to gather training data for prediction, Prometheus relies on passive measurement of QoS from real mobile devices, while we propose an offline sampling technique that efficiently samples QoS values close to the boundary of different QoE classes. ExBox [19] is a QoE-aware admission control mechanism for WiFi networks that leverages IQX hypothesis model [25] to estimate QoE of incoming flows and then classifies them as admissible or non-admissible.

QoE Aware Traffic Management. Traffic prioritization is a known technique to mitigate in-network bufferbloat [46], [31], [29]. Prioritizing traffic from certain applications using per-class queuing is also recommended by IETF as one of the best practices for active queue management on network devices [14]. As a part of IEEE 802.11e [6] standard, Wireless Multimedia (WMM) service is proposed and supported by commercial WiFi routers to classify and prioritize the traffic of certain types of apps. However, it requires end-device input and classification. In fact WMM service has been always enabled in all our experiments. Bozkurt *et al.* [17] and Martin *et al.* [42] propose traffic management schemes for home networks that rely on users' preference and apps' input, respectively, to prioritize the traffic. The idea of application-aware adaptation through collaboration between the system and individual apps was initially proposed by Noble *et al.* [47]. In this paradigm, the OS monitors QoS metrics, notifies applications of changes in the metrics, and enforces resource allocation decisions made by the applications. Jiang *et al.* [32] propose a network paradigm where apps and network providers can collaborate by exchanging information such as QoE data. In comparison to these approaches, our proposed traffic management framework is designed for unmodified and unaware apps, *i.e.,* it does not need to communicate with end-host and it does not require users' input. Several prior efforts focus on the problem of *bandwidth allocation* for various type of apps, specifically video traffic. FESTIVE [34] is a client side solution which provides a trade-off between fairness, stability and efficiency. Q-Point [23] and QFF [27] are SDN-based approaches to address fairness and maximize aggregated QoE of multiple competing clients simultaneously watching video. In comparison with these approaches that focus on a single type of app, our bandwidth allocation technique can be applied to different types of apps. As a practical QoE-aware traffic management framework, we cover all types of

[4]Traffic of mobile video is four times the traffic of app download and file sharing [3]

apps, and classify and allocate the resources with respect to their QoS-to-QoE model.

VII. Conclusion

In this paper, we propose offline generation of per-app models mapping app-independent QoS metrics to app-specific QoE metrics. By building QOEBOX, a QoE-based traffic management framework, we show how network operators can utilize these QoS-to-QoE models to optimally allocate the resources between various types of apps and improve end-user QoE. We design, implement, and evaluate two direct applications of the model, as a QOEBOX modules for WiFi access points: a traffic classification and prioritization and an optimal fair QoE-aware bandwidth allocation scheme.

Acknowledgment

We thank the anonymous reviewers and our shepherd, Ellen Zegura, for their helpful feedback. This work is supported in part by NSF under the grants CCF-1628991 and CNS-1629763.

References

[1] Apple: Configuration Profile Reference. https://developer.apple.com/library/content/featuredarticles/iPhoneConfigurationProfileRef/Introduction/Introduction.html.

[2] *UsageReplayer* github repository. https://github.com/AndroidUsageReplayer/AndroidUsageReplayer.

[3] Ericsson mobility report, june 2016. https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-june-2016.pdf.

[4] ExoPlayer. https://google.github.io/ExoPlayer.

[5] ExoPlayer: Adaptive video streaming on Android - YouTube. https://www.youtube.com/watch?v=6VjF638VObA.

[6] IEEE SA - 802.11e-2005 – part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications - amendment 8: Medium access control (mac) quality of service enhancements. https://standards.ieee.org/findstds/standard/802.11e-2005.html.

[7] ITU-T P.1203 (11/2016): Models and tools for quality assessment of streamed media. http://handle.itu.int/11.1002/1000/13158.

[8] Measure Performance with the RAIL Model. https://developers.google.com/web/fundamentals/performance/rail.

[9] Meraki Documentation: MDM Configuration Settings. https://documentation.meraki.com/SM/Profiles_and_Settings/Configuration_Settings.

[10] nDPI: Open and Extensible LGPLv3 Deep Packet Inspection Library. http://www.ntop.org/products/deep-packet-inspection/ndpi/.

[11] netem. https://wiki.linuxfoundation.org/networking/netem.

[12] WebRTC Native Code. https://webrtc.org/native-code/.

[13] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan. Prometheus: Toward Quality-of-experience Estimation for Mobile Apps from Passive Network Measurements. In *Proc. of HotMobile*, 2014.

[14] F. Baker and G. Fairhurst. IETF Recommendations Regarding Active Queue Management . RFC 7567, Internet Engineering Task Force, 2015.

[15] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A Quest for an Internet Video Quality-of-experience Metric. In *Proc. of HotNets-XI*, 2013.

[16] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a Predictive Model of Quality of Experience for Internet Video. In *Proc. of ACM SIGCOMM*, 2013.

[17] I. N. Bozkurt and T. Benson. Contextual Router: Advancing Experience Oriented Networking to the Home. In *Proc. of SOSR*, 2016.

[18] P. Casas, M. Seufert, and R. Schatz. YOUQMON: A System for On-line Monitoring of YouTube QoE in Operational 3G Networks. *SIGMETRICS Perform. Eval. Rev.*

[19] A. Chakraborty, S. Sanadhya, S. R. Das, D. Kim, and K.-H. Kim. ExBox: Experience Management Middlebox for Wireless Networks. In *Proc. of CoNEXT*, 2016.

[20] Q. A. Chen, H. Luo, S. Rosen, Z. M. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau. QoE Doctor: Diagnosing Mobile App QoE with Automated UI Control and Cross-layer Analysis. In *Proc. of IMC*, 2014.

[21] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone Energy Drain in the Wild: Analysis and Implications. In *Proc. of the ACM SIGMETRICS*, 2015.

[22] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proc. of ACM SIGCOMM*, 2011.

[23] O. Dobrijevic, A. J. Kassler, L. Skorin-Kapov, and M. Matijasevic. Q-POINT: QoE-Driven Path Optimization Model for Multimedia Services. In *Proc. of WWIC*, 2014.

[24] S. Egger, P. Reichl, T. Hoßfeld, and R. Schatz. "Time is bandwidth"? Narrowing the gap between subjective time perception and Quality of Experience. In *Proc. of IEEE ICC*, 2012.

[25] M. Fiedler, T. Hossfeld, and P. Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24(2):36–41, 2010.

[26] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan. An Internet-Wide Analysis of Traffic Policing. In *Proc. of ACM SIGCOMM*, 2016.

[27] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming. In *Proc. of FhMN*, 2013.

[28] A. Gerber, J. Pang, O. Spatscheck, and S. Venkataraman. Speed Testing Without Speed Tests: Estimating Achievable Download Speed from Passive Measurements. In *Proc. of IMC*, 2010.

[29] Y. Guo, F. Qian, Q. A. Chen, Z. M. Mao, and S. Sen. Understanding On-device Bufferbloat for Cellular Upload. In *Proc. of IMC*, 2016.

[30] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proc. of SIGCOMM*, 2013.

[31] N. Iya, N. Kuhn, F. Verdicchio, and G. Fairhurst. Analyzing the impact of bufferbloat on latency-sensitive applications. In *Proc. of IEEE ICC*, 2015.

[32] J. Jiang, X. Liu, V. Sekar, I. Stoica, and H. Zhang. EONA: Experience-Oriented Network Architecture. In *Proc. of HotNets*, 2014.

[33] J. Jiang, V. Sekar, I. Stoica, and H. Zhang. Shedding Light on the Structure of Internet Video Quality Problems in the Wild. In *Proc. of CoNEXT*, 2013.

[34] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proc. of CoNEXT*, 2012.

[35] M. Katsarakis, R. C. Teixeira, M. Papadopouli, and V. Christophides. Towards a Causal Analysis of Video QoE from Network and Application QoS. In *Proc. of InternetQoE*, 2016.

[36] H. Kellerer, U. Pferschy, and D. Pisinger. *The Multiple-Choice Knapsack Problem*, pages 317–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[37] S. Khirman and P. Henriksen. Relationship between quality-of-service and quality-of-experience for public internet service. In *Proc. of PAM*, 2002.

[38] T. Kimura, M. Yokota, A. Matsumoto, K. Takeshita, T. Kawano, K. Sato, H. Yamamoto, T. Hayashi, K. Shiomoto, and K. Miyazaki. QUVE: QoE Maximizing Framework for Video-Streaming. *J. Sel. Topics Signal Processing*, 2017.

[39] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs. In *Proc. of IMC*, 2012.

[40] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[41] T. Mangla, E. Zegura, M. Ammar, E. Halepovic, K.-W. Hwang, R. Jana, and M. Platania. VideoNOC: Assessing Video QoE for Network Operators Using Passive Measurements. In *Proc. of MMSys*, 2018.

[42] J. Martin and N. Feamster. User-driven Dynamic Traffic Prioritization for Home Networks. In *Proc. of W-MUST*, 2012.

[43] S. McIlroy, N. Ali, and A. E. Hassan. Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store. *Empirical Software Engineering*, 21(3):1346–1370, Jun 2016.

[44] H. Nam, K.-H. Kim, and H. Schulzrinne. QoE Matters More Than QoS: Why People Stop Watching Cat Videos. In *Proc. of INFOCOM*, 2016.

[45] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen. PhoneLab: A Large Programmable Smartphone Testbed. In *Proc. of SENSEMINE*, 2013.

[46] K. Nichols and V. Jacobson. Controlling Queue Delay. *Queue*, 2012.

[47] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile Application-aware Adaptation for Mobility. In *In Proc. of SOSP*, 1997.

[48] L. Ravindranath, J. Padhye, S. Agarwal, R. Mahajan, I. Obermiller, and S. Shayandeh. AppInsight: Mobile App Performance Monitoring in the Wild. In *Proc. of OSDI*, 2012.

[49] U. Reiter, K. Brunnström, K. De Moor, M.-C. Larabi, M. Pereira, A. Pinheiro, J. You, and A. Zgank. *Factors Influencing Quality of Experience*, pages 55–72. Springer International Publishing, 2014.

[50] R. Schatz, T. Hoßfeld, and P. Casas. Passive YouTube QoE Monitoring for ISPs. In *Proc. of IMIS*, 2012.

[51] Q. Xu, Y. Liao, S. Miskovic, M. Baldi, Z. M. Mao, A. Nucci, and T. Andrews. Automatic Generation of Mobile App Signatures from Traffic Observations. In *Proc. of IEEE INFOCOM*, 2015.

[52] S. Xu, S. Sen, Z. M. Mao, and Y. Jia. Dissecting VOD Services for Cellular: Performance, Root Causes and Best Practices. In *Proc. of IMC*, 2017.

[53] Y. Xu, C. Yu, J. Li, and Y. Liu. Video Telephony for End-consumers: Measurement Study of Google+, iChat, and Skype. In *Proc. of IMC*, 2012.

[54] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, and Z. M. Mao. SAMPLES: Self Adaptive Mining of Persistent Lexical Snippets for Classifying Mobile Application Traffic. In *Proc. of MOBICOM*, 2015.

[55] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, and Y. Wang. Profiling Skype video calls: Rate control and video quality. In *Proc. of INFOCOM*, 2012.