QoE Inference Without Application Control

Ashkan Nikravesh, David Ke Hong, Qi Alfred Chen, Harsha V. Madhyastha, Z. Morley Mao
University of Michigan
{ashnik, kehong, alfchen, harshavm, zmao}@umich.edu

ABSTRACT

Network quality-of-service (QoS) does not always directly translate to users' quality-of-experience (QoE), e.g., changes in a video streaming app's frame rate in reaction to changes in packet loss rate depend on various factors such as the adaptation strategy used by the app and the app's use of forward error correction (FEC) codes. Therefore, 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 (OS). However, today, entities such as ISPs and OSes typically do not have a convenient way of obtaining input from applications on user QoE.

To address this problem, we propose *offline* generation of per-application models mapping application-independent QoS metrics to corresponding application-specific QoE metrics, thereby enabling entities (such as ISPs and OSes) that can observe a user's network traffic to *infer* the user's QoE, in the absence of direct input. In this paper, we describe how such models can be generated and present our results from two popular video applications with significantly different QoE metrics. We also showcase the use of these models for ISPs to perform QoE-aware traffic management and for the OS to offer an efficient QoE diagnosis service.

1 Introduction

For applications that users access over the Internet (e.g., video, VoIP, Web), knowledge of the user's quality of experience (QoE) is valuable in various ways. When dealing with congestion, any ISP can shape traffic in a manner sensitive to the impact that its traffic management will have on the QoE of its users (e.g., throttling every flow to only that extent that does not significantly degrade QoE for the corresponding users). An application's servers can leverage knowledge of user QoE to appropriately adapt its delivery of traffic to its users (e.g., a video service can reduce the video bitrate to eliminate the rebuffering delays being 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, application delivery adaptation, and user experience diagnosis are stymied by a basic limitation: determining a user's QoE on a particular application requires software on the user's device that is capable of measuring QoE metrics for that application and reports this information to the entity (OS, ISP, or application server) implementing the QoE-aware mechanism. This limitation stems from several reasons.

- **Application-specific QoE metrics.** The metrics that capture user QoE vary significantly across applications, e.g., rebuffering delays for video, PESQ 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 application.
- **Lack of API to communicate QoE.** In cases where the user interacts with an application via client software offered by that application's provider, that client is able to measure the user's QoE and relay such information to the application's servers. However, there typically does not exist an interface via which an application'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 application's own servers to discover user-perceived QoE because users often access applications via client software not developed by the application 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 quality of service (QoS) metrics (e.g., allocating an equal share of the bottleneck link's bandwidth to all flows, irrespective of which applications those flows correspond to) does not accurately account for

Internet-QoE, August 22-26, 2016, Florianopolis , Brazil
 ISBN 978-1-4503-4425-8/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2940136.2940145

users' quality of experience. 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 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 application-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 QoE metrics. While such a model is impractical in general, our key observation is that such QoS-to-QoE models are indeed feasible on a *per-application* basis.

In this paper, we first describe how application-specific models that map QoS metrics to corresponding objective QoE metrics can be generated. We present results for two applications: a video conferencing app AppRTC, and the video player underlying YouTube. In both cases, we find significant non-linearities between QoS and QoE, validating the need for our models. A key challenge that we tackle in generating these models is to adaptively sample the QoS metric space to obtain good coverage of all transitions where changes in QoS result in changes in QoE, without having to exhaustively cover the whole QoS metric space.

In addition, we present how the per-application QoS-to-QoE models, once generated, can be utilized in two different scenarios: QoE-aware traffic management by ISPs, and QoE diagnosis as a service by the OS. In the former case, we show how our models can enable ISPs both to identify QoE degradation for any user and to determine the minimum increase in QoS for the user's traffic that is necessary to improve the user's QoE to a desired level. In the latter scenario, our QoS-to-QoE models generated offline enable an OS-level QoE diagnosis service to efficiently identify whether network conditions are the cause for poor QoE and to trigger collection of more heavyweight logging (*e.g.,* lock contention) only when necessary.

2 Generating QoS-to-QoE models

Since the entities (such as ISPs and OSes) that could benefit from knowledge of user QoE often do not have visibility into this information, we seek to equip them with the ability to infer QoE based on network QoS metrics. A single mapping from QoS to QoE values does not exist, given the wide variance in how QoE is measured across different applications, *e.g.,* frame rate in video conferencing and page load time in web browsing. Because of the differences in the protocols used by different applications, even generating a separate QoS-to-QoE model for every *application type* is infeasible. For instance, Skype and Google Hangout use different techniques to deal with packet loss [18]; therefore, even with the same packet loss rate, users may experience different QoE when using the two applications. As a consequence, we focus on generating a QoS-to-QoE model on a per-application basis.

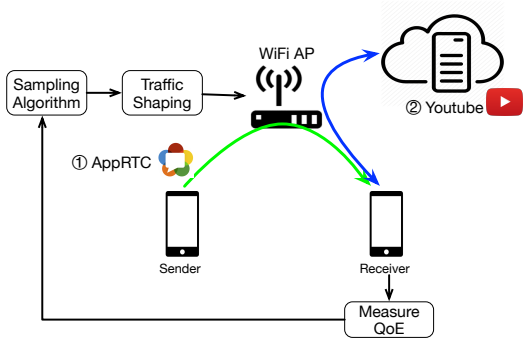


Figure 1: **Our experimental setup for generating QoS-to-QoE mappings. Traffic shaping at WiFi AP applied to (1) video conferencing using AppRTC, and (2) YouTube video streaming using ExoPlayer.**

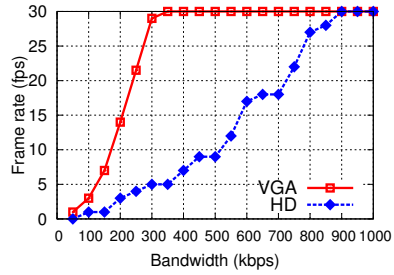
2.1 Model Generation Setup

For any app, we build the QoS-to-QoE model for it by running the app in a testbed, in which we can control the network conditions experienced by the app. Since most apps are cloud based and we lack visibility into their implementation and the protocols they use, we treat every app as a black-box and use both UI automation and app instrumentation techniques (*e.g.,* QoE Doctor [9]) to measure user-perceived QoE. As depicted in Figure 1, we run every app such that it can communicate with its own application server, as app developers may use proprietary protocols making it impossible to emulate server-side algorithms (*e.g.,* adaptation strategy). 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. In order to construct the mapping by measuring the QoE for any particular combination of values for the QoS metrics, we emulate different network settings for latency, bandwidth, and loss rate by applying traffic shaping using `tc` at the WiFi access point and measure its corresponding QoE value at the client.

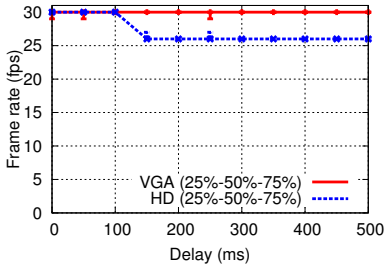
For each network setting, we wait until QoE stabilizes, as there might be a delay wherein application tries to adapt. For example, since YouTube uses the moving average of the past few chunks' throughput value to select the bitrate and may play buffered video chunks, in our experiments, on average it takes 19.62 seconds for video bitrate to stabilize after an injected bandwidth change.

2.2 QoS-to-QoE Mapping

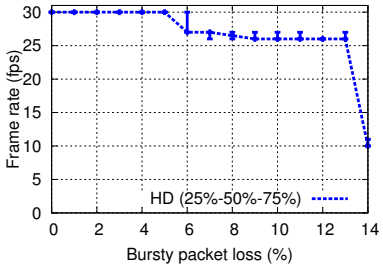
To construct the mapping from individual QoS metrics to the corresponding QoE value, we vary one QoS metric at a time, keeping the other metrics fixed. Here, we describe our experience in generating this mapping for two apps: AppRTC [5] and YouTube video streaming using ExoPlayer [1]. AppRTC is a video conferencing app developed by Google that uses Chrome's built-in WebRTC implementation and shares the same WebRTC code base as Google Hangout. In order to play YouTube videos, we use the ExoPlayer library that provides a pre-built video player for Android with DASH and is currently used by YouTube and Google Play Movies [2].



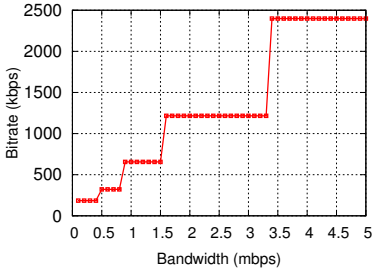
(a) **Mapping bandwidth to frame rate for VGA and HD video quality.**



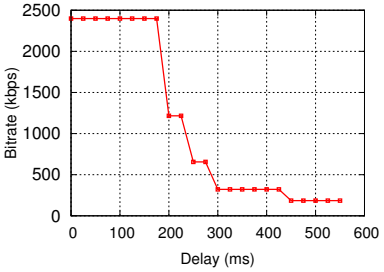
(b) **Mapping delay to frame rate for VGA and HD video quality.**



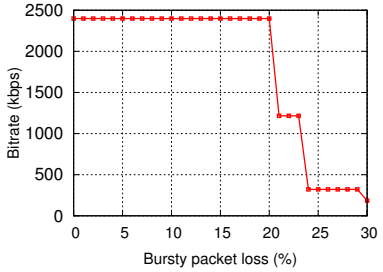
(c) **Mapping packet loss to frame rate for HD video quality.**



(d) **Mapping bandwidth to video bitrate.**



(e) **Mapping delay to video bitrate.**



(f) **Mapping packet loss to video bitrate.**

Figure 2: **Mapping various QoS metrics to frame rate (QoE) for AppRTC (a,b,c) and YouTube (d,e,f).**

AppRTC and YouTube represent two app types with different QoE metrics and requirements: video conferencing and on-demand video streaming. For video conferencing, frame rate and end-to-end video delay are the key QoE metrics, whereas for on-demand video streaming, video bit-rate and rebuffering frequency are considered as QoE metrics.

In order to minimize disruption, both apps adapt their QoE to variations in QoS. AppRTC adapts its encoded/decoded frame rate to variability in bandwidth. As shown in Figures 2(a)(b)(c), various QoS metrics affect QoE differently. Since AppRTC uses Forward Error Correction (FEC) in its encoding of media, it can tolerate considerable amount of packet loss (5%). However, it is highly sensitive to bandwidth variations. Moreover, for all three metrics, we observe that every change in QoS values does not necessarily lead to a change in QoE; the frame rate changes only at certain transitions in QoS. We made the same observation for YouTube (Figures 2(d)(e)(f)), wherein the QoE metric (*i.e.,* video bitrate) is even more discrete. This is particularly important to consider for the purpose of traffic management, where the impact of throttling or allocating more bandwidth on application's QoE is important for network operators. In §3, we show how ISPs can tune QoS metrics and control users' perceived QoE by using the QoS-to-QoE mapping for the corresponding application.

2.3 Adaptive Sampling of the QoS Metric Space

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. Hence, 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 (*e.g.,* 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.

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)
```

We describe our algorithm in Algorithm 1. For simplicity, we present the version of our algorithm for the case where we have two classes of QoE – *Bad* and *Good* – and n QoS metrics. The algorithm is easily extensible to more than two classes of QoE. We demonstrate the result of sampling for AppRTC in Figure 3. For simplicity, two QoS metrics (*i.e.,* bandwidth and packet loss) are sampled. As shown, our sampling algorithm is able to clearly identify the boundary between the two classes of QoE.

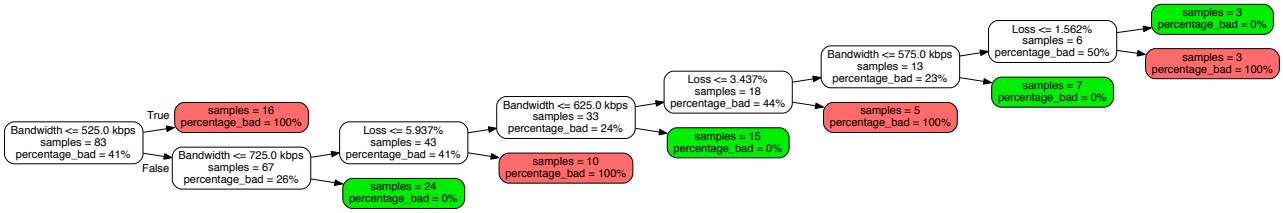


Figure 4: **Decision tree generated from QoS-to-QoE mappings for application AppRTC (Figure 3).** ***percentage_bad*** **denotes the number of bad QoE instances over total instances.**

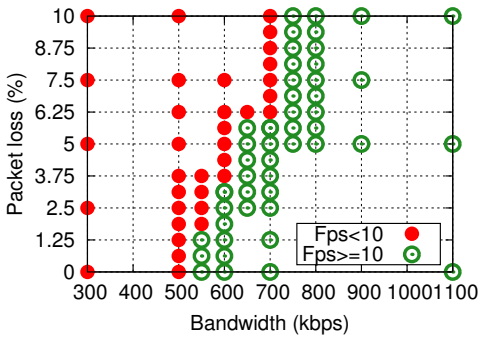


Figure 3: **Sampled QoS values based on Algorithm 1**

3 Use-Case Scenarios

In this section, we demonstrate the utility of per-application QoS-to-QoE models by applying them in two scenarios.

3.1 QoE-Aware Traffic Management

Network operators may use various traffic shaping techniques to optimize the use of their network. For example, an ISP may throttle some of the flows traversing a congested link.

We argue that network operators should not treat all traffic equally and must be responsive to dissimilar application demands. To do so, using existing deep packet inspection techniques and tools such as nDPI [3] or other application traffic classification techniques [19, 17], network operators are able to map every flow to its corresponding application. Then, in the event of congestion, network operators can choose to handle application traffic in a manner that minimizes QoE degradation for their users. For example, network operators can throttle delay-tolerant bulk software updates more in order to reduce the impact on interactive applications. In this case, per-application QoS-to-QoE mappings can help identify optimal traffic shaping parameters to minimize the impact on users.

Moreover, identifying the extent of QoE degradation from their traffic shaping measures is also important for network operators. As shown by previous studies [13, 15], when QoE degrades to below a certain threshold, users become frustrated and they may quit using the app or abandon the service. The QoS-to-QoE models can enable operators to detect when a user is experiencing bad QoE, and then identify the additional resources (*e.g.,* bandwidth) that must be allocated in order to improve QoE to tolerable levels.

3.1.1 *QoS Measurement*

To apply the model to infer QoE, network operators need to measure end-host QoS and use it as input for the model. Since network operators of all ISPs do not necessarily have visibility into the edge, the possibility of inferring end-host QoS depends on the metric and the protocol specification. For example, if QoS is measured in the cellular core network (*e.g.,* GGSN), downlink throughput of a video conferencing traffic measured by the carrier might differ from actual user-perceived throughput, as the last-mile is typically the bottleneck in cellular networks. Moreover, 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, there exist tools and techniques to measure bandwidth, packet loss, and delay from passive analysis of user traffic [11]. For instance, to infer packet loss and delay from TCP traffic, ISPs can keep track of sequence numbers and TCP handshake RTTs. For UDP traffic, if the protocol includes timestamp and sequence number, it is possible to estimate delay and measure packet loss. For instance, Real-time Transport Protocol (RTP), which is a popular protocol for real-time applications and is used by WebRTC, includes both timestamp and sequence number in the header. If the payload is encrypted, network operators can still measure TCP/UDP flow throughput.

3.1.2 *Application of the Model*

To succinctly map measured QoS values to the corresponding class of QoE, the QoS-to-QoE mappings generated as described in the previous section can be used to generate a decision tree. For example, the decision tree in Figure 4, which is generated based on the sampled data in Figure 3, identified different thresholds for bandwidth and loss rate to classify the QoS space into two classes of bad (red leaves) and good (green leaves) QoE. In the case of degraded QoE, an ISP needs to identify how much a flow's QoS must be changed to improve the application's QoE, *i.e.,* the flow's state needs to move from a leaf labeled as bad QoE to one with good QoE. Since the parent of any bad QoE leaf has at least one descendant with good QoE, we can leverage search algorithms (*e.g.,* breadth-first search) to traverse the subtree and find all the leaf nodes which are classified as good QoE. To improve QoE, the ISP must improve QoS metrics according to the thresholds in the path between the bad QoE leaf and good QoE leaf. Therefore, among all identified good

QoS metrics	Description
`upload_traffic_rate`	Upload traffic rate at device
`download_traffic_rate`	Download traffic rate at device
`thread_CPU_usage`	CPU usage by *thread*
`thread_NET_block`	Wait time for network I/O by *thread*
`thread_DISK_block`	Wait time for disk I/O by *thread*
`thread_LOCK`	Wait time on system locks by *thread*

Table 1: **Summary of system-wide QoS metrics**

QoE nodes, we select the one that requires minimal change in QoS metrics.[1]

To evaluate the utility of our use of adaptive sampling when generating the QoS-to-QoE model, we compare the accuracy of our model with an equivalent decision tree generated with the same number of *randomly sampled* data points. When we consider 20 random combinations of QoS values (25% of the size of training data), our model can achieve 10% higher accuracy than random sampling in estimating the corresponding QoE. Furthermore, since our model can identify accurate boundaries between different classes of QoE, for QoS values correctly predicted as bad QoE by both models, the change in QoS prescribed by our model is on average 29% less than with random sampling.

3.2 QoE Diagnosis as a Service

The OS at an end-user device has an accurate and holistic view of network and system-level QoS metrics, which are helpful for diagnosing QoE degradation in mobile applications. The client software for mobile applications, such as web browser, Skype or AppRTC, usually come with QoE monitoring modules. We envision that an application maintains an offline network QoS-to-QoE mapping and keeps track of its network QoS and QoE metrics during run time. An OS-based QoE diagnosis service can diagnose QoE problems using system-level QoS metrics for any observed QoE degradation which cannot be explained using the offline mapping. Once notified, the service starts to collect QoE metrics from the application and QoS metrics from the system to construct an end-device QoS-to-QoE mapping for explaining the current QoE problem. The OS also samples the runtime periodically to obtain end-device QoS metrics under normal QoE condition.

3.2.1 Methodology

By leveraging event-based runtime tracing [20, 4], the OS on a device can extract general, system-wide QoS metrics. Moreover, using feature selection techniques like decision tree, one can evaluate how well each QoS metric can distinguish QoE degradation from normal QoE and identify a specific subset of QoS metrics as preconditions for a QoE problem. This QoS-to-QoE mapping naturally pinpoints the root cause of QoE problems and may provide actionable hints for diagnosing the QoE problem.

We propose a general approach for diagnosing QoE problems as an OS service for mobile applications. First, QoS metrics are extracted from execution traces and QoE metrics are collected from the application as ground-truth labels. Then the QoS metrics along with QoE labels are fed as input to train a decision tree for classifying normal and degraded QoE instances. The tree structure defines a QoS-to-QoE mapping and the paths to leaves consisting of degraded QoE instances characterize the QoS precondition for different types of QoE problems. We choose decision tree to construct this QoS-to-QoE mapping because it is a scalable supervised learning technique and the top-to-bottom layout of its nodes corresponds to the relative importance of each QoS metric contributing to QoE.

[1]We normalized the QoS values and use Euclidean distance as a measure of the change required in QoS metrics to improve QoE.

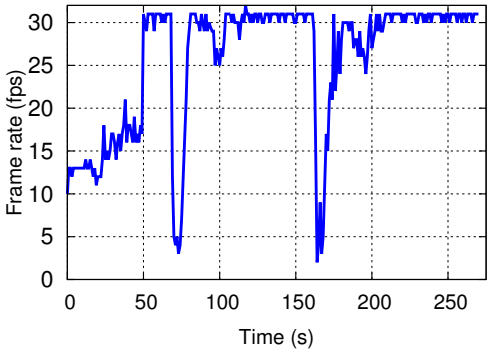


Figure 5: **Time series of output frame rate for AppRTC's HD video conferencing on a Nexus 6 phone.**

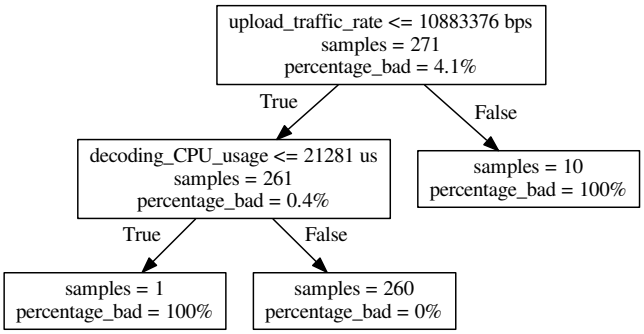


Figure 6: **Decision tree classifying normal and degraded QoE instances.**

3.2.2 Case Study

We apply our approach to diagnose QoE degradation in video conferencing through the AppRTC framework between two Nexus 6 phones in a controlled environment. Network and execution traces are collected from **`tcpdump`** and Panapptticon [20]. System-wide QoS metrics, listed in Table 1, are extracted from raw traces. QoE metrics are collected from the monitoring module of AppRTC to trigger the diagnosis. We set the time granularity over which QoS and QoE metrics are computed to be 1 second in our experiment. Finer-grained QoS metrics can be extracted for further diagnosis.

Figure 5 shows the change of output frame rate (QoE metric) over time on one device during an HD quality video conferencing session. For each time interval, we label its QoE as degraded if the number of frames encoded is less than 10 and otherwise normal. There are altogether 11 QoE degradation instances in this 271-second video session.

Using aforementioned QoS metrics, we train a decision tree (shown in Figure 6), which pinpoints two QoS metrics relevant to the QoE degradation, namely the upload traffic rate at the device and the CPU usage of frame decoding thread for AppRTC. Particularly, `upload_traffic_rate` appears as the most significant contributing factor to QoE, since over 90% of degraded QoE instances occur when the upload traffic rate at the device exceeds 10.88 Mbps. One degraded QoE instance does not associate with high upload traffic rate, but is likely caused by the insufficient CPU share allocated to the frame decoding thread of AppRTC. By further examining Panappticon traces and the device, we find a background application EventLoggingApp running on the device that periodically uploads system traces to a server, which causes network and CPU interference to AppRTC in receiving and decoding video frames. Therefore, we observe intermittent low frame rates in Figure 5.

Informed by our diagnosis, we shape background traffic by limiting its bandwidth usage below 8Mbps using **`tc`**, and discover that traffic shaping significantly mitigates video frame drops for AppRTC. Degraded QoE instances no longer occur when EventLoggingApp is uploading traces to a server.

4 Related Work

Previous work leverages predictive models to estimate QoE within the network. Schatz *et al.* [16] present methods to estimate the number of stalling events and their duration for YouTube using network level measurements. YOUQMON [7] is a system that can detect stalling events in YouTube by analyzing the traffic collected in the 3G core network, and then map it to Mean Opinion Score (MOS), which is a subjective QoE metric. Casas *et al.* [8] study the impact of downlink bandwidth and latency on the overall quality (MOS) and acceptance rate of five popular smartphone applications through a subjective study. Compared to these works that focus on video streaming specific methods or a specific set of QoS/QoE metrics, our work is more broadly applicable to a wide range of apps and QoS/QoE metrics. Since collecting objective QoE metrics (*e.g.,* MOS) from users is time consuming and lacks repeatability, our model maps QoS metrics to *objective* app-specific QoE metrics, which can be later translated to corresponding *subjective* QoE metrics using existing models [14].

The closest work to ours is Prometheus [6], which estimates application 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 application protocol and network conditions, and (2) the non-linear relationship between QoS metrics and user satisfaction [12, 10]. Moreover, Prometheus relies on passive measurement of QoS from real mobile phones, whereas we propose an offline sampling technique that samples QoS values close to the boundary of different QoE classes to more efficiently gather training data for prediction.

5 Conclusion and Future Work

We propose offline generation of per-application models mapping QoS metrics to application QoE metrics. To build such models, we design an efficient sampling technique to gather the training data for generating a decision tree based model representation, and demonstrate how such a model can be used for the purposes of traffic management and QoE diagnosis. We show that our proposed sampling technique can provide better accuracy in QoE prediction.

For future work, we are deploying the constructed model for popular applications into a public WiFi access point to evaluate the performance of the QoE-aware traffic management module. For the OS-side QoE diagnosis service, we plan to create a feedback loop where this diagnosis service can further automatically solve the diagnosed problems. One direction is to design a scheduling module at the OS level, which can allocate less CPU resources to the competing processes or prioritize the application traffic.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback. This work was supported in part by NSF under CNS-1059372 and CNS-1345226.

6 References

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

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

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

[4] Systrace. http://developer.android.com/tools/help/systrace.html.

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

[6] 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.

[7] 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.*

[8] P. Casas, M. Seufert, F. Wamser, B. Gardlo, A. Sackl, and R. Schatz. Next to You: Monitoring Quality of Experience in Cellular Networks from the End-devices. *IEEE Transactions on Network and Service Management*, PP(99), 2016.

[9] 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.

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

[11] 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.

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

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

[14] J. Morfitt and I. Cotanis. Mapping objective voice quality metrics to a MOS domain for field measurements, 2008. US Patent 7,327,985.

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

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

[17] 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.

[18] 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.

[19] 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.

[20] L. Zhang, D. R. Bild, R. P. Dick, Z. M. Mao, and P. Dinda. Panappticon: Event-based Tracing to Measure Mobile Application and Platform Performance. In *Proc. of CODES+ISSS*, 2013.