NTNU
Norwegian University of
Science and Technology

A platform for gathering insight into users experiences and their contexts on mobile devices

Stian Michael Årsnes

Master of Science in Communication Technology
Submission date: January 2017
Supervisor: Min Xie, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication

Title: A platform for gathering insight into users experiencesand their conte

Student: Stian Michael Årsnes

Problem description: Quality of Experience (QoE) is about how the user experiences or perceives a certain service or product. It can also be explained as the degree of annoyance or satisfaction of the users. We can get insight into this by collecting QoE data. QoE may be influenced by a variety of factors that are either subjective or objective to the user. One of the main methods in collecting subjective QoE data is user feedback which will be how the platform collects QoE data.

Quality of Service (QoS) is the actual performance of a system as seen by the user. QoS can consist of a large variety of metrics. QoS is often times defined by a Service Level Agreement (SLA) which is a set of guaranteed performance metrics a provider gives to its customer.

Users are not particularly forgiving. With sufficient information about them and what annoys them or makes them happy, providers can take necessary measures to ensure user satisfaction. This is becoming more and more important as users have shifted to the center of the stage and the ability to satisfy them decides which providers succeed. Our new mobile life comes with a set of challenges. There are a lot of different types of technologies, networks, hardware and software used in them. There can be great performance differences between these technology, network, hardware and software entities and when there is an annoyance it can also be difficult to pin point what and where it lies. Maybe what the users assume is poor network performance is caused by something else or vice versa?

For providers to have the necessary information to make users satisfied, they need a way to collect this information. There are not a lot of tools that let us easily collect user feedback together with QoS context data on mobile devices. The ones that do exist focus on specific areas and contexts. We want to make it as easy as possible for providers to gather QoS and QoE data for their mobile service independent of the type of service they provide. An easy way to collect such data is important so that providers can focus on making the best service/application or product (Hereafter only referred to as service).

As a user-oriented approach is becoming continuously more important among providers and developers, the importance of being able to collect user feedback in a discrete way together with associative QoS metrics is not to be underestimated. We already have an existing platform that is able to gather the user feedback data on android devices in a discrete way. It consists of an overlay button that users can

push to give feedback at the moment of annoyance. The platform also gathers some generic context data about the user.

This project will be a continuation of the existing platform. The aim is to make the platform more useful by transforming the platform into a modular platform with an API for easy implementation into any android project. The platform will be modular in the way that providers choose what modules they would like to use to track QoS context data on users mobile devices upon feedback. What is great about making the platform modular is that the platform can continuously be expanded with new modules.

A new functionality to be added to the platform is the possibility to gather and store data and events that occur within a time period. Storing collected data within a time period is useful for analyzing changes in context and how the changes affect the system. This way service providers can better pinpoint problems with their service.

A list of some modules that will be implemented:

1. **A QoE question module** which lets providers define their own questions to ask users when users give feedback.
2. **A location module** that gathers users location info within a time period. It is a separate module as location requires specific permissions from users. This info is submitted upon user feedback.
3. **A network information module** that gathers general network information within a time period from the user. This network information is submitted upon user feedback.
4. **An online debug log module** where provider submitted debug logs within a time period are gathered. This debug log is submitted upon user feedback.
5. **A video player module** that contains a video player that providers can implement into their systems in order to gather QoS metrics for videos. The video player has a connection to our platform so data from the player is collected. This module is made for testing purposes, to easily test the platform but also to potentially gather data on the video.

We will also improve the website of the platform to give more possibilities for providers to configure the platform to fit their needs. Providers will be able to choose what modules they want to be active for their users. They can also get visualization of the data that has been collected, which has been stored in the cloud. This way providers can focus more on their product and fitting it to user needs.

We will end up with a platform that gathers QoS context data together with the QoE feedback data locally on users mobile devices. We will give providers a broader image of what is affecting their users, by gathering data from provider enabled modules. The data is gathered within a given time period before user feedback submission. We will then store this data and visualize it to them on the website. This will help providers locate problems, even problems they did not even know was important to their users. That way providers can focus more on their product and fitting it to their users needs.

Responsible professor: Min Xie, ITEM
Supervisor: Min Xie, ITEM

Abstract

Users are getting more demanding every day, and it is more important than ever for providers to really understand what their users feel and think when using their product, application or service. This why Quality of Experience (QoE) has become more important the last few years. There is a variety of tools that help providers to retrieve QoE and Quality of Service (QoS) data from users, but most of them are too rigid, too specific or offer little flexibility for providers to answer questions that are important to them. There are large amounts of metadata that may be collected from mobile devices. Some of this data can be used to identify issues that cause poor user QoE. When issues are identified, they can be managed by the provider to prevent future annoyance, which will result in higher user retention. So how can we give providers the flexibility they need to associate metadata that helps them together with user feedback, so they may properly understand their users needs?

In this study we show a possible solution developed with the flexibility of the provider in mind. We developed a QoE feedback platform which allows providers to collect their own data, and answer questions that are important to them, by letting providers define their own feedback alternatives. With one line of code we provide providers with a two-way communication between the platform and Provider Applications (PAPPs). With the communication, the platform is able to receive data specific to PAPPs, which we otherwise would not be able to collect, or possibly even predict would be important to the provider. This data can be used by the providers to answer questions, and discover relationships between collected provider data and the users QoE feedback. We also give providers control over how resources are used on users devices, by letting them enable or disable what types context data the platform should gather in the background.

After developing the changes to the platform we tested it in two different user tests using two different test applications that we developed. The tests show us that the concept works, and seem to confirm that the platform is non-intrusive to users. The resulting platform has become a non-annoying, discrete, lightweight platform which provides providers with flexibility to determine how and what to measure.

Sammendrag

Brukere blir mer kravstore hver dag som går. Det er viktigere nå enn noen gang for tilbydere å forstå hva brukerne deres føler og tenker når de bruker deres produkt, applikasjon eller tjeneste. Dette er hvorfor QoE has blitt viktigere de siste par årene. Det finnes diverse verktøy som hjelper tilbydere å hente QoE og QoS data fra brukere, men de fleste av dem er altfor rigide, spesifikke eller så tilbyr de for lite fleksibilitet for tilbydere slik at de kan få svar på spørsmål som er viktige for dem. Det finnes en utrolig mengde metadata som kan hentes fra mobile enheter. Noe av dataen kan brukes for å identifisere problemer som skaper dårlig QoE blant brukere. Når problemer er identifisert kan de bli håndtert av tilbyder for å unngå fremtidig misnøye som vil resultere i at de beholder flere brukere. Så hvordan kan vi gi tilbydere den fleksibiliteten de trenger for å assosiere metdata som hjelper dem med tilbakemeldinger fra brukere, slik at de bedre kan forstå brukernes behov?

I denne studien viser vi en mulig løsning som vi utviklet med tilbyderes fleksibilitet i sinnet. Vi utviklet en QoE tilbakemeldings plattform som lar tilbydere samle egne data og som svarer på spørsmål som er viktige for dem ved å la tilbydere definere egne tilbakemeldings alternatives. Med en linje kode tilbyr vi tilbydere en to-veis kommunikasjon mellom plattformen og tilbyder applikasjoner. Med kommunikasjonen på plass kan plattformen motta data som er spesifikk til tilbydernes applikasjoner. Denne dataen kunne vi ellers ikke ha samlet, eller forutse at er viktig for tilbyder. Dataen som samles kan brukes av tilbydere for å svare på spørsmål de har, og finne viktige forhold mellom tilbydere data som er samlet og brukeres QoE tilbakemeldinger. Vi gir også tilbydere kontroll over hvordan ressurser brukes på brukernes enheter, ved å la dem aktivere og deaktivere hvilke typer kontekst data plattformen skal samle i bakgrunnen.

Etter å ha utviklet endringene til plattformen testet vi den i to forskjellige bruker tester, med to forskjellige test applikasjoner som vi utviklet. Testen viser oss at konseptet virker, og bekrefter at plattformen har blitt en ikke-irriterende, diskre, lettvektet plattform som tilbyr tilbydere fleksibilitet til å bestemme hvordan og hva som skal måles.

Foreword

Contents

List of Figures

List of Acronyms

DASH Dynamic Adaptive Streaming over HTTP.

ES6 ECMAScript2015.

GUI Guided User Interface.

HAS HTTP Adaptive Streaming.

IDE Integrated Development Environment.

JS JavaScript.

ms milliseconds.

NTNU Norwegian University of Science and Technology.

OSAPP Overlay Service Application.

PAPP Provider Application.

PCL Platform Client Library.

QoE Quality of Experience.

QoS Quality of Service.

RPC Remote Procedure Call.

SDK Software Development Kit.

SLA Service-Level Agreement.

SoC Separation of concerns.

Chapter 1
Introduction

1.1 Motivation

Users are getting more demanding every day, they are not even willing to wait more than 2 seconds for your application to launch [Res15]. It is more important now than ever for providers to really understand what their users feel and think when using their products, applications and services. This why QoE has moved further into the spotlight and become more important the last few years. There have been a variety of tools that help providers to retrieve QoE data from users, but most of them are specific to a certain type of study, rigid or offer little flexibility to providers. There are large amounts of user context metadata and QoS data that may be collected from mobile devices. Important metrics can be identified, and used to help providers make sense of user submitted QoE data and prevent future repetition of user annoyance. So how can we give providers the flexibility they need to properly understand their users needs?

We want to provide a platform which gives providers a lot of flexibility for what type of data they want to collect, and how to collect it. We want to provide further flexibility to providers by letting them define what feedback alternatives to display to their users, and what will happen after a feedback submission has been made. The

platform needs to be non-intrusive so that it does not introduce further annoyance to the users.

1.2 Contribution

1.2.1 The problem

There is a lack of QoE measurement tools that gives providers the flexibility to customize the platform to their needs and send their own provider-specific metadata. This is important so that providers can receive answers to questions that are important to them. These questions may be unrelated to video streaming, networking or other mainstream subjects which has a lot of tools. So how can we add these changes to the platform in a way that is easy for providers to use, does not effect the performance of the system significantly, and does not further annoy the user?

1.2.2 The platform

The feedback platform extends [År16], and provides some basic tools for letting providers collect user QoE feedback with some user-device metadata. However, for this platform to be truly useful for providers it needs more flexibility. This is why we made it possible for providers to define their own feedback alternatives, which are to be displayed to the user. So that providers may find answers to the questions they are curious about, rather than to be restricted by a predefined set of alternatives that give no value to the provider, and does not answer their questions. To properly evaluate the QoE collected by the device at user feedback we need to let providers provide their own PAPP specific metadata together with the user feedback. We want to develop an Android Library that we call the Platform Client Library (PCL) which lets PAPPs communicate with the platform. The communication lets PAPPs send provider-specific metadata, together with user feedback submissions. This way providers are free to send the type of data that fits their circumstances, like application specific QoS data which can help providers to find relationships between QoE and QoS metrics that are important to their specific area. These discoveries may help providers retain their users and improve their overall delight.

1.2.3 The objectives

The main objective of the thesis is to make the platform my flexible to providers. We want accomplish this by making a way for the OSAPP and the PAPPs to communicate with each other using a PCL. The secondary objective of the thesis is to develop two test applications that we can use to test the platform to make sure the concept works. Afterwards we want to test these applications with real users.

1.3 Outline

The following is an outline of the proceeding chapters of the thesis.

Chapter 2: Related work

Contains the research we did around the subject which created the basis for our work. It also contains reflections made during and after the development stages.

Chapter 3: Methodology

This chapter contains the methodologies used when working on this project.

Chapter 4: Platform Design

This chapter contains the overview of the platform as well as a description of important design decisions.

Chapter 5: Platform Development

This chapter contains more detailed information about the platform and its inner workings.

Chapter 6: Video Test Application - Simula

This chapter contains the development process of the Video Test Application that was requested by Simula, and was used for user testing the platform.

Chapter 7: DASH Video Test Application

This chapter contains the development process of the DASH Video Test Application which was developed to test the PCL and simulate a provider scenario.

Chapter 8: User Testing

This chapter contains the user testing process and result of testing the applications built during this project.

Chapter 9: Discussion

This chapter contains a set of discussion made about the platform, possible improvements and the field itself.

Chapter 10: Conclusion

This chapter contains the conclusion and suggested future work on the platform.

The Appendices

The appendices contain more detailed information about the design and development processes. It also contains visual representations of the applications and how they work.

Chapter 2
Related work

2.1 QoE and QoS on mobile devices

We use the [Qua13] definition of QoE and QoS. They define QoE as the degree of delight or annoyance of the user of an application or service. While QoS is usually used to characterize telecommunication services and their ability to live up to a certain standard described by a Service-Level Agreement (SLA), we have decided to use the extended scope defined in [Qua13] which extends the definition to include the performance aspects of any physical system. [MFTG10] on the quantitative relationship between QoE and QoS shows that there is a logarithmic relationship between them, so it is important to consider them together as QoS has an important impact on QoE. What providers want to do is map the relationship between a variety of QoS metrics to perceived user QoE. This can either be done actively by recruiting users, collecting QoS and QoE data from them and analyzing the results, or passively by using machine-learning as surveyed in [AM14] and [AW13]. [AW13] also suggests mapping models based on statistical analysis for active mapping and machine-learning techniques.

When providers have mapped QoS to QoE metrics they can start to predict the QoE of the user, hence work to avoid annoyance, increase delight and have better user retention. [AW13] states that, "the holy grail of QoE subjective measurement is to predict it from the objective measurements; in other words predict QoE from a given set of QoS parameters of vice versa". To make estimations that give better predictions it is important to be quantitative and objective when collecting QoE data as mentioned by [BH10]. [BH10] also emphasizes the importance of collecting both objective and subjective QoE data, as it gives higher validity to the QoE score and therefore also to future predictions.

The Ericsson Mobility Report [Eri16] shows that mobile traffic has almost doubled for each year since 2011, and smartphone subscriptions will increase from today 3.9 billion to 6.8 billion in 2022. This makes the mobile platform particularly important

for QoE and QoS studies. There is a vast amount of products, applications and services for mobile devices that span across several different usage categories. These usage categories range from social networking and video streaming, to banking and the weather. Mobile providers should be involved to discover metrics that are important to their field, and to discover user dissatisfaction before users flee their service.

Network service providers and researchers have focused heavily on QoE and QoS for years, in order to improve their services. Even though [Eri16] shows that 90% of users are subscribed to a 3G mobile network or better there are still challenges with varying signal strengths, cell congestion, handover between nodes and other. The platform or PAPPs can be extended to use [mob] or [ANM] to measure network QoS performance data. Good network performance is important to keep users happy. This is particularly visible for video streaming as the internet is best effort, which causes challenges for providers. [Eri16] predicts that within 2022 video will account for 75% of all mobile traffic. According to [BJVI13], "in order for services to perform well, adaptiveness was introduced", and it was introduced in a large scale. The findings of [MSS] emphasize the benefits of adaptive video streaming, indicating that adaption could reduce stalling by 80%, which is found by [FDZ11] to have the largest impact on user engagement across all types of content. There are however, also issues with adaptive streaming. Switching between video qualities as mentioned by [MSS] does affect user QoE, but more importantly the amount of time spent at a specific video quality. The authors of [BJVI13] suggested that little testing had been done, so they made their own test testing classical against adaptive video streaming. Their findings indicate that the best approach might not always be to adapt, showing that their results found the more classic approach to be better in some circumstances.

QoE is however, not only important to network service providers, and not all annoyance on mobile devices is related to network. For application providers [Res15] shows that 55% of users hold applications accountable for poor performance, not the network or the device. The survey also shows that 53% of app users will remove an application if it stops responding or crashes, 36% stop using mobile apps because of heavy battery usage and 49% expect a mobile app to respond in 2 seconds or less. [MSTG16] indicates that "monitoring network parameters is not sufficient to directly infer the resulting QoE. In contrast the application-layer monitoring, show high correlations to the subjectively experienced quality, and thus, are better suited for QoE monitoring". Letting providers send PAPP specific metadata can prove particularly useful to discover new QoE and QoS mappings. It gives a lot of flexibility to providers, and gives providers of different categories the opportunity to play along as most QoE tools are directed towards movie streaming and network metrics.

2.1.1 Testing in lab environments

Testbeds for QoE research such as [PVM08], [HL13] and [OAL] are useful for researching QoE and QoS at a general level. However, they do not truly reflect the real-world scenarios of everyday users as they roam around town, takes the bus and goes on hikes up in the mountain. We want to get closer to the user and to do that, why not then observe and receive QoE and QoS data from the users natural environment. There are several ways for providers to get in contact with real users, [THTG14] provides a set of best practices and suggests statistical methods for discovering unreliable ratings. [THTG14] also states that, "the advantages of QoE crowdtesting lie not only in the reduced time and costs for the tests, but also in a large and diverse panel of international, geographically distributed users in realistic user settings."

2.1.2 Similar tools, libraries and platforms

There are a few tools, libraries and platforms that have been developed for measuring QoE and QoS in the users own environment. [FWS15] which is a tool for analyzing QoE of YouTube HTTP Adaptive Streaming (HAS) specifically, and TeleAbarth which collects location, network and device specific data together with QoE while using any application. TeleAbarth is the most similar platform to the one we are developing and offers instant polls, which prompts users for feedback using the notification bar. It also collects data from any application and service, and it also provides flexibility to providers by letting them configure parameters and feedback questions. TeleAbarth measures more QoS data then we do, but this can be expanded into the existing platform modules. There are two main conceptual differences between TeleAbarth and our platform. The first is that we want the user to give feedback on their own terms, at the exact moment they want to give the user feedback. Instead of the provider taking charge, we let the users take charge of when they want to give feedback. To make this possible users need to have some Guided User Interface (GUI) they can interact with that is not disturbing to the user. This is where the feedback overlay button comes into play. However, we also give the possibility for providers to prompt feedback when desired. The second is that our platform provides even further flexibility to providers, letting them send their own provider-specific measurement data together with user feedback. There is also [QACL14] which is also able to get PAPP specific data, even without PAPP involvement. However, to do this they need to re-sign the apk file of the PAPP which is not preferable.

2.2 Potential use

2.2.1 Simula and the MONROE project

Simula is a research laboratory located in Oslo, Fornebu. They are currently working on the MONROE project that is a collaboration between multiple network research institutions and providers. They are researching mobile technologies. They are currently doing a set of experiments using stationary nodes in a variety of locations that stream from a DASH server. While streaming they capture the network traffic data into a pcap file and save a video corresponding to the quality delivered while streaming. They want to use this feedback platform to do experiments on the dash server.

2.2.2 Adaptable software in ubiquitous computing

The survey [KKP10] defines the ubiquitous and adaptable software as software that changes according to the context of the user. Adaptable software has the ability to change requirements during run-time, which gives the possibility to feed them with QoE and QoS data. HAS solutions are examples of adaptive technology and its capabilities. The feedback platform helps providers gather QoE and QoS metrics from their users and their contexts. This data can be used to map QoS to specific QoE metrics that can help providers realize what makes their users annoyed, so that they can improve their services by predicting QoE and adapting their software accordingly.

Chapter 3
Methodology

3.1 Literature review

To build the basis for our research we have conducted a literary review of research papers and articles we found that are related to our research challenges. They build the basis for our designs and work, so that we can develop new and interesting solutions on top of the existing ones.

3.2 Agile build methodology

We have decided to use the agile development method in order to ensure as much flexibility as possible in the software development process. We wanted the possibility to work in batches also known as sprints in agile programming, so that we could test and evaluate the progress qualitatively for each part of the development process. Also we wanted to be able to make changes, fix bugs and change direction if needed. Since android has such a vast amount of different devices it is hard to predict device specific bugs, so agile development also seems well suited for mobile development. The agile method lets us do this. The sprints have been decided between the student and the supervisor.

3.3 Testing methodologies

We have used a mix between qualitative and quantitative research methods. While developing the platform and the two test applications using the agile method we were frequently asking qualitative questions and fixing bugs as they appear in order to best deliver what the customer wants.

When an application has been ready to be tested among users we have done quantitative studies to evaluate whether the application was useful and how users evaluated the QoE of the applications. We gathered some of this data through the feedback platform itself when users gave feedback on the experiences they had using

one of the test applications. We further evaluated the applications and platform using questionnaires to better understand how the users QoE was.

Chapter 4
Platform design

4.1 Introduction

To be able to deliver a feedback platform that is flexible, easy to use and efficiently measures QoE with metadata on behalf of providers we have continued the work on the existing [År16] platform. We have also designed and developed two applications that were used in user testing, these are the "Video test application" and "DASH Video test application" respectively.

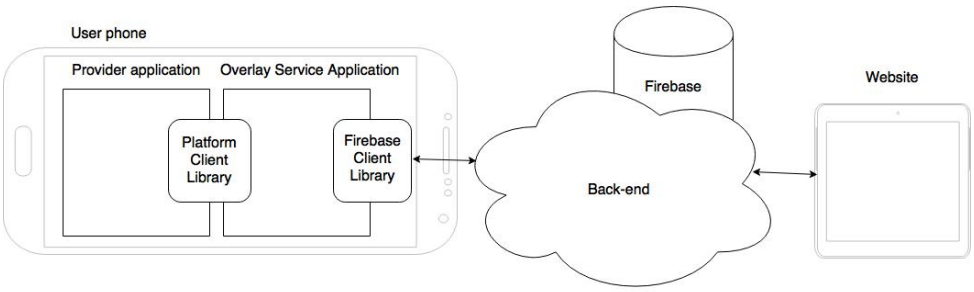

Figure 4.1: Figure of general platform design and data flows

4.2 Platform overview

The feedback platform consists of four parts as shown in figure Figure 4.1.

1. The Service Overlay Application
2. The Platforms Client Library
3. The Firebase Back-end

4. The website for data visualization and provider customization

The OSAPP(1) is an android application for mobile devices that monitors user-specific metadata and gathers user feedback upon interaction. The OSAPP(1) also communicates with any PAPPs connected to the platform using the PCL(2) developed for android. Providers can specify their own platform configurations on the platform website(4), which will be in effect whenever the application is connected to the platform. The PCL lets providers send their own monitored provider-specific metadata to the OSAPP upon customer feedback submission. Metadata is data that is specific to the context of the user, and can be used to determine the cause for a users delight or annoyance. When submitted, the user feedback and the monitored metadata is sent to the back-end(3) for storage. The user feedback, monitored metadata collected by the platform can then be viewed on the website to get a better understanding of what annoys or delights users.

4.3 The Service Overlay Application

4.3.1 Overview

The OSAPP is an android service and an application. It is developed as an android service so that it can always work in the background and be available at all times. This application is always running, even when providers remove the feedback overlay button described in [År16]. This is so that the OSAPP can always collect metadata and receive messages from PAPPs. OSAPP lets users give feedback on anything, anywhere by connecting to PAPPs using the PCL. The OSAPP consists of 4 main modules:

(a) The Feedback Module

(b) The Monitoring Module

(c) The Communication Module

(d) The Back-end Module

4.3.2 The Feedback Module

The feedback module(a) is responsible for handling the user feedback process and consists of two sub-modules, the Feedback Selection Module and the Feedback Path Module. The feedback selection module consists of the feedback GUI, a feedback overlay button and a selection menu from [År16][1]. The Feedback Path Module

[1] More about the Feedback selection module in Chapter B Section B.3.1

which we have included into the platform, handles the feedback selection menu logic and communicates with the Back-end module(d) to retrieve feedback paths and then provides this path to the feedback selection module. A feedback path[2] is the possible steps from when a user decides to give feedback until the feedback has been submitted and an action has been sent to the PAPP. To be able to take the user through provider specified feedback paths the Feedback Path Module communicates with the Back-end Module(d) in order to get provider specified feedback alternatives.

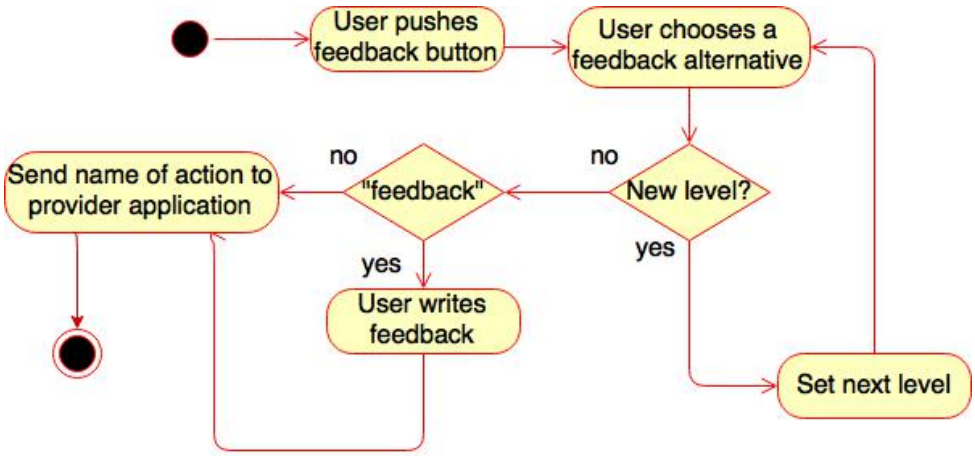

Figure 4.2: An activity diagram that shows the feedback process.

When a user wants to give feedback he interacts with the Feedback Module, which provides users with a GUI for giving feedback and choosing among feedback alternatives shown to the user. These feedback alternatives can either be generic or feedback stages defined by a provider. Provider defined feedback stages contain several levels of feedback alternatives specific to the context of the user. Whenever a feedback alternative is chosen, the next step can either lead to a new level of feedback alternatives or to an action as illustrated in Figure 4.2. An action can be to submit the feedback and continue with what the user was doing, to exit the submission or some provider defined action like "submit feedback and show movie selection screen". This can be useful in cases where the provider would like to do something in the PAPP on user feedback submission. When user feedback has been submitted, the OSAPP will use the Communication Module(3) to send the PAPP a message telling them what action the feedback process ended with.

[2]More about feedback paths in Appendix B Section B.3.2

4.3.3 The Monitoring Module

The monitoring module(b) monitors user-specific metadata and keeps a record of the last 5 minutes recorded by the module. This module has four sub-modules, the Time Collection Module, the Device Collection Module, the Location Module and the Network Module. Table 7.1 shows metadata collected by these sub-modules which has not changed from [År16] however, now they are separate sub-modules that can be expanded instead of one module that collects all the metadata. These sub-modules are responsible for collecting metadata specific to their category. The network module collects network related data, the location module collects location related data and the device module collects general device related data. We can add more sub-modules for monitoring metadata if needed. We added a Time Collection Module to initiate metadata monitoring, and to keep a record of metadata that has been monitored. It is responsible for sampling, recording and keeping monitored metadata samples of a certain time interval until users give feedback.

Table 4.1: The metadata collected by the Device Collection Module.

Android id:	A unique android id that identifies the current user.
Running apps:	A list of all the running foreground apps recorded at the moment of collection.
Phone model:	The model of the phone being used for the feedback.
Latitude:	The latitude of the user at the given time.
Longitude:	The longitude of the user at the given time.
Network type:	The type of network being used. I.e. WiFi, Mobile etc...
Sub network type:	The sub network type being used i.e. LTE, EDGE, etc...
Time:	The time a collection has been gathered.

Metadata that changes over time

Certain metadata changes over time, this is why we need to keep a record of samples collected in a time interval. So that we can attach a "user journey" to their feedback, which means that we can observe their context and how it has changed. The Time Collection Module puts a timestamp on all collections made, that can be used to identify a collection, its order among collections and determine when it was made. There are several scenarios where monitoring metadata that changes over time can be useful like for example when a user switches between two networks, the network switch could cause a delay that is responsible for the users annoyance. Therefore it is important not only to have a record of the users current network, but also the previous network the user was connected to. It is also useful to keep location data collected over a time interval, so that we can see the physical path taken by the user. If a lot of users have the same issue in the same location, or moving between the same

two nodes it could be a local issue caused by a faulty network node or something similar. By collecting application data we can find patterns and determine whether other applications or services have a tendency to interrupt or cause performance issues.

Provider flexibility

Providers have the flexibility to enable or disable what collection modules they would like to use while their PAPP is running on users phones using the website(4). This is useful for providers, so that they may decide what metadata is important to them and reduce the resource consumption of the platform. This will be even more important as functionality is added to the specific collection modules and as new collection modules are added to the platform.

Android restrictions

The idea is to be able to use the platform to monitor anything that is happening on the users devices. This is a bit problematic on Android, because Android gives us restricted access to other applications, their processes and their data. This is to avoid application exploits and data theft from malicious sources. This is why we need the communication module(c), so that we can let providers monitor PAPP specific metadata in-app themselves, or by using monitoring modules specified in the PCL. We will not make monitoring modules within the PCL in this study however, this can easily be added onto the PCL in future studies. At the moment we just want to make it possible for providers to send their own monitored metadata.

4.3.4 The Communication Module

The communication module(c) is responsible for handling communication between the OSAPP and the PAPPs. The module together with the PCL helps solve the issue of not being able to collect metadata specific to the PAPP, by allowing providers to send their own specific metadata to the OSAPP. Whenever a user is giving feedback, the OSAPP sends a message to the PAPP using the communication module to indicate that the user is about to give and submit a feedback. The PAPP can then start sending its own metadata that is to be associated with the feedback the user is about to submit. This gives providers a lot of flexibility when it comes to the metadata they can collect and associate with a feedback.

The communication module is also responsible for sending two types of events to the PAPPs, actions to be performed after user's feedback and notifications of the current state of the OSAPP. Actions are defined in the feedback paths and they are messages sent to the PAPPs after feedback submission so that PAPPs may act if needed. Notifications on the other hand tells the PAPPs what the current status

of the OSAPP is. This can be whether or not the GUI of the OSAPP is currently hidden by the user or not or whether the PAPP should start sending its data. We want to notify the PAPPs when the OSAPP GUI is hidden so that providers can adapt to the situation if desired. They can in example define a button in their own GUI which sends a message which is received at the communication module of the OSAPP, prompting it to open the feedback menu and start the feedback process.

4.3.5 The Back-end Module

The Back-end module(d) consists of two sub-modules which communicates with the Firebase [Firc] back-end, the Send Feedback Module and the Retrieve Path Module. The Send Feedback Module takes all the collected data and sends it to the back-end similarly to [År16]. The only difference is that we now have made it into its own sub-module and that we now need the option to send PAPP specific metadata.

The Retrieve Path Module however, is responsible for retrieving feedback path elements from the back-end. When a user is about to give a feedback, it is the Retrieve Path Module in correspondence with the Feedback Path Sub-module that makes sure that the user is shown the correct feedback alternatives, which corresponds with their current context. It is also responsible to retrieve any other provider specific configurations, such as what collection modules to enable or disable while the PAPP is running and what labels to put on the buttons in the OSAPPs feedback GUI.

When there is no provider connected to the overlay application the platform displays a default set of feedback alternatives to the user. However, providers have the possibility to use the new provider features of the website to define their own customized sets of feedback paths, that fit into their application contexts. The overlay application finds the fitting feedback paths from the back-end upon connection with a provider application.

4.4 The Platform Client Library

The PCL(2) is an Android library we developed that delivers tools that help PAPPs to use the platform and communicate with the OSAPP. While connected to the platform they will receive events such as actions or notifications from the OSAPP as mentioned in Section 4.3.4. They can also use the PCL to notify the OSAPP of the current stage the application is in, so that the back-end module(d) can retrieve the feedback paths associated to the specific PAPP and stage. Since it is not inherently possible for the OSAPP to collect data from the PAPPs as mentioned in Section 4.3.3 "Android restrictions", the PCL provides a way for providers to easily connect to the OSAPP, so that they can use the connection to send their own metadata to it and attach it to a user's feedback as it is being sent to the back-end for storage.

4.5 The Firebase Back-end

We use Firebase [Firc] as the backend for our platform as in [År16], the only difference is that we have updated to the new Firebase as it was recently bought by Google. It is a cloud platform, which has a set of tools and services developers can use for their projects. We use the real-time database service, which lets us easily store data in a NOSQL database and receive real-time events from it to display on the website(4). We wanted to use a NOSQL database as it is shown to be particularly useful for scaleable, fast, easy to use and high performing big data databases as shown in [LM13]. Firebase eases production by providing its own Software Development Kit (SDK) for developers as used in the Back-end Module Section 4.3.5. Another important feature of Firebase is the possibility to read and write data at real-time. This is useful in cases where knowledge of feedback is time-sensitive in order to fix issues and retain users.

4.6 The Website

The website(4) consists of three parts, one for visually displaying collected feedback and metadata, another to give providers the possibility to configure the platform according to their needs, and the last to let providers define their own feedback paths. The visualization part is based on [År16], while the configuration and feedback path parts are new additions to the website. The configuration part of the website provides a GUI to providers where they may enable or disable collection modules as explained in Section 4.3.3 "Provider flexibility". The feedback path part of the website gives providers a GUI where they can define feedback path elements such as stages, levels, feedback alternatives and actions as explained in Section 4.3.2. These parts may be extended to include more functionality as the platform is developed further.

4.7 Video Test Application - Simula

4.7.1 Introduction

The video test application was initially meant to test the platform in the MONROE project environment in collaboration with Simula. They already had nodes that measured the network performance of video streaming a set of videos from a Dynamic Adaptive Streaming over HTTP (DASH) server. These nodes collect the network traffic data of DASH video streaming sessions into pcap files and then saves the video in the quality that corresponds to the network traffic data gathered. The plan was to evaluate their data and see a pattern between the QoE data collected at the mobile devices and the different performance and QoS metrics gathered by Simulas nodes into the pcap files. This way we could evaluate whether the user tests done on the Video test application showed QoE collections that would match what we saw in

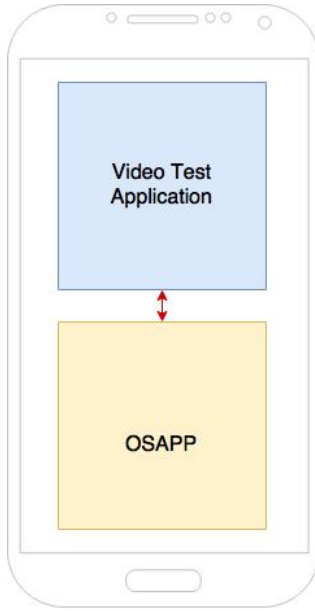

Figure 4.3: An overview of the Video Test Application and how it relates to the platform.

the pcap files of the different videos. Unfortunately there were some issues with the DASH server so we were not able to use Simulas videos for this test as intended, and therefore we were not able to look for a connection between the node collection data and feedback given by the platform.

4.7.2 Overview

The application communicates directly with the OSAPP as the PCL was not finished yet, as can be seen here Figure 4.3. Users selects a random video that it starts downloading over a HTTP connection, and then displays it to the user. The user can then choose to give a feedback using the feedback platform, whenever the user is annoyed or delighted by anything while using the application. The user can switch to another video if he wants. He switch a video either by using the menu button in the application GUI to take the user to a list of available videos, or by pushing the "Watch another video" action button during a feedback process.

Because the videos are "pre-streamed" to the nodes, we do not stream the videos to the users as this may cause uncertainty in our QoE feedback. Because the videos were already streamed once and saved according to how it performed on the node. So if we stream the videos, it would be double streaming and it would be difficult to

identify whether their annoyance is with the quality reduction as a result of streaming to the node or the reduction of quality resulting from streaming the video to the user. Because of this issue we have to download the movies in their entirety to the users before displaying them.

The idea was to use the measurements made by the node for provider specific data. However, since there was an issue with the DASH server during the development and test period we were not able to use this data in a meaningful way. Because of this, we decided to collect some provider specific data ourselves from the PAPP which in this case is the Video Test Application.

4.8 DASH Video Test Application

4.8.1 Introduction

We wanted to reduce the dependency to Simulas measurement nodes and the video streaming performance metrics gathered on them. We wanted instead to measure the performance on the devices themselves, illustrating the power of the platform and making the real world and its many mobile contexts the laboratory for our experiments.

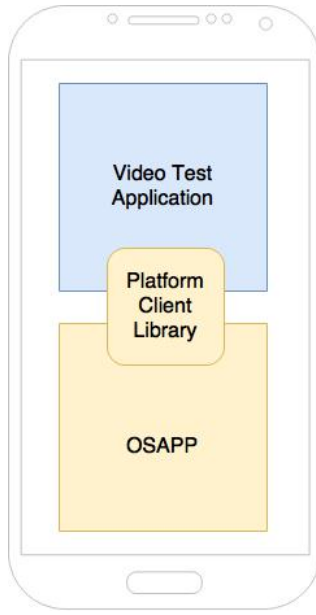

Figure 4.4: An overview of DASH and how it relates to the platform.

4.8.2 Overview

The DASH Video Test Application presents the users with several possible movie alternatives in a list. The user can then select a movie and it will start streaming. The DASH Video Test Application which is a PAPP uses the PCL to communicate with the OSAPP, as shown in Figure 4.4. The application is meant to show the potential, flexibility and value of the platform. A part of the work here is also to identify what metrics could be useful to track. The Android ExoPlayer [Exo] framework provides us with a lot of possibilities to track data from different events.

Chapter 5
Developing the platform

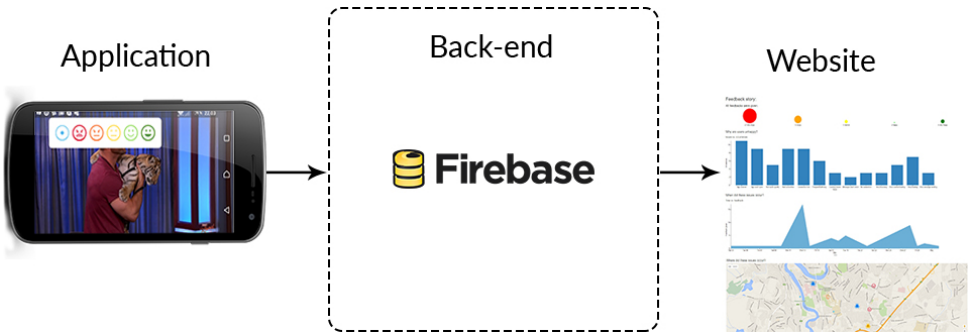

Figure 5.1: Figure of general platform design and data flows

5.1 Introduction

This chapter introduces the design and development process of the new platform features and changes.

5.1.1 Defining default actions

The default actions of the platform are:

Continue which submits the feedback and lets users continue with what they were doing.

Feedback which displays a view where users can write their own personalized feedback.

5.1.2 Defining default feedback paths

The default feedback path has only one stage that is "default".

The feedback levels contain pairs of feedback alternatives and their corresponding next path element.

- level0
  - Audio: level1audioquality
  - Other: feedback
  - Video: level1videoquality
- level1audioquality
  - Fuzzy audio: continue
  - No sound: continue
  - Other: feedback
  - Poor sound quality: continue
- level1videoquality
  - Bad image: continue
  - No video: continue
  - Other: feedback
  - Poor video quality: continue
  - Video jitter: continue

Download time too long whose next path element points to the default action "continue".

Other whose next path element points to the default action "feedback". The "feedback" action displays a view where the user can submit a written feedback.

5.2 Separation of Concerns

The original platform did not have very good Separation of concerns (SoC). To make the platform more developer friendly for future work on the platform and to make the platform more logical we have separated the code and functionality into several modules with sub-modules. Details can be found in Appendix C Section C.2. SoC or making the code more modular will make it easier to work with specific parts of the

code as concerns are logically encapsulated together into modules. We want to try to achieve high cohesion and low coupling between each module as according to [Lap07]. While high cohesion means that we want all elements within a module to have a strong relationship to each other. Low coupling means that we do not want different modules to be dependent on functionality from each other to function or do its task.

5.3 The Service Overlay Application

5.3.1 The Feedback Module

First we created necessary variables to keep track of the context and state of the OSAPP. We use the state variables to store provider information such as "customer id" and "stage" that is sent to us from the PAPP. We also keep track of the current "level" to navigate the feedback path, provider specified button labels and the provider configuration to be used for the Monitoring Module. We also keep track of whether the feedback overlay button has been hidden by the user or not, so we can notify the PAPP on connection.

The Feedback Selection Sub-module

After this we inserted the existing overlay feedback button and feedback selection menu of [År16] into the Feedback Selection Sub-module. We then made it possible to change the values of the selection menu at run-time to accommodate provider specified feedback paths. We also made it possible to hide the feedback overlay button in case it annoys a user. This is then recorded in the state variables, so that when PAPPs are connected to the OSAPP they will be sent a "removed" notification, telling them that the overlay button is currently visually hidden. Because the OSAPP is a service we can still operate in the background and the PAPPs can send it messages. When providers are notified of the button being hidden, they may implement their own feedback button that is integrated into their GUI. When that button is pushed, the PAPP may send a "prompt" message to the OSAPP which tells the feedback selection module to start the feedback process.

The Feedback Path Sub-module

Then we created the Feedback Path Sub-module which is the "brain" of the feedback selection module, it looks at the current state variables "customer id", "stage" and "level" to determine what feedback alternatives to retrieve using the back-end module. To do this the feedback path sub-module constructs and records Firebase references that point to the data in the back-end. Whenever the user goes through a feedback process using the GUI of the feedback selection module, we use the feedback path module to retrieve the next step on the feedback path. The feedback path module uses the Firebase references with the back-end module to retrieve the next step from

the back-end. Steps can either be levels leading to new feedback alternatives or actions. The feedback path module also keeps track of events made in the feedback selection module. It handles button presses as well as feedback alternative selection and decides what should happen next. Either it can send a message to the PAPP using the back-end module or retrieve the next level of feedback alternatives.

——————SE PÅ—————— The Back-end module puts the retrieved feedback alternatives into a HashMap called alternatives where the feedback alternatives are the keys, and the feedback path they lead to are the values. These feedback alternatives are then displayed to the user. To display the feedback alternatives in the Feedback Selection Module we need to have them in list form. We use the getStringArray[1] in the AlternativesHelper helper class that takes a HashMap, and returns a List of the keys in the HashMap. So when we feed our alternatives HashMap to the getStringArray we are returned a List of the feedback alternatives of the current level. We want to display the "Other" alternative as the last, this is the feedback alternative that lets users write their own personal feedback. To display the other alternative as the last we check through the List to see if it exists, if it does we remove it and add it again to the end of the list so that it will be last. After processing the list we display it by inserting it into the view as a new ArrayAdapter. The ArrayAdapter widget of the selection menu lets us display the List and handle clicks by referencing their position.

After selecting a feedback alternative using the Feedback Selection Sub-module we check the alternatives HashMap to see if the next element in the path is another level using the nextLevelCheck[2] in the AlternativesHelper helper class. If the check returns true, the next element in the path is another level of feedback alternatives to be displayed in the list. If the check returns false, the next element in the level is an action. We have two default platform actions that are "continue" that displays the continue button and "feedback" that displays the written feedback view to the user. The continue button works as the submit button for written feedback. It submits the feedback and sends "continue" action message to the PAPP. If the next path element is a provider defined action button, we retrieve it from the database using the Back-end Module.

If the next path element is another feedback level we update the state variable of the level and the database reference of where to find it. We then use the back-end module's Retrieve Path module to get the next level. If the next path element is an action we use the back-end module to retrieve it with the action button text to be displayed to the user

[1] Appendix - Development - Overlay Service Application - getStringArray
[2] Appendix - Development - Overlay Application - nextLevelCheck

The Feedback Path Sub-module is also responsible for setting the action button text. Some actions are optional to the user. This is why we developed an action button that defines an optional action to the user. If the user prefers to just "continue" he can push the default continue button however, if the user wants to perform the specific action he can push the action button and the action will be sent to the PAPP to accommodate the user with the requested action.

The initial level of a stage is always "level0" and after submitting a feedback this is the value we reset the level variable to. However, in cases where a user has chosen a feedback alternative that leads to a new level, all we have to do is to change the level variable and retrieve the new set of feedback alternatives that is associated to that particular feedback level by using the Back-end Module on the reference.

```
pathsGroupRef = database.getReference("provider_paths");
actionsGroupRef = database.getReference("actions");
buttonGroupRef = database.getReference("buttons");
configurationGroupRef = database.getReference("configurations");
```

5.3.2 The Monitoring Module

First we separated the metadata collected by [År16] into the Device Collection Module, Network Collection Module and Location Collection Module. We then made the Time Collection Module, which initiates scheduler as a Executors.newSingleThreadScheduleExecutor() which "creates a single-threaded executor that can schedule commands to run after a given delay, or to execute periodically".[3]. We then use the schedulers scheduleAtFixedRate method to collect samples every 30 seconds. We chose this rate so that we do not consume too unnecessary resources, but still are able to get a pretty good idea of what has changed in the user context, without loosing too much information between collections.

When the Monitoring Module is started by the OSAPP it needs the service context in order to collect data as well as a configuration. The configuration is provider specific and is retrieved from the back-end. It contains a set of Boolean values that determine what collection modules should be run by the scheduler.

The Time Collection Module adds the collected metadata into an ArrayList. The ArrayList should contain the samples taken every 30 seconds of the last 5 minutes. On scheduler task execution we check to see if the size of the ArrayList containing the collection data is less than 10, if it is, we add the new collection to the ArrayList. Otherwise, we remove an element from the list before adding a new collection to it.

[3] [Andc]

```
scheduler.scheduleAtFixedRate
  (new Runnable() {
      public void run() {
        if (location) {
        locationData = new LocationData(context);
        } else {
        locationData = null;
        }

      ...
      if (collectionDatas.size() < 10) {
        collectionDatas.add(new CollectionData(locationData, networkData, appData
        position++;
      } else {
        collectionDatas.remove(0);
        collectionDatas.add(new CollectionData(locationData, networkData, appData
      }
  }
}, 0, 30, TimeUnit.SECONDS);
```

To simplify the ArrayList and to avoid nesting collections we defined the CollectionData model, it is a model that is fed the data of the different collection models. It then generates a timestamp to determine the time the metadata was collected. The timestamp is later used as a key for the data collected.

The monitoring module is quite lightweight as it stores samples taken into an ArrayList rather than to store them in a local or external database which could consume unnecessary device resources. There is also a limited amount of samples so we do not consume too much memory. The ArrayList containing metadata is sent together with the user feedback on submission. We only send the ArrayList of the last 5 minutes, which makes sure that we do not overflow the database with useless data. The ArrayList approach is also easy to develop which is another benefit of it. However, in case of disruption of service we are not able to retrieve this data anymore as it was only located in the memory connected to the process.

5.3.3 The Communication Module

We then developed the communication module of OSAPP by following the documentation and examples by Samsung [Sam] and Android [Andg] on communication between remote services and activities. We changed the manifest to support communication, by exposing our OSAPP remotely and letting other applications connect to it by binding to it.

```
<service android:name="no.ntnu.stian.evryback.FeedbackOverlayService"
  android:exported="true"
  android:process=":exported">
  <intent-filter>
    <action android:name=
    "no.ntnu.stian.evryback.FeedbackOverlayService.ACTION_BIND"/>
  </intent-filter>
</service>
```

We then define an instance of the Messenger [Andf] class whose IBinder [Ande] we can use to provide Remote Procedure Call (RPC) capabilities. We initialize the messenger with a Handler[Andd] for incoming messages. We will refer to this Handler as the incoming message handler, see Appendix C Section C.4.1. The incoming message handler extends Handler, which lets us "send and process Message objects"[4]. We override the handleMessage method of the extended Handler, which is where incoming messages are received, and where we process them. We then defined a set of communication flags to determine how to process the data we receive from the provider application, see Table 5.1 for a full list of flags. A flag is sent together with each Message object.

Table 5.1: A table showing the different communication flags used by the platform. See Appendix C Section C.4.2 for more details.

MESSAGE_TYPE_REGISTER	Initiates a connection with the platform. Contains PAPPs Messenger with Handler.
MESSAGE_TYPE_UNREGISTER	Disconnects a provider application from the platform.
MESSAGE_TYPE_TEXT	Used to transfer actions and notifications between the OSAPP and the PAPPs.
MESSAGE_TYPE_CUSTOMERID	Indicates that the Message contains the providers "customer id".
MESSAGE_TYPE_STAGE	Used to define what stage or "scenario" the application is currently in.
MESSAGE_TYPE_PROMPT	Prompts OSAPP to start a feedback process.
MESSAGE_TYPE_EXTRAS	Indicates that the Message contains provider-specific metadata.

[4] [Andd]

To initiate communication when neither the PAPP or OSAPP have each others Ibinder interfaces referencing their Messenger objects, we use the bind action that we remotely exported from the OSAPP using its manifest. When a PAPP binds to the service using bindService, they get the IBinder of the OSAPP's Messenger[5]. The PAPP then uses the IBinder to create a Messenger it can use to communicate with the OSAPP. To provide a way for the service to communicate with the application, we send a message using the MESSAGE_TYPE_REGISTER that contains the Messenger of the PAPP. When the OSAPP receives this flag,it will look for the attached Messenger of the PAPP, and add it to its list of client Messengers, so that it can use it for communicating with the PAPP.

After a connection has been initiated between the OSAPP and PAPP, we can start sending Message objects between the two. To send messages from the OSAPP to PAPPs we define a sendToActivity[6] method that takes a CharSequence as input and sends it to the PAPP using the MESSAGE_TYPE _TEXT flag[7]. We use this flag to send actions and notifications to the PAPP.

For example, at the end of a feedback path we will send the defined action to the PAPP using the sendToActivity where the action is then bundled and attached to a Message object with the appropriate flag. When sending the Message we look through the list of provider application Messengers, and use them to perform a RPC to send the Message to the handler of the PAPP connected to the platform. In cases where a PAPP has been closed or does not exist anymore, it will be removed from the list.

Platform defined events to send from the SOAPP to the PAPP's

Table 5.2 shows us the set of events that may be sent from the OSAPP to the PAPP. As mentioned in Section 4.3.4 there are two types of events, actions and notifications. Actions determine what should happen at the end of a feedback process and is the final step in a feedback path. All feedback paths lead to actions, they can either be default like "continue" and "stop", or they can be defined by the provider. This gives providers the flexibility and possibility to act after user feedback submission, and try to please the user. Notifications however, notifies the PAPPs of the current state of the OSAPP. For example, when a user pushes the feedback button, it notifies the PAPP with the "pressed" message, so that the PAPP can send its own metadata if desired.

[5]Described in greater detail in Appendix C Section C.4.3
[6]More detailed information about the sendToActivity method: Appendix C Section C.4.4
[7]It probably should have been ACTION instead of TEXT

Table 5.2: The list of events that can be sent to PAPPs.

provider actions:	Provider defined action messages.
continue:	Indicates that the user would like to continue with what he was doing.
stop:	Indicates that the user wants to exit the application.
pause:	Tells the PAPP to pause what it is doing.
removed:	Tells the PAPP that the feedback button is hidden.
pressed:	Tells the PAPP that the user has pushed the feedback button and the PAPP should start to send its metadata.

Handling other important incoming messages from the PAPP's

When a MESSAGE_TYPE_CUSTOMERID Message is received, we set the new customer id and retrieve provider-specific configurations from the back-end using the Back-end Module. When a MESSAGE_TYPE_STAGE Message is received we set the new stage and update the database reference path to where feedback levels should be retrieved from. When a MESSAGE_TYPE_PROMPT Message is received it means that the provider wants to start a feedback process. We then open the feedback dialog window of the platform, which lets users rate their mood and give feedback. Finally, we decided to have a flag for all provider-specific metadata collected by the PAPP which is MESSAGE_TYPE_EXTRAS. Because we have no way of identifying every type of data that could be interesting for the provider to collect, and we want the platform to be flexible. This metadata is to be uploaded together with a user feedback.

5.3.4 The Back-end Module

Both sub-modules of the back-end module utilizes the Firebase SDK [Firc] to communicate with the back-end. We started by making the Send Feedback Sub-module which is inspired by how [År16] sends data to the back-end. We just define the data a bit differently, the monitored metadata is now sent together as a collection under the "data_collection" path in the database.

userData

The userData model is initialized in the onCreate of the OSAPP. It requires the application context and is populated by the metadata collected by the Monitoring Module and provider-specific metadata sent by the PAPP using MESSAGE_TYPE_EXTRAS flag. We also store user feedback here as they select feedback alternatives during a feedback process. The userData object contains all metadata that is to be submitted.

The Send Feedback Sub-module

We then developed the Send Feedback Sub-module which uses the FeedbackManager class[8]. The FeedbackManager takes the userData object and the customer id as parameters, and then organizes the metadata of the userData model into a HashMap to fit the structure of the database. It extracts each collection element into separate sections or key values. We then send it to the back-end by creating a new database location using the Firebase push()[9] function, and then using the Firebase updateChildren function we update the location with the data in our HashMap.

The Retrieve Path Sub-module

We defined the getInfoFromDatabase[10] method to initialize the firebase listeners to listen to the references we made to the database. We define three listeners, a levelListener that retrieves level data, a buttonListener that retrieves button data, a configListener to retrieve provider configuration data for the Time Collection Module, and an actionListener that retrieves action data. We use the ValueEventListener [Firb] that reads and listens for changes to the entire data snapshots of a referenced path within the levelListener, buttonListener, configListener and actionListener.

Whenever we are requested by the Feedback Path Sub-Module to retrieve a level, we iterate the data snapshot of the levelListener to populate a feedback alternative HashMap called alternatives, where we have the feedback alternatives as keys and the next elements in the path as the values. If the next element is another level we run the levelListener again to get the new level data. We also iterate the data snapshot of the buttonListener and actionListener and set the button texts according to the referenced values. The configListener is started by the PAPP when it sends it customed id, then it checks the configuration reference of the PAPP that is connected to make a configuration Map of collection modules and whether or not they should be enabled.

5.4 The Platform Client Library

Can use default handler or define a custom incoming handler to receive notifications and actions upon feedback submission.

Initiating communication with the Overlay Service Application

To be able to communicate Message objects with the OSAPP we continued to follow the tutorial by Samsung [Sam], see Figure 5.2 for illustration. As mentioned earlier

[8] See Appendix C Section C.5.2
[9] [Fira]
[10] See Appendix C Section C.5.1

in Section 5.3.3, the OSAPP has remotely exported an action to bind to it, so that we can initialize communication. We use the MessagingService class, see Appendix D Section D.2.1 to bind to the OSAPP. We initialize the MessagingService through its constructor. The constructor takes the customer id, current stage, application context of the PAPP and optionally a custom Handler for incoming Message objects.

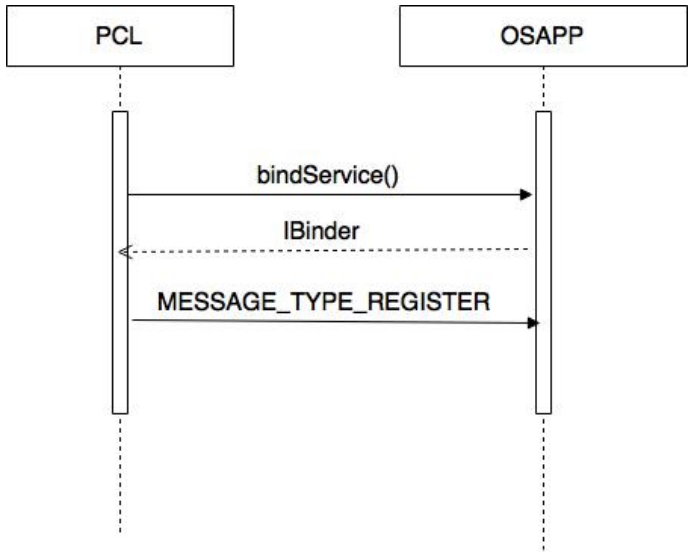

Figure 5.2: A sequence diagram showing the initiation process of the communication between the PAPP and OSAPP.

To bind to the OSAPP we use the application context of the PAPP, so that we can perform bindService[11] on it to bind to the OSAPP. bindService takes as parameters an intent that points to the package and remotely exported action of the OSAPP, a connection interface to monitor the state of the OSAPP, and a flag to indicate that the service should be automatically created as long as there is a binding that exists.

```
public MessagingService(String customer, String stage,
Context context, Handler customIncomingMessageHandler) {
        ...
        mMessenger = new Messenger(customIncomingMessageHandler);
        intent = new Intent();
        intent.setAction("no.ntnu.stian.evryback
        .FeedbackOverlayService.ACTION_BIND");
        intent.setPackage("no.ntnu.stian.evryback");
```

[11] [Andb]

```
        context.bindService(intent, serviceConnection,
        context.BIND_AUTO_CREATE);
    }
```

We then developed the connection interface of type ServiceConnection that we called serviceConnection. The serviceConnection handles the connection status of the connection to the OSAPP. On connection, the serviceConnection retrieves the Messenger of the OSAPP and saves it in a variable, so we can use it for sending Message objects. After receiving the Messenger of the OSAPP, we use the new communication channel to send a MESSAGE_TYPE_REGISTER message with a reference to the Messenger of the PAPP. This messenger either holds the default Handler for incoming messages or a provider customized version of the incoming message Handler.

Sending messages to the service

After receiving the Messenger of the OSAPP we send two messages to it. One message with the customer id of the PAPP and one to set the current stage. We do this by using the sendToService method, this is similar to the sendToActivity method of the OSAPP mentioned in "Overlay Service Application - Messenger and platform communication". The only difference is that we send it to the Messenger of the OSAPP instead. However, to be able to handle the sending of the HashMap of provider specific collection data we use the sendHashMapToService instead. It also works similarly however, it takes a parameter of type HashMap instead. This hashMap contains the provider specific collection data. The HashMap is inserted into the Bundle that is sent with the message using Bundle's putSerializable method which lets us insert serializable values into a bundle.

```
public void sendHashMapToService(HashMap<String, Object> hashMap, int sendType) {

        if (serviceConnected) {

            Message msg = Message.obtain(null, sendType);

            Bundle b = new Bundle();
            b.putSerializable("map", hashMap);

            msg.setData(b);
            ...
            }
```

```
}
```

Handling incoming messages

We have also defined a default Handler for incoming messages that we have called IncomingHandler. The IncomingHandler only logs the incoming message, so it is not very useful. Therefore we encourage providers to define their own Handler's for incoming messages. This also gives providers the possibility to handle default platform actions as well as their own customized actions.

Provider specific collection data and logging data

The providers are free to send any collection data they want as long as the data is serializable and is put into a HashMap with key-value pairs. Otherwise there are no limitations as we want the platform to be as flexible as possible in regards to providers needs. This data is sent to the OSAPP using the MESSAGE_TYPE_EXTRAS. The provider can also store log data here to be sent with the submitted feedback. We decided it was unnecessary to develop modules in the library to collect data. It can be added to the project in the future, but will make the library complexity and size larger than necessary. At the moment the library is very lightweight and flexible. It is up to the provider to decide what data they want to collect and send with the submitted feedback.

5.5 The Back-end

Back-end database structure

Because we worked on the database structure and used it before making the website, we made the structure by inserting it directly into the database using Postman. We made "default" alternatives for general use cases where no PAPP is connected to the platform. You can find the structure of the new database in "Design - Structuring the back-end database".

Providers can use REST to send data afterwards using REST

Every feedback has a unique key, by using the customers id and the unique key providers can send collection data gathered outside the mobile environment that has some association to the feedback. They can do this by sending a REST request containing the data they want to connect to their feedback by referencing the path of the feedback.

Submitting provider customization to the back-end without the website

Originally we wanted to update the existing website of the platform to handle this however, because of time constrains we were unable to do it in time. So at the moment provider customization can be done by sending an update REST request to the firebase back-end to update the locations in the database. This method can be used for customizing feedback paths and what collection modules to enable or disable.

Updating the back-end to the new firebase

Because Firebase was bought by google since the platform initially built there have been some major changes. One of them is a new developer console, when updating to the new console it is not possible to revert back to the old one.

5.5.1 Structuring the Back-end database

We have planned a database structure for how the information should be organized to be easier to use. Firebase uses a JSON structured NOSQL database.

```
"actions": {
        "provider1": {
            "action1": "String to be sendt to provider application.",
            "action2": ""
            //...
        },
        "provider2": {}
        //...
    },
    "buttons": {
        "provider1": {
            "button_bottom": "Customized exit button string.",
            "button_top": "Customized continue button string."
        },
        "provider2": {}
        //...
    },
    "configurations": {
        "provider1": {
            "apps": "Boolean to indicate whether to collect a list of running apps
            "collection": "Boolean whether to collect data at all.",
            "location": "Boolean to indicate whether to collect location data or no
            "network": "Boolean to indicate whether to collect network data or not.
```

```
    },
    "provider2": {},
    //...
},
"messages": {
    "provider1": {
        "message1": "String message to be showed to user.",
        "message2": "",
        //...
    },
    "provider2": {}
    //...
},
"provider_paths": {
    "provider1": {
        "stage1": {
            "level1": {
                "feedbackAlternative1": "Next path element",
                "feedbackAlternative2": ""
                //...
            },
            "level2": {}
            //...
        },
        "stage2": {}
        //...
    },
    "provider2": {}
    //...
}
```

5.6 The website

We started developing a simple website for provider customization. However, the customization part consists of a selection screen where you can choose between the two test providers we have included into the project. The website is still under development and did not have time to finish during the project period. So we will write down what we would have developed with enough time.

Customizing general metadata collection

There may be cases where providers want to limit the resources used by the feedback platform itself, so that the platform does not affect the application's performance. In these cases providers have the possibility to enable or disable certain metadata collection modules on the website using toggles. If i.e. a provider is only interested in location data, they can disable the network and app data modules or vice versa.

Customizing feedback paths

The provider will be able to customize feedback paths on the website. In a form or something similar the provider can customize their own stages, levels, feedback alternatives and actions. This gives providers a lot of flexibility to use the platform according to their needs and enforces the concept of being able to give feedback on anything rather than to restrict them to predetermined scenarios.

5.7 Summary

So after designing and developing changes to the platform we have ended up with a platform with four parts:

1. An OSAPP that communicates with PAPPs and lets users give feedback on anything, anywhere and at any time. We have increased the flexibility of the platform by allowing PAPPs to send provider specific collection data to the OSAPP of user QoE feedback submission.
2. A Platform Client Library that simplifies the use of the platform for providers. The library handles the communication on behalf of PAPPs and delivers a set of methods providers can use to send messages to the OSAPP.
3. A Firebase based back-end solution that lets us store feedback data and which provides real-time data events.
4. A website that lets providers customize the feedback platform according to their needs. By letting them define feedback paths and collection configurations.

Chapter 6
Developing the Video Test Application - Simula

6.1 Introduction

The Video test application has only one activity where all the code is situated, the VideoActivity class. It communicates with the OSAPP directly. We have defined a set of actions and made other design decisions you can find in Appendix E.

Table 6.1: The metadata collected by the Device Collection Module.

	General metadata collected
Current pos:	The current playback position of the video in milliseconds.
Video name:	The name of the video currently being played. Was meant to be used to locate the pcap file that was associated to the video. But ended up being just a general indicator of what video the user gave feedback to.
	Metadata relating to the "Download" stage of the application.
Buffer length:	The length of the buffer.
Total size:	The total size of the file in bytes.
Download time:	The amount of time that has spanned since the download was initiated until the feedback button was pushed.

Communicating with the Overlay Service Application

When we built the Video test application we had not yet built the Platform Client Library. Actually the Platform Client Library Messenger functionality was inspired by the work done developing this app. As with the OSAPP and the Platform Client Library we used the Samsung [Sam] and Android [Andg] tutorials to develop the communication between the OSAPP and PAPP which is the Video test application

in this case. There are no significant differences between the code in the Platform Client Library and this application.

The application has gone through a few phases, it started by sending video playback position and name with their own flags, MESSAGE_TYPE_VIDEOPOSITION and MESSAGE_TYPE_VIDEONAME. However, this was a bit unnecessary as not all applications connected to the platform are presumably going to be video related. So to reduce unnecessary communication back and forth between the OSAPP and the Video test application we started collecting the data into a HashMap and sent it using the MESSAGE_TYPE_EXTRAS flag instead.

Random video selection

Because we were not able to stream the videos we needed for the test we found a few videos on the internet that we could use instead. We put the addresses into a list and when the application starts a random number generator will determine which list index to play.

Downloading a video

Then we added the class DownloadFilesTask[1] for downloading videos asynchronously using a downloadFile method we defined that is based on an example by Android [Anda] and a post on stackoverflow[2]. When a download is started we also start a timer, so that we can measure the start time against the time a feedback is given to get the total download time for feedback submission. A HTTP connection is started between the application and the video host, then we determine where the file should be saved. After determining where the file should be saved, we record the total file size of the content and start the download using a while loop that is true until there is no more data left in the input data stream. Inside this loop we update the buffer length and downloaded size as the download progresses. We display these in a progress bar, so that the user has some context for what is happening. If a user submits a feedback these variables will be sent to the platform in a HashMap using the MESSAGE_TYPE_EXTRAS flag.

When the download has finished the onPostExecute method of the DownloadFilesTask class runs. Here we set the new stage, "playing" and send it to the platform. After sending it to the platform we start playing the video using the startMovie method we defined for initializing playback.

[1] Appendix: Classes and methods for downloading video files

[2] Stackoverflow: http://stackoverflow.com/questions/35551850/how-to-show-the-file-name-in-progressbar-for-file-uploading-in-android

Displaying a video

To display the videos we use the Android built in videoView [Andh] widget. The videoView widget simplifies the process of displaying a video. All we need to define and add to the videoView is the path of the video, a media controller to handle playback using the Android MediaController[3] widget and a setOnCompletionListener that lets us do something when the video has finished its playback. When the video has finished its playback we use the setOnCompletionListener to send the "end" stage to the OSAPP. We also prompt the user for a feedback by sending a Message object with the MESSAGE_TYPE_PROMPT flag to the OSAPP. The videoView is initialized in the playVideo and startMovie methods that are run either when a DownloadFileTask finishes or streaming directly from a remote URL.

Streaming a video

To stream a video all you have to do is uncomment startMovie() at the end of playVideo and comment out code relating to file download. Then insert the remote URL path into the videoView using the setVideoPath method.

Video selection menu and watching another video

We also designed a video selection menu for the application using a ScrollView that is hidden until needed. The selection menu can be opened using the "WATCH OTHER" button at the top right of the Layout or by receiving the "Watch another movie" action after a user has submitted a feedback to the feedback platform.

6.1.1 Summary

So after a few unfortunate incidences with our customer that prevented us from following the initial goals, we have ended up with a Video test application with several feedback stages. The Video test application selects a random video, downloads it and then plays it. We can change the video using a video selection menu that we can open either by using the "WATCH OTHER" button or by receiving the "Watch another movie" action from the platform after user submission.

[3]MediaController: https://developer.android.com/reference/android/widget/MediaController.html

Chapter 7
DASH video test application

7.1 Introduction

We developed the DASH Video test application using the PCL. It is a test application that can be used in the real-world which gathers device metadata. To prove that it works in practice. You can find actions defined for the system as well as design decisions made in Appendix F

General streaming metrics

Adaptiveness: A Boolean indicating whether or not the stream is adaptive.

Average bitrate: The average bitrate since the stream started until feedback.

Buffer caught up count: The number of chunks the buffer was living up to the indicated bitrate.

Buffer falling behind count: The number chunks the buffer was falling behind the indicated bitrate.

Buffer for playback: The duration of media in ms that needs to be buffered before starting or resuming playback.

Buffer for playback after rebuffer: The duration of media in ms that needs to be buffered before starting playback after a rebuffering event.

Buffer max: The maximum duration of media in ms that the player will attempt to buffer.[1]

Buffer min: The minimum duration of media in ms that the player will attempt to ensure is buffered at all times.

[1]Default buffer: https://google.github.io/ExoPlayer/doc/reference/com/google/android/exoplayer2/DefaultLoadControl.html

Buffer percentage: The percentage of the media that has been buffered.

Buffer position: The position of the buffer

Buffering count: The number of times the stream has buffered.

Dropped frames count: The number of frames that have been dropped.

Dropped frames timespan: The duration of frames that have been dropped.

Load data: Metrics specific to each streamed chunk of data.

Logg: An application logg containing logged events from the DASH Video test application.

Player states: A list of player states and at what time they occur.

Track change count: The number of times a DASH track has changed so far into the stream.

Video name: The name of the video being streamed.

Table 7.1: The metadata collected by the Device Collection Module.

	General metadata collected
Current pos:	The current playback position of the video in milliseconds.
Video name:	The name of the video currently being played. Was meant to be used to locate the pcap file that was associated to the video. But ended up being just a general indicator of what video the user gave feedback to.
	Metadata relating to the "Download" stage of the application.
Buffer length:	The length of the buffer.
Total size:	The total size of the file in bytes.
Download time:	The amount of time that has spanned since the download was initiated until the feedback button was pushed.

Player states

The player states are recorded events with a timestamp as key to indicate when the different states occurred. A player state can either be Idle, Buffering, Seeking or Ready.

Logg

We use the logg as a proof of concept and record PAPP logg events into it.

Load data - Metrics of individual streamed chunks

When a chunk finishes it records an event with metrics that are related to that specific chunk. The chunk metrics are stored with a timestamp as key to indicate when the chunks finished. The metrics we gather from chunks are:

Actual bitrate: The actual bitrate the chunk was streamed with.

Actual media time: The current playback position of the media.

Bytes loaded: The amount of bytes that was loaded with the chunk.

Load duration: The duration of media in ms that was loaded.

Media end time: Where this chunk ends in ms at playback.

Media start time: Where this chunk starts in ms at playback.

Session time: How long the session has lasted in ms.

Track id: Indicates what track was used for the chunk.

Track bitrate: The bitrate indicated on the current track.

Track fps: The fps indicated by the current track.

Track height: The media height of the current track, in pixels.

Track mimetype: Indicates the mimetype of the current track.

Track selection reason: Indicates why the current track was chosen if information is available, otherwise unknown.

Track width: The media width of the current track, in pixels.

7.1.1 Communicating with the platform

The DASH video test application communicates to the platform using the Platform Client Library. The Platform Client Library enables us to connect to the platform by binding to it, receive the platforms Messenger with its attached Handler and to send our own Messenger with a customized Handler for incoming messages.

7.2 Development

7.2.1 ExoPlayer DEMO as basis for our application

Android already has a DEMO application they released that uses the ExoPlayer. Their ExoPlayer demo application serves two purposes as stated by their team [Exo]:

1. "To provide a relatively simple yet fully featured example of ExoPlayer usage. The demo app can be used as a convenient starting point from which to develop your own application."
2. "To make it easy to try ExoPlayer. The demo app can be used to test playback of your own content in addition to the included examples."

Therefore, to save time and to avoid doing unnecessary work we decided to use the ExoPlayer demo application as the basis for this project.

The ExoPlayer demo application comes with 5 classes:

DemoApplication: A "placeholder application to facilitate overriding Application methods for debugging and testing".

EventLogger: Logs ExoPlayer events. This includes playback events, streaming events etc...

PlayerActivity: An "activity that plays media using SimpleExoPlayer".

SampleChooserActivity: "An activity for selecting from a list of samples".

TrackSelectionHelper: A "helper class for displaying track selection dialogs".

These classes gives us the basis for our application and provides functionality for DASH media streaming. There are only a few changes we have to make in order to accommodate our needs.

The first change we make is to add the name, uri and extension of the videos on Simulas DASH video streaming server into the json media list. The media list is used by the SampleChooserActivity which displays a menu to select a media sample for playback.

7.2.2 Using the application context to store provider metadata collection state

Because the provider metadata we want to collect is event based and spread out into the different classes we want to store them in the application state space, so that

we can send all the provider specified metadata together in one HashMap using the MESSAGE_TYPE_EXTRAS flag. To save events into the application context we start by modifying the DemoApplication class a bit to accommodate this. We define fields for all the different provider metadata we want to collect, then we generate getters and setters for each of the fields. These fields can then be saved to and accessed using the application context.

```
//Inside DemoApplication
public class DemoApplication extends Application {
private int defaultBufferForPlayback;

    public void setDefaultBufferForPlayback(int defaultBufferForPlayback) {
    this.defaultBufferForPlayback = defaultBufferForPlayback;
  }

}

//Inside any Activity:
public class AnyActivity extends Activity implements ... {
...
    private DemoApplication appContext;
onCreate(Bundle savedInstanceState) {

     //Defining a variable for the application context.
appContext = ((DemoApplication)getApplicationContext());

        ...
        //Setting a variable in the application context.
appContext.setDefaultBufferForPlayback(defaultLoadControl.DEFAULT_BUFFER_FOR_PLAYB
}
}
```

To initialize a new application context state, or to reset the existing application context state we defined the initializeState method that sets all the provider metadata back to a start value. This method is also used by referencing the applciation context.

7.2.3 Collecting provider metadata events

We collect provider metadata from the PlayerAcitivy and EventLogger classes. We modified them a bit to collect provider metadata that we desire and save them in the application context.

PlayerActivity

The PlayerActivity handles the SimpleExoPlayer and its initialization. On initialization it constructs a DefaultLoadControl which is included into the SimpleExoPlayer. It contains a set of default buffer values used for streaming media, such as "Buffer for playback", "Buffer for playback after rebuffer", "Buffer max" and "Buffer min". The DefaultLoadCOntrol is initialized with the player in initializePlayer. We take these values and store them into the application context using their setters.

```
private void initializePlayer() {
  if (player == null) {
   ...
      defaultLoadControl = new DefaultLoadControl();
      trackSelectionHelper = new TrackSelectionHelper(trackSelector, videoTrackSele
      player = ExoPlayerFactory.newSimpleInstance(this, trackSelector, defaultLoadC
            drmSessionManager, preferExtensionDecoders);

  ...
      appContext.setDefaultBufferForPlayback(defaultLoadControl.DEFAULT_BUFFER_FOR_
      appContext.setDefaultBufferForPlaybackAfterRebuffer(
      defaultLoadControl.DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_MS);
      appContext.setDefaultMaxBuffer(defaultLoadControl.DEFAULT_MAX_BUFFER_MS);
      appContext.setDefaultMinBuffer(defaultLoadControl.DEFAULT_MIN_BUFFER_MS);
  ...
  }
}
```

EventLogger

The EventLogger is where the majority of the events are recorded and stored into the application context. The event logger listens for changes in the TrackSelector, the SimpleExoPlayer and when using the MediaSource interface which handles "a source of media consisting of one or more MediaPeriods/chunks"[2] when building media using DASH or other streaming methods. The EventLogger also listens to audio and video debug events as well as the Id3 container and when managing drm sessions however, even if they could prove useful we do not record these events in this experimental application.

```
private void initializePlayer() {
```

[2]MediaSource interface: https://google.github.io/ExoPlayer/doc/reference/com/google/android/exoplayer2/source/MediaSource.html

```
...
    eventLogger = new EventLogger(appContext);

    TrackSelection.Factory videoTrackSelectionFactory =
          new AdaptiveVideoTrackSelection.Factory(BANDWIDTH_METER);
    trackSelector = new DefaultTrackSelector(mainHandler, videoTrackSelectionFacto
    trackSelector.addListener(this);
    trackSelector.addListener(eventLogger);
    trackSelectionHelper = new TrackSelectionHelper(trackSelector, videoTrackSelec

    player = ExoPlayerFactory.newSimpleInstance(this, trackSelector, defaultLoadCo
          drmSessionManager, preferExtensionDecoders);

    player.addListener(eventLogger);

    eventLogger.setPlayer(player);
    eventLogger.setDefaultLoadControl(defaultLoadControl);

}

private MediaSource buildMediaSource(Uri uri, String overrideExtension) {
    int type = Util.inferContentType(!TextUtils.isEmpty(overrideExtension) ? "." +
        : uri.getLastPathSegment());
    switch (type) {
      case C.TYPE_SS:
        return new SsMediaSource(uri, buildDataSourceFactory(false),
            new DefaultSsChunkSource.Factory(mediaDataSourceFactory), mainHandler,
      case C.TYPE_DASH:
        return new DashMediaSource(uri, buildDataSourceFactory(false),
            new DefaultDashChunkSource.Factory(mediaDataSourceFactory), mainHandle
      case C.TYPE_HLS:
        return new HlsMediaSource(uri, mediaDataSourceFactory, mainHandler, eventL
      case C.TYPE_OTHER:
        return new ExtractorMediaSource(uri, mediaDataSourceFactory, new DefaultEx
            mainHandler, eventLogger);
      default: {
        throw new IllegalStateException("Unsupported type: " + type);
      }
    }
  }
```

The EventLogger implements a bunch of event listener interfaces for listening to events that we can use and store. The following are the event listener interfaces that concerns us and what type of events they listen for:

ExoPlayer.EventListener: This interface listens to changes in the players state. In our case the SimpleExoPlayer.

VideoRendererEventListener: This interface listens to video renderer events.

AdaptiveMediaSourceEventListener: This interface listens to adaptive MediaSource events.

TrackSelector.EventListener: This interface listens to track selection events.

ExoPlayer.EventListner

Using the ExoPlayer.EventListner[3] interface we use the onPlayerStateChanged method to add player state events into a HashMap in the application context. We use the current time in milliseconds (ms) as the key to indicate when the event happened and the player state as the value. A player can either be "Buffering", "Ready", "Ended" or "Idle". The time between a player state event and another player state event indicates the time the player is in a specific state. We use the onPositionDiscontinuity that fires an event when the user "seeks" or continues playback at another point of the media file using the media controller. As with the other events this event gets added to the HashMap with the current time in ms as the key and the player state, in this case "Seeking" as the value.

VideoRendererEventListener

Using the VideoRendererEventListener interface we only consider the onDroppedFrames method that gives us the number of frames dropped by the renderer while streaming. This value is incremented onto the existing value in the application context.

AdaptiveMediaSourceEventListener

Using the AdaptiveMediaSourceListener we consider methods that help us to determine the metadata concerning media chunks. We use the onLoadStarted, onLoadError and onLoadCompleted methods to handle media chunk events. The onLoadStarted initializes a timer to determine when a chunk starts to download. In case of error loading the chunk we report an "Error trying to load chunk" logg event that we save into the logg HashMap in the application context with a timestamp as the

[3] Appendix: ExoPlayer.EventListener specific methods

key. If a chunk successfully loads the onLoadCompleted will catch the event and provide us with some data we can use. From this we capture the "Load duration", "Media start time", "Media end time" and "Bytes loaded". We use the bytesLoaded and loadDuration to calculate the "Actual bitrate". We use the received Format of the chunk, which contains information about the track used when downloading the chunk. The Format of the chunk provides us with the "Track bitrate", "Track id", "Track height", "Track width", "Track mimetype", "Track fps" and "Track selection reason". We then get the "Actual media time" by referencing getCurrentPosition on the player.

——————PUT IN APPENDIX?——————

```
public void onLoadCompleted(DataSpec dataSpec, int dataType, int trackType, Format
      int trackSelectionReason, Object trackSelectionData, long mediaStartTimeMs,
      long mediaEndTimeMs, long elapsedRealtimeMs, long loadDurationMs, long bytes

    float diffInSeconds = (now-mLastBytesLoadedTime) / 1000;

    //Calculate bitrate etc...
    this.logBytesLoadedInSeconds(bytesLoaded, diffInSeconds);

    HashMap<String, Object> loadData = new HashMap<String, Object>();
    mLastBytesLoadedTime = now;

    if (trackFormat != null) {
    //Put provider specific metadata into the loadData HashMap to be sent to the p
      loadData.put("load_duration", loadDurationMs);
      loadData.put("bytes_loaded", bytesLoaded);
      ...

      //Use chunk Format to determine track information.
      loadData.put("track_bitrate", trackFormat.bitrate);
      loadData.put("track_id", trackFormat.id);
      ..
      loadData.put("actual_bitrate", getObservedBitrate());
      appContext.addLoadData(System.currentTimeMillis(), loadData);
    }
  }

  //Logs bytes loaded in seconds so we can use it to calculate the bitrate.
  private void logBytesLoadedInSeconds(long bytes, float seconds){
```

```
    mBytesLoaded += bytes;
    mBytesLoadedSeconds += seconds;
    if(mBytesLoadedSeconds > 0){
      double bytesPerSecond = mBytesLoaded / mBytesLoadedSeconds;
      double bitsPerSecond = bytesPerSecond * 8; // (8 bits in a byte)
      getObservedBitrate();
      if(bitsPerSecond < mIndicatedBitrate){
        // buffer is falling behind!
        appContext.incrementBufferFallingBehindCount();
      }else {
      //Buffer is caught up.
        appContext.incrementBufferCoughtUpCount();
        }
      }
    }

//Calculates the observer bitrate of a chunk.
    public int getObservedBitrate(){
      if(mBytesLoadedSeconds != 0){
        double bytesPerSecond = mBytesLoaded / mBytesLoadedSeconds;
        double bitsPerSecond = bytesPerSecond * 8; // (8 bits in a byte)
        return (int)bitsPerSecond;
      }
      return 0;
    }
```

TrackSelector.EventListener

We use the TrackSelector.EventListener's onTrackSelectionsChanged which handles track selection events together with the getAdaptiveSupportString which checks a track to determine what type of adaptiveness the stream is using.

7.2.4 Communicating with the platform

After collecting the provider specific metadata in the application context we need a way to send it to the back-end. To do this we included the Platform Client Library into the project and initialized the MessagingService into the PlayerActivity by building it using its constructor.

```
messagingService = new MessagingService("dashdemo", stage, appContext, new CustomIn
```

To initialize the MessagingService we need three or four parameters. The customer id and stage to set the correct state in the feedback platform. A context to bind the with the OSAPP. Optionally we can include a customized Handler for incoming messages like we have done with the CustomIncomingMessageHandler class in our case. This is all the lines of code we need to use the library to communicate with the feedback platform using sendToActivity and sendHashMapToActivity with complementary flags which we can access using the MessagingService we constructed.

CustomIncomingMessageHandler

Because we want to receive and handle actions from the feedback platform we have defined our own customized incoming message Handler. We use incoming "pressed" messages to indicate when a user is about to give a feedback so we can send our provider specific metadata to be submitted with the feedback. "pause" and "continue" messages indicate whether to pause or continue the playback of the video. "Watch another movie" indicates that the user wants to watch another movie, so he is shown the video selection screen.

7.3 Summary

We have now designed and developed on top of Androids ExoPlayer demo application an application which utilizes the Platform Client Library for communication, and lets us stream media files from Simulas DASH server. To play stream videos we use the SimpleExoPlayer player. The ExoPlayer library comes with a set of event listeners to record useful events on the player, chunks and tracks so we can save them into an application context state containing the provider specific metadata.

Chapter 8
User Testing

8.1 Introduction / Overview

The user testing was performed from 12:00 to approximately 16:30 on the 24th of November 2016 in Norwegian University of Science and Technology (NTNU) room R60 and the hallway of "Realfagsbygget".

8.1.1 Funding

We want to direct our thanks to the NTNU who supported us with financial resources to buy food to attract users for testing.

8.2 User test - Video Test Application

8.2.1 Test goals

- Check for bugs on the feedback platform and Video test application.
- Evaluate user feedback against Simula MONRONE node data. This was not possible as their DASH videos were not available in the test period.
- Evaluate users QoE of using the feedback platform.
- Evaluate users QoE of using the Video test application.

8.2.2 Preparations

User test event registration page

First we sat up a page with Eventsmart[1], a free online tool for creating and managing events. The page has a short description of the experiment, the duration, location,

[1] Eventsmart: https://eventsmart.com

time and food to be served. Users could use the site to easily register to the event. The event registration page can be accessed here: https://everyback.eventsmart.com/events/android-app-user-testing-spring-rolls-soda/

Finding users

I requested the student coordinator of our faculty Laurent Pacquerau (Check) to send an e-mail to all the students in a mailing list with an attached event sign up page. However, only about 6 signed up for the event and only one student actually showed up. It was a huge letdown, so we decided to try to recruit users around the university. First we tried to contact people that were sitting in groups, but this was almost impossible as everyone was busy reading for their exams. Then we tried positioning ourselves in the university hallway. There we placed the spring rolls and cups of soda for interested user testers. After about three hours we were able to get about 11 more users to do the user test.

8.2.3 Other preparations

We booked a room for the event at NTNU and ordered 100 spring rolls in advance for the day of the user testing hoping to have about 20 users at the event. We were very excited to start user testing as we were walking up to the university with 100 spring rolls and a 6-pack of Coca Cola, so one can imagine the letdown and humor of greeting only one person in the room.

8.2.4 User context

- 1 user in the room designated for user testing.
- 11 users using a stand in the hallway of the university. Only 9 of them filled out the questionnaire that we sent them.

8.2.5 How testing was done

First the participants installed the feedback platform application onto their phone. Afterwards they were asked to install the video test application. They were then instructed to watch videos and give feedback whenever they felt like it, on anything they felt like giving feedback on. They were instructed to do this until they wanted to stop. Afterwards they were presented with a questionnaire. A lot of the users ended up using our phone as users that did the test in the hallway were more busy and did not have time to install the applications on their phones. We would then send the questionnaire to their facebook or email address.

8.2.6 Questionnaire

The questionnaire was made using a google drive schema that can be viewed here https://docs.google.com/forms/d/e/1FAIpQLSdGJn1656kI7eUogpaHf60rU1hgMqJW-WIvnyGmYTP04jy viewform. All the answers were anonymous and the users agreed to let us use this anonymous data for the thesis when they pushed submit form as according to the disclaimer in the introduction of the questionnaire.

Privacy disclaimer

"This information is kept anonymous. And will be used in research. By submitting this form you agree that this information can be used for research purposes."

Questions

- Segmentation questions:
  - What is your age?
  - What is your gender?
  - What is your nationality?
- Questions about Video test application (1 very bad - 5 very good):
  - How was the video you just watched?
  - How do you feel about the quality of the video application?
  - How was the video application visually?
  - How was the video application functionally?
  - Please write down what you did not like about the application:
- Questions about Overlay Service Application (1 very bad - 5 very good):
  - How was the feedback application experience?
  - How do you feel about the quality of the feedback application?
  - How was the feedback application visually?
  - How was the feedback application functionally?
  - Was the hovering button annoying? (1 very annoying - 5 Not annoying)
  - Please write down what you did not like about the application:
  - Did the feedback application make sense to you?
  - In your own words, explain when and why to use the application:
  - Do you have any suggestions for how the application could be improved?
- General question:
  - Do you have any final feedback?

8.2.7 Results

User segment

We ended mainly up with 24 year old Norwegian males.

Video test application

We got varied feedback on the Video test application and discovered a few bugs that needed to be fixed. This is the issue with android and its variety of devices, something that works on one device does not necessarily work without problems on another. The Video test application scored quite averagely in the test, and most users seemed satisfied enough. However, we think that the 10 out of 12 users who did the user test and submitted the form was not enough to give any clear results.

Overlay Service Application

This part was a bit more interesting as we can compare the results to the test of the original platform [2]. Generally the students were satisfied with the functionality and visual appeal of the OSAPP. Everyone indicated that they understood what the OSAPP was meant to do however, one student notes that "It's not obvious when you want to give feedback". One user wrote that it would be nice if "the application was integrated with netflix to give him better suggestions". We observed that the concept worked and user feedback was being submitted to the back-end.

8.2.8 Considerations

Changing of experimental context

It has to be noted that because we were recruiting most of the users did not test for longer than 5 minutes. This is because we had to recruit most users from a stand in the university hallway. These users were more in a hurry than the user who met for the event. This probably caused extra uncertainty in the feedback received through the questionnaire.

Concern about the reliability of the test

Because of the different circumstances the users were recruited from and the lack of users recruited in total we do not consider this test very reliable however, got some basic insight into the feedback trend and we found some bugs to fix.

[2] [År16]

8.3 User test - DASH Video Test Application

8.3.1 Introduction

This user test was performed the 13. December 2016 from 18:00-22:00 in the "Jesu Kristi Kirke av Siste Dagers Hellige" church building at Eirik Jarls Gate 5.

8.3.2 Test goals

- Check for bugs on the feedback platform and DASH video test application.
- Evaluate provider specific metadata to look for interesting patterns.
- Evaluate what users think about sharing information with companies from their phones.
- Evaluate users QoE of using the feedback platform.
- Evaluate users QoE of using the DASH video test application.

8.3.3 Preparations

Finding users

After learning from the past faults, we decided to try another way of finding users. This time we contacted organizations and asked if they wanted to be a part of the user testing in exchange for a Christmas meal. Eventually the young adults group of "The Church of Jesus Christ of Latter-Day Saints" in Trondheim decided that they wanted to help. They had 21 people that wanted to participate in the event however, 19 showed up. We were a lot more satisfied with this group as most of them showed up however, it took a lot of preparation making the food.

The last 9 users were recruited among friends where 4 were from Italy, one from Peru and 4 from Norway. They did the test on their own phones and submitted the questionnaire afterwards.

Preparing the Christmas food

We started preparing Norwegian Christmas specials such as "ribbe" and "pinnekjøtt" starting at about 15:00. It was quite the preparation, but worth it as we were able to have a lot more users this time around.

8.3.4 User context

- 19 users in designated room for user testing.

– 9 users by sending them the .apk files and questionnaire to be filled afterwards.

8.3.5 How testing was done

As the users started eating we borrowed them one by one to do the user test. We tested most of the users on our mobile phone, so that we did not have to go through the hassle of installing it on every device. Only the online users tested on their own devices.

All user testers were first given a short introduction of the platform, what data is being collected, the DASH video test application and the purpose of the test. Then asked to test the application and give feedback if they desired to do so. Each user tested the platform between 10-15 minutes. After testing they were asked to fill out the questionnaire as we introduced the next user to the test.

8.3.6 Questionnaire

Some improvements were made to the questionnaire resulting in the following questionnaire:

- Segment questions:
  - What is your age?
  - What is your gender?
  - What is your nationality?
- General feedback questions:
  - How was the video you just watched (1 very bad - 5 very good)?
  - Did you give any feedback or use the feedback application during your session? Why or why did you not give feedback?
  - Do you have any final feedback?
- DASH Video test application questions (1 very bad - 5 very good):
  - How do you feel about the quality of the DASH Video test application?
  - How was the DASH Video test application visually?
  - How was the DASH Video test application functionally?
  - Please write down what you did not like about the DASH video test application:
- Overlay Service Application questions - Evryback (1 very bad - 5 very good):

- How was your experience using the OSAPP?
- How do you feel about the quality of the OSAPP?
- How was the OSAPP visually?
- How was the OSAPP functionally?
- Was the hovering button annoying? (1 very annoying - 5 Not annoying)
- If you thought it was annoying, explain why:
- Please write down what you did not like about the OSAPP:
- Did the OSAPP make sense to you?
- In your own words, explain when and why to use the OSAPP:
- Do you have any suggestions for how the OSAPP could be improved?
- How acceptable would the amount of data shared be to you in an everyday setting?
- How do you feel about sharing your data with companies and why?

8.3.7 Results

User segment

We had a very diverse group of users this time of all age groups and quite evenly spread out between the genders with a slight majority of males. The users came from 9 nationalities including 8 from Norway, 6 from the United States, 4 from Italy, 3 from Peru, 2 from Finland, 1 British, 1 Chilean, 1 from the Netherlands and 1 from Syria.

DASH Video test application

Most users were satisfied with the DASH video test application visually and functionally. However, a lot of users wrote that they did not like the video selection menu structure. This is understandable as it was not particularly appealing. Otherwise most users seemed generally satisfied with the application except for a few personal preferences and some bugs that we discovered.

Overlay Service Application

Most users were satisfied with the OSAPP with over 80% of them rating all questions 4 or better. Only 7% of the users found the hovering feedback button to be quite annoying. One of them wrote "Well if you are aware of it, then it can be a bit annoying", the test and the user statements seems to validate the findings found in the original platform test[3] where the results showed that most people were satisfied

[3] [År16]

with the platform and forgot about the hovering feedback button when they did not need it. The situations that made the button annoying were usually related to buttons hidden under the hovering feedback button, or problems with the button blending with the background which can be fixed.

One user noted "If I'm experiencing a problem and I know the feedback will come through and potentially fix the problem, I can see myself using it. If the positive feedback actually matters in some way, as in developers or other users being able to see it, I might use that as well". indicating that he would use the platform if it would make a positive difference.

Acceptance of data collection on mobile phone

Over 80% of the users were neutral to the question or thought that it was acceptable to share the data that they were told was shared with the feedback platform on an everyday basis. One user wrote that "Fine as long as data is used to improve user services" and another one "Don't mind it as long as it's not used for advertising", which seems to be the general feeling among the users. There were of course also some skeptics that mentioned privacy and that it depends on the data shared. We were a bit surprised to see that such a big majority of the users were either neutral or accepting of the data collection, as we see ourselves as a bit more skeptical.

Evaluating the metadata gathered

Because the test was performed in an indoor environment with very good internet speeds there was not a lot of useful metadata patterns that we could discover. There was only one significant discovery and that was people being annoyed with the amount of time it took for some videos to start, or the time from the initial playerState "Buffering" to "Ready".

8.3.8 Considerations

Acquaintances

The 9 users who did the user test online were acquaintances, and could therefore be a bit biased in their answers.

Chapter 9
Results and Discussions

9.1 Changes to some of the initial goals

9.1.1 No time for website

As the semester was approaching its end and we had to start writing on the thesis we realized that there was simply no time to finish the development of the changes on the website, as the other more essential changes had takes more time than expected. However, all the platform functionality is there to enable provider customization. As the situation is right now providers can use REST calls as we did using Postman to update the back-end with their specific customization requirements. This includes what collection modules to have enabled when a PAPP is connected to the OSAPP as well as provider customized feedback paths. However, there is no provider friendly way to visualize the data at the moment.

9.1.2 Video player and debug logging as modules in Platform Client Library vs. bare minimum

We were initially going to make a video player module and a logging module inside the Platform Client Library initially, but decided not to do so. This is because we do not know what the providers will use the platform for, therefore it will be bloatware to providers who do not need it. We decided to rather stick with the bare minimum to keep the library lean and lightweight, keeping only the essential parts needed for communication. Providers are however free to send any provider specific data they can imagine as a HashMap with the flag MESSAGE_TYPE_EXTRAS, therefore making the platform very flexible to all types of providers and provider specific metadata. This includes and is not limited to video performance data and provider submitted debug logs.

9.2 What metrics to track?

One of the purposes of testing was to figure out what metrics are important and should be tracked. Because of the circumstances of the tests there was not a lot of clear winners. If we had tested in circumstances with poorer internet speeds or in a buss or other mobile environments switching between networks it would probably have been different. However, we did see a connection between annoyance with a video being slow to start and the time between the initial "Buffering" and "Ready" state. So we can at least say that the playback state and when it changes are important metrics.

9.2.1 Overfitting

One issue with platforms like this that gathers context metadata is to not overfitt, particularly when providers can submit any data they would like. To avoid overfitting we want to try to keep measurements simple in complex systems. However, we do believe the benefits of letting providers provide any metadata they would like overshadows these concerns. Though we do suggest that providers choose their metrics carefully, so that they may receive meaningful results.

9.3 The potential provider specified feedback paths

This is the set of stages or states that a provider application goes through that has its own set of feedback alternatives. If i.e. the user is watching a video using the provider application the provider might want to give feedback alternatives specific to watching a video. Like the video is buffering too much, the buffering is too long or the video quality is poor. When we let providers decide the alternatives we let them answer the questions they are wondering about, rather than answering questions that might not even be important to the provider. It gives providers a lot of flexibility at determining the direction of their own QoE studies.

9.4 Platform

9.4.1 Does the design actually work?

Our user tests seem to indicate that the design works. There were only a few users that were annoyed by the hovering feedback button however, we have also made a solution to solve this issue. We did not use our solution in the test applications however, users have the possibility to hide the overlay feedback button. When that happens a message is sent to the PAPPs telling them that the button is removed. They can then implement a "feedback button" inside their own GUI that can send a

MESSAGE_TYPE_PROMPT flag to the platform when it is tapped. This prompts the feedback platform to display the feedback selection menu.

9.5 User testing

9.5.1 How to recruit users for testing?

It is always hard to recruit users to testing, we learned that this is particularly hard during the weeks students are reading for their exams. However, we seem to have a much greater success rate recruiting whole organizations rather than individuals. It took a lot less time to get the necessary amount of test users, and we only needed to convert one organization to have success. It was also effective to offer food for the organization versus to individuals. Individuals seemed too busy, individuals in an organizations were gathering together anyway and needed a pause for food. It took quite a while to arrange, make the food and serve it, but it was worth it in regards to users converted per hour spent working on them.

9.5.2 Crowdsourcing user tests

We were unfortunately not able to test user crowdsourcing because of limited resources. However, we think that this could be a useful tool for future tests. Of course there will be some uncertainty in the results however, we can use statistical analysis to sort out the weeds as suggested by [THTG14].

Chapter 10
Conclusion and further work

10.1 Conclusion

There are a lot of platforms and tools out there. Many of them collect more metadata than our platform and has a proven track record. However, there is a lack of QoE measurement tools that gives providers the flexibility to customize the platform to their needs and send their own provider specific context metadata. So we proposed a solution that lets providers do this, without effecting the performance of the system significantly or annoying the user.

To make this possible we have made it possible for providers to make their own feedback paths, with provider defined feedback alternatives and actions. Then we provided a way for providers to gather and send their own metadata together with user submitted feedback, using an android library we call the Platform Client Library. This enables providers to gather metadata from their own applications according to what they need, rather than according to what the platform restricts them to do. We have also made it possible for providers to enable and disable what types of metadata the platform should actively gather in the background in order to have more control of the resource consumption of the platform and weed out metadata that the provider deems unnecessary. These changes offer a lot of flexibility to the providers.

So we suggest this platform to anyone who needs a way to collect user feedback in a non-intrusive way together with user metadata measured by the paltform and by PAPPs. Some metadata that can only be collected by the PAPP itself and they need a way to provide this together with user feedback, which is what we offer for providers who want to use the platform. This platform is for providers who want flexibility in how to measure their data. Also it provides the possibility to test in the real-world, among real users, using real applications, and does not require a laboratory.

10.2 Future work

10.2.1 Finish the website

We think that finishing the website is the most important step missing of this platform. The website is crucial to integrate providers properly into the platform and to let them customize it in an easy way.

10.2.2 More extensive user tests

The platform needs better and more extensive user testing to determine whether the platform itself does not annoy the user. Also it needs more testing in real world provider cases, where providers want to know something specific about the user and we can prove the potential of the platform.

10.2.3 Edge screens

Now that edge screens are becoming more popular a possible alternative discrete solution to the feedback overlay button that should be tested is the possibility to drag a feedback button from into visibility using dragging gestures on the edge.

10.2.4 Big data and QoE prediction

"The platform has the possibility to gather huge amounts of data about users and their context. The goal of the application that gathers the QoE data is to collect enough data for the platform to reach its own demise, to have complete user satisfaction and therefore making the app itself obsolete. With enough data gathered with the platform we can better understand and map users annoyances to the users context. When we are able to understand what annoys the user and what causes their annoyance, we are able to predict and prevent the possible annoyances before they happen. This is very useful and can change the game when it comes to user satisfaction and retention of users."[1]

[1] [År16]

References

[AM14] Sana Aroussi and Abdelhamid Mellouk. Survey on machine learning-based qoe-qos correlation models. *Computing, Management and Telecommunications (ComManTel), 2014 International Conference*, pages 200–204, April 2014.

[Anda] Android. Asynctask. Reference website: https://developer.android.com/reference/android/os/AsyncTask.html.

[Andb] Android. Bound services. Reference website: https://developer.android.com/guide/components/bound-services.html.

[Andc] Android. Executors. Reference website: https://developer.android.com/reference/java/util/concurrent/Executors.html.

[Andd] Android. Handler. Reference website: https://developer.android.com/reference/android/os/Handler.html.

[Ande] Android. Ibinder. Reference website: https://developer.android.com/reference/android/os/IBinder.html.

[Andf] Android. Messenger. Reference website: https://developer.android.com/reference/android/os/Messenger.html.

[Andg] Android. Service - developer guides: Topic 6 remote messenger service samples. Reference website: https://developer.android.com/reference/android/app/Service.html.

[Andh] Android. Videoview. Reference website: https://developer.android.com/reference/android/widget/VideoView.html.

[ANM] S. Xu D. Choffnes A. Nikravesh, H. Yao and Z. M. Mao. Mobilyzer: An open platform for controllable mobile network measurements. *Proceedings of the 13th International Conference on Mobile Systems, Applications and Services (MobiSys), Florence, Italy, 2015.*

[AW13] Mohammed Alreshoodi and John Woods. Survey on qoe and qos correlation models for multimedia services. *International Journal of Distributed and Parallel Systems (IJDPS)*, 4, May 2013.

[BH10] Peter Brooks and Bjørn Hestnes. User measures of quality of experience: Why being objective and quantitative is important. *IEEE Network*, 24:8–13, Match-April 2010.

[BJVI13] Poul E. Heegaard Bjørn J. Villa, Katrien De Moor and Anders Instefjord. Investigating quality of experience in the context of adaptive video streaming findings from an experimental user study. *Conference: Norsk Informatikkonferanse 2013 (NIK 2013), At Stavanger, Norway*, pages 122–133, 2013.

[Eri16] Ericsson. Ericsson mobility report. November 2016.

[Exo] ExoPlayer. Open source media framework for android. Reference website: http://google.github.io/ExoPlayer/.

[FDZ11] Asad Awan Ion Stoica Dilip Joseph-Aditya Ganjam Jibin Zhan Florin Dobrian, Vyas Sekar and Hui Zhang. Understanding the impact of video quality on user engagement. *ACM SIGCOMM Computer Communication Review - SIGCOMM '11*, 41:362–373, August 2011.

[Fira] Firebase. Read and write data on android. Reference website: https://firebase.google.com/docs/database/android/read-and-write.

[Firb] Firebase. valueeventlistener. Reference website: https://firebase.google.com/docs/reference/android/com/google/firebase/database/ValueEventListener.

[Firc] Google Firebase. Firebase: Tools for building and developing mobile applications. Platform website: https://firebase.google.com/.

[FWS15] Pedro Casas Ralf Irmer Phuoc Tran-Gia Florian Wamser, Michael Seufert and Raimund Schatz. Yomoapp: A tool for analyzing qoe of youtube http adaptive streaming in mobile networks. *Networks and Communications (EuCNC), 2015 European Conference*, pages 239 – 243, July 2015.

[HL13] Jie Hui and Kevin Lau. T-mobile qoe lab: making mobile browsing faster and open research problems. *MobiCom '13 Proceedings of the 19th annual international conference on Mobile computing and networking*, pages 239–242, September 2013.

[KKP10] Nearchos Paspallis Konstantinos Kakousis and George A. Papadopoulos. A survey of software adaptation in mobile and ubiquitous computing. *Enterprise Information Systems*, 4:355–389, November 2010.

[Lap07] Philip A. Laplante. Basic software engineering principles. In *What every engineer should know about software engineering*, pages 85–88. CRC Press - Taylor and Francis Group, 2007.

[LM13] Yishan Li and Sathiamoorhy Manoharan. A performance comparison of sql and nosql databases. *Communications, Computers and Signal Processing (PACRIM), IEEE Pacific Rim*, pages 15–19, 2013.

[MFTG10] Tobias Hossfeld Markus Fiedler and Phuoc Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24:36–41, March-April 2010.

[mob] Mobiperf: An open source application for measuring network performance on mobile platforms.

[MSS] Pedro Casas Ralf Irmer Phuoc Tran-Gia Michael Seufert, Florian Wamser and Raimund Schatz. Youtube qoe on mobile devices: Subjective analysis of classical vs. adaptive video streaming. *Wireless Communications and Mobile Computing Conference (IWCMC), 2015 International*.

[MSTG16] Florian Wamser Nikolas Wehner Raimund Schatz Michael Seufert, Pedro Casas and Phuoc Tran-Gia. Application-layer monitoring of qoe parameters for mobile youtube video streaming in the field. *Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pages 411–416, 2016.

[OAL] Rafael Garcia Miguel Peón-Quirós Vincenzo Mancuso Thomas Hirsch Tobias Dely Jonas Werme Kristian Evensen Audun Hansen Stefan Alfredsson Jonas Karlsson Anna Brunstrom Ali Safari Khatouni Marco Mellia Marco Ajmone Marsan Roberto Monno Ozgu Alay, Andra Lutu and Hakon Lonsethagen. Measuring and assessing mobile broadband networks with monroe. *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium*, pages 1–3.

[PVM08] Blazej Lewcio Frank Steuer-Marcel Wältermann Pablo Vidales, Niklas Kirschnick and Sebastian Möller. Mobisense testbed: Merging user perception and network performance. *Tridentcom*, pages 18–20, March 2008.

[QACL14] Sanae Rosen Z. Morley Mao Karthik Iyer Jie Hui Kranthi Sontineni Qi Alfred Chen, Haokun Luo and Kevin Lau. Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis. *IMC '14 Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 151–164, 2014.

[Qua13] Qualinet. Qualinet white paper on definitions of quality of experience version 1.2. March 2013.

[Res15] Dimensional Research. Mobile app usage and abandonment survey. January 2015.

[Sam] Samsung. Effective communication between service and activity. Samsung Developers Online Technical Document can be found here: http://developer.samsung.com/technical-doc/view.do?v=T000000087.

[THTG14] Matthias Hirth Bruno Gardlo Julian Habigt Klaus Diepold Tobias Hossfeld, Christian Keimel and Phuoc Tran-Gia. Best practices for qoe crowdtesting: Qoe assessment with crowdsourcing. *IEEE Transactions on Multimedia*, 16:541 – 558, February 2014.

[År16] Stian Michael Årsnes. A platform for gathering user experience. 2016.

Appendix

Appendix

A.1 Where to get apks

The apks can be retrieved here: https://drive.google.com/drive/folders/0Bw0bsmXewlEySzRwb2h4d

A.2 Where to get source code

The source code can be retrieved here: https://drive.google.com/drive/folders/0Bw0bsmXewlEyVWJ

A.3 Where to find the website

The website can be displayed here: http://everyback.firebaseapp.com/

Appendix B
OSAPP Design

B.1 The OSAPP in use - How it looks and works

From Figure B.1 we can see a feedback process defined by one specific feedback path. It starts by pushing the blue button in step 1. Then the user can give his emotional rating at step 2. Then the user is introduced to step 3, a list of feedback alternatives. If the user then pushes "Bad quality" he is shown the feedback selections in step 4. Ultimately the user can in step 5 either choose to "continue", "exit" the application or "watch another movie" which is a provider defined action we have made for this use case.

B.2 Requirements specification

This is the requirements specification for the agile development process of the OSAPP.

B.2.1 Development goals

The developer goals helps us to stay on track and reach our goal during the development process.

- Make a flexible platform solution where it is possible for providers to customize the platform to their needs.
- The ability for providers to easily implement the platform into their own applications and services.
- The ability for providers to provide their own collected context data to the platform together with a feedback when a feedback has been submitted.

B.2.2 Functional requirements

The functional requirements is a set of required functionality we need to develop to reach our development goals.

- Providers can submit their own QoE feedback alternatives/paths to be used by the platform when users use the providers application.
- Providers can submit their own context data together with the feedback.
- Providers can choose what measurement modules they would like to use.
- Providers can submit debug code to the platform.
- The platform should gather context data within a time period.

B.2.3 Non-functional requirements

The non-functional requirements gives a guideline for how functionality should be developed.

- More modularity for future development
- Easy to use
- Not annoying
- Reliability
- Security

B.2.4 Use cases

These are a set of use cases used for communicating platform functionality and understanding user needs.

Use case - Youtube on the bus:

Frank is taking the bus to work. While in the bus the user wants to watch a youtube video using his phone. He starts watching, as he watches he notices that the quality has become poor. He doesn't know why the quality became so poor, but it is starting to annoy him. He remembers the feedback button and pushes it. He then pushes the angry face because of his annoyance. He is then introduced to a couple of alternatives specific to watching a youtube video. He chooses poor video quality and submits it. After submission the youtube application increases his video quality. He notices that the video has to buffer longer, but he is okay with it and would rather have better quality than less buffering.

Use case - Surfing on an airplane:

Cate is taking an airplane to japan, it is a 14 hour plane ride. Luckily she has got internet and her phone on the airplane! She is not a big fan of the overlay button of the feedback platform, so she removes it. She starts surfing the web using the airline's own in-app browser. Initially it was all well and dandy, but then things took a turn. The sites just would not load. She tried again and again, nothing... Eventually she pushed the menu button in the airline's application, and she sees an alternative in the menu called "Give feedback". She pushes it, it opens the feedback platform mood selection menu. She chooses the angry mood and is then displayed with a set of feedback alternatives that are specific to her using the in-app browser. She chooses the "Pages won't load" alternative and submits her feedback. She is tired and a bit frustrated, so she decides to sleep the rest of the trip.

B.3 Feedback module

B.3.1 Overlay feedback button

Even though the application is a service we have called it an overlay application, because it has an user interface that hovers on top of all applications. Services are not really supposed to have an user interface, so some workarounds were made using the [SYSTEM_ALERT_WINDOW] permission that allows us to create a window on top of all windows, and the window manager to make, handle and display that window. This lets us display a button that is always hovering on top of all applications, inspired by Facebook's chatheads. This button can be pushed whenever the user wants to give feedback.

B.3.2 Feedback paths

B.4 Provider stages

This is the set of stages or states that a provider application goes through that has its own set of feedback alternatives. If i.e. the user is watching a video using the provider application the provider might want to give feedback alternatives specific to watching a video. Like the video is buffering too much, the buffering is too long or the video quality is poor.

Provider paths

A provider path has sets of stages, levels and feedback alternatives that may point to specific actions to be performed. They are specific to the current provider application stage or state.

Question stage level

A question stage level has a set of defined feedback alternatives. When a feedback alternative is chosen it can either refer to a new level within the stage or to an action.

Buttons

Providers can choose to change the name of existing buttons, like the exit or continue button.

Action button

At the end of a feedback path before submission the provider might want to provide an alternative option to the basic continue or exit buttons. We call these alternatives actions.

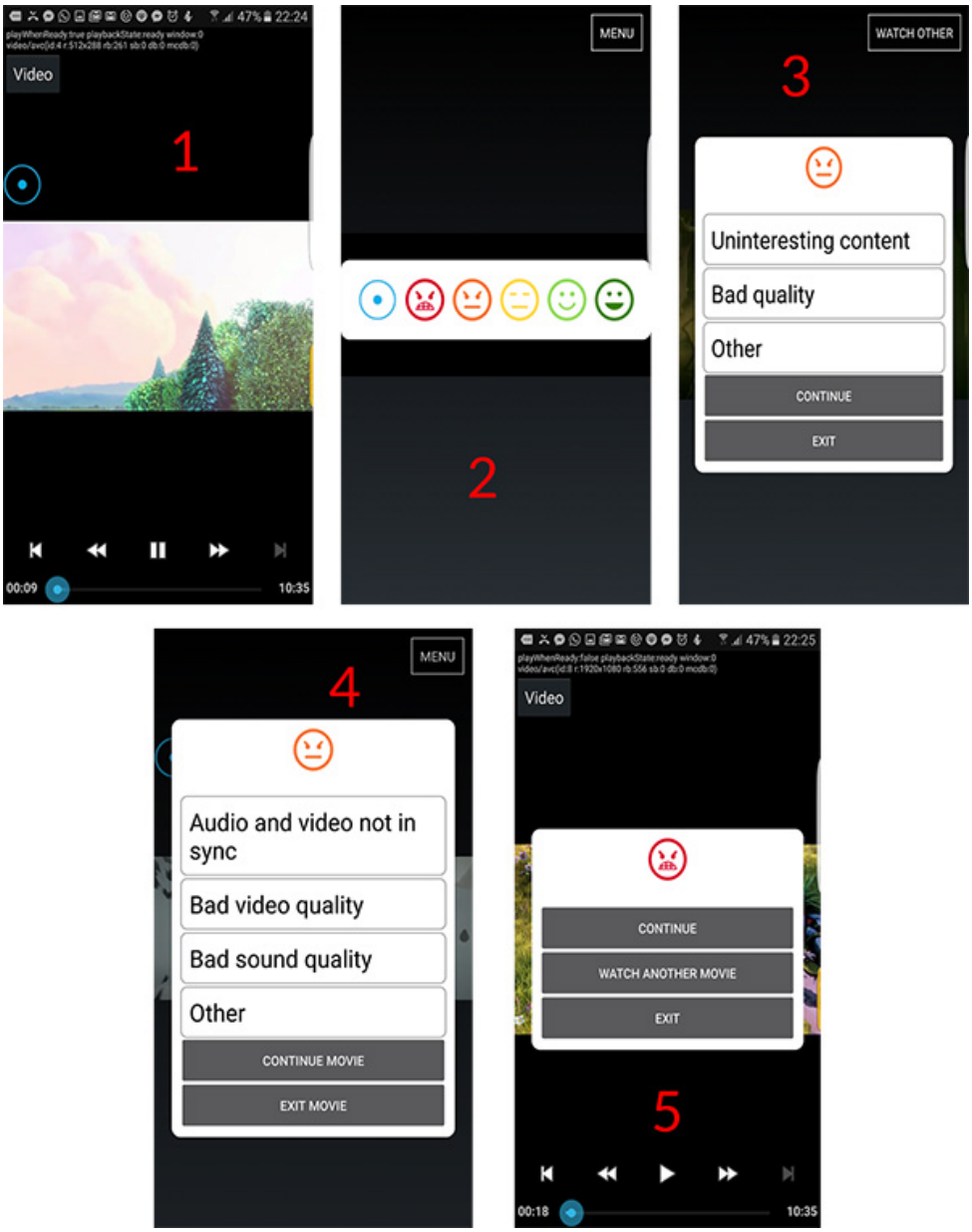

Figure B.1: An activity diagram that shows the feedback process.

Appendix C
OSAPP Development

C.1 Development environment

C.1.1 Tools used

Android studio: An Integrated Development Environment (IDE) for developing Android applications.

Firebase Client Library: A client library provided by Firebase that contains a lot of features that simplifies the process of using Firebase.

C.2 Separation of Concerns

We have suggested the following separation of code within our project:

C.2.1 Classes and packages

Previously the platform was written in only a few classes and with a few modules. We separated the code into different classes and packages as you can see in...

collectors: This is a package that contains all the classes that contains functionality for collecting data from the phone. To separate this even further we separate each collection group into their own separate classes. Network collection data into its own class, location collection data into another class and application information collection data into its own class.

helper: This package contains all helper classes. The helper classes contain general functionality to support the other classes.

interfaces: This package contains all interfaces.

managers: This package contains all manager classes. A manager class is a class that handles a specific set of operations. We use two manager classes, one to organize and send feedback called FeedbackManager and one to collect data at an interval and keep a record of collected data called TimeCollectionManager.

models: This package contains all the data models that contain data to be sent to the back-end database. Here we have a model for each of the set of data collected. One for application data, location data, network data and user data in the end. User data only needs to be collected once, so it does not need a manager to handle collections.

C.2.2 Reorganizing the existing code

First we start the SoC process by making the defined packages into the project. These are as previously mentioned, collectors, helpers, interfaces, managers and models. Then we separated the existing platform code into separate modules. We started by defining different collector classes:

AppInfoCollector which contains methods for gathering data about running applications.

LocationCollector which contains methods for gathering location data.

NetworkCollector which contains methods for gathering network data.

Then we defined a set of model classes:

AppData a model for data relating to running applications.

LocationData a model for all location related data.

NetworkData a model for all network related data.

C.3 The Feedback Module

C.3.1 Initializing the Service Overlay Application

In the onCreate of the OSAPP we want to set the initial state of the platform. We first initialize firebase and define necessary references we are going to use to retrieve data from the NOSQL database. The functionality for handling communication with the back-end is provided by the Firebase Client Library. We have to reference the "provider_paths", "actions", "configuration" and "buttons" locations in the database so we can retrieve data from them. We then initialize a default customerID, stage

and set the level to the initial "level0" if none has been set yet by a PAPP connected to the OSAPP. Whenever a PAPP disconnects from the OSAPP we also return to this default state. We also start the TimeCollectionManager so that it will collect data in the background every 30 seconds.

```
...
//Firebase initialization
FirebaseApp.initializeApp(this);
database = FirebaseDatabase.getInstance();
pathsGroupRef = database.getReference("provider_paths");
actionsGroupRef = database.getReference("actions");
buttonGroupRef = database.getReference("buttons");
configurationGroupRef = database.getReference("configurations");

//Setting necessary states to determine feedback path.
if (customerID == null) {
  customerID = "default";
}
if (stage == null) {
  stage = "default";
}
if (level == null) {
  level = "level0";
}

//Starting the TimeCollectionManager.
timeCollectionManager = new TimeCollectionManager(context);
...
```

C.4 The Communication Module

C.4.1 IncomingHandler

This is the OSAPPs handler for incoming messages.

```
//Defining the communication flags.
  public static final int MESSAGE_TYPE_REGISTER = 1;
    public static final int MESSAGE_TYPE_TEXT = 3;
    public static final int MESSAGE_TYPE_UNREGISTER = 2;
    public static final int MESSAGE_TYPE_CUSTOMERID = 4;
    public static final int MESSAGE_TYPE_STAGE = 5;
```

```
public static final int MESSAGE_TYPE_VIDEOPOSITION = 6;
public static final int MESSAGE_TYPE_VIDEONAME = 7;
public static final int MESSAGE_TYPE_PROMPT = 8;
public static final int MESSAGE_TYPE_EXTRAS = 9;
static Messenger responseMessenger = null;

class IncomingHandler extends Handler {

    @Override
    //Handle incoming messages.
    public void handleMessage(Message msg) {
        Bundle b;
        String message;

        //Filter message according to communication flag.
        switch (msg.what) {
            case MESSAGE_TYPE_TEXT:
                b = msg.getData();
                message = b.getCharSequence("data").toString();
                break;

            case MESSAGE_TYPE_REGISTER:
                serviceClients.add(msg.replyTo);
                responseMessenger = msg.replyTo;
                if (feedbackOverlayIsRemoved) {
                    sendToActivity("removed");
                }
                break;

            case MESSAGE_TYPE_UNREGISTER:
                serviceClients.remove(msg.replyTo);
                responseMessenger = msg.replyTo;
                customerDisconnected();
                break;

            case MESSAGE_TYPE_CUSTOMERID:
                b = msg.getData();
                message = b.getCharSequence("data").toString();
                customerID = message;
                customerConnected(customerID);
                break;
```

```
            case MESSAGE_TYPE_STAGE:
                b = msg.getData();
                message = b.getCharSequence("data").toString();
                stage = message;
                Log.d("STAGE", stage);
                level = "level0";
                customerRef = questionGroupRef.child(customerID);
                stageRef = customerRef.child(stage);
                levelRef = stageRef.child(level);
                break;

            case MESSAGE_TYPE_VIDEOPOSITION:
                b = msg.getData();
                message = b.getCharSequence("data").toString();
                userData.setVideoPosition(message);
                break;

            case MESSAGE_TYPE_VIDEONAME:
                b = msg.getData();
                message = b.getCharSequence("data").toString();
                userData.setVideoName(message);
                break;

            case MESSAGE_TYPE_PROMPT:
                showRatingDialog(params.x, params.y);
                break;

            case MESSAGE_TYPE_EXTRAS:
                b = msg.getData();
                userData.setExtras((HashMap) b.getSerializable("map"));
                break;

            default:
                super.handleMessage(msg);
        }

    }
}
```

———————DESCRIBE MORE———————-

C.4.2 More on communication flags

All communication between the overlay application and the provider application is flagged. The flag indicates what type of message is being sent so it can easily be handled by the handler on arrival.

MESSAGE_TYPE_REGISTER

"Connects" the provider application with the platform by registering its handler to the platform. This is the initialization message between the provider application and the platform. It lets the platform register the provider applications messenger and handle what happens after the registration process. (Such as "removed" or no msg)

MESSAGE_TYPE_UNREGISTER

Disconnects a provider application from the platform.

MESSAGE_TYPE_TEXT

Used to transfer actions and notifications between the OSAPP and the PAPPs.

MESSAGE_TYPE_CUSTOMERID

Indicates that the Message contains the providers "customer id".

MESSAGE_TYPE_STAGE

Used to define what stage or "scenario" the application is currently in.

MESSAGE_TYPE_PROMPT

Used to prompt users for feedback in cases where the provider thinks this is useful. This kind of goes against the concept of letting users give feedback on their own demand. However, this gives providers more flexibility if needed.

MESSAGE_TYPE_EXTRAS

Indicates that the Message contains provider-specific metadata.

C.4.3 Binding the PAPP to the OSAPP

The services onBind method returns a communication channel to the service. This is retrieved by the PAPPs using bindService which initiates the connection between the OSAPP and PAPP. The onBind

onBind

```
@Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return messenger.getBinder();
    }
```

bindService

```
intent = new Intent();
        intent.setAction("no.ntnu.stian.evryback.FeedbackOverlayService.ACTION_BIN
        intent.setPackage("no.ntnu.stian.evryback");
        context.bindService(intent, connectionBinder, context.BIND_AUTO_CREATE);
```

C.4.4 sendToActivity

```
static void sendToActivity(CharSequence text) {

        if (responseMessenger == null) {
        //No response messenger has been set
        } else {
            Bundle data = new Bundle();
            data.putCharSequence("data", text);

            Message msg = Message.obtain(null, MESSAGE_TYPE_TEXT);
            msg.setData(data);

            for (int i=serviceClients.size()-1; i>=0; i--) {
                try {
                    serviceClients.get(i).send(msg);
                } catch (RemoteException e) {
                    // The client is dead.  Remove it from the list;
                    // we are going through the list from back to front
                    // so this is safe to do inside the loop.
                    serviceClients.remove(i);
                    e.printStackTrace();
                }
            }
        }
    }
```

C.5 The Back-end Module

C.5.1 getInfoFromDatabase()

```
private static void getInfoFromDatabase() {
        levelListener = new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                alternatives = new HashMap<String, String>();
                for (DataSnapshot alternativesSnapshot: dataSnapshot.getChildren())
                    alternatives.put(alternativesSnapshot.getKey(), alternativesSna
                }

                List<String> alternativesStringArray = alternativesHandler.getStrin

                if (alternativesStringArray.contains("Other")) {
                    alternativesStringArray.remove("Other");
                    alternativesStringArray.add(alternativesStringArray.size(), "Ot
                }

                listView.setAdapter(new ArrayAdapter<String>(getInstance(),
                R.layout.feedback_list_item, R.id.itemName, alternativesStringArray
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                //Some firebase error
            }
        };

        customerRef = pathsGroupRef.child(customerID);
        stageRef = customerRef.child(stage);
        levelRef = stageRef.child(level);

        levelRef.addListenerForSingleValueEvent(levelListener);

        buttonListener = new ValueEventListener() {

            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                buttonBottom.setText(dataSnapshot.child("button_bottom").getValue(S
                buttonTop.setText(dataSnapshot.child("button_top").getValue(String.
```

```
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
             //Some firebase error
            }
        };

        actionListener = new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                actionBtn.setText(dataSnapshot.getValue(String.class));
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                Log.d("Firebase error", databaseError.toString());
            }
        };

        configListener = new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                for (DataSnapshot configSnapshot: dataSnapshot.getChildren()) {
                    configuration.put(configSnapshot.getKey(), configSnapshot.getV
                }
                timeCollectionManager.setConfig(configuration);
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                Log.d("Firebase error", databaseError.toString());
            }
        };

        if (buttonsRef != null) {
            buttonsRef.addListenerForSingleValueEvent(buttonListener);
        }
    }
```

C.5.2 FeedbackManager

```
public class FeedbackManager {

    private String dateTimeInformation;
    private UserData userData;
    private DatabaseReference database;
    private DatabaseReference feedbackRef;

    public FeedbackManager(UserData userData, String customerID) {
        database = FirebaseDatabase.getInstance().getReference();
        this.userData = userData;

        GeneralHelper generalHelper = new GeneralHelper();
        dateTimeInformation = generalHelper.getDateTime();

        feedbackRef = database.child(customerID);

        Map<String, Object> feedback = new HashMap<String, Object>();
        DatabaseReference newFeedbackRef = feedbackRef.push();
        feedback.put("android_id", userData.getAndroidId());
        feedback.put("phone_model", userData.getPhoneModel());
        ...
        }

//Update the location with the new data.
        newFeedbackRef.updateChildren(feedback);
    }

}
```

C.5.3 Feedback Path Sub-module handling GUI events

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
                final String item = (String) parent.getItemAtPosition(position);
                userData.addToIssueList(level, item);
                String nextElement = (String) alternatives.get(item);
                level = nextElement;
                if (alternativesHelper.nextLevelCheck(nextElement)) {
                    levelRef = stageRef.child(nextElement);
```

```
                levelRef.addListenerForSingleValueEvent(levelListener);
            } else if(nextElement.equals("continue")) {
                listView.setVisibility(View.GONE);
                buttonTop.setVisibility(View.VISIBLE);
            } else if (nextElement.equals("feedback")) {
                listView.setVisibility(View.GONE);
                feedbackText.setVisibility(View.VISIBLE);
                submitBtn.setVisibility(View.VISIBLE);
            } else {
                customerActions = actionsGroupRef.child(customerID);
                actionRef = customerActions.child(nextElement);
                actionRef.addListenerForSingleValueEvent(actionListener);
                listView.setVisibility(View.GONE);
                actionBtn.setVisibility(View.VISIBLE);
                buttonTop.setVisibility(View.VISIBLE);
            }
        }
    });
```

Appendix D
Platform Client Library

D.1 Development environment

D.1.1 Tools used

Android studio: An IDE for developing Android applications.

D.2 Development

D.2.1 MessagingService

```
private ServiceConnection serviceConnection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {

         //Retrieving the Messenger of the service.
            serviceMessenger = new Messenger(service);
            serviceConnected = true;

            // Register our messenger also on Service side:
            Message msg = Message.obtain(null, MESSAGE_TYPE_REGISTER);
            msg.replyTo = mMessenger;

            try {
                serviceMessenger.send(msg);
            } catch (RemoteException e) {
                // We always have to trap RemoteException
// (DeadObjectException
                // is thrown if the target Handler no longer exists)
```

```
                e.printStackTrace();
            }
            sendToService(customer, MESSAGE_TYPE_CUSTOMERID);
            sendToService(stage, MESSAGE_TYPE_STAGE);
        }

//Connection dropped
        @Override
        public void onServiceDisconnected(ComponentName className) {
            serviceMessenger = null;
            serviceConnected = false;
        }
    };
```

D.2.2 serviceConnection

```
private ServiceConnection serviceConnection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {

         //Retrieve the Messenger of the Overlay Service Application.
            serviceMessenger = new Messenger(service);
            serviceConnected = true;

            // Register our messenger also on Service side:
            Message msg = Message.obtain(null, MESSAGE_TYPE_REGISTER);
            msg.replyTo = mMessenger;

//Try to send the message with the Messenger object of the Provider Application.
            try {
                serviceMessenger.send(msg);
            } catch (RemoteException e) {
                // Handle RemoteException in case we are not able to send message t
            }
            //Send initializing messages to the Overlay Service Application.
            sendToService(customer, MESSAGE_TYPE_CUSTOMERID);
            sendToService(stage, MESSAGE_TYPE_STAGE);
        }

        //Connection dropped.
        @Override
```

```
    public void onServiceDisconnected(ComponentName className) {
    //Code to handle disconnect.
        ...
    }
};
```

Appendix E

Video Test Application - Simula

E.1 Development environment

Tools used

Android studio: An IDE for developing Android applications.

E.2 Requirement specification

E.2.1 Development goals

- Make it possible for users to give feedback on downloading and watching a video.
- Make an application that communicates with the platform.
- Have an application that can be used for user testing of the platform.

E.2.2 Functional requirements

- Randomly select a video to be downloaded (Originally between a set of videos provided by Simula).
- Download the video.
- Play the downloaded video.
- Choose another video if desired.
- Be able to send feedback stages to the platform.
- Give feedback based on the specific stage the application is in.
- Continue to watch movie after feedback, watch another movie or exit.

E.2.3 Non-functional requirements

- Reliability to try to avoid application annoyance so feedback can be focused on the videos.

E.2.4 Defining actions

We have only defined one action for the Video test application.

watchother: This action contains the String "Watch another movie" that is set as text on the action button of the platform. When pushed the action message will be sent to the Video test application so we can display a view for selecting another video to watch.

E.2.5 Defining application specific feedback paths

download stage

Contains a level with feedback alternatives related to downloading the video file.

- level0
  - Download time too long: continue
  - Other: feedback

end stage

Contains levels of feedback alternatives related to being finished with watching a video.

- level0
  - Other: feedback
  - Quality: level1
  - Slow video start: watchother
  - Uninterested in content: watchother
- level1
  - Frozen video: watchother
  - Loss of sound: watchother
  - Non-sync audio and video: watchother

- Other: feedback
- Poor image quality: watchother
- Poor sound quality: watchother

playing stage

Contains levels of feedback alternatives related to the video itself as it is playing.

- level0
  - Bad quality: level1
  - Other: feedback
  - Uninteresting content: watchother
- level1
  - Audio and video not in sync: continue
  - Bad sound quality: level2audio
  - Bad video quality: level2video
  - Other: feedback
- level2audio
  - Loss of sound: continue
  - Other: feedback
  - Poor sound quality: continue
- level2video
  - Frozen video: continue
  - Other: feedback
  - Poor image quality: continue

E.2.6 Use cases

Use case - Downloading a video file

Ben is carpooling with a friend to work. He pulls up his phone and starts the Video test application to watch a video. The connection is a bit poor, so the download of the video file is taking forever. This annoys Ben, so he decides to give a rating. He sees two alternatives, "Download time too long" and "Other". He chooses "Download time too long" as it fits with his annoyance. He then pushes the continue button to continue the download process.

Use case - Playing a video

Susie is watching a video at home using the Video test application on her phone. The application plays a random video to her. She thinks the video she is currently watching is super boring, so she pushes the feedback button. She is then shown 3 feedback alternatives, "Bad quality", "Other" and "Uninteresting content". She pushes "Uninteresting content" as it fits her needs. She is then displayed 3 buttons, Continue, Watch another movie and Exit. She decides to give the application another chance and pushes watch another movie. She is then shown a list of videos and chooses "Big Buck Bunny". She enjoys this video a lot more and watches it until the end. When the video reaches the end, the feedback mood selection menu opens. She pushes the happy smiley as she was satisfied with the video.

E.3 Development

E.3.1 Starting a movie

```
public void startMovie() {
        videoView.setVideoPath(url.toString());

        mediaController.setAnchorView(videoView);
        mediaController.setMediaPlayer(videoView);
        videoView.setMediaController(mediaController);
        videoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mediaPlayer) {
                stage = "end";
                sendToService(stage, MESSAGE_TYPE_STAGE);
                sendToService("", MESSAGE_TYPE_PROMPT);
            }
        });
        videoView.start();
    }

    public void playVideo(int vNumber) {
        localPath = VideoActivity.this.getFilesDir().toString() + "/" + localPaths.
        Log.d("localpath", localPath + VideoActivity.this.getFilesDir().toString())
        Log.d("vid address", vidAddresses.get(vNumber));
        extraMap.put("video_name", videoNames.get(vNumber));
        Log.d("play_video", "localPath: " + localPath);
        try {
            url = new URL(vidAddresses.get(vNumber).toString());
```

```
            Log.d("url", url.toString());
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }

        showProgress(vidAddresses.get(vNumber));

        if (url != null) {
            stage = "playing";
            sendToService(stage, MESSAGE_TYPE_STAGE);
            downloadFileTask = new DownloadFilesTask().execute(url);
            //startMovie();
        }
    }
```

Appendix F
DASH Video Test Application

F.1 Development environment

F.1.1 Tools used

Android studio: An IDE for developing Android applications.

Platform Client Library: A Platform Client Library we made to help providers communicate with the feedback platform.

ExoPlayer: A media framework provided by Android that lets us stream media. It contains a lot of useful metrics that we can collect upon streaming.

F.2 Requirement specification

F.2.1 Development goals

- Make it possible for users to give feedback on watching a DASH video.
- Identify and use useful data measurements on the PAPP.
- Use the Platform Client Library to communicate with the platform.
- Have an application that can be used for user testing of the platform.
- User test in real world mobile scenarios rather than on nodes.

F.2.2 Functional requirements

- Select a video from Simulas DASH server.
- Streaming DASH videos using Androids ExoPlayer framework.

- Collect important provider metadata.
- Choose another video if desired.
- Be able to send feedback stages to the platform.
- Give feedback based on the specific stage the application is in.
- Continue to watch movie after feedback, watch another movie or exit.

F.2.3 Non-functional requirements

- Reliability to try to avoid application annoyance so feedback can be focused on the videos.

F.2.4 Defining application specific actions

We have only defined one action for the DASH. It is the same as for the Video test application.

- watchother: "Watch another movie"

F.3 Defining application specific feedback paths

end stage

Contains levels of feedback alternatives with their corresponding next path element related to being finished with watching a video.

- level0
  - Audio: level1audio
  - Other: feedback
  - Slow video start: continue
  - Uninteresting video: watchother
  - Video: level1video
- level1audio
  - No sound: continue
  - Other: feedback
  - Poor sound quality: continue

- level1video
  - Changing video quality: watchother
  - Frozen video: continue
  - Other: feedback
  - Poor video quality: continue
  - Video not starting: continue

prestream stage

Contains a level with feedback alternatives with their corresponding next path element related to what the user is doing before streaming a video, like i.e. using the menu.

- level0
  - Bored: feedback
  - Other: feedback
  - Poor design: feedback

streaming stage

Contains levels of feedback alternatives with their corresponding next path element related to watching a streaming video.

- level0
  - Audio: level1audio
  - Buffering too long: continue
  - Other: feedback
  - Uninteresting video: watchother
  - Video: level1video
- level1audio
  - No sound: continue
  - Other: feedback
  - Poor sound quality: continue
- level1video
  - Changing video quality: continue

- Frozen video: continue
- Other: feedback
- Poor video quality: continue
- Video not starting: continue

Appendix G
Website development

G.1 Development environment

G.1.1 Tools used

Node JS[1] Event-driven I/O server-side JavaScript (JS) environment.

Express JS[2] is a minimalist web framework for node we use to set up a simple server on top of node.

ECMAScript2015 (ES6)[3] is a scripting language specification for JS. It lets us make more complex browser-based applications in JS easier.

Webpack[4] is a module builder we use to transpile ES6 code and make a JS bundle from the modules.

React[5] is a JS library for creating user interfaces with various features by Facebook and Instagram.

Firebase and reactFire API[6] is a cloud based back-end service that stores data in a NOSQL database and can be managed by their reactFire API.

React d3[7] is a JS library for charts as react components.

React google maps[8] is a JS library for helping us set up google maps components in react components.