Test Generation for Performance Evaluation of Mobile Multimedia Streaming Applications

Mustafa Al-tekreeti[1], Kshirasagar Naik[1], Atef Abdrabou[2], Marzia Zaman[3] and Pradeep Srivastava[3]
[1]*University of Waterloo, Ontario, Canada*
[2]*UAE University, Al-Ain, U.A.E.*
[3]*Technologie Sanstream, Quebec, Canada*

Keywords: Multimedia Mobile Streaming, Performance Testing, Coverage Criteria.

Abstract: In this paper, we propose a model based test generation methodology to evaluate the impact of the interaction of the wireless network and the application (app) configurations on the performance of a mobile multimedia streaming app. The methodology requires four artefacts as inputs, namely, a behaviour model of the software under test (SUT), a network model, a test coverage criterion, and desired performance levels. The methodology consists of three steps. First, two performance models are developed: mathematical and simulation. Second, to evaluate the end-user quality of experience (QOE), test generation is formulated as an inversion problem. To account for different types of performance models, the inversion problem is solved as an optimization problem. Third, the necessary information to execute test cases is inferred using the simulation model. Two test coverage criteria are proposed: user-experience (UE) and user-experience-and-input-interaction (UEII). The mathematical performance model for a streaming app is developed using Markov chain. To account for realistic network behaviours, the Markov chain is solved using the supplementary variable technique (SVT). A reusable network model is developed for a mobile device that has a network access through a WiFi LAN. Finally, the effectiveness of the methodology is evaluated in comparison with random testing.

1 INTRODUCTION

Performance is an important property of software systems, having a vital impact on user's experience. In the mobile systems domain, the main theme is being context sensitive (Liu et al., 2015), imposing extra requirements on mobile software development. Being able to communicate with many network types necessitates testing whether the app will perform as required under different environmental and contextual scenarios (Diaz et al., 2010). However, testing mobile apps for network behaviour is challenging since it requires multidisciplinary expertise. Another important aspect of mobile apps is the emphasis on the user experience. Therefore, there is a need for performance testing methodologies that take into account both the network behaviour and the end-user's QOE.

In this paper, we consider an important category of networked apps, which are mobile multimedia streaming apps. Figure 1(a) depicts the main elements of the system model. We aim to evaluate the interaction of network operating parameters (NOPs) and app configuration parameters (ACPs) on the performance

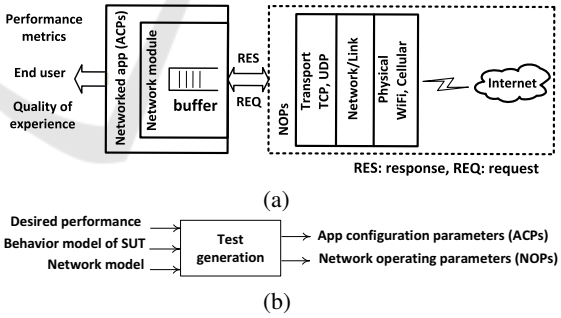

Figure 1: System model and test generation scheme.

of a mobile streaming app. We assume the app to be functionally correct. NOPs are a set of controllable parameters that model the network condition, such as data rate. NOPs are network technology dependent. ACPs represent a set of app configuration settings that have an impact on the performance metric under consideration, such as the size of the receiving buffer.

The main idea in generating tests is shown in Figure 1(b). Determining test inputs that lead to certain performance behaviour is akin to solving the *inversion* problem (Kumar et al., 2015). The inver-

Al-tekreeti, M., Naik, K., Abdrabou, A., Zaman, M. and Srivastava, P.
Test Generation for Performance Evaluation of Mobile Multimedia Streaming Applications.
DOI: 10.5220/0006609302250236
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 225-236
ISBN: 978-989-758-283-7

sion problem is the problem of inferring the *causes* by observing the *effects*. This problem is solved in three main steps: system parametrization, forward modelling, and inverse modelling (Tarantola, 2005). In this work, system parametrization corresponds to identifying both ACPs and NOPs. Forward modelling corresponds to the performance model development. Inverse modelling is the optimization problem formulation that when solved test input is generated.

The desired performance level is a quantitative measure of the performance metric under consideration. Generally, performance metrics are evaluated using statistical measures such as mean, percentage, and probability. The network model captures how the quality of network service quantified in terms of NOPs impacts the performance behaviour of the SUT. Different metrics are used to model network quality of service. In streaming apps, user experience is mainly influenced by the frame inter-arrival time delay. The behaviour model of SUT is an abstraction of the app dynamics that capture the performance metric under consideration. In this work, we use activity diagrams to describe this model.

Since performance metrics are statistical measures, extra information is required to execute the generated test cases. Test execution parameters (TEPs) encompass all the necessary information to execute test cases. From a statistical point of view, each test case is an experiment. Therefore, we need to know how many times the experiment should be repeated, or for how long it should be executed, so that the output is statistically reliable. Thus, we aim to design a set of test cases where each test case is basically a set of ACPs, NOPs, TEPs and the expected performance level. In other words, given the SUT is executed with the determined parameters of ACPs, NOPs, and TEPs, the observed performance level is statistically equivalent to the expected performance level if the SUT is correctly implemented from the performance point of view. Because performance metrics are mainly continuous, infinite number of performance levels and test cases are anticipated. Therefore, test selection strategies are needed to generate an effective set of performance test cases.

In this paper, we propose a methodology to generate test cases to evaluate a mobile multimedia streaming app. We adopt a black-box model-based testing approach (Siavashi and Truscan, 2015). The methodology is realized by a procedure of three steps. First, two performance models (mathematical and simulation) to capture the interactions between the SUT and the network are developed using Markov chains. Second, generating tests to evaluate the end-user experience is formulated as an inversion problem and solved as an optimization problem. Third, TEPs are inferred using the simulation model. To enhance the quality of the generated test cases, two test coverage criteria are proposed: i) user experience (UE) and ii) user experience and input interaction (UEII). In the UE criterion, test cases are generated to fully cover the identified categories of end-user experience. In the UEII criterion, test cases are generated to cover end-user experience and interactions of the input parameters simultaneously. We develop a reusable network model for a mobile app that downloads data via a WiFi interface and over the User Datagram Protocol (UDP). In summary, we make the following contributions:

- we propose a test generation methodology to evaluate the impact of the interaction of network and app configurations on the performance of a mobile streaming app;
- we propose two testing coverage criteria to enhance the quality of the generated test cases; and
- we show by means of a procedure how TEPs are inferred using the performance simulation model.

The paper is organized as follows. In Section 2, the related works are reviewed. In Section 3, the proposed methodology is introduced. In Section 4, we use an app example to illustrate the steps of the methodology. In Section 5, the efficacy of the proposed methodology is evaluated. In Section 6, the work is concluded and key challenges in applying the proposed methodology are discussed.

2 RELATED WORKS

In literature, considerable efforts have been made to integrate performance analysis with the software development life cycle. A comprehensive summary can be found in (Koziolek, 2010; Balsamo et al., 2004). The main objective in this research is to conduct performance analysis to evaluate design alternatives while the software is still in the development process, whereas our objective is to generate test cases and develop test selection strategies for performance testing. Our approach is orthogonal to theirs, but complementary to the early-stage performance testing phase.

Frequently, performance testing is viewed as load testing. Load testing is used to test large-scale multi-user transaction based software systems, such as web sites and database systems (Jiang and Hassan, 2015). In contrast, we target software apps that are developed for mobile devices, where network access is accomplished via wireless technologies. This difference in scope leads to a core distinction between our

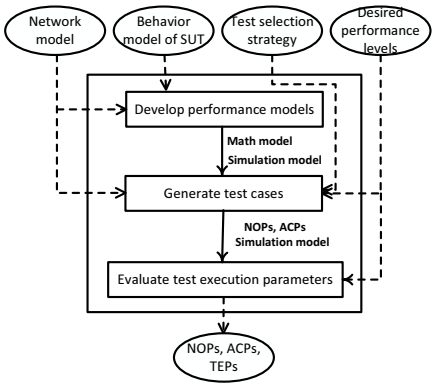

Figure 2: The main steps of the methodology.

work and the model-based load testing. The developed models in load testing is network-technology independent, while our approach models explicitly the network technology.

In spite of the much research efforts going into performance testing, test generation to evaluate the network impact on the performance of mobile apps has not received much attention (Liu et al., 2015; Joorabchi et al., 2013). The main focus is on testing networked apps for functional requirements (Sebih et al., 2014; Walls et al., 2015), designing software profilers to debug communication errors (Diaz et al., 2010), or providing test execution beds to evaluate the impact of the environment on wireless mobile apps (Satoh, 2004).

In software engineering, simulation models have been widely used in different software development activities. However, the emphasis is on using simulation models to evaluate alternative design choices (Kim et al., 2013), or to design test cases to verify software systems represented as simulation models (Matinnejad et al., 2016). In our work, we provide a procedure based on two statistical procedures to infer test execution parameters from simulation models.

3 THE METHODOLOGY

In this section, our test generation methodology is introduced. Figure 2 shows the main steps, the inputs, and the expected output of the methodology. We start by discussing the methodology input requirements. Then, we explain the methodology steps.

3.1 Inputs to the Methodology

The methodology requires four different artefacts as inputs. In this section, we describe them briefly:

3.1.1 Behaviour Model of the SUT

This model should describe how the app-network interactions impact the performance metric under consideration. According to Figure 1(a), the app-network interactions in streaming apps are well modelled by capturing the buffering behaviour of the app. The outcomes of this task are the behaviour model of the SUT and the set of app configuration parameters (ACPs) that affect the considered performance metric. We use activity diagrams to describe this model.

3.1.2 Network Model

This model should capture how the wireless network affects the considered performance metric. In general, network models are determined by the technology (WiFi or cellular) and the transport protocol (TCP or UDP). As shown in Figure 1(a), the app interacts with the network through a basic *request-response* (REQ-RES) mechanism. In multimedia streaming, the network's impact can be captured by modelling the RES inter-arrival time delay, which is a random variable. The expected outcomes of network modelling are the probability distribution of this random variable and the NOPs. To obtain the distribution, we employ distribution fitting using the first two moments: the mean and variance. Assuming the UDP protocol, we develop in Appendix A mathematical expressions for the mean and variance of packet (RES) inter-arrival time delay for a mobile user in a WiFi network.

3.1.3 Desired Performance Levels

The methodology requires a set of levels of the performance metric under consideration. In this work, we are interested in application level performance metrics that directly relate to the end-user's quality of experience (QOE). For example, the user experience of file transfer apps is assessed using two metrics: goodput and transfer time performance (Ivanovici and Beuran, 2010). Both are ratio metrics on a scale from 0 to 1, where 1 represents the best performance. Therefore, desired performance levels are merely numerical values sampled from the interval [0,1]. How many of those levels are needed and how they are chosen are addressed in the test selection strategies.

3.1.4 Test Selection Strategies

In general, a test selection strategy encodes the main objectives of the testing process. Satisfaction of testing objectives is measured using coverage criteria. In this methodology, we propose two coverage criteria (UE and UEII). Initially, we need to introduce some

notations. The sets of NOPs and ACPs are denoted as S_{NOP} and S_{ACP} with cardinalities n and m, respectively. Therefore, we have $n+m$ input parameters $p_1, p_2, ..., p_{n+m}$. To generate a test case, we assign a specific value vp_i to each parameter $p_i \in S_{NOP} \cup S_{ACP}$, where $vp_i \in Vp_i$, the set of permissible values of the parameter p_i, $1 \leq i \leq n+m$. Thus, a test case t_j is basically a tuple of the form (vp_1^j, vp_2^j, ..., vp_{n+m}^j, l_j), where l_j is the expected performance level.

i) User Experience (UE) Coverage Criterion

Herein, the objective is to generate test cases to cover the whole spectrum of the considered performance metric. However, since the performance spectrum is most likely to be continuous, an infinite number of test cases are needed. To generate a minimal set of test cases, partition testing (Grindal et al., 2005) is applied. The idea is to partition the parameter space into multiple regions where all the points of the same region are equivalent from the testing point of view. In our work, we apply partition testing to performance metrics, utilizing the fact that the end-user perception of the performance behaviour is discontinuous and can be characterized in terms of a few specific categories (QOE categories). Given R categories, we divide the performance spectrum W into R non-overlapped regions $r_1, r_2, ..., r_R$ such that $W = \bigcup_i^R r_i$. The number of QOE categories is app type dependent. Next, a performance level l_i is selected for each region such that $l_i \in r_i$, $1 \leq i \leq R$. Last, the corresponding test input vp_1^i, vp_2^i, ..., vp_{n+m}^i for l_i is determined. Procedure 1 summarizes the steps needed to generate test cases that satisfy this criterion:

- **Procedure 1:** Test selection strategy to achieve the UE coverage criterion
- **Inputs:** The number of QOE categories R
- **Outputs:** A test suite T of at least R test cases

S1: Partition W into R regions $r_1, r_2, ..., r_R$;
S2: Select the set S_l={l_j: $l_j \in r_j$, $1 \leq j \leq R$};
S3: $\forall\, l_j \in S_l$, generate test inputs $vp_1^j, vp_2^j, ..., vp_{n+m}^j$.

ii) User Experience and Input Interaction (UEII) Coverage Criterion

It may be noted that the UE coverage criterion is an output based criterion. However, satisfying this criterion is not enough to assure the quality of the app, because the designed test suite does not adequately cover the input space of the SUT. In combinatorial testing, it is emphasized that the effectiveness of the generated test cases increases as the coverage of the interactions of the input parameters increases (Yılmaz et al., 2014). Therefore, we are interested in generating test cases that satisfy both aspects of the SUT: the input space and the performance behaviour. For this purpose, we extend Procedure 1 as follows. First, we generate the set T_S of R seed test cases using Procedure 1. This set does cover the performance spectrum. Then, to enhance input space coverage, we use the seed tests to generate follow-up test cases so that a combinatorial metric is satisfied. The combinatorial metric is applied on subsets g_1, g_2, ..., g_G of the $S_{NOP} \cup S_{ACP}$ set, where $G \geq 1$. These subsets are constructed such that the parameters in which their interactions are important to cover are grouped together into a subset. A set of follow-up test cases T_{ij} is generated for every subset g_j and seed test $s_i \in T_S$. The parameters' values vp_1^i, vp_2^i, ..., vp_{n+m}^i of the follow-up test cases are determined as follows. The values of the parameters of the g_j subset are determined using the combinatorial metric. The remaining parameters $\{p : p \in S_{INP} - g_j\}$ are assigned the same values of the seed test case s_i.

The input space is constrained by conditions imposed by the network, the SUT, and by the condition that the expected performance levels for the follow-up test cases should remain within the same performance region of the test seed. That is, given the sets T_S, $S_G = \{g_1, g_2, ..., g_G\}$, C (the set of constraints), and a combinatorial metric b, $T_{ij} = Pert(g_j, s_i, C, b)$, $1 \leq j \leq G$, $1 \leq i \leq R$, where *Pert* realizes the follow-up test generation using the combinatorial coverage metric b. Therefore, test generation to satisfy UEII criterion is basically a combinatorial test generation with constrained parameters. The generated test suite T is the union of the follow-up test sets T_{ij} and the seed tests T_S. The following procedure summarizes the steps explained before:

- **Procedure 2:** Test selection strategy to achieve the UEII coverage criterion
- **Inputs:** R, G, C, and b
- **Outputs:** A test suite $T = \bigcup_{i,j} T_{ij} \bigcup T_S$.

S1: Generate the set T_S using "Procedure 1";
S2: Create the set S_G={g_j: $g_j \subset S_{INP}$, $1 \leq j \leq G$};
S3: $\forall\, j, i$, $T_{ij} = Pert(g_j, s_i, C, b)$.

Hence, the UEII criterion subsumes the UE criterion.

3.2 The Procedure of the Methodology

As shown in Figure 2, the methodology's steps are:

3.2.1 Develop Performance Models

By the performance model we mean any mathematical representation that quantitatively captures the im-

pact of the interaction of NOPs and ACPs on the performance of the SUT. In this work, we employ the Markovian framework to develop the performance models. This framework is appropriate when the system state is defined by the buffering behaviour of the SUT. We use supplementary variable technique (Cox and Miller, 1977) to solve the model. This technique is used if the stochastic process is not Markovian, allowing for more practical interactions between the SUT and the network to be modelled. In this methodology, two performance models are developed: mathematical and simulation. The simulation model is used to verify the mathematical model and in the test generation process as well. In Appendix B, the performance model (Equations (23-30)) of the considered app example is developed. This step requires the network model and behaviour model of the SUT.

3.2.2 Generate Test Cases

NOPs and ACPs are found by formulating test generation as an inversion problem. In order to determine the input that leads to a certain output, an inverse relationship should be derived. For most mathematical models, deducing a closed form for the inverse relationship may not be feasible. Furthermore, the structure of some models is unknown as in simulation models. Therefore, we cast the inversion problem as a root finding problem. Given the desired performance level $l_i \in S_l$, the test input is basically the root that satisfies the relationship:

$$Perf\_model(p_1, p_2, ..., p_{n+m}) - l_i = 0 \quad (1)$$

where *Perf_model(...)* represents the performance model. The roots (NOPs and ACPs values) can be found by reformulating Equation (1) as a minimization problem:

$$\textbf{Minimize} \quad |Perf\_model(p_1, p_2, ..., p_{n+m}) - l_i| \quad (2)$$

where $|\cdot|$ is the absolute value operator. We minimize the absolute of the difference to force the solver that the required minimum is zero. In this work, we use the mathematical performance model as the objective function, although the simulation model can also be used. The minimization problem is constrained by the conditions imposed by the network model and the semantics of the SUT behaviour.

3.2.3 Evaluate Test Execution Parameters

We employ the simulation model to determine the TEPs. We determine the parameters in two stages. First, using a univariate sequential procedure called Law and Carson (abbreviated as L&C) (Law, 2015), we estimate the mean run length for the simulation model to reach steady-state and use this value as an estimate of the mean execution time of the test case. We build a point estimator $\hat{T}_x$ and a confidence interval $CI(\hat{T}_x)$ so that the estimated value for the considered performance metric is within a pre-specified error from the true value. Second, the rest of TEPs are inferred simultaneously by utilizing the Bonferroni inequality (Charnes, 1995). This inequality provides a lower bound for the overall confidence level $(1-\zeta)$ given that the overall significant level ζ is equal to the sum of the individual significant levels. We construct individual confidence intervals using the Independent Replication Sequential (IRS) procedure (Law, 2015). The precision of estimation is controlled by the *relative error* in estimation γ. Procedure 3 summarizes how to infer TEPs using the simulation model:

- **Procedure 3**: Determine TEPs using the simulation model
- **Inputs**: The test case (vp_1^j, vp_2^j, ..., vp_{n+m}^j, l_j), γ, ζ, and the number of replications
- **Outputs**: The corresponding TEPs values

S1: Invoke the L&C procedure to obtain $\hat{T}_x$ and $CI(\hat{T}_x)$;

S2: Choose ζ_1, ζ_2, ..., ζ_k so that $\sum_{i=1}^{k} \zeta_i = \zeta$;

S3: Invoke IRS procedure to obtain the estimated mean and the confidence interval for the remaining TEPs.

4 USING THE METHODOLOGY

In this section, we apply the proposed methodology on an example of a mobile multimedia streaming app. The considered performance quality is the smoothness of the streaming as perceived by the end user. We assume that the app utilizes the UDP protocol and the last hop to the end user is through a wireless connection using a WiFi hotspot that implements the IEEE 802.11 protocol. We start this section by defining the behaviour model of the SUT and the network model. Then, we apply the proposed methodology to generate test cases using both test selection strategies.

4.1 Behaviour Model of the SUT

We assume that the SUT implements a *progressive* streaming in which both frame downloading and decoding are interleaved. The app behaviour is modelled by two main components: *downloader* and *player*. Both components interact with each other through a *playback buffer*. Figure 3 shows the desired behaviour of the SUT. At the beginning, the app

is in the *Buffering* phase. In this phase, the downloader starts fetching media frames from the network and queues them in the buffer, while the player is still off. The app remains in this phase until the data level in the playback buffer reaches a certain limit usually known as a *high watermark* level (*M*). This level determines the length of the buffering phase and thereby the length of the time period the user has to wait before the player starts playing. Also, this level determines when the app stops asking for new frames. The downloader resumes fetching media frames whenever data level drops below a certain limit, known as a *low watermark* level (*L*). This level represents the minimum amount of data in the buffer to ensure smooth playback. To capture end user experience, we consider *the frequency of rebuffering events* as the performance metric (Mok et al., 2011). This metric has a direct relationship with the frequency of visiting the *Empty Buffer* state. The SUT behaviour is characterised by three configuration parameters (ACPs): playback buffer size (*B*), *M*, and *L*.

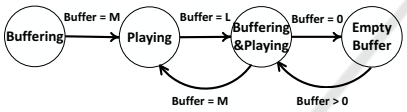

Figure 3: The behaviour model of the app example.

4.2 Wireless Network Model

The streaming is through a WiFi AP. We assume all the fluctuations in the wireless channel and in the wired network manifest as a time delay. Thus, packet loss is negligible. The probability distribution (CDF) of the frame inter-arrival time delay is matched with Hyper-Erlang distribution using the mean and variance that are given by Equations (11) and (13) in Appendix A, respectively. The network impact is captured by three operating parameters (NOPs): data rate *D*, the mean rate of frame arrival at the AP per user λ, and the number of end users *N* connected to the AP. To validate the matched CDF, we conduct simulation experiments using the Network Simulator NS2. Figure 4 shows both the empirical and analytical CDFs.

4.3 The Procedure

Given the network model and the behaviour model of the SUT, now we apply the methodology procedure:

4.3.1 Performance Models

We develop two performance models: mathematical and simulation. The performance metric under consideration correlates with the fraction of time of being in the *Empty Buffer* state out of the total time of streaming. Since the performance metric is a steady-state metric, the *Buffering* phase is not included in the models. To evaluate the performance metric, the stationary distribution of the playback buffer length is required. We are only interested in the probability of having zero frames in the playback buffer π_0.

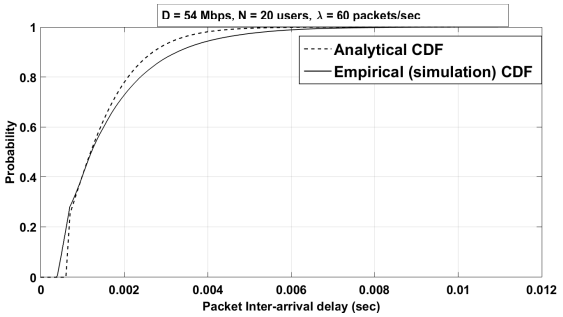

Figure 4: The empirical and analytical CDFs.

To facilitate the modelling process, we assume that the frame decoding rate is exponentially distributed. Nevertheless, the stochastic process is still not Markovian, as the frame arrivals are not exponentially distributed. Therefore, we develop the model using the supplementary variable technique as shown in Appendix B. The performance model is pictorially shown by the Markov chain in Figure 7 and mathematically described by the set of Equations (23-30).

To verify the performance model, Figure 5 shows the performance metric π_0 with different buffer sizes and for both simulation and mathematical models, where μ represents the mean rate of frame decoding, f is a tuning parameter to control the relation between the frame arrival rate and μ; p and k are Hyper-Erlang distribution parameters. In the simulation model, we simulate a streaming session of 30 minutes. Each simulation experiment is repeated 50 times.

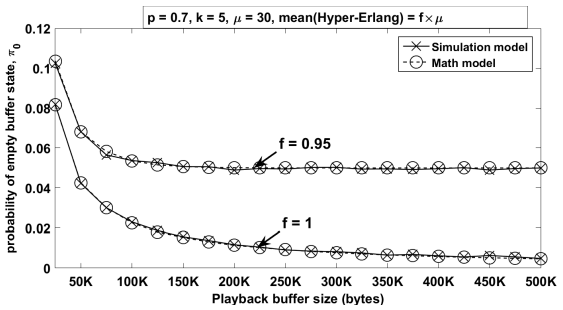

Figure 5: The considered performance metric (π_0) versus buffer size for both performance models.

4.3.2 Test Generation

Before solving the optimization problem, the constraints and bounds of the input parameters have to

be defined. The ACPs (B, M, and L) are defined as integer number of multimedia frames. The semantics of the SUT introduce the following two constraints:

$$M \leq B \text{ and } L \leq M-1. \quad (3)$$

The high watermark cannot be higher than the buffer size, and the low watermark cannot be equal or higher than the high watermark. For the NOPs, according to the IEEE 802.11 a/g standard data rate D can take either of the following values: 6, 9, 12, 18, 24, 32, 48, or 54 Mbps. It mainly relates to the quality of the wireless connection between the AP and the end user. Regarding the number of users N, the network model is validated with the number of users that ranges from 4 to 30. λ is the only continuous parameter. Using the upper and lower bounds of N and D and the constraints imposed by the network, we bound λ between 10 and 416 $packets/sec$, and we represent this parameter by the following 42 discrete values [10,20,30,...,410,416].

In multimedia streaming, the mean encoding rate at the server is set according to the end-user device characteristics. Thus, we assume that the mean arrival rate to the end user $1/E_r$ (Equation 11) is equal to the mean decoding rate (μ) (Li et al., 2009), i.e.:

$$\mu = \frac{1}{E_r} \quad (4)$$

Solving Equation (4) in terms of NOPs (λ, N, and D), a non-linear equality constraint is obtained. Since most optimization solvers do not easily accommodate non-linear equality constraints, we assume that the mean of the packet inter-arrival time delay falls in a closed interval around $1/\mu$. Thus:

$$\frac{k1}{\mu} \leq E_r \leq \frac{k2}{\mu} \quad (5)$$

where $k1$ and $k2$ are tunable parameters introduced to control the width of the closed interval. By doing so, we relax the non-linear equality constraint to two non-linear inequalities that are easier to deal with.

Another constraint that should be taken into consideration is that the traffic intensity (ρ) at the AP should be less than 1:

$$\rho < 1 \quad (6)$$

Otherwise, the buffer at the AP will build up infinitely. Therefore, the optimization problem has five constraints (inequalities 3, 5, and 6).

i) Test Generation using UE Coverage Criterion

In **Procedure 1.S1**, the performance spectrum is partitioned according to the end-user QOE categories. In multimedia streaming and using the probability of empty buffer state π_0 as a performance metric, three different end-user experiences are reported (Mok et al., 2011). If π_0 is less than 2%, the video quality is high; between 2% and 15%, the quality is medium; and above 15%, the quality is poor. Hence, we divide the performance spectrum into the reported three regions (R=3). Then, we select a performance level for each region $\{\pi_0 = 0.01, \pi_0 = 0.05, \pi_0 = 0.2\}$. Solving the minimization problem for each of them, the corresponding network and SUT parameters' values are determined as shown in part (a) of Table 1 (the left most seven columns). The buffer size B is bounded between 10 and 40 frames, the mean of the decoding rate μ is 30 $frames/sec$, and the parameters $k1$ and $k2$ are 0.75 and 1.25, respectively.

ii) Test Generation using UEII Criterion

We consider the three generated test cases that are listed in part (a) of Table 1 as test seeds T_S (**Procedure 2.S1**). We utilize the combinatorial coverage metric *each-choice* (Grindal et al., 2005) to enhance the input space coverage. We choose to cover the interaction of S_{ACP} and S_{NOP} independently (**Procedure 2.S2**)(i.e., G=2, g_1=S_{ACP}, g_2=S_{NOP}). For g_1 subset, we apply each-choice criterion for high watermark (M) and low watermark (L) only, since the playback buffer size (B) does not directly affect the system output. The parameters B, D, λ, and N are kept fixed on seed' values. The same procedure is applied for g_2 subset.

To generate the follow-up test cases (**Procedure 2.S3**), we use the combinatorial tool ACTS v3 (ACT, 2016). Because the input parameters are constrained by non-linear constraints, we first identify the parameter values that satisfy the constraints and then we apply the combinatorial testing criterion. Applying **Procedure 2.S3** for g_1 subset, we get 33 (T_{11}), 40 (T_{12}), and 9 (T_{13}) follow-up test cases for the performance regions $[0, 0.02]$, $(0.02, 0.15]$, and $(0.15, 1]$, respectively. As an example, we show the set T_{13} below:

$$T_{13} = \{(10,5),(11,1),(12,1),(13,1),(8,7),(6,4),$$
$$(5,3),(9,6),(7,2)\}$$

The first element of each tuple is M and the second is L. For g_2 subset, we get 5 (T_{21}), 3 (T_{22}), and 3 (T_{23}) follow-up test cases for the regions $[0, 0.02]$, $(0.02, 0.15]$, and $(0.15, 1]$, respectively. As an example, we show the set T_{23} below:

$$T_{23} = \{(54\text{M}, 110, 15),(54\text{M}, 150, 11),(54\text{M}, 330, 5)\}$$

The first element of each tuple is D, the second is λ, and the third is N. The remaining parameters' (B, M, and L) values are fixed on test seed values. Hence, the designed test suite T using the UEII criterion is the union of the sets T_{11}, T_{12}, T_{13}, T_{21}, T_{22}, T_{23}, and T_S.

4.3.3 Determining TEPs

For the app example, each test case is a streaming session with certain configuration parameters. To execute each test case, the length of the streaming session and the size of the multimedia file need to be determined. Since we have two TEPs parameters only, we do the estimations without the need to use Bonferroni inequality (**Procedure 3.S2**). The used values for γ, ζ, and the number of replications are 0.075, 0.1, and 10, respectively. We build a point estimator and a confidence interval independently for the mean test case execution time $\hat{\mathtt{T}}_x$ (**Procedure 3.S1**) and the mean file size $\hat{\mathtt{F}}_s$ (**Procedure 3.S3**) so that the estimated probability of the empty playback buffer state ($\hat{\pi}_0$) is within a pre-specified error from the true value. We estimate $\hat{\mathtt{T}}_x$ and $\hat{\mathtt{F}}_s$ for the three test cases as shown in part (b) of Table 1. We gauge the adequacy of the estimated simulation time by controlling the width of the confidence interval $\mathtt{CI}(\hat{\pi}_0)$ through the parameter γ. As expected, test case execution time is test case dependent. Moreover, as π_0 increases, the required time to reach steady-state decreases.

5 EVALUATION OF THE APPROACH

We use random testing as a baseline to evaluate the effectiveness of the proposed methodology. Herein, we use the phrases test configurations and test cases interchangeably. As the implementation of the SUT is not available, we generate test cases randomly using the procedure shown in Figure 6, where R is the number of performance regions, Q is the number test cases per region, and C is the coverage criterion. We first use the UE coverage criterion. We use the developed performance model to evaluate the performance behaviour l_c (π_0) of the configuration t_c. To anticipate the incurred cost of the random test generation, we keep track of four types of test configurations: Invalid executable test configurations (IETCs), Invalid non-executable test configurations (INTCs), Valid-and-useful test configurations (VTCs), and Valid-but-not-useful test configurations (VNTCs).

A test configuration t_c is invalid if the chosen parameters' values do not satisfy the constraints imposed by the network model, SUT, or both. If t_c does not satisfy the network requirements only (invalid NOPs), the SUT can still execute, while if t_c does not satisfy the constraints imposed by the SUT (invalid ACPs), it is not executable. We assume that the SUT implements the necessary logic to catch out inconsistent ACPs. Therefore, we have two types of invalid test configurations: executable (IETCs) and non-executable (INTCs). It is important to differentiate between them as IETCs are more expensive than INTCs from the time cost point of view. If t_c satisfies all the imposed constraints, it is a valid configuration. Moreover, if this valid configuration increases the coverage of the designed test suite so far, it is considered as a valid-and-useful configuration. Otherwise, it is a valid-but-not-useful test configuration.

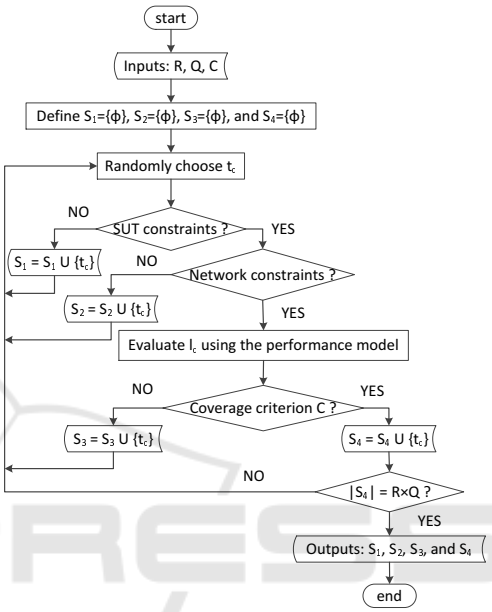

Figure 6: The implemented flowchart of test generation using random testing. The sets S_1, S_2, S_3, and S_4 are the sets of INTCs, IETCs, VNTCs, and VTCs, respectively.

To estimate the incurred cost of generating a test suite of size $R \times Q$, we design an experiment with R and Q as controllable factors. The obtained results are shown in Table 2. The results are basically the median of 10 repetitions. For example, to randomly generate a test suite with one test case ($R = 1$, $Q = 1$), the incurred cost is approximately the sum of the time cost of running 566 IETCs and one VTCs, while in our framework, we need to execute the SUT with one VTC only. Since performance metrics are mostly statistical, the time needed to observe the performance behaviour l_c of the real system is not trivial. As we employ a heuristics based optimization formulation which solely depends on function (performance model) evaluations to find the optimal point (test case), random testing can be better than ours if the performance model evaluation is more expensive than running the real SUT and/or if the employed optimizer needs more model evaluations than random testing. For the first condition, even if the performance is modelled using a simulation model, many techniques have been proposed to speed up simulation

Table 1: The augmented set of test cases. $\hat{T}_x$ is measured in `minutes`, D is measured in *Mbps* and $\hat{F}_s$ is measured in `MB`.

(a) test cases							(b) test execution parameters (TEPs)			
π_0	B	M	L	D	λ	N	$\hat{T}_x$	CI($\hat{T}_x$)	$\hat{F}_s$	CI($\hat{F}_s$)
0.01	34	34	4	18	131.965	7	159.288	[145.385,173.2]	425.8	[425.32,426.28]
0.05	38	31	7	32	162.8702	8	109.226	[74.161,144.291]	280.2018	[279.77,280.63]
0.2	24	7	2	6	98.9693	4	6.4	[4.651,8.148]	13.8505	[13.780,13.921]

Table 2: The cost of random testing in terms of the number of test configurations that need to be executed for different cases.

R	Q	Suite size	IETCs	INTCs	VNTCs	VTCs
1	1	1	566	1795.5	0	1
1	2	2	1471	5016	0	2
1	3	3	1849.5	6149	0	3
2	1	2	3193	10317	3	2
2	2	4	4817	15966	6.5	4
2	3	6	11472	38262	14.5	6
3	1	3	18290	60399	29.5	3
3	2	6	40018	1.3266×10^5	66.5	6
3	3	9	1.0785×10^5	3.5653×10^5	178.5	9

executions, while real system executions cannot be accelerated. For the second condition, many heuristics based optimization algorithms are available in literature that can perform better than our optimization solver. Indeed, within the used solver, many strategies can be used to fine tune its performance. In conclusion, there is still much room to enhance the performance of our framework compared to random testing.

In addition, as R increases, the incurred time cost increases and reaches astronomical values as the case with $R = 3$ and $Q = 3$. In this scenario, the time cost is approximately the sum of the time cost of running 1.0785×10^5 IETCs, 178.5 VNTCs, and 9 VTCs. In theory, as the number of performance regions (R) increases, the width of each region decreases and thereby the probability of getting a valid test case using random testing decreases. In contrast, the time cost of generating a test case by solving the inversion problem does not depend on the width of the region. Compared to random search, our optimization based approach employs a guided search to figure out valid test configurations. This conclusion also applies to UEII coverage criterion, as UEII metric builds upon UE and combinatorial metrics.

6 DISCUSSION AND CONCLUSION

In this paper, a model based test generation methodology was proposed to evaluate the impact of the interaction of the app configuration parameters and the network operating parameters on the performance of mobile multimedia apps. The methodology required four different artefacts as inputs: a behaviour model of the SUT, a network model, desired performance levels, and coverage criteria. The methodology comprised three steps: performance models development, test inputs generation, and estimation of TEPs. Test generation was formulated as an inversion problem and solved as a minimization problem. To generate effective test cases, two coverage criteria were proposed: i) user experience (UE) and ii) user experience and input interaction (UEII). We applied the proposed methodology on a mobile multimedia streaming app example. The effectiveness of the methodology was empirically evaluated in comparison with random testing. The incurred time cost to generate a test suite using random testing was estimated to be much more than the cost of our framework.

In model-based testing, model development is the most critical operation that is difficult to automate. In this work, constraints derivation is another intensive operation that is fortunately amenable to automation especially with the promising advances that have been made in symbolic computing. Moreover, solving the inversion problem as a black box optimization problem has enabled test generation regardless of the internal structure of the performance model and the used stochastic notation.

In the app example, the main observation was the high computing cost of evaluating the mathematical

model compared to the simulation model when the buffer size was beyond a certain value. To mitigate this issue, we can employ the simulation model in solving the inversion problem. In literature, there is an increasing interest in using simulation models in optimization problems (Gosavi, 2014). To enhance the quality of the generated test cases, we proposed the UEII coverage criterion. However, the network model has constrained NOPs by non-linear constraints, making test generation using combinatorial metrics very complicated. To overcome this issue, we exhaustively checked all combinations for the imposed constraints. However, this approach might be very expensive in terms of execution time, especially for systems with a large number of parameters and/or parameter values, which indicates the need for more powerful mechanisms to address such scenarios.

REFERENCES

(2016). Advanced combinatorial testing system (acts).

Balsamo, S. et al. (2004). Model-based performance prediction in software development: A survey. *IEEE Trans. on Soft. Eng.*, 30(5):295–310.

Bianchi, G. (2000). Performance analysis of the ieee 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535–547.

Charnes, J. M. (1995). Analyzing multivariate output. In *Proc. of the 27th conf. on Winter simulation*, pages 201–208. IEEE Computer Society.

Cox, D. R. and Miller, H. D. (1977). *The theory of stochastic processes*, volume 134. CRC Press.

Diaz, A., Merino, P., and Rivas, F. J. (2010). Mobile application profiling for connected mobile devices. *IEEE Pervasive Computing*, 9(1):54–61.

German, R. (2000). *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. John Wiley & Sons, Inc.

Gosavi, A. (2014). *Simulation-based optimization: parametric optimization techniques and reinforcement learning*, volume 55. Springer.

Grindal, M., Offutt, J., and Andler, S. F. (2005). Combination testing strategies: a survey. *Software Testing, Verification and Reliability*, 15(3):167–199.

Ivanovici, M. and Beuran, R. (2010). Correlating quality of experience and quality of service for network applications. In Adibi, S., editor, *Quality of service architectures for wireless networks: performance metrics and management*, chapter 15, pages 326–351. IGI Global, Pennsylvania, USA.

Jain, R. (1991). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

Jiang, Z. and Hassan, A. (2015). A survey on load testing of large-scale software systems. *IEEE Trans. on Soft. Eng.*, 41(11):1091–1118.

Joorabchi, M. E., Mesbah, A., and Kruchten, P. (2013). Real challenges in mobile app development. In *Int. Symp. on Emp. Soft. Eng. and Meas.*, pages 15–24. IEEE.

Kim, Y. et al. (2013). Validating software reliability early through statistical model checking. *IEEE software*, 30(3):35–41.

Koziolek, H. (2010). Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658.

Kumar, R. et al. (2015). Inverting a steady-state. In *Proc. of the 8th Int. Conf. on Web Search and Data Mining*, pages 359–368. ACM.

Law, A. M. (2015). *Simulation modeling and analysis*. McGraw-Hill, NY, fifth edition.

Li, M., Claypool, M., and Kinicki, R. (2009). Playout buffer and rate optimization for streaming over ieee 802.11 wireless networks. *ACM Trans. on Multimedia Computing, Communications, and Applications*, 5(3):26.

Liu, Y., Xu, C., and Cheung, S.-C. (2015). Diagnosing energy efficiency and performance for mobile internetware applications. *Software, IEEE*, 32(1):67–75.

Matinnejad, R. et al. (2016). Automated test suite generation for time-continuous simulink models. In *38th Int. Conf. on Soft. Eng.*, pages 595–606. ACM.

Mok, R. K., Chan, E. W., and Chang, R. K. (2011). Measuring the quality of experience of http video streaming. In *12th IFIP/IEEE Int. Symp. on Integrated Network Management and Workshops*, pages 485–492. IEEE.

Satoh, I. (2004). Software testing for wireless mobile computing. *IEEE Wireless Communications*, 11(5):58–64.

Sebih, N. et al. (2014). Software model checking of udp-based distributed applications. In *2nd Int. Symp. on Comp. and Net.*, pages 96–105. IEEE.

Siavashi, F. and Truscan, D. (2015). Environment modeling in model-based testing: concepts, prospects and research challenges: a systematic literature review. In *19th Int. Conf. on Eval. and Assess. in Soft. Eng.*, page 30. ACM.

Tarantola, A. (2005). *Inverse problem theory and methods for model parameter estimation*. siam.

Walls, R. J. et al. (2015). Discovering specification violations in networked software systems. In *26th Int. Symp. on Soft. Reliab. Eng.*, pages 496–506. IEEE.

Yılmaz, C. et al. (2014). Moving forward with combinatorial interaction testing. *Computer*, 47(2):37–45.

A PACKET DELAY STATISTICS IN A WIFI NETWORK

In this part, we derive analytical expressions for the mean and variance of the packet (or frame) inter-arrival time delay for streaming over a UDP protocol and via a WiFi network. We assume that the WiFi AP operates in the Distributed Coordination Function

(DCF) mode. In this mode, a station (mobile device or AP) can only send a MAC frame if the channel is sensed idle for a DIFS (distributed inter-frame space) interval of time. If the sender transmits and does not receive an ACK within a certain amount of time, a collision is detected and the frame has to be rescheduled for transmission. In this case, the sender has to wait for extra random amount of time after a complete DIFS interval of being sensed idle to send the frame. This random amount of time is called the back-off interval. We assume data exchange between stations are achieved using the four-way handshaking scheme. Therefore, the time needed to successfully send a packet T_s is given by (Bianchi, 2000):

$$T_s = T_{RTS} + T_{CTS} + 3 \times T_{SIFS} + T_{DIFS} + T_{ACK} + T \quad (7)$$

where T_{RTS}, T_{CTS}, T_{ACK}, and T_{SIFS} are the transmission times of ready-to-send packet, clear-to-send packet, ACK packet, and short inter-frame space time interval, respectively. The packet transmission time T depends on the packet size P and on the data rate D of the wireless connection:

$$T = T_{PHY} + \frac{H_{MAC} + H_{UDP} + P \times 8}{D} \quad (8)$$

where T_{PHY}, H_{MAC}, and H_{UDP} represent the PHY layer overhead, MAC layer header size, and UDP header size (20 Bytes), respectively. We assume that the WiFi AP has infinite buffer size, so the probability of packet loss due to AP buffer overflow is negligible. When a packet reaches the head of the AP buffer, the time duration seen by this packet from this instant to the instant at which it is successfully delivered to the end user corresponds to the service time of the packet S_t and it's mean is given by:

$$E[S_t] = T_s + \sigma \times \frac{W}{2} \quad (9)$$

where σ is the slot time interval. The back-off counter is a uniform random variable in [0,W].

Since the variance in packet service time is negligible, we assume that the AP as an M/D/1 queueing system. Therefore, the mean of the packet inter-arrival time delay E_r at the end user is basically the mean of the packet response time at the AP, which is the sum of the mean of queueing time delay and the packet service time (Jain, 1991):

$$E_r = E[S_t] + \frac{\rho \times E[S_t]}{2 \times (1-\rho)} \quad (10)$$

where ρ is the traffic intensity at the access point. Therefore, with few simplifications, the mean packet inter-arrival time delay is given by:

$$E_r = \frac{(2-\rho)}{2 \times \alpha \times (1-\rho)} \quad (11)$$

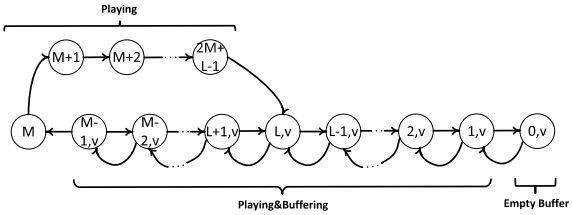

Figure 7: The state diagram of the application.

where α is the packet service rate at the AP. Consequently, the variance in packet inter-arrival time delay *Vard* is mainly due to the variance in the AP queueing waiting time and it is given by (Jain, 1991):

$$V_{ard} = \frac{N\lambda E[S_t^3]}{3(1-\rho)} + \frac{(N\lambda E[S_t^2])^2}{4(1-\rho)^2} \quad (12)$$

where N and λ are the number of users and the packet mean arrival rate for each user at the AP. With few manipulations, the variance in packet inter-arrival time delay settles down to the following equation:

$$V_{ard} = \frac{\rho \times (4-\rho)}{12 \times \alpha^2 \times (1-\rho)^2} \quad (13)$$

B PROBABILITY DISTRIBUTION OF THE PLAYBACK BUFFER LENGTH

In multimedia streaming, the end-user quality of experience correlates with the probability of empty buffer state π_0. Therefore, the stationary distribution of the playback buffer length should be evaluated. Since the buffer dynamics' process is not Markovian, the model is solved using SVT. In this technique, the state of the Markov chain should be redefined so that all the necessary information to determine the next state is available in the current state. Thus, the stochastic process is defined as follows:

$$\{B(t), V(t), t \geq 0\} \quad (14)$$

where $B(t)$ is a discrete random variable that represents the playback buffer length at time t, and $V(t)$ is a continuous random variable that represents the elapsed time from the last frame arrival till time t. Figure 7 shows the Markov chain of the SUT. All states that are expected to receive frames are augmented with the supplementary variable v to account for the elapsed time since the last arrival. Our objective is to find the steady state probability distribution of $B(t)$. The derivation depends on examining the short-term behaviour of the chain. For each state, a difference equation is derived using the basic law

of total probability of two mutually exclusive events. For state 0, it is:

$$\begin{aligned}\pi_0(t+\Delta t, v+\Delta t) =& \pi_0(t,v)(1-\lambda(v)\Delta t) \\ &+ \pi_1(t,v)\mu\Delta t + o(\Delta t)\end{aligned}$$

where $o(\Delta t)$ is the probability of having more than one event in a short interval Δt. For states $1, 2, ...,$ and $M-2$, it is:

$$\begin{aligned}\pi_n(t+\Delta t, v+\Delta t) =& \pi_n(t,v)(1-\lambda(v)\Delta t - \mu\Delta t) \\ &+ \pi_{n+1}(t,v)\mu\Delta t + o(\Delta t), \quad 1 \le n \le M-2\end{aligned}$$

For the states $M-1$ and M, they are:

$$\begin{aligned}\pi_{M-1}(t+\Delta t, v+\Delta t) =& \pi_{M-1}(t,v)(1-\lambda(v)\Delta t - \mu\Delta t) \\ &+ o(\Delta t)\end{aligned}$$

$$\begin{aligned}\pi_M(t+\Delta t) =& \pi_M(t)(1-\mu\Delta t) \\ &+ \int_0^\infty \pi_{M-1}(t,v)\lambda(v)\Delta t\,\mathrm{d}v + o(\Delta t)\end{aligned}$$

For the last $M-L-1$ states, It is:

$$\begin{aligned}\pi_n(t+\Delta t) =& \pi_n(t)(1-\mu\Delta t) + \pi_{n-1}(t)\mu\Delta t \\ &+ o(\Delta t), \quad M+1 \le n \le 2M-L-1\end{aligned}$$

The corresponding differential and partial differential equations are:

$$\frac{\partial \pi_0(t,v)}{\partial t} + \frac{\partial \pi_0(t,v)}{\partial v} = -\pi_0(t,v)\lambda(v) + \pi_1(t,v)\mu \tag{15}$$

$$\begin{aligned}\frac{\partial \pi_n(t,v)}{\partial t} + \frac{\partial \pi_n(t,v)}{\partial v} =& -\pi_n(t,v)(\lambda(v)+\mu) \\ &+ \pi_{n+1}(t,v)\mu, \quad 1 \le n \le M-2\end{aligned} \tag{16}$$

$$\frac{\partial \pi_{M-1}(t,v)}{\partial t} + \frac{\partial \pi_{M-1}(t,v)}{\partial v} = -\pi_{M-1}(t,v)(\lambda(v)+\mu) \tag{17}$$

$$\frac{\mathrm{d}\pi_M(t)}{\mathrm{d}t} = -\pi_M(t)\mu + \int_0^\infty \pi_{M-1}(t,v)\lambda(v)\mathrm{d}v \tag{18}$$

$$\begin{aligned}\frac{\mathrm{d}\pi_n(t)}{\mathrm{d}t} =& -\pi_n(t)\mu + \pi_{n-1}(t)\mu, \\ & M+1 \le n \le 2M-L-1\end{aligned} \tag{19}$$

subject to the following boundary conditions:

$$\pi_0(t,0) = 0 \tag{20}$$

$$\pi_n(t,0) = \int_0^\infty \pi_{n-1}(t,v)\lambda(v)\mathrm{d}v, 1 \le n \le M-1\backslash\{L\} \tag{21}$$

$$\pi_L(t,0) = \int_0^\infty \pi_{L-1}(t,v)\lambda(v)\mathrm{d}v + \pi_{2M-L-1}(t)\mu \tag{22}$$

For steady-state analysis, there is no need to specify initial conditions. To simplify solving the above equations, there is a tactic to remove $\lambda(v)$ from the equations reported in (German, 2000). Applying this tactic on Equations (15-18, 20-22), yields:

$$\begin{aligned}\frac{\partial p_0(t,v)}{\partial t} + \frac{\partial p_0(t,v)}{\partial v} &= p_1(t,v)\mu \\ \frac{\partial p_n(t,v)}{\partial t} + \frac{\partial p_n(t,v)}{\partial v} &= -p_n(t,v)\mu + p_{n+1}(t,v)\mu, \\ & \quad 1 \le n \le M-2 \\ \frac{\partial p_{M-1}(t,v)}{\partial t} + \frac{\partial p_{M-1}(t,v)}{\partial v} &= -p_{M-1}(t,v)\mu \\ \frac{\mathrm{d}\pi_M(t)}{\mathrm{d}t} &= -\pi_M(t)\mu + \int_0^\infty p_{M-1}(t,v)f(v)\mathrm{d}v\end{aligned}$$

subject to the following boundary conditions:

$$\begin{aligned}p_0(t,0) &= 0 \\ p_n(t,0) &= \int_0^\infty p_{n-1}(t,v)f(v)\mathrm{d}v, \quad 1 \le n \le M-1\backslash\{L\} \\ p_L(t,0) &= \int_0^\infty p_{L-1}(t,v)f(v)\mathrm{d}v + \pi_{2M-L-1}(t)\mu\end{aligned}$$

where $p_n(t,v)$ is the instantaneous rate function of $\Pi_n(t,v)$, the corresponding cumulative distribution of $\pi_n(t,v)$, and $f(v)$ is the probability density function of the inter-arrival time delay v. As $t \to \infty$, the system reaches steady-state and the behaviour does not depend any more on the time. Hence, the state equations become as follows:

$$\frac{\mathrm{d}p_0(v)}{\mathrm{d}v} = p_1(v)\mu \tag{23}$$

$$\frac{\mathrm{d}p_n(v)}{\mathrm{d}v} = -p_n(v)\mu + p_{n+1}(v)\mu, \quad 1 \le n \le M-2 \tag{24}$$

$$\frac{\mathrm{d}p_{M-1}(v)}{\mathrm{d}v} = -p_{M-1}(v)\mu \tag{25}$$

$$0 = -\pi_M\mu + \int_0^\infty p_{M-1}(v)f(v)\mathrm{d}v \tag{26}$$

$$0 = -\pi_n\mu + \pi_{n-1}\mu, \quad M+1 \le n \le 2M-L-1 \tag{27}$$

subject to the following boundary conditions:

$$p_0(0) = 0 \tag{28}$$

$$p_n(0) = \int_0^\infty p_{n-1}(v)f(v)\mathrm{d}v, \quad 1 \le n \le M-1\backslash\{N\} \tag{29}$$

$$p_L(0) = \int_0^\infty p_{L-1}(v)f(v)\mathrm{d}v + \pi_{2M-L-1}\mu. \tag{30}$$

The Equations (23-30) describes the dynamics of the SUT. The probability distribution of the states are:

$$\pi_n = \int_0^\infty p_n(v)\bar{F}(v)\mathrm{d}v, \quad 0 \le n \le M-1 \tag{31}$$

where $\bar{F}(v)$ is the complementary cumulative distribution function of v. The solution of Equations (23-31) with the normalization equation, $\sum_{n=0}^{2M-L-1}\pi_n = 1$, gives the stationary distribution of the playback buffer length. We are only interested in π_0.