Methodologies for Evaluating User Centric Performance of Mobile Networked Applications

by

Mustafa Mohammed Abduljabbar Al-tekreeti

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Luiz Fernando Capretz Professor, Dept. of Electrical and Computer Engineering, Western University
Supervisor:	Kshirasagar Naik Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Fakhreddine Karray Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	Ali Elkamel Professor, Dept. of Chemical Engineering, University of Waterloo
Other Member(s):	Mohamed-Yahia Dabbagh Lecturer, Dept. of Electrical and Computer Engineering, University of Waterloo

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Performance is an important attribute of mobile software applications, having a direct impact on end-user's experience. One of the obstacles that make software performance testing difficult to pursue is the lack of performance requirements that complicates the process of verifying the correctness of the test case output. Moreover, compared to other platforms, mobile applications' quality assurance is more challenging, since their functionality is affected by the surrounding environment. In this work, we propose methodologies and frameworks to evaluate the impact of interaction of the quality of the wireless network connection and application configurations on performance behaviour and performance robustness of a mobile networked application as perceived by the end user. We follow a model-based approach.

The thesis starts by defining the system model of software applications that we target, the network stack that the application is assumed to use to provide the service to the end user, and the metric used to capture the quality of the provided network service. Then, an analytical performance model that captures the application-network interactions is developed using the Markovian framework. To model realistic interactions with the network, the performance model is developed and solved using supplementary variable technique (SVT). The model is intensively verified with simulation.

Furthermore, two input network models are analytically developed. In both models, the mobile application is assumed to have a wireless network access through a WiFi access point that implements IEEE 802.11 protocol. In the first model, data transfer is achieved using user datagram protocol (UDP), while in the second, data transfer is accomplished using transmission control protocol (TCP). For the TCP model, two scenarios are considered. In the first scenario, an application data unit (APDU) is assumed to fit in one TCP packet, while in the second scenario, an APDU is assumed to fit in multiple TCP packets. All models are verified using the well-known NS2 network simulator.

Third, we propose a model based test generation methodology to evaluate the impact of the interaction of the environment, the wireless network, and the application configurations on the performance of a mobile networked application. The methodology requires four artefacts as inputs, namely, a behaviour model of the software under test, a network model, a test coverage criterion, and a set of desired performance levels. The methodology consists of three steps: performance model development, test generation, and estimation of test execution parameters. To evaluate the end-user quality of experience, test generation is formulated as an inversion problem and solved as an optimization problem. To generate an efficient set of test cases, two test coverage criteria are proposed: user experience (UE)

and user experience and input interaction (UEII). Test execution optimizations are inferred using a performance simulation model. To show the applicability of the methodology, two mobile networked app examples are used: multimedia streaming and web browsing. The effectiveness of the methodology is evaluated by comparing the time cost to design a test suite with random testing. The obtained results are very promising.

Fourth, to minimize the incurred cost of performance model evaluations, we utilize metamorphic testing to generate test cases. Metamorphic testing is a technique that is proposed to alleviate the test oracle problem. By utilizing certain inherent properties of the system under test (metamorphic relations), test cases are generated and verified without the need to know the expected output of each individual test case in advance. By hybridizing our proposed test generation methodology with metamorphic testing, the time cost of generating a test suite is reduced tremendously. We first generate a limited set of seed test cases using our test generation methodology. Then, we generate a set of follow-up test cases by utilizing the developed network models as metamorphic relations and without the need to invoke the performance model. Follow-up test generation is formulated as a maximization problem. The objective is to maximize the distance between a seed test case and follow-up test cases so that to generate a non-redundant set of test cases. Three distance metrics are used: Euclidean, squared Euclidean, and Manhattan. The modified methodology is used to generate test cases for a multimedia streaming application. We empirically evaluate the modified test generation methodology using two evaluation metrics: the incurred time cost and the percentage of redundancy in the generated test suite. The obtained results show the advantage of the modified methodology in minimizing the cost of test generation process.

Fifth, we propose a third methodology to evaluate the impact of the wireless network conditions on robustness of performance of adaptive and non-adaptive mobile networked applications. Software robustness is mainly about how the system behaves under stressful conditions. In this work, we target performance robustness under stressful network conditions. The proposed methodology consists of three steps and it requires three different artefacts as inputs. To quantify robustness, two metrics (static and dynamic robustness) are proposed. The main challenge in evaluating robustness is the combinatorial growth of network-application interactions that need to be evaluated. To mitigate this issue, we propose an algorithm to limit the number of interactions, utilizing the monotonicity property of the performance model. To evaluate the dynamic robustness metric, the ability of the adaptive application to tolerate degraded network conditions has to be evaluated. This problem is formulated as a minimization problem. The methodology is used to evaluate the performance robustness of a mobile multimedia streaming application. The effectiveness of the proposed methodology is evaluated. The obtained results show three to five

times reduction in total cost compared to the naive approach in which all combinations are exhaustively evaluated.

Acknowledgements

My thanks and praise first and foremost goes to Almighty God, the most merciful and compassionate, for giving me the knowledge, opportunity, and strength during the journey of this work and in all my life.

Along the way, several people deserve sincere recognition. I am most grateful to my advisor, Professor Kshirasagar Naik for his incomparable guidance and patience during the work. He had been a great mentor and friend. Over the years, he had always been available to guide me in the right direction. His valuable advice, generous help and time, and constant support were the secret of achieving all this work.

I would like also to thank Dr. Atef Abdrabou for his continuous guidance and helpful discussion during the development of the network models. His knowledge and experience in queueing theory and wireless network modelling had absolutely shortened the time and the efforts to finish this important part of the work.

I would like also to extend my appreciation and thanks to my thesis committee members, Professor Ali Elkamel, Professor Fakhreddine Karray, Dr. Mohamed-Yahia Dabbagh, and Professor Luiz Fernando Capretz for sparing their time to review this research work.

I would like to acknowledge the Ministry of Higher Education and Scientific Research of Iraq for financially supporting this research work. Thanks to this generous fund, this thesis was made possible.

I would like to thank my family for their continuous and unconditional support for all my endeavours. My father, Mohammed Al-tekreeti, my mother, Zuhal Abdulrazzaq, and my brothers, Mohaimen and Maytham, have been the greatest supporters in my life. I am grateful to them for their unconditional love, support, encouragement and the sacrifices they made to help me.

Cordial thanks go to my beloved wife, Ghadah, for the endless understanding, support, patience and care she showed during this critical period in my career. Despite being extremely busy, she has always been available to uplift me when I felt down and to congratulate me when I achieved. I am by all means indebted to her.

Finally, I thank all my teachers, colleagues and friends who helped and supported me in different ways throughout this work. God bless you all!

Dedication

This thesis is dedicated to my dearest children: Yousif, Abdullah, Anas, Abdulrahman, and Ibraheem,

I love you all

Table of Contents

List of Tables

List of Figures

Abbreviations

ACK	MAC Acknowledgement message frame
ACPs	Application Configuration Parameters
AP	Access Point
APDU	Application Data Unit
APIs	Application Programming Interfaces
CDF	Cumulative Distribution Function
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear-To-Send MAC message frame
DATA	MAC DATA message frame
DCF	Distributed Coordination Function
DIFS	Distributed Inter-Frame Space
DTMC	Discrete Time Markov Chain
ET	Error Type
FEF	Fault-Error-Failure
FELF	Fault-Error-Low quality of experience-Failure
FT	Fault Type
HWM	High Watermark
IEEE	Institute of Electrical and Electronics Engineers
IETCs	Invalid Executable Test Configurations
INTCs	Invalid Non-executable Test Configurations
IOLTS	Input Output Labelled Transition Systems

LOE	Low quality Of Experience
LWM	Low Watermark
MAC	Medium Access Control
MBT	Model Based Testing
NFC	Near Field Communication
NOPs	Network Operating Parameters
NS2	Network Simulator 2
PASTA	Poisson Arrivals See Time Averages
PCF	Point Coordination Function
PDUs	Protocol Data Units
Pert	function that realizes the follow-up test generation to satisfy UEII metric
PHD	Phase Type Distribution
QOE	Quality Of Experience
REQ	Request
RES	Response
RTS	Request-To-Send MAC message frame
SIFS	Short Inter-Frame Space
SUT	Software Under Test
SVT	Supplementary Variable Technique
TCP	Transmission Control Protocol
TEPs	Test Execution Parameters
TTCN-3	Testing and Test Control Notation version 3
UDP	User Datagram Protocol
UE	User Experience
UEII	User Experience and Input Interaction
UML	Unified modelling language
VNTCs	Valid but Not Useful Test Configurations
VTCs	Valid and useful Test Configurations

List of Symbols

$B(t)$	the playback buffer length at time t
B^{tcp}	the TCP buffer size at the end user
B	the playback buffer size
C_s^{APDU}	the coefficient of variation of APDU service time at the AP
C_B	the back-off counter
$CI(T_x)$	the confidence interval of the statistic T_x
C_s	the coefficient of variation of the service time
C_p	the number of parallel TCP connections
C_r	the coefficient of variation of packet response time
C	the chosen coverage criterion
D	the data rate
E_r^{APDU}	the mean APDU response time at the AP
$E[X]$	the expected value of the random variable X
E_r^f	the mean of response time for the test case f
E_r	the mean UDP packet response time at the AP
E_r^{tcp}	the mean TCP packet response time at the AP
$\bar{F}(v)$	a complementary cumulative distribution function of an arrival process
$\hat{F}_s$	the estimated mean of the multimedia file size
G	the number of subsets of S_{INP}
H_{MAC}	the MAC layer header size
H_{TCP}^{ack}	the TCP ACK packet header size
H_{TCP}	the TCP packet header size
H_{UDP}	the UDP packet header size
I_a	the set of constraints imposed by the application behaviour model
I_n	the set of constraints imposed by the network model
I	the set of constraints imposed by the application and the network
K	the size of APDU in TCP packets

L	the low watermark level of the playback buffer
M	the high watermark level of the playback buffer
N_t	maximum number of trails to generate a non-redundant test case
N	the number of mobile users connected to AP
P_{ack}	the TCP ACK packet payload size
$\Pi_n(t, v)$	the cumulative distribution function of $\pi_n(t, v)$
P_j	the probability distribution of the burst size at the AP
P	the UDP/TCP packet payload size
Q^{APDU}	the mean buffer length in terms of APDU at the AP
Q_L	the mean buffer length at the AP
Q	the number of test cases per performance region
R_a	region of acceptable performance
R_f	region of failed performance
R	number of QOE categories (performance regions)
S_1	the set of INTCs
S_2	the set of IETCs
S_3	the set of VNTCs
S_4	the set of VTCs
S_{ACP}	the set of application configuration parameters
S_C	the set of candidate application configurations
S_G	the super set of the g's subsets
S_{INP}	the set of ACPs and NOPs
S_{NOP}	the set of network operating parameters
S_N	the set of valid network scenarios
S_b	the batch service time
$\hat{S}_N$	the sorted set of valid network scenarios S_N
S_l	the set of desired performance levels
S_{tcp}	the total TCP packet service time
S_t	the UDP packet service time
S	the set of randomly generated test cases
T_{ACK}	the propagation time of ACK frame
T_{CTS}	the propagation time of CTS frame
T_{DIFS}	the DIFS time interval
T_j^F	the set of failed test cases for c_j application configuration
T_{PHY}	the time overhead of the physical layer
T_{RTS}	the propagation time of RTS frame
T_j^R	the set of robust test cases for c_j application configuration

T_{SIFS}	the SIFS time interval
T_S	the set of seed test cases
T^{udp}	the propagation time of the UDP packet
T^{ack}	the TCP ACK packet propagation time
T_{ij}	the set of test cases generated using the s_i seed test case for subset g_j
T_q	the packet mean waiting time at the AP
T_{rtt}	the packet round trip time
T_s^{ack}	the TCP ACK packet transmission time
T_s^{tcp}	the TCP packet transmission time
T_{st}	the total service time at the AP
T_s^{udp}	the UDP packet transmission time
T^{tcp}	the TCP packet propagation time
$\hat{T}_x$	the estimated mean of test case execution time
T	the test suite
U_b	the number of browsing users currently connected to the AP
U	the back-off time
$V(t)$	the elapsed time from last arrival to time t
V_{ard}^{APDU}	the variance of APDU response time at the AP
VP_i	the set of values of the ith parameter
V_{ard}^{f}	the variance of response time for the test case f
V_{ard}	the variance of packet response time at the AP
V_{ard}^{tcp}	the variance of TCP packet response time at the AP
W_R	the whole performance spectrum
W_b	the web page size in TCP packets
W	the contention window size
$Y\%$	the percentage of redundancy in the generated test cases
Z	number of consecutive TCP packets at the AP (burst size) that belong to the same end user
α_n	the mean service rate at the end user
α	the mean packet service rate at the access point
c_j	a valid candidate application configuration $c_j \in S_C$
ϵ	error margin to identify equivalent test cases
$f(v)$	the probability density function of the arrival process
γ	the relative error in the confidence interval
g_i	the ith subset of S_{INP}
k	number of exponential random variables in an Erlang random variable
λ_{AP}	the total mean rate of TCP packets at AP

$\lambda(v)$	the mean of arrival rate as a function of the supplementary variable v
λ	the mean packet arrival rate at the AP per user
l_c	the performance level associated with the configuration t_c
l^f	the performance level associated with a follow-up test case
l_j	a performance level of the jth test case
l^s	the performance level associated with a seed test case
l_{th}	the threshold performance level
μ	the mean of play rate
m	the number of application configuration parameters
n_k	a valid network scenario $n_k \in S_N$
n	the number of network operating parameters
$o(\Delta t)$	the probability of having more than one event in Δt
ω	the mean of the rate of an exponential random variable
$p_n(t, v)$	the instantaneous rate function of $\Pi_n(t, v)$
$\pi_n(t, v)$	the joint probability density function of being in state n (having n frames/packets) at time t and the time since last arrival is v
π_n	the stationary probability density function of being in state n
p_i	an arbitrary parameter of ACPs and NOPs
p	weight of an Erlang random variable in a Hyper-Erlang random variable
ρ_j^d	the dynamic robustness of c_j application configuration
ρ^d	the overall dynamic robustness of the application
ρ_n	the traffic intensity at the end user
ρ_j^s	the static robustness of c_j application configuration
ρ	the traffic intensity at the access point
r_i	the ith performance region that corresponds to the ith QOE category
σ	the time of the slot
s_i	the ith seed test case
$std(S_t)$	the standard deviation of S_t
t_j	an arbitrary test case
vp_i^f	the value of the parameter p_i in a follow-up test case
vp_i^j	the value of the ith parameter in the jth test case
vp_i^s	the value of the parameter p_i in a seed test case
ζ	the significant level

Chapter 1

Introduction

1.1 Motivation and Challenges

Performance is one of the important non-functional properties of software systems, having a vital impact on the user's experience. Typically, response time and throughput are the main figures of merit used to analyse software performance. Nowadays, however, new metrics are taken into account, such as energy and wireless bandwidth [5]. From small scale systems, such as smartphones, to large scale systems, such as data centres and servers, power consumption is a key performance indicator. Additionally, performance is arguably the most important non-functional property that affects software architecture [7]. Therefore, there is a growing interest in software systems' performance modelling, testing, and analysis [92], [117].

One of the obstacles that make software performance testing difficult to pursue is the lack of performance requirements [116]. Software testing, by definition, is the process of verifying the provided software services against their specifications. This is in part due to that the main focus in the development process being on how to make the software run, leaving very limited time for the performance at the end of the process [131]. In addition, performance requirements are sometimes hard to describe. They often depend on non-deterministic factors that are mostly out of the control of the software system, such as, workload size, user behaviour, and network condition [34], [106], [117]. Consequently, missing the performance requirements complicates the process of verifying the correctness of the test case output. This problem is well known in software testing as the lack of required test oracles [15].

With the proliferation of hand-held devices, real-time mobile applications such as voice-over-ip, streaming, video conferencing, and on-line gaming [121] have dominated over their desktop counterparts, in which end-user experience is mainly affected by the quality of the wireless connection. In addition, the main theme of hand-held devices is being context sensitive [42, 86], imposing extra requirements on mobile software development. Being able to communicate with many network types ranging from Near Field Communication (NFC) to global communication using cellular networks necessitates testing whether the application would perform as required under different environmental and contextual scenarios [45]. Another important aspect of mobile devices 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 quality of end-user experience [91].

Software robustness is another important property that directly relates to the end user experience. According to the IEEE standard, software robustness is defined as *the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions* [35]. The definition indicates two situations in which robustness of the software has to be investigated: invalid inputs and stressful environmental conditions. In literature, software robustness has mainly been evaluated from the correctness point of view using invalid or erroneous inputs. Less attention has been given to develop frameworks and methodologies to evaluate the impact of the environment on robustness of non-functional properties, such as performance [60]. This may be due to that the process of developing environmental models requires multidisciplinary expertise.

To address these challenges, we adopt a system level black-box model-based testing (MBT) approach [2, 111, 104]. In few words, *model-based testing is a variant of testing that relies on explicit behaviour models that encode the intended behaviours of software under test (SUT) and/or the behaviour of its environment. It encompasses the processes and techniques for the automatic derivation of abstract test cases from abstract models, the generation of concrete tests from abstract tests, and the manual or automated execution of the resulting concrete test cases* [111]. Model based testing has successfully been applied in industry [18]. As software applications become more complex, designing test suites from the code level after implementation is not always feasible. Abstracting the software behaviour and/or the environment affecting this behaviour has many benefits for the whole development cycle. It allows to focus on certain aspects of the system and neglect the less important ones (divide and conquer) [2]. Another advantage of MBT is the automation of test case generation. Moreover, as the developed model usually represents the required behaviour, it helps in developing and validating the (missing) performance requirements, and thereby creating the required test oracles [111]. In addition, model-based approach facilitates the systematic incorporation of the environment in the testing process [104].

1.2 The Performance Attribute of Software Systems

In this work, we assume that the software system under test (SUT) has intensively been tested, and it is correct from the functionality point of view. Normally, testing for software non-functional properties, such as performance, is carried out after ensuring that the SUT is functioning properly. This assumption is fairly practical, since evaluating the performance of a malfunctioned system does not make any sense. To understand the implication of this assumption, we define the relationship between software functionality and performance. We propose an integrated model that captures both software properties.

In software testing literature, the terms *fault*, *error*, and *failure* are well defined [9], [8]:

- a *failure* occurs whenever the external behaviour of a system does not conform to that prescribed by the system specification;
- an *error* is a state of the system which, in the absence of any corrective action by the system, could lead to a failure which would not be attributed to any event subsequent to the error; and
- a *fault* is the adjudged cause of an error.

A fault might be introduced in the requirements, design, and/or implementation. An error represents an internal, non-observable state that the program may pass through when the fault is executed. Typically, a program's internal state is defined by the local and global data entities, such as variables, objects, and flags that the program use. Figure 1.1 depicts the fault, error, and failure chain, where the arrows signify a cause-effect relationship between the corresponding concepts. In this work, we call this model the Fault-Error-Failure (FEF) model.

In the FEF model, only software functionality is taken into account, whereas performance aspects are ignored. In most software domains, problems are more related to non-functional attributes than to software functional correctness [34]. Many cases have been reported in which a performance degradation is perceived in the provided service while the application continues to work without producing incorrect responses [116], or the application takes very long time to fail [88]. These scenarios are not captured by the FEF model.

To account for such scenarios, we propose a new model to capture both software properties: performance and functionality. Our new model is called F(Fault)-E(Error)-L(Low Quality of Experience)-F(Failure) model, abbreviated as the FELF model. In this model,

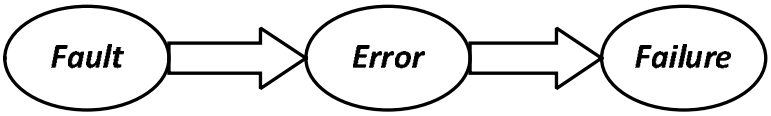

Figure 1.1: Fault-Error-Failure model.

triggering a fault may cause observable performance degradation before a software failure. Hence, a fault may cause the program to enter an error state that may lead to performance degradation that, in turn, may lead to a failure. For the end user, this performance degradation can negatively impact his experience; thus, we name this state as a Low Quality of Experience (LOE) state. Consequently, the software system can encounter three different types of faults:

- Fault Type 1 (FT1): faults that cause failure in functionality only;
- Fault Type 2 (FT2): faults that cause failure in functionality and performance; and
- Fault Type 3 (FT3): faults that cause failure in performance only.

Figure 1.2 shows the proposed FELF model. An FT1 fault represents the situation where the end-user does not perceive any performance degradation before the failure. FT1 faults correspond to the faults captured by the old FEF model. Thus, the FEF model is a special case of the FELF model. An FT2 fault may lead to an error state that may cause a performance degradation perceived by the end-user as an LOE. The LOE, in turn, may lead to a failure. This performance degradation can be considered as an early *observable* symptom of the failure. Therefore, the failure is in both software properties: functionality and performance.

In case of FT3, the fault may lead to an error state that, in turn, may lead to performance degradation only. The end-user observes this performance degradation while the system continues to work properly. Therefore, the failure is on performance only. We conjecture that among the faults that may lead to this scenario is the type of faults that occurs in the environment, such as a network service with degraded quality. Figure 1.3 depicts the three types of faults from the view point of a program's states; the Venn diagram shows the relationship between the error states due to the three types of faults in one software system. As shown, ET2 represents the overlapped area between ET1 and ET3, since it represents the error states that lead to a failure in functionality and performance. Thus, the assumption of having a functionally correct system means that any performance degradation that is perceived by the end user would be due to a fault of type FT2 or FT3.

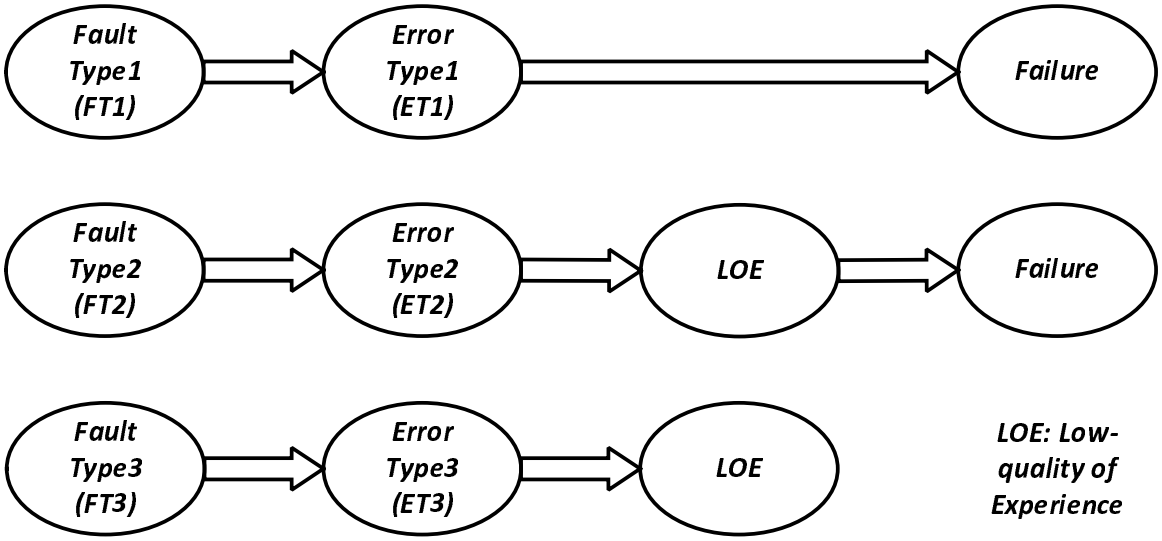

Figure 1.2: The proposed Fault-Error-LOE-Failure model.

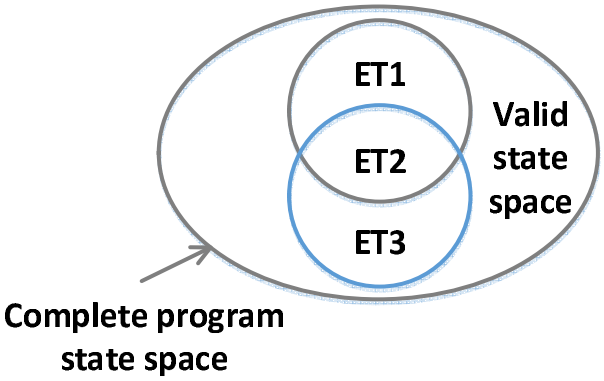

Figure 1.3: Program execution state space.

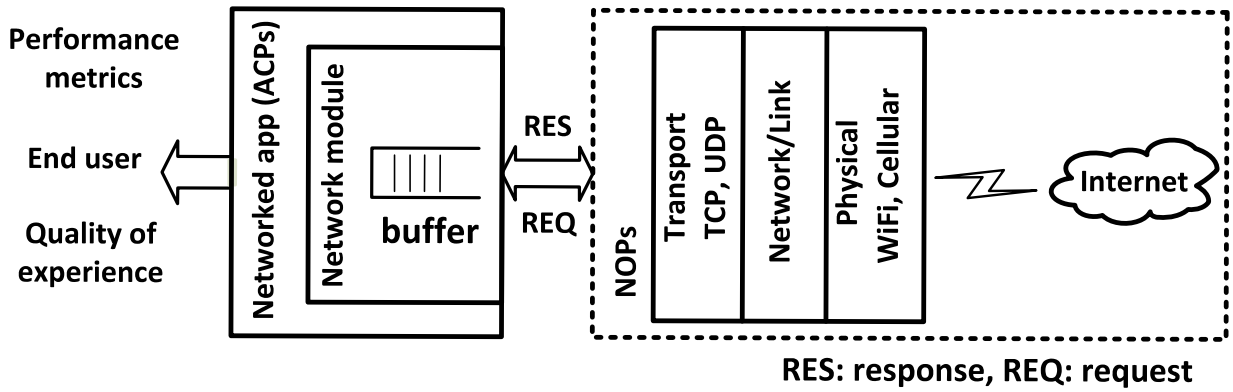

Figure 1.4: The system model of the application under test.

1.3 Problem Formulation

In this work, we focus on mobile networked applications. Figure 1.4 depicts the main elements of the system model. The application is assumed to be configurable by a set of parameters called application configuration parameters (ACPs). ACPs represent a set of configuration settings that have an impact on the performance behaviour, such as the size of the receiving buffer. Also, the quality of the network service is assumed to be captured by a set of parameters called network operating parameters (NOPs), such as data rate. NOPs can be identified at different layers along the network stack. They mainly depend on the wireless communication technology. In this work, we assume that the mobile application has network access through a WiFi access point that implements IEEE 802.11 protocol standard. Furthermore, data transfer to and from the mobile device is assumed to be achieved using either transmission control protocol (TCP) or user datagram protocol (UDP). The objective is to evaluate the interaction of network operating parameters (NOPs) and application configuration parameters (ACPs) on the end user experience.

Furthermore, we consider waiting time as a metric that captures the quality of the provided network service. For real-time applications, it has been reported that the end user experience is mainly affected by the amount of waiting time delay [65]. Waiting time can be during software service consumption as in streaming, voice-over-ip, and video conferencing, or before service consumption as in web browsing, file transfer applications, and also streaming. Accordingly, we divide mobile networked applications into the following two groups:

- **Group I:** this group represents applications where end user experience is mainly

affected by the waiting time during service consumption; and

- **Group II:** this group represents the applications where end user experience is affected by the waiting time delay before service consumption.

For both application groups, the interaction between the application and the network is modelled through a basic *Request-Response* mechanism as shown in Figure 1.4. Thus, the network's impact can be captured by modelling the RES inter-arrival time delay, which is modelled as a random variable. For applications of the first group, the performance behaviour is mainly dependent on the buffering behaviour, while the performance behaviour of applications of the second group does not depend on buffering. Nevertheless, we believe this model fits both types of mobile applications, as a buffer with zero size is a special case of this model.

To achieve the objectives of this work, we mainly follow the well-established software testing theories and paradigms. At this point, we need to introduce some notations. The sets of NOPs and ACPs are denoted as S_{NOP} and S_{ACP}, respectively. Thus, the input parameter set of the SUT is $S_{INP} = S_{NOP} \cup S_{ACP}$. The cardinality numbers of S_{NOP} and S_{ACP} are n and m, respectively. Therefore, we have $n + m$ parameters $p_1, p_2, ..., p_{n+m}$. A test case or a test configuration t_j is basically a set of values (vp_1^j, vp_2^j, ..., vp_i^j, ..., vp_{n+m}^j, l_j) where $vp_i^j \in Vp_i$, the set of permissible values of the parameter p_i, $1 \leq i \leq n + m$, and l_j is the expected performance level of the test case t_j.

1.4 Research Objectives and Contributions

In this work, three model based methodologies are proposed. The first two methodologies are proposed to generate test cases to evaluate the impact of the interaction of the wireless network and application configurations on the performance behaviour of mobile networked applications. The difference between the two methodologies is that the second methodology is proposed for applications with computationally expensive performance models. The third methodology is proposed to evaluate the impact of the interaction of wireless network and application configurations on performance robustness of mobile networked applications. In all methodologies, the performance behaviour and performance robustness are evaluated from the experience of end user point of view. In the remaining part of this section, we briefly describe the research objectives and we enumerate the main contributions of the work.

1.4.1 The proposed test generation methodology

In the first Methodology, a model-based test generation framework is proposed to evaluate the impact of the interaction of ACPs and NOPs on the performance behaviour of mobile networked applications. 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 a set of desired performance levels. The methodology consists of three steps. First, two performance models are developed: analytical and simulation. Second, to evaluate the end-user quality of experience (QOE), test generation is formulated as an inversion problem and solved as an optimization problem. An inversion problem is the problem of inferring the inputs, given the output of the system. Third, the necessary information to execute test cases (TEPs) is inferred using the simulation performance model.

To enhance the quality of the generated test cases, two test coverage criteria are proposed: user-experience (UE) and user-experience-and-input-interaction (UEII). The objective of UE criterion is to generate test cases to fully cover the whole spectrum of the performance behaviour, while the objective of the UEII criterion is to cover both the performance behaviour and the input parameters' interactions at the same time. Because the network model constrains NOPs by non-linear constraints, state of art combinatorial test generation tools are not applicable directly. To overcome this problem, we constrain the input space first. Then, we generate test cases to satisfy UEII criterion. The generated test cases are validated using the performance simulation model.

Using two application examples of mobile multimedia streaming (group I) and web browsing (group II), we explain the different steps of the proposed methodology. We evaluate the effectiveness of the methodology by comparing the estimated time cost to generate a test suite with random testing. The obtained results acknowledge that casting test generation as an optimization problem is more cost effective than random testing.

1.4.2 The modified test generation methodology

In model-based performance testing, the core component is the performance model. However, whether it is simulation or analytical, performance models are computationally expensive especially for software applications of Group I. Therefore, in the second methodology, we combine our first test generation methodology with metamorphic testing to minimize the incurred cost of test generation. From now onward, we call the first methodology as the proposed test generation methodology and the second methodology as the modified test generation methodology.

Metamorphic testing is a technique that is proposed to alleviate the test oracle problem. By utilizing certain inherent properties of the system under test and given a set of seed test cases, follow-up test cases are generated without the need to invoke expensive performance models. These properties are normally called metamorphic properties. In general, seed test cases can be generated using any test generation technique. However, in this thesis, we generate seed test cases using our proposed test generation methodology because test seeds have special requirements.

In this methodology, test oracles are considered available but expensive due to the high cost of performance model evaluation. We develop metamorphic relations using the less complicated network models. We show that metamorphic properties that are developed using network input models are of two types: productive and non-productive. We formulate follow-up test generation as a maximization problem. We employ three different distance metrics as a fitness function. We maximize the distance between a seed test case and a follow-up test case to generate non-redundant test cases. The applicability of the approach is demonstrated using a multimedia streaming application example. We propose the notion of redundancy to measure the quality of the generated test suite. The effectiveness of the modified methodology is evaluated in comparison with the original proposed methodology using two evaluation metrics: the incurred time cost and the percentage of redundancy in the generated test suite.

1.4.3 Performance robustness evaluation

The second main objective of this thesis is to design a methodology to evaluate the impact of the interaction of ACPs and NOPs on performance robustness of mobile networked applications. We consider two types of applications: adaptive and non-adaptive. The objective is to enable conducting robustness analysis early in the development process. To capture robustness, two metrics are proposed. The first metric, static robustness, is proposed to quantify the robustness of non-adaptive mobile applications. The second metric, dynamic robustness, is proposed to quantify the robustness of adaptive mobile applications.

The methodology requires the availability of three models as inputs: the network model, the behaviour model of the application, and the performance model. The methodology consists of three main steps. First, all valid network conditions and candidate application configurations are determined. Second, the network-application interactions are evaluated using the performance model. Third, the corresponding robustness metric is quantified. For adaptive applications, the evaluation of dynamic robustness metric needs the evaluation of

the ability of the application to adapt in case of a degraded network condition. The process of figuring out new ACPs that can tolerate the degraded network service is formulated as a minimization problem.

However, to evaluate the impact of the wireless network conditions on the performance robustness, all application and network interactions have to be enumerated and evaluated, which may leads to the well-known state explosion problem. To mitigate this issue, we propose an algorithm to limit the number of interactions, utilizing the monotonicity property of the performance model. A mobile multimedia streaming application is used to illustrate the proposed methodology. The effectiveness of the proposed methodology is evaluated in comparison with the naive approach in which all the network-application interactions are exhaustively evaluated. The obtained results are very promising.

1.4.4 Performance model development

The core component of any model based performance testing and evaluation is the performance model. In software testing, performance models are necessary to provide the required test oracles, by which the output of each test case is verified. In this work, we divide mobile networked applications into two groups. The first group represents the type of applications in which the end user experience is mainly affected by the time delay while the application is running, while the second group represents the applications in which end user experience is mainly affected by time delay before the application starts. We develop analytical and simulation performance models for both types of applications. We show that only applications of the first group need a performance model to be built from scratch. For applications of the second group, the already developed model that captures the wireless network behaviours can be reused as a performance model. For the applications of the first group, the analytical model is developed using Markov chain. The model is solved using the supplementary variable technique (SVT), allowing for more realistic network scenarios to be considered in the process.

1.4.5 Wireless network model development

Another important component of this work is the network model. In this thesis, a mobile networked application is assumed to have wireless network access through a WiFi access point that implements IEEE 802.11 network protocol standard. We consider the two de facto Internet transport protocols: UDP and TCP. We develop three network models. In the first model, data transfer from the access point to the mobile device is assumed to be

achieved using the UDP transport protocol, while in the second and third models, data transfer is achieved using the TCP protocol.

In the first and second models, each application data unit (ADPU) is assumed to fit in one UDP or TCP packet, while in the third model, this assumption is relaxed; the APDU can fit in more than one TCP packet. In each model, two analytical expressions for the packet/APDU time delay are developed: mean and variance. Then, a probability distribution is matched for each scenario. The matched analytical cumulative distribution functions are validated with simulation using the NS2 network simulator.

1.5 Literature Review

1.5.1 Performance testing and evaluation

In general, software testing can be invoked at any step during the design process. It can be performed at the unit, integration, or system level. In literature, considerable efforts have been made on integrating performance analysis with the development life cycle. In this approach, different software design artefacts are augmented with performance values and then transformed into stochastic models, such as Petri nets, simulation models, or queueing networks. A comprehensive summary can be found in [78, 11]. The main objective in this research trend 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 to test the SUT after implementation. They assume that the performance requirements are available so that they can annotate the design diagrams with performance values, whereas in our work, performance requirements are elicited as one of the by-products of the model-based testing process. In fact, 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. In contrast, we target software applications 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 work and the model-based load testing. The developed models in load testing are network technology independent, while our approach models explicitly the network technology. In addition, test input for load testing can be generated simultaneously while the real SUT executes (on-line) or independently (off-line). In the off-line approach, test loads are designed from the source code using static analysis

techniques such as data flow analysis [122] and symbolic execution [129, 130], using the operational profile (workload characterization) [116, 10], or using design models annotated with statistics derived from the operational profile and past data, such as UML use-case diagrams [41, 37], Markov chains [47, 114, 93], and probabilistic time automata [1]. In the on-line approach, there is a feedback from the real system to the test generation process to refine the test loads. Different techniques are used to enhance the test loads, such as machine learning [54], linear programs [127, 128], Bayesian estimation [52], hill climbing [14, 13], and genetic algorithms [56, 44]. A comprehensive survey of load testing can be found in [68]. Our testing methodology can share some of the concepts with load testing, however, the approach and scope of our work are distinct. In fact, our approach is off-line, yet casting the test generation as an inversion problem resembles the feedback in on-line load testing.

1.5.2 Performance testing in mobile devices

In the mobile domain, because of the short development cycle compared to other software domains, most of the performance testing activities are conducted after implementations [72]. Another observation is that most testing frameworks follow a white-box approach due to many reasons. First, most applications are of small to moderate sizes (tens to hundreds of lines of code). Second, many mobile applications are developed by non-professional software developers which increases the probability of code level mistakes. Third, black-box testing needs special skills in order to develop different types of models. Although a white-box approach in mobile testing has its advantages, it does not facilitate a systematic consideration of the environment in the testing process [104].

In spite of the much research efforts going into performance testing in the mobile software domain, test generation to evaluate the network impact on app performance has not received much attention [86]. In this regard, we conducted a survey targeting any published work in test generation for networked (wired or wireless) software systems. We observed that the main focus is on testing for functional requirements. For example in [97], a model checking approach is used to verify UDP-based software systems. Since UDP protocol does not implement any mechanism to ensure data integrity, a tool is provided to check for faults in the software part that is responsible for the communication. The tool can introduce three types of errors in communication: packet duplication, packet loss, and packet reordering. In another work [113], networked software systems are checked for violations in functional specifications using fuzz testing and model refinement. In both, the main focus is on functional requirements.

In trying to enable mobile software developers to debug their applications for communication errors, a software profiler is provided in [45]. It analyses the communication link and generate metrics to capture different aspects, such as connectivity and power consumption. It also allows for doing measurements correlation to pinpoint the error. This work belongs to the efforts of conducting performance analysis early in the development process, while our objective is designing test suites to be used after implementation.

Providing test execution beds to evaluate the performance of networked applications is very important. In [96], an emulator based test bed is designed to evaluate the impact of physical mobility and hands-over in 4G networks on the operation of wireless mobile applications. It does not address how test scenarios are designed and what is the coverage criterion to use.

In terms of the end-user quality of experience (QOE), the work reported in [25] had developed a test bench to automate end-user QOE capturing for Android applications. Thus, it is meant to facilitate test execution, while in our work, we utilize QOE characterization to guide test selection process to evaluate the performance from the end-user point of view and to minimize the size of the test suites.

1.5.3 Combinatorial testing with constraints

In combinatorial testing literature, handling constraints while generating test inputs has been tackled using two approaches [74]. In the first approach, the parameters' combinations that should not appear in the test suite are first identified [82, 125]. Then, heuristics based algorithms are employed to generate test inputs. In the second approach, the forbidden tuples are first expressed as logical constraints [126, 50]. Then, the validity of the generated test input is checked by casting the problem as a Boolean satisfiability problem. In both approaches, the constraints and parameters are Boolean. The only work that considered numerical constraints is the work reported in [79]. In this work, linear numerical constraints are converted to binary constraints and test generation is formulated as a binary optimization problem. We are not aware of any published work that considers non-linear numerical constraints. In our work, we overcome the complexity of the constraints by constrain the input space before starting test generation. That is, we remove the forbidden combinations from the test candidates' pool.

1.5.4 Using simulation models in testing

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 [75, 12], or to design test cases to verify software systems represented as simulation models [20, 89]. In our work, test execution parameters are inferred from simulation models to concretize the generated abstract test cases using two univariate statistical procedures, while in literature, statistical techniques, such as hypothesis testing, are mainly used to verify the output of test cases generated from simulation models [57, 58].

1.5.5 Software robustness

In literature, software testing is the main approach used to evaluate software robustness [103]. However, robustness is mainly tackled from the functionality correctness point of view. The relation between robustness and other software qualities like testability and performance have not been addressed adequately [60]. Furthermore, robustness testing has been used to verify different types of systems, such as commercial off-the-shelf components, communication protocols, mission-critical systems, and operating systems [102]. It is mainly achieved by injecting the interface of the SUT with a sequence of faulty inputs with the intention to make the SUT crash or degrade. Invalid test inputs are generated using different techniques. In communication protocols, fault injection is realized by mutating protocol data units (PDUs) [119]. To facilitate test generation and maintenance, test cases are specified using TTCN-3 specification language [69, 118, 71, 70]. In addition, the authors in [124] have considered network protocols with time constraints. In our work, we preserve the correctness of the protocol syntactically and disturb another protocol property, namely, the packets timeliness to evaluate the performance robustness.

The main challenge in testing adaptive and configurable systems is the state explosion problem [105]. This issue has been tackled mainly using combinatorial interaction testing (CIT). For example, a trimming strategy using pair-wise interaction coverage metric is employed to reduce the number of possible faulty PDUs needed to test the robustness of communication protocols [112]. However, for systems with large state/configuration space, even pair-wise combinatorial metric does not scale up as in [61], where similarity based metrics are proposed to generate test cases. The main idea is that dissimilar test cases are more likely to trigger inefficiencies in the system. Furthermore, CIT metrics are computationally expensive. To mitigate this cost, similarity based heuristics are used in a search-based approach to generate test cases that fully or partially satisfy a given CIT metric [62]. In systems that have operational profiles, execution traces are clustered [38]

and benchmarks [24] are developed and used to limit the number of possible scenarios. In our work, CIT based metrics are not applicable, since the proposed robustness metrics require the classification of all network-app interactions. Also, similarity based metrics are not viable because dissimilar network configurations do not necessarily lead to different performance behaviours. In fact, due to model non-linearity, there exist network scenarios with different configuration settings, but their performance behaviours are statistically equivalent. In our work, we propose an algorithm to minimize the number of application-network interactions, utilizing the monotonicity property of the performance model.

In a more formal approach, the works in [110, 95] have proposed a framework to test real-time systems and embedded systems for robustness, respectively. In both works, the SUT is modelled using Input-Output-Labelled-Transition-Systems (IOLTS). To generate test inputs, hazards are inserted in SUT nominal specifications to model a hostile environment. Because operating systems are stateful, a framework is developed in which both valid and invalid inputs are used to evaluate the robustness of operating systems at different states [38]. The valid inputs are used to drive the SUT to the required state and then a sequence of invalid inputs is injected.

In addition, testing is used to verify the robustness of adaptive software systems. In reference [24], a robustness-driven resilience evaluation framework is proposed for self-adaptive software systems. It assumes that the SUT is designed using the Rainbow platform. In this architecture based platform, the system is composed of two main modules: the managing and managed modules. The invalid inputs are injected into the managing module (controller) through the probes that are employed to sense the state of the system. The framework assumes that the controller is stateful. The main focus is on functionality correctness.

Regarding robustness quantifications, a five-category based robustness classification scheme is proposed to judge the severity of the system failure [77]. It is proposed for large-scale systems, such as operating systems. In another work, robustness is quantified using a mutation score [16]. Both works evaluates robustness from the correctness stand point.

1.5.6 Metamorphic testing

Recently, two survey papers in metamorphic testing are published in two prestigious journals, showing the momentum that this technique is gaining as an effective approach to alleviate the test oracle problem and to automate testing activities [99, 31]. The efficacy of metamorphic testing in solving the test oracle problem was empirically demonstrated [85]. In literature, metamorphic testing has been used in different software domains, such

as numerical algorithms [28], machine learning [120], network simulators [32], query based systems [84], to name a few. However, the main focus is the functional properties of the software system. Using metamorphic testing for performance testing is still not explored [99, 100].

Two main points have been tremendously addressed in literature of metamorphic testing: derivation of metamorphic relations and the quality assurance of the derived relations in fulfilling the objective of the testing process. In most cases, metamorphic relations are derived manually and the main observation of this process is that it requires domain knowledge of the SUT. In reference [85], metamorphic relations that are derived by non-professional software developers (university students) had been empirically shown to be effective enough in triggering real software defects. Regarding the quality of the derived metamorphic relations, most of the studies utilize mutation analysis to show how powerful derived metamorphic relations in fault detection. In reference [98], metamorphic testing had been successfully used to detect previously unknown real faults in YouTube and Spotify web APIs. In efforts to automate the derivation of metamorphic relations, inference of program invariants technique was employed for regression testing [108]. Differences in metamorphic properties that are inferred across multiple software versions may indicate the existence of flaws in the SUT.

To rank metamorphic relations according to their capability in detecting failures, a dissimilarity index was proposed in reference [26]. The main idea is that follow-up test cases that have dissimilar execution profiles compared to the execution profile of the seed test case are more likely to detect failures. Using this index, metamorphic relations are prioritized, enabling software developers to use the most powerful metamorphic properties. Execution profiles are evaluated using two structural coverage metrics: statement coverage and branch coverage. Dissimilarity index is quantified using distance metrics. In another similar work, structural based coverage criteria are directly used to evaluate the quality of metamorphic relations [46].

Using metamorphic testing for non-functional properties is still an open question. In one of few attempts to utilize this technique, a framework was proposed to test software modules to be deployed in wireless sensors for both functional correctness and power efficiency [29]. Metamorphic relations are developed utilizing the idea that adjacent wireless sensors should behave in a similar way (program versioning). That is, cross-referencing is used among sensors that are close in proximity to alleviate the oracle problem. In a recent work, metamorphic testing is used successfully to test for cybersecurity [33]. It was used to detect security related issues in software obfuscation systems. In another work, a proof of concept is established for using metamorphic testing to test for performance [101]. Software examples and case studies from software product lines are used to show the feasibility of

the technique to detect performance bugs. In this thesis, we utilize metamorphic testing to minimize the incurred cost of performance model evaluations that are necessary to generate the required test oracles.

1.6 Thesis Outline

The remaining part of the thesis is organized as follows:

- In Chapter 2, we develop an analytical performance model for a group I representative application example, multimedia streaming, using the Markovian framework. First, the functional requirements of the application are defined. Then, the model is developed and solved using the supplementary variable technique (SVT). Last, the developed model is validated using simulations;

- In Chapter 3, three reusable input network models are analytically developed. First, the distribution coordination function (DCF) of the IEEE 802.11 standard is introduced. Second, analytical expressions for the mean and variance of packet/APDU inter-arrival time delay are derived assuming data transfer is achieved using both UDP and TCP protocols. Third, a probability distribution is matched for each scenario using the derived mean and variance expressions. Last, the matched probability distributions are empirically validated using the NS2 simulator tool;

- In Chapter 4, a model based test generation methodology is proposed. First, how test generation can be formulated as an inversion problem is explained. Second, methodology input requirements and steps are introduced. Third, Two application examples of multimedia streaming and web browsing are used to explain the methodology. At the end, methodology effectiveness is evaluated in comparison with random testing;

- In Chapter 5, the proposed test generation methodology of Chapter 4 is modified using metamorphic testing. First, the employed metamorphic relations are introduced. Then, input requirements, steps, and expected outputs of the modified methodology are explained. Third, the modified methodology is applied on an example of multimedia streaming application. Last, the effectiveness of the methodology is evaluated in comparison with our original test generation methodology;

- In Chapter 6, a model based methodology to evaluate performance robustness of adaptive and non-adaptive applications is proposed. First, two performance robustness metrics are suggested. Second, the proposed methodology to evaluate both

robustness metrics is introduced. Third, the proposed methodology is used to evaluate the performance robustness of a multimedia streaming application. Last, the effectiveness of the proposed methodology is evaluated; and

- In Chapter 7, the work is concluded and pointers for future works are provided.

Chapter 2

The Performance Model

As explained in Chapter 1, the performance model should capture the impact of the waiting time delay on experience of the end user. Accordingly, we classify mobile networked applications into two groups. For the first group, the performance metric under consideration that correlates with end user experience has a non-trivial relationship with the waiting time delay, while for the second group, the performance metric has a simple linear relationship with waiting time delay. In fact, there is no need for developing a performance model for the second group of applications as will be shown later. In Group II, the end user experience is directly captured by the network model, whereas in Group I, the performance behaviour depends on the buffering behaviour of the application. In other words, we can say that the performance model for applications of the second group is a special case of the performance model for applications of the first group.

Therefore, the main objective of this chapter is to develop an analytical performance model for applications of Group I. Without loss of generality, we choose multimedia streaming as a representative application for this group. We develop a performance model for this application, where the performance metric under consideration is the smoothness of streaming as seen by the end user. First, we define the behaviour model of the multimedia streaming application. Then, we develop an analytical performance model using the Markovian framework. At the end, we validate the model using simulations.

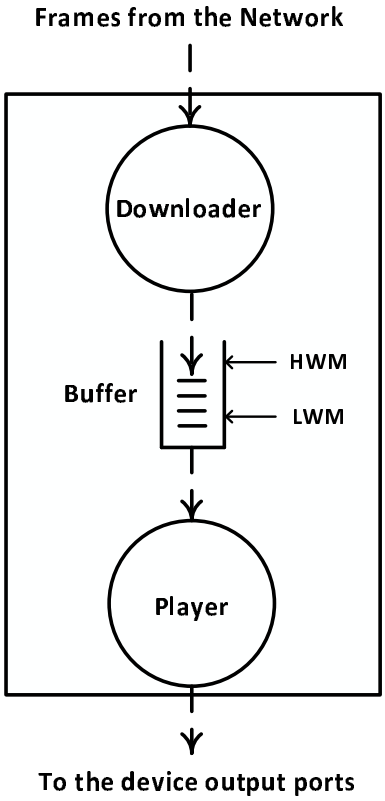

Figure 2.1: The general work flow of the multimedia streaming application. The abbreviations HWM and LWM stand for high watermark and low watermark levels, respectively.

2.1 Functional Requirements of The Multimedia Streaming Application

The dynamics of streaming applications are captured if the buffering behaviour is modelled. In this application example, we assume that the SUT implements a *progressive* streaming mechanism in which both frame downloading and decoding (playing) are interleaved [87]. The application behaviour is modelled by two main components: a *downloader* and a *player*. Both components interact with each other through a buffer known as a *playback buffer*. Figure 2.1 depicts the general work flow of the streaming application.

At the beginning, the application 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 application remains in this phase until the data level in the playback

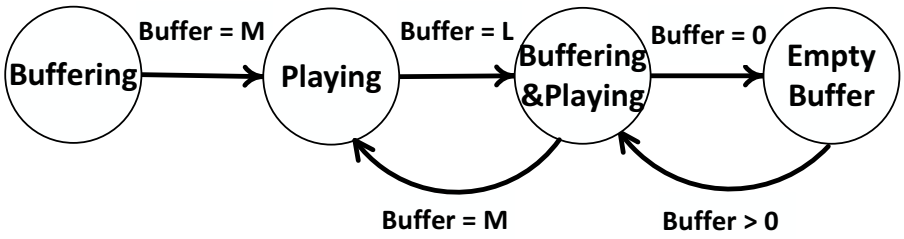

Figure 2.2: The behaviour model of the multimedia streaming application.

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 application 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 continuous playback. Figure 2.2 shows the state diagram that represents the desired behaviour of the SUT.

In multimedia streaming, many performance metrics are proposed to capture end-user experience. In this chapter, we consider *the frequency of rebuffering events* as the performance metric [90]. This metric positively correlates with the waiting time delay that the end user may experience. The SUT behaviour is characterised by three configuration parameters (ACPs): playback buffer size (B), high watermark (M), and low watermark (L). The "state" in this diagram represents the activity in which the software is currently running. The text on the arrow from state x to state y, represents the condition that triggers a transition from state x to state y.

2.2 The Considered Performance Metric

The performance metric under consideration has a direct relationship with how often the application is visiting the "Empty Buffer" state in Figure 2.2. Thus, the performance metric correlates with the fraction of time of being in the *Empty Buffer* state out of the total time of streaming. This behaviour is well modelled using the Markovian stochastic process. To make use of this approach, certain requirements should be fulfilled. We model the data level in the playback buffer as a stochastic process. The stationary distribution of the process corresponds to the steady-state distribution of the playback buffer length. We

are only interested in the probability of having zero frames in the playback buffer. Since the performance metric under consideration is a steady state metric, the buffering phase is not included in the performance model.

In general, to utilize the Markovian framework, the time delay between frame arrivals and the service (play) time of each frame should be exponentially distributed. In this work, applications are assumed to have network access via a wireless connection. Therefore, the time delay between arrivals cannot be assumed to be exponentially distributed. Furthermore, the frame play (service) time is not exponentially distributed in reality. In literature, both frame rate and frame size are assumed to be deterministic (constant) in order to simplify the analysis [83]. According to MPEG encoding scheme, frame size depends on the frame type which can be I, B, or P [49]. In addition, many techniques have been proposed to absorb network uncertainties by tuning the frame encoding rate [43]. Therefore, in this work, we assume the frame decoding (playing) rate is exponentially distributed. Nevertheless, the stochastic process that models the buffer length is still not Markovian, as the frame arrivals are not exponentially distributed.

When either arrival or service process is not Markovian, there are four choices to consider. The first and easiest one is to develop a discrete time Markov chain (DTMC) to approximate the stochastic process. Sometimes, DTMC does model very well the behaviour especially when the operating region is limited. Since our objective in developing the model is to design test suites that capture the whole behaviour spectrum, DTMC model did not give us a good approximation as compared to the simulated behaviour. The remaining choices are Markov chains with Phase-Type Distributions (PHD) [22], Semi-Markov processes [80], and Markov chains with Supplementary Variable Technique (SVT) [39], [40]. The main concept of PHD is to represent the non-exponential distribution by mixing certain number of exponential distributions. The main drawback of using this technique is the state explosion problem especially when the size of the Markov chain is large. In our case, the size of the Markov chain depends on the playback buffer size and the video frame size which can result in a chain with thousands of states. Also, fitting an empirical distribution using a PHD distribution is not always possible. This makes PHD approach is not practical for our application.

Mathematical modelling using Semi-Markov process is sometimes not straight forward. The main idea of this approach is to observe the stochastic process (playback buffer length) at certain time instants where the process has the memoryless property. Then, a DTMC is embedded at those prescribed time instants. If the arrival process is not Markovian, as in our case, we have to embed the DTMC at the time instants directly after frame departure events. If the service process is the non-Markovian one, a DTMC should be embedded at frame arrivals.

The next step is easier if the arrival process is Markovian not the service process which is the opposite of our case. If the arrival is Markovian, the PASTA (Poisson arrivals see time averages) principle is applied (the Poisson arrival sees the same buffer length distribution as a random observer [107]) which facilitates the derivation of the state probability. If the arrival process is not Markovian, there is a trick to obtain the probability of the state at any time by developing a duality relationship with another system of known parameter distributions as reported in [64], [73]. In fact, duality relationships are only available for special cases, and sometimes they are difficult to establish. In this work, we are trying to use a general framework to develop performance models that can be easily extrapolated to other scenarios. Hence, we choose the forth approach, performance modelling using the supplementary variable technique. In theory, this technique can be applied even when arrival and service processes are both non-Markovian [6].

Basically, in order to use Markovian modelling approach, the memoryless property should be preserved. That is, in Markov chain words, given the current state, the probability of the next state is independent from the history. This means, in order to determine the next process state all the needed information should be available in the current state, and how the process has reached the current state does not matter. In our model, the distribution of the frame inter-arrival time delay is not exponential. Thus, in order to determine the next state (buffer length), the elapsed time since the previous frame arrival (or the remaining time for the next frame arrival [63]) should be known, i.e., the evolution of the process does depend on the history.

To overcome this problem, the state of the Markov chain should be redefined so that all the needed information to determine the next state is available in the current state. This can be done if the stochastic process is defined as follows:

$$\{B(t), V(t), t \geq 0\} \tag{2.1}$$

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. Thus, the stochastic process underlying the performance behaviour of the software system is discrete-continuous in state space and continuous in time. Figure 2.3 shows the Markov chain of the application software. 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.

Now, our objective is to find the steady state probability distribution of the playback buffer length. The derivation depends on examining the short-term behaviour of the chain. Generally, in any continuous-time Markov chain, the probability of having one event in a

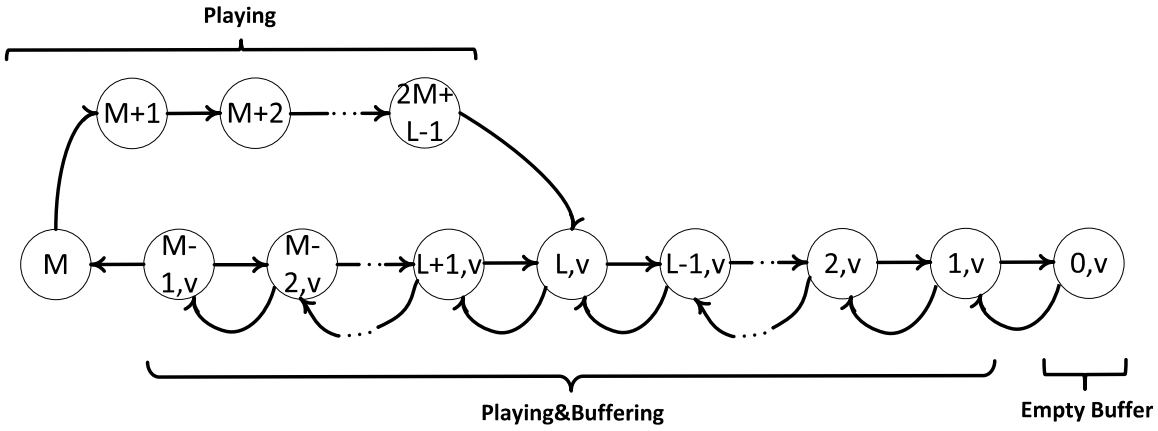

Figure 2.3: The state diagram of the application.

short time interval Δt is $q\Delta t$, where q represents the rate of events. The probability of not having an event in Δt is $1 - q\Delta t$. The probability of having more than one event in Δt is $o(\Delta t)$, where $o(\Delta t)$ tends to 0 when Δt approaches 0. Now, to get the dynamics of the stochastic process, a differential (or a partial differential) equation is derived for each state in Figure 2.3. The equation is formed following the very basic law of total probability of two mutually exclusive events. For state 0, the difference equation is:

$$\pi_0(t + \Delta t, v + \Delta t) = \pi_0(t, v)(1 - \lambda(v)\Delta t) + \pi_1(t, v)\mu\Delta t + o(\Delta t) \tag{2.2}$$

The physical meaning of Equation (2.2) is as follows: The probability of being in state 0 after Δt of time, $\pi_0(t + \Delta t, v + \Delta t)$, is equal to the probability of being originally in state 0 at time t and no arrival happens in the interval Δt, plus the probability of being originally in state 1 at time t and 1 frame is played in the interval Δt, where $\lambda(v)$ and μ are the arrival and service rates, respectively. Using the same concept, the states $1, 2, ...,$ and $M - 2$ have the same difference equation that is given by:

$$\begin{aligned} \pi_n(t + \Delta t, v + \Delta t) =& \pi_n(t, v)(1 - \lambda(v)\Delta t - \mu\Delta t) + \\ & \pi_{n+1}(t, v)\mu\Delta t + o(\Delta t), \quad 1 \leq n \leq M - 2 \end{aligned} \tag{2.3}$$

For the states $M - 1$ and M, the difference equations are:

$$\pi_{M-1}(t + \Delta t, v + \Delta t) = \pi_{M-1}(t, v)(1 - \lambda(v)\Delta t - \mu\Delta t) + o(\Delta t) \tag{2.4}$$

$$\pi_M(t + \Delta t) = \pi_M(t)(1 - \mu\Delta t) + \int_0^\infty \pi_{M-1}(t, v)\lambda(v)\Delta t \mathrm{d}v + o(\Delta t) \tag{2.5}$$

For the last $M-L-1$ states, the difference equations are also the same and it is given by:

$$\pi_n(t+\Delta t) = \pi_n(t)(1-\mu\Delta t) + \pi_{n-1}(t)\mu\Delta t + o(\Delta t), \quad M+1 \leq n \leq 2M-L-1 \tag{2.6}$$

To get the state differential equations, few algebraic manipulations are needed. For state 0, starting from Equation (2.2):

$$\pi_0(t+\Delta t, v+\Delta t) - \pi_0(t,v) = -\pi_0(t,v)\lambda(v)\Delta t + \pi_1(t,v)\mu\Delta t + o(\Delta t)$$

Add and subtract $\pi_0(t, v+\Delta t)$, divide both sides by Δt, and take the limit as $\Delta t \to 0$:

$$\begin{aligned}\Rightarrow \lim_{\Delta t\to 0} \frac{\pi_0(t+\Delta t, v+\Delta t) - \pi_0(t, v+\Delta t)}{\Delta t} + \lim_{\Delta t\to 0} \frac{\pi_0(t, v+\Delta t) - \pi_0(t,v)}{\Delta t} &= -\pi_0(t,v)\lambda(v) \\ &+ \pi_1(t,v)\mu + \lim_{\Delta t\to 0} \frac{o(\Delta t)}{\Delta t}\end{aligned}$$

As the last term evaluates to 0, the above equation settles down to the following partial differential equation (pde):

$$\frac{\partial \pi_0(t,v)}{\partial t} + \frac{\partial \pi_0(t,v)}{\partial v} = -\pi_0(t,v)\lambda(v) + \pi_1(t,v)\mu. \tag{2.7}$$

Using the same procedure, the differential and partial differential equations of the remaining states are obtained and they are listed below:

$$\frac{\partial \pi_n(t,v)}{\partial t} + \frac{\partial \pi_n(t,v)}{\partial v} = -\pi_n(t,v)(\lambda(v)+\mu) + \pi_{n+1}(t,v)\mu, \quad 1 \leq n \leq M-2 \tag{2.8}$$

$$\frac{\partial \pi_{M-1}(t,v)}{\partial t} + \frac{\partial \pi_{M-1}(t,v)}{\partial v} = -\pi_{M-1}(t,v)(\lambda(v)+\mu) \tag{2.9}$$

$$\frac{\mathrm{d}\pi_M(t)}{\mathrm{d}t} = -\pi_M(t)\mu + \int_0^\infty \pi_{M-1}(t,v)\lambda(v)\mathrm{d}v \tag{2.10}$$

$$\frac{\mathrm{d}\pi_n(t)}{\mathrm{d}t} = -\pi_n(t)\mu + \pi_{n-1}(t)\mu, \quad M+1 \leq n \leq 2M-L-1 \tag{2.11}$$

Subject to the following boundary conditions:

$$\begin{aligned}&\pi_0(t,0) = 0, \\ &\pi_n(t,0) = \int_0^\infty \pi_{n-1}(t,v)\lambda(v)\mathrm{d}v, \quad 1 \leq n \leq M-1\backslash\{L\}, \\ &\pi_L(t,0) = \int_0^\infty \pi_{L-1}(t,v)\lambda(v)\mathrm{d}v + \pi_{2M-L-1}(t)\mu.\end{aligned} \tag{2.12}$$

Because our objective is the steady-state analysis (not the transient), there is no need to specify the initial conditions, since stationary distribution is independent of the initial condition. To simplify the procedure of solving the above equations, there is a strategy to remove $\lambda(v)$ from the equations reported in [39].

Let $Pr[B(t) = n, V(t) \leq v + \Delta v | V(t) > v]$ be the conditional probability of having n frames in the buffer and the elapsed time for the next frame to arrive is less than or equal to $v + \Delta v$, given that the elapsed time to arrive is greater than v. Applying the definition of the conditional probability, yields:

$$Pr[B(t) = n, V(t) \leq v + \Delta v | V(t) > v] = \frac{Pr[B(t) = n, v < V(t) \leq v + \Delta v]}{Pr[V(t) > v]}$$

Dividing the above equation by Δv and taking the limit as $\Delta v \to 0$, yields:

$$\lim_{\Delta v \to 0} \frac{Pr[B(t) = n, v < V(t) \leq v + \Delta v]}{Pr[V(t) > v] \Delta v} = \frac{\pi_n(t, v)}{\bar{F}(v)} \triangleq p_n(t, v)$$

Where $\bar{F}(v)$ is the complementary cumulative distribution function of the frame inter-arrival time delay. The last result represents the instantaneous rate function of $\Pi_n(t, v)$, the corresponding cumulative distribution of $\pi_n(t, v)$, and we abbreviate it as $p_n(t, v)$:

$$\therefore \pi_n(t, v) = p_n(t, v) \bar{F}(v), \quad 0 \leq n \leq M - 1 \tag{2.13}$$

Differentiate Equation (2.13) two times, one with respect to t and the other with respect to v and add the result:

$$\Rightarrow \frac{\partial \pi_n(t, v)}{\partial t} = \frac{\partial p_n(t, v)}{\partial t} \bar{F}(v) \quad \text{and} \quad \frac{\partial \pi_n(t, v)}{\partial v} = \frac{\partial p_n(t, v)}{\partial v} \bar{F}(v) - p_n(t, v) f(v)$$

$$\Rightarrow \frac{\partial \pi_n(t, v)}{\partial t} + \frac{\partial \pi_n(t, v)}{\partial v} = \frac{\partial p_n(t, v)}{\partial t} \bar{F}(v) + \frac{\partial p_n(t, v)}{\partial v} \bar{F}(v) - p_n(t, v) f(v) \tag{2.14}$$

In Equation (2.14), we used the relation $\lambda(v) = \frac{f(v)}{\bar{F}(v)}$, where $f(v)$ is the probability density function of the inter-arrival time delay. Then, substitute Equations (2.13) and (2.14) in Equations (2.7, 2.8, 2.9, 2.10, and 2.12) to get the final time-dependent state equations:

$$\frac{\partial p_0(t, v)}{\partial t} + \frac{\partial p_0(t, v)}{\partial v} = p_1(t, v) \mu \tag{2.15}$$

$$\frac{\partial p_n(t, v)}{\partial t} + \frac{\partial p_n(t, v)}{\partial v} = -p_n(t, v) \mu + p_{n+1}(t, v) \mu, \quad 1 \leq n \leq M - 2 \tag{2.16}$$

$$\frac{\partial p_{M-1}(t,v)}{\partial t} + \frac{\partial p_{M-1}(t,v)}{\partial v} = -p_{M-1}(t,v)\mu \tag{2.17}$$

$$\frac{\mathrm{d}\pi_M(t)}{\mathrm{d}t} = -\pi_M(t)\mu + \int_0^\infty p_{M-1}(t,v)f(v)\mathrm{d}v \tag{2.18}$$

$$\frac{\mathrm{d}\pi_n(t)}{\mathrm{d}t} = -\pi_n(t)\mu + \pi_{n-1}(t)\mu, \quad M+1 \leq n \leq 2M-L-1 \tag{2.19}$$

Subject to the following boundary conditions:

$$\begin{aligned} &p_0(t,0) = 0, \\ &p_n(t,0) = \int_0^\infty p_{n-1}(t,v)f(v)\mathrm{d}v, \quad 1 \leq n \leq M-1\backslash\{L\}, \\ &p_L(t,0) = \int_0^\infty p_{L-1}(t,v)f(v)\mathrm{d}v + \pi_{2M-L-1}(t)\mu. \end{aligned} \tag{2.20}$$

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:

$$\begin{aligned} &\frac{\mathrm{d}p_0(v)}{\mathrm{d}v} = p_1(v)\mu, \\ &\frac{\mathrm{d}p_n(v)}{\mathrm{d}v} = -p_n(v)\mu + p_{n+1}(v)\mu, \quad 1 \leq n \leq M-2, \\ &\frac{\mathrm{d}p_{M-1}(v)}{\mathrm{d}v} = -p_{M-1}(v)\mu, \\ &0 = -\pi_M\mu + \int_0^\infty p_{M-1}(v)f(v)\mathrm{d}v, \\ &0 = -\pi_n\mu + \pi_{n-1}\mu, \quad M+1 \leq n \leq 2M-L-1. \end{aligned} \tag{2.21}$$

Subject to the following boundary conditions:

$$\begin{aligned} &p_0(0) = 0, \\ &p_n(0) = \int_0^\infty p_{n-1}(v)f(v)\mathrm{d}v, \quad 1 \leq n \leq M-1\backslash\{L\}, \\ &p_L(0) = \int_0^\infty p_{L-1}(v)f(v)\mathrm{d}v + \pi_{2M-L-1}\mu. \end{aligned} \tag{2.22}$$

Now, the set of Equations (2.21) and (2.22) describes the dynamics of the playback buffer length. The probability distribution of the states can be evaluated using the following equation:

$$\pi_n = \int_0^\infty p_n(v)\bar{F}(v)\mathrm{d}v, \quad 0 \leq n \leq M-1 \tag{2.23}$$

To obtain the stationary distribution of the stochastic process underlying the playback buffer length, we need to solve the system of equations (2.21, 2.22, and 2.23) with the normalization equation that is given by:

$$\sum_{n=0}^{2M-L-1} \pi_n = 1. \tag{2.24}$$

Unfortunately, the solution of the problem does not give the performance metric under consideration (π_0) in a closed form expression.

The number of equations that should be solved depends on M and L which depends on the playback buffer size. Thus, it is not easy to solve them manually in order to get the probability of empty buffer state. Therefore, it is more convenient to formulate the equations in a matrix form and solve them by the computer. For this purpose, we can follow the same notation that is developed in [51]. The solution gives the stationary probability distribution of the Markov chain $(\pi_0, \pi_1, ..., \pi_{2M-L-1})$. We are only interested in π_0.

2.3 Performance Model Validation

To verify the performance modelling process, we develop a simulation performance model. Figure 2.4 shows the probability of empty playback buffer state ($\boldsymbol{\pi_0}$) with different buffer sizes for both simulation and analytical performance models, where μ represents the mean rate of frame decoding and f is a tunable parameter that is used to control the relation between the frame arrival rate and the decoding rate. In the simulation model, we simulate a streaming session of 30 minutes. Each simulation experiment is repeated 50 times. Thus, the reported probability of empty buffer state is the mean of 50 experiments.

An important point to mention, as playback buffer size increases, the time overhead of evaluating the analytical model (Equations (2.21-2.24)) becomes comparable with the time overhead of evaluating the simulation model. In fact, this issue is already observed in Markov modelling when the chain size is large and the underlying process is not fully Markovian [51].

2.4 Summary

In this chapter, both analytical and simulation performance models are developed for mobile networked applications of group I, where waiting time delay that affects end user

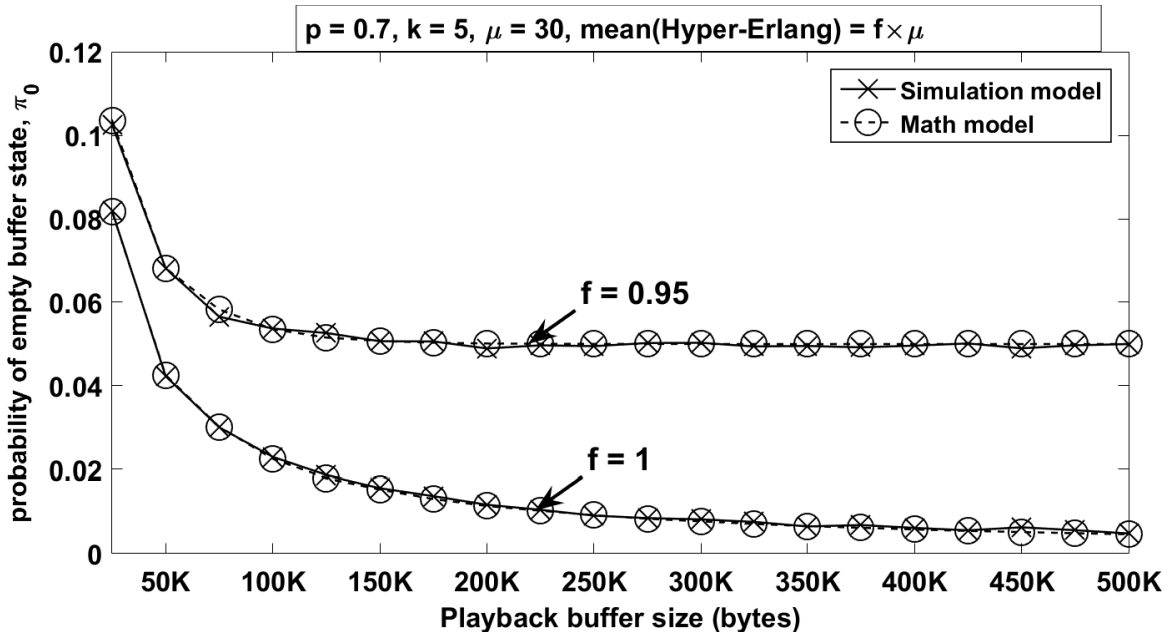

Figure 2.4: The considered performance metric versus playback buffer size.

experience occurs during service consumption. We developed the models for a multimedia streaming application as a representative application example for this group. We developed the analytical model using a Markov chain. Because the stochastic process underlying the playback buffer length is not Markovian, the Markov chain is developed and solved using the supplementary variable technique. This technique enables for frame arrivals to be modelled using any probability distribution, allowing for more realistic wireless network behaviours to be considered in the process. The main observation of this modelling process is the computationally high cost of the analytical performance model for group I applications.

In the next chapter, input network models are developed for mobile networked applications that have a wireless network access via a WiFi access point.

Chapter 3

Input Network Models

Basically, a network model relates analytically the metric that captures the quality of the network service, which is the waiting time delay, with the identified network operating parameters (NOPs). Waiting time delay is verily a random variable and we need to evaluate its probability distribution. We utilize distribution fitting using the first two moments: mean and variance. Therefore, two analytical expressions are derived for the mean and variance of waiting time delay. The mobile device is assumed to have a wireless network connection via a WiFi access point (AP) that implements the Distribution Coordination Function mode of operation of the IEEE 802.11 protocol standard. We consider two scenarios. In the first scenario, data transfer is achieved using the UDP transport protocol, while in the second scenario, data transfer is achieved using the TCP transport protocol. In both scenarios, we assume there are N mobile users that are connected to the Internet through the same WiFi access point. The direction of data traffic is majorly from the Internet to the mobile devices via the WiFi access point. We first assume that the size of the transport layer packet (UDP or TCP), MAC frame, and application data unit (APDU) are equal. Later on, we relax this assumption to make the model more suitable for a wide range of applications.

3.1 IEEE 802.11 Protocol Standard

As explained in the system model, the application interacts with the network through a basic *request-response* (REQ-RES) mechanism and the network's impact can be captured by modelling the RES inter-arrival time delay. The objective of this analysis is to develop

two analytical expressions for the mean and variance of RES (packet/APDU) inter-arrival time delay.

Nowadays, IEEE 802.11 becomes the de facto protocol for the wireless local area networks (WLANs). It can operate in one of two modes of operation: a contention based medium access known as a Distributed Coordination Function (DCF) and a contention free medium access known as a Point Coordination Function (PCF) [17]. In this work, we only consider the DCF scheme. This scheme employs a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism to manage the access to the shared medium among multiple users. In this mode of operation, a station (mobile device or AP) can only send a MAC frame, if and only if, the channel is sensed idle for a Distributed Inter-Frame Space interval of time (DIFS). If the channel is sensed busy from the beginning or while waiting for the DIFS interval to elapse, the station has to wait for an extra random amount of time after a complete DIFS interval of being sensed idle. This random amount of time is called the back-off interval. It is uniformly distributed in the interval [0, W], where W represents the Contention Window size. Through this random interval, the Collision Avoidance mechanism is implemented.

In practice, the back-off interval is an integer number of equally spaced time slots of length σ. After a DIFS interval of the channel being idle, a number is picked randomly from the interval [0, W] and loaded into the back-off counter. At the beginning of each time slot, the channel is sensed. If it is idle, the counter is decremented by 1; otherwise, the counter is frozen. The counter resumes decrementing whenever the channel is sensed again idle for a DIFS interval. The station sends the MAC frame when the back-off counter reaches zero.

In the DCF mode, stations can exchange data using two different schemes. In the first scheme, usually known as DATA/ACK scheme, the sender station sends the DATA (MAC frame) directly after the back-off counter hits the zero. If it is received correctly, the receiver station waits for a Short Inter-Frame Space (SIFS) interval of time and then sends back an ACK frame. If the DATA or ACK frame collides, the sender station has to reschedule the transmission event again. In the second scheme, the sender firstly sends a "request-to-send" (RTS) message to the receiver. Upon receiving RTS message, the receiver station waits for a SIFS interval of time and then responds back with a "clear-to-send" (CTS) message. When the sender receives the CTS message, he waits for a SIFS time interval and then sends the DATA frame. If the DATA frame is received correctly, the receiver station waits again for a SIFS time interval and then responds back with an ACK frame. This four-way handshaking mechanism has two advantages. It handles the hidden station problem and it reduces the collision time especially when long DATA messages are exchanged.

3.2 Data Transfer Using A UDP Protocol

In this section, we analyse the interaction of the UDP protocol with IEEE 802.11 standard. User Datagram Protocol, abbreviated as UDP, is an unreliable and connectionless transport protocol. It does not implement any flow or congestion control mechanisms [59]. It is suitable for short message transfers as in network management activities and it is also used for delay sensitive applications such as multimedia streaming.

Assuming a UDP packet, MAC frame, and APDU are of equal size, the packet transmission time T_s^{udp}, time needed to successfully transfer a frame from the AP to the end user, is given by:

$$T_s^{udp} = T_{RTS} + T_{CTS} + 3 \times T_{SIFS} + T_{DIFS} + T_{ACK} + T^{udp} \tag{3.1}$$

Where T_{RTS}, T_{CTS}, and T_{ACK} are propagation time of a ready-to-send frame, clear-to-send frame, and ACK frame, respectively; T_{SIFS} and T_{DIFS} are the short inter-frame space and distributed inter-frame space time intervals, respectively. The packet propagation time T^{udp} depends on the packet size and on the data rate D of the wireless connection:

$$T^{udp} = T_{PHY} + \frac{H_{UDP} + H_{MAC} + 8 \cdot P}{D} \tag{3.2}$$

Where T_{PHY}, H_{UDP}, H_{MAC}, and P represent the physical layer overhead in seconds, UDP packet header size in bits, MAC header size in bits, and UDP packet payload size in bytes, respectively. We assume that the WiFi access point has an 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 is corresponding to the service time of the packet. Packet service time S_t is given by:

$$S_t = T_s^{udp} + \sigma \times C_B \tag{3.3}$$

Where C_B is the back-off counter value. Thus, the mean packet service time $E[S_t]$ is given by:

$$E[S_t] = T_s^{udp} + \sigma \times E[C_B] \tag{3.4}$$

In this analysis, we assume the contention window size is fixed on W. Thus, the mean of the back-off counter is basically the mean of a uniform random variable. Hence, Equation (3.4) becomes:

$$E[S_t] = T_s^{udp} + \sigma \times \frac{W}{2} \tag{3.5}$$

We assume that packet arrivals to the AP follow a Poisson process. For wired networks, it is a quite reasonable assumption. Since the data traffic is assumed to be unidirectional from the AP to mobile end users, the collision time is negligible. Moreover, since the contention window size is assumed to be fixed, the variance in packet response time is mainly due to the variance in the AP queue waiting time. As a result, packet service time can be assumed to be deterministic as mathematically proven below:

$$\begin{aligned} C_s = \frac{std(S_t)}{E[S_t]} = \frac{\sqrt{E[S_t^2] - E[S_t]^2}}{E[S_t]} = \sqrt{\frac{E[S_t^2]}{E[S_t]^2} - 1} \\ = \sqrt{\frac{{T_s^{udp}}^2 + \sigma T_s^{udp} W + \frac{1}{3}\sigma^2 W^2}{{T_s^{udp}}^2 + \sigma T_s^{udp} W + \frac{1}{4}\sigma^2 W^2} - 1} \approx 0 \end{aligned} \tag{3.6}$$

Where $std(S_t)$ and C_s are the standard deviation and the coefficient of variation of the service time S_t, respectively. Therefore, assuming the AP as an M/D/1 queueing system, the mean of the packet inter-arrival time delay at the end user is basically the mean packet response time at the AP. The sum of the queueing time delay and the service time represents the packet response time and it is given by [67]:

$$E_r = \frac{(2-\rho)}{2 \cdot \alpha \cdot (1-\rho)} \tag{3.7}$$

Where α is the mean packet service rate ($= 1/E[S_t]$) at the AP. The traffic intensity at the access point ρ is given by:

$$\rho = \frac{N \cdot \lambda}{\alpha} \tag{3.8}$$

Where λ is the mean rate of packet arrival into the AP per mobile user, and N is the number of mobile users that are connected to the AP.

The next step is to figure out the variance in the packet inter-arrival time delay V_{ard}. For M/D/1 queue, it is given by [67]:

$$V_{ard} = \frac{\rho \cdot (4-\rho)}{12 \cdot \alpha^2 \cdot (1-\rho)^2} \tag{3.9}$$

Using the mean and variance (E_r and V_{ard}), we fit a distribution for the packet inter-arrival time delay. If the coefficient of variation of the packet inter-arrival time delay C_r is less than 1, a Hyper-Erlang distribution is often used in the matching process. A Hyper-Erlang distribution is a weighted mixture of a number of independent Erlang distributions. An

Erlang random variable is a random variable that has a distribution with two parameters k and ω. It is the sum of k independent exponential random variables with mean $1/\omega$ each. In this work, we assume the arrival process is a Hyper-Erlang of two Erlang distributions of the same parameter ω. This type of a distribution is useful if only the first two moments (mean and variance) are used in the matching process [4]. Respectively, the probability density function and the cumulative distribution function for a Hyper-Erlang random variable of two Erlang variables are given below:

$$\begin{aligned} f(v) &= p\omega\frac{(\omega v)^{k-2}}{(k-2)!}e^{-\omega v} + (1-p)\omega\frac{(\omega v)^{k-1}}{(k-1)!}e^{-\omega v}, \quad v > 0 \\ F(v) &= 1 - \sum_{j=0}^{k-1}\frac{(\omega v)^j}{j!}e^{-\omega v} + p\frac{(\omega v)^{k-1}}{(k-1)!}e^{-\omega v}. \end{aligned} \tag{3.10}$$

Hyper-Erlang distribution has three parameters k, p, and ω. k should be an integer and p is a probability between 0 and 1. They can be determined as follows:

$$\frac{1}{k} \leq {C_r}^2 \leq \frac{1}{k-1} \tag{3.11}$$

$$p = \frac{1}{1+{C_r}^2}[k\cdot{C_r}^2 - \sqrt{k\cdot(1+{C_r}^2) - k^2\cdot{C_r}^2}] \tag{3.12}$$

$$\omega = \frac{k-p}{E_r} \tag{3.13}$$

where

$${C_r}^2 = \frac{V_{ard}}{{E_r}^2} \tag{3.14}$$

3.3 Model Validation for The UDP Scenario

To validate that the matched Hyper-Erlang cumulative distribution function (CDF) has captured the dynamics of packet inter-arrival time delay, we conduct simulation experiments using the Network Simulator tool NS2. We simulate a wireless local area network that has one access point and a group of users. The simulated scenario is a file download, where a number of users are downloading a big file through the AP simultaneously. Figures 3.1 and 3.2 show both the empirical CDF that is generated from the simulation and the analytical matched CDF. We consider IEEE 802.11 g protocol with parameter values as given in Table 3.1. The packet payload size P is 1500 Bytes.

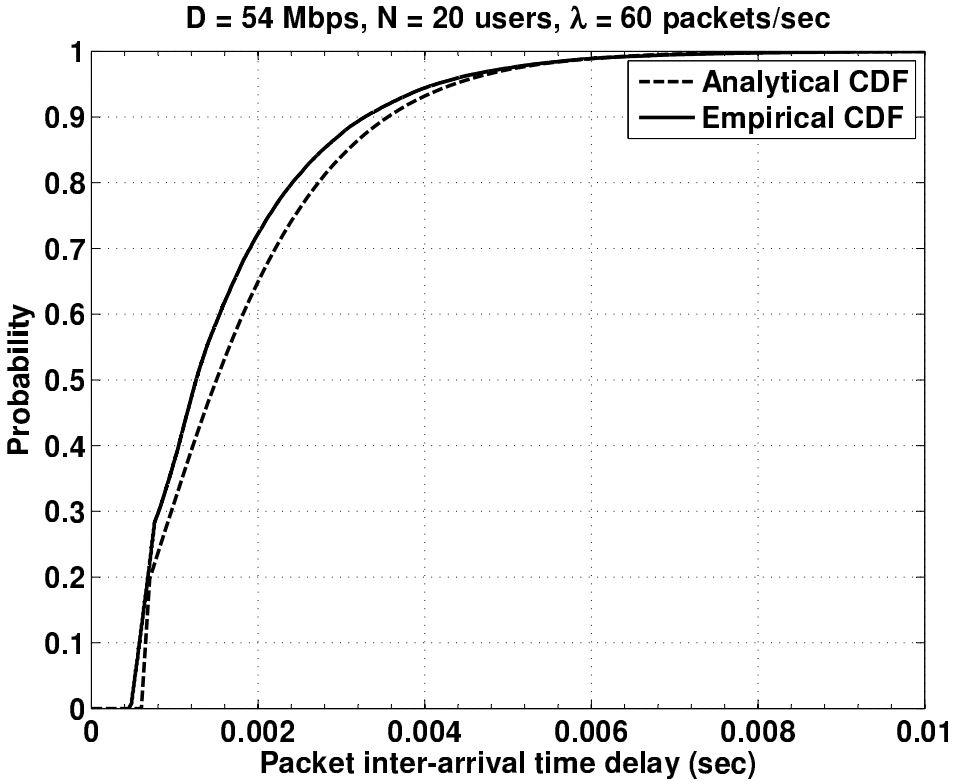

Figure 3.1: The empirical and analytical cumulative distribution functions for a mean arrival rate of 60 packets/sec, end users of 20, and data rate of 54 Mbps.

Table 3.1: IEEE 802.11g protocol parameters.

System Parameter	**Value**
H_{MAC}	246 $bits$
H_{UDP}	160 $bits$
T_{PHY}	20 μs
T_{RTS}	47 μs
T_{CTS}	39 μs
T_{ACK}	39 μs
T_{SIFS}	16 μs
T_{DIFS}	34 μs
W	32
σ	9 μs

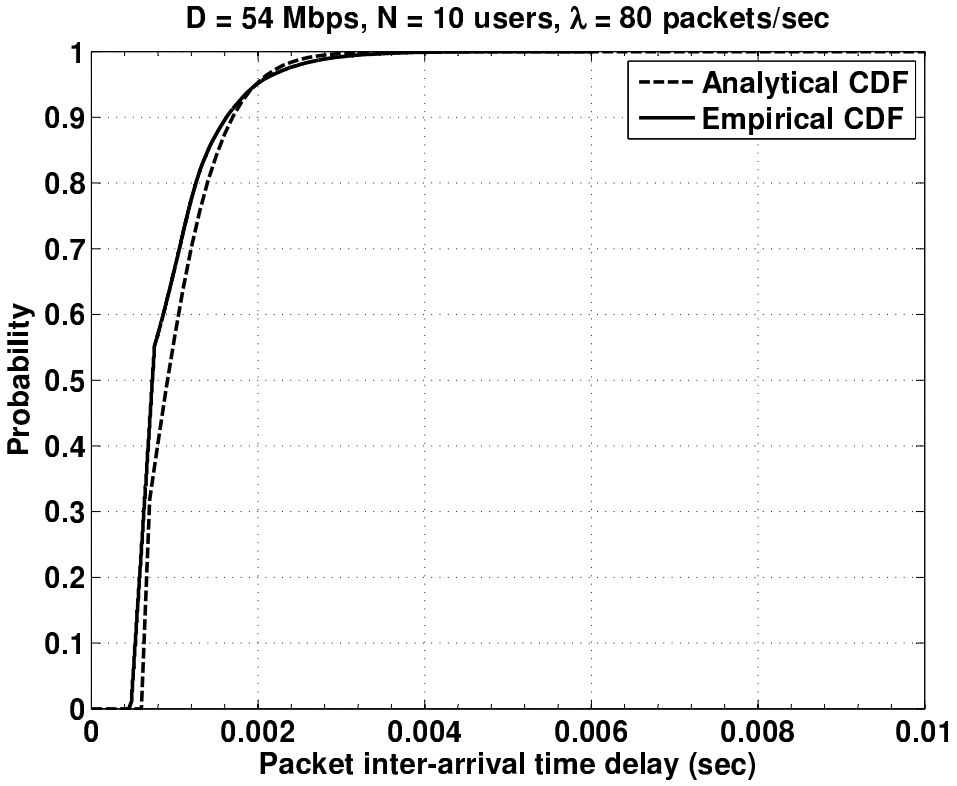

Figure 3.2: The empirical and analytical cumulative distribution functions for a mean arrival rate of 80 packets/sec, end users of 10, and data rate of 54 Mbps.

3.4 Data Transfer Using A TCP Protocol

In this section, we analyse the interaction of TCP protocol with IEEE 802.11 standard. Compared to UDP protocol, TCP protocol is a reliable transport protocol that implements both flow and congestion control mechanisms. Congestion control is a mechanism that is employed to protect network routers from being overwhelmed with packets, while control flow is a mechanism that is used in order not to overflow end-point receivers with packets more than the capacity of their TCP buffers. In TCP protocol, each packet should be acknowledged by the receiver. Thus, we have two types of traffic. In this work, we assume TCP packets flow from the wired network part (Internet) to the mobile users via the AP, and TCP ACKs flow back from the mobile users to the AP. All these mechanisms make TCP protocol more reliable, but at the same time, they make packet delay analysis more complicated compared to UDP scenario.

As in UDP scenario, we assume that the AP has an infinite buffer space. Thus, the system throughput is limited by the TCP buffer size in mobile devices. Congestion window size at the AP is assumed to grow until it reaches the receiver TCP buffer size of mobile devices. The time needed to successfully deliver a TCP packet T_s^{tcp} is the same as in the UDP scenario which is given by Equation (3.1). The TCP packet propagation time T^{tcp} is also the same as in Equation (3.2) except H_{UDP} is replaced by H_{TCP}, the header size of the TCP packet. Similarly, the time needed to successfully deliver a TCP acknowledge (ACK) packet T_s^{ack} from the mobile user to the AP is given by:

$$T_s^{ack} = T_{RTS} + T_{CTS} + 3 \cdot T_{SIFS} + T_{DIFS} + T_{ACK} + T^{ack} \tag{3.15}$$

where T^{ack} is the TCP ACK propagation time and it is given by:

$$T^{ack} = T_{PHY} + \frac{H_{TCP}^{ack} + H_{MAC} + 8 \cdot P_{ack}}{D} \tag{3.16}$$

where H_{TCP}^{ack} and P_{ack} represent TCP layer header size in bits and payload size in bytes for a TCP ACK packet, respectively.

To derive an expression for the mean response time E_r^{tcp} at the AP, we use the same assumption that TCP packet arrivals at the AP follow a Poisson process. The mean rate of TCP packets arrival λ_{AP} is given by:

$$\lambda_{AP} = \frac{N \cdot B^{tcp}}{T_{rtt}} \tag{3.17}$$

Where B^{tcp} is the TCP buffer size at the end user. The round trip time of the TCP packet T_{rtt} is dominated by the delay of the wireless network part and can be approximated by:

$$T_{rtt} = \frac{1}{\alpha_n} + E_r^{tcp} \tag{3.18}$$

Where α_n is the mean service rate of TCP ACK packets at the mobile end user. By ignoring packets collision time, back-off delays, and utilizing the fairness property of the DCF operation mode of the IEEE 802.11 protocol, the mean service rate α at the AP is given by [3, 23]:

$$\alpha = \frac{1}{\rho_n \cdot N \cdot T_s^{ack} + T_s^{tcp}} \tag{3.19}$$

Where ρ_n is the traffic intensity at the mobile end user and on long-run, it can be approximated by:

$$\rho_n = \frac{1}{N} \tag{3.20}$$

Thus, mean service rate at the AP is approximately given by:

$$\alpha = \frac{1}{T_s^{tcp} + T_s^{ack}} = \frac{1}{T_{st}} \tag{3.21}$$

Where T_{st} is the total transmission time of TCP and ACK packets. Assuming the AP as an M/D/1 queueing system, the mean response time at the AP is given by Equation (3.7) with α now is as given by Equation (3.21) and ρ as given by the following equation:

$$\rho = \frac{N \cdot B^{tcp}}{T_{rtt} \cdot \alpha} \tag{3.22}$$

Now, we find an expression for the TCP buffer size B^{tcp} at the end user. Using the same rationale, the mean service rate for TCP ACKs at the mobile end user is given by:

$$\alpha_n = \frac{1}{\rho \cdot T_s^{tcp} + \rho_n \cdot (N-1) \cdot T_s^{ack} + T_s^{ack}} \tag{3.23}$$

Substituting Equation (3.23) in Equation (3.18) and the result in Equation (3.17) yield the following expression for the TCP buffer size:

$$B^{tcp} = \frac{\rho}{N} \times [\frac{\rho \cdot T_s^{tcp} + \rho_n \cdot (N-1) \cdot T_s^{ack} + T_s^{ack}}{T_{st}} + \frac{(2-\rho)}{2 \cdot (1-\rho)}] \tag{3.24}$$

Finding out an expression for the variance of response time in TCP scenario is more challenging than the UDP case. Assuming heavy traffic analysis, the traffic intensity at

the AP is very close to one. By observing the simulation, the buffer length at the AP is almost constant at a level that is very close to the mean buffer length given by an M/D/1 queueing system [67]:

$$Q_L = \frac{\rho^2}{2 \cdot (1 - \rho)} + \rho \tag{3.25}$$

Using Little theorem, both mean total service time and response time at the AP are related by the following:

$$\lambda_{AP} \times E_r = Q_L \Rightarrow E_r \approx \frac{Q_L}{\alpha} \Rightarrow E_r \approx Q_L \times T_{st} \tag{3.26}$$

With negligible buffer length variance, the variance in response time is mainly due to the variance in service time, which is mainly due to the variance in the back-off process (collision time is neglected). The variance in back-off time delay is basically the variance of a uniform random variable. Therefore, the variance in response time V_{ard}^{tcp} can be approximated by:

$$V_{ard}^{tcp} = 2 \cdot Q_L \cdot \frac{W^2 - 1}{12} \cdot \sigma^2 \tag{3.27}$$

Where the factor 2 in the last equation is to account for waiting of both TCP and its ACK packets.

Using the mean and variance, we fit the packet inter-arrival time delay with a Gamma distribution. This distribution has two parameters: shape and rate. They are related to the mean and variance by the following relationships:

$$\begin{aligned} shape &= \frac{(E_r^{tcp})^2}{V_{ard}^{tcp}} \\ rate &= \frac{E_r^{tcp}}{V_{ard}^{tcp}} \end{aligned} \tag{3.28}$$

3.5 Model Validation for The TCP Scenario

The fitted distribution is validated with simulation using NS2 tool. Figures 3.3, 3.4, and 3.5 show both the matched and the empirical cumulative distribution functions for three different scenarios.

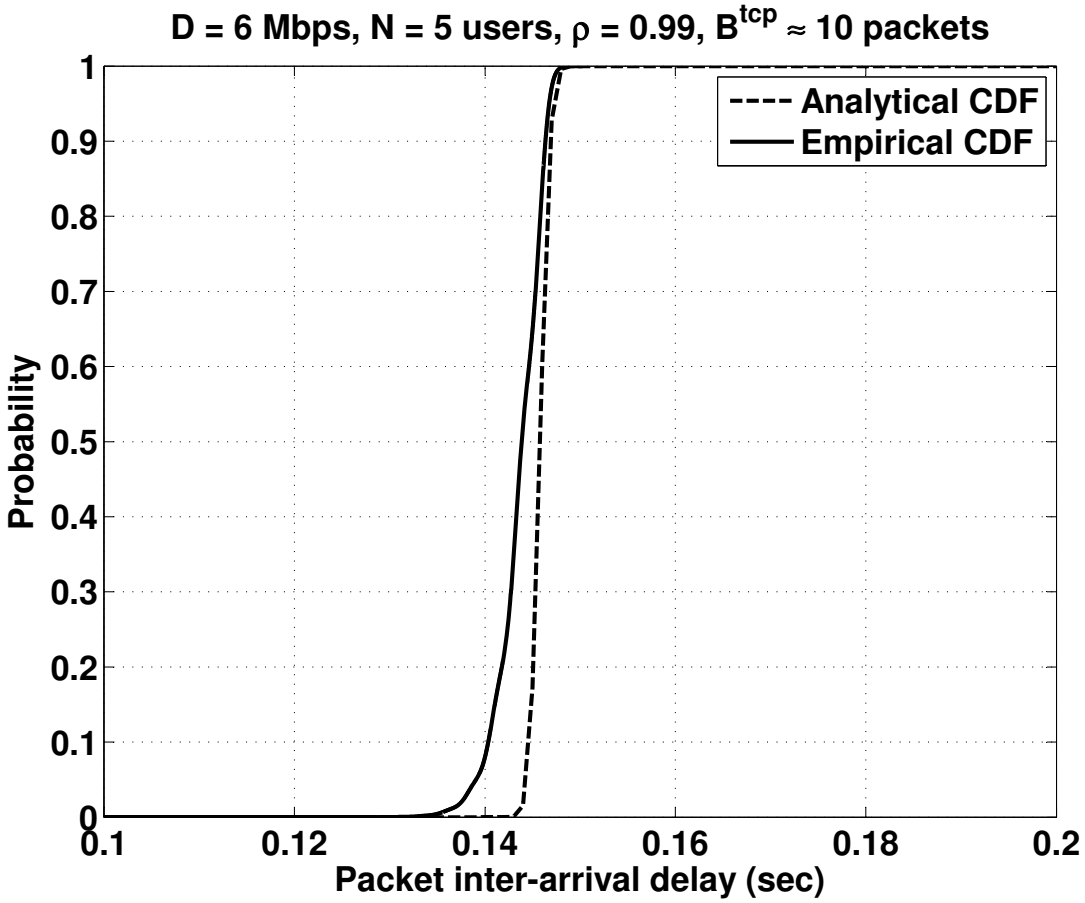

Figure 3.3: The empirical and analytical cumulative distribution functions for traffic intensity of 0.99, TCP buffer size of 10 packets, end users of 5, and data rate of 6 Mbps.

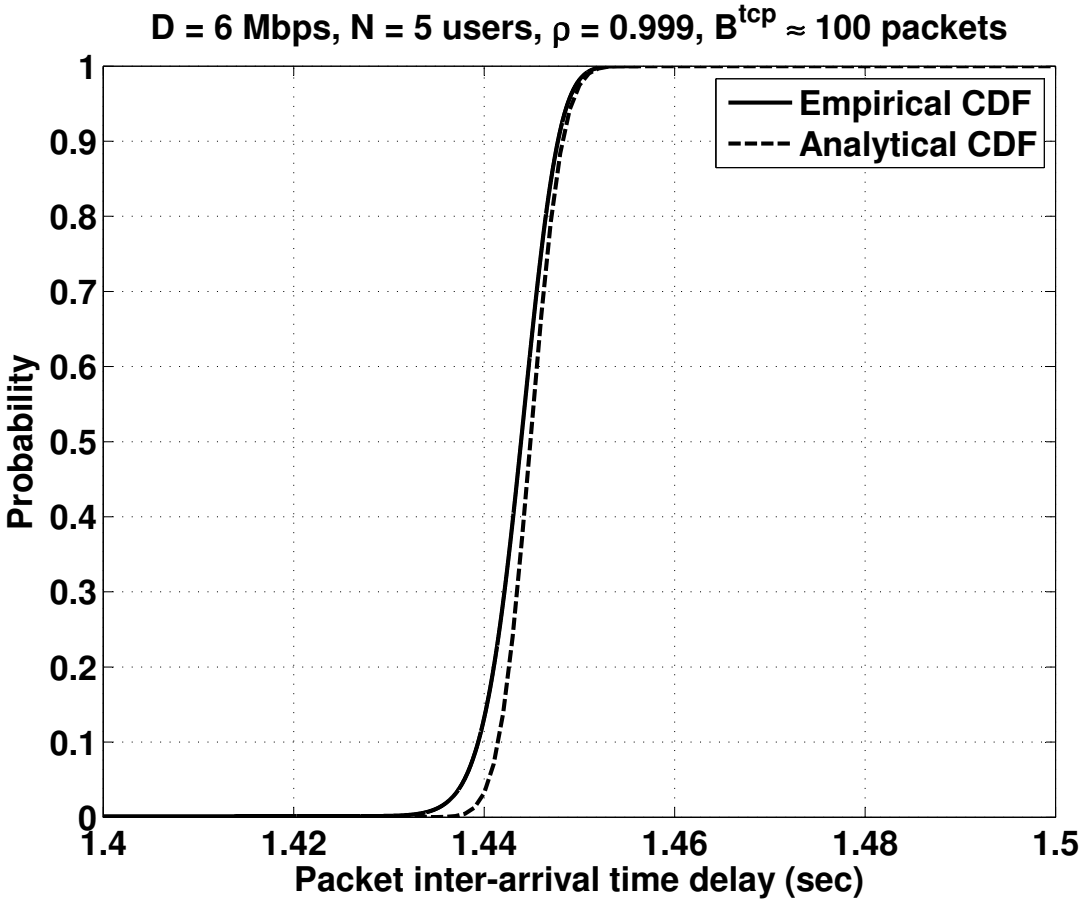

Figure 3.4: The empirical and analytical cumulative distribution functions for traffic intensity of 0.999, TCP buffer size of 100 packets, end users of 5, and data rate of 6 Mbps.

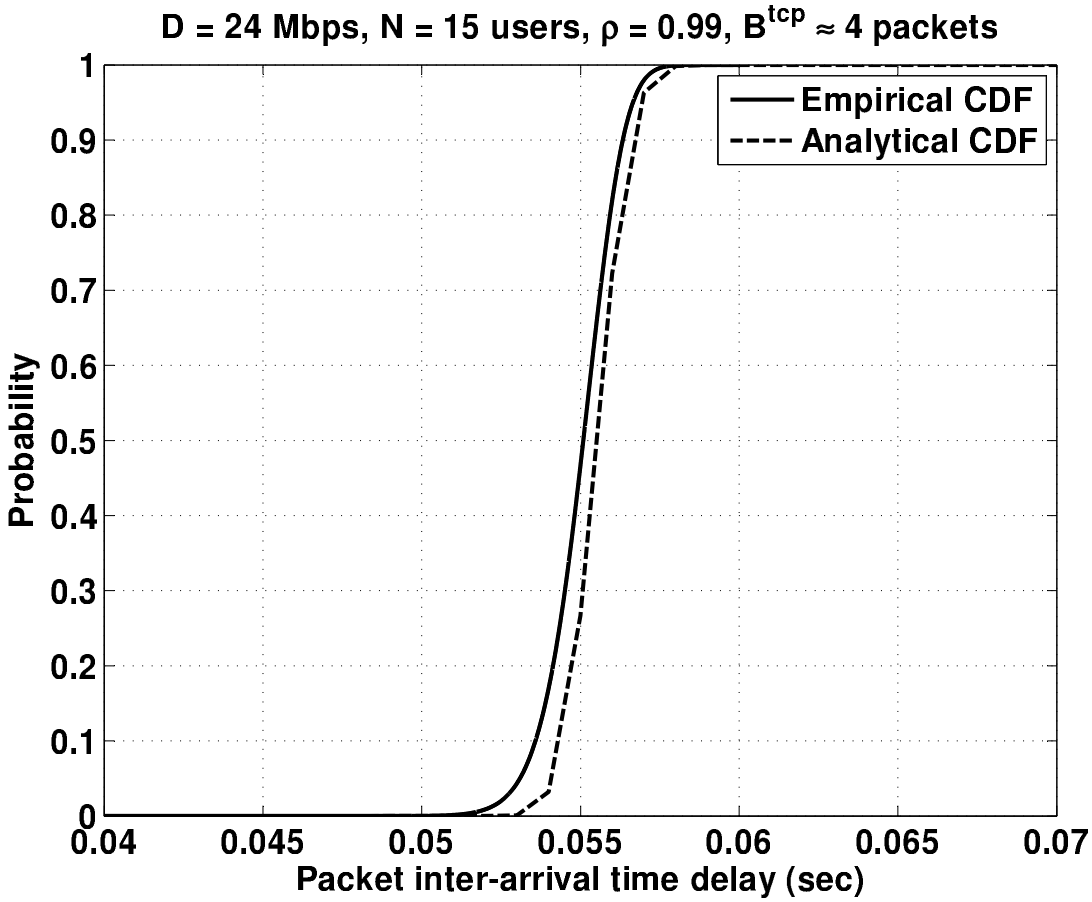

Figure 3.5: The empirical and analytical cumulative distribution functions for traffic intensity of 0.99, TCP buffer size of 4 packets, end users of 15, and data rate of 24 Mbps.

3.6 The TCP Scenario With Multiple Packet APDUs

In the previous section, we assume that a TCP packet, MAC frame, and APDU are all of the same size. In this section, we relax this assumption and develop a network model for applications with data units that fit into multiple TCP packets and MAC frames. The size of TCP packets and MAC frames are still assumed to be equal, which is practically acceptable. We assume for an arbitrary application that one APDU on average is partitioned and transferred using K TCP packets (or MAC frames). Thus, the previous network model is a special case where K is equal to one.

When $K > 1$, the mean inter-arrival time delay of an APDU at the end user is determined by the mean of three delay components at the AP: TCP packets inter-arrival time delay, TCP packets waiting time delay, and TCP packets service time. What makes time delay analysis more complicated than the previous scenario is the overlap of the mentioned three time delay components for packets that belong to the same APDU. Therefore, to evaluate the overlap, we need first to anticipate the mean of each delay component compared to the other two. Let $1/\lambda$ be the mean TCP packet inter-arrival time delay per end user at the AP, $1/\alpha$ be the mean TCP packet service time, and T_q be the mean waiting time for TCP packets at the AP. Now, we figure out analytically the magnitude of each component in relative to the other two delay components and determine as well the conditions under which such relationships are valid. In reality, no model can stay valid anywhere all the time; every model comes with its assumptions and limitations.

We start by inferring the relationship between $1/\lambda$ and $1/\alpha$. They are related by the traffic intensity relationship:

$$\begin{aligned} &\rho = \frac{N \cdot \lambda}{\alpha} \Rightarrow \frac{\lambda}{\alpha} = \frac{\rho}{N} \\ &\because \ \rho < 1 \text{ and } N > 1 \\ &\therefore \frac{\lambda}{\alpha} < 1 \Rightarrow \lambda < \alpha \Rightarrow \boxed{\frac{1}{\lambda} > \frac{1}{\alpha}} \end{aligned} \tag{3.29}$$

Therefore, the inter-arrival packet delay on average is greater than packet service time. This relation is valid if there are two or more mobile users connected to the AP and the traffic intensity is less than 1.

The relation between $1/\alpha$ and T_q is deduced as follows. For an M/D/1 queueing system,

the mean queueing time in the AP buffer is given by:

$$\begin{aligned} T_q &= \frac{\rho}{2 \cdot \alpha \cdot (1-\rho)} \\ \Rightarrow \frac{T_q}{1/\alpha} &= \frac{\rho}{2 \cdot (1-\rho)} \\ \Rightarrow \frac{T_q}{1/\alpha} > 1 &\Rightarrow \boxed{T_q > \frac{1}{\alpha}} \end{aligned} \tag{3.30}$$

Thus, the queueing delay on average is greater than the packet service time. The last result depends on our previous assumption of heavy traffic analysis, where ρ is very to 1.

The last relationship that we want to deduce is between T_q and $1/\lambda$. Using the same relationship of mean queueing delay in an M/D/1 queue, the relationship is derived as follows:

$$\begin{aligned} \frac{T_q}{1/\lambda} &= \frac{\lambda \cdot \rho}{2 \cdot \alpha \cdot (1-\rho)} \\ \Rightarrow \frac{T_q}{1/\lambda} &= \frac{\rho^2}{2 \cdot N \cdot (1-\rho)} \\ \Rightarrow \frac{T_q}{1/\lambda} > 1 &\Rightarrow \boxed{T_q > \frac{1}{\lambda}} \end{aligned} \tag{3.31}$$

Assuming heavy traffic analysis, this result is valid if the number of users N are less than 49, which is beyond our requirements. Therefore, compared to each other, the three delay components are related by the following inequality:

$$\boxed{\frac{1}{\alpha} < \frac{1}{\lambda} < T_q} \tag{3.32}$$

Using this inequality, the overlap among the three components is depicted in Figure 3.6. Therefore, the end to end mean inter-arrival time delay for an APDU of K TCP packets can be approximated by:

$$E_r^{APDU} = \frac{K-1}{\lambda} + \frac{1}{\alpha} + T_q \tag{3.33}$$

Where the mean service time of TCP packets $1/\alpha$ in the last equation is the sum of the value reported in Equation (3.21) and the mean delay due to the back-off process ($\frac{(W-1)\cdot\sigma}{2}$).

Developing a closed form expression for the variance of the inter-arrival APDU time delay is quite challenging. We obtain an expression with a good approximation by modelling the AP as a batch service queueing system $M/D^{[a,b]}/1$ with a greedy policy, where b

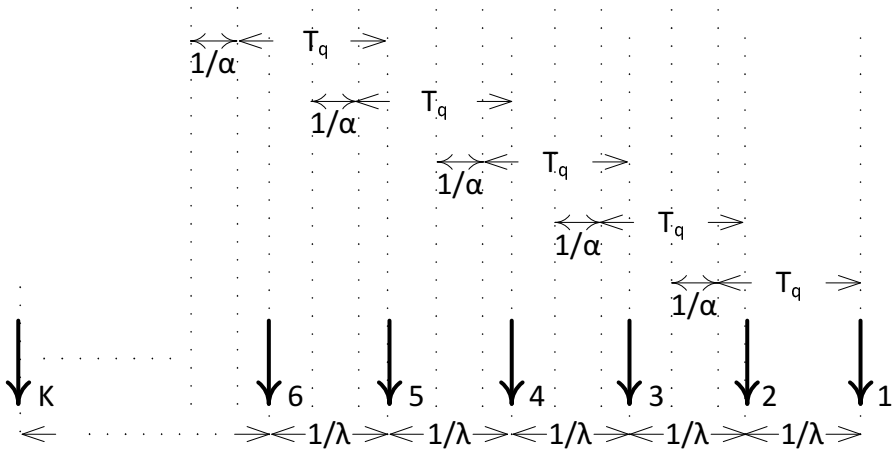

Figure 3.6: The inter-arrival, waiting, and service time delay of K TCP packets at the access point, which constitute one full APDU.

is the maximum number of TCP packets to be served at the same time (batch size), and a is the minimum number of TCP packets in the AP buffer to start the service process. In our scenario, b is the size of the APDU, which is K TCP packets, and a is equal to 1. The main point in using this queueing system is to obtain an expression for the AP mean queue length in terms of APDUs. Then, we reuse Equation 3.27 for the variance of APDU response time. The mean queue size in terms of APDUs is given by [19]:

$$Q^{APDU} = \frac{\rho^2}{1-\rho} \cdot \frac{C_a^2 + C_s^{APDU^2}}{2} \cdot e^{-\frac{2(1-\rho)(1-C_a^2)}{3\rho(C_a^2+C_s^{APDU^2})}} \tag{3.34}$$

Where C_a is the coefficient of variation for the APDU inter-arrival time. With Poisson TCP packets arrival, C_a^2 is basically $1/K$. The coefficient of variation of APDU service time C_s^{APDU} is more complicated. For an APDU of K TCP packets, the probability distribution of the burst size Z (the number of consecutive TCP packets in the AP buffer for the same end user) is given by:

$$P_j = Pr[z = j] = \frac{N-1}{N^j} \qquad 1 \leq j \leq \infty \ , \ N \geq 2 \tag{3.35}$$

Therefore, for a batch system with a batch size equals to K, the batch service time as a random variable is given by:

$$S_b = Z \cdot S_{tcp} = Z \cdot [T_s^{tcp} + T_s^{ack} + U] \tag{3.36}$$

Where U is the back-off time delay random variable. Assuming Z and U are independent random variables, the mean batch service time is given by:

$$E[S_b] = E[Z \cdot S_{tcp}] = E[Z] \cdot E[S_{tcp}] \tag{3.37}$$

The mean packet service time is given by:

$$\begin{aligned} E[S_{tcp}] &= T_s^{tcp} + T_s^{ack} + E[U] \\ &= T_s^{tcp} + T_s^{ack} + \frac{(W-1)\cdot\sigma}{2} \end{aligned} \tag{3.38}$$

The mean of the burst size can be evaluated using the definition of expectation of a discrete random variable as follows:

$$\begin{aligned} E[Z] &= \sum_j j \cdot P_j = \sum_{j=1}^{K-1} j \cdot P_j + \sum_{j=K}^{\infty} K \cdot P_j \\ &= \frac{N^K - 1}{N^K - N^{K-1}} \end{aligned} \tag{3.39}$$

Therefore, the mean service time of the APDU is obtained by substituting Equations (3.38 and 3.39) into Equation (3.37).

Utilizing the same independence property, the variance of the batch service time is given by:

$$Var(S_b) = E[Z^2] \cdot E[S_{tcp}^2] - E[Z]^2 \cdot E[S_{tcp}]^2 \tag{3.40}$$

Where $E[Z^2]$ and $E[S_{tcp}^2]$ are evaluated by:

$$\begin{aligned} E[Z^2] &= \sum_j j^2 \cdot P_j = \sum_{j=1}^{K-1} j^2 \cdot P_j + \sum_{j=K}^{\infty} K^2 \cdot P_j \\ &= \frac{N^{K+1} + N^K - N - 2\cdot K \cdot N + 2\cdot K - 1}{N^{K+1} - 2\cdot N^K + N^{K-1}} \end{aligned} \tag{3.41}$$

and

$$\begin{aligned} E[S_{tcp}^2] &= E[(T_s^{tcp} + T_s^{ack} + U)^2] \\ &= (T_s^{tcp} + T_s^{ack})^2 + (T_s^{tcp} + T_s^{ack}) \cdot (W-1) \cdot \sigma + \frac{(W-1)\cdot(2\cdot W - 1)\cdot\sigma^2}{6} \end{aligned} \tag{3.42}$$

Therefore, the variance of batch service time is obtained by substituting Equations (3.38, 3.39, 3.41, and 3.42) in Equation (3.40). Consequently, the coefficient of variation of APDU

service time C_s^{APDU} is evaluated ($= \frac{Var(S_b)^{1/2}}{E[S_b]}$). Given the mean AP buffer size Q^{APDU}, the variance of the APDU inter-arrival time delay at the end user can be approximated by:

$$V_{ard}^{APDU} = 2 \cdot (Q^{APDU})^2 \cdot \frac{W^2 - 1}{12} \cdot \sigma^2 \cdot K \tag{3.43}$$

3.7 Validation of The TCP Model With APDUs of Multiple Packets

We match with Gamma distribution as well. Figures 3.7, 3.8, and 3.9 show cumulative distribution functions that are obtained analytically and using simulation. The traffic intensity ρ is assumed to be fixed on 0.99 (heavy traffic analysis). The error is around 5%.

3.8 Summary

In this chapter, input network models are derived analytically for a mobile networked application with a wireless network connection via a WiFi access point. Data transfer are assumed to be achieved using both TCP and UDP protocols. In each model, two analytical expressions are developed for the mean and variance of an APDU inter-arrival time delay. Then, a probability distribution is matched using the mean and variance expressions. For the UDP scenario, the waiting time delay is matched with a Hyper-Erlang distribution, while for the TCP scenario, it is matched with a Gamma distribution.

In the next chapter, a test generation methodology is proposed for performance evaluation. We will show how the developed network and performance models are used to generate a set of test cases to evaluate the impact of the interaction of ACPs and NOPs on the experience of the end user.

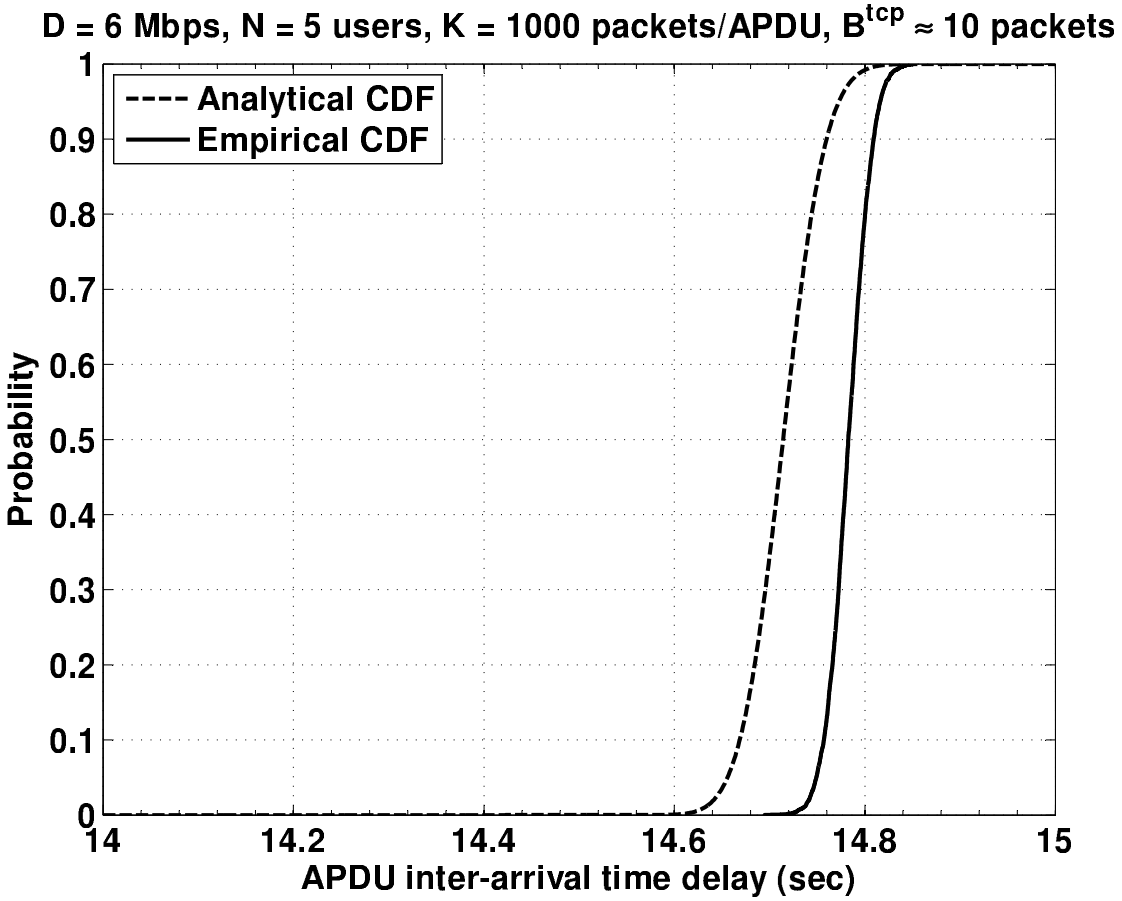

Figure 3.7: The empirical and analytical cumulative distribution functions for TCP buffer size of 10 packets, end users of 5, APDU size of 1000 packets, and data rate of 6 Mbps.

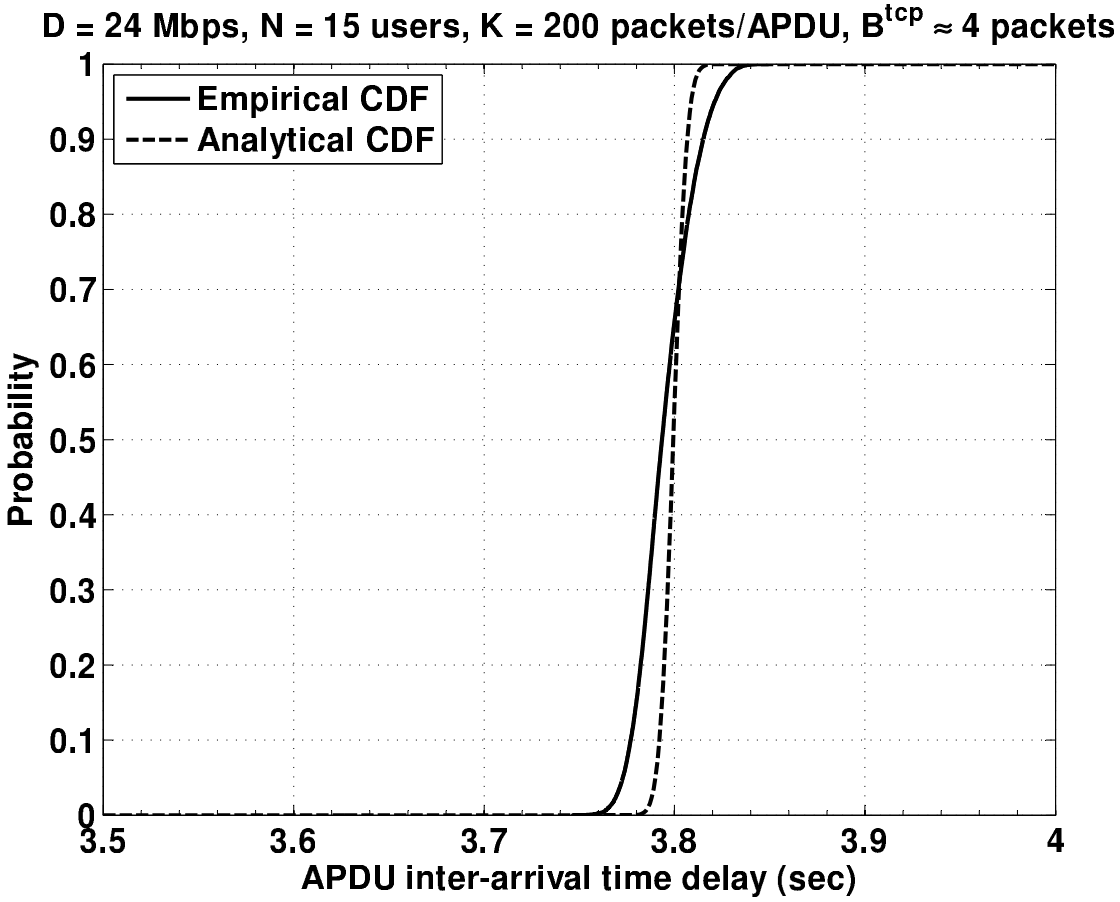

Figure 3.8: The empirical and analytical cumulative distribution functions for TCP buffer size of 4 packets, end users of 15, APDU size of 200 packets, and data rate of 24 Mbps.

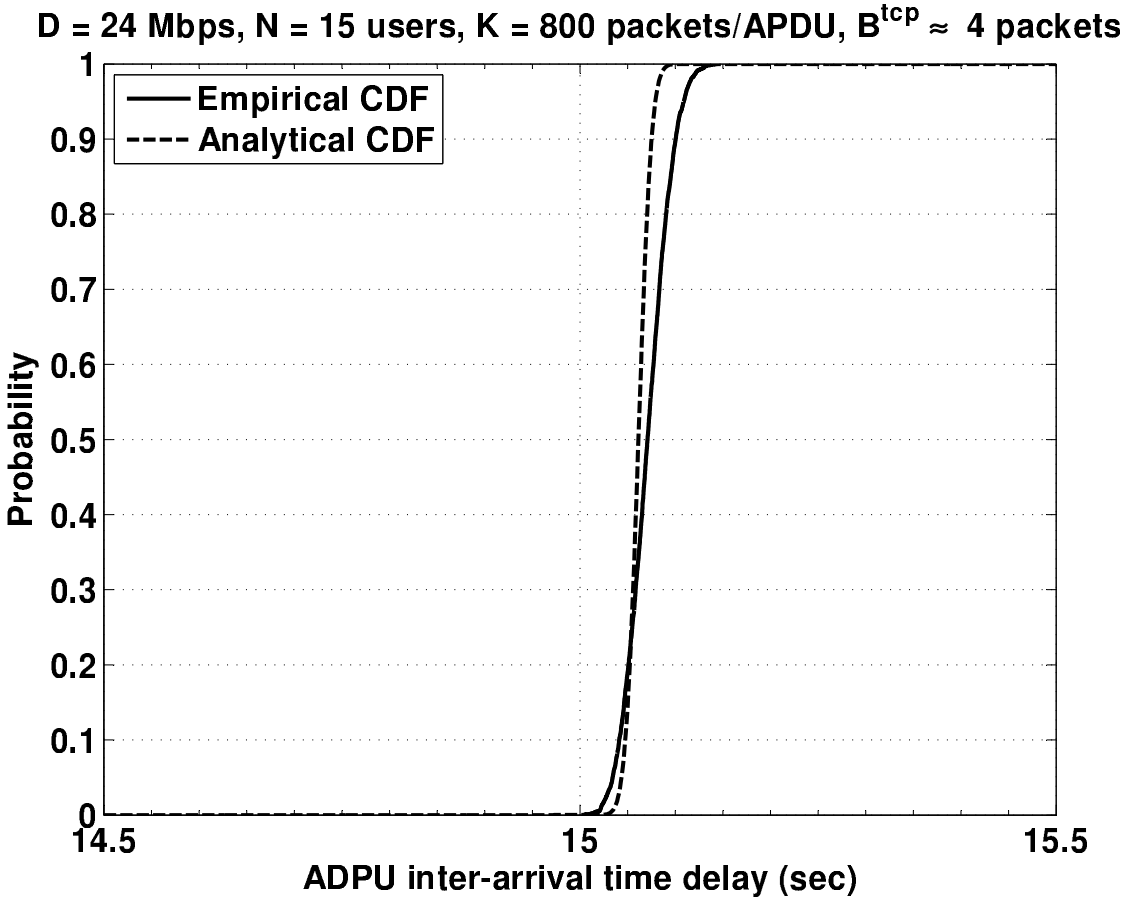

Figure 3.9: The empirical and analytical cumulative distribution functions for TCP buffer size of 4 packets, end users of 15, APDU size of 800 packets, and data rate of 24 Mbps.

Chapter 4

A Test Generation Methodology for Performance Evaluation

In this chapter, a model-based test generation methodology is proposed to evaluate the impact of the interaction of application configuration parameters (ACPs) and network operating parameters (NOPs) on the performance behaviour of mobile networked applications. Figure 4.1 shows the main steps, the inputs, and the expected output of the methodology. At the beginning, we explain how test generation can be formulated as an inversion problem. Then, we discuss the methodology requirements. At the end, the methodology is used to generate test cases to evaluate the performance of two mobile application examples. The first example is a multimedia streaming application that represents the applications of the first group where end user experience is affected by the time delay during service consumption. The second example is a web browsing application that represents the second group of applications where end user experience is affected by the time delay before the service starts. We show that testing the performance of applications of the second group is much easier than applications of the first group. We evaluate the effectiveness of the proposed methodology by comparing the time cost to generate a test suite with random testing.

4.1 Introduction

The core objective of this work is to evaluate the performance of mobile networked applications as perceived by the end user. Therefore, test generation is formulated as an inversion problem, which is solved as an optimization problem. Inversion problem is the

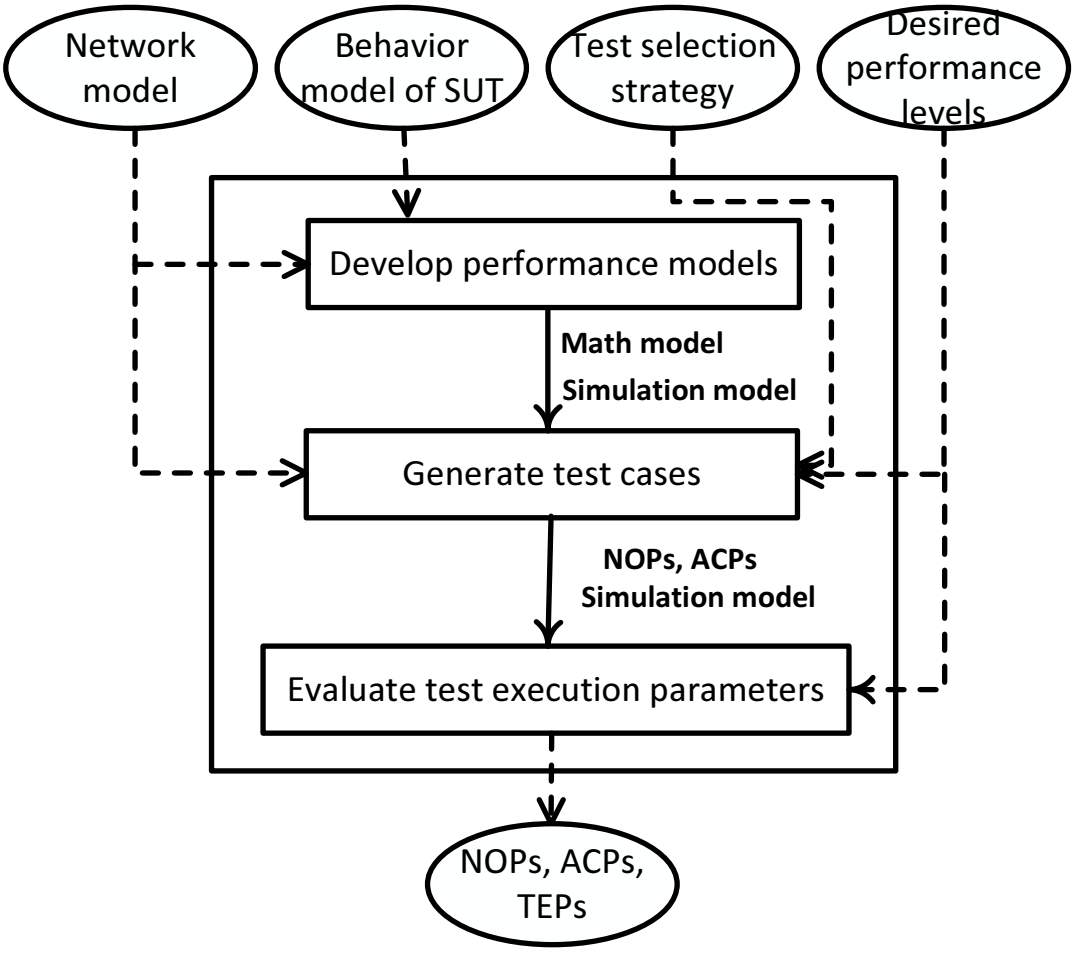

Figure 4.1: The proposed model based test generation methodology.

problem of inferring the *causes* by observing the *effects*. It is quite similar to solving program predicates in white-box approach to determine the test input that leads to the execution of certain program paths. This problem is solved in three main steps: system parametrization, forward modelling, and inverse modelling [109]. In this methodology, 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 inversion problem is casted as an optimization problem because for most models, inverse relationships are not available in closed forms.

4.2 Inputs to The Methodology

To generate test cases, the proposed methodology requires four different artefacts as inputs. In this section, we describe them briefly:

4.2.1 The behaviour model of the SUT

This model describes in terms of ACPs how the application-network interactions impact the performance metric under consideration. For applications of the first group, application-network interactions are clearly observed in the buffering behaviour. Thus, the behaviour model of the SUT for this group should explicitly feature the buffer dynamics. In literature, different diagrams are used to model software dynamics. For example, using UML diagrams, the de facto software modelling language, software behaviours are commonly described using state diagrams, activity diagrams, and/or collaboration diagrams. In this work, we use activity diagrams to describe this model. The outcomes of this task are the behaviour model of the SUT and a set of constraints in terms of ACPs that encode the operational semantics of the application. In Chapter 2, we developed a behaviour model for a mobile streaming application as shown in Figure 2.2. In this chapter, we complete the model by defining the set of constraints that encode the behaviour of the streaming application, and we develop another behaviour model for a web browsing application.

4.2.2 The network model

This model should capture how the wireless network affects the considered performance metric. In general, network model development is mainly determined by the technology

(WiFi or cellular) and the transport protocol (TCP or UDP). According to the system model in Figure 1.4, the application interacts with the network through a basic *request-response* (REQ-RES) mechanism. Thus, the network's impact can be captured by modelling the RES inter-arrival time delay, which is modelled as a random variable. The expected outcomes of network modelling are the probability distribution of this random variable, the network operating parameters (NOPs), and any assumptions and/or conditions that are made during model development. Those conditions and assumptions are required in the second step of the methodology. To obtain the distribution, we employ distribution fitting using the first two moments: the mean and variance [4]. In Chapter 3, we have developed network models for both UDP and TCP protocols. In all these models, the application is assumed to have a wireless network access through a WiFi access point that implements the IEEE 802.11 protocol standard.

4.2.3 Desired performance levels

The methodology requires a set of levels of the performance metric under consideration. Generally, performance metrics are evaluated using statistical measures such as mean, percentage, and probability. In this work, we are interested in application level performance metrics that directly relate to the end user quality of experience (QOE) [48]. For example, the user experience of file transfer applications is assessed using two metrics: goodput and transfer time performance [66]. 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 levels are needed and how they are chosen are addressed in test selection strategies.

4.2.4 Test selection strategies

In general, a test selection strategy encodes the main objectives of the testing process. In black-box testing, all testing activities are carried out in terms of software (or software model) inputs and outputs. Because the input space of software systems is often unbounded, test selection strategies are needed to design a finite effective set of test cases. In this methodology, we propose two coverage criteria: user experience (UE) coverage criterion and user experience and input interaction (UEII) coverage criterion. Now, we explain how to generate test cases using the two proposed coverage criteria:

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 [55] is applied. This technique is mainly used to generate test cases from input space models for functional testing. 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 this work, we apply partition testing to performance metrics, utilizing the fact that the end-user's perception of the performance behaviour is intermittent and it can be characterized by a limited number of categories. These categories are normally called Quality of Experience (QOE) categories.

Given R quality of experience categories, we divide the performance spectrum W_R into R non-overlapped regions $r_1, r_2, ..., r_R$ such that $W_R = \bigcup_i^R r_i$. The number of QOE categories is application type dependent. It also depends on the available resources for the testing process, because test suite size directly relates to R. Then, a performance level l_i is selected for each region such that $l_i \in r_i$, $1 \leq i \leq R$. Thereafter, the corresponding test input vp_1^i, vp_2^i, ..., vp_{n+m}^i for l_i is determined by solving the inversion problem. As an example, we can divide the interval [0,1] of the goodput metric according to the end-user QOE into (say) three regions: if goodput is between 0.9 and 1 ($r_1 = [1, 0.9]$), the quality is *excellent*; between 0.9 and 0.7 ($r_2 = (0.9, 0.7]$), the quality is *good*; and less than 0.7 ($r_3 = (0.7, 0]$), the quality is *poor*. Then, three performance levels are picked, e.g., $l_1 = 0.95$, $l_2 = 0.85$, and $l_3 = 0.5$, to represent the identified categories. The following procedure 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 R test cases

  S1: Partition the performance metric spectrum W_R into R regions $r_1, r_2, ..., r_R$;

  S2: Select a set of performance levels S_l=$\{l_j$: $l_j \in r_j$, $1 \leq j \leq R\}$;

  S3: Generate the test input $vp_1^j, vp_2^j, ..., vp_{n+m}^j$ for each desired performance level $l_j \in S_l$.

Step 3 of this procedure is further explained in the next section. This criterion is achieved if at least one desired performance level from each QOE category is within the generated test cases.

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 application, 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 [123]. Therefore, in the second criterion, 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 the procedure of the UE coverage criterion.

First, we generate a set of R seed test cases T_S using "Procedure 1". This set does cover the whole performance spectrum. Then, to enhance input space coverage, we use the seed test cases 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_{INP} set, where $G \geq 1$. These subsets are constructed such that the parameters in which their interactions are important to cover are grouped into a subset. For example, one may partition the S_{INP} set into two subsets ($G = 2$): $g_1 = S_{ACP}$ and $g_2 = S_{NOP}$. A set of follow-up test cases T_{ij} is generated for every subset g_j and seed test s_i. The parameters' values vp_1, vp_2, ..., vp_{n+m} of the follow-up test cases are determined as follows. The values of the parameters of the g_j subset are determined using a combinatorial coverage metric. The remaining parameters $\{p : p \in S_{INP} - g_j\}$ are assigned the same values of the parameters 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 region of the performance of the seed test case. That is, given the sets T_S, $S_G = \{g_1, g_2, ..., g_G\}$, I (the set of constraints), and a combinatorial metric b, $T_{ij} = Pert(g_j, s_i, I, b)$, $1 \leq j \leq G$, $1 \leq i \leq R$, where $Pert$ realizes the follow-up test generation using the combinatorial coverage metric b. Therefore, test generation to satisfy this 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 test cases T_S. The main steps to satisfy this criterion are summarized in the following procedure:

- **Procedure 2**: Test selection strategy to achieve the UEII coverage criterion
- **Inputs**: R, G, I, and b
- **Outputs**: A test suite T

  S1: Generate the set T_S of R seed test cases 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, and given the set I and the metric b, $T_{ij} = Pert(g_j, s_i, I, b)$. The designed test suite is $T = \bigcup_{i,j} T_{ij} \bigcup T_S$.

The UEII criterion subsumes the UE criterion. Using the two application examples, we show later how to generate test cases to satisfy both test coverage criteria.

4.3 The Methodology Procedure

The methodology consists of three main steps as shown in Figure 4.1. In this section, we illustrate how to execute each step. In the next section, we show that by means of two application examples.

4.3.1 Develop performance models

As mentioned previously, performance models are developed from scratch for applications of the first group only. For applications of the second group, performance models are directly elicited from the network model. Generally, a performance model is any mathematical representation that quantitatively captures the impact of the interaction of the network operating parameters (NOPs) and the application configuration parameters (ACPs) on the performance of the SUT. In literature, many stochastic notations have been used to develop software performance models, such as simulation models, stochastic Petri nets, and queueing networks [36]. Furthermore, different model transformation frameworks have been proposed to guide performance model generation from UML based software design models [21]. In this work, we employ the Markovian modelling framework to develop the performance models. This framework is appropriate especially when the system state is defined by the buffering behaviour of the SUT. We use supplementary variable technique (SVT) [39], [40] to solve the model. This technique is used if the stochastic process that models the behaviour is not Markovian, allowing for more practical interactions between the SUT and the network to be considered.

In this methodology, two performance models are developed: analytical and simulation. The simulation model is used to verify the analytical model and in the test generation process as well. In Chapter 2, we have developed analytical and simulation performance models for a multimedia streaming application. We use the analytical model in the second step of the methodology, while the simulation model is used in the third step. This task requires the network model and the behaviour model of the SUT.

4.3.2 Generate test cases

Each test case is basically a set of ACPs, NOPs, TEPs and the expected performance level. If the SUT is executed with the determined parameters in the three sets (ACPs, NOPs, and TEPs), then the observed performance level is statistically equivalent to the expected performance level if and only if the SUT is correctly implemented from the performance point of view. In this step, network and SUT configuration parameters are determined by solving the inversion problem. 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 mathematical 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 inputs are basically the root that satisfies the following relationship:

$$Perf\_model(p_1, p_2, ..., p_{n+m}) - l_i = 0 \tag{4.1}$$

Where $Perf\_model(...)$ represents the performance model. The roots (NOPs and ACPs values) can be found by reformulating the previous equation as a minimization problem as follows:

$$\textbf{Minimize} \quad |Perf\_model(p_1, p_2, ..., p_{n+m}) - l_i| \tag{4.2}$$

Where $|.|$ is the absolute value operator. We minimize the absolute of the difference to force the solver that the required minimum is zero. In this chapter, we use the analytical 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. The constraints should be formulated in terms of the chosen input parameters. Beside the performance model, this task requires the network model and the desired performance levels. The following procedure summarizes the steps needed to determine ACPs and NOPs:

- **Procedure 3**: Determination of NOPs and ACPs values
- **Inputs**: The performance model, the network model, and the set of desired performance levels S_l
- **Outputs**: Test input values $vp_1^j, vp_2^j, ..., vp_{n+m}^j$ and the set of constraints I

  S1: Define the constraints in terms of input parameters p_1, p_2, ..., p_{n+m};

  S2: $\forall\, l_i \in S_l$, solve the minimization problem of Equation 4.2.

4.3.3 Determine TEPs

Both analytical and simulation performance models can be used in estimating test execution parameters. In this methodology, we opt to use the simulation model. We determine the parameters in two stages. First, we estimate the mean test execution time using a univariate sequential procedure called Law and Carson (abbreviated as L&C) [81]. This procedure is proposed to estimate the number of observations needed to get a point estimator with a pre-specified confidence interval. We estimate the mean run length for the simulation model to reach steady-state and use this value as an estimate for the mean test case execution time. We build a point estimator and a confidence interval so that the estimated value for the considered performance metric is within a pre-specified error from the true value. This procedure relies on batch means evaluations. In sequential procedures, the length of the simulation run is incrementally increased and new observations are added to the sample until an acceptable confidence interval is obtained.

In general, the main issue in simulation output analysis is how to insure the independence of the observations that are used in the estimation and in confidence interval construction. From the statistical point of view, any reliable statistical inference should depend on independent and identically distributed random variables, which is mostly not the case in simulation experiments. If this issue is not handled properly, the sample variance is biased (underestimated) and the resultant confidence interval is not accurate. To test observations independence, L&C procedure relies on estimating the lag-1 correlation of the batch means. If it is less than a certain pre-specified value, it is assumed that the observations are uncorrelated (independent). Then, using those observations, a point estimator and a confidence interval are obtained. If the constructed interval is within a certain pre-specified length, it is accepted and accordingly the length of the simulation run is determined. The precision of estimation is controlled by introducing a tunable parameter γ, called the *relative error* in estimation.

According to the *central limit theorem*, given a sequence of n_r independent and identically distributed random variables, X_1, ..., X_{n_r} of an arbitrary distribution with mean β and variance σ^2, the sample mean $\bar{X}(n_r)$ is normally distributed with mean β and variance σ^2/n_r as n_r goes to ∞ (practically $n_r > 30$). If n_r is relatively small, the sample mean is t distributed. We construct the confidence interval using the t distribution as the normal distribution is a special case of the t distribution. Thus, the confidence interval for the

population mean is given by:

$$\begin{aligned} & \bar{X}(n_r) \pm \delta \,, \\ \text{where} \qquad & \bar{X}(n_r) = \frac{1}{n_r} \sum_{i=1}^{n_r} X_i \,, \\ & \delta = t_{n_r-1,1-\zeta/2} \sqrt{\frac{var(X)}{n_r}} \,, \\ \text{and} \qquad & var(X) = \frac{1}{n_r - 1} \sum_{i=1}^{n_r} [X_i - \bar{X}(n_r)]^2 \,. \end{aligned} \tag{4.3}$$

The parameter δ is known as the *half length confidence interval*, $t_{n_r-1,1-\zeta/2}$ is the upper point for the t distribution with $n_r - 1$ degrees of freedom, and ζ is the significant level. The physical meaning of the confidence interval is as follows: *with* $(1-\zeta)\cdot 100\%$ *confidence, the interval* $[\bar{X}(n_r)-\delta, \bar{X}(n_r)+\delta]$ *does cover the true mean* β *of the population for* $(1-\zeta)\cdot 100\%$ *of the time*. Thus:

$$\beta \in [\bar{X}(n_r) - \delta, \bar{X}(n_r) + \delta] \tag{4.4}$$

Since $\bar{X}(n_r)$ lies at the middle of the confidence interval, hence:

$$\begin{aligned} & 1 - \zeta = \text{Prob}\{|\bar{X}(n_r) - \beta| \leq \delta\} \\ \Rightarrow \quad & 1 - \zeta = \text{Prob}\{\frac{|\bar{X}(n_r) - \beta|}{|\beta|} \leq \frac{\delta}{|\beta|}\} \end{aligned} \tag{4.5}$$

Where $|\bar{X}(n_r)-\beta|/|\beta|$ represents the *relative error* γ in the constructed confidence interval. The meaning of the last equation is: given n_r samples X_1, ..., X_{n_r} and the true mean of the population distribution β, we are $(1 - \zeta) \cdot 100\%$ confident that the relative error in sample mean estimation at most is equal to the ratio of the half length confidence interval to the true mean of the distribution.

In the second stage, the rest of TEPs are inferred simultaneously by utilizing the Bonferroni inequality [30]. 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 (ζ_1, ζ_2, ..., ζ_h) for the remaining h TEPs. We construct individual confidence intervals using the Independent Replication Sequential procedure [81]. In few words, a point estimator and a confidence interval are obtained using a pre-specified number of independent simulation run replications. Then, if the relative error γ is below a certain

pre-defined value, it is done; otherwise, an extra simulation run replication is achieved and a new point estimator and confidence interval are made using the updated sample. The procedure continues until the condition for the relative error is satisfied. The following procedure summarizes the steps needed to estimate TEPs:

- **Procedure 4**: Determine TEPs using the performance 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 the estimated mean of the test case execution time $\hat{T}_x$ and the confidence interval $CI(\hat{T}_x)$;

  S2: Choose the values for ζ_1, ζ_2, ..., ζ_h so that $\sum_{i=1}^{h} \zeta_i = \zeta$;

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

4.4 Using The Proposed Methodology

In this section, we apply the proposed methodology on two representative mobile application examples: a multimedia streaming application and a web browsing. In both applications, we assume the last hop to the end user is through a wireless connection using a WiFi hotspot that implements the IEEE 802.11 protocol.

4.4.1 Test generation for a multimedia streaming application

In this application example, multimedia frames are assumed to be transferred using UDP protocol, where each frame fits in a single UDP packet. The considered performance quality is the smoothness of the streaming as seen by the end user. 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.

Behaviour model of the SUT

The behaviour model of the application is defined in Figure 2.2 in Chapter 2. The ACPs (B, M, and L) are defined as an integer number of multimedia frames. The semantics of

the SUT introduce the following two constraints (**Procedure 3.S2**):

$$\begin{aligned} M &\leq B, \\ L &\leq M - 1. \end{aligned} \tag{4.6}$$

The high watermark cannot be higher than the buffer size, and the low watermark cannot be equal or higher than the high watermark.

Wireless network model

The end-user is streaming via a WiFi hotspot. The network model is developed in Chapter 3. We assume all the fluctuations in the wireless channel and the queueing effects of the different routers along the path from the server to the client device manifest as a time delay. That is, packet loss is negligible. The probability distribution of the frame inter-arrival time delay is matched with a Hyper-Erlang distribution. The network impact is captured by three operating parameters (NOPs): data rate (D), the mean rate of frame arrival into the access point (λ) per user, and the number of end users (N) connected to the AP. The data rate according to the IEEE 802.11 a/g standard can be in one 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 mobile devices that ranges from 4 to 30. The parameter λ is the only continuous. Using the upper and lower bounds of N and D and the constraints imposed by the network model, we bound λ between 10 and 416 $packet/sec$.

Performance models

Analytical and simulation performance models are developed in Chapter 2.

Given the network model, the behaviour model of the SUT, and the performance models, we can start applying the methodology procedure. We apply Procedure 3 to determine the test input. In **Procedure 3.S1**, to determine the objective function, the set of desired performance levels S_l should be specified. This task is part of the test selection criteria and we discuss it when **Procedure 3.S3** is applied.

In multimedia streaming, the mean encoding rate at the server is set according to the end-user device characteristics. Consequently, it is assumed that the mean arrival rate to the end user $1/E_r$ (Equation 3.7) is equal to the mean decoding rate (μ) [83]. That is:

$$\mu = \frac{1}{E_r} \tag{4.7}$$

Including this constraint does exclude the scenarios where the bottleneck is in the computing capability of the mobile device. We are only interested in scenarios where degraded end user experience is mainly due to degraded network quality. Solving Equation (4.7) 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:

$$\frac{k1}{\mu} \leq E_r \leq \frac{k2}{\mu} \tag{4.8}$$

where $k1$ and $k2$ are parameters introduced to control the width of the closed interval around $1/\mu$. By doing so, we relax the non-linear equality constraint into two non-linear inequalities that are easier to deal with (if $k1$ and $k2$ are both set to 1, the equality constraint (4.7) is reproduced). Another constraint that should be taken into consideration is that the traffic intensity ρ at the AP should be less than 1. Otherwise, the buffer at the AP will build up infinitely:

$$\rho < 1 \tag{4.9}$$

Therefore, the optimization problem has five constraints given by Equations (4.6), (4.8), and (4.9). Now, we show how test cases are generated using the two proposed coverage criteria:

i) Test input generation using the 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 as a performance metric, three different end-user experiences are reported ($R = 3$) [90]. If the probability of empty buffer state is less than 2%, the video quality is high; between 2% and 15%, the quality is medium; and for more than 15%, the quality is considered poor. Therefore, we divide the performance behaviour according to the reported three regions. Then, we select a desired level for each region $\{l_1 = 0.01, l_2 = 0.05, l_3 = 0.2\}$. Solving the minimization problem for each performance level (**Procedure 3.S3**), the corresponding network and SUT parameters' values are determined as shown in Table 4.1. The buffer size B is bounded between 10 and 40 frames, the mean of the decoding rate μ is 30 fps, and the parameters $k1$ and $k2$ are assigned as 0.75 and 1.25, respectively.

Table 4.1: Test cases to satisfy UE coverage criterion. D is in Mbps.

l_i	B	M	L	D	λ	N
0.01	34	34	4	18	131.965	7
0.05	38	31	7	32	162.8702	8
0.2	24	7	2	6	98.9693	4

ii) Test input generation using the UEII coverage criterion

We consider the three generated test cases that are listed in Table 4.1 as test seeds T_S (**Procedure 2.S1**). We utilize the combinatorial coverage metric *each-choice* [55] to enhance the input space coverage. We represent the only continuous parameter λ by the following 42 discrete values [10,20,30,...,410,416]. For **Procedure 2.S2**, we choose to cover the interaction of S_{ACP} and S_{NOP} independently (i.e., G=2 and g_1=S_{ACP}, g_2=S_{NOP}). For g_1 subset, we apply each-choice coverage for high watermark (M) and low watermark (L) only, since the playback buffer size (B) does not directly affect the system output. That is, the parameters B, D, λ, and N are kept fixed on seed parameters' values. The same procedure is applied for g_2 subset.

To automate the process, we can benefit from the available combinatorial test generation tools. Unfortunately, we cannot use anyone of them directly, because the type of constraints in our case is more complex than what is supported in the available tools. To mitigate this issue, we first find out the permissible parameter values that satisfy the constraints and then we apply the combinatorial testing criterion. This approach enables us to use the available tools without any modifications, but sometimes, it might be expensive. The combinatorial tool ACTS v3 [27] is used in this process.

Applying the procedure for the g_1 subset, we get 9 (T_{11}), 40 (T_{12}), and 33 (T_{13}) test cases for the performance regions $(0.15, 1]$, $(0.02, 0.15]$, and $[0, 0.02]$, respectively:

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

$$\begin{aligned} T_{12} = \{&(6,5),(7,6),(8,6),(9,4),(38,37),(38,36),(38,35),(37,34),(38,33),\\ &(33,32),(34,31),(38,30),(30,29),(29,28),(28,27),(27,26),(26,25),\\ &(25,24),(24,23),(23,22),(22,21),(21,20),(20,19),(19,18),(18,17),\\ &(17,16),(16,15),(15,14),(14,13),(13,12),(12,11),(11,10),(10,9),\\ &(38,8),(37,6),(36,5),(35,4),(35,4),(34,3),(33,2),(32,1),(31,7)\} \end{aligned}$$

$$T_{13} = \{(10,9),(11,10),(12,11),(13,12),(14,13),(15,14),(16,15),(17,16),$$
$$(18,17),(19,18),(20,19),(21,20),(22,21),(23,22),(24,23),(25,24),$$
$$(26,25),(27,26),(32,27),(33,28),(30,29),(33,30),(34,31),(33,32),$$
$$(34,33),(33,8),(32,7),(31,6),(30,5),(29,3),(28,2),(34,1),(34,4)\}$$

The first element of each tuple is M and the second is L. The remaining parameters' values are fixed on test seed values. For the g_2 subset, using the same procedure and test seeds, we get 5 (T_{21}), 3 (T_{22}), and 3 (T_{23}) test cases for the regions $[0, 0.02]$, $(0.02, 0.15]$, and $(0.15, 1]$, respectively:

$$T_{21} = \{(6\text{M}, 30, 13),(32\text{M}, 50, 26),(32\text{M}, 100, 13),(32\text{M}, 130, 10),(32\text{M}, 260, 5)\}$$

$$T_{22} = \{(54\text{M}, 110, 15),(54\text{M}, 150, 11),(54\text{M}, 330, 5)\}$$

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

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. Some redundancy is observed in the generated follow-up test cases. Therefore, the designed test suite T according to the UEII criterion is the union of the sets T_{11}, T_{12}, T_{13}, T_{21}, T_{22}, T_{23}, and the set of seed test cases T_S.

Determining TEPs

In the third step, test case execution parameters (TEPs) are determined. For the running example, each test case is a streaming session with certain configuration parameters. To efficiently execute each test case, the length of the streaming session and the size of the multimedia file should be determined. We cannot conclude the file size from the streaming session time length, since the player rate is not deterministic. Since we have two TEPs parameters only, we do the estimations without the need to use Bonferroni inequality (**Procedure 4.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 4.S1**) and the mean file size $\hat{\mathtt{F}}_s$ (**Procedure 4.S3**) so that the estimated probability of the empty playback buffer state ($\hat{l}$) 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 listed in Table 4.1. The augmented test cases are shown in Table 4.2. As shown, the confidence interval for the test execution time $\mathtt{CI}(\hat{\mathtt{T}}_x)$ is somewhat wide. It can be narrowed by increasing the number of replications. We gauge the adequacy of the estimated simulation time by controlling the width of the confidence interval $\mathtt{CI}(\hat{l})$ through

Table 4.2: The augmented set of test cases. $\hat{T}_x$, D, and $\hat{F}_s$ are measured in `minutes`, `Mbps`, and `MB`, respectively.

l_i	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]

the parameter γ. As expected, test case execution time is test case dependent. Moreover, as l increases, the required time to reach steady-state decreases.

4.4.2 Test generation for a web browsing application

Web browsers are normally using TCP as a transport protocol. Each web page consists of a main object and a set of embedded objects of different sizes. In this work, we assume the total web page size is basically an integer number of TCP packets. The considered performance metric is the mean waiting time to download a web page. As for multi-streaming application, we start by defining the behaviour model and the network model, then, we apply the proposed methodology to generate test cases.

Behaviour model of the SUT

In this application example, we assume that the end user is in a browsing session in which the mean size of a web page is W_b. In most modern browsers, a web page can be downloaded using multiple parallel TCP connections. We model multiple TCP connections that each browsing user has as a multiple mobile nodes that are connected to the WiFi access point. That is, given U_b browsing users that can utilize up to C_p parallel TCP connections each, we have at most $U_b \times C_p$ mobile nodes. This requirement introduces the following equality constraint that links NOPs to ACPs:

$$U_b \times C_p = N \tag{4.10}$$

where N is one of the NOPs that represents the number of mobile nodes connected to the WiFi AP. Thus, each browsing user is equivalent to C_p mobile nodes. Because of the long-term fairness property of the IEEE 802.11 protocol, we assume that each connection equally likely downloads $\frac{W_b}{C_p}$ packets. Thus, the mean waiting time to browse a web page of

size W_b TCP packets using C_p parallel TCP connections is equivalent to the mean waiting time to browse a web page of size $\frac{W_b}{C_p}$ TCP packets using one TCP connection, which corresponds to the value given by Equation 3.33 in Chapter 3. Therefore, the following constraint should be satisfied:

$$K = \frac{W_b}{C_p} \tag{4.11}$$

where K is another network operating parameter that corresponds to the size of APDU in TCP packets. Thus, we have three ACPs: the number of browsing users U_b, the web page size W_b, and the number of parallel TCP connections C_p.

Wireless network model

The network model is developed in Chapter 3 with four NOPs: the data rate D, the number of mobile nodes N, the TCP buffer size B^{tcp}, and the size of application data unit K in TCP packets per mobile node. The model is developed with the assumption of heavy traffic analysis in which the traffic intensity is assumed to be fixed on 0.99. Therefore, the network model does not introduce any explicit constraints.

Performance models

Since the considered performance metric is the mean waiting time to download a web page, we can reuse the analytical expression of Equation 3.33 as a performance model.

To solve the optimization problem, we need to define the bounds for the ACPs and NOPs (**Procedure 3.S1**). The parameters N and D are as defined in multimedia streaming example. The size of APDU (K) is bounded between 10 and 1000 TCP packets. The size of the TCP buffer (B^{tcp}) is left unbounded since it is an output parameter that depends on other NOPs. For the ACPs, each browsing user can browse using up to 5 parallel TCP connections. We can have up to 30 browsing users (U_b) at the same time. The web page size (W_b) is bounded between 10 and 1000 TCP packets as K. Therefore, the optimization problem is constrained by the two equality constraints of Equations (4.10 and 4.11).

i) Test generation using UE coverage criterion

Regarding the end user experience of web browsing, we divide the spectrum of the performance metric under consideration into 3 regions (**Procedure 1.S1**). If the mean waiting time to download a web page is less than 3 seconds, the experience is excellent. If

Table 4.3: Test cases to satisfy UE coverage criterion. D is in Mbps.

l_i	W_b	U_b	C_p	D	N	K	B^{tcp}
1.5	300	5	2	48	10	150	5
5	384	9	2	18	18	192	3
15	528	13	2	9	26	264	2

Table 4.4: The augmented set of test cases. $\hat{T}_x$ and D are measured in `seconds` and `Mbps`, respectively.

l_i	W_b	U_b	C_p	D	N	K	B^{tcp}	$\hat{T}_x$	$CI(\hat{T}_x)$
1.5	300	5	2	48	10	150	5	2615	$[1.9944 \times 10^3, 3.2356 \times 10^3]$
5	384	9	2	18	18	192	3	4729	$[3.9302 \times 10^3, 5.5278 \times 10^3]$
15	528	13	2	9	26	264	2	14810	$[1.0446 \times 10^4, 1.9175 \times 10^4]$

it is between 3 and 10 seconds, it is good. If it is greater than 10 seconds, the experience is considered bad. Then, we select the following 3 desired performance levels $\{l_1 = 1.5, l_2 = 5, l_3 = 15\}$. Solving the inversion problem for each performance level, the corresponding values for ACPs and NOPs are listed in Table 4.3 (**Procedure 3.S3**).

ii) Test generation using UEII coverage criterion

Test case generation to satisfy UEII criterion is exactly the same as explained in the previous multimedia streaming example. Therefore, the details are not included.

Determining TEPs

All the necessary information to execute the designed test cases is available except the time length of the browsing session. Given the expected mean waiting time l_i to browse a web page, we can build a point estimator $\hat{\mathtt{T}}_x$ and a confidence interval $\mathtt{CI}(\hat{\mathtt{T}}_x)$ of the browsing session time length using L&C procedure (**Procedure 4.S1**). The values for γ and ζ are as defined in the previous example. Table 4.4 shows both the mean and the confidence interval of the browsing session length for each test case listed in Table 4.3. The number of replications is 10.

4.5 Evaluation of The Methodology

We use random testing as a baseline to evaluate the effectiveness of the proposed test generation methodology. Figure 4.2 shows how random testing is normally used to generate a test suite of size $R \times Q$, where R is the number of performance regions, Q is the number of required test cases per region, and C is the coverage criterion. The set S holds the generated test cases. In this section, we use the phrases test configurations and test cases interchangeably. The test configuration t_c is basically a set of values of ACPs and NOPs. Because the implementation of the SUT is not available, the shown procedure is not directly applicable. Therefore, we modify the procedure as shown in Figure 4.3. We use the developed performance model instead of the SUT's implementation to evaluate the performance behaviour l_c of the configuration t_c. To anticipate the incurred cost of random test generation, we keep track of the following types of test configurations:

- Invalid executable test configurations (IETCs),
- Invalid non-executable test configurations (INTCs),
- Valid-and-useful test configurations (VTCs),
- Valid-but-not-useful test configurations (VNTCs).

The 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) or both the network and SUT (invalid NOPs and ACPs), the configuration is not executable. We assume 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 because 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 considered a valid-but-not-useful test configuration.

To estimate the incurred cost of generating a test suite of size $R \times Q$ using random testing, we design an experiment with R and Q that can take values from one to three. Thus, we have in total nine test generation scenarios. We repeat the experiment for each scenario 10 times. The obtained results for UE criterion are shown in Table 4.5. The

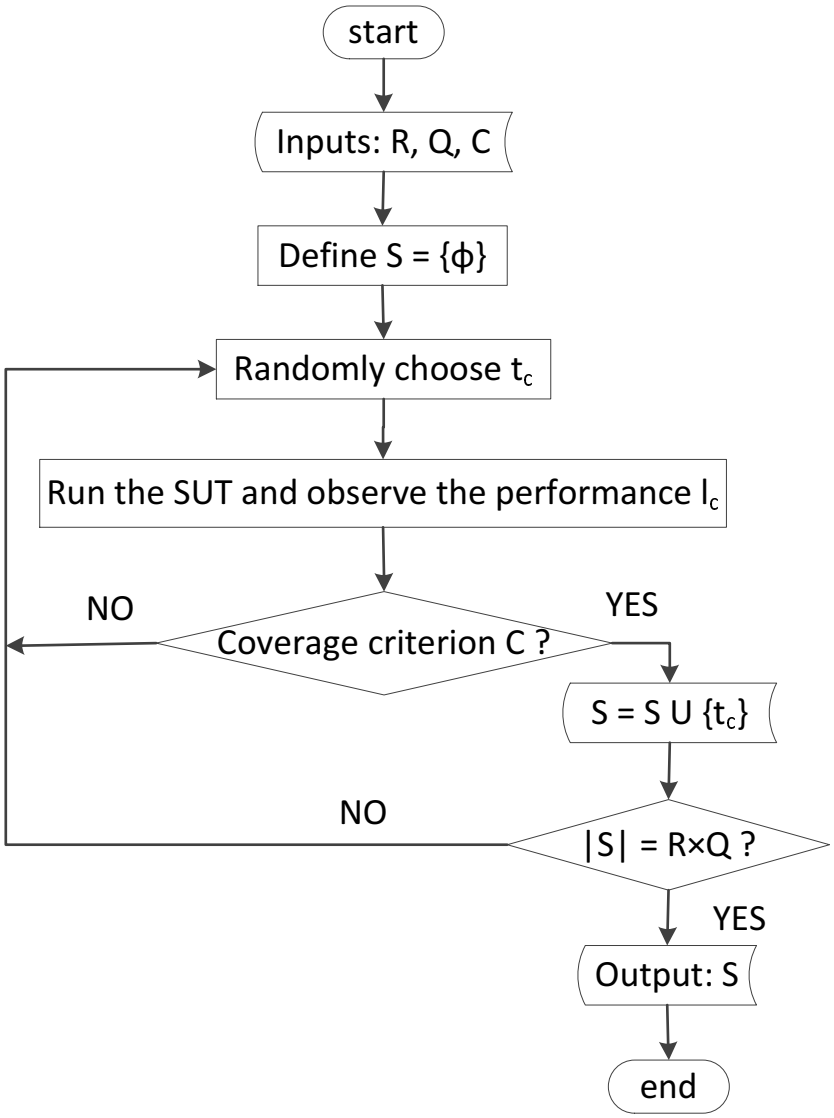

Figure 4.2: The flowchart of test generation using random testing.

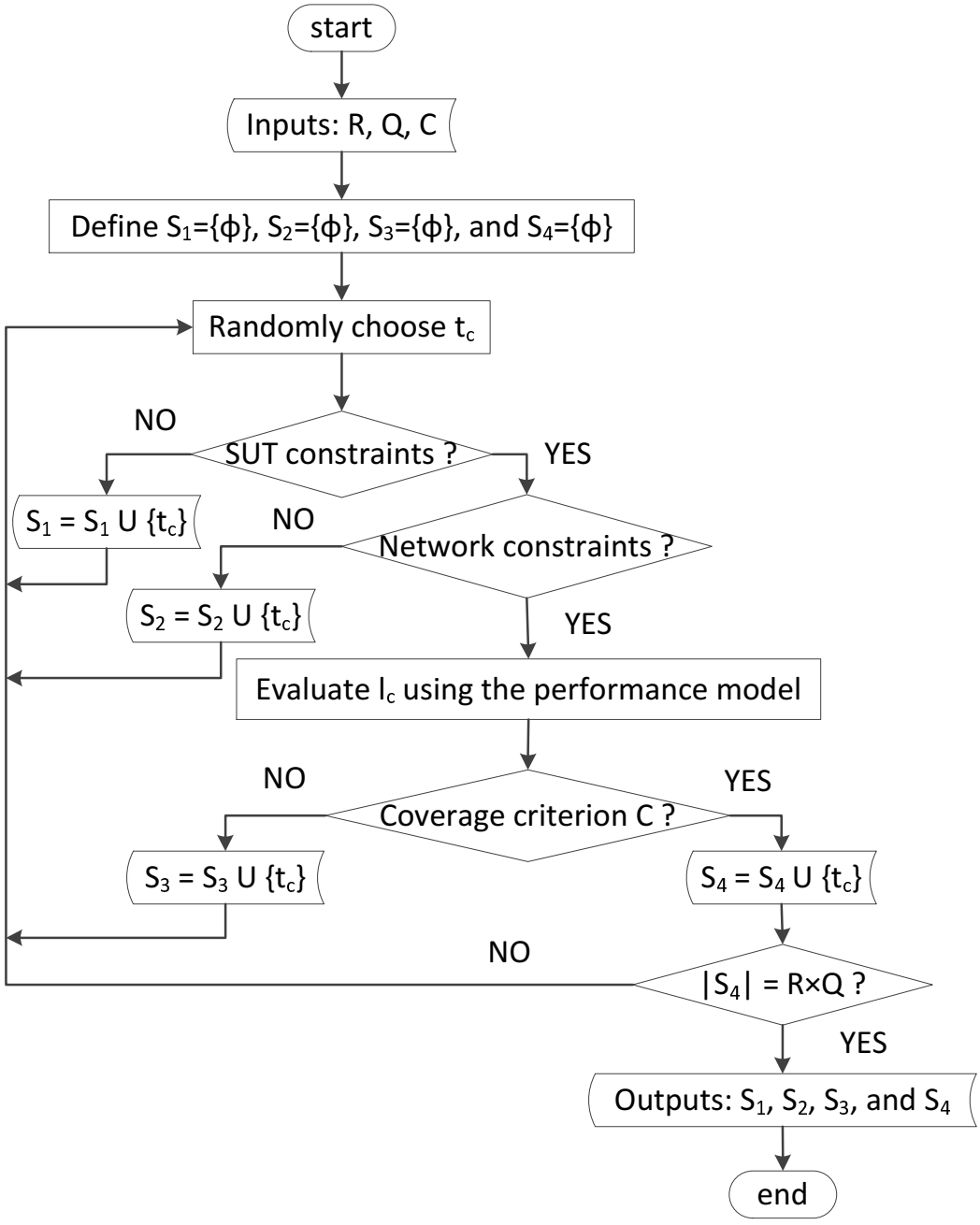

Figure 4.3: The modified flowchart of test generation using random testing. The sets S_1, S_2, S_3, and S_4 are the sets of INTCs, IETCs, VNTCs, and VTCs, respectively.

multimedia streaming application is used in this evaluation. 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 time 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 has been proposed to speed up simulation executions, while real system executions cannot be accelerated. For the second condition, many heuristics based optimization formulations are available in literature that can perform better than the employed optimizer (genetic algorithm). Indeed, within the same optimizer, many strategies can be used to fine tune the performance of the optimizer. In summary, 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 limits as in 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 as shown in Table 4.5. 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 test generation framework (optimization based approach) employs a guided search to figure out the valid test configuration. The results also apply for UEII coverage criterion since UEII builds on (subsumes) UE criterion.

4.6 Applicability of The Methodology

In general, performance degradation can be due to an error in the SUT or an error in the environment (network). As the SUT is assumed to be functionally correct, a performance degradation may happen due to lacking the necessary resources, such as CPU and memory, for a given network quality. Therefore, this methodology is proposed to generate test cases to evaluate the impact of the interaction of network and application configurations on the performance behaviour of mobile networked applications. A networked application is

Table 4.5: The cost of random testing for different scenarios.

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

an application in which user perceived performance is sensitive to the network condition. Because of the semantic gap between software design models and stochastic notations, proposing a unified framework to test different application categories is unattainable.

The main observation for the first group of applications is the high cost of the analytical performance model evaluations compared to the simulation model when the buffer size increases beyond certain limits. To mitigate this issue, we can offload model expensive evaluations to the cloud, or 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 [53],[94]. For real complex systems, it may not be possible to develop any type of analytical models. For such cases, simulation based approach is the only way to do the analysis

In this regard, it is worthy to indicate that using simulation models to predict performance measures is not seamless. In the two application examples, we use the simulation model to infer about test execution parameters. We notice that as the performance metric under consideration became very small, the required simulation run length became very long that in some cases led to out of memory errors. In brief, most statistical procedures depend on covariance evaluations of the performance metric to determine whether the simulation observations are independent or not. If the performance level is close to zero, the covariance value is not defined. Therefore, it is not recommended to assign performance levels that are close to zero.

To enhance the quality of the generated test cases, we propose a UEII coverage criterion to cover the performance spectrum and the input space at the same time. Indeed, network side imposes more complicated constraints than what state of art combinatorial testing

tools can support. To overcome this issue, we exhaustively check all combinations for the imposed constraints. Then, we apply the combinatorial criterion on the combinations that satisfy the constraints. However, this approach may 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. In addition, a redundancy has been noticed when test cases are generated using the UEII test selection criterion. Therefore, effective strategies are needed to minimize redundancy in designed test suites.

4.7 Summary

In this chapter, a model based test generation methodology to evaluate the impact of the interaction of wireless network conditions and application configurations on the performance behaviour of mobile networked applications and thereby on the experience of the end user. The methodology requires four artefacts as inputs and it consists of three main steps. Test generation is formulated as an inversion problem and solved as an optimization problem. To generate an effective set of test cases, two test selection criteria are proposed (UE and UEII). The proposed methodology is used to generate test cases for two representative applications: multimedia streaming and web browsing. The effectiveness of the methodology is evaluated by comparing the incurred time cost to generate a test suite of a specific size with random testing. The results show that our guided search based (optimization based) test generation is noticeably better than random search based test generation.

The main observation is the computationally high cost of performance models for the first group of applications. Moreover, test generation to satisfy UEII coverage criterion may lead to redundant test cases. Therefore, in the next chapter, we combine the proposed test generation methodology with metamorphic testing to minimize the cost of performance models. Furthermore, we propose a metric to evaluate redundancy and adopt a strategy to identify and minimize redundant test cases.

Chapter 5

The Modified Test Generation Methodology Using Metamorphic Testing

In Chapter 4, we proposed a model based test generation methodology to evaluate the impact of wireless network-application interactions on end user experience of mobile networked applications. However, to model realistic software systems, the developed performance models are often computationally expensive. Therefore, in this chapter, we modify the proposed test generation methodology to minimize this cost, utilizing a well-known testing technique called metamorphic testing [99, 31]. First, we give a brief introduction to metamorphic testing. Second, we introduce the modified methodology. Third, we use it to generate a set of test cases for an application example. At the end, we evaluate the effectiveness of the modified methodology in comparison with the methodology proposed in Chapter 4 using two evaluation metrics: time cost and redundancy.

5.1 Test Generation Using Metamorphic Testing

In model based testing, different types of models are developed and used to generate test cases and to infer the required test oracles. In performance testing, analytical and/or simulation models are often used to infer the performance behaviour of each test scenario. However, for some software systems, performance models might be computationally expensive. Therefore in this chapter, we utilize metamorphic testing to minimize the incurred

cost by reducing the number of performance model evaluations that are needed in the test generation process.

Metamorphic testing is a technique that is proposed to alleviate the test oracle problem [99, 31]. By utilizing certain inherent properties of the system under test (SUT), test cases are generated and verified without the need to know in advance their individual expected outputs. These properties are normally called metamorphic relations. To explain the idea, assume we want to test a software code that implements the *sine* trigonometric function [115]. We know from college level mathematics that $\sin(\theta) = \sin(\pi - \theta)$. Therefore, without the need to know the exact value of $\sin(\theta)$ (test oracle), we can utilize this rule (metamorphic relation) to test the code.

Test generation using metamorphic testing presumes the availability of one or more test cases that are commonly called seed test cases. The expected output of seed test cases may not be known. Seed test cases are generated using any test generation technique, such as random testing. In this work, seed test cases are generated using the methodology proposed in Chapter 4. Then, using generated seed test cases, follow-up test cases are generated and verified using the specified metamorphic relations.

In general, a defect in the SUT is detected using metamorphic testing if the observed output of the seed and corresponding follow-up test cases do not confirm with the metamorphic property that is used in follow-up test generation. That is, there is no need to have the expected output in advance for each individual test case (test oracle). In this work, test oracles are assumed to be available but expensive due to the high cost of performance model evaluation. In this elaboration, we name the approach when metamorphic testing is used with our proposed test generation methodology as the modified test generation methodology.

In the proposed methodology, we develop a performance model to capture the performance behaviour from the end user point of view, and we develop a network model to capture the variation in the wireless network quality. The main observation is the computationally high cost of the performance model (simulation or analytical) especially for the first group of applications, compared to the network model. The input network model is far less complicated than the performance model and it is also independent from the SUT, making it as a good candidate to be utilized in deriving the necessary metamorphic relations.

In the network modelling process, we match the probability distribution of the random variable that models the waiting time delay using the two analytically developed mean and variance expressions. Therefore, given a set of seed test cases that are generated using our model based test generation methodology, we utilize the mean and the variance expressions

as metamorphic relations to generate follow-up test cases.

5.2 Metamorphic Relations for Performance Testing

In literature different types of metamorphic relations are used to generate test cases. In general, metamorphic relations with isotropic conditions (conditions with equality constraint) are considered more effective in detecting problems than metamorphic relations with inequality constraints [120]. Relations with equality constraints are tighter and more likely to be violated. However, metamorphic relations with equality constraints that are derived using input network models are not productive. Even though the NOPs of the follow-up test cases may seem to be different from the NOPs of the seed test case, the execution profile of the follow-up test cases that are generated using equality metamorphic relations is totally equivalent to that of the seed test case. Thus, metamorphic relations with equality constraints are only applicable if they are derived from the SUT. Therefore in this work, we use metamorphic testing in a way to guarantee generating follow-up test cases that have a diversified performance behaviour compared to the seed test cases.

Therefore, metamorphic relations with inequality constraints are the only available choice in our case. Another feature that makes inequality metamorphic relations in performance testing more applicable is that the end user perceives the performance behaviour of the application intermittently. Since the main objective of this work is to evaluate the performance from the end user point of view, we partition the spectrum of the performance behaviour according to the categories of the end user experience. Consequently, if the category of the performance behaviour of a test case is known, no need to evaluate the performance model to determine the required test oracle.

5.3 The Modified Methodology

Test generation using metamorphic testing consists of two general steps: seed test generation and follow-up test generation. In this work, seed test cases have special requirements that make seed test generation using other test generation techniques, such as random testing, very expensive. The first requirement is that the performance levels of the seed test cases are chosen to fall on the boundaries of the R performance regions. The second requirement is elaborated as follows: given $l^{s2} > l^{s1}$, then, $vp_i^{s2} \geq vp_i^{s1}$ if the parameter $p_i \in S_{NOP} \cup S_{ACP}$ increases (does not decrease) with increasing the performance metric under consideration and $vp_j^{s2} \leq vp_j^{s1}$ if the parameter $p_j \in S_{NOP} \cup S_{ACP}$ decreases (does

not increase) with increasing the performance metric, where l^{s1} (l^{s2}), vp_i^{s1} (vp_i^{s2}), and vp_j^{s1} (vp_j^{s2}) are the expected performance level of the first (second) seed test case, the assigned value for p_i in the first (second) seed test case, and the assigned value for p_j in the first (second) seed test case, $1 \leq i, j \leq n+m$, $i \neq j$, respectively. If $l^{s2} < l^{s1}$, then, $vp_i^{s2} \leq vp_i^{s1}$ if the parameter $p_i \in S_{NOP} \cup S_{ACP}$ decreases (does not increase) with decreasing the performance metric and $vp_j^{s2} \geq vp_j^{s1}$ if the parameter $p_j \in S_{NOP} \cup S_{ACP}$ increases (does not decrease) with decreasing the performance metric. In other words, the second requirement of seed test cases necessitates a monotonic performance model. The two requirements are necessary to ensure that the performance behaviour of the follow-up test cases falls within the targeted region.

The modified methodology is realized by four distinct steps. First of all, the spectrum of the performance behaviour is partitioned into R non-overlapped regions according to the categories of end user experience. Second, the behaviour of each parameter with respect to the performance metric under consideration is determined (directly or inversely proportional). Third, $R-1$ seed test cases are generated. Fourth, follow-up test cases are generated.

For efficient and effective testing process, it is recommended to generate a set of diverse non-redundant test cases. We utilize distance metrics to capture diversity. The objective is to maximize the distance between the seed test case and the follow-up test cases, given that the performance behaviour of the follow-up test cases falls within the required performance region. Thus, follow-up test generation is formulated as a maximization problem:

$$\textbf{Maximize} \quad d(vp_1^s - p_1, vp_2^s - p_2, ..., vp_n^s - p_n) \tag{5.1}$$

where $d(...)$ is the fitness function that captures the employed distance metric, and vp_1^s, vp_2^s, ..., and vp_n^s are the NOPs' values of the seed test case. Since each performance region is bounded by two seeds (except the first and last regions), either one can be used in the objective function formulation. The optimization problem is subject to the condition that the follow-up test cases should remain within the targeted region. We utilize the mean and variance expressions of the network model to formulate the constraints to assure this requirement. Thus, the optimization problem is subject to the following two conditions: The mean and variance of the packet inter-arrival time delay of the follow-up test cases should respectively be less than the mean and variance of the upper seed test case and larger than the mean and variance of the lower seed test case. In addition, the optimization problem is further subject to the constraints imposed by the network model. In summary, the required steps to generate test cases using the modified methodology are listed in the following procedure:

- **Procedure**: Test generation using the modified methodology
- **Inputs**: Network model (metamorphic relations), performance model, number of performance regions R, number of follow-up test cases per region Q, and the employed distance metric.
- **Outputs**: a set of $R \times Q$ test cases.

  S1: Partition the spectrum of the performance behaviour into R regions r_i, r_2, ..., r_R;

  S2: $\forall\ p_i \in S_{NOP} \cup S_{ACP}$, $1 \leq i \leq n+m$, determine $\frac{\partial l}{\partial p_i}$ whether it is positive or negative;

  S3: Generate $R-1$ seed test cases using the proposed methodology;

  S4: $\forall\ r_i$, $1 \leq i \leq R$, solve the maximization problem of Equation (5.1) Q times.

In the last step of the procedure, we repeat solving the optimization problem with the same objective function (using the same seed) to generate Q follow-up test cases for each region. We randomly initialize the optimization solver, relying on that the problem has multiple local optima. Otherwise, we have to generate additional seeds and use them to generate the remaining test cases.

5.4 Using the Methodology

In this section, we use the modified methodology to generate test cases to evaluate the end user experience of a mobile multimedia streaming application. The performance behaviour of the application under test is assumed to be mainly affected by three configuration parameters (ACPs): the size of the playback buffer (B), the high watermark level (M), and the low watermark level (L). The network quality is modelled using three NOPs: the data rate of the wireless connection (D), the mean packet arrival rate at the access point per mobile user (λ), and the number of mobile users connected to the access point (N). That is, each test case is a set of seven values, the values assigned to B, M, L, D, λ, N, and the value of the expected performance level.

The considered performance metric that models the end user experience is the frequency of having re-buffering events. The objective is to evaluate the impact of the interaction of the NOPs and ACPs on the performance behaviour of the application. Tables 5.1, 5.2, 5.3, and 5.4 show the seed and follow-up test cases for $R = 3$ and $Q = 5$, respectively.

Table 5.1: Seed test cases used in follow-up test generation.

π_0	**B**	**M**	**L**	**D**	λ	**N**
0.1023	30	18	13	9M	69.6786	6
0.2044	30	18	13	9M	27.9667	20

Table 5.2: Follow-up test cases for the first region $(0, 0.1023]$.

B	**M**	**L**	**D**	λ	**N**
30	18	13	32M	46.3473	30
30	18	13	24M	44.2485	25
30	18	13	24M	111.1991	10
30	18	13	48M	224.6139	7
30	18	13	6M	12.9677	30

The distance metric used in the optimization formulation (Equation (5.1)) is the Manhattan distance metric. The maximization problem is constrained by the two conditions of Equations (4.8 and 4.9) and by the following four inequality metamorphic relations:

$$\begin{aligned} & E_r^f \leq E_r^{s2}, \\ & V_{ard}^f \leq V_{ard}^{s2}, \\ & E_r^f \geq E_r^{s1}, \quad \text{and} \\ & V_{ard}^f \geq V_{ard}^{s1}, \end{aligned} \tag{5.2}$$

where E_r^f, E_r^{s1}, E_r^{s2}, V_{ard}^f, V_{ard}^{s1}, and V_{ard}^{s2} are the mean packet inter-arrival delay for the follow-up test case, mean packet inter-arrival delay for the first seed test case, mean packet inter-arrival delay for the second seed test case, variance of packet inter-arrival delay for the follow-up test case, variance of packet inter-arrival delay for the first seed test case, and variance of packet inter-arrival delay for the second seed test case, respectively. For the UDP scenario, the mean and variance of the packet inter-arrival time delay are given by Equations (3.7 and 3.9). For this example, the performance level of the second seed test case is higher than the performance level of the first seed test case. The application configuration parameters B, M, and L are all inversely proportional with respect to the performance metric, as the network operating parameters λ and D, whereas N is directly proportional with the performance metric.

Table 5.3: Follow-up test cases for the second region $(0.1023, 0.2044]$.

B	**M**	**L**	**D**	λ	**N**
30	18	13	6M	56.561	7
30	18	13	6M	30.5908	13
30	18	13	9M	93.2225	6
30	18	13	18M	51.6197	18
30	18	13	32M	48.3161	27

Table 5.4: Follow-up test cases for the third region $(0.2044, 0.3069]$.

B	**M**	**L**	**D**	λ	**N**
30	18	13	9M	34.9584	16
30	18	13	9M	139.8372	4
30	18	13	9M	24.3189	23
30	18	13	18M	133.0048	7
30	18	13	18M	66.5024	14

5.5 Evaluation of The Approach

The effectiveness of the modified methodology is compared with the proposed methodology using two evaluation metrics: the incurred time cost to generate a test suite and the redundancy percentage in the generated test suite. Redundant test cases are test cases when executed do not carry any additional information to what is already available about the system under test. In other words, the expected performance behaviour of the redundant test case is equivalent to one of the test cases in the designed test suite. Mathematically, to measure the quality of the designed test suite, we define the percentage of redundancy ($Y\%$) in a test suite as follows:

$$Y\% = \frac{\text{No. of redundant test cases}}{\text{size of the designed test suite}} \times 100\% \tag{5.3}$$

We have two types of redundancy in test suites: explicit and implicit redundancy. A test case is explicitly redundant if it can be identified without the need to know the expected performance behaviour. In this case, the set of parameters' values of the redundant test case is syntactically equivalent to the parameters of one test case in the designed test suite. Sometimes, it is not straight forward to identify equivalent test cases, especially when some

of the parameters are continuous due to rounding errors. A test case is implicitly redundant if it is difficult to identify it without knowing the expected performance behaviour. In this case, the set of parameters' values of the redundant test case is syntactically different but semantically equivalent to one of the test cases in the designed test suite. Although the set of parameters of the implicitly equivalent test cases are not all the same, the expected performance behaviours are equivalent. In this case, the equivalent test cases are presumed to satisfy some non-productive metamorphic properties.

To evaluate the percentage of redundancy in the designed test suite, two additional parameters are introduced: error margin (ϵ) and maximum number of trails (N_t). We also need to identify the non-productive metamorphic relationship. We need an error margin because of the rounding error that we may have when some of the ACPs and NOPs are real continuous. The new generated test case is considered redundant if it does satisfy the non-productive metamorphic relationship within the given error margin (ϵ) with respect to one of the already generated test cases. To generate follow-up test cases using the same seed test case, we repeatedly solve the maximization problem starting from a random initial point, hoping to have a new solution. We stop if solving the optimization problem for the specified maximum number of trails (N_t) does not lead to a new non-redundant test case.

Both metrics (ϵ and N_t) have impact on the incurred time cost to design a test suite and on the percentage of redundancy. Technically, the error margin (ϵ) has impact on false positives (reported non-redundant but it is a redundant test case) and false negatives (considered redundant but it is a non-redundant test case). Therefore, we design an experiment for this purpose. We assume the spectrum of the performance behaviour of the application under test is partitioned into three regions. The experiment has three factors: ϵ, N_t, and the required number of test cases per region (Q). We consider five scenarios for Q: 1, 2, 3, 4, and 5 tests/region. Three different distance metrics are used in formulation of the optimization problem: Euclidean, squared Euclidean, and Manhattan.

5.5.1 Redundancy in the test suite

Tables 5.5, 5.6, and 5.7 show the percentage redundancy in the designed test suite as the number of test cases per region increases from one to five using three different distance metrics as a fitness function and using different values for ϵ and N_t. The reported values are the mean of five experiments. The results with $\epsilon = 0.001$ and $N_t = 10$ are the best, and Manhattan metric shows a slightly less redundancy than the other two metrics. As shown, redundancy increases as the number of test cases per region increases using same seed test cases.

Table 5.5: Redundancy percentages using three distance metrics ($\epsilon = 0.001$ and $N_t = 5$).

	The number of tests per region (Q)				
Fitness function	**Q = 1**	**Q = 2**	**Q = 3**	**Q = 4**	**Q = 5**
Euclidean	0%	10%	13.33%	26.67%	32%
Squared Euclidean	0%	0%	13.33%	26.67%	29.33%
Manhattan	0%	3.33%	13.33%	26.67%	29.33%

Table 5.6: Redundancy percentages using three distance metrics ($\epsilon = 0.005$ and $N_t = 10$).

	The number of tests per region (Q)				
Fitness function	**Q = 1**	**Q = 2**	**Q = 3**	**Q = 4**	**Q = 5**
Euclidean	0%	26.67%	42.22%	56.67%	65.33%
Squared Euclidean	0%	33.33%	40%	56.67%	60%
Manhattan	0%	26.67%	46.67%	55%	62.67%

Table 5.7: Redundancy percentages using three distance metrics ($\epsilon = 0.001$ and $N_t = 10$).

	The number of tests per region (Q)				
Fitness function	**Q = 1**	**Q = 2**	**Q = 3**	**Q = 4**	**Q = 5**
Euclidean	0%	3.33%	8.89%	25%	26.67%
Squared Euclidean	0%	10%	8.89%	20%	29.33%
Manhattan	0%	0%	8.89%	23.33%	25.33%

5.5.2 The incurred time cost

We evaluate the incurred time cost of generating a test suite of a specific size using both the modified and proposed methodologies. Each scenario is repeated five times and we evaluate the 95% confidence interval for the mean time cost to generate the test suite. The results show no significant differences in the incurred time cost among the three distance metrics. Therefore, we compare the time cost for Manhattan metric only with the proposed methodology. Figure 5.1 shows the obtained results. In each scenario, we need two seed test cases to generate the follow-up test cases. As shown, the time cost to generate follow-up test cases is negligible compared to the time cost to generate the two seed test cases. This makes the time cost across the five scenarios almost constant for the modified methodology.

5.5.3 The impact of increasing the number of seed test cases

In all previous experiments, two seed test cases are used to generate follow-up test cases, since the spectrum of the performance region is partitioned into three regions ($R = 3$). This is the minimum requirement for $R = 3$. However, to eliminate the redundancy in the generated test suite, we conjecture that increasing the number of seed test cases would achieve this goal. Nevertheless, the side effect of increasing the number of seed test cases would be the increase in the incurred time cost. Therefore, we design an experiment to evaluate the trade-off between the reduction in redundancy and the increase in time cost when the number of seed test cases is increased. We consider only four scenarios for the number of tests per region (2, 3, 4, and 5), since the test suite generated with two seeds for $Q = 1$ does not show any redundancy.

Table 5.8 shows the percentage of redundancy in the test suite where three seed test cases are used in the follow-up test generation. The error margin is kept fixed on 0.001 while the maximum number of trails takes three different values: 5, 10, and 15. The used distance metric is Manhattan. The results are the mean of nine experiments. As shown, the percentage of redundancy has tremendously reduced by using one extra seed test case. Figure 5.2 shows the mean incurred time cost and the 95% confidence interval for the proposed methodology and modified methodology with two and three seed test cases. The error margin is 0.001 and the maximum number of trails is 10. As the number of tests per region increases, the modified methodology becomes more cost effective compared to the proposed methodology. Although the number of experiments is not comprehensive, the obtained results show that a compromise can be achieved to get a cost effective non-redundant set of test cases.

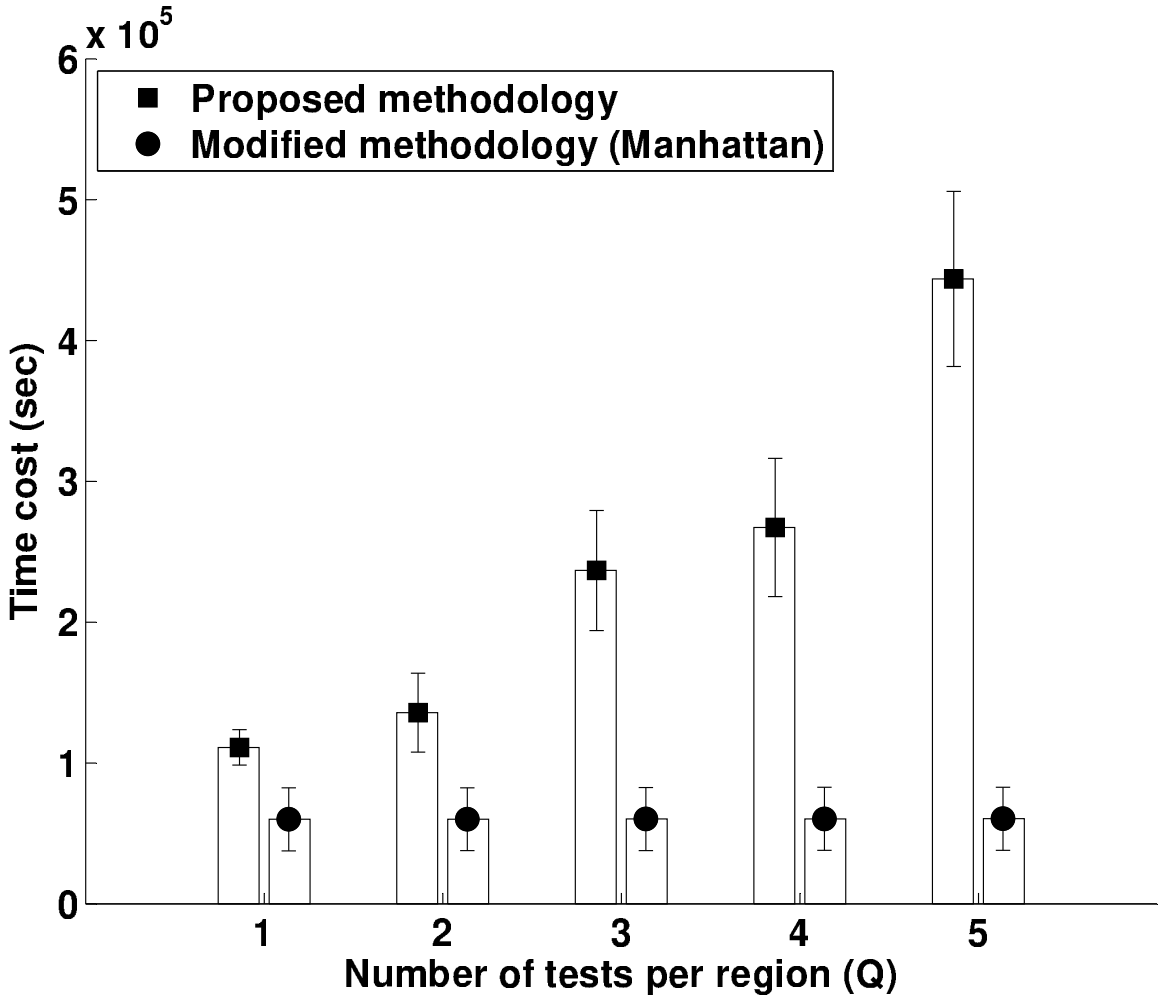

Figure 5.1: The 95% confidence interval of the mean of the incurred time cost to generate a test suite using both the proposed methodology and modified methodology. The values for ϵ and N_t are 0.001 and 10, respectively.

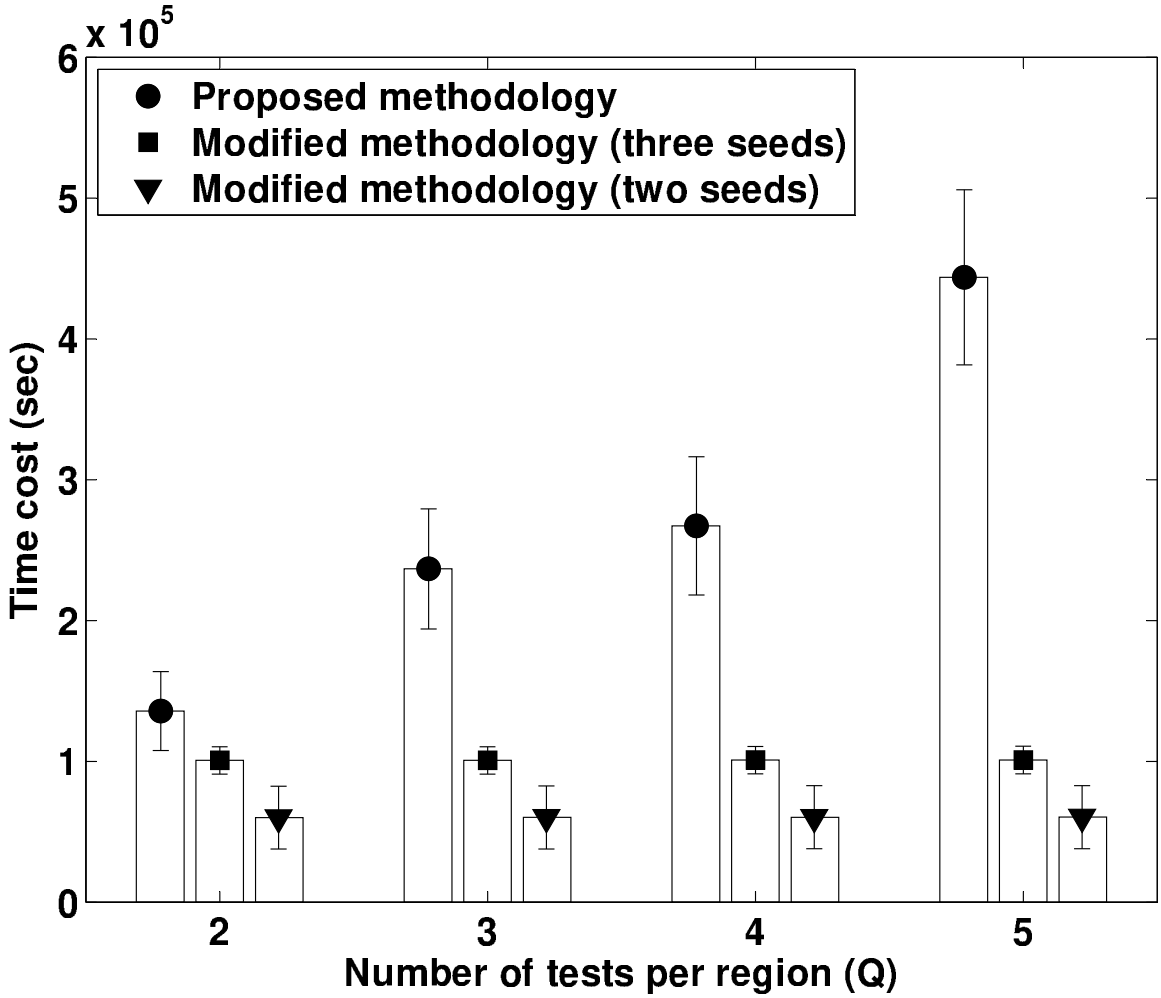

Figure 5.2: The 95% confidence interval of the mean of the incurred time cost to generate a test suite using both the proposed methodology and modified methodology with two and three seed test cases. The values for ϵ and N_t are 0.001 and 10, respectively.

Table 5.8: Redundancy percentages using Manhattan distance metric with $\epsilon = 0.001$.

	No. of tests per region (Q)			
$\mathbf{N_t}$	**Q = 2**	**Q = 3**	**Q = 4**	**Q = 5**
5	0%	1.85%	4.94%	11.36%
10	0%	0%	3.7%	11.11%
15	0%	3.7%	3.7%	5.56%

5.6 Summary

In this chapter, we show the possibility of using metamorphic testing to minimize the cost of generating a test suite for performance evaluation. We generate seed test cases using our proposed test generation methodology. Then, a set of follow-up test cases are generated without the need to invoke the computationally expensive performance model. We show that equality metamorphic relations derived from input models are not productive. Therefore, we use the two expressions of mean and variance of packet inter-arrival time delay as inequality metamorphic relations. We formulate follow-up test generation as a maximization problem. We use three different distance metrics as a fitness function: Euclidean, squared Euclidean, and Manhattan. The obtained results do not show any noticeable preference for one of the metrics over the others. The methodology is used to generate test cases for the multimedia streaming application. The modified methodology is compared with the proposed methodology using two evaluation metrics: incurred time cost and redundancy. The obtained results do prove the advantage of combining our proposed methodology with metamorphic testing to mitigate the computationally high cost of performance models.

In the next chapter, a model based methodology is proposed to evaluate the impact of ACPs and NOPs interactions on another important property, performance robustness, of mobile adaptive and non-adaptive applications.

Chapter 6

Performance Robustness of Mobile Networked Applications

In this chapter, we propose a model based methodology to evaluate the impact of wireless network quality and application configurations on performance robustness of adaptive and non-adaptive mobile networked applications. Non-adaptive applications are applications in which ACPs are configured during the design or before operation phase. In contrast, adaptive applications are applications that have the ability to reconfigure ACPs on the fly to mitigate a degraded network service. Two robustness metrics are proposed (static and dynamic) to evaluate the robustness of the two types of applications. Both proposed metrics require all network-application interactions to be evaluated that may lead to the well known state explosion problem. We propose a strategy to avoid the exhaustive evaluation of the interactions utilizing the monotonicity property of the performance model. Figure 6.1 depicts the proposed methodology. First, we explain the inputs, main steps, and the expected output of the methodology. Then, we apply the methodology to evaluate the performance robustness of a mobile multimedia streaming application. Last, we evaluate the effectiveness of the methodology in comparison with the naive approach where the network-application interactions are exhaustively evaluated.

6.1 The Methodology Inputs

The proposed methodology requires three inputs: the behaviour model of the application, the network model, and the performance model. The specifications of these models are

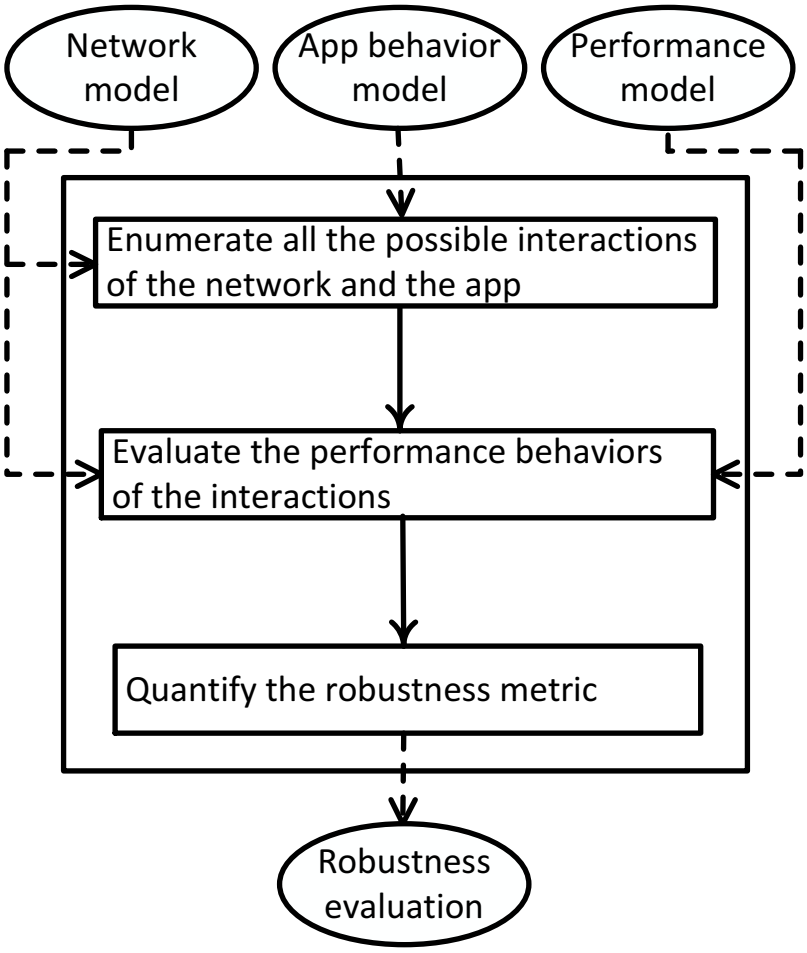

Figure 6.1: The proposed methodology.

explained in Chapter 4. The quality of network service is captured using time delay and the application is assumed to have a wireless network connection through a WiFi access point.

6.2 The Details of The Methodology

The expected output of the methodology is a quantification measure of the performance robustness of the application. To elaborate on that, we need to introduce some notations. According to the perception of the end user, the performance behaviour is characterized by a set of multiple categories. In this chapter, we partition the spectrum of the application performance behaviour into two non-overlapped regions: acceptable region (R_a) and failure region (R_f). If the performance level l_j of the test case t_j falls within the acceptable region (i.e., $l_j \in R_a$), the associated application configuration settings vp_1^j, vp_2^j,..., vp_m^j for $p_i \in S_{ACP}$, are considered robust for the given network condition vp_1^j, vp_2^j,..., vp_n^j, $p_i \in S_{NOP}$; otherwise (i.e., if $l_j \in R_f$), it is considered a failed or non-robust test case.

To achieve the first step of the methodology, we need to identify the set of all valid network scenarios S_N and the set of valid candidate application configurations S_C. Valid combinations are combinations that do not violate the assumptions and conditions made during the development of the network and application behaviour models. The set S_C in the case of non-adaptive applications basically represents the configurations in which their robustness are required to be investigated to identify, for example, the most robust configuration. In the case of adaptive applications, the set S_C can be any subset of ACPs, since the application can dynamically move from one configuration setting to another. Given the set S_C, we evaluate the robustness of the application's configuration $c_j \in S_C$ by investigating the performance behaviour of this configuration under all possible network scenarios. Consequently, we can determine the sets of robust and failed test cases for this configuration. The static robustness of the application configuration c_j is determined as follows:

$$\rho_j^s = \frac{|T_j^R|}{|T_j^R| + |T_j^F|} \tag{6.1}$$

where $|T_j^R|$ and $|T_j^F|$ are the cardinality numbers of the sets of robust and failed test cases for configuration c_j, respectively. The robustness metric ρ^s is bounded between 0 (not robust) and 1 (totally robust). However, this step might be expensive if the performance model evaluation is an intensive operation as in simulation models and/or if the set of configuration-network's interactions is big. To mitigate this cost, we propose an algorithm to limit the number of application-network interactions that need to be evaluated.

Basically, this algorithm utilizes the monotonicity property of the performance model. This property enables sorting the valid network scenarios and thereby reducing the number of performance model evaluations as in the binary search algorithm [76]. The algorithm works as follows. Given the sets S_N and S_C, instead of exhaustively evaluating all the individual interactions of S_N and S_C, we first pick a configuration $c_i \in S_C$ and evaluate the impact of all network scenarios S_N on the performance of c_i. Then, we sort the set of network scenarios S_N according to the results of the first configuration c_i. For the remaining configurations $S_C - \{c_i\}$, we are not interested in knowing the exact performance level of each configuration-network interaction. We are only interested in classifying them into robust or failed interactions.

Using the sorted network scenarios $\hat{S}_N$, the interactions of each one of the remaining configurations with network scenarios has one of two possibilities. The first possibility is that the configuration is robust (failed) for all network scenarios. The second possibility is that the configuration is robust for some scenarios and failed for others. For the first possibility, there is an opportunity to tremendously reduce the number of model evaluations to two or even to one regardless of the size of $\hat{S}_N$ if the first and last network scenarios have the same impact on the performance behaviour (both robust or failed). In this case, there is no need to evaluate the other in between scenarios. If the first and last scenarios have an opposite impact on the performance (the second possibility), we next evaluate the scenario in the middle. If the impact of this scenario is similar to the impact of the first (last) scenario, we do not need to evaluate the impact of the scenarios falling between the first (last) and the middle. Then, the middle scenario becomes the first (last) scenario. We keep evaluating the middle and halving the set $\hat{S}_N$ until the impact of all network scenarios is determined. The steps are summarized by the following procedure:

- **Procedure 1**: Static robustness evaluation
- **Inputs**: The performance model, the constraints imposed by the application behaviour I_a and network models I_n, and the threshold performance level l_{th}
- **Outputs**: The static robustness ρ^s_j, $1 \leq j \leq |S_C|$

  S1: Determine the set of candidate configurations $S_C = \{\bigcup_{i\geq 1}\langle vp^i_1, vp^i_2, ..., vp^i_k, ..., vp^i_m\rangle, p_k \in S_{ACP}, 1 \leq k \leq m\}$ that satisfy I_a;

  S2: Determine the set of valid network scenarios $S_N = \{\bigcup_{i\geq 1}\langle vp^i_1, vp^i_2, ..., vp^i_k, ..., vp^i_n\rangle, p_k \in S_{NOP}, 1 \leq k \leq n\}$ that satisfy I_n;

  S3: Pick a configuration $c_j \in S_C$ and evaluate its robustness against all network scenarios S_N;

S4: Determine the sets T_j^R and T_j^F:
$T_j^R = \{\bigcup\langle c_j, n_i\rangle : n_i \in S_N, l_j \in R_a\}$, $T_j^F = \{\bigcup\langle c_j, n_k\rangle : n_k \in S_N, l_j \in R_f\}$;

S5: Sort the set S_N in ascending or descending order using the results of the previous step and get $\hat{S}_N$;

S6: For each of the remaining configurations, execute the algorithm in Figure 6.2 and determine the sets T_i^R and T_i^F, $1 \leq i \leq |S_C| - 1$ as in S4;

S7: Evaluate the static robustness of the application configuration j (ρ_j^s), $1 \leq j \leq |S_C|$, using Equation (6.1).

By evaluating the robustness score of each candidate configuration, we can identify the most robust configuration.

To evaluate the robustness of adaptive mobile applications, we extend Procedure 1. We assume that the application can dynamically adapt by reconfiguring ACPs when the network quality degrades. Thus, dynamic robustness accounts for the test cases with degraded performance that can be alleviated by reconfiguring ACPs. Therefore, we need to explore the configuration space of each failed test case to figure out new ACPs' values that can tolerate the degraded network condition. This step is achieved for every set of failed test cases T_j^F, $1 \leq j \leq |S_C|$. Then, using the updated sets $\hat{T}_j^F$ and $\hat{T}_j^R$, the dynamic robustness ρ_j^d is evaluated using Equation (6.1). The overall robustness of the system ρ^d is the mean of the robustness of the set S_C:

$$\rho^d = \frac{1}{|S_C|} \sum_{j=1}^{|S_C|} \rho_j^d \tag{6.2}$$

We formulate the problem of figuring out the configuration that mitigates degraded network quality as an optimization problem. ACPs are mostly related to the allocated resources. Therefore, the main objective in this formulation is to find out a configuration that makes the application performance behaviour acceptable for the given network condition with minimal allocated resources. Mathematically, the problem is formulated as follows:

$$\textbf{Minimize} \quad \mathit{Resources}(p_1, p_2, ..., p_m) \tag{6.3}$$

subject to the condition:

$$\mathit{Perf\_model}(p_1, p_2, ..., p_m) \geq l_{th} \tag{6.4}$$

where $\mathit{Resources}(...)$ is the utility function that quantifies the amount of allocated resources, $p_i \in S_{ACP}$, $1 \leq i \leq m$, and $\mathit{Perf\_model}(...)$ is the performance model. This

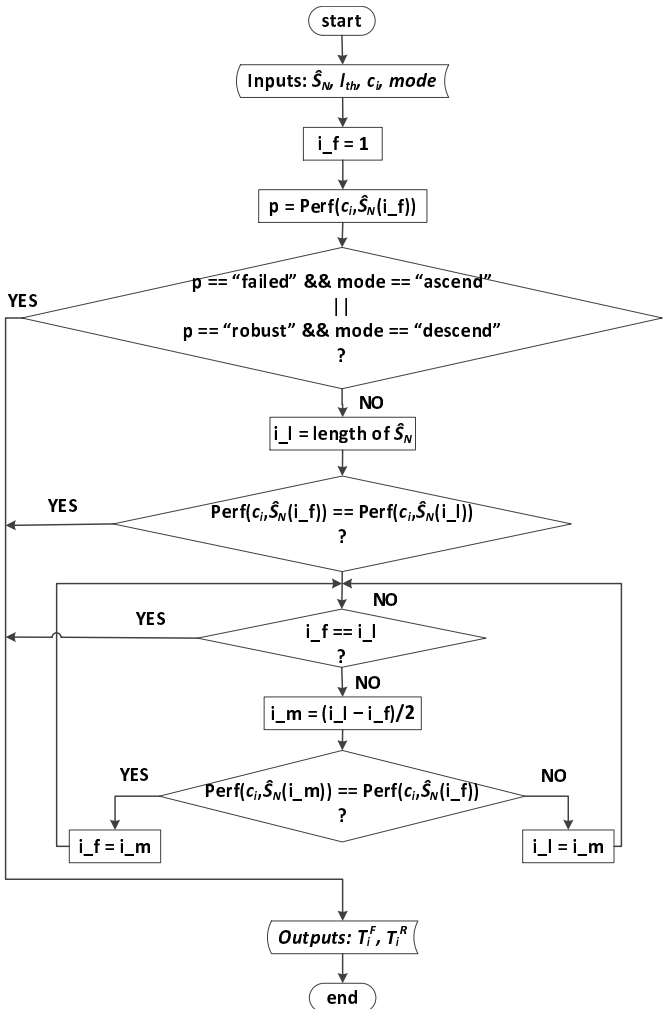

Figure 6.2: The algorithm used to determine the robustness of a configuration c_i against the sorted network scenarios $\hat{S}_N$. The output of the algorithm is the sets of failed and robust test cases T_i^F and T_i^R, respectively. The symbols i_f, i_l, and i_m are the indices of the first, last, and middle network scenarios in the sorted set $\hat{S}_N$. **Perf(**c_i**,**$\hat{S}_N(i)$**)** represents the performance model evaluation and determination of the robustness category of the configuration c_i with the values of NOPs of the network scenario $\hat{S}_N(i)$. The *mode* is either *ascend* or *descend*.

formulation assumes that the performance behaviour is better for higher values. The minimization problem may further be constrained by conditions imposed by the application's semantics. If a solution exists, the values of p_1, p_2,..., p_m constitute, along with the given NOPs' values, a robust test case. If a solution is not possible, this means the degraded performance behaviour due to this network condition cannot be resolved by reconfiguring the application's ACPs. Therefore, to evaluate ρ^d, the following procedure is applied:

- **Procedure 2**: Dynamic robustness evaluation
- **Inputs**: The performance model, I_a, I_n, and l_{th}
- **Outputs**: The dynamic robustness ρ^d

  S1: Determine the sets of robust and failed test cases T_j^R and T_j^F, $1 \leq j \leq |S_C|$, using Procedure 1;

  S2: $\forall i, j$, $1 \leq i \leq |T_j^F|$, $1 \leq j \leq |S_C|$, solve the optimization problem given by Equations (6.3) and (6.4) to identify alternative ACPs' values. If such a configuration setting exists, we delete the test case t_i from the set T_j^F and add the updated version of t_i (with the new ACPs' values) to the set T_j^R. Then, using the updated sets $\hat{T}_j^F$ and $\hat{T}_j^R$, the dynamic robustness ρ_j^d is evaluated using Equation (6.1). The overall robustness of the system ρ^d is evaluated using Equation (6.2).

6.3 Robustness Analysis of A Mobile Streaming Application

In this section, we show how to use the two developed procedures to evaluate the performance robustness of a mobile multimedia streaming application. We characterize the behaviour model of this application, as shown in Chapter 2, by three ACPs: the size of the playback buffer (B), the high watermark (M), and the low watermark (L). The application behaviour introduces two constraints (i.e., $|I_a| = 2$) that are given by Equation (4.6). We assume the streaming is done through a WiFi access point. The network quality is modelled using three NOPs: the data rate of the wireless link (D), the mean rate of frame arrivals at the access point per mobile user (λ), and the number of mobile users connected to the access point (N). The network model also imposes two constraints (i.e., $|I_n| = 2$) that are given by Equations (4.7 and 4.9). The first constraint encodes the requirement that the mean rate of frame arrival to the end user is equal to the mean rate

of frame decoding. The second one encodes the assumption that the traffic intensity at the access point should remain less than 1. The performance metric under consideration is the probability of having zero frames in the playback buffer (l). Using this metric, three different end user experiences are reported [90]. If l is less than 2%, the video quality is high; between 2% and 15%, the quality is medium; and above 15%, the quality is poor.

To evaluate the static robustness using Procedure 1, we partition the performance spectrum into two regions: acceptable region ($0 \leq l < 0.15$) and failed region ($0.15 \leq l$), i.e., $l_{th} = 0.15$. We consider the following three candidate configurations: $c_1 = \langle 16,\ 10,\ 4 \rangle$, $c_2 = \langle 38,\ 31,\ 7 \rangle$, and $c_3 = \langle 34,\ 34,\ 4 \rangle$, where the first, second, and third element of each tuple are the values of B, M, and L, respectively (**Procedure 1.S1**). In **Procedure 1.S2**, we figure out all the possible network scenarios. The data rate D and the number of mobile users N are both discrete, while λ is continuous. The data rate can be any one of the following rates: 6, 9, 12, 18, 24, 32, 48, and 54 Mbps. The number of users is bounded between 4 and 30. Using I_n and the boundary values of N and D, the parameter λ is bounded between 10 and 416 packets/sec. We discretize λ into the following 82 integer values: 10, 15, 20,..., 410, and 415 packets/sec. To determine all the possible network conditions, we exhaustively evaluate all the possible NOPs to determine the combinations that satisfy I_n. Out of the 17712 combinations, only the following 27 are valid:

$$\begin{aligned} S_N = \{&(6\text{M}, 15, 26), (6\text{M}, 30, 13), (32\text{M}, 50, 26), (6\text{M}, 65, 6), (48\text{M}, 75, 21), (48\text{M}, 105, 15), \\ &(32\text{M}, 45, 29), (18\text{M}, 155, 6), (48\text{M}, 175, 9), (18\text{M}, 185, 5), (24\text{M}, 185, 6), (48\text{M}, 225, 7), \\ &(32\text{M}, 65, 20), (48\text{M}, 315, 5), (54\text{M}, 330, 5), (32\text{M}, 100, 13), (32\text{M}, 130, 10), (32\text{M}, 145, 9), \\ &(32\text{M}, 260, 5), (32\text{M}, 325, 4), (54\text{M}, 55, 30), (54\text{M}, 75, 22), (54\text{M}, 110, 15), (54\text{M}, 150, 11), \\ &(54\text{M}, 165, 10), (54\text{M}, 235, 7), (54\text{M}, 275, 6)\} \end{aligned}$$

where the first, second, and third element of each tuple is D, λ, and N, respectively. In **Procedure 1.S3**, we pick c_1 and evaluate the performance behaviour of c_1 using the set S_N. In **Procedure 1.S4**, the sets of robust and failed test cases for c_1 are determined ($|T_1^R| = 19$ and $|T_1^F| = 8$):

$$\begin{aligned} T_1^R = \{&(16, 10, 4, 6\text{M}, 15, 26), (16, 10, 4, 6\text{M}, 30, 13), (16, 10, 4, 32\text{M}, 50, 26), \\ &(16, 10, 4, 6\text{M}, 65, 6), (16, 10, 4, 18\text{M}, 185, 5), (16, 10, 4, 24\text{M}, 185, 6), \\ &(16, 10, 4, 32\text{M}, 65, 20), (16, 10, 4, 54\text{M}, 330, 5), (16, 10, 4, 32\text{M}, 100, 13), \\ &(16, 10, 4, 32\text{M}, 130, 10), (16, 10, 4, 32\text{M}, 260, 5), (16, 10, 4, 32\text{M}, 325, 4), \\ &(16, 10, 4, 54\text{M}, 55, 30), (16, 10, 4, 54\text{M}, 75, 22), (16, 10, 4, 54\text{M}, 110, 15), \\ &(16, 10, 4, 54\text{M}, 150, 11), (16, 10, 4, 54\text{M}, 165, 10), (16, 10, 4, 54\text{M}, 235, 7), \\ &(16, 10, 4, 54\text{M}, 275, 6)\}, \end{aligned}$$

$$T_1^F = \{(16, 10, 4, 48\text{M}, 75, 21), (16, 10, 4, 48\text{M}, 105, 15), (16, 10, 4, 32\text{M}, 45, 29),$$
$$(16, 10, 4, 18\text{M}, 155, 6), (16, 10, 4, 48\text{M}, 175, 9), (16, 10, 4, 48\text{M}, 225, 7),$$
$$(16, 10, 4, 48\text{M}, 315, 5), (16, 10, 4, 32\text{M}, 145, 9)\},$$

In **Procedure 1.S5**, we sort S_N in ascending order using the results of c_1 to get $\hat{S}_N$. In **Procedure 1.S6**, we use $\hat{S}_N$ and the algorithm of Figure 6.2 to determine the robustness of c_2 and c_3. The robust and failed test case sets are determined as follows ($|T_2^R| = 27$, $|T_2^F| = 0$, $|T_3^R| = 27$, and $|T_3^F| = 0$):

$$T_2^R = \{(38, 31, 7, 6\text{M}, 15, 26), (38, 31, 7, 6\text{M}, 30, 13), (38, 31, 7, 32\text{M}, 50, 26),$$
$$(38, 31, 7, 6\text{M}, 65, 6), (38, 31, 7, 48\text{M}, 75, 21), (38, 31, 7, 48\text{M}, 105, 15),$$
$$(38, 31, 7, 32\text{M}, 45, 29), (38, 31, 7, 18\text{M}, 155, 6), (38, 31, 7, 48\text{M}, 175, 9),$$
$$(38, 31, 7, 18\text{M}, 185, 5), (38, 31, 7, 24\text{M}, 185, 6), (38, 31, 7, 48\text{M}, 225, 7),$$
$$(38, 31, 7, 32\text{M}, 65, 20), (38, 31, 7, 48\text{M}, 315, 5), (38, 31, 7, 54\text{M}, 330, 5),$$
$$(38, 31, 7, 32\text{M}, 100, 13), (38, 31, 7, 32\text{M}, 130, 10), (38, 31, 7, 32\text{M}, 145, 9),$$
$$(38, 31, 7, 32\text{M}, 260, 5), (38, 31, 7, 32\text{M}, 325, 4), (38, 31, 7, 54\text{M}, 55, 30),$$
$$(38, 31, 7, 54\text{M}, 75, 22), (38, 31, 7, 54\text{M}, 110, 15), (38, 31, 7, 54\text{M}, 150, 11),$$
$$(38, 31, 7, 54\text{M}, 165, 10), (38, 31, 7, 54\text{M}, 235, 7), (38, 31, 7, 54\text{M}, 275, 6)\},$$

$$T_2^F = \{\phi\},$$

$$T_3^R = \{(34, 34, 4, 6\text{M}, 15, 26), (34, 34, 4, 6\text{M}, 30, 13), (34, 34, 4, 32\text{M}, 50, 26),$$
$$(34, 34, 4, 6\text{M}, 65, 6), (34, 34, 4, 48\text{M}, 75, 21), (34, 34, 4, 48\text{M}, 105, 15),$$
$$(34, 34, 4, 32\text{M}, 45, 29), (34, 34, 4, 18\text{M}, 155, 6), (34, 34, 4, 48\text{M}, 175, 9),$$
$$(34, 34, 4, 18\text{M}, 185, 5), (34, 34, 4, 24\text{M}, 185, 6), (34, 34, 4, 48\text{M}, 225, 7),$$
$$(34, 34, 4, 32\text{M}, 65, 20), (34, 34, 4, 48\text{M}, 315, 5), (34, 34, 4, 54\text{M}, 330, 5),$$
$$(34, 34, 4, 32\text{M}, 100, 13), (34, 34, 4, 32\text{M}, 130, 10), (34, 34, 4, 32\text{M}, 145, 9),$$
$$(34, 34, 4, 32\text{M}, 260, 5), (34, 34, 4, 32\text{M}, 325, 4), (34, 34, 4, 54\text{M}, 55, 30),$$
$$(34, 34, 4, 54\text{M}, 75, 22), (34, 34, 4, 54\text{M}, 110, 15), (34, 34, 4, 54\text{M}, 150, 11),$$
$$(34, 34, 4, 54\text{M}, 165, 10), (34, 34, 4, 54\text{M}, 235, 7), (34, 34, 4, 54\text{M}, 275, 6)\},$$

$$T_3^F = \{\phi\}.$$

Only the configuration c_1 has failed test cases, while all the test cases generated by c_2 and c_3 are robust. Using Equation (6.1), the static robustness ρ_1^s, ρ_2^s, and ρ_3^s of c_1, c_2, and c_3 are 0.703, 1, and 1, respectively (**Procedure 1.S7**). If the application is configured using c_1's ACPs values for example, the application exhibits robust performance behaviour for 70.37% of the time, assuming that the network conditions are equally likely.

If the application can reconfigure on the fly to overcome poor network conditions, the dynamic robustness metric is used to evaluate the robustness of the application's performance. To evaluate ρ^d, we apply **Procedure 2.S2** using the determined failed and robust test cases. For the configuration c_1, we have eight failed test cases (i.e., $|T_1^F| = 8$). To figure out alternate ACPs values for each failed configuration, the minimization problem is formulated as follows:

$$\textbf{Minimize} \quad B \tag{6.5}$$

subject to the conditions:

$$\begin{aligned} Perf\_model(B, M, L) < 0.15 \\ M \leq B, \quad L \leq M - 1 \end{aligned} \tag{6.6}$$

For the application example, the ACPs are related to the memory space allocated for the playback buffer. Therefore, the objective is to determine an application configuration that mitigates the given degraded network condition and moves the performance behaviour to the acceptable region with as minimal buffer size as possible. Thus, the buffer size B represents the utility ($Resources(...)$) that needs to be minimized. The first constraint does ensure that the new ACPs' values (B, M, and L) do make the performance behaviour acceptable. The other two constraints are imposed by the operational semantics of the application (I_a). Using this formulation, we find new ACPs' values for each of the eight failed test cases of T_1^F as shown in Table 6.1. Hence, the updated failed and robust test cases become as follows ($|\hat{T}_1^R| = 27$ and $|\hat{T}_1^F| = 0$):

$$\begin{aligned} \hat{T}_1^R = \{&(16, 10, 4, 6\text{M}, 15, 26), (16, 10, 4, 6\text{M}, 30, 13), (16, 10, 4, 32\text{M}, 50, 26), \\ &(16, 10, 4, 6\text{M}, 65, 6), (10, 10, 8, 48\text{M}, 75, 21), (10, 10, 8, 48\text{M}, 105, 15), \\ &(16, 16, 14, 32\text{M}, 45, 29), (14, 14, 13, 18\text{M}, 155, 6), (10, 10, 8, 48\text{M}, 175, 9), \\ &(16, 10, 4, 18\text{M}, 185, 5), (16, 10, 4, 24\text{M}, 185, 6), (10, 10, 8, 48\text{M}, 225, 7), \\ &(16, 10, 4, 32\text{M}, 65, 20), (10, 10, 8, 48\text{M}, 315, 5), (16, 10, 4, 54\text{M}, 330, 5), \\ &(16, 10, 4, 32\text{M}, 100, 13), (16, 10, 4, 32\text{M}, 130, 10), (16, 16, 14, 32\text{M}, 145, 9), \\ &(16, 10, 4, 32\text{M}, 260, 5), (16, 10, 4, 32\text{M}, 325, 4), (16, 10, 4, 54\text{M}, 55, 30), \\ &(16, 10, 4, 54\text{M}, 75, 22), (16, 10, 4, 54\text{M}, 110, 15), (16, 10, 4, 54\text{M}, 150, 11), \\ &(16, 10, 4, 54\text{M}, 165, 10), (16, 10, 4, 54\text{M}, 235, 7), (16, 10, 4, 54\text{M}, 275, 6)\}, \end{aligned}$$

Table 6.1: The new ACPs values for the set T_1^F. The old ACPs values is (16,10,4)

degraded NOPs	**new ACPs**
(48M, 75, 21)	(10, 10, 8)
(48M, 105, 15)	(10, 10, 8)
(18M, 155, 6)	(14, 14, 13)
(48M, 175, 9)	(10, 10, 8)
(48M, 225, 7)	(10, 10, 8)
(48M, 315, 5)	(10, 10, 8)
(32M, 145, 9)	(16, 16, 14)
(32M, 45, 29)	(16, 16, 14)

$$\hat{T}_1^F = \{\phi\}$$

Therefore, there is a configuration for each failed test case that overcomes the degraded network condition and moves the performance behaviour to the acceptable robust region. Hence, the dynamic robustness for c_1 is 1, and because ρ_2^d and ρ_3^d are both 1, the overall dynamic robustness ρ^d is 1.

6.4 Evaluation of The Methodology

To evaluate the effectiveness of the proposed methodology, we compare the cost of evaluating robustness using our methodology with the cost of exhaustively evaluating all application-network interactions. We call the latter as the naive approach. We use two metrics to quantify the incurred cost: the number of actual model evaluations and time cost. We design an experiment with two controlling factors: the number of candidate configurations and the number of network scenarios. We repeat the experiment 10 times for the proposed approach only, since the naive approach is deterministic. For the proposed approach, we evaluate the 95% confidence interval of the mean for the number of model evaluations and for the time cost.

Figure 6.3 and 6.4 show the time cost and the required number of performance model evaluations, respectively, to evaluate static robustness for different number of candidate configurations. The number of network scenarios is fixed on 27. As shown, the gain in performance is about 3 to 5 times and it increases as the number of candidate configurations increases. Because the time cost of performance model evaluation is configuration

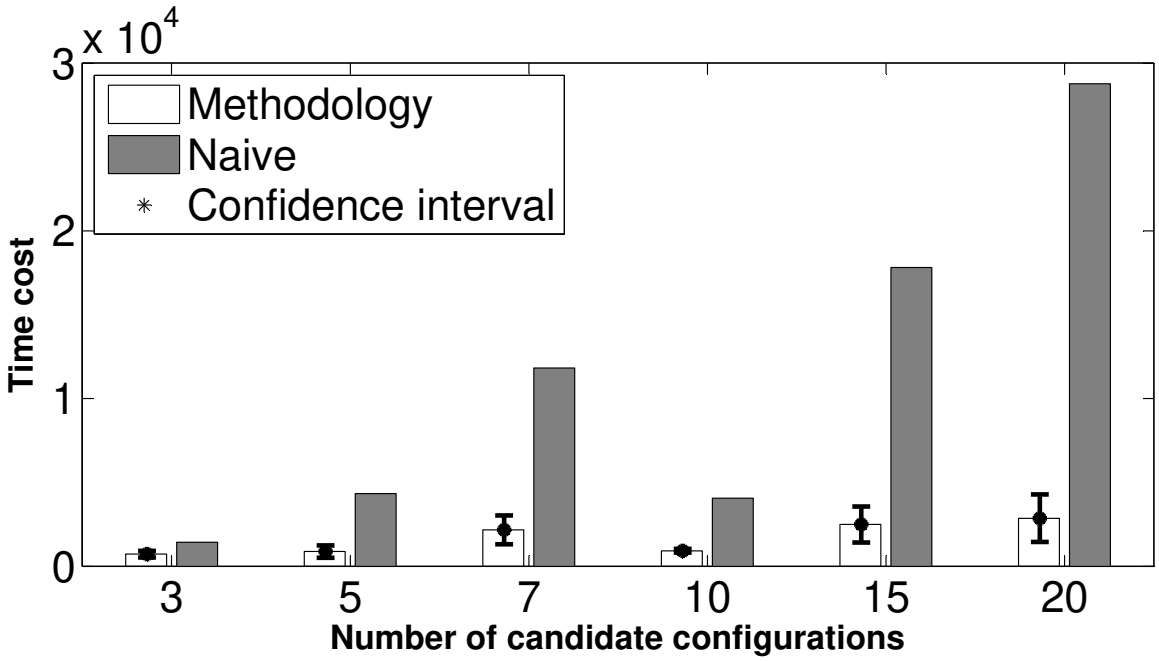

Figure 6.3: The time cost in seconds to evaluate robustness for different numbers of candidate configurations. The number of network scenarios is fixed on 27.

dependent, the time cost to evaluate robustness for 10 configurations is coincidentally less than the time cost for 7 configurations. Since the sorting overhead is also configuration dependent, we pick the first configuration and the mode of sorting (ascend or descend) randomly.

Figure 6.5 and 6.6 show the same metrics, but now the number of candidate configurations is fixed on 5 and the number of network scenarios is changing. The same observations still apply. Although the number of experiments is not comprehensive, we can conclude that the proposed approach overtakes the performance of the naive approach.

6.5 Summary

In this chapter, we propose a model based methodology to evaluate the impact of the interaction of ACPs and NOPs on performance robustness of two types of mobile networked applications: adaptive and non-adaptive. We propose two robustness metrics to quantify performance robustness of both types of applications. To evaluate performance

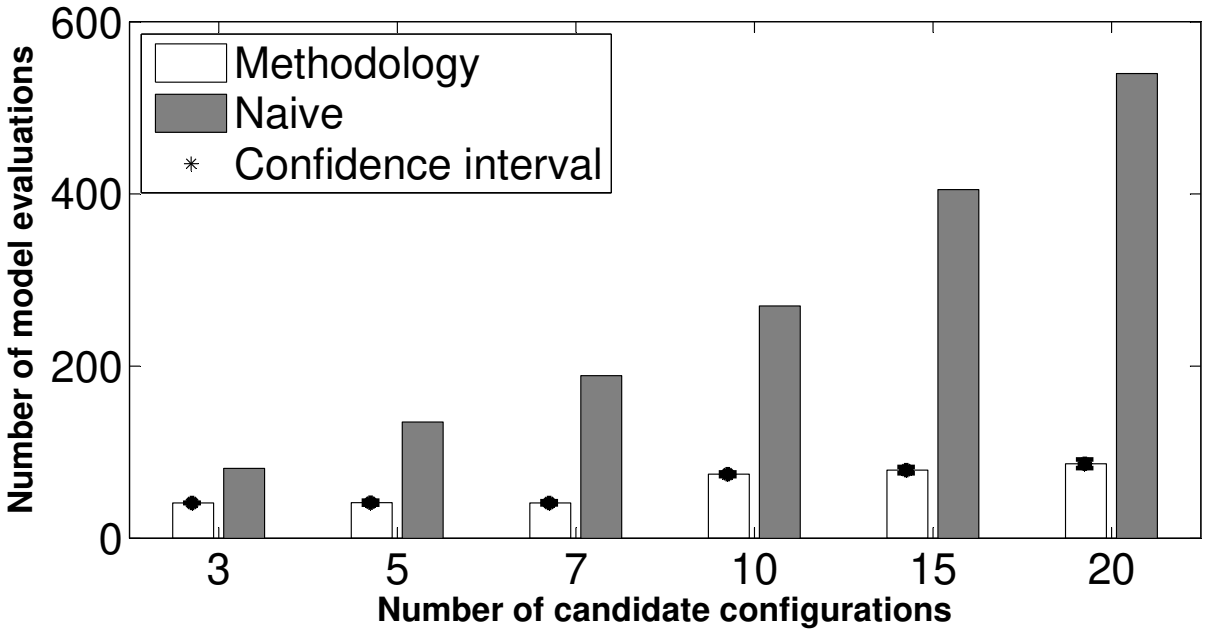

Figure 6.4: The required number of performance model evaluations to evaluate robustness for different numbers of candidate configurations. The number of network scenarios is fixed on 27.

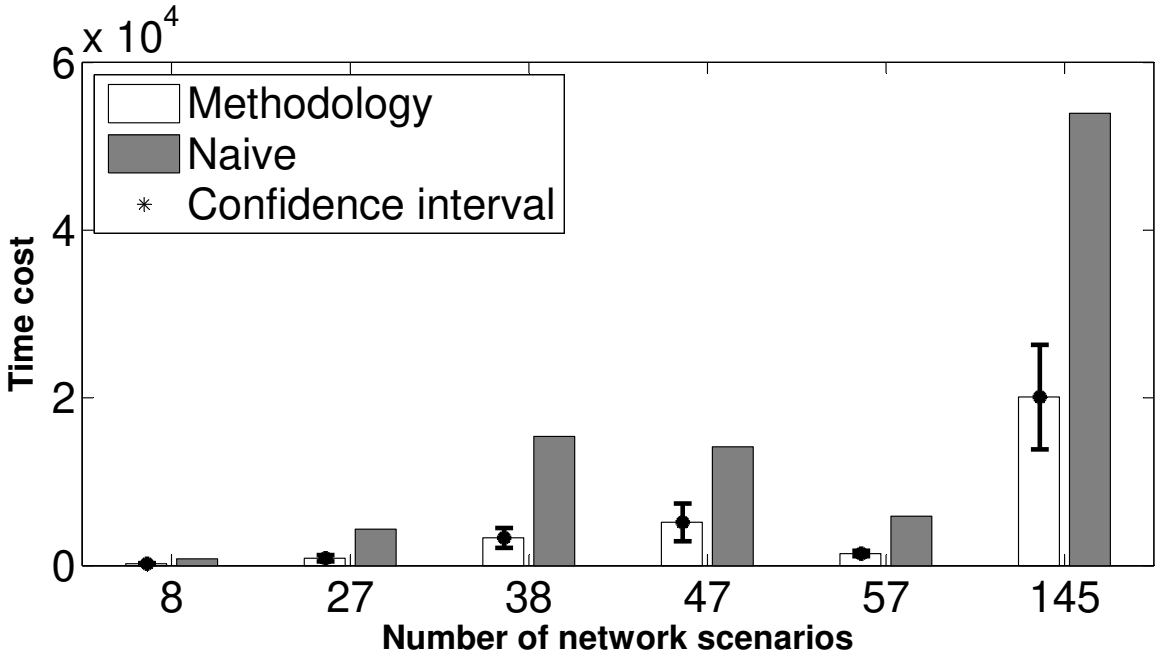

Figure 6.5: The time cost in seconds to evaluate robustness for different numbers of network scenarios. The number of candidate configurations is fixed on 5.

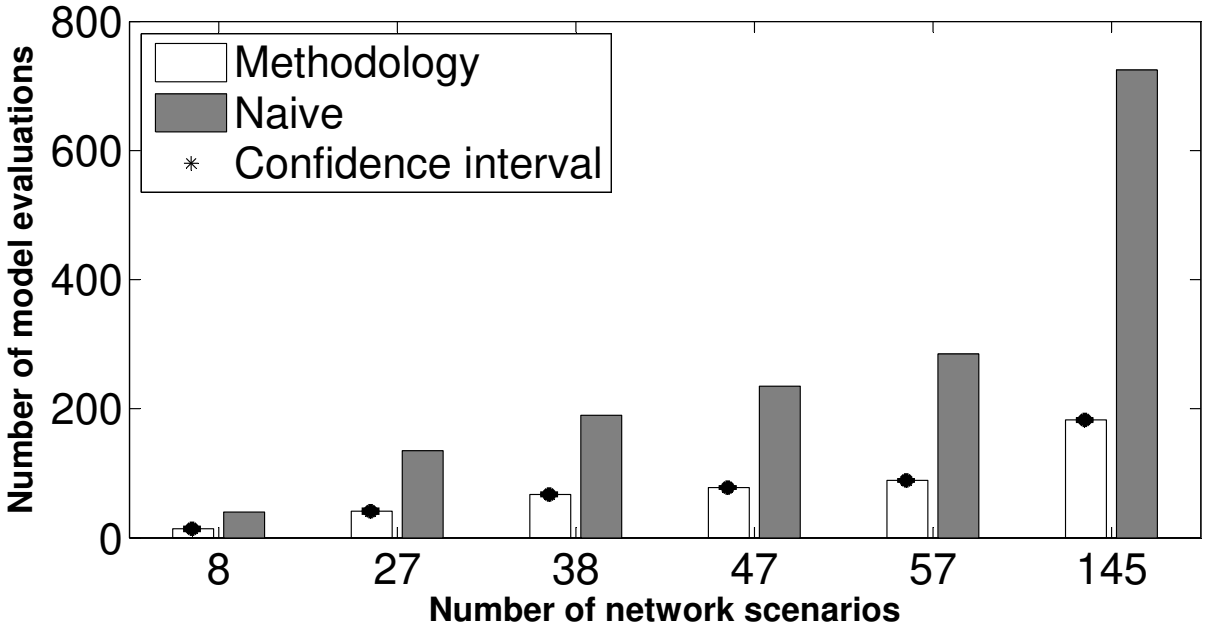

Figure 6.6: The required number of performance model evaluations to evaluate robustness for different numbers of network scenarios. The number of candidate configurations is fixed on 5.

robustness, all network-application interactions have to be evaluated that may lead to the state explosion problem. To overcome this problem, we propose an algorithm utilizing the monotonicity property of the performance model. The methodology is demonstrated using a multimedia streaming application example. For adaptive applications, evaluating the robustness metric needs the evaluation of the ability of the application to reconfigure ACPs in case of degraded NOPs. This problem is formulated as a minimization problem. The effectiveness of the proposed methodology is evaluated using two metrics: the number of performance model evaluations and the time cost. The obtained results show 3-5 times reductions in the incurred cost compared to the naive approach in which all network-application interactions are evaluated.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

Mobile devices nowadays are pervasive because they provide to end users a wide range of services and also a convenient way to stay on-line. Among the factors that affect the quality of provided services is the quality of the wireless network connection. Therefore, in this thesis, we developed methodologies and frameworks to enable software developers to investigate the impact of the interaction of network quality and mobile application configurations on the quality of experience of end users. We focused on an important non-functional property of mobile networked applications that directly relates to the end user experience, which is performance.

First of all, in Chapter 1, we developed an integrated model to understand how faults can lead to a failure in functionality, performance, or both. This model is very important to understand the implication of our first assumption that the software application under investigation is correct from the functionality point of view. We used waiting time delay as a metric to model the quality of the wireless network service. Accordingly, we classified mobile networked applications into two groups: group I and group II. Software applications in which end user experience is mainly affected by the waiting time delay during service consumption was classified under group I, whereas, software applications in which end user experience is mainly affected by the waiting time delay before service consumption was classified under group II.

The core component of any performance evaluation framework is the performance model of the SUT. Therefore, in Chapter 2, an analytical performance model was developed for

mobile networked applications of group I. For group II, we have showed that mobile applications did not need a performance model to be developed from scratch. The developed network model that analytically captured waiting time delay could be reused as a performance model. The performance model for group I applications was developed using Markov chain. To model more realistic network behaviours, the performance model was developed and solved using the supplementary variable technique (SVT). The analytical performance model was validated with simulations. The main observation was the computational high cost of the performance analytical model for applications of group I.

In Chapter 3, input network models were developed. The interaction between a mobile networked application and the network was modelled using a basic request-response mechanism. Therefore, network quality was captured by modelling "response" waiting time delay, which was assumed to be a random variable. A distribution fitting was used to obtain the cumulative distribution function of the waiting time delay using the first two moments: mean and variance. Analytical expressions for the mean and variance of the waiting time delay were developed for applications in which an APDU can fit in a single or multiple packets. The application was assumed to access the network through a WiFi access point that implements IEEE 802.11 g protocol standard. Both TCP and UDP transport protocols were considered. The developed models were validated with simulations using the network simulator NS2.

Software testing is the most used approach to assure the quality of software applications. Therefore, in Chapter 4, a model based test generation methodology was proposed to evaluate the impact of interactions of wireless network quality and application configurations on performance behaviour of a mobile networked application. The methodology requires four different artefacts as inputs: a behaviour model of the SUT, a network model, a test selection criterion, and a set of desired performance levels. The desired performance level is a quantitative measure of the performance metric under consideration. 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. Two test coverage criteria are proposed for this purpose. The first criterion is user experience (UE) and the second is user experience and input interaction (UEII). In UE criterion, test cases are generated to fully cover the identified categories of end-user experience. In UEII criterion, test cases are generated to cover the categories of end-user experience and the interaction of the input parameters (ACPs and NOPs) simultaneously. Input parameters interactions are explored using combinatorial criteria.

The methodology is realized by a procedure of three main steps. In the first step, two performance models (analytical and simulation) to capture the interactions between the

SUT and the network are developed. In the second step, test inputs (ACPs and NOPs) are generated. Test generation was formulated as an inversion problem and solved as an optimization problem. In the third step, test execution parameters (TEPs) are inferred using the performance simulation model. TEPs are the necessary information to concretize the abstract generated test cases and make them executable. Since performance metrics are statistical measures, the execution of each test case is an experiment from the statistical point of view. 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.

Because the network model constrains NOPs by non-linear constraints, state of art combinatorial test generation tools are not applicable directly. To overcome this problem, we constrain the input space first. Then, we apply the combinatorial testing criterion to generate test cases. The generated test cases are validated using the performance simulation model. Two mobile application examples were used to demonstrate the proposed methodology: multimedia streaming (group I) and web browsing (group II).

The effectiveness of the methodology was evaluated by comparing its time cost to generate a test suite of a specific size with random testing. The obtained results have shown the advantage of our test generation approach over random test generation. In random testing, the incurred time cost to generate a test case is inversely proportional with the width of the targeted performance region, while in our optimization based approach, the incurred time cost is independent from the width of the performance region. Furthermore, both test generation and execution in random testing are achieved at the same time using the real implementation of the system, whereas in our model based approach, test generation is achieved first using the developed models. Then, the generated test suite is executed on the real implementation of the system. In general, test execution on the real system is more costly than test execution on system models especially for performance testing. Even with simulation models, test execution can be accelerated, while test execution on real systems cannot be accelerated.

In general, performance models are developed to provide the required test oracles. However, for some software applications (such as group I), performance models, whether analytical or simulation, are computationally expensive. Therefore, in Chapter 5, we modified the test generation methodology that was proposed in Chapter 4 by utilizing a well known testing technique called metamorphic testing. Metamorphic testing is a technique proposed to mitigate the test oracle problem. In this thesis, test oracles are available but expensive due to the high cost of the performance model. Thus, the main idea of the modified test generation methodology was to minimize the number of performance model evaluations by generating a limited number of seed test cases. Then, a set of follow-up test cases are generated using the seed test cases and certain metamorphic relations. We

reused the two light analytical expressions for the mean and variance of waiting time delay as inequality metamorphic relations. We showed that equality metamorphic relations that are derived from input network models are not productive.

We formulated follow-up test generation as an optimization problem. We used three different distance metrics as a fitness function: Euclidean, squared Euclidean, and Manhattan. The objective is to maximize the distance between seed test cases and follow-up test cases to design a non-redundant set of test cases. The obtained results did not show any advantage of one of the distance metrics over the other two. We used the modified methodology to generate a set of test cases for the multimedia streaming example. The effectiveness of the modified methodology was evaluated in comparison with the proposed test generation methodology of Chapter 4 using two evaluation metrics: the incurred time cost to generate a test suite and the percentage of redundancy in the generated test suite. Although the number of conducted experiments was not comprehensive, the obtained results were very promising. The incurred time cost of the modified methodology is far less than the time cost of the proposed methodology but with some redundancy in the designed test suite. The percentage of redundancy increases with the size of the test suite. We showed that the percentage of redundancy can tremendously be decreased by increasing the number of seed test cases that, in turn, may increase the time cost. We showed an effective compromise could be established between time cost and redundancy.

Lastly, in Chapter 6, we proposed a model based methodology to evaluate the impact of interactions of wireless network conditions and application configurations on performance robustness of adaptive and non-adaptive mobile networked applications. The main challenge in evaluating the robustness of configurable software applications is that the number of configurations that need to be evaluated can rapidly increase to unmanageable limits. We utilized the monotonicity property of the performance model to trim the number of application-network interactions. The methodology required three artefacts as inputs: the network model, the application behaviour model, and the performance model. The methodology was realized by three steps. To quantify robustness, two metrics were proposed. The first metric, static robustness, was proposed to quantify the performance robustness of non-adaptive mobile applications, while the second, dynamic robustness, was proposed for adaptive mobile applications. To evaluate the dynamic robustness metric, the ability of the adaptive application to tolerate degraded network conditions has to be evaluated. This problem was formulated as a minimization problem. The proposed methodology was used to evaluate the performance robustness of a mobile multimedia streaming application against wireless network fluctuations. The application example showed the applicability of the proposed methodology and highlighted the challenges/opportunities for further enhancements. The effectiveness of the proposed methodology was evaluated. The obtained

results showed three to five times reduction in total cost compared to the naive approach in which all combinations are exhaustively evaluated.

7.2 Future works

The main objective of the work was to show how performance of mobile networked applications can be evaluated from the end user point of view. We have used a model based software testing approach to achieve this goal. Nevertheless, there is always room for improvement. In the remaining part of this section, we list some of the directions for future works:

- In Chapter 2, we have developed an analytical performance model for group I applications. We have assumed that the arrival time was generally distributed, while the service (play) time was exponentially distributed to make the model tractable within the available resources. Nevertheless, the utilized technique to develop and solve the model, supplementary variable technique, can be used even if both arrival and service times are assumed to be generally distributed. Instead of solving a set of differential equations with one supplementary variable, the performance model is now described by a set of partial differential equations with two supplementary variables.

- In chapter 3, we have developed network models for applications that access network via a WiFi AP that implements IEEE 802.11 g standard and using both TCP and UDP transport protocols. The assumption was that the direction of data transfer is mainly from the Internet to the mobile devices via a WiFi AP. However, to consider mobile applications that send and receive data in both directions, such as voice-over-ip and on-line gaming, the developed network models should be extended to support such applications. Another interesting direction is to develop network models for other types of wireless technology such as cellular networks and Bluetooth.

- In Chapter 4, we compared our proposed test generation methodology with random testing to show the effectiveness of the proposed methodology. The main overhead in random testing is being both test generation and execution are done simultaneously using the real implementation of the SUT. Performance metrics are normally statistical measures in which the time cost of their evaluation is not trivial especially when evaluated using the real implementation. Therefore, model based random testing can be a promising approach to minimize the time cost of using classical random testing in performance evaluation.

- In interaction combinatorial testing, input parameters should have finite number of values. Otherwise, partition testing is used to limit the number of values. The main 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. However, figuring out regions of equivalent points is not always obvious. For example, one of the parameters of the UDP network model is continuous, which is λ, the mean arrival rate of packets to the WiFi AP per mobile user. It is not obvious what are the regions of equivalent points for λ. In Chapter 4, we have sampled λ space irrespectively. Since test generation was formulated as an optimization problem, an interesting point to pursue is to investigate the possibility of using sensitivity analysis to guide the sampling process of the parameter space. The idea is to consider the range in a parameter's value that does not affect the quality of the optimal solution as the region of equivalent points.

References

[1] Fredrik Abbors, Tanwir Ahmad, Dragos Truscan, and Ivan Porres. Model-based performance testing in the cloud using the mbpet tool. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 423–424. ACM, 2013.

[2] Fredrik Abbors and Dragoş Truş Can. Approaching performance testing from a model-based testing perspective. In *Advances in System Testing and Validation Lifecycle (VALID), Second International Conference on*, pages 125–128. IEEE, 2010.

[3] A. Abdrabou and W. Zhuang. Stochastic delay guarantees and statistical call admission control for ieee 802.11 single-hop ad hoc networks. *IEEE Transactions on Wireless Communications*, 7(10):3972–3981, 2008.

[4] Ivo Adan and Jacques Resing. *Queueing Theory: Ivo Adan and Jacques Resing*. Eindhoven University of Technology. Department of Mathematics and Computing Science, 2001.

[5] Abdurhman Albasir, Kshirasagar Naik, Bernard Plourde, and Nishith Goel. Experimental study of energy and bandwidth costs of web advertisements on smartphones. In *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*, pages 90–97. IEEE, 2014.

[6] Attahiru Sule Alfa and TSS Rao. Supplementary variable technique in stochastic models. *Probability in the Engineering and Informational Sciences*, 14(02):203–218, 2000.

[7] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. How do software architects consider non-functional requirements: An exploratory study. In *Requirements Engineering Conference (RE), 2012 20th IEEE International*, pages 41–50. IEEE, 2012.

[8] Thomas Anderson and John C. Knight. A framework for software fault tolerance in real-time systems. *IEEE Trans. on Soft. Eng.*, SE-9(3):355–364, 1983.

[9] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33, 2004.

[10] Alberto Avritzer, Joe Kondek, Danielle Liu, and Elaine J Weyuker. Software performance testing based on workload characterization. In *Proceedings of the 3rd international workshop on Software and performance*, pages 17–24. ACM, 2002.

[11] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *Software Engineering, IEEE Transactions on*, 30(5):295–310, 2004.

[12] Simonetta Balsamo and Moreno Marzolla. A simulation-based approach to software performance modeling. *SIGSOFT Softw. Eng. Notes*, 28(5):363–366, September 2003.

[13] Cornel Barna, Marin Litoiu, and Hamoun Ghanbari. Autonomic load-testing framework. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 91–100. ACM, 2011.

[14] Cornel Barna, Marin Litoiu, and Hamoun Ghanbari. Model-based performance testing (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 872–875. ACM, 2011.

[15] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2015.

[16] Benoit Baudry, Yves Le Traon, and J-M Jézéquel. Robustness and diagnosability of oo systems designed by contracts. In *Proc. of the 7th Int. Software Metrics Symp.*, pages 272–284. IEEE, 2001.

[17] Giuseppe Bianchia, Sunghyun Choib, and I Tinnirello. Performance study of ieee 802.11 dcf and ieee 802.11 e edca. In Benny Bing, editor, *Emerging technologies in wireless LANs: theory, design, and deployment*, chapter 4, pages 63–103. Cambridge University Press, New York, USA, 2008.

[18] Robert V Binder, Bruno Legeard, and Anne Kramer. Model-based testing: where does it stand? *Communications of the ACM*, 58(2), 2015.

[19] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[20] Lionel Briand, Shiva Nejati, Mehrdad Sabetzadeh, and Domenico Bianculli. Testing the untestable: model testing of complex software-intensive systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 789–792. ACM, 2016.

[21] Fabian Brosig, Philipp Meier, Steffen Becker, Anne Koziolek, Heiko Koziolek, and Samuel Kounev. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. *Software Engineering, IEEE Transactions on*, 41(2):157–175, 2015.

[22] Peter Buchholz. *Input Modeling with Phase-Type Distributions and Markov Models: Theory and Applications*. Springer, 2014.

[23] L. X. Cai, Xuemin Shen, J. W. Mark, L. Cai, and Yang Xiao. Voice capacity analysis of wlan with unbalanced traffic. *IEEE Transactions on Vehicular Technology*, 55(3):752–761, 2006.

[24] Javier Cámara, Rogério de Lemos, Nuno Laranjeiro, Rafael Ventura, and Marco Vieira. Robustness-driven resilience evaluation of self-adaptive software systems. *IEEE Transactions on Dependable and Secure Computing*, 14(1):50–64, 2017.

[25] Gerardo Canfora, Francesco Mercaldo, Corrado Aaron Visaggio, Mauro D'Angelo, Antonio Furno, and Carminantonio Manganelli. A case study of automating user experience-oriented performance testing on smartphones. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 66–69. IEEE, 2013.

[26] Yuxiang Cao, Zhi Quan Zhou, and Tsong Yueh Chen. On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions. In *Quality Software (QSIC), 2013 13th International Conference on*, pages 153–162. IEEE, 2013.

[27] Computer Security Resource Center. Advanced combinatorial testing system (acts). http://csrc.nist.gov/groups/SNS/acts/, 2016.

[28] FT Chan, TY Chen, Shing Chi Cheung, MF Lau, and SM Yiu. Application of metamorphic testing in numerical analysis. In *Proceedings of the IASTED International Conference on Software Engineering (SE'98)*, 1998.

[29] WK Chan, Tsong Y Chen, Shing Chi Cheung, TH Tse, and Zhenyu Zhang. Towards the testing of power-aware software applications for wireless sensor networks. In *Int. Conference on Reliable Software Technologies*, pages 84–99. Springer, 2007.

[30] John M Charnes. Analyzing multivariate output. In *Proceedings of the 27th conference on Winter simulation*, pages 201–208. IEEE Computer Society, 1995.

[31] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, 51(1):4, 2018.

[32] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, and Shengqiong Wang. Conformance testing of network simulators based on metamorphic testing technique. In *Formal Techniques for Distributed Systems*, pages 243–248. Springer, 2009.

[33] Tsong Yueh Chen, Fei-Ching Kuo, Wenjuan Ma, Willy Susilo, Dave Towey, Jeffrey Voas, and Zhi Quan Zhou. Metamorphic testing for cybersecurity. *Computer*, 49(6):48–55, 2016.

[34] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, 2009.

[35] IEEE Standards Coordinating Committee et al. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[36] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-based software performance analysis*. Springer Science & Business Media, 2011.

[37] Leandro T Costa, Ricardo M Czekster, Flávio M de Oliveira, Elder de M Rodrigues, Maicon B da Silveira, and Avelino F Zorzo. Generating performance test scripts and scenarios based on abstract intermediate models. In *SEKE*, pages 112–117, 2012.

[38] Domenico Cotroneo, Domenico Di Leo, Francesco Fucci, and Roberto Natella. Sabrine: State-based robustness testing of operating systems. In *Proc. of the 28th IEEE/ACM Int. Conf. on Automated Soft. Eng.*, pages 125–135. IEEE, 2013.

[39] D. R. Cox. The analysis of non-markovian stochastic processes by the inclusion of supplementary variables. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):433–441, 1955.

[40] David Roxbee Cox and Hilton David Miller. *The theory of stochastic processes*, volume 134. CRC Press, 1977.

[41] Maicon B Da Silveira, Elder de M Rodrigues, Avelino F Zorzo, Leandro T Costa, Hugo V Vieira, and Flávio M De Oliveira. Generation of scripts for performance testing based on uml models. In *SEKE*, pages 258–263, 2011.

[42] Valeria Lelli Leitao Dantas, Fabiana Gomes Marinho, Aline Luiza da Costa, and Rossana MC Andrade. Testing requirements for mobile applications. In *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*, pages 555–560. IEEE, 2009.

[43] Luca De Cicco and Saverio Mascolo. An adaptive video streaming control system: Modeling, validation, and performance evaluation. *Networking, IEEE/ACM Transactions on*, 22(2):526–539, 2014.

[44] Massimiliano Di Penta, Gerardo Canfora, Gianpiero Esposito, Valentina Mazza, and Marcello Bruno. Search-based testing of service level agreements. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1090–1097. ACM, 2007.

[45] Almudena Diaz, Pedro Merino, and F Javier Rivas. Mobile application profiling for connected mobile devices. *IEEE Pervasive Computing*, 9(1):54–61, 2010.

[46] Junhua Ding, Tong Wu, Jun Q Lu, and Xin-Hua Hu. Self-checked metamorphic testing of an image processing program. In *Secure Software Integration and Reliability Improvement (SSIRI), 2010 Fourth International Conference on*, pages 190–197. IEEE, 2010.

[47] Karl Doerner and Walter J Gutjahr. Extracting test sequences from a markov software usage model by aco. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 2465–2476. Springer, 2003.

[48] Markus Fiedler, Tobias Hossfeld, and Phuoc Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *Network, IEEE*, 24(2):36–41, 2010.

[49] Borko Furht. *Encyclopedia of multimedia*. Springer Science & Business Media, 2008.

[50] Shiwei Gao, Binglei Du, Yaruo Jiang, Jianghua Lv, and Shilong Ma. An efficient algorithm for pairwise test case generation in presence of constraints. In *Systems and Informatics (ICSAI), 2nd International Conference on*, pages 406–410. IEEE, 2014.

[51] Reinhard German. *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. John Wiley & Sons, Inc., 2000.

[52] Alim Ul Gias and Kazi Sakib. An adaptive bayesian approach for url selection to test performance of large scale web-based systems. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 608–609. ACM, 2014.

[53] Abhijit Gosavi. *Simulation-based optimization: parametric optimization techniques and reinforcement learning*, volume 55. Springer, 2014.

[54] Mark Grechanik, Chen Fu, and Qing Xie. Automatically finding performance problems with feedback-directed learning software testing. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 156–166. IEEE, 2012.

[55] Mats Grindal, Jeff Offutt, and Sten F Andler. Combination testing strategies: a survey. *Software Testing, Verification and Reliability*, 15(3):167–199, 2005.

[56] Yuanyan Gu and Yujia Ge. Search-based performance testing of applications with composite services. In *Web Information Systems and Mining, 2009. WISM 2009. International Conference on*, pages 320–324. IEEE, 2009.

[57] Ralph Guderlei and Johannes Mayer. Statistical metamorphic testing: testing programs with random output by means of statistical hypothesis tests and metamorphic testing. In *7th Int. Conf. on Quality Software (QSIC)*, pages 404–409. IEEE, 2007.

[58] Ralph Guderlei, Johannes Mayer, Christoph Schneckenburger, and Frank Fleischer. Testing randomized software by means of statistical hypothesis tests. In *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*, pages 46–54. ACM, 2007.

[59] Mahbub Hassan and Raj Jain. *High performance TCP/IP networking*. Prentice Hall Upper Saddle River, NJ, 2003.

[60] Mohammad Mahdi Hassan, Wasif Afzal, Martin Blom, Birgitta Lindström, Sten F Andler, and Sigrid Eldh. Testability and software robustness: A systematic literature

review. In *Software Engineering and Advanced Applications (SEAA), 41st Euromicro Conference on*, pages 341–348. IEEE, 2015.

[61] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(1):6, 2013.

[62] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans. on Soft. Eng.*, 40(7):650–670, 2014.

[63] W Henderson. Alternative approaches to the analysis of the m/g/1 and g/m/1 queues. *J. Oper. Res. Soc. Japan*, 15:92–101, 1972.

[64] M Hlynka and T Wang. Comments on duality of queues with finite buffer size. *Operations research letters*, 14(1):29–33, 1993.

[65] T. Hossfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen. Initial delay vs. interruptions: Between the devil and the deep blue sea. In *2012 Fourth International Workshop on Quality of Multimedia Experience*, pages 1–6, 2012.

[66] Mihai Ivanovici and Razvan Beuran. Correlating quality of experience and quality of service for network applications. In Sasan Adibi, editor, *Quality of service architectures for wireless networks: performance metrics and management*, chapter 15, pages 326–351. IGI Global, Pennsylvania, USA, 2010.

[67] Raj Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

[68] Z.M. Jiang and A.E. Hassan. A survey on load testing of large-scale software systems. *Software Engineering, IEEE Transactions on*, 41(11):1091–1118, Nov 2015.

[69] Chuanming Jing, Zhiliang Wang, Xingang Shi, Xia Yin, and Jianping Wu. Mutation testing of protocol messages based on extended ttcn-3. In *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, pages 667–674. IEEE, 2008.

[70] Chuanming Jing, Zhiliang Wang, Xia Yin, and Jianping Wu. A formal approach to robustness testing of network protocol. In *IFIP International Conference on Network and Parallel Computing*, pages 24–37. Springer, 2008.

[71] William Johansson, Martin Svensson, Ulf E Larson, Magnus Almgren, and Vincenzo Gulisano. T-fuzz: Model-based fuzzing for robustness testing of telecommunication protocols. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, pages 323–332. IEEE, 2014.

[72] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. Real challenges in mobile app development. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pages 15–24. IEEE, 2013.

[73] Fikri Karaesmen and Surendra M Gupta. Duality relations for queues with arrival and service control. *Computers & operations research*, 24(6):529–538, 1997.

[74] Sunint Kaur Khalsa and Yvan Labiche. An orchestrated survey of available algorithms and tools for combinatorial testing. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 323–334. IEEE, 2014.

[75] Youngjoo Kim, Okjoo Choi, Moonzoo Kim, Jongmoon Baik, and Tai-Hyo Kim. Validating software reliability early through statistical model checking. *IEEE software*, 30(3):35–41, 2013.

[76] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[77] Philip Koopman, John Sung, Christopher Dingman, Daniel Siewiorek, and Ted Marz. Comparing operating systems using robustness benchmarks. In *Proc. of Reliable Distributed Systems, The 16th Symp. on*, pages 72–79. IEEE, 1997.

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

[79] Peter M Kruse, Jurgen Bauer, and Joachim Wegener. Numerical constraints for combinatorial interaction testing. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 758–763. IEEE, 2012.

[80] Vidyadhar G Kulkarni. *Introduction to modeling and analysis of stochastic systems*. Springer New York, 2011.

[81] Averill M Law. *Simulation modeling and analysis*. McGraw-Hill New York, fifth edition, 2015.

[82] Longshu Li, Yingxia Cui, and Yun Yang. Combinatorial test cases with constraints in software systems. In *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*, pages 195–199. IEEE, 2012.

[83] Mingzhe Li, Mark Claypool, and Robert Kinicki. Playout buffer and rate optimization for streaming over ieee 802.11 wireless networks. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 5(3):26, 2009.

[84] Mikael Lindvall, Dharmalingam Ganesan, Ragnar Árdal, and Robert E Wiegand. Metamorphic model-based testing applied on nasa dat–an experience report. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 129–138. IEEE, 2015.

[85] H. Liu, F. C. Kuo, D. Towey, and T. Y. Chen. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, 40(1):4–22, 2014.

[86] Yepang Liu, Chang Xu, and Shing-Chi Cheung. Diagnosing energy efficiency and performance for mobile internetware applications. *Software, IEEE*, 32(1):67–75, 2015.

[87] Kevin J Ma, Radim Bartos, Swapnil Bhatia, and Raj Nair. Mobile video delivery with http. *Communications Magazine, IEEE*, 49(4):166–175, 2011.

[88] Rivalino Matias, Kishor S Trivedi, and Paulo Romero Martins Maciel. Using accelerated life tests to estimate time to software aging failure. In *21st IEEE Int. Symp. on Soft. Reliab. Eng.*, pages 211–219, 2010.

[89] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. Automated test suite generation for time-continuous simulink models. In *proceedings of the 38th International Conference on Software Engineering*, pages 595–606. ACM, 2016.

[90] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. Measuring the quality of experience of http video streaming. In *12th IFIP/IEEE Int. Symp. on Integ. Network Management and Workshops*, pages 485–492. IEEE, 2011.

[91] Fatih Nayebi, Jean-Marc Desharnais, and Alain Abran. The state of the art of mobile application usability evaluation. In *CCECE*, pages 1–4, 2012.

[92] Alessandro Orso and Gregg Rothermel. Software testing: a research travelogue (2000–2014). In *Proceedings of the on Future of Software Engineering*, pages 117–132. ACM, 2014.

[93] Stacy J Prowell. Using markov chain usage models to test complex systems. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 318c–318c. IEEE, 2005.

[94] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[95] Antoine Rollet and Fares Saad-Khorchef. A formal approach to test the robustness of embedded systems using behaviour analysis. In *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on*, pages 667–674. IEEE, 2007.

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

[97] Nazim Sebih, Franz Weitl, Cyrille Artho, Masami Hagiya, Yoshinori Tanabe, and Mitsuharu Yamamoto. Software model checking of udp-based distributed applications. In *2014 Second International Symposium on Computing and Networking*, pages 96–105. IEEE, 2014.

[98] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés. Metamorphic testing of restful web apis. *IEEE Transactions on Software Engineering*, 0(0):1–1, 2017.

[99] Sergio Segura et al. A survey on metamorphic testing. *IEEE Transactions on software engineering*, 42(9):805–824, 2016.

[100] Sergio Segura, Javier Troya, Amador Durán, and Antonio Ruiz-Cortés. Performance metamorphic testing: motivation and challenges. In *Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER), 2017 IEEE/ACM 39th International Conference on*, pages 7–10. IEEE, 2017.

[101] Sergio Segura, Javier Troya, Amador Durán, and Antonio Ruiz-Cortés. Performance metamorphic testing: A proof of concept. *Information and Software Technology*, 98:1–4, 2018.

[102] Syed Muhammad Ali Shah, Daniel Sundmark, Birgitta Lindström, and Sten F Andler. Robustness testing of embedded software systems: An industrial interview study. *IEEE Access*, 4:1859–1871, 2016.

[103] Ali Shahrokni and Robert Feldt. A systematic review of software robustness. *Information and Software Technology*, 55(1):1–17, 2013.

[104] Faezeh Siavashi and Dragos Truscan. Environment modeling in model-based testing: concepts, prospects and research challenges: a systematic literature review. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, page 30. ACM, 2015.

[105] Bento Rafael Siqueira, Fabiano Cutigi Ferrari, Marcel Akira Serikawa, Ricardo Menotti, and Valter Vieira de Camargo. Characterisation of challenges for testing of adaptive systems. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*, page 11. ACM, 2016.

[106] Connie U Smith. Introduction to software performance engineering: Origins and outstanding problems. In *Formal Methods for Performance Evaluation*, pages 395–428. Springer, 2007.

[107] William J Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[108] Fang-Hsiang Su, Jonathan Bell, Christian Murphy, and Gail Kaiser. Dynamic inference of likely metamorphic properties to support differential testing. In *Automation of Software Test (AST), 2015 IEEE/ACM 10th International Workshop on*, pages 55–59. IEEE, 2015.

[109] Albert Tarantola. *Inverse problem theory and methods for model parameter estimation*. siam, 2005.

[110] Abbas Tarhini, Antoine Rollet, and Hacène Fouchal. A pragmatic approach for testing robustness on real-time component based systems. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, page 143. IEEE, 2005.

[111] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012.

[112] Arunchandar Vasan and Atif M Memon. Aspire: Automated systematic protocol implementation robustness evaluation. In *Software Engineering Conference, Australian Proceedings*, pages 241–250. IEEE, 2004.

[113] Robert J Walls, Yuriy Brun, Marc Liberatore, and Brian Neil Levine. Discovering specification violations in networked software systems. In *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*, pages 496–506. IEEE, 2015.

[114] Gwendolyn H Walton and Jesse H. Poore. Generating transition probabilities to support model-based software testing. *Software: practice and experience*, 30(10):1095–1106, 2000.

[115] Elaine J Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[116] Elaine J Weyuker and Filippos I Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering*, 26(12):1147–1156, 2000.

[117] Murray Woodside, Greg Franks, and Dorina C Petriu. The future of software performance engineering. In *Future of Software Engineering, 2007. FOSE'07*, pages 171–187. IEEE, 2007.

[118] Yang Xiang, Zhiliang Wang, and Xia Yin. Sip robustness testing based on ttcn-3. In *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*, pages 122–128. IEEE, 2009.

[119] Shu Xiao, Sheng Li, Xiangrong Wang, and Lijun Deng. Fault-oriented software robustness assessment for multicast protocols. In *Network Computing and Applications, 2nd IEEE Int. Symp. on*, pages 223–230. IEEE, 2003.

[120] Xiaoyuan Xie, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544–558, 2011.

[121] Qiang Xu, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. Proteus: network performance forecast for real-time, interactive mobile applications. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 347–360. ACM, 2013.

[122] Cheer-Sun D. Yang and Lori L. Pollock. Towards a structural load testing tool. *SIGSOFT Softw. Eng. Notes*, 21(3):201–208, 1996.

[123] Cemal Yılmaz, Sandro Fouche, Myra B Cohen, Adam Porter, Gülşen Demiröz, and Uğur Koç. Moving forward with combinatorial interaction testing. *Computer*, 47(2):37–45, 2014.

[124] Xia Yin, Zhiliang Wang, Chuanming Jing, and Jianping Wu. A formal approach to robustness testing of network protocol with time constraints. *Security and Communication Networks*, 4(6):622–632, 2011.

[125] Linbin Yu, Feng Duan, Yu Lei, Raghu N Kacker, and D Richard Kuhn. Constraint handling in combinatorial test generation using forbidden tuples. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pages 1–9. IEEE, 2015.

[126] Linbin Yu, Yu Lei, Mehra Nourozborazjany, Raghu N Kacker, and D Richard Kuhn. An efficient algorithm for constraint handling in combinatorial test generation. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 242–251. IEEE, 2013.

[127] Jian Zhang and Shing Chi Cheung. Automated test case generation for the stress testing of multimedia systems. *Software: Practice and Experience*, 32(15):1411–1435, 2002.

[128] Jian Zhang, Shing-Chi Cheung, and Samuel T Chanson. Stress testing of distributed multimedia software systems. In *Proc. of the IFIP TC6 WG6.1 Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification*, pages 119–133. Kluwer, BV, 1999.

[129] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Automatic generation of load tests. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 43–52. IEEE Computer Society, 2011.

[130] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Compositional load test generation for software pipelines. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 89–99. ACM, 2012.

[131] Junzan Zhou, Bo Zhou, and Shanping Li. Automated model-based performance testing for paas cloud services. In *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, pages 644–649. IEEE, 2014.