MŰEGYETEM 1782

Performance Testing and Performance Improvement Methods for Communicating Systems

Levente Erős

MSc. in Technical Informatics

Department of Telecommunications and Media Informatics
Doctoral School of Informatics
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics

PhD. Dissertation

Supervised by:

Dr. Tibor Csöndes

Honorary Associate Professor

Department of Telecommunications and Media Informatics
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics

Budapest, Hungary

2012.

Abstract

In this thesis, I propose methods for solving yet unsolved problems of black-box performance testing of communicating systems. The main motivation of creating the presented methods was the fact that while black-box conformance testing has evolved methods for test generation and execution, black-box performance testing lacks such methods, and performance tests are designed in an ad-hoc way.

The thesis has three main parts. The first two parts focus on two different aspects of performance testing, while the third part discusses a problem, which is based on the second part.

In the first part of the thesis, I deal with a problem of performance test execution namely, how to assign load generating software entities to the hosts of the performance test environment in order to exploit the capacities of the test environment as much as possible. After proving the NP-completeness of the problem, I formulate it as an integer linear program, and propose two heuristic methods for solving it. Then, the efficiency of the proposed methods are evaluated.

In the second part, I focus on a more specific case of performance testing in which, the performance requirement that the system under test (SUT) has to fulfill is the number of request messages to be processed within a second while serving a given number of users. I propose a performance testing method, which automatically checks whether the SUT fulfills this performance requirement. The presented method builds on the functional modeling techniques used in conformance testing. I compare the accuracy of the proposed method to that of an ad-hoc performance testing method used in the industry.

The third part of the thesis deals with the problem of increasing the performance of the SUT in case it has failed to pass the performance testing method presented in the second part of the thesis. After proving the NP-completeness of the problem, I formulate it as an integer linear program, and propose a heuristic algorithm for solving it. I also evaluate the efficiency of the two methods.

Kivonat

Disszertációmban kommunikáló rendszerek fekete doboz alapú teljesítménytesztelésének megoldatlan problémáival foglalkozom. Míg a fekete doboz alapú konformanciatesztelés kiforrott tesztgeneráló és -végrehajtó eljárásokkal bír, a fekete doboz alapú teljesítménytesztelésről ez nem mondható el, s így a teljesítmény-tesztek tervezése ad-hoc módon történik. A bemutatott eljárások kidolgozásának motivációja pontosan ez, azaz a kommunikáló rendszerek fekete doboz alapú teljesítménytesztelése elméleti hátterének kifejletlensége.

A disszertáció három fő részből áll. Az első két rész a teljesítmény-tesztelés két különböző területével foglalkozik, míg a harmadik részben egy, a második részben tárgyalt eljárásra épülő problémát mutatok be.

Az értekezés első részében a teljesítményteszt-végrehajtás egy olyan problémájával foglalkozom, amely során a tesztkörnyezet hosztjaihoz úgy kell hozzárendelni a tesztelt rendszert terhelő forgalmat generáló szoftverentitásokat, hogy a tesztkörnyezet szabad kapacitásait minél jobban kihasználjuk. NP-teljességének belátását követően a problémát felírom egészértékű lineáris programként, illetve bemutatok két heurisztikus algoritmust a probléma megoldására. Végül összehasonlítom a két ismertetett eljárás hatékonyságát.

A második részben a teljesítménytesztelés egy specifikusabb esetével foglalkozom, amikor is a tesztelt rendszerrel szemben támasztott teljesítménykövetelmény a rendszer által másodpercenként feldolgozandó kérésüzenetek száma adott számú felhasználó párhuzamos kiszolgálása mellett. Ismertetek egy teljesítménytesztelési eljárást, amely képes annak automatikus ellenőrzésére, hogy a tesztelt rendszer megfelel-e a fent említett két teljesítménykövetelménynek. A bemutatott eljárás a konformanciatesztelés területén használt funkcionális modellezési módszereken alapul. A szakasz végén a bemutatott eljárás pontosságát egy, az iparban használt ad-hoc eljárásával hasonlítom össze.

Az értekezés harmadik része a második részben bemutatott teljesítmény-tesz-

telési eljáráson megbukott rendszerek teljesítményének korrekciójával foglalkozik. A probléma NP-teljességének bebizonyítását követően azt egy egészértékű lineáris programként írom fel, illetve bemutatok egy heurisztikus algoritmust a probléma megoldására. A szakaszt a bemutatott megoldási eljárások hatékonyságának vizsgálatával zárom.

Acknowledgments

First of all, I would like to thank my supervisors, Dr. Tibor Csöndes for all the time and energy he has put into consulting me and guiding my research work all through the past years, and Dr. Sarolta Dibuz for her useful advices and support.

I am also very grateful to Dr. Gyula Csopaki for inviting me to research the field of testing as an MSc student and for his continuous guidance and support.

Furthermore, I would like to thank my colleagues and roommates, Dr. Péter Babarczi, Gábor Árpád Németh, and Zoltán Novák for their unconditional help and the great work athmosphere they provided. I would also like to thank my colleague, József Ernő Marton, for his technical advices.

Also, I would like to thank the High Speed Networks Laboratory (HSNLab) for providing the technical and financial background for my research work, especially Dr. Róbert Szabó, Dr. Attila Vidács, and Dr. Sándor Molnár for their useful advices related to my publications, and Erzsébet Győri for all her help.

My acknowledgments would not be complete without also thanking my head of department Dr. Tamás Henk, Dr. Edit Halász, Prof. Dr. Gyula Sallai, Dr. Gusztáv Adamis, and Dr. Gábor Kovács for their valuable advices on finalizing my thesis.

At last, but not at least I would like to thank all my family members for supporting me all through my student years, for the exceptional studying oppurtunities they provided, and for helping me overcome all the obstacles arisen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Testing plays a vital role in the development of a communicating system implementing a certain communication protocol. After the implementation phase, the developed system is regarded as a black box and different kinds of tests are executed against it in order to check whether it corresponds to its different kinds of requirements. Both the implementation and the testing phases use the specifications of the System Under Test (SUT from now on) as their inputs (see Figure 1.1). The implementation phase has well-developed methodologies, but these are not covered in this thesis. In the testing phase, among many others, the fulfillment of the conformance and performance requirements of the SUT are tested. In the rest of this chapter, I am going to review the current state of the art regarding the level of automation in conformance testing and performance testing and identify some problems, which are related to performance testing and for which, I propose solutions in this thesis.

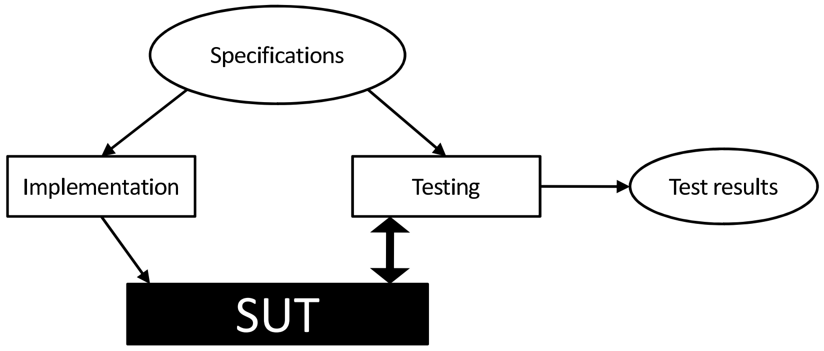

Figure 1.1: Development of communicating systems

1.1 The Evolution of Conformance Testing

In the field of telecommunications, conformance testing investigates whether the SUT implements the communicating protocol that it should implement according to its conformance requirements. Its basic concepts are documented in [1] and [2].

Conformance testing has an evolved scientific background. The different stages of evolution of conformance testing methodologies are presented in the following. These steps of evolution are also valid for test automation in general.

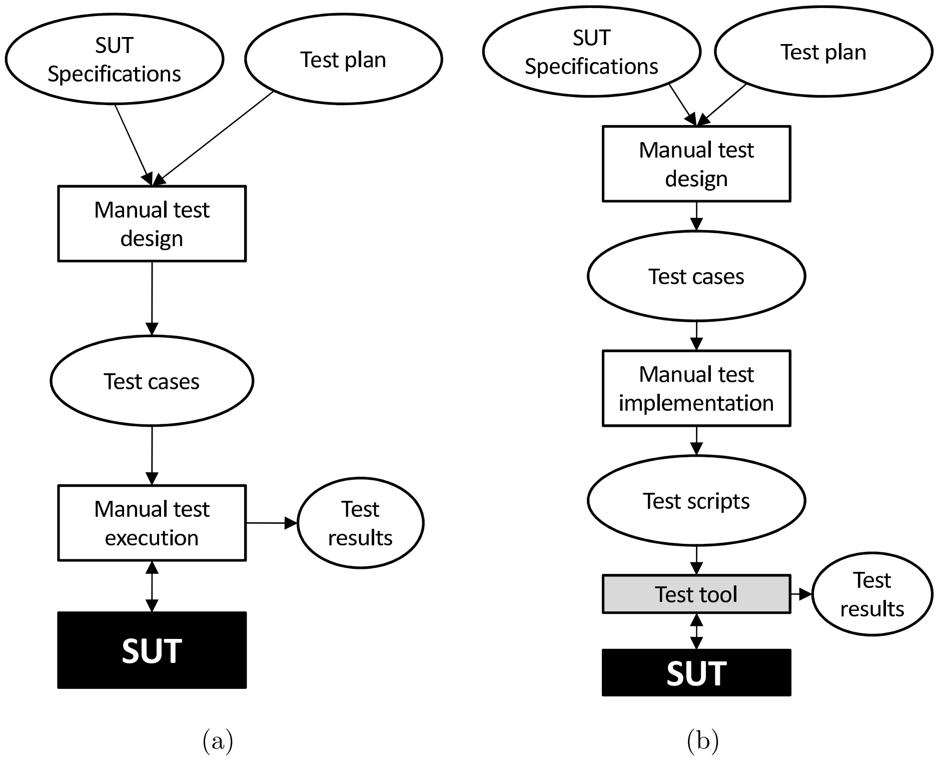

Figure 1.2: Manual (a), and script based (b) test design

Figure 1.2(a) shows the steps of manual testing as the earliest form of conformance testing. When manually generating test cases, the test designer uses the specifications of the SUT as an input, and a test plan, which includes the purposes/goals of the test. Based on these inputs, the test designer designs the test cases manually, on a high level of abstraction. The so created test cases, are then passed from the test designer to the test engineer. Following the steps described in the test cases, the test engineer executes the test manually, by interacting with the SUT, and recording the outputs that the SUT returns for different inputs. In this stage thus, both the design and the execution of tests were fully manual.

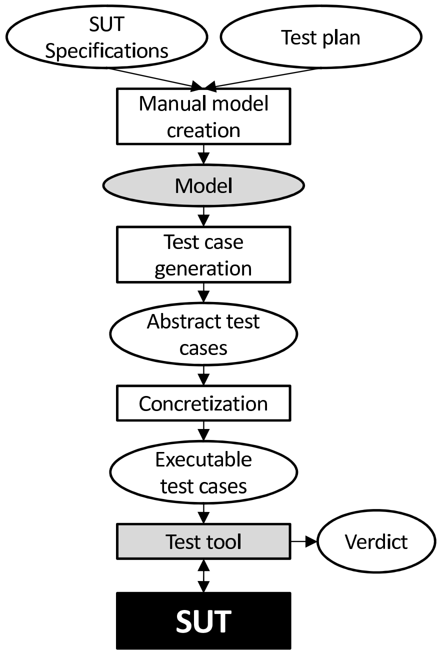

Figure 1.3: Model-based test design

The so-called script based tests make the process of testing easier by automating the test execution phase. As can be seen in Figure 1.2(b) in this case, the test designer does the same job as in the previous case that is, designs the necessary test cases based on system requirements and the test plan. The test engineer on the other hand, rather than manually executing the test based on the high-level test cases, implements a test script. The so created test script then interacts with the SUT that is, it sends different inputs to the SUT, and observes the outputs returned by the SUT. Based on these observations, the script generates a test verdict as its output.

Although script based testing makes test execution a lot easier than in the case of fully manual testing, it is unable to solve the problems raised by manual test design. Thus, tests remain unstructured, their coverage is incomplete, the number of necessary test cases can be huge and thus, they are hard to maintain. In other words, the test is designed in an ad-hoc way, the quality of the test depends on the expertise of the test designer, and the design and maintenance of test cases is time consuming, and costly.

The above mentioned problems of manual test design can be solved by using *model-based testing* methods. In the case of model-based testing (Figure 1.3), instead of manually designing test cases, the test designer has to define the formal model of the SUT on an abstract level [3]. From this abstract model, the model-based testing tool first generates abstract test cases. Then, by concretizing the abstract test cases, the tool generates executable test cases, which can be executed against the SUT [4].

The most widely used models for the formal description of the SUT in model-based testing, are the different finite state machine-based models. These are summarized by Lee et al. [5], and Dorofeeva et al. [6]. Another formal description technique that can be also used for defining the functional behavior of a communicating system, is the labeled transition system [7].

Based on the different finite state machine models, many methods have been developed for testing the conformance of a communicating system (i.e. for testing whether the SUT corresponds to a finite state machine). [8–14] present the basic formal methods for conformance test derivation, while [15] compares the performance of four of these methods. Lee et al. [16] investigate the existence and identification of distinguishing sequences of finite state machines, which are used for identifying the current state of the SUT. Pap et al. [17], and Brinksma et al. [18] propose test generation and test selection methods. Nemeth et al. [19], and Fakih et al. [20] propose incremental methods for maintaining the test set of a changing conformance specification. Ghedamsi et al. [21,22], and Celikkan et al. [23] present methods capable of generating diagnostic tests, which, beyond identifying that the SUT does not correspond to its conformance specifications, are capable of localizing the fault. Finally, Tretmans [24] studies conformance testing based on labeled transition systems.

Based on the huge amount of test generation methods, many model-based testing tools have been developed throughout the years, for example Lutess [25], Lurette [26], GATeL [27], TVEDA [28], and AutoLink [29], but the three most widely used tools are Elvior TestCast Generator, [30], Spec Explorer [31], and Conformiq Tool Suite [32].

1.2 Problems of Performance Testing

As we have seen in the previous section, conformance testing has taken a long journey from conformance modeling through test derivation methods, to widely used testing tools. Performance testing on the other hand, has not reached the same level of automation and its methodologies are far from being as much evolved as those of conformance testing, although performance testing is widely used in the industry. A performance test measures different performance characteristics of the SUT, and is executed against the SUT once it has been found to correspond to its conformance requirements. Some types of performance tests have been classified under different names. *Load testing* examines how the SUT behaves when increasing the number of requests sent to it. The goal of *stress testing* is to find the breaking point of the system, i.e. the load under which the system breaks. Finally, *scalability testing* examines how the system behaves under high loads of data sent to it. Currently, black-box performance tests are designed and executed as shown in Figure 1.4.

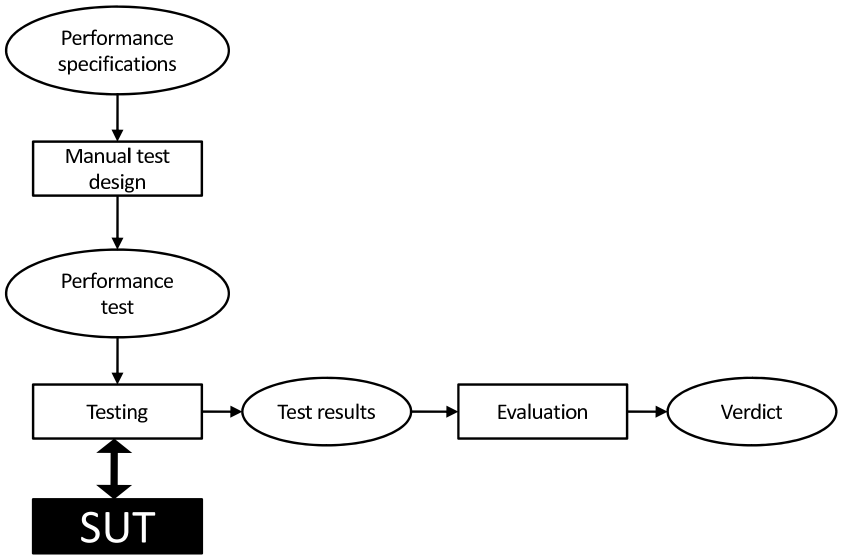

Figure 1.4: Performance testing

Thus, black-box performance tests are designed manually, in an ad-hoc way nowadays, without any theoretical background. As the figure shows, based on the performance specifications, the test designer creates and executes the performance test relying on their expertise in the field of performance testing. The test then returns some numerical results that a test engineer has to evaluate and based on which he or she has to decide whether the SUT has passed the performance test or not.

Besides the lack of rigorousness, another problem of performance testing is how to exploit the capacities of the test environment:

During a performance test, the test environment has to generate a relatively high load towards the SUT. The test environment however, is built from universal hardware unlike the SUT, which is built and thus, optimized for a specific purpose.

The reason for this is that the test environment should be able to be reused for different performance tests, which test systems, each of which might be optimized for different purposes. In the industry thus, load generating software entities are used for generating stress towards the SUT. Moreover, for stressing the SUT, multiple hosts are used in the test environment, which are somehow assigned and then execute these load generator entities. This situation demands methods for assigning load generator entities to the hosts of the test environment closely to optimal, in order to exploit the capacities of the test environment as much as possible.

In this thesis, I propose methods for solving the above described two problems of performance testing.

Based on the above open problems of performance testing, my research objectives were

- to define efficient load distribution methods for assigning the load generating software entities to the hosts of the test environment,
- to define a model-driven performance testing method, which outperforms the ad-hoc methods currently used in the industry, and
- to create methods for efficiently improving the performance of the SUT after it has failed the performance test.

Accordingly, the rest of the thesis is organized as follows:

In Chapter 2, I propose load distribution methods for distributing load generator entites among the hosts of the test environment. These methods aim at exploiting the capacities of testing hosts as much as possible.

In Chapter 3, I deal with the theory of performance testing, more specifically, with the subproblem, where the performance requirement of the SUT is the maximal number of request messages that it has to be able to serve within a second while serving a given number of users simultaneously. In this chapter, I propose a model-driven performance testing method, which automatically determines whether the SUT fulfills this performance requirement.

In Chapter 4, I propose methods for correcting the performance of the SUT after it has failed a performance test, which is executed according to the method described in Chapter 3. The presented methods aim at determining how to improve the number of request messages the SUT is able to serve within a second, at minimal cost.

Chapter 2

Load Distribution in a Performance Testing Environment

As I have mentioned in Chapter 1, the SUT is built for a specific purpose, and its performance is optimized for serving this specific purpose, while the hosts used in the performance test environment (testing hosts from now on) are usually universal hosts, which are used for testing the performance of multiple kinds of SUTs. During a performance test, the test environment has to generate a relatively high load towards the SUT, for a relatively long time. As I have also mentioned, this is achieved by using multiple testing hosts (THs from now on) in the test environment which, together are capable of generating this load (see Figure 2.1).

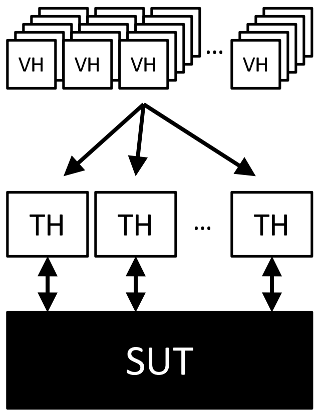

Figure 2.1: Assigning THs to VHs

The load generators or virtual hosts (VHs from now on) used for generating stress towards the SUT, run on THs and usually, the number of VHs is larger than the number of THs. Each TH has a total capacity, while each VH has a required capacity, and a VH can only be assigned to a TH if, for the whole duration of execution of the VH, the free capacity of the TH is greater than or equal to the required capacity of the VH. Each VH has to be either assigned to a TH or dropped. This assignment has to be carried out by a test controller entity. The objective of the test controller during VH assignment is to maximize the average utilization of THs (that is, to maximize the load generated by the test environment). In the rest of this chapter, I will have the assumption that the total capacity needed to generate the load that the SUT has to be stressed by (that is, the total VH capacity to be assigned) is between $TC - \delta$ and $TC + \delta$ at all times, where TC is the aggregated capacity of THs, and δ is constant. Beyond its required capacity, for each VH, its starting time and execution time is defined.

The rest of this chapter goes as follows: In Section 2.1, I review the related work in the subject. In Section 2.2, I formally define the load distribution problem, and prove its NP-completeness. In Section 2.3, I formulate the load distribution problem as a binary linear program and propose a solution for solving the problem as a series of binary linear programs. In Section 2.4, I propose a heuristic solution for the load distribution problem. I close the chapter by simulation results comparing the efficiency of the two proposed methods and that of a greedy algorithm in Section 2.5 and by the conclusions of the chapter in Section 2.6.

2.1 Related Work

The problem described above (the load distribution problem from now on) is very similar to the task assignment problem, in which tasks have to be assigned to processors [33]. For the different variants of the task assignment problem, many solutions have been published. [34] presents a family of heuristic algorithms for an extended task assignment problem, in which incompatible tasks can be modelled. [35] presents a solution according to which all the tasks are assigned to clusters the number of which equals the number of processors. After creating the clusters, each cluster is assigned to a processor. [36] introduces a heuristic algorithm for minimizing communication costs. The objective of the two algorithms presented in [37] is also to minimize communication costs between tasks. On the other hand, the method in [38] tries to minimize the total amount of time needed to execute all the tasks by using bin packing techniques. [39] proposes the formulation of and two heuristic algorithms for a special case of the task assignment problem in which, each processor is constrained in the number of tasks it can handle. [40] uses particle swarm optimization for task assignment. [41] proposes a graph matching based method finding the optimal solution of the task assignment problem. The methods presented in [42] also finda the optimal solution. One of them works by reducing the search space while the other one executes the task assignment algorithm in parallel to save running time. Finally, [43] proposes a method by which, the efficiency of already existing task assignment heuristics can be improved.

Unfortunately, non of the above solutions are applicable for solving the load distribution problem, since despite its similarity, the load distribution problem differs from the task assignment problem at multiple points: First, the objective of the assignment is different in the two cases; unlike the task assignment problem, in the case of the load distribution problem, there is no need to optimize the solutions for minimizing communication costs and deal with processor connectivity issues. Second, in the case of the load distribution problem, each test component has a predefined starting time, while in the task assignment problem, the starting time of tasks can be arbitrary. As a result of the above, my goal was to develop solutions for the task assignment problem, which will be presented in this section.

2.2 Problem Definition and Complexity

The load distribution problem is defined on a discrete time axis composed of atomic time slots. The problem is defined as follows:

Given are the set of THs $\mathcal{TH} = \{TH_i\}$ and the set of VHs $\mathcal{VH} = \{VH_i\}$. Each TH has one attribute $TH_i = (TC_i)$, where TC_i is the total capacity of TH_i. Each VH has three attributes $VH_i = (ST_i, RT_i, C_i)$, where ST_i is the starting time of VH_i (i.e. the number of the time slot in which VH_i starts), RT_i is the execution time of VH_i (i.e. the number of time slots that the execution of VH_i takes), and C_i is the required capacity of VH_i.

The problem to be solved is as follows: For each $VH_i \in \mathcal{VH}$, choose the value of assignment function $\sigma \in \mathcal{VH} \rightarrow \mathcal{TH}$ from domain $\mathcal{D} = \mathcal{TH} \bigcup \{\emptyset\}$ such that, Formulas 2.1 and 2.2 are true. Choosing TH_j as the value of $\sigma(VH_i)$ corresponds to assigning VH_i to TH_j, while choosing $\emptyset$ as the value of $\sigma(VH_i)$ corresponds to dropping VH_i. In Formula 2.1, U is a lower limit for the total TH capacity. If u is the total TH utilization (used TH capacity to total TH capacity ratio), then

$U = u \sum_{i=1}^{|\mathcal{TH}|} TC_i t_{max}$. In the formula and the rest of the chapter, t_{max} denotes the last time slot ($t_{max} = \max_{k:\, VH_k \in \mathcal{VH}} (ST_k + RT_k - 1)$).

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=1}^{t_{max}} \sum_{\substack{k:\sigma(VH_k)=TH_i \wedge \\ \wedge ST_k \leq j \wedge \\ \wedge j \leq ST_k+RT_k-1}} C_k \geq U \tag{2.1}$$

$$\forall (i : TH_i \in \mathcal{TH}) : \forall (j = 1, \ldots, t_{max}) : \sum_{\substack{k:\sigma(VH_k)=TH_i \wedge \\ \wedge ST_k \leq j \wedge \\ \wedge j \leq ST_k+RT_k-1}} C_k \leq TC_i \tag{2.2}$$

Formula 2.1 states that the total utilization of THs should be above the lower limit U, while Formula 2.2 expresses that for each TH_i, in each time slot, the aggregated capacity of VHs running on TH_i, must be lower than or equal to the total capacity of TH_i.

In the following, I prove the NP-completeness of the above defined problem by reducing an arbitrary instance of the NP-complete knapsack problem with equal value and weight functions. During the proof, I first show that the problem is in NP by showing that it can be decided in polynomial time whether an arbitrary solution candidate is a solution (a witness) of the load distribution problem or not. After showing that the problem is in NP, I am going to prove its NP-completeness by defining a mapping that transforms an arbitrary instance of the NP-complete variant of the knapsack problem to an instance of the load distribution problem in polynomial time, and by proving that the so obtained instance of the load distribution problem is solvable exactly if the corresponding instance of the knapsack problem is solvable.

The main idea of the proof is that the special case of the load distribution problem in which there is only a single time slot, and each VH starts and ends in that time slot, is identical to the NP-complete variant of knapsack problem, in which the cost and the weight of each element are equal to each other. Thus, in this variant of the knapsack problem, there is a single value (x) assigned to each element and packing the elements in the knapsack corresponds to selecting a set of elements for which, the sum of the x values is greater than or equal to a lower bound and lower than or equal to an upper bound.

Proof: An assignment defining the value of $\sigma(VH_i)$ for each $VH \in \mathcal{VH}$ is an appropriate witness, since the fulfillment of Formulas 2.1 and 2.2 can be verified in $O(|\mathcal{VH}| + |\mathcal{TH}|)$ time that is, in linear time. This means that the load distribution problem is in NP. Now, we have to prove that an arbitrary instance of the knapsack problem with equal value and weight functions can be transformed to an instance of the load distribution problem in polynomial time.

The knapsack problem is defined as follows:

Given are a set G, for all of its elements g_j a positive integer $v(g_j)$ and a positive integer $w(g_j)$ and positive integers V and W. The question to be answered is as follows: Is there a subset $G' \subseteq G$ such that the following inequalities are true?

$$\sum_{g_j \in G'} w(g_j) \leq W \tag{2.3}$$

$$\sum_{g_j \in G'} v(g_j) \geq V \tag{2.4}$$

The knapsack problem remains NP-complete if $v(g_j) = w(g_j)$ that is, if the value and weight functions of elements are identical [44]. Thus, let us rename functions w and v to x. From the so obtained instance of the knapsack problem, an instance of the load distribution problem can be created using the following assignments:

$$\begin{array}{rcl} U & := & V \\ t_{max} & := & 1 \\ \mathcal{TH} & := & \{TH_1\} \\ TC_1 & := & W \\ \mathcal{VH} & := & \{VH_i | g_i \in G\} \\ \forall(i : VH_i \in \mathcal{VH}) : ST_i & := & 1 \\ \forall(i : VH_i \in \mathcal{VH}) : RT_i & := & 1 \end{array}$$

$$\begin{array}{rcl} \forall(i : VH_i \in \mathcal{VH}) : C_i & := & x(g_i) \\ \forall(i : VH_i \in \mathcal{VH}) : \sigma(VH_i) = TH_1 & \Leftrightarrow & g_i \in G' \\ \forall(i : VH_i \in \mathcal{VH}) : \sigma(VH_i) = \emptyset & \Leftrightarrow & g_i \notin G' \end{array}$$

According to the above assignments, there will be a single TH the total capacity of which equals the capacity of the knapsack. Each element of the knapsack problem corresponds to a VH in the load distribution problem with its weight and value being equal to the capacity of the corresponding VH. The above created instance of the load distribution problem is one dimensional that is, each VH runs in the only time slot, which is time slot 1. The lower limit of total TH utilization U equals V, which is the lower value limit in the knapsack problem.

Using the above assignments, Formula 2.3 takes the following form:

$$\sum_{\sigma(VH_i)=TH_1} C_i \leq TC_1 \tag{2.5}$$

Using the assignments, Formula 2.4 takes the following form:

$$\sum_{\sigma(VH_i)=TH_1} C_i \geq U \tag{2.6}$$

Since the above created instance of the load distribution problem has a single TH and a single time slot, and all the VHs run in this single time slot (i.e. for time slot 1, $ST_k \leq j \wedge j \leq ST_k + RT_k - 1$ is true for each $i : VH_i \in \mathcal{VH}$), Formula 2.2 is identical to Formula 2.5 and Formula 2.1 is identical to Formula 2.6, in the case of this instance of the load distribution problem. Thus, Formula 2.3 can be transformed to Formula 2.1 and Formula 2.4 can be transformed to Formula 2.2. The reduction can be carried out in O($|G|$) time that is, in linear time. This means that the knapsack problem with equal value and weight functions is Karp reducible to the load distribution problem.

And finally, since the knapsack problem with equal value and weight functions is Karp reducible to the load distribution problem in linear time, and the load distribution problem is in NP, the load distribution problem is NP-complete.

■

Note: The load distribution problem is a special case of a problem, which is similar to the two-dimensional bin packing problem and only differs from it in its objective function [45]. Thus, the two-dimensional bin packing problem might be a good candidate for proving the NP-hardness of the load distribution problem.

2.3 An ILP Based Heuristic Solution for the Load Distribution Problem

According to the previous section, the load distribution problem is NP-complete. An effective way to find the optimal solution of an NP-complete problem is formulating it as an integer linear program (ILP), and then solving this ILP.

In the case of the load distribution problem, the ILP is a binary linear program (BLP) meaning that each variable the value of which has to be found can only take

0 or 1 as its value. Before formulating the load distribution problem as a BLP, the boolean variable a_{kj} has to be defined (for easier readability) as follows:

$$a_{kj} = \begin{cases} 0 & \text{if } ST_k \leq j \wedge ST_k + RT_k - 1 \geq j \\ 1 & \text{otherwise} \end{cases} \tag{2.7}$$

Furthermore, a new TH $TH_{|\mathcal{TH}|+1}$ has to be introduced. Assigning VH_k to $TH_{|\mathcal{TH}|+1}$ represents dropping VH_k. The total capacity of this TH is infinite or technically, it is equal to the sum of the capacities of all VHs (in order for each VH to be able to be assigned to it), formally:

$$\begin{gathered} \mathcal{TH}' = \mathcal{TH} \bigcup \{TH_{|\mathcal{TH}|+1}\}, \text{ where} \\ TH_{|\mathcal{TH}|+1} = (TC_{|\mathcal{TH}|+1}) \text{ and} \\ TC_{|\mathcal{TH}|+1} = \sum_{k:\, VH_k \in \mathcal{VH}} C_k \end{gathered} \tag{2.8}$$

After all the above, the BLP formulation of the load distribution problem is as follows. The unknown variables the values of which have to be found are variables s_{ki}. The value of s_{ki} is 1 if VH_k gets assigned to TH_i, otherwise its value is 0.

Maximize:

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=1}^{t_{max}} \sum_{k=1}^{|\mathcal{VH}|} a_{kj} s_{ki} C_k \tag{2.9}$$

Subject to:

$$\forall(i : TH_i \in \mathcal{TH}') : \forall(j = 1, 2, \ldots, t_{max}) : \sum_{k=1}^{|\mathcal{VH}|} a_{kj} s_{ki} C_k \leq TC_i \tag{2.10}$$

$$\forall(k : VH_k \in \mathcal{VH}) : \sum_{i=1}^{|\mathcal{TH}'|} s_{ki} = 1 \tag{2.11}$$

$$\forall(k : VH_k \in \mathcal{VH}) : \forall(i : TH_i \in \mathcal{TH}') : s_{ki} \in \{0, 1\} \tag{2.12}$$

The objective function of the problem, Formula 2.9, means that the average utilization of all THs must be maximal that is, the volume of non-dropped VHs must be maximal, where the volume of a VH means the product of its capacity and execution time. $TH_{\mathcal{TH}+1}$ is not taken into consideration by the objective function, since $TH_{\mathcal{TH}+1}$ was only introduced to represent VH dropping, and the objective is to maximize the average utilization of the THs included in set $\mathcal{TH}$.

Formula 2.10 is a constraint stating for each $TH_i \in \mathcal{TH}$ that in each time slot, the aggregated capacity of all the VHs running on TH_i is lower than or equal to the total capacity of TH_i.

Formulas 2.11 and 2.12 define constraints for the s_{ki} values expressing that from among the s_{ki} variables belonging to VH_k, exactly one equals 1, while the others equal 0. Thus, each VH is assigned to exactly one TH included in set $\mathcal{TH}$.

In the above formulation, the number of inequalities that Formula 2.10 produces is $|\mathcal{TH}|t_{min}$. This number of inequalities is unnecessarily large, since Formula 2.10 can be only violated in the time slots of VH assignment that is, only in those time slots j for which, $\exists(i : TH_i \in \mathcal{TH}) : ST_i = j$. Thus, it is enough to define Formula 2.10 for those time slots, in which a VH starts. Taking this into consideration, Formulas 2.13 to 2.16 give an equivalent formulation of the load distribution problem, which omits the unnecessary inequalities produced by Formula 2.10.

Maximize:

$$\sum_{i=1}^{|\mathcal{TH}|}\sum_{j=1}^{t_{max}}\sum_{k=1}^{|\mathcal{VH}|} a_{kj}s_{ki}C_k \tag{2.13}$$

Subject to:

$$\forall(i : TH_i \in \mathcal{TH}') : \forall(l : VH_l \in \mathcal{VH}) : \sum_{k=1}^{|\mathcal{VH}|} a_{kST_l}s_{ki}C_k \leq TC_i \tag{2.14}$$

$$\forall(k : VH_k \in \mathcal{VH}) : \sum_{i=1}^{|\mathcal{TH}'|} s_{ki} = 1 \tag{2.15}$$

$$\forall(k : VH_k \in \mathcal{VH}) : \forall(i : TH_i \in \mathcal{TH}') : s_{ki} \in \{0, 1\} \tag{2.16}$$

The number of equations and inequalities that the above formulation produces is lower than the number of equations and inequalities of the first formulation of the problem (if $|\mathcal{VH}| < t_{max}$). However, as the later presented simulation results show, the time needed to solve the BLP formulated by Formulas 2.13 to 2.16 can be extremely large and technically, the BLP cannot be solved in most of the scenarios.

The above issue can be solved as follows: First, the time axis should be divided up into time windows, which consist of a given number of time slots. The number of time slots contained in a time window is called the window size, and will be denoted by W, from now on. Then, starting from the first time window, a BLP is formulated and solved for each time window. The BLP formulated for time window n defines the assignment constraints of those VHs which start in time window n. The formulation of this BLP depends on (some of the) VH assignments made in the earlier time windows. That is, the solution of the BLP formulated for a given time window serves as an input of BLP formulation of (some) further time windows. The reason for this is that the assignment of VH_k to TH_i reduces the available capacity of TH_i not only for the time window in which VH_k was assigned to TH_i, but for the whole duration of the execution of VH_k.

Each BLP formulated for a time window will give an optimal solution for the corresponding time window, but the global solution, which is thus, composed of a series of sub-optimal solutions will not be optimal in general. The time needed to solve a series of BLPs however, can be reasonable in some scenarios (even though the sub-problems are NP-complete as well).

Formulas 2.17 to 2.20 formulate the load distribution problem for time window n, with window size W. Similarly to Formula 2.14, Formula 2.18 requires the aggregated capacity of VHs currently running on each TH to be lower than or equal to the total capacity of the corresponding TH. This constraint is only defined for those time windows in which, a VH starts. In the formulation below, the value of S_k is a reference to the TH to which, VH_k was assigned before time window n. That is, $S_k = i$, if and only if VH_k was assigned to TH_i before time window n. If VH_k starts after the time window preceding time window n, then $S_k = -1$. This means that the VH assignments made in earlier time windows serve as an input of formulating the BLP for the current time window. If $S_k = |\mathcal{TH}| + 1$, then VH_k is dropped.

Maximize:

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=(n-1)W+1}^{nW} \sum_{\substack{k:\, VH_k \in \mathcal{VH} \wedge \\ \wedge ST_k \geq (n-1)W+1 \wedge \\ \wedge ST_k \leq nW}} a_{kj} s_{ki} C_k \tag{2.17}$$

Subject to:

$$\begin{gathered} \forall(i : TH_i \in \mathcal{TH}') : \\ \forall(l : VH_l \in \mathcal{VH} \wedge ST_l \geq (n-1)W + 1 \wedge ST_l \leq nW) : \\ \sum_{\substack{k:\, VH_k \in \mathcal{VH} \wedge \\ \wedge ST_k \geq (n-1)W+1 \wedge \\ \wedge ST_k \leq nW}} a_{kST_l} s_{ki} C_k \leq TC_i - \sum_{\substack{k:\, VH_k \in \mathcal{VH} \wedge \\ \wedge S_k = i}} a_{kST_l} C_k \end{gathered} \tag{2.18}$$

$$\forall(k : VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W + 1 \wedge ST_k \leq nW) : \sum_{i=1}^{|\mathcal{TH}'|} s_{ki} = 1 \tag{2.19}$$

$$\begin{gathered}\forall(k : VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W + 1 \wedge ST_k \leq nW) : \\ \forall(i : TH_i \in \mathcal{TH}') : s_{ki} \in \{0,1\}\end{gathered} \tag{2.20}$$

For the whole duration of the test, the VH assignment is carried out by Algorithm 1, which uses the above BLP formulation. The BLP defined by Formulas 2.17 to 2.20 for time window n, is denoted by BLP_n in the algorithm.

Algorithm 1 gets sets $\mathcal{TH}$ and $\mathcal{VH}$ as its input, and outputs the S_k values belonging to each VH.

Algorithm 1: ILP based solution of the load distribution problem

```
   input : $\mathcal{TH}$, $\mathcal{VH}$, $W$
   output: $\bigcup_{k:\,VH_k \in \mathcal{VH}} \{S_k\}$
1  foreach $VH_k \in \mathcal{VH}$ do
2      $S_k := -1$;
3  $n := 1$;
4  while $(n-1)W + 1 \leq t_{max}$ do
5      Formulate and solve $BLP_n$;
6      foreach $k : VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W + 1 \wedge ST_k \leq nW$ do
7          foreach $i : TH_i \in \mathcal{TH}$ do
8              if $s_{ki} = 1$ then
9                  $S_k := i$
```

In lines 1 and 2, the algorithm initializes each S_k to initial value -1. From line 4, the algorithm runs iterations, one for each time window. Within an iteration, in line 5, the algorithm formulates and solves the BLP for the current time window. In lines 6 to 9, based on the calculated s_{ki} values of the BLP, the algorithm assigns the S_k value of each of those VHs, which were assigned to a TH (or dropped) in the current time window. By the end of the algorithm, each S_k value is known.

2.4 A Bin Packing Based Heuristic Solution for the Load Distribution Problem

As we will see in the Section 2.5, there are many scenarios in which, the ILP based solution of the load distribution problem cannot be used due to its huge running time, even with a small window size. Thus, I have created a heuristic algorithm for solving the problem. The main idea of the heuristic algorithm is as follows:

When assigning VH_k to a TH, other VHs which are assigned and executed relatively closely in time to VH_k have an effect on which THs VH_k can be assigned to, since these VHs occupy TH capacity for their whole execution time, which can be overlapping with the execution time of VH_k. Thus, when trying to assign VHs executed closely in time to each other, without the aggregated capacity of VHs running on a TH at a given point in time exceeding the total capacity of the TH, we have to solve a problem, which is similar to the bin packing problem, where the items packed into bins must fit into the bins and where the assignability of a specific item is affected by the assignments of the other items [46].

Based on the above, we can suspect that applying bin packing heuristics for assigning VHs executed relatively closely to each other, we can achieve a higher average utilization than the average utilization achieved by the greedy algorithm, which always assigns the VH with the next lowest starting time to the first TH that has enough free capacity for the whole duration of the execution of the VH under assignment. In order to be able to assign VHs executed relatively closely in time to each other, the presented heuristic algorithm divides up the $[1, \ldots, t_{max}]$ interval into time windows of size W, and considers the load distribution problem to be a series of bin packing-like problems, one for each time window, each of which is then solved using a heuristic algorithm similar to the first fit descending (FFD) algorithm [47].

According to the above, the presented heuristic algorithm uses time windows, just like the ILP based solution, but for different reasons. While in the case of the ILP based solution, the goal of using time windows was to reduce the time needed to solve the load distribution problem, in the case of the heuristic algorithm, the goal was to be able to regard the problem as a series of bin packing-like problems.

Algorithm 2 shows the steps of the heuristic algorithm used for solving the load distribution problem. If in the algorithm, S_k equals $|\mathcal{TH}| + 1$ then VH_k is dropped.

Algorithm 2: Heuristic solution for the load distribution problem

```
    input  : $\mathcal{TH}$, $\mathcal{VH}$, $W$
    output: $\bigcup_{k:\,VH_k \in \mathcal{VH}} \{S_k\}$
1   foreach $VH_k \in \mathcal{VH}$ do
2       $S_k := -1$;
3   $n := 1$;
4   while $(n-1)W + 1 \leq t_{max}$ do
5       $\mathcal{VH}_n := \{VH_k | VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W + 1 \wedge ST_k \leq nW\}$;
6       $\mathbf{VH}_n \leftarrow$ sort $\mathcal{VH}_n$ by $C_k$ descending; Return $k$;
7       $k := 1$;
8       while $k \leq |\mathbf{VH}_n|$ do
9           $i := 1$;
10          while $i \leq |\mathcal{TH}|$ do
11              if $\forall (j : a_{(\mathbf{VH}_n[k])j} = 1) : C_{\mathbf{VH}_n[k]} \leq TC_i - \sum_{\substack{l:\,VH_l \in \mathcal{VH} \\ \wedge S_l = i}} a_{lj} C_l$ then
12                  $S_{\mathbf{VH}_n[k]} = i$;
13                  break;
14              $i := i + 1$;
15          if $S_k = -1$ then
16              $S_k := |\mathcal{TH}| + 1$;
17          $k := k + 1$;
18      $n := n + 1$;
```

Just like the ILP based solution, Algorithm 2 gets $\mathcal{TH}$, $\mathcal{VH}$, and W as its input, and returns the S_k values.

After initializing each S_k to -1 in lines 1 and 2, the heuristics runs iterations, one for each time window. In line 5, the algorithm creates set $\mathcal{VH}_n$ from those VHs, which start in the current time frame. Then, from line 6 to 17 the heuristics assigns each VH from $\mathcal{VH}_n$ to a TH in TH', using a heuristic algorithm similar to the first fit descending algorithm, according to the following:

In line 6, the elements of $\mathcal{VH}_n$ are sorted by capacity in descending order and their references are put into vector $\mathbf{VH}_n$. Then for each VH_k for which, k is an element of $\mathbf{VH}_n$, starting from the VH_k element with the largest capacity, the algorithm finds the first TH from $\mathcal{TH}$, which has enough free capacity to execute

VH_k that is, the first TH the free capacity of which is greater than or equal to the capacity of VH_k in each time slot in which VH_k is running. If the algorithm fails to find such a TH, then VH_k is assigned to $TH_{|\mathcal{TH}|+1}$ that is, VH_k is dropped.

Thus, if we regard time windows as the atoms of VH assignment, the heuristics is greedy, since once it has assigned the VHs starting in a given time window to THs, this assignment cannot be changed in latter time windows, and serves as an input of the assignment problems to be solved in latter time windows. However, as the simulation results will show, using the presented heuristic algorithm, the average utilization obtained by the heuristics can be higher than the utilization obtained by the greedy algorithm.

2.5 Simulation Results

In this section, I present simulation results examining the running time of and the average TH utilization achieved by the greedy VH assignment method mentioned in Section 2.4 and the two methods introduced in Sections 2.3 and 2.4. I have run simulations in multiple scenarios, each of which is described by the following parameters:

- $|\mathcal{TH}|$ that is, the number of testing hosts,
- TC_{avg}, which is the average TH capacity (denoted by TC in Figures 2.12, 2.13, and 2.14),
- $Step_{TC}$ for which, $TC_i = TC_{avg} - \left(\frac{|TH|-1}{2} - i\right) Step_{TC}$ (denoted by $Step$ in Figures 2.12, 2.13, and 2.14),
- C_{avg}, which is the average VH capacity and equals 30 in each simulation scenario, and
- ϵ_C for which, VH capacities are generated on interval $[C_{avg} - \epsilon_C, C_{avg} + \epsilon_C]$, with uniform distribution (denoted by ϵ in Figures 2.12, 2.13, and 2.14).

The five simulation scenarios presented in this section are selected from among the 16 simulation scenarios that I have investigated.

During the simulations, the aggregated capacity of all active VHs was on interval $[|\mathcal{TH}| * TC_{avg} - C_{avg} - \epsilon_C, |\mathcal{TH}| * TC_{avg} + C_{avg} + \epsilon_C]$, in each time slot. In each simulation scenario, I measured the average utilization and running time of each method as a function of window size W. Since W is not a parameter of the greedy VH assignment algorithm, the average utilization of the greedy algorithm is represented as a horizontal, dotted line in the following figures, while its average running time is represented by horizontal lines with markers in Figure 2.14.

First, let us examine a simulation scenario in which, the highest average utilization achieved by the heuristic algorithm is not significantly higher than the average utilization of the greedy algorithm.

In the first scenario, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 0$, while $\epsilon_C = 4$ that is, the deviation of VH capacities is not large compared to the average VH capacity. Figure 2.2 shows the aggregated capacity of VHs to be assigned to THs in this scenario, as a function of time. The average of this aggregated capacity is the aggregated capacity of all THs.

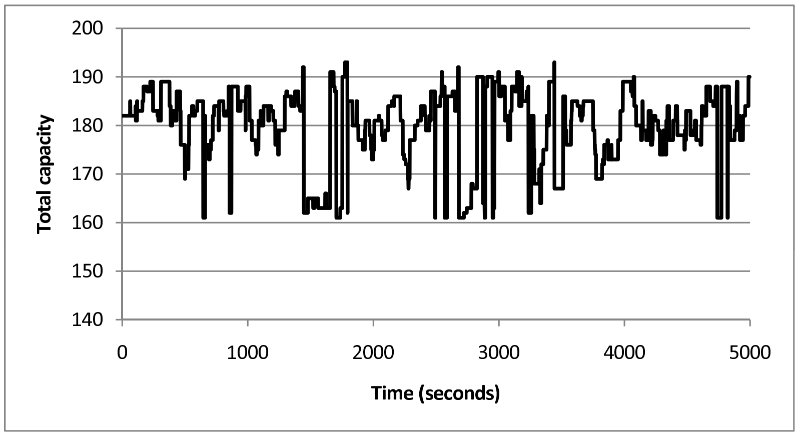

Figure 2.2: Required capacity, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 0$, $\epsilon_C = 4$

Figure 2.3 shows the average utilization achieved by the three methods. As can be seen in the figure, the average utilization achieved by the ILP based algorithm is higher than the average utilization of the greedy algorithm for each window size and it gets higher the bigger the window size gets. The maximal average utilization achieved by the heuristic algorithm is not significantly higher at its maximum, than the average utilization achieved by the greedy algorithm, in this case. The heuristic algorithm can increase the average utilization of THs by only 3 percent, while the ILP based algorithm can increase the average TH utilization of the greedy algorithm by 8 percent, in this scenario.

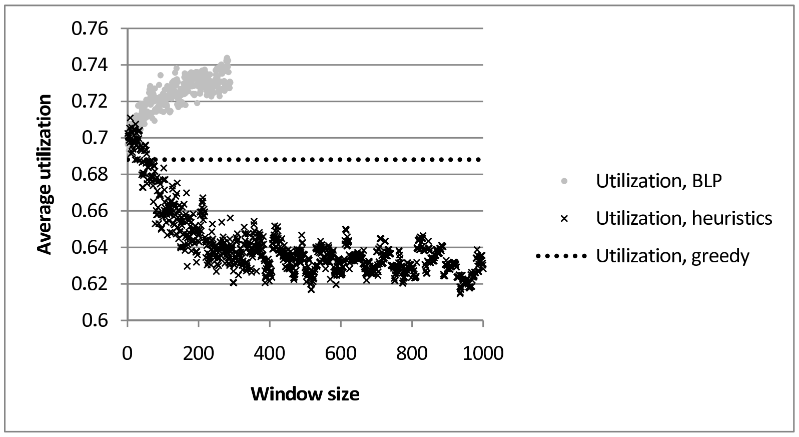

Figure 2.3: Average utilization, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 0$, $\epsilon_C = 4$

From the results of the first scenario, we can see that if the deviation of VH capacities is small compared to the average VH capacity, the heuristic algorithm does not perform well. This is confirmed by some other simulation scenarios as well, which gave similar results and which are not presented in this section.

In the second scenario, let us see how the proposed methods perform if the deviation of VH capacities is increased. In this scenario, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 0$, and $\epsilon_C = 20$.

Figure 2.4 shows the aggregated capacity of active VHs as a function of time in the second scenario, the average of which, equals the aggregated TH capacity.

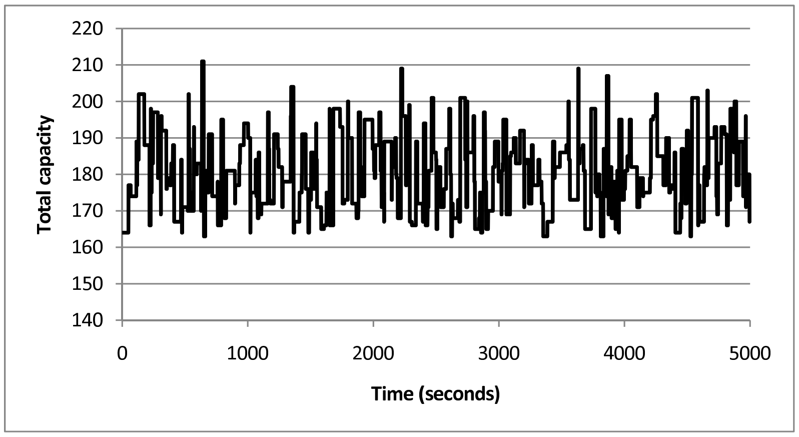

Figure 2.4: Required capacity, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 0$, $\epsilon_C = 20$

Figure 2.5 shows the average utilization achieved by each method. As it can be seen in the figure, the utilization of the ILP based method gets larger as the window size is increased. The figure also shows that the average utilization achieved by the heuristic method is higher than the average utilization of the greedy algorithm for almost every window size. In this scenario, the heuristic method performs better than in the first scenario, since the maximal average utilization it can achieve is by 9 percent higher than the average utilization of the greedy algorithm. The ILP based algorithm performs better as well. The highest average utilization it achieves is by 13 percent higher than the average utilization of the greedy algorithm.

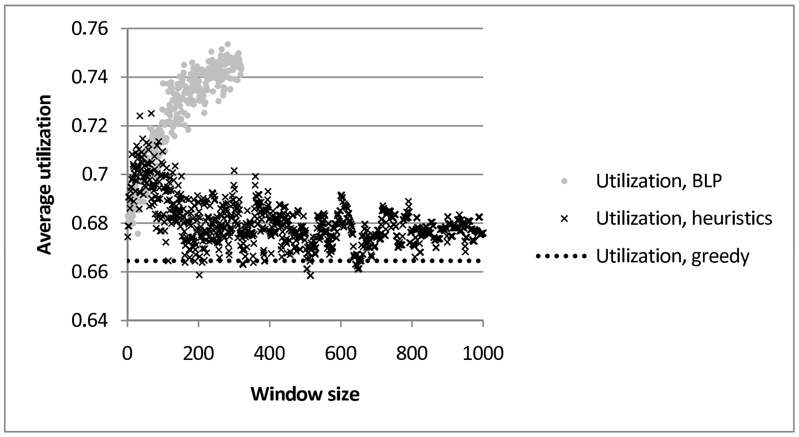

Figure 2.5: Average utilization, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 0$, $\epsilon_C = 20$

As the results of the second scenario (and other non-presented scenarios) have shown, as the deviation of VH capacities is increased, both the heuristic and the ILP based algorithms perform better.

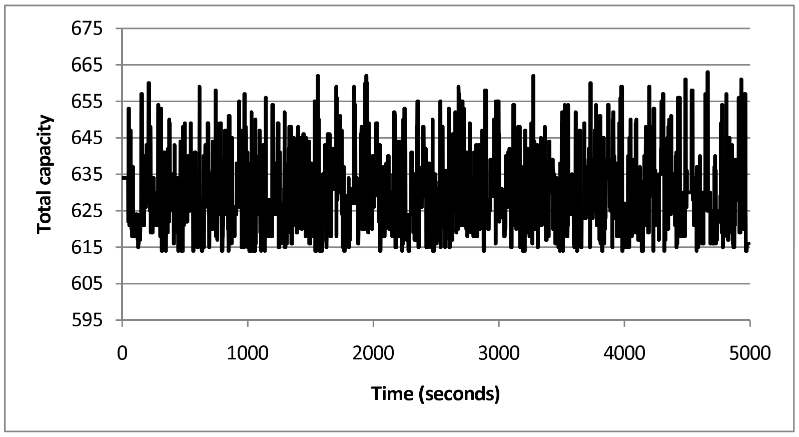

Figure 2.6: Required capacity, $|\mathcal{TH}| = 3$, $TC_{avg} = 210$, $Step_{TC} = 0$, $\epsilon_C = 20$

In the third scenario, I have investigated, what happens to the performance of the algorithms if the quotient of the average TH capacity and the average VH capacity is higher than in the second scenario. In this scenario, $|\mathcal{TH}| = 3$, $TC_{avg} = 210$, $Step_{TC} = 0$, while $\epsilon_C = 20$. In this case, the ILP based solution could not be applied due to its huge running time, even for small window sizes. Figure 2.6 shows the aggregated capacity of active VHs in each time slot, in this scenario. The aggregated VH capacity equals the aggregated TH capacity, in average.

Figure 2.7 shows the average utilization of the greedy and the heuristic algorithm. According to the figure, the performance of the heuristic method is not significantly higher than that of the greedy algorithm, in this scenario. The maximal average utilization achieved by the heuristic algorithm is only by 2 percent higher than the average utilization achieved by the heuristic algorithm.

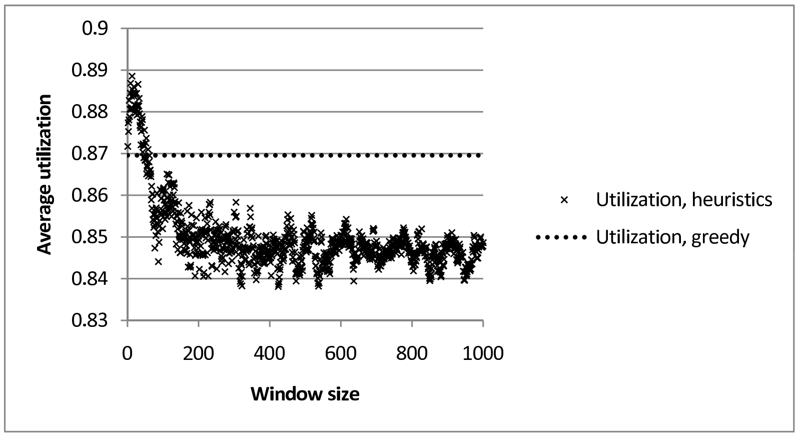

Figure 2.7: Average utilization, $|\mathcal{TH}| = 3$, $TC_{avg} = 210$, $Step_{TC} = 0$, $\epsilon_C = 20$

According to the results of the third scenario (and some non-presented scenarios with similar results), the performance of the heuristic algorithm gets worse as the quotient of the average TH capacity and the average VH capacity is increased.

Let us now investigate two scenarios in which, $TC_{avg} = 60$, and $\epsilon_C = 20$ that is, two scenarios in which the quotient of the average TH capacity and the average VH capacity is lower than in the third scenario and the deviation of VH capacities is higher than in the first scenario.

In the fourth scenario, I have changed the parameters of the second scenario so that TH capacities are not uniform. In this scenario $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 5$, while $\epsilon_C = 20$. Figure 2.8 shows the aggregated capacity of active VHs as a function of time, in this scenario. Just like in the previous scenarios, the aggregated capacity of active VHs equals the aggregated capacity of THs, in average.

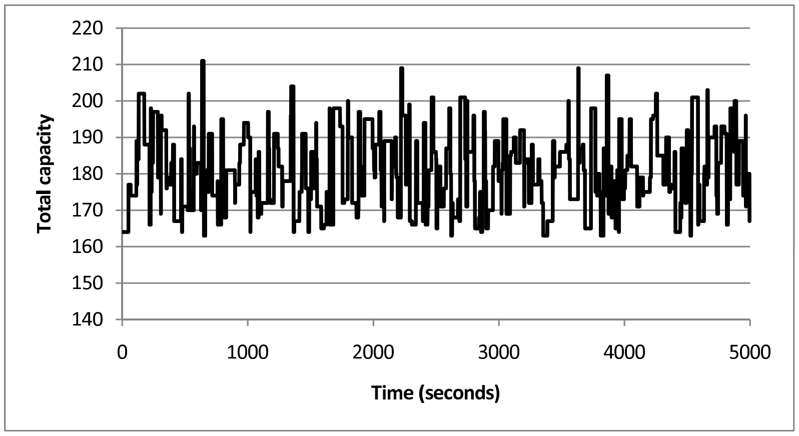

Figure 2.8: Required capacity, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 5$, $\epsilon_C = 20$

Figure 2.9 shows the average utilization of each method, in the fourth scenario. According to the figure, the heuristic algorithm produces higher average utilization values for each window size than the greedy algorithm, while the ILP based method gets more efficient as the window size increases.

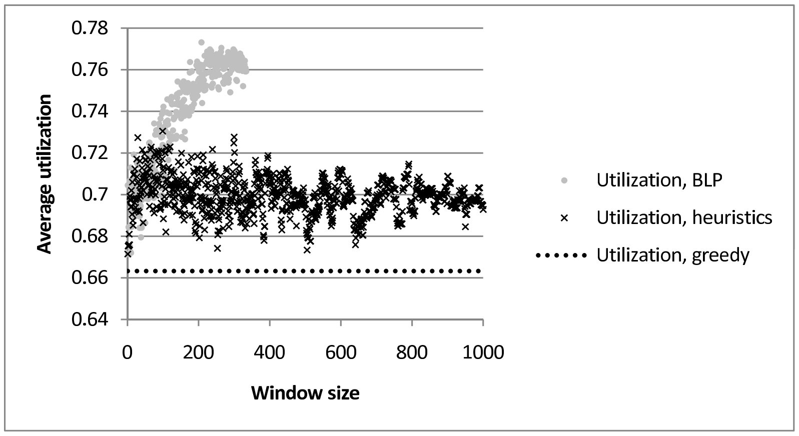

Figure 2.9: Average utilization, $|\mathcal{TH}| = 3$, $TC_{avg} = 60$, $Step_{TC} = 5$, $\epsilon_C = 20$

According to the results of the fourth scenario, in this case, the algorithms perform just as well as in the second scenario. The maximal average utilization achieved by the heuristic method is by 10 percent higher than the average utilization of the greedy algorithm, while the average utilization of the ILP based method is by 17 percent higher than that of the greedy algorithm at maximum.

Finally in the fifth scenario, let us see how the performance of the presented methods changes if the number of hosts of the fourth scenario is increased from 3 to 9. Thus in the fifth scenario, $|\mathcal{TH}| = 9$, $TC_{avg} = 60$, $Step_{TC} = 5$, and $\epsilon_C = 20$.

Figure 2.10 shows the aggregated active VH capacity as a function of time, which equals the aggregated TH capacity in average, just like in all the previous cases.

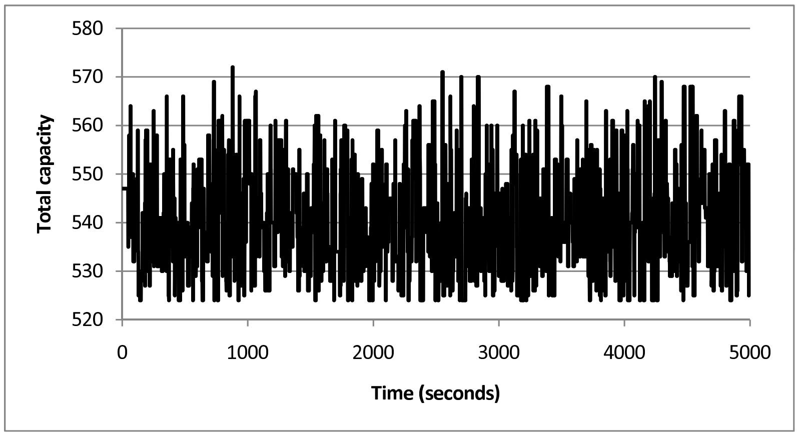

Figure 2.10: Required capacity, $|\mathcal{TH}| = 9,\ TC_{avg} = 60,\ Step_{TC} = 5,\ \epsilon_C = 20$

In this case, just like in the third scenario, the running time of the ILP based algorithm was extremely long even for small window sizes and thus, it could not be executed. Figure 2.11 shows the average TH utilization achieved by the bin packing based heuristic and the greedy methods, with different window sizes.

Figure 2.11 shows that the heuristic algorithm gives a higher average utilization value than the greedy algorithm, for each window size.

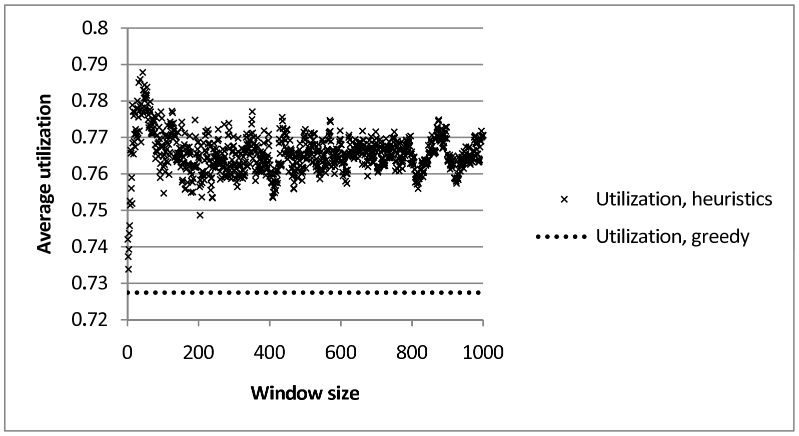

Figure 2.11: Average utilization, $|\mathcal{TH}| = 9,\ TC_{avg} = 60,\ Step_{TC} = 5,\ \epsilon_C = 20$

As the results of the fifth simulation scenario have shown, the performance of the heuristic algorithm remains good compared to that of the greedy algorithm if the number of testing hosts is increased. The maximal average utilization achieved by the heuristic algorithm is by 8 percent higher than the average utilization achieved by the greedy algorithm.

Figure 2.12 shows the running time of the ILP based heuristics on a logarithmic vertical axis, Figure 2.14 shows the running times of the greedy algorithm, while Figure 2.13 shows the running times of the bin packing based heuristics as a function of window size, for the different simulation scenarios. As it can be seen in the figures, both heuristic solutions have reasonable running times. The running time of the ILP based heuristic on the other hand, grows exponentially as the window size increases and as mentioned earlier, in two simulation scenarios it could not be executed at all due to its huge running time. As it can also be observed in the figure, the curves representing the running time of the ILP based solution start with a descent. The reason for this descent is the overhead caused by having to formulate a large number of binary linear programs in the case of lower window sizes.

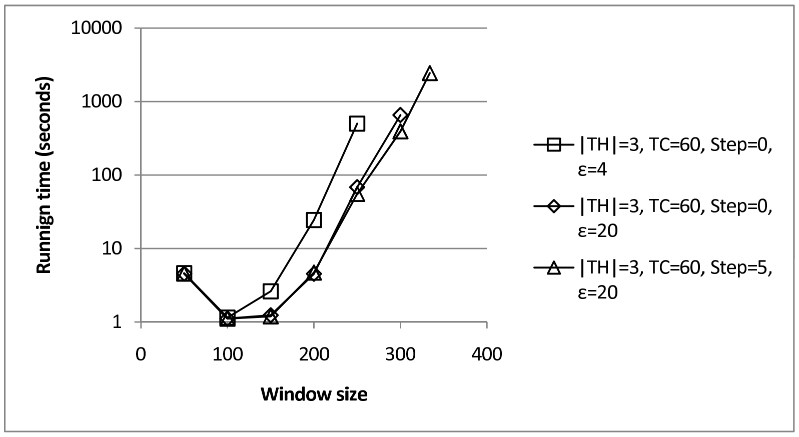

Figure 2.12: Running times of the ILP based algorithm

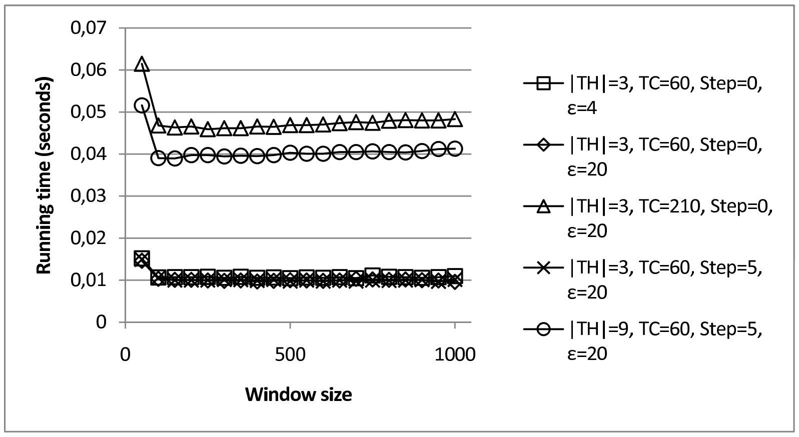

Figure 2.13: Running times of the bin packing based algorithm

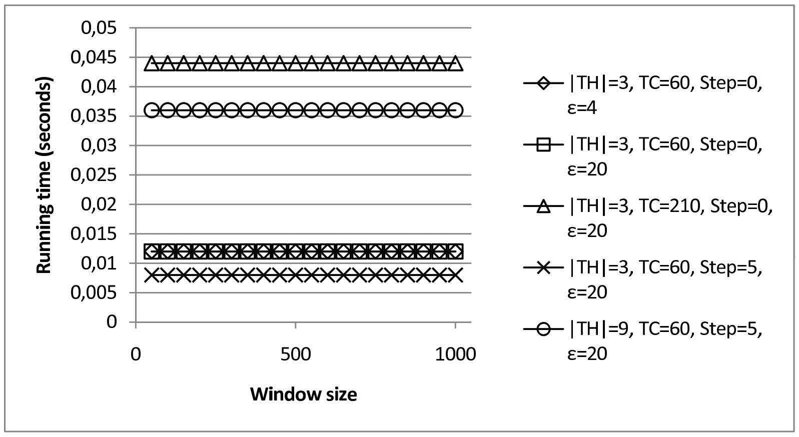

Figure 2.14: Running times of the greedy algorithm

Table 2.1 summarizes the above presented results by showing the maximal average utilization values achieved by each method in each scenario.

Scenario				Average utilization		
$\lvert\mathcal{TH}\rvert$	TC_{avg}	$Step_{TC}$	ϵ_C	BLP	Heuristics	Greedy
3	60	0	4	0.743811	0.710993	0.688083
3	60	0	20	0.753542	0.725086	0.664499
3	210	0	20	–	0.888547	0.869548
3	60	5	20	0.773141	0.730494	0.663302
9	60	5	20	–	0.78783	0.727412

Table 2.1: Maximal average utilization values achieved by different methods in different scenarios

As it can be observed in the figures plotting the average utilization values obtained by each method, the values close to the maximal average utilization value are concentrated to a narrow window size interval, in the case of the heuristic algorithm, and the average utilization obtained by the heuristic algorithm decreases both if the window size is increased beyond this interval and if it is decreased below this interval. The reason for this is as follows:

If the window size is too small then the heuristic algorithm takes too few VH assignments into consideration at one time and thus, approaches the greedy algorithm (which takes a single VH into consideration at one time). If however, the window size is too large, then there might be VHs in the same time window, which are executed far from each other in time. The assignabilities of these VHs are not affected by each other and thus, the problem of assigning VHs of the same, large time window is not similar to an instance of the bin packing problem anymore. Thus, using bin packing heuristics for large window sizes does not make any sense.

As it can also be observed in the above figures, the bigger the window size is, the more effective the ILP based solution gets in means of average utilization. The reason for this is that increasing the window size makes the sub-problem of assigning VHs of the same window size approach the problem of assigning all VHs to THs in which, $W = t_{max}$, and for which, the ILP based method finds the optimal solution.

The simulation results have shown that there is always a window size for which, the heuristic algorithm achieves a higher average utilization value than the one achieved by the greedy method. We can furthermore, draw the following conclusions:

As the quotient of the average TH capacity and the average VH capacity decreases or the deviation of VH capacities increases, the average utilization achieved by the proposed methods gets more significantly higher than the average utilization achieved by the greedy algorithm. The ILP based method can always achieve a higher average utilization than the heuristic method. However, it can only be applied if the quotient of the average TH capacity and the average VH capacity is relatively small and there are a relatively few THs, since this is the only case in which its running time is not unreasonable. The running time of the heuristic method is however, about the same as that of the greedy algorithm, while its maximal average utilization is approximately halfway between the maximal average utilization of the BLP method and the average utilization of the greedy algorithm (even in the cases where it does not perform significantly better than the greedy algorithm). Thus, the heuristic algorithm can be used more effectively than the greedy algorithm in almost any scenario, but it is the only option if the number of THs is big, and the quotient of the average TH capacity and average VH capacity is small. The scenario in which the greedy algorithm might be the best choice is the one in which, the deviation of VH capacities is relatively small and the quotient of the average TH capacity and the average VH capacity is relatively big. The reason for this is the following:

In this case, the ILP based algorithm cannot be used due to its huge running time. The heuristic algorithm on the other hand, does not achieve a maximal average utilization, which is significantly higher than the one achieved by the greedy algorithm. Moreover, while the greedy algorithm needs $O(|TH||VH|)$ steps to finish, a single execution of the heuristic algorithm needs $O(|TH||VH|c)$ steps, where c is the number of steps needed to sort the VHs before the assignment. This difference might not be significant in the practice, but the effective running time of the heuristic algorithm can be by orders of magnitude higher than that of the greedy algorithm, since in order to find the maximal average utilization provided by the heuristic algorithm, it has to be executed for multiple window sizes. According to the simulations, the time window size that gives the highest average utilization in the case of the heuristic algorithm, is between 0 and the average VH execution time, which is around 90 in the above scenarios, or it is slightly higher than the average VH execution time.

2.6 Conclusions

The goal of this chapter was to improve the efficiency of test component assignment in performance test environments. The motivation of the chapter was that test components are assigned to the testing hosts of the test environment in a greedy way and thus, I expected that by using more sophisticated methods, the average utilization of testing hosts can be increased. After formulating the load distribution problem, I have investigated methods created for solving the similar task assignment problem and found that because of the differences of the two problems, neither of these solutions are applicable for solving the load distribution problem.

In order to improve the average utilization of testing hosts reached by the greedy method, I have defined different heuristic solutions for solving the problem. After evaluating the efficiency of my heuristics against that of the greedy method, I found that in most of the cases, the heuristics reach a significantly higher average utilization of testing hosts than the greedy algorithm.

Chapter 3

A Model-Driven Performance Testing Method for Communicating Systems

As I have mentioned in Chapter 1, unlike conformance testing, the field of black-box performance testing of communicating systems lacks an evolved theoretical background, including automatic methods for checking whether the SUT fulfills its performance requirements, for example whether it is able to serve a specified number of request messages within a second while serving a specified number of users in parallel. Unlike conformance tests, black box performance tests are mainly designed in an ad-hoc way in the industry. The disadvantage of this approach is the inaccuracy of the performance test results meaning that the deviation of ad-hoc performance measurements is larger than necessary, as we will see.

To solve the above described problem of performance testing, in this chapter, I introduce an automatic model-driven performance testing method. The presented method interacts with the SUT, and based on this interaction and on the functional characterics of the SUT, it creates its formal performance model. Based on this performance model, the method determines whether the SUT fulfills the performance requirement of serving the required number of request messages within a second while serving a given number of users.

This chapter is organized as follows: In Section 3.1, I give a summary on the related work in the subject. In Section 3.2, I discuss the performance requirements that I deal with, and the motivations of creating the proposed performance testing method. In Section 3.3, I formally define my performance model used for testing, then in Section 3.4, I show how to create the structure of the performance model of the SUT. In Section 3.5, by performing measurements on the SUT, I show how to complete the performance model of the SUT. In Subsections 3.5.1 and 3.5.2 I show how to calculate CW_{usr} and CE_{usr}, respectively, based on the performance model of the SUT. I close the section by experimental results comparing my performance testing method to an ad-hoc performance testing method used in the industry, in Section 3.6 and a summary of the results achieved in Section 3.7.

3.1 Related Work

As I have mentioned in Chapter 1, In the field of conformance testing there are formal methods for automatically checking whether a physical system conforms its specifications. These methods include:

- formal methods for modeling the functional requirements of the SUT, and
- methods for checking whether the SUT corresponds to this formal model

In the field of performance testing, there exist different solutions for defining performance tests. Schieferdecker et al. have proposed PerfTTCN [55], which extends TTCN-2 (Tree and Tabular Combined Notation ver. 2) [56]. It introduces many new language elements for describing performance testing environments and for conducting performance tests. As a language extension of TTCN-3 (Testing and Test Control Notation ver. 3) [57], Grabowski et al. introduce TimedTTCN-3 [58]. TimedTTCN-3 extends TTCN-3 and makes it capable of testing the real-time behavior of the SUT. Mingwei et al. extend concurrent TTCN (which is a version of TTCN-2) to be able to test the performance of communication protocols [59, 60].

The performance property I dealt with (the number of messages the system is able to process within a second while serving a given number of users) is however, more complex than the ones the above mentioned methods can measure. For the measurement of this property, ad-hoc methods are used in the industry [61]. Thus, my goal was to create a more efficient model-driven performance testing method during which, a formal performance model of the SUT is created and then the performance characterictics to be measured are derived from this performance model.

In the literature, there already existed papers written on performance modeling. In [48], Kemper et al. present a performance model for communicating systems, based on SDL (Specification and Description Language) [49]. Youness et al. [50] use a stochastic Petri net [51] while El-Karaksy et al. use a timed Petri net [52] for modeling the performance of a communication protocol [53]. Marsan et al. model the performance of CSMA/CD bus LANs by a timed Petri net based model [54]. These papers also *verify* the models presented, that is, they analytically prove that their models correspond to the specifications, but they do not deal with *validating a physical system* that is, they do not offer methods for testing whether a physical implementation corresponds to the performance model.

Thus, the solution for the problem was to create a performance model, which could represent a physical system and based on which, a performance testing method could be defined for calculating the number of messages the represented system is able to process within a second while serving a given number of users in parallel.

3.2 Performance Requirements and Motivations

As I have mentioned earlier, the presented method is given as an input CR_{usr}, which is the number of messages to be processed within a second while serving usr users (number of messages per second for short). The method evaluates the performance of the SUT automatically, in other words it determines the number of messages the SUT can process within a second while serving usr users simultaneously.

CR_{usr} as a performance requirement is ambiguous. Thus, I interpreted this notion in two ways. CR_{usr} can be disambiguated as CWR_{usr}, which is the required number of messages that the SUT has to be able to serve within a second, in worst case while serving usr users. By worst case, I mean that the SUT has to be able to serve CWR_{usr} messages per second given any sequence of requests it receives from the users. CR_{usr} can also be disambiguated as CER_{usr}, which is the expected number of messages the SUT has to be able to serve within a second while serving usr users. I am going to state as a requirement of my performance testing method the ability of measuring both CW_{usr}, which is the worst-case number of messages and CE_{usr}, which is the expected number of messages the SUT is able to serve within a second, while serving usr users simultaneously. The performance testing method introduced in this section attempts to solve the problems raised by the ad-hoc performance testing methods, which are widely applied in the industry nowadays [61].

The ad-hoc performance testing method measures the performance of the tested system in a straightforward way, by simply emulating the real-life environment the SUT will have to operate in. Thus, this method runs a number of software entities, each emulating a user that the SUT has to serve. The number of software entities run by the test environment equals usr. During the test, each emulated user sends one request message after another to the SUT, and counts the number of responses the SUT has sent in reply. At the end of the test, the number of messages the SUT is able to serve within a second is calculated as the quotient of the number of all responses the SUT has sent to the software entities, and the duration of the test. This method can calculate CE_{usr}. However, as we will see during the experiments, the deviation of the CE_{usr} values measured by this method is larger than necessary. The ad-hoc method is furthermore, unable to calculate CW_{usr}. Thus based on its measurements, we can only state that CW_{usr} is lower than or equal to the lowest CE_{usr} value measured by the ad-hoc method.

Contrarily to the ad-hoc method, the proposed method is capable of calculating both CE_{usr} and CW_{usr}. My method uses a formal performance model as a tool for conducting the performace test and for modeling the SUT. The proposed method has two phases. In the first phase, from the information already known about the SUT, and from measurements performed on the SUT, the test environment creates the performance model of the SUT. In the second phase, the method derives both CW_{usr} and CE_{usr} from the so created performance model, in an analytical way. The first phase is discussed in Sections 3.3 and 3.4, while the second phase is discussed in Section 3.5. As we will see during the experiments, my method gives correct measurement results that is, the CE_{usr} value it calculates is the CE_{usr} value calculated by the ad-hoc method, while the CW_{usr} value it calculates is lower than each ad-hoc measurement. As we will also see, the deviation of the CE_{usr} values measured by my method within a given amount of time is smaller than the deviation of the CE_{usr} values measured by the ad-hoc method within the same amount of time.

3.3 The Timed Communicating Finite Multistate Machine

In this section, I introduce the Timed Communicating Finite Multistate Machine model, which will be used for conducting the performance test. At the end of the test, this model is a performance model of the SUT. In the definition below, I first give the formal definition of the TCFMM, then give a brief informal description. After that, I explain the non-trivial constraints of the definition one-by-one. In the definition, τ_0 denotes the time of the beginning of the execution.

Definition 1 *The Timed Communicating Finite Multistate Machine (TCFMM) is a 10-tuple:*

$$\begin{aligned}
TCFMM = \;& (I, O, S, s_0, T, U, H, \delta, \chi, \sigma),\ \textit{where}\\
1.\;& T \subseteq I \times O \times S \times S \times \mathbb{R}^+\\
2.\;& \forall t_i, t_j \in T((t_i = (i_i, o_i, s_{from_i}, s_{to_i}, d_i) \wedge t_j = (i_j, o_j, s_{from_j}, s_{to_j}, d_j) \wedge\\
& s_{from_i} = s_{from_j} \wedge i_i = i_j) \Rightarrow t_i = t_j)\\
3.\;& \chi \in H \to U,\ \chi\ \textit{is bijective}\\
4.\;& s_0 \in S\\
5.\;& \sigma \in \mathbb{R}^+ \times U \to S \cup \{\oslash\}\\
6.\;& \forall(u \in U) : \sigma(\tau_0, u) = s_0\\
7.\;& \delta \in \mathbb{R}^+ \times H \times I \to S \times O \times \mathbb{R}^+\\
8.\;& \forall(t_i \in T, h \in H, \tau \in \mathbb{R}^+) : \sigma(\tau, \chi(h)) = s_{from_i} \Rightarrow \delta(\tau, h, i_i) = (s_{to_i}, o_i, d_i)\\
& \textit{and invalid inputs are dropped.}\\
9.\;& \forall(\tau, h, i, s, o, d : \delta(\tau, h, i) = (s, o, d)) :\\
& (\forall(\phi : 0 < \phi < d) : \forall(u \in U - \{\chi(h)\}) :\\
& \mathbf{P}(\sigma(\tau + \phi, u) = \oslash) = 1 \Rightarrow (\sigma(\tau + d, \chi(h)) = s \wedge \forall(\epsilon : 0 < \epsilon < d) :\\
& \sigma(\tau + \epsilon, \chi(h)) = \oslash)) \wedge (\neg(\forall(\phi : 0 < \phi < d) : \forall(u \in U - \{\chi(h)\}) :\\
& \mathbf{P}(\sigma(\tau + \phi, u) = \oslash) = 1) \Rightarrow \exists(d' : d' < d) :\\
& (\sigma(\tau + d', \chi(h)) = s \wedge \forall(\epsilon : 0 < \epsilon < d') : \sigma(\tau + \epsilon, \chi(h)) = \oslash)) \wedge\\
& \wedge \exists(t_i \in T) : s_{from_i} = \sigma(\tau, \chi(h)) \wedge s_{to_i} = s \wedge i_i = i \wedge o_i = o \wedge d_i = d
\end{aligned}$$

In the definition above, I is the set of inputs, O is the set of outputs of the TCFMM, S is the set of states with s_0 being the initial state, and T is the set of state transitions. H is the set of users communicating with the TCFMM, while U is the set of tokens in the TCFMM, each one belonging to a user. Finally, σ is the current state function, δ is the state transition function, and χ is the function which assigns each user to a token.

The TCFMM is an extension of the Finite State Machine model. It has states connected by transitions, just like an FSM. One of my goals was to extend the FSM model to be able to formally represent a communicating system which communicates with multiple users simultaneously. The other goal was to represent the number of messages the system is capable of serving within a second. To achieve the earlier goal, I have introduced tokens, while to achieve the latter goal, I have introduced the time parameter to the model. As a result, the TCFMM model works as follows:

The users communicating with the TCFMM send inputs to it and receive outputs from it. Each user h is represented by a token $\chi(h) = u$ in the TCFMM. Each token resides at a state of the TCFMM at some points in time, and is under state transition at some other points in time. Function $\sigma(\tau, u)$ returns the state at which token u resides at time τ. If $\chi(h) = u$, the current state of token u is the current state of the protocol instance (or server thread) communicating with user h. The state transitions of different tokens are performed in parallel. The TCFMM is called 'Multistate', because the tokens can be distributed amongst multiple states at a given time of execution. Calling function δ with parameters $\tau \in \mathbb{R}^+$, $h \in H$, and $i \in I$ corresponds to user h sending input i to the TCFMM at time τ. For this request, function δ returns the state (s) to which token $\chi(h)$ will be transferred, and the output (o) which is sent to user h. δ also returns delay d. Let us assume that for the whole duration of the state transition, the system is continuously stressed by requests from all users. Delay d is then the amount of time that has elapsed between time τ and the time of sending o to h and transferring $\chi(h)$ to s.

After the informal description of the TCFMM model above, let us now see what Constraints $1 - 9$ of Definition 1 declare.

According to Constraints 1 and 2 of the definition, each transition t_i is described by five parameters namely, input i_i, output o_i, originating state s_{from_i}, destination state s_{to_i}, and delay d_i. Furthermore, the input and originating state of a transition identifies the transition and thus, there cannot be two transitions with the same originating states and input values in set T. This latter requirement implicates that the TCFMM is deterministic. According to Constraint 3, function χ assigns exactly one token u_i to each user h_i. In Constraint 5, if the state of a token $u \in U$ equals $\oslash$, then token u is under transition (it does not have a state). Constraint 6 says that at the beginning of execution (at time τ_0), each token resides at initial state s_0. According to Constraint 8, if token $\chi(h)$ resides at the originating state of transition t_i at time τ, then $\delta(\tau, i_i, h) = (s_{to_i}, o_i, d_i)$ is a legal state transition. According to Constraint 9, for each state transition $\delta(\tau, h, i) = (s, o, d)$, the following is true: If each token $u \neq \chi(h)$ is under transition between τ and $\tau + d$ with a probability of 1, then at time $\tau + d$, token $\chi(h)$ is at state s, and until then, $\chi(h)$ is under transition. Otherwise, there exists a duration $d' < d$ for which, between τ and $\tau + d'$, $\chi(h)$ is under transition, and at time $\tau + d'$, $\chi(h)$ is at state s. A token u being under transition for a given period of time with a probability of 1 means that u might visit states within that period of time, but the amount of time u spends at a state, between two transitions is 0. According to Constraint 9 furthermore, for each state transition $\delta(\tau, h, i) = (s, o, d)$, there exists a transition t_i for which,

- the input i sent by user h to the TCFMM is the input of t_i,
- the output o returned by δ is the output of t_i,
- the originating state of t_i is the state at which token $\chi(h)$ resides at time τ,
- the destination state of t_i is the state to which δ transfers token $\chi(h)$, and
- the amount of time the state transition takes equals the delay of t_i.

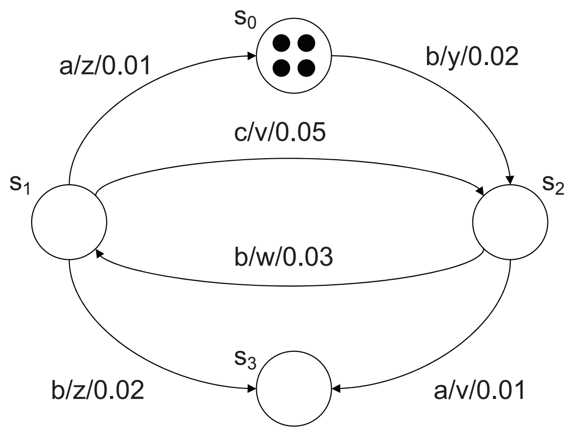

Figure 3.1: Graphical representation of the TCFMM

Figure 3.1 shows the graphical representation of a TCFMM. The transition parameters written on each transition are *input/output/delay*, respectively. All the tokens of the TCFMM reside in s_0.

3.4 Creating the Test Execution Model

In this section, I show how the test environment creates the functional structure of the TCFMM, which represents the SUT. The TCFMM created according to this section only models the functional behavior of the SUT; it has all the states, transitions, and tokens. Each of its transitions t_i has an input and an output value but its delay values d_i are yet unknown.

Beyond representing the functional structure of the SUT, the model created in this section is used for conducting the performance test during which the d_i value of each $t_i \in T$ is measured on the SUT. Once the transition delays are measured, a complete performance model of the SUT can be created. From this complete performance model, the test environment is able to derive CW_{usr}, and CE_{usr}, according to my testing method.

The functional structure of the TCFMM is based on the FSM model to which the SUT corresponds, according to the conformance test previously run on it. Let M'' denote the FSM to which the SUT corresponds. Following the FSM definition in [5], M'' is defined in the following way:

$$\begin{aligned}
&M'' = (I'', O'', S'', \delta'', \lambda''), \text{ where}\\
&\delta'' : S'' \times I'' \rightarrow S'' \text{ is the state transition function}\\
&\lambda'' : S'' \times I'' \rightarrow O'' \text{ is the output function}
\end{aligned}$$

Let $M = (I, O, S, s_0, T, U, H, \delta, \chi, \sigma)$ denote the TCFMM, which is used for conducting the performance test. M is constructed in two steps from M''. In the first step, based on M'', a TCFMM $M' = (I', O', S', s'_0, T', U', H', \delta', \chi', \sigma')$ is created. In the second step, a little modification is made to M', and so, M is constructed. M' is created from M'' according to the following assignments:

- $S' = S''$
- $s'_0 =$ the initial state of M''
- $O' = O''$
- $I' = I''$
- $U' = \{u_i | i = 1, ..., usr\}$
- $H' = \{h_i | i = 1, ..., usr\}$
- $\forall(h_i \in H') : \chi'(h_i) = u_i$
- $T' = \{t_i = (i_i, o_i, s_{from_i}, s_{to_i}, \oslash) : \exists(i \in I'', s \in S'') : s_{from_i} = s \wedge s_{to_i} = \delta''(s, i) \wedge$
  $\forall t_j = (i_j, o_j, s_{from_j}, s_{to_j}, \oslash) : (s_{from_i} = s_{from_j} \wedge i_i = i_j) \Rightarrow t_i = t_j\}$

Functions σ' and δ' are derived from the above assignments, and from Constraints

5 to 9 of Definition 1. According to the last assignment above, the delay value of each transition is unknown (equals $\oslash$). In order to make the TCFMM deterministic, the last assignment does not allow for multiple transitions with the same inputs and originating states and with different outputs or destination states.

In the second step, M' is transformed to $M = (I, O, S, s_0, T, U, H, \delta, \chi, \sigma)$, according to the following assignments:

- $I = I'$
- $O = O'$
- $s_0 = s'_0$
- $U = U'$
- $H = H'$
- $\forall(h_i \in H) : \chi(h_i) = u_i$
- $T = \{t_i | \exists(t_j \in T') : (\exists(t_k \in T') : s_{from_k} = s_{to_j} \wedge s_{from_j} = s_{from_i} \wedge s_{to_j} = s_{to_i} \wedge$
  $\wedge i_j = i_i \wedge o_j = o_i \wedge d_j = d_i) \vee$
  $\vee(t_{to_i} = s_0 \wedge \exists(t_j \in T') : (\nexists(t_k \in T') : s_{from_k} = s_{to_j}) \wedge$
  $\wedge s_{from_i} = s_{from_j} \wedge i_i = i_j \wedge o_i = o_j \wedge d_i = d_j)\}$
- $S = \{s_i | s_i \in S' \wedge \exists(t_j \in T') : s_{from_j} = s_i\}$

Functions σ and δ are derived from the above assignments, and from Constraints 5 to 9 of Definition 1. According to the above, M' is transformed to M by taking each transition which leads to a sink state (i.e. a state with no outgoing transitions) in M', and by redirecting these transitions to s_0 (these transitions will be called sink transitions from now on).

Before explaining why all sink transitions had to be redirected to s_0, let us see what it means that M is used for conducting the performance test. Placing *usr* tokens into M means that during the performance measurement, the test environment will emulate the maximal number of users the SUT has to be able to handle. During testing, moving token u along transition t_i from state s_{from_i} to state s_{to_i} corresponds to the test environment sending input i_i to the SUT and then waiting to receive output o_i from the SUT, in the name of user h, where $\chi(h) = u$. In other words, the test environment uses the TCFMM for tracing the state changes of all the protocol instances (or server threads) running on the SUT.

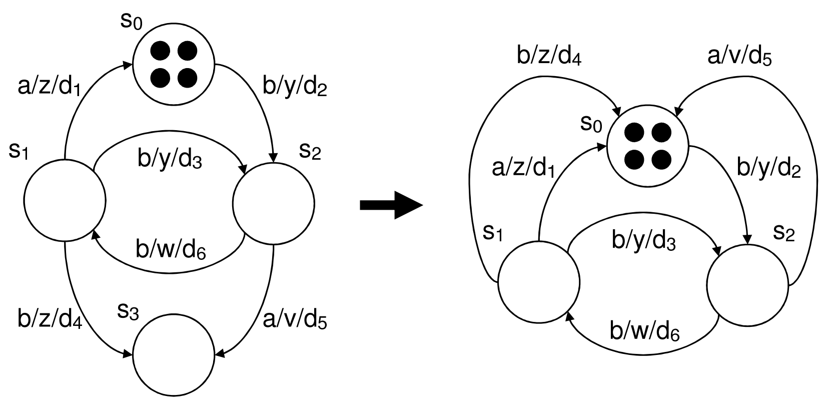

Figure 3.2: Redirecting sink transitions to s_0

The reason for redirecting sink transitions to s_0 is the following: Let us assume that M' is used for conducting the performance test. According to the above, M' can contain sink states. If a token u reaches one of these sink states s, the tester will not be able to send any request messages (inputs) to the SUT in the name of user h, where $\chi(h) = u$. That is, the effective number of users the SUT has to serve simultaneously will be decreased by one as a consequence of token u being stuck at s. In other words, token u will go inactive. If however, M is used for conducting the performance test that is, if each sink transition is redirected to s_0, then every time a token goes through one of these sink transitions, it reappears at s_0 instead of going inactive. This is identical to the situation when for each user h that sends its last request to the SUT, a new user h' appears. As h' is a new user, the input messages it is allowed to send to the SUT are the inputs of the transitions leading out of s_0. As an example, Figure 3.2 shows two TCFMMs. The TCFMM in the left side of the figure has a single sink state, s_3. The TCFMM in the right side of the figure is created from the TCFMM on the left by redirecting all the sink transitions to s_0.

3.5 Performance Evaluation

In this section, I am going to show how to measure the transition delays of the SUT and thus, how to make M a complete performance model of the SUT. I am also going to show, how to calculate CW_{usr}, and CE_{usr} from the measured d_i values.

During testing, the test environment runs a number of user entities in parallel.

The number of user entities used for testing equals usr. Each user entity plays the role of a user, and exchanges messages with the SUT. No user entity can be idle during testing that is, upon receiving an output o_i from the SUT, the user entity has to send an input i_j to the SUT immediately, where $s_{to_i} = s_{from_j}$. This way, the SUT is forced to process usr requests in parallel at all times.

Let us now see how the delay d_i of transition t_i is measured. Delay d_i is calculated based on multiple delay measurements the values of which are stored in set $D_i = \{d_{ij} | j = 1, 2, \ldots\}$. Let us assume that token u_k resides at s_{from_i} in M. When the user entity representing user h_k sends input i_i to the SUT, it starts to measure the amount of time elapsed until receiving o_i from the SUT. Delay $d_{i(|D_i|+1)}$ (that is, the next element of D_i) will be the amount of time elapsed between sending i_i to and receiving o_i from the SUT. After the performance measurement is over, each delay d_i is calculated from the elements of D_i, as follows:

$$d_i = \frac{1}{|D_i|} \sum_{j=1}^{|D_i|} d_{ij} \tag{3.1}$$

Thus, d_i is the average amount of time that the SUT needs to respond to i_i, while continuously being stressed by usr requests. This corresponds to the definition of d in Constraint 10 of Definition 1.

Once d_i is known for each $t_i \in T$, M is a complete performance model of the SUT. Based on M, CW_{usr} and CE_{usr} can be calculated as described in the following.

3.5.1 Calculating the Worst-case Performance of the System Under Test

In this subsection, I show how to calculate CW_{usr} based on M. First, let us define what it means that a system is able to process CWR_{usr} messages per second in worst case.

Definition 2 *The SUT is said to be able to process CWR_{usr} messages per second if for an arbitrary infinite input sequence s,*

$$\lim_{t \to \infty} \frac{F_t^s}{t} \geq CWR_{usr} \tag{3.2}$$

In the formula, F_t^s denotes the number of state transitions of the SUT measured for time length t while inducing the SUT by s.

The above fraction is the reciprocal of the average amount of time needed to process one input message of s. Since the amount of time needed to process any input sequence of s equals a transition delay which takes its value from a finite set, this average delay does have a limit and thus, the limit in the above formula exists too.

According to the above formula, a system is said to be able to process CWR_{usr} messages per second in worst case if it processes at least CWR_{usr} messages per second when induced by an *arbitrary* infinite sequence of inputs. The number of messages the SUT processes within a second is calculated as an average for a long (optimally infinite) period of time. This way, the definition hides transient stages when the SUT underperforms and needs more time than usual to process an input message. Thus, this definition prevents systems that underperform for a relatively short time from failing the performance test. This corresponds to the real-life situation, where users do not care if the system they use slows down for a short while if otherwise, it performs as expected.

Let us now see what requirement the SUT has to meet in order to fulfill Formula 3.2. In other words, let us give the sufficient and necessary requirement of a system corresponding to Formula 3.2.

As a first attempt let us examine, whether the following is a suitable sufficient and necessary requirement:

$$\forall(i : t_i \in T) : d_i \leq \frac{1}{CWR_{usr}} \tag{3.3}$$

A system fulfilling Formula 3.3 does fulfill Formula 3.2, since the delay of each transition is lower than or equal to the reciprocal of CWR_{usr} and thus, each transition *by itself* is able to serve CWR_{usr} messages per second. As we will see however, this formula is only a sufficient but not a necessary requirement, thus Formula 3.2 can be fulfilled even when Formula 3.3 is violated.

Before giving a sufficient and necessary requirement for fulfilling Formula 3.2, let us define set C as follows:

$$\begin{aligned} C = \; & \{c_i | c_i = \{t_j, t_{j+1}, \ldots, t_{j+m}\} \wedge \forall(n : 0 \leq n \leq m \wedge n \in \mathbb{N}) : t_n \in T \wedge \\ & \wedge s_{from_j} = s_{to_{j+m}} \wedge \forall(n : 1 \leq n \leq m \wedge n \in \mathbb{N}) : s_{to_{n-1}} = s_{from_n} \wedge \\ & \wedge \nexists(t_k \in c_i, t_l \in c_i) : (t_k \neq t_l \wedge s_{to_k} = s_{to_l})\} \end{aligned} \tag{3.4}$$

That is, C is the set of all transition cycles in the TCFMM. In the above formula and the rest of the thesis, cycles are considered to be ordered sets of transitions. After all the above, the sufficient and necessary requirement of a system processing CWR_{usr} messages per second in worst case is as follows:

$$\forall(c_i \in C) : \sum_{t_j \in c_i} d_j \leq \frac{|c_i|}{CWR_{usr}} \tag{3.5}$$

In the following, I am going to prove the sufficiency and necessity of this requirement.

Proof: To prove the sufficiency of Formula 3.5, let us imagine an infinite random walk in the TCFMM. Let us maintain a multiset TT of the traversed transitions. When a transition is traversed, it is added to the multiset. If the newly added transition constitutes a cycle c_i with some other transitions in the multiset, each transition $t_i \in c_i$ is immediately removed from the multiset. Before going on, a lemma has to be proven:

Lemma 1 $|TT| \leq |T|$.

Proof: According to the above, the loops are removed immediately from TT. Thus, TT contains a subgraph of one of the spanning trees of the TCFMM. Since any spanning tree of the TCFMM contains $|S| - 1$ transitions, $|TT| \leq |S| - 1$. Furthermore, since the TCFMM is strongly connected, $|T| \geq |S|$. Hence, $|T| - 1 \geq |TT|$. ■

As a consequence of Lemma 1, in M, a relatively long (infinite) walk is composed of a relatively big (infinite) number of cycles and some transitions not belonging to any cycles (standalone transitions from now on). Let us assume that Formula 3.5 is true. Then the following is true for each traversed cycle c_i:

$$CWR_{usr} \sum_{t_j \in c_i} d_j \leq |c_i| \tag{3.6}$$

Each cycle c_i might be traversed more than once during the walk. Let $\mathcal{T}_i$ denote the number of times c_i was traversed during the walk. The above inequality can be further transformed as follows:

$$CWR_{usr} \mathcal{T}_i \sum_{t_j \in c_i} d_j \leq \mathcal{T}_i |c_i| \tag{3.7}$$

Since the above inequality is true for each traversed cycle c_i, the following inequality will also be true:

$$CWR_{usr} \sum_{c_i \in C} \left(\mathcal{T}_i \sum_{t_j \in c_i} d_j \right) \leq \sum_{c_i \in C} \mathcal{T}_i |c_i| \tag{3.8}$$

The above can be further transformed as follows:

$$\frac{\sum\limits_{c_i \in C} \mathcal{T}_i |c_i|}{\sum\limits_{c_i \in C} \left(\mathcal{T}_i \sum\limits_{t_j \in c_i} d_j \right)} \geq CWR_{usr} \tag{3.9}$$

The amount of time needed to traverse a cycle c_i once is $\sum\limits_{t_j \in c_i} d_j$, while the number of transitions that occur during a single traverse of this cycle is $|c_i|$. Based on this, F_t^s and t from Formula 3.2 are as follows:

$$F_t^s = \sum_{c_i \in C} \mathcal{T}_i |c_i| + |TT| \tag{3.10}$$

$$t = \sum_{c_i \in C} \left(\mathcal{T}_i \sum_{t_j \in c_i} d_j \right) + \sum_{t_i \in |TT|} d_i \tag{3.11}$$

In the above formulas, $|TT|$ is the number of standalone transitions at the end of the traverse, while $\sum\limits_{t_i \in |TT|} d_i$ is the amount of time needed to traverse these standalone transitions. Based on Formulas 3.10 and 3.11, for our long walk, Formula 3.2 will be as follows:

$$\frac{\sum\limits_{c_i \in C} \mathcal{T}_i |c_i| + |TT|}{\sum\limits_{c_i \in C} \left(\mathcal{T}_i \sum\limits_{t_j \in c_i} d_j \right) + \sum\limits_{t_i \in |TT|} d_i} \geq CWR_{usr} \tag{3.12}$$

We have to prove that if Formula 3.5 is true then Formula 3.2 is true. In our walk, the number of cycle traverses are optimally infinite, while according to Lemma 1, the number of traversed standalone transitions is limited, and thus, the amount of time spent by traversing cycles is optimally infinite, while the amount of time spent by traversing standalone transitions is limited. Thus, in the nominator of Formula 3.12, $\sum\limits_{c_i \in C} \mathcal{T}_i |c_i|$ outweighs $|TT|$, and in its denominator, $\sum\limits_{c_i \in C} \mathcal{T}_i \sum\limits_{t_j \in c_i} d_j$ outweighs $\sum\limits_{t_i \in |TT|} d_i$.

This means that as the duration of the walk goes to infinity, the left side of Formula 3.2 holds to the left side of Formula 3.9. And finally, since Formula 3.9 is true, in the case of an optimally infinite walk, Formula 3.12 will be true.

To prove the necessity of Formula 3.5, let us take the indirect way again. Let us assume that the system fulfills Formula 3.2, even though there is a cycle c_i for which, the following is true:

$$\sum_{t_j \in c_i} d_j > \frac{|c_i|}{CWR_{usr}} \tag{3.13}$$

Let us assume that our infinite walk only traverses c_i over and over again, and the total number of traverses of c_i is $\mathcal{T}_i$. Then the total number of transitions F_t^s, and the total amount of time of our walk t will be as follows:

$$F_t^s = \mathcal{T}_i |c_i| \tag{3.14}$$

$$t = \mathcal{T}_i \sum_{t_j \in c_i} d_j \tag{3.15}$$

Formula 3.13 can be transformed as follows:

$$CWR_{usr} > \frac{\mathcal{T}_i |c_i|}{\mathcal{T}_i \sum\limits_{t_j \in c_i} d_j} \tag{3.16}$$

Based on Formulas 3.14, and 3.15, Formula 3.16 states that the total number of transitions in the walk divided by the total amount of time of the walk is smaller than CWR_{usr}, which contradicts Formula 3.2.

■

Let us now transform Formula 3.5 as follows:

$$\forall (c_i \in C) : CWR_{usr} \leq \frac{|c_i|}{\sum\limits_{t_j \in c_i} d_j} \tag{3.17}$$

To find CW_{usr}, we have to find the maximal messages per second value that if we substitute CWR_{usr} with, the above formula will be true for each cycle. This value is as follows:

$$CW_{usr} = \min_{c_j \in C} \left\{ \frac{|c_i|}{\sum\limits_{t_j \in c_i} d_j} \right\} \tag{3.18}$$

Thus, the above formula takes the cycles that if the the SUT traverses, it processes the lowest number of messages within a second (bottleneck cycles from now on). CW_{usr} is the number of messages per second we measure while traversing these bottleneck cycles.

3.5.2 Calculating the Expected Performance of the System Under Test

If the SUT corresponds to Formula 3.5, it is able to serve CWR_{usr} messages per second in worst case. However, the users communicating with the SUT in its latter real-life environment might send an input message to the system more likely than another input message at a given state of execution. This behavior of users assigns different probabilities to transitions having the same originating state. The users will experience that the number of messages the SUT processes within a second is CW_{usr} exactly if they force the SUT to traverse the bottleneck cycles over and over again. Due to the different transition probabilities however, the probability of traversing these bottleneck cycles might be extremely low. In this case, the users experience that the number of messages the SUT processes within a second is higher than CW_{usr}. In this subsection, I show how to calculate CE_{usr}, which is the number of messages the system processes within a second, according to the experience of the users.

Let us assume that for a user h, token $\chi(h)$ resides at state s_{from_i}. Then the probability of t_i, p_i denotes the probability of user h sending i_i to the SUT (p_i is equal for each user). Let us furthermore define p_{kl} as follows:

$$p_{kl} = \sum_{i:t_i \in T \wedge s_{from_i} = s_k \wedge s_{to_i} = s_l} p_i \tag{3.19}$$

Thus, if $\chi(h) = s_{from_i} = s_k$, p_{kl} is the probability of user h sending to the SUT the input message of any transition leading from s_k to s_l. In other words, p_{kl} is the probablility of token $\chi(h)$ transferring from s_k to s_l (p_{kl} is equal for each user and token). Let furthermore z_i denote the *stationary state probability* of state s_i that is, the probability of token u transferring to state s_i *at any time* (z_i is equal for each token and user). To calculate CE_{usr}, we first have to calculate z_i for each state s_i from the following equation system, where $n = |S| - 1$ that is, the number of states in M minus one:

$$\begin{aligned}
z_0 &= z_0 p_{00} + z_1 p_{10} + ... + z_n p_{n0} \\
z_1 &= z_0 p_{01} + z_1 p_{11} + ... + z_n p_{n1} \\
&\vdots \\
z_n &= z_0 p_{0n} + z_1 p_{1n} + ... + z_n p_{nn}
\end{aligned} \tag{3.20}$$

To explain the above, $z_k p_{kl}$ is the probability of a token u transferring from s_k to s_l *at any time*. By summing up the $z_j p_{jl}$ values for each $j : s_j \in S$ on the right side of the equations, we thus get the probability of an arbitrary token u transferring to s_l at any time. In the above equation system, one equation is unnecessary, since it can be expressed by the other equations. In the following, I am going to express the last equation by the preceding ones. First, let us add up the first n equations, and then substract $(z_0 + z_1 + ... + z_n)$ from both sides of the sum. The resulting equation is the following:

$$\begin{aligned}
-z_n &= z_0(p_{00} + p_{01} + ... + p_{0(n-1)} - 1) \\
&+ z_1(p_{10} + p_{11} + ... + p_{1(n-1)} - 1) \\
&\vdots \\
&+ z_n(p_{n0} + p_{n1} + ... + p_{n(n-1)} - 1)
\end{aligned}$$

Since $p_{j0} + p_{j1} + ... + p_{j(n-1)} + p_{jn} = 1$ for each $j = 1, ..., n$, it is legal to replace the coefficient of each z_i as follows:

$$-z_n = z_0(-p_{0n}) + z_1(-p_{1n}) + ... + z_0(-p_{nn})$$

By multiplying the above equation by -1 we get the last equation of the original equation system which is therefore, unnecessary. After removing the last equation, the number of equations will be n, while the number of variables will still be $n + 1$ thus, the equation system will not have a definite solution. The equation system however, does not yet imply that the probability of a token u transferring to any of the states is 1, which is expressed by the following equation:

$$z_0 + z_1 + ... + z_n = 1$$

By adding the above equation to the equation system, the number of variables will equal the number of equations. The final form of the equation system is as follows:

$$\begin{aligned}
0 &= z_0(p_{00} - 1) + z_1 p_{10} + \ldots + z_n p_{n0} \\
0 &= z_0 p_{01} + z_1(p_{11} - 1) + \ldots + z_n p_{n1} \\
&\vdots \\
0 &= z_0 p_{0(n-1)} + \ldots + z_n(p_{(n-1)(n-1)} - 1) + \\
&+ z_n p_{n(n-1)} \\
1 &= z_0 + z_1 + \ldots + z_n
\end{aligned}$$

The above equation system is identical to the following matrix equation:

$$\boldsymbol{F}\boldsymbol{z} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \text{ where } \boldsymbol{z} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_n \end{bmatrix}, \text{ and } \boldsymbol{F} = \begin{bmatrix} p_{00} - 1 & p_{10} & \ldots & p_{n0} \\ p_{01} & p_{11} - 1 & \ldots & p_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0(n-1)} & p_{1(n-1)} & \cdots & p_{n(n-1)} \\ 1 & 1 & \ldots & 1 \end{bmatrix} \quad (3.21)$$

If $\det \boldsymbol{F} \neq 0$, the above matrix equation gives a definite solution for the z_i values [62]. Based on the z_i values, CE_{usr} is calculated as follows, where z_{from_i} is the stationary state probability of state s_{from_i}:

$$CE_{usr} = \frac{1}{\sum\limits_{t_i \in T} d_i z_{from_i} p_i} \quad (3.22)$$

In the above formula, $z_{from_i} p_i$ is the stationary firing probability of transition t_i that is, the probability of a user h sending i_i to the SUT *at any time* (this probability is equal for each user). The formula weighs the delay d_i of each transition by the stationary firing probability of the transition and then sums up this product for each transition t_i. This way in the denominator, we get the expected transition delay of the SUT, which is the expected delay between sending an input and receiving an output from the SUT. The reciprocal of this expected transition delay is the expected number of messages the SUT is able to process within a second or in other words, CE_{usr}.

3.6 Experimental Results

In this section, I present experimental results comparing my performance testing method to the ad-hoc method. With the experiments, my goals were:

- to prove the correctness of my method meaning that my method calculates the same CE_{usr} value as the ad-hoc method,
- to prove that my method is more accurate than the ad-hoc method that is, given the same amount of time for testing, the deviation of the CE_{usr} values measured by my method is lower than the deviation of the values measured by the ad-hoc method, and
- to show that the CW_{usr} value calculated by my method is lower than any value measured by the ad-hoc method.

During the experiments, the test environment realized 100 users in parallel towards the SUT, which was a separate, physical host. As in the practice of performance testing, the users realized by the test environment are not physical, but emulated users which, from the viewpoint of the SUT, seem to be separate, physical users. Thus, the SUT experiences that it has to serve 100 separate, physical users. The test environment can either realize the ad-hoc testing method or the performance testing method proposed above. The experimental environment used for obtaining the measurements presented in this section is shown in Figure 3.3.

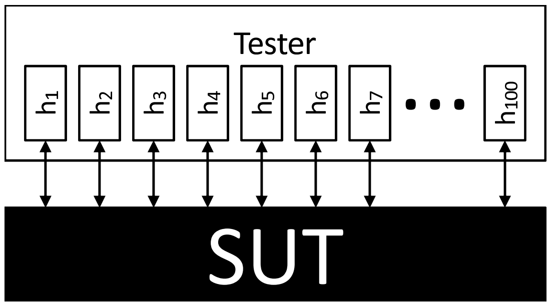

Figure 3.3: Experimental environment

The SUT host is given the number of users to be served, and the FSM it has to realize towards each served user. For each user connecting to the SUT, it creates a server thread which realizes the given FSM towards the user, separately from all the other server threads. The only aspect that I simulated during my experiments was the work the SUT needed to respond to different requests. This is simulated as follows: The SUT is given as an input an integer number w assigned to each legal state transition in the realized FSM thus, to each legal state-input pair (s, i). The SUT is furthermore given a function $M(\epsilon)$, which generates coefficients m_h with a given distribution, where $1 - \epsilon < m_h \leq 1 + \epsilon$. The SUT host works as follows: Let us assume that the server thread communicating with user h is at state s, and in the FSM, $\delta''(s, i) = s^*$, and $\lambda''(s, i) = o$. Upon receiving input i from user h, the server thread starts to increment an integer by one starting from zero, step by step. Once this integer reaches wm_h, where m_h is newly generated, the SUT sends o to user h, and transfers to state s^*. If the thread reaches a sink state, it resets itself to s_0 immediately to serve a new user. Counting from 0 to wm_h simulates the work or processor time needed by the SUT to respond for i. Value w represents the mean processor time needed to respond for i. To get the real processor time that the SUT needs to respond for i at a given point in time, w is multiplied by the newly generated coefficient m_h. Each time the server thread is at state s and user h sends i to it, a new m_h value is generated, and the real processor time will be wm_h. Thus, the value of wm_h will have a distribution. The reason for spreading the processor times needed by the SUT to answer for different requests is that, a performance tested system has some properties, which are not directly observable when regarding the SUT as a black box. As an effect of these properties however, the performance tested system might need different amounts of processor time to respond to a given input message at different occasions and thus, the users communicating with the SUT might experience that the response delays of the SUT are not constant at different points in time. The reason for simulating the processor time needed by the SUT to respond to different requests is that this way, I was able to manipulate the processor time distribution of the SUT and to observe how my performance testing method performs when using different M functions. In other words, I was able to manipulate the input of the compared performance testing methods (the SUT), and observe how they perform in different scenarios.

When executing the ad-hoc performance testing method, the test environment is given as an input usr, the FSM implemented by the SUT, and positive number D, which is the duration of the performance test. It is furthermore given a probability p assigned to each legal state transition that is, to each legal state-input pair (s, i) of the FSM. Let us assume that the server thread serving user h is at state s. Probability p is then the probability of user h sending i to the SUT. If t_i is a transition of M', $i_i = i$, $o_i = \lambda''(s,i)$, $s_{from_i} = s$, and $s_{to_i} = \delta''(s,i)$, then probability p of state-input pair (s,i) equals p_i.

The ad-hoc tester runs usr user instances in parallel ($usr = 100$, in our case). Each user instance works as follows: In the beginning, each server thread in the SUT is at state s_0. Based on the p probabilities assigned to each legal state-input pair (s,i), where $s = s_0$, it chooses and sends an input i^* to the SUT, and waits for o^*, where $\lambda''(s_0,i^*) = o^*$, and $\delta''(s_0,i^*) = s^*$. Upon receiving o^* from the SUT, it chooses an input i^{**} based on the p probabilities, where (s^*, i^{**}) is a legal state-input pair and sends it to the SUT immediately, and so on. If the user instance drives its server thread to a sink state, it imediately starts to act as a new user. This means that the user instance immediately sends an input i^{***} to the SUT, where (s_0, i^{***}) is a legal state-input pair. This goes on until D is elapsed. Each user instance counts the number of messages received from the SUT. Let N_i denote the number of messages that user instance i has received from the SUT. The number of messages the SUT is able to process within a second is then calculated as follows: First, the number of outputs that each user instance has received from the SUT is summed up, and then this sum is divided by the duration of the performance test that is, $\frac{\sum_{i=1}^{usr} N_i}{D}$ is calculated. As I have mentioned earlier, the ad-hoc component outputs CE_{usr} but it is unable to calculate CW_{usr}.

When executing the proposed performance testing method, the test environment is given as an input usr, the structure of the TCFMM created according to Section 3.4, and an integer I, which is the number of times each transition delay of the SUT has to be measured (or the number of iterations). At the end of the performance test, $\forall(i : t_i \in T) : I = |D_i|$. The test environment is furthermore given the probability p_i of each transition t_i. In this case, the test environment works as described in Section 3.5 and outputs CW_{usr} and CE_{usr}.

I have run three series of experiments, each series with a different function $M(\epsilon)$. This means that in each series of experiments, the measured transition delays had a different distribution. In each series, I have run 100 ad-hoc tests, where $D = 10, 20, \ldots, 1000$, and 50 measurements by the proposed method. The number of iterations of the proposed method was $I = 1, 2, \ldots, 5$ and for the same iteration count I, I have run 10 measurements by the proposed method.

In the deviation figures of this section, the deviation belonging to time t is obtained as follows: In the case of the ad-hoc method, the 20 values for which, the amount of time needed for the measurement was in the interval $(t - 200; t]$ are taken and the deviation of these values is calculated. In the case of the proposed method, one deviation value is calculated from those measurements, which used the same iteration count I. Thus, for the proposed method, I have plotted five discrete deviation values, one for each I value.

In the tables of this section, D_{ad-hoc} denotes the deviation of measurements of the ad-hoc method, while $D_{proposed}$ denotes the deviation of measurements of the proposed method in messages per second.

In the first series of experiments, $M(\epsilon) = 1.0$ that is, the work (processor time) needed by the SUT to answer i_i was the same each time the tester sent i_i to the SUT.

Figure 3.4 shows the CE_{usr} values measured using each method, and the CW_{usr} values measured by the proposed method, as a function of the running time of the performance test. To obtain each marker in the figure and in all the figures of this section, the tester has run a measurement emulating 100 users. The running time of the test is determined by I in the case of the proposed method and by D, in the case of the ad-hoc method.

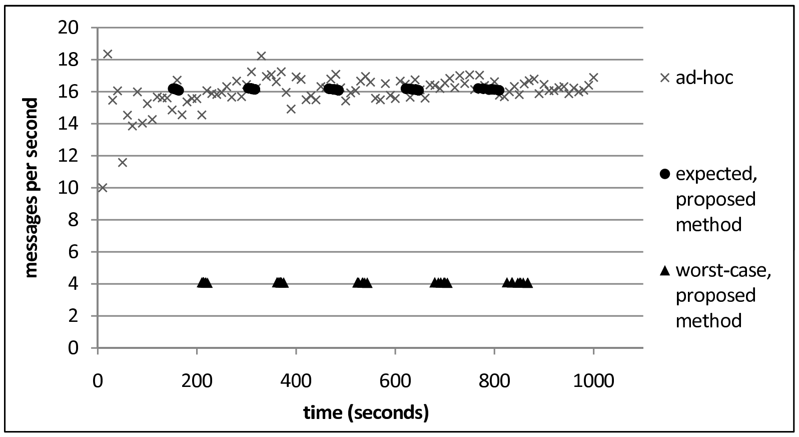

Figure 3.4: Measurements taken by the ad-hoc and the proposed method, in the first series of experiments

The mean value of CE_{usr} is the same for both methods, which proves that in this case, my method gives a correct CE_{usr} value. The deviation of the measurements of the proposed method is however, lower than that of the measurements taken by the ad-hoc method.

As it can also be seen in Figure 3.4, all of the CE_{usr} values measured by the ad-hoc method are higher than any of the CW_{usr} values measured by the proposed method and there is a relatively big difference between the lowest CE_{usr} value and the CW_{usr} values. However, as it can be seen in Figure 3.5, the values measured by the ad-hoc method can approach the worst-case values. To achieve this, in the ad-hoc tester, the transition probabilities of users have to be adjusted so that each user is likely to traverse one of the bottleneck cycles.

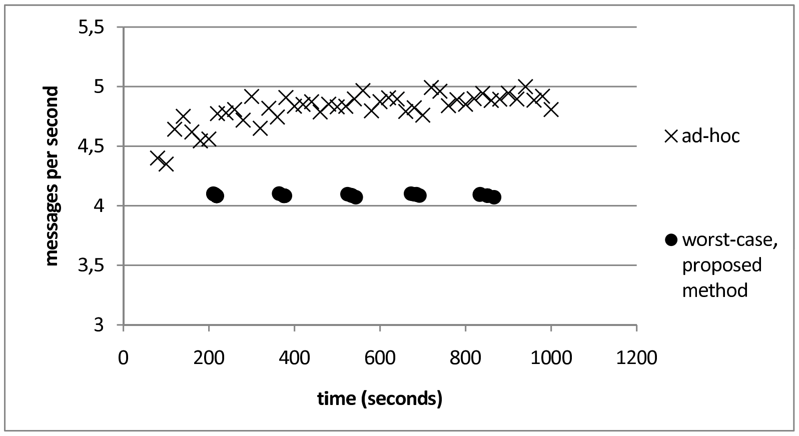

Figure 3.5: Measurements of the ad-hoc method approaching CW_{usr}, in the first series of experiments

Returning to the accuracy of the methods, Figure 3.6 and Table 3.1 show the deviations of the measured CE_{usr} values for both methods, as a function of the running time of the test. Figure 3.6 has a logarithmic vertical axis.

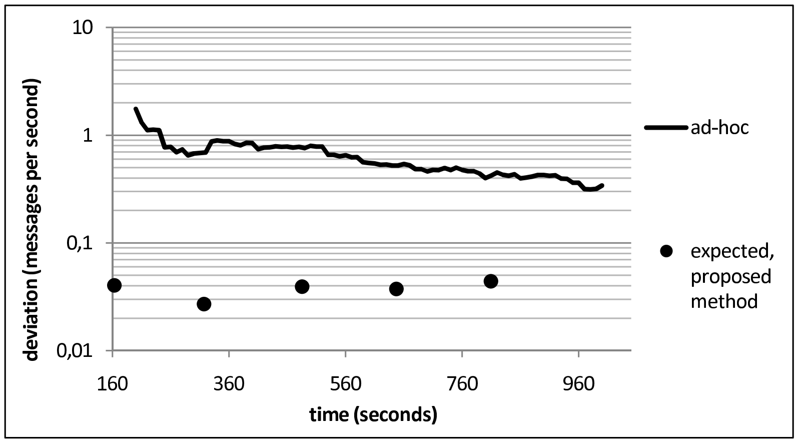

Figure 3.6: Deviation of measurements of the ad-hoc and the proposed method, in the first series of experiments

Time (seconds±5 seconds)	D_{ad-hoc}	$D_{proposed}$
320	0.69	0.03
400	0.77	0.04
650	0.52	0.04
810	0.42	0.04

Table 3.1: Measurement deviations of the ad-hoc and the proposed method, in the first series of experiments

According to Figure 3.6 and Table 3.1, given the same amount of time for testing, the deviation of the measurements obtained by the proposed method is lower by an order of magnitude than the deviation of the ad-hoc measurements.

In the second and third series of experiments, I have examined what happens to the measured CE_{usr} values if function $M(\epsilon)$ does not generate constant m_i coefficients, and thus, the response delays of the SUT are not constant. In other words, in the second and third series of experiments, the processor time needed by the SUT to process a given input message is not constant.

Figure 3.7 shows the histogram of values generated by function $M(1)$, in the second series of experiments. According to the figure, The distribution of the m_h coefficients is close to a normal distribution.

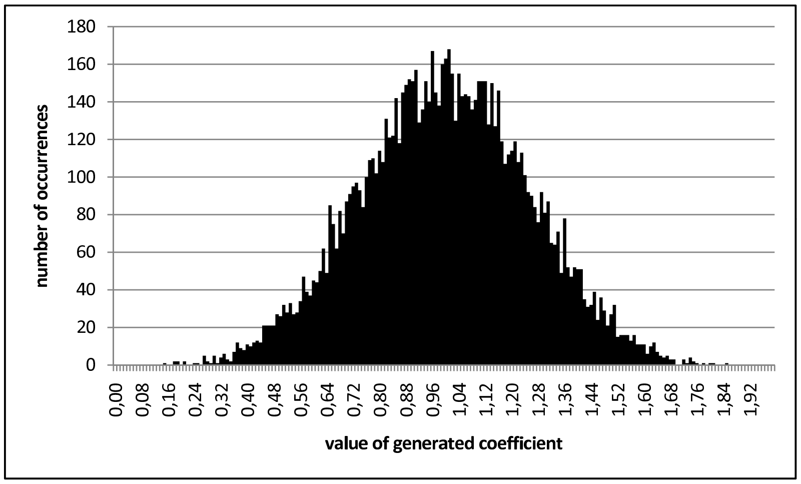

Figure 3.7: Histogram of $M(1)$ used in the second series of experiments

Figure 3.8 shows the CE_{usr} values measured by each method, in the second series of experiments. The mean value of the measurements is the same for both methods thus, the proposed method gives correct results in this case as well.

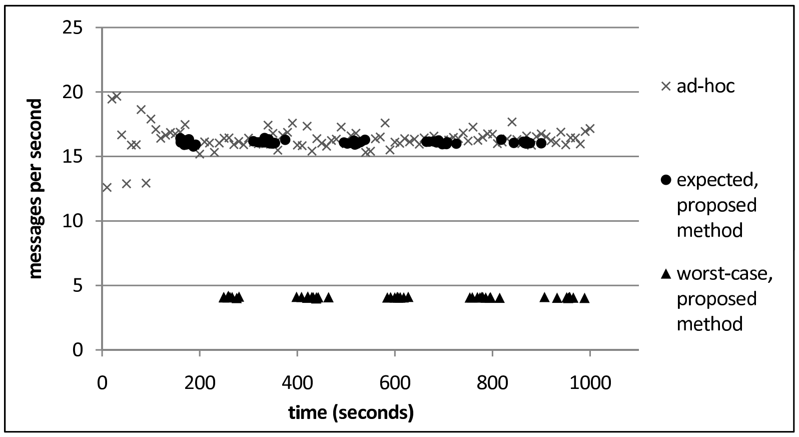

Figure 3.8: Measurements taken by the ad-hoc and the proposed method, in the second series of experiments

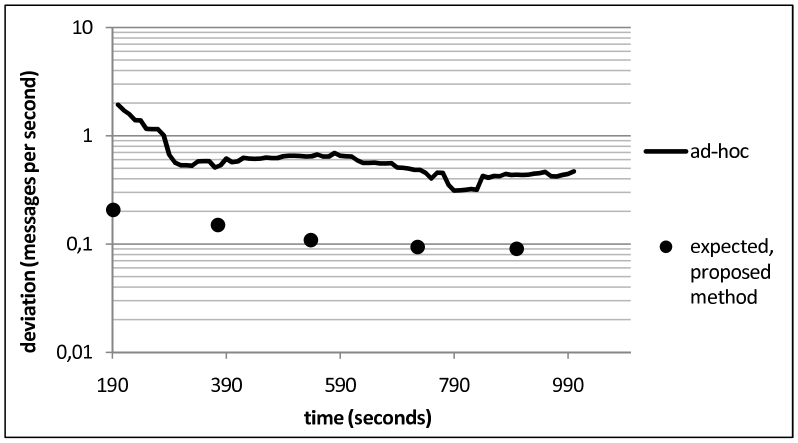

Figure 3.9: Deviation of measurements of the ad-hoc and the proposed method, in the second series of experiments

Time (seconds±5 seconds)	D_{ad-hoc}	$D_{proposed}$
390	$0,53$	0.15
540	0.64	0.11
730	0.48	0.09
900	0.44	0.09

Table 3.2: Measurement deviations of the ad-hoc and the proposed method, in the second series of experiments

Figure 3.9 having a logarithmic vertical axis and Table 3.2 show the deviation of the CE_{usr} values measured by each method, in the second series of experiments.

According to the figure, and the table, the deviation of measurements of the proposed method is still significantly lower than the deviation of the measurements of the ad-hoc method.

Figure 3.10 shows the histogram of values generated by function $M(1)$ in the third series of experiments. In this series, the distribution of the m_h values is a uniform distribution on interval $(0, 2]$.

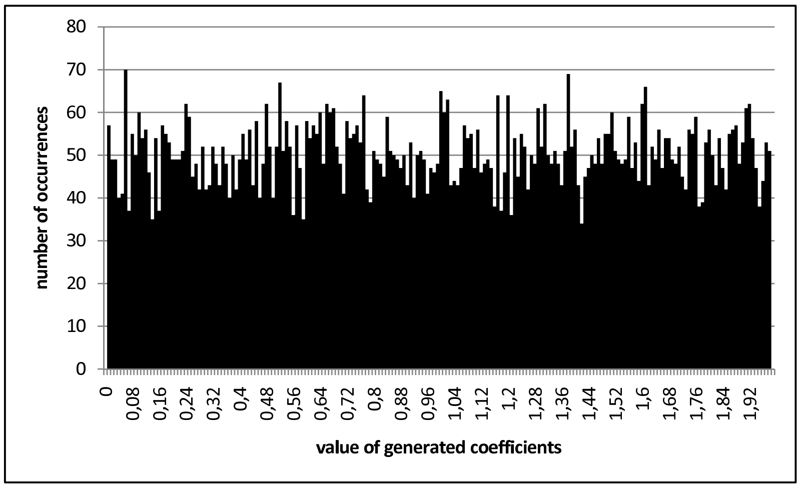

Figure 3.10: Histogram of $M(1)$ used in the third series of experiments

Figure 3.11 shows the CE_{usr} values measured by each method, in the third series of experiments. The mean values of measurements of the two methods are the same thus, the proposed method gives correct results in this case as well.

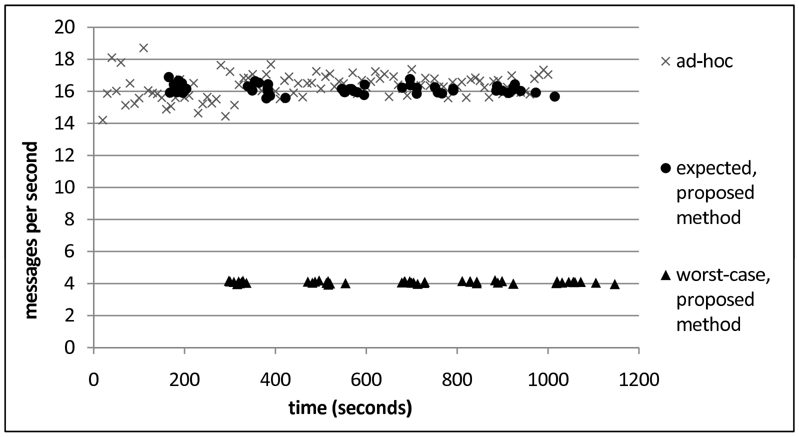

Figure 3.11: Measurements taken by the ad-hoc and the proposed method, in the third series of experiments

Figure 3.12 and Table 3.3 show the deviation of the CE_{usr} values measured by each method, in the third series of experiments. According to the figure and the table, even in this extreme case, the deviation of the values measured by the proposed method is approximately the half of the deviation of the ad-hoc measurements.

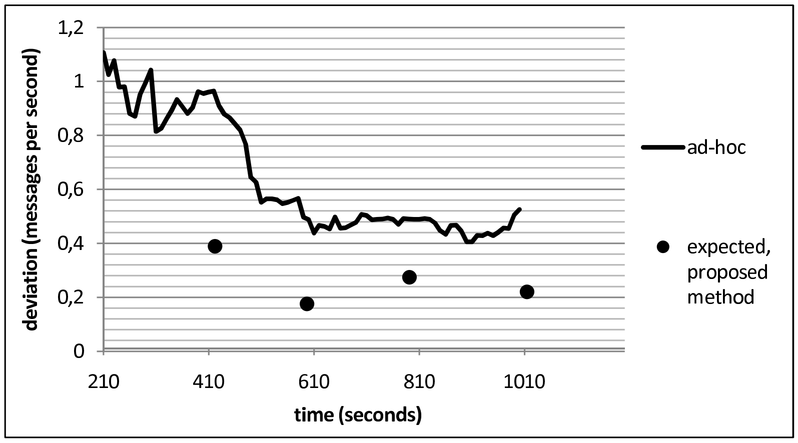

Figure 3.12: Deviation of measurements of the ad-hoc and the proposed method, in the third series of experiments

Time (seconds±5 seconds)	D_{ad-hoc}	$D_{proposed}$
200	1.1	0.34
420	0.96	0.39
600	0.49	0.17
790	0.49	0.27

Table 3.3: Measurement deviations of the ad-hoc and the proposed method, in the third series of experiments

To sum up the experimental results presented in this section, my simulations have proven that the performance measurements taken by the proposed method are correct. The experiments also show that the deviation of values measured by the proposed method is significantly lower than the deviation of the ad-hoc measurements taken in the same amount of time. This means that the proposed method is more accurate than the ad-hoc method.

3.7 Conclusions

In this chapter, I have proposed a model-driven performance testing method for calculating the number of messages that the SUT is capable of serving within a second while serving a given number of users in parallel. The motivation of creating the method is that this performance property is currently measured using ad-hoc methods in the industry, which are not efficient. The proposed method first creates a performance model that represents the SUT and then, based on the model, calculates the number of messages that the SUT is capable of serving within a second.

As the experimental results at the end of this chapter have shown, the deviation of the measurements produced by the proposed method is significantly lower than those of the ad-hoc method. Thus, the objective of this chapter has been reached.

Chapter 4

Worst-Case Performance Correction of Communicating Systems

In the previous chapter, I have shown how to measure CW_{usr}, which is the worst-case number of messages that the SUT is able to serve within a second. If the performance test finds that $CW_{usr} < CWR_{usr}$ that is, the worst-case number of messages that the SUT is able to serve within a second, is lower than the worst-case number of messages that the SUT should be able to serve within a second, according to its performance requirements, then the performance of the SUT should be augmented to a level at which, it fulfills this requirement.

Increasing the number of messages that the SUT is able to process within a second in worst case, is achieved by reducing its transition delays. Each transition delay is reducible by predefined amounts, which may vary from transition to transition. Furthermore, each transition delay reduction has a cost.

In this chapter, I propose performance correction methods, which aim at increasing CW_{usr} to the desired level by decreasing (some of the) transition delays of the SUT. Beyond increasing CW_{usr} to CWR_{usr} or above, the objective of the presented methods is to carry out this correction at minimal cost.

This chapter is organized as follows: In Section 4.1, I define the worst-case performance correction problem. In Section 4.2, I prove that the problem is NP-complete. In Section 4.3, I formulate the problem as a binary linear program, while in Section 4.4, I introduce a heuristic algorithm for solving the problem. I close the chapter with simulation results in Section 4.5 and with a conclusion in Section 4.6.

4.1 Definition of the Worst-Case Performance Correction Problem

The worst-case performance correction problem is formally defined as follows:

Given are the set $T = \{t_i\}$ of transitions, and the set $C = \{C_i\}$ of cycles, each cycle $C_i = \{t_j\}$ being a set of transitions, and each transition t_j having a delay value d_j. Given are a positive number CWR_{usr}, a positive number K, assigned to each transition t_i a variable (a so-called correction factor) $0 < q_i \leq 1$, a set $Q_i = \{q_{ij}\}$, where $q_{ik} < q_{i(k+1)}$ for each $k = 1, \ldots, |Q_i| - 1$, and $q_{i|Q_i|} = 1$. Furthermore for each transition, given is a monotonic decreasing function $Cost_i(x)$ for which $Cost_i : (0, 1] \rightarrow \mathbb{R}^+$, $Cost_i(1) = 0$. The question to be answered is as follows: Is it possible to choose the value of each q_i so that $\exists(q_{ij} \in Q_i) : q_i = q_{ij}$, and the following two inequalities are true?

$$\forall(c_i \in C) : \sum_{j:t_j \in c_i} d_j q_j \leq \frac{|c_i|}{CWR_{usr}} \tag{4.1}$$

$$\sum_{i:t_i \in T} Cost_i(q_i) \leq K \tag{4.2}$$

To further explain the above, q_i is a factor representing the reduction of d_i. The reduced delay of t_i is $d_i q_i$. $Cost_i(q_i)$ is the cost of the delay reduction of transition t_i. $\forall(i : t_i \in T) : Cost_i(1) = 0$, because if $q_i = 1$, the delay of t_i is not reduced, and its delay reduction does not cost anything. Finally, K is an upper bound for the cost of correcting the delays of all transitions.

Formula 4.1 expresses that after the delay correction, the system has to meet Formula 3.5, while Formula 4.2 expresses that the total cost of delay correction must not exceed K.

4.2 Complexity of the Worst-case Performance Correction Problem

In this section, I am going to prove the NP-completeness of the worst-case performance correction problem by reducing the NP-complete knapsack problem using the

Karp reduction [46].

First, I show that the problem is in NP by showing that it can be decided in polynomial time whether an arbitrary solution candidate is a solution (a witness) of the worst-case performance correction problem or not. After showing that the problem is in NP, I am going to prove its NP-completeness by defining a mapping that transforms an arbitrary instance of the NP-complete knapsack problem to an instance of the worst-case performance correction problem in polynomial time, and by proving that the so obtained instance of the worst-case performance correction problem is solvable exactly if the corresponding instance of the knapsack problem is solvable.

For the better readability of the proof, let me shortly introduce its main idea. When reducing an instance of the knapsack problem to the worst-case performance correction problem, a single transition cycle is constructed, in which, each transition corresponds to an element of the knapsack problem. Each transition delay can be reduced to one single value or left uncorrected. Including an element in the knapsack correspoonds to reducing the corresponding transition delay. The amount of delay correction achieved by each transition corresponds to the value of the corresponding element, while its cost corresponds to the weight of the corresponding element.

Proof: Before beginning the proof, let us redefine the worst-case performance correction problem using the attributes of the first definition, as follows:

Given are the set $T = \{t_i\}$ of transitions, and the set $C = \{C_i\}$ of cycles, where each cycle $C_i = \{t_j\}$ is a set of transitions, and each transition $t_j = \{(d_{jk}, c_{jk})\}$ is a set of delay-cost pairs, where delay $d_{jk} = d_j q_{jk}$, cost $c_{jk} = Cost_j(q_{jk})$, and $k : 1 \leq k \leq |Q_j|$ is an integer ($|t_j| = |Q_j|$). The question to be answered is as follows: Is it possible to choose exactly one delay-cost pair $(\hat{d}_j, \hat{c}_j) \in t_j$ from each transition t_j so that $\forall (i : C_i \in C) : \sum\limits_{j:t_j \in C_i} \hat{d}_j \leq \frac{|C_i|}{CWR_{usr}}$ and $\sum\limits_{j:t_j \in T} \hat{c}_j \leq K$, where K is an upper bound for the cost of correcting the delays of the transitions?

To further explain the above, for each transition t_i, $d_{i|t_i|}$ (or $d_{i|Q_i|}$) is the original delay of the transition, i.e. the measured delay d_i of the transition and $c_{i|t_i|} = c_{i|Q_i|} = Cost_i(q_{i|t_i|}) = Cost_i(q_{i|Q_i|}) = Cost_i(1) = 0$.

A set $\hat{T}$ of the chosen $(\hat{d}_j, \hat{c}_j)$ pairs is an appropriate witness, since given this set (containing $|T|$ elements), checking whether the elements of $\hat{T}$ give an appropriate solution can be done as follows: First, we sum up the $\hat{d}_j$ values and check whether the sum is lower than or equal to $\frac{|C_i|}{CWR_{usr}}$ for each $C_i \in C$. Then we sum up the $\hat{c}_j$

values and check whether their sum is lower than or equal to K. This operation can be carried out in $O(|C||T|)$ time, that is, in polynomial time. Thus, the worst-case performance correction problem is in NP.

Now we have to reduce the knapsack problem to an instance of the worst-case performance correction problem. The knapsack problem is defined as follows:

Given are a set G, for all of its elements g_j a positive integer $v(g_j)$ and a positive integer $w(g_j)$ and positive integers V and W. The question to be answered is as follows: Is there a subset $G' \subseteq G$ such that the following inequalities are true?

$$\sum_{g_j \in G'} w(g_j) \leq W \tag{4.3}$$

$$\sum_{g_j \in G'} v(g_j) \geq V \tag{4.4}$$

Let us now take this definition of the knapsack problem and reduce it to an instance of the worst-case performance correction problem. First of all, to each $g_j \in G$ of the knapsack problem, a transition t_j is assigned, such that $t_j = \{(d_{j1}, c_{j1}), (d_{j2}, c_{j2})\}$, and $\bigcup_j t_j = T$. The rest of the variables of the resulting worst-case performance correction problem are as follows, where $\forall(j : t_j \in T) : \delta_j > 0$ is arbitrary:

$$\begin{array}{ccc}
d_{j1} & := & \delta_j \\
c_{j1} & := & w(g_j) \\
d_{j2} & := & v(g_j) + \delta_j \\
c_{j2} & := & 0 \\
C & := & \{C_1\} \\
C_1 & := & T \\
CWR_{usr} & := & \frac{|G|}{\sum_{g_j \in G}(v(g_j)+\delta_j)-V} \\
K & := & W
\end{array}$$

According to the assignments above, in the resulting graph there will be exactly one cycle containing all the transitions. Furthermore, each transition t_j will have two delay-cost pairs. Choosing pair (d_{j1}, c_{j1}) in the worst-case performance correction problem corresponds to including g_j in G' in the knapsack problem, while choosing pair (d_{j2}, c_{j2}) corresponds to not including g_j in G'.

Now we have to show that the knapsack problem is solvable if and only if the corresponding worst-case performance correction problem is solvable.

Let us assume that the above defined worst-case performance correction problem is solvable. This means that for each $t_j \in T$ there is a $(\hat{d}_j, \hat{c}_j) \in t_j$ pair such that the following inequalities are true:

$$\sum_{j:t_j \in C_1} \hat{d}_j \leq \frac{|C_1|}{CWR_{usr}} \tag{4.5}$$

$$\sum_{j:t_j \in T} \hat{c}_j \leq K \tag{4.6}$$

Using the assignments defined earlier in this proof, Inequality 4.5 can be transformed as follows:

$$\begin{gathered} \sum_{g_j \in G} (v(g_j) + \delta_j) - \sum_{g_j \in G'} v(g_j) \leq \\ \leq \frac{|C_1|}{\frac{|G|}{\sum\limits_{g_j \in G} (v(g_j)+\delta_j) - V}} \end{gathered} \tag{4.7}$$

The reason for transforming the left side of Inequality 4.5 as seen in the equation above is the following:

According to the assignments defined earlier in the proof, $d_{j1} = \delta_j = v(g_j) + \delta_j - v(g_j)$. That is, each $\hat{d}_j$ value includes a $v(g_j) + \delta_j$ component either if it equals d_{j1} or d_{j2}. Thus, by summing up the $\hat{d}_j$ values on the left side of Inequality 4.5, a $\sum\limits_{g_j \in G} (v(g_j) + \delta_j)$ component will appear on the left side of Inequality 4.7. A $\hat{d}_j$ value has a further $-v(g_j)$ component exactly if it equals d_{j1} and a $\hat{d}_j$ value equals d_{j1} exactly if $g_j \in G'$. Thus $-v(g_j)$ has to be added to the left side of Inequality 4.7 for each $g_j \in G'$.

Since $|G| = |C_1|$, Inequality 4.7 can be reduced as follows:

$$\sum_{g_j \in G} (v(g_j) + \delta_j) - \sum_{g_j \in G'} v(g_j) \leq \sum_{g_j \in G} (v(g_j) + \delta_j) - V \tag{4.8}$$

$$V \leq \sum_{g_j \in G'} v(g_j) \tag{4.9}$$

According to the assignments defined earlier in this proof, Inequality 4.6 can be transformed as follows:

$$\sum_{g_j \in G'} w(g_j) \leq W \tag{4.10}$$

The explanation for the left side of Inequality 4.10 is the following:

$\hat{c_j} = c_{j2} = 0$ exactly if $g_j \notin G'$ and $\hat{c_j} = c_{j1} = w(g_j)$ exactly if $g_j \in G'$ thus, the left side of Inequality 4.10 will be the sum of the $w(g_j)$ values of those g_j elements which are included in G'.

As a consequence of the transformations of Inequalities 4.5 and 4.6, our worst-case performance correction problem will be solvable exactly if Inequalities 4.9 and 4.10 are true, but as Inequality 4.9 is identical to Inequality 4.4 and Inequality 4.10 is identical to Inequality 4.3, our worst-case performance correction problem will be solvable if and only if the corresponding knapsack problem is solvable.

Since the transformation of the knapsack problem to an instance of the worst-case performance correction problem can be carried out in $O(|G|)$ that is, in linear time and the knapsack problem is solvable if and only if the corresponding worst-case performance correction problem is solvable, the knapsack problem is Karp reducible to the worst-case performance correction problem.

And finally, since the NP-complete knapsack problem is Karp reducible to the worst-case performance correction problem and the worst-case performance correction problem is in NP, the worst-case performance correction problem is NP-complete.

■

4.3 ILP formulation of the Worst-Case Performance Correction Problem

Since the worst-case performance correction problem is NP-complete, an effective way to find its optimal solution is formulating it as an integer linear program, which will be a binary linear program (BLP) in our case, and solving this linear program. The optimal solution is the solution having the lowest cost. The worst-case performance correction problem can be formulated as a BLP as follows, where $cst_{ij} = Cost_i(q_{ij})$:

Minimize:

$$\sum_{i:t_i \in T} \sum_{j=1}^{|Q_i|} s_{ij}\, cst_{ij} \tag{4.11}$$

Subject to:

$$\forall (i : t_i \in T) : \sum_{j=1}^{|Q_i|} s_{ij} = 1 \tag{4.12}$$

$$\forall c_i \in C : \sum_{j:t_j \in c_i} d_j \sum_{k=1}^{|Q_i|} s_{jk} q_{jk} \leq \frac{|c_i|}{CWR_{usr}} \tag{4.13}$$

$$\begin{gathered}\forall (i : t_i \in T) : \forall (j = 1, 2, \ldots, |Q_i|) : \\ s_{ij} \in \{0, 1\}\end{gathered} \tag{4.14}$$

The unknown variables the values of which have to be found when solving the binary program are the s_{ij} variables. The value of each s_{ij} has to be set to 0 or 1 (Equation 4.14). Variables s_{ij}, where $j = 1, \ldots, |Q_i|$ are used for selecting the correction factor of transition t_i. As a solution of the BLP above, for each transition t_i, there is exactly one s_{ij} variable the value of which is 1. All the other s_{ij} variables belonging to t_i are set to 0 (consequence of Equations 4.12 and 4.14). If the value of s_{ij} is 1 then $q_i = q_{ij}$ and thus, correcting the delay of t_i costs $Cost_i(q_{ij})$.

As mentioned above, Equations 4.12 and 4.14 are responsible for choosing the correction factors legally that is, each s_{ij} is 0 or 1 and for each t_i, exactly one s_{ij} equals 1, while the others equal 0. On the left side of Inequality 4.13, $\sum_{k=1}^{|Q_i|} s_{jk} q_{jk}$ equals correction factor q_i of transition t_i. Thus, Inequality 4.13 means that the corrected delay of each cycle c_i has to be lower than or equal to $\frac{|c_i|}{CWR_{usr}}$ (this corresponds to Inequality 4.1). Finally, in the objective function (Formula 4.11), $\sum_{j=1}^{|Q_i|} s_{ij} cst_{ij}$ equals $Cost_i(q_i)$ (the cost of correcting the delay of transition t_i). The value of $Cost_i(q_i)$ is chosen from set $\{cst_{ij} | j = 1, \ldots, |Q_i|\}$ by the appropriate s_{ij} variable set to 1. Thus, the objective function expresses that the total cost of correcting the transition delays should be minimal.

4.4 A Heuristic Solution for the Worst-Case Performance Correction Problem

In this section, I am going to introduce a heuristic algorithm for solving the worst-case performance correction problem. The algorithm is optimized for the case when $Cost_i(x) = -\gamma_i \log_a x$, where $\gamma_i > 0$ is a constant assigned to transition t_i. Algorithm 3 shows how my heuristic method works. In the algorithm, r is the so-called refreshing granularity.

Algorithm 3: Heuristics for solving the worst-case performance correction problem

```
input  : $T$, $C$, $CWR_{usr}$, $r$, $\{Q_i | i : t_i \in T\}$, $\{\gamma_i | i : t_i \in T\}$
output: $\bigcup_{i:t_i \in T} \{q_i\}$
1  foreach $i : t_i \in T$ do
2      $clist_i := \{j | t_i \in c_j\}$
3  foreach $i : t_i \in T$ do
4      $q_i := q_{i1}$
5  if $\exists(c_i \in C) : \sum_{j:t_j \in c_i} d_j q_j > \frac{|c_i|}{CWR_{usr}}$ then
6      return "unsolvable";
7  foreach $i : t_i \in T$ do
8      $current_i := 1$
9  foreach $i : t_i \in T$ do
10     if $|Q_i| > 1$ then
11         $\alpha_i := \left\lceil \frac{|clist_i| d_i r (q_{i(current_i+1)} - q_{i(current_i)})}{\gamma_i} \right\rceil$
12 while $\exists i : (current_i < |Q_i|$
   $\wedge(\forall j \in clist_i : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + q_{i(current_i+1)} d_i \leq \frac{|c_j|}{CWR_{usr}}))$ do
13     foreach $i : t_i \in T$ do
14         if $current_i < |Q_i|$ then
15             $\alpha_i := \alpha_i - 1$
16     foreach $i : t_i \in T$ do
17         if $\alpha_i = 0 \wedge current_i < |Q_i|$
           $\wedge \forall j \in clist_i : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + q_{i(current_i+1)} d_i \leq \frac{|c_j|}{CWR_{usr}})$ then
18             $q_i := q_{i(current_i+1)}$;
19             $current_i := current_i + 1$
20     if $current_i < |Q_i|$ then
21         $\alpha_i := \left\lceil \frac{|clist_i| d_i r (q_{i(current_i+1)} - q_{i(current_i)})}{\gamma_i} \right\rceil$
22 return $\bigcup_{i:t_i \in T} q_i$;
```

The algorithm first sets the value of each correction factor q_i to its minimal value q_{i1} and then it runs iterations and increases each q_i to its next smallest legal value (i.e. from q_{ik} to $q_{i(k+1)}$) more or less frequently. Before the first iteration and upon each increasement of q_i, the algorithm sets the value of α_i, which is the number of iterations that have to pass until the next increasement of q_i. The key step of the algorithm is this latter one that is, determining how many iterations have to pass until the next increasement of each q_i in order to keep the cost of delay reduction minimal.

In the following, I am going to explain the algorithm in detail.

In lines 1 and 2, to each transition t_i a set $clist_i$ is constructed storing references to each of the cycles including t_i.

The algorithm sets each correction factor q_i to its minimal (and most expensive) value q_{i1} in lines 3 and 4, while in line 5, the algorithm checks whether using these lowest possible values of correction factors, Inequality 4.1 is fulfilled for each cycle c_i or not. If not, the problem is unsolvable.

In lines 7 and 8, the algorithm initializes variable $current_i$ for each q_i, which will store the index of that element of Q_i, the value of which is equal to the current value of q_i (i.e. $q_{icurrent_i} = q_i$).

Line 11 (along with line 21) is the key step of the algorithm. In this step, for each transition t_i, the value of α_i is determined. α_i is the number of iterations that have to pass until the next increasement of q_i. To explain the formula α_i is assigned, let us look at the following few examples.

Let us take a TCFMM consisting of two states s_0, and s_1, and two transitions t_1, and t_2, such that t_1 is originated in s_0 and destinated in s_1 while t_2 is originated in s_1 and destinated in s_0. Thus, in this TCFMM there is exactly one directed cycle $c_i = \{t_1, t_2\}$. Let b denote the maximal allowed cycle delay we would like to achieve by correcting delays d_1, and d_2. In this case, $b = \frac{|c_1|}{CWR_{usr}} = \frac{2}{CWR_{usr}}$. Let us furthermore assume that in this example, the co-domains of q_1 and q_2 are continuous and the cost functions are $Cost_1(x) = -\gamma_1 \log_a x$ and $Cost_2(x) = -\gamma_2 \log_a x$, where $\gamma_1 = \gamma_2 = 1$. That is, the two cost functions are the same and they are from among the cost functions for which the heuristic algorithm is optimized.

Because of the continuous co-domains and the single directed cycle in the TCFMM, for the most cost-effective solution, the corrected cycle delay will be equal to b and not lower than it. Thus, $d_1q_1 + d_2q_2 = b$. The latter equation means that $q_2 = \frac{b-d_1q_1}{d_2}$.

The objective of the problem is to minimize the total cost of delay reduction by choosing the appropriate q_i values, formally $\min\limits_{q_1,q_2,\ldots,q_{|T|}} \sum\limits_{i:t_i\in T} Cost_i(q_i)$, which in our case equals $\min\limits_{q_1,q_2}(-(\log_a q_1 + \log_a q_2))$. The latter formula, using that $q_2 = \frac{b-d_1q_1}{d_2}$, can be further transformed as follows:

$$\min_{q_1,q_2}(-(\log_a q_1 + \log_a q_2)) \rightarrow \min_{q_1,q_2}(-\log_a q_1q_2) \rightarrow$$
$$\min_{q_1}\left(-\log_a\left(-\tfrac{d_1}{d_2}q_1^2 + \tfrac{q_1b}{d_2}\right)\right) \rightarrow \max_{q_1}\left(-\tfrac{d_1}{d_2}q_1^2 + \tfrac{q_1b}{d_2}\right) \rightarrow$$
$$\min_{q_1}\left(\tfrac{d_1}{d_2}q_1^2 - \tfrac{q_1b}{d_2}\right) \rightarrow \min_{q_1}\left(\sqrt{\tfrac{d_1}{d_2}}q_1 - \tfrac{b}{2d_2}\sqrt{\tfrac{d2}{d1}}\right)^2 - \tfrac{b^2}{4d_2^2}\tfrac{d_2}{d_1}$$

Since the second component of the above formula does not contain q_1, it is minimal if the first component $\left(\sqrt{\frac{d_1}{d_2}}q_1 - \frac{b}{2d_2}\sqrt{\frac{d2}{d1}}\right)^2$ is minimal. Since the first component is a square, it is minimal if it equals 0 that is, if $q_1 = \frac{b}{2d_2}\sqrt{\frac{d_2}{d_1}}\sqrt{\frac{d_2}{d_1}} = \frac{b}{2d_2}\frac{d_2}{d_1} = \frac{b}{2d_1}$. In this case, $q_2 = \frac{b-d_1q_1}{d_2} = \frac{b}{2d_2}$.

Thus, the cost of correcting the two transition delays will be minimal if $d_2q_2 = \frac{b}{2}$, and $d_1q_1 = \frac{b}{2}$ that is, if the corrected delay values d_1q_1 and d_2q_2 are equal. Based on this, we can suspect that in the case of a directed cycle which consists of more than two transitions, the cost of correcting the cycle delay will be minimal if each corrected delay d_iq_i is equal. If however, the corrected transition delays of the cycle are equal, they equal $\frac{1}{CWR_{usr}}$. This is the consequence of Inequality 4.1, which takes the following form for the optimal q_i values of this continuous problem:

$$\sum_{i:t_i\in c_1} d_iq_i = \frac{|c_1|}{CWR_{usr}} \tag{4.15}$$

As the consequence of the above, in the continuous case with a single directed cycle, correction factor q_i equals $\frac{1}{d_1\, CWR_{usr}}$.

Let us now constrain the above described continuous case to a non-continuous case of the problem in which, a positive integer Gr is given, and the legal values of each correction factor q_i are values $q_{i1} = \frac{1}{Gr}, q_{i2} = \frac{2}{Gr}, \ldots, q_{iGr} = \frac{Gr}{Gr} = 1$ (thus, set Q_i is the same for each transition t_i). Let us furthermore assume, that the TCFMM we are dealing with consists of a single directed cycle. In this case, the appropriate correction factor for transition t_i can be achieved if the initial value q_{i1} of q_i is incremented in every $\lceil d_ir\rceil$-th iteration of the algorithm (see the explanation of coefficient r later in this section). For example, if d_2 is twice as large as d_1 and thus, q_1 has to be around $2x$ and q_2 has to be around x then q_1 has to be incremented twice as frequently as q_2. Let $period_i$ denote the number of iterations that have to pass between two subsequent increasements of q_i. To fulfill the above requirement, one coefficient of $period_i$ is d_i, which is responsible for making the corrected delay values approximately equal to each other. In this case, $period_i = \lceil d_i r \rceil$.

Still not relaxing the non-continuous problem of the previous paragraph to the problem allowing for arbitrary correction factors, let us assume that the TCFMM has multiple cycles and some of its transitions are included in more than one of these cycles (this is the general case). In this case, it is more cost-effective to reduce the delays of transitions included in many cycles by a bigger amount than the delays of transitions included in fewer cycles. The reason for this is as follows: By reducing the delay of a transition included in n cycles, n cycle delays will be reduced and thus, the left side of n instances of Inequality 4.1 will be reduced, while the cost of this reduction is not multiplied by n. Based on the above, we can suspect that if we further weigh $period_i$ by $|clist_i|$ (the number of cycles including t_i), then the total cost of delay reduction will be closer to optimal. I have confirmed this suspicion by running simulations. Thus, in this case, $period_i = \lceil |clist_i| d_i r \rceil$.

Let us now relax the above described problem to one where the Gr values of different transitions can be different. Let Gr_i denote the Gr value assigned to t_i. If in this case, the value of each correction factor q_i is incremented in every $\lceil |clist_i| d_i r \rceil$-th iteration, the final values of the correction factors will be incorrect, since the algorithm does not take into consideration the differences between the Gr_i values. Let us assume that the TCFMM has two states and two transitions t_1 and t_2 constituting a single cycle that is, $|clist_1| = |clist_2| = 1$. If $Gr_1 = 4$, and $Gr_2 = 2$, then $q_{11} = 0.25$, $q_{12} = 0.5$, $q_{13} = 0.75$, $q_{14} = 1$, $q_{21} = 0.5$, and $q_{22} = 1$. Since the TCFMM includes a single cycle, the corrected delay values of the two transitions should be about the same at the end of the algorithm. To achieve this, q_1 should be incremented twice as frequently as q_2. This way, $period_i$ has to be further weighed by $\frac{1}{Gr_i}$, which is the difference between any two subsequent legal values of q_i. Thus, in this case, $period_i = \lceil |clist_i| d_i r \frac{1}{Gr_i} \rceil$.

Let us now assume that the legal values of each correction factor q_i are arbitrary. In this case, a small modification of the above solution is needed. In the case of the original problem, two subsequent increasements of a correction factor q_i might increase q_i by different amounts. Thus, rather than predefining $period_i$, for each q_i, in the beginning of the algorithm and upon each increasement of q_i, the number of iterations to pass before the next increasement has to be redefined and set to

$$\alpha_i = \lceil |clist_i| d_i r(q_{i(current_i+1)} - q_{icurrent_i}) \rceil.$$

Thus, instead of $\frac{1}{Gr_i}$, which was the difference between any two subsequent legal values of q_i in the above paragraph, $\lceil |clist_i| d_i r \rceil$ has to be weighed by the difference between the current and next legal value of q_i.

To explain why $\frac{1}{\gamma_i}$ is a coefficient of α_i, let us take the problem described in the above paragraph and assume that there are different cost functions assigned to each transition and the cost function of transition t_i is $Cost_i(x) = -\gamma_i \log_a x$. Let us look at an example in which, among the transitions of the TCFMM, there are transitions $t_1, t_2, \ldots, t_\gamma$ such that $\forall(i : 1 \leq i \leq \gamma - 1) : s_{to_i} = s_{from_{i+1}} \wedge \forall(i : 1 \leq i \leq \gamma) : \nexists(j : t_j \in T) : j \neq i \wedge (s_{to_j} = s_{to_i} \vee s_{from_j} = s_{from_i})$. That is, these transitions constitute a chain the states of which have exactly one incoming transition (except for the originating state of the first transition of the chain) and one outgoing transition (except for the destination state of the last transition of the chain). Therefore, $clist_1 = clist_2 = \ldots = clist_\gamma$. Let us furthermore assume that $Q_1 = Q_2 = \ldots = Q_\gamma$, $d_1 = d_2 = \ldots = d_\gamma = \frac{d}{\gamma}$, and $Cost_1(x) = Cost_2(x) = \ldots = Cost_\gamma(x) = -\log_a x$. Since the sets of correction factors are identical for each transition, in each iteration of the algorithm, $current_1 = current_2 = \ldots = current_\gamma$. As a consequence, at the end of the iteration in which the correction factors of these transitions were increased,

$$\alpha_1 = \alpha_2 = \ldots = \alpha_\gamma = \lceil |clist_1| d_1 r(q_{1(current_1+1)} - q_{1(current_1)}) \rceil = \\ \left\lceil \frac{|clist_1| dr(q_{1(current_1+1)} - q_{1(current_1)})}{\gamma} \right\rceil = \alpha.$$

This means that between the next and last increasement of its correction factor, each transition has to wait α iterations. Let us now consider the chain of transitions $t_1, \ldots, t_\gamma$ as a single (virtual) transition t. The delay of t will be $\sum_{i=1}^{\gamma} d_i = d$. Each time the correction factors of transitions $t_1, \ldots, t_\gamma$ are increased, they are increased in the same iteration. Thus, if Q is the set of correction factors of t, then $Q = Q_1 = \ldots = Q_\gamma$, and in each iteration of the algorithm, t has to wait the same number of iterations until its next correction factor increasement as transitions $t_1, \ldots, t_\gamma$. Consequently, at the end of the iteration in which the correction factors of the transitions in the chain (and transition t) were increased, the number of iterations that had to pass until the next increasement of the correction factor of t was

$$\alpha = \left\lceil \frac{|clist_1|dr(q_{1(current_1+1)} - q_{1(current_1)})}{\gamma} \right\rceil .$$

If the correction factor of t is x however, its cost is $-\gamma \log_a(x)$, which is γ times as much as $Cost_1(x) = Cost_2(x) = \ldots = Cost_\gamma(x) = -\log_a x$. The reason for this is that based on the above, the correction factor of t will equal x if the correction factors of all γ transitions in the chain equal x. In the case of γ being an integer, each of the above steps of transforming a set of transitions to a single transition can be carried out in the other direction (i.e. transfoming a transition t to a chain of sub-transitions $t_1, \ldots, t_\gamma$) and thus, if the cost function of t is $Cost_i(x) = -\gamma_i \log_a x$, its α value equals

$$\alpha_i = \left\lceil \frac{|clist_1|dr(q_{1(current_1+1)} - q_{1(current_1)})}{\gamma} \right\rceil .$$

I suspected that including $\frac{1}{\gamma_i}$ as a coefficient in α_i gives a total cost of correction closer to optimal even if γ_i is not an integer.

The reason for including refreshing granularity r in α_i is that in order to make α_i an integer value, the ceiling value of $\frac{|clist_i|d_i r(q_{i(current_i+1)} - q_{icurrent_i})}{\gamma_i}$ is taken (I have chosen the ceiling value instead of the floor value to avoid the illegal case when $\alpha_i = 0$). If $\left\lceil \frac{|clist_i|d_i(q_{i(current_i+1)} - q_{icurrent_i})}{\gamma_i} \right\rceil$ (not including r) is a small integer, then by taking its ceiling value, some of the accuracy of $\frac{|clist_i|d_i(q_{i(current_i+1)} - q_{icurrent_i})}{\gamma_i}$ is lost. If however, $\frac{|clist_i|d_i(q_{i(current_i+1)} - q_{icurrent_i})}{\gamma_i}$ is multiplied by r which is a large integer and then the ceiling value of this enlarged product is taken, some of this otherwise lost accuracy can be preserved. The value to be chosen for r depends on $|clist_i|d_i$. The smaller $|clist_i|d_i$ is, the greater r has to be to preserve the same amount of accuracy. During our simulations presented in Section 4.5, we required r to be larger than or equal to $\lceil \frac{100}{\min\limits_{i:t_i \in T} |clist_i|d_i} \rceil$.

After setting the value of α_i for the first time, from line 8, the algorithm runs iterations. While there exists a correction factor q_i which does not equal 1 and which can be augmented to its next legal value without violating Inequality 4.1, the algorithm starts a new iteration. In each iteration, the algorithm decrements the α_i value of each $q_i < 1$. If for a correction factor $q_i < 1$, $\alpha_i = 0$, and q_i can be increased to its next legal value without violating Inequality 4.1, then q_i is increased to its next legal value. The iterations go on until no further correction factor increasement is possible.

4.5 Simulation Results

In this subsection, I present simulation results comparing the performance of my heuristic algorithm to that of the BLP formulated in Subsection 4.2.

The simulations were run on a total of 6100 TCFMMs having 20 states and 20 to 80 transitions. For each number of transitions, I have generated 100 semi-random TCFMMs. This means that the d_i and γ_i values and Q_i sets assigned to transitions t_i of each TCFMM were generated randomly. The structure of the TCFMMs having the same number of transitions was the same. The structures of TCFMMs having the same number of transitions were built incrementally. The structure belonging to the first group of TCFMMs had 20 transitions. Each structure having $n : 20 < n \leq 80$ transitions was constructed by taking the structure having $n - 1$ transitions and by adding a random transition to it. Transition delays were generated with uniform distribution on interval $(0, 2.0]$ that is, the mean transition delay was 1.0. The number of legal correction factors ($|Q_i|$) of each transition t_i was a random integer generated on interval $[1, 20]$, with uniform distribution. The elements of each Q_i were generated on interval $(0, 1)$, with uniform distribution, except for $q_{i|Q_i|}$ which was 1 for each t_i. The cost function of delay reduction used for the simulations was $Cost_i(x) = -\gamma_i \ln(x)$ for each transition t_i. The γ_i values were generated on interval $[1, 5]$, with uniform distribution.

The simulations investigante the time and cost-efficiency of my heuristic algorithm (denoted by H in the figures) and those of the BLP, which always finds the optimal solution that is, the solution with the lowest possible cost. In general, the time needed to solve a binary linear program was unreasonable. Due to this, I was unable to run the binary program for denser TCFMMs. Thus, I have also solved a series of linear programs (denoted by LP in the figures), which could be solved in polynomial time. In the linear programs, instead of the constraint defined in Formula 4.14, I defined the following constraint:

$$\begin{gathered} \forall(i : t_i \in T) : \forall(j = 1, 2, \ldots, |Q_i|) : \\ 0 \leq s_{ij} \leq 1 \end{gathered} \tag{4.16}$$

Thus, contrarily to the binary linear program, in the linear program, the s_{ij} parameters are not constrained to be integer values. The optimal solution of the LP is not guaranteed to be a legal solution of the original problem. The cost of an LP problem is however, always lower than or equal to the cost of the corresponding

BLP problem and thus, it is a lower estimate for the cost of the solution of the corresponding BLP. To this lower estimate, we can compare the cost obtained by the heuristic algorithm.

I have also investigated the cost and time-efficiency of a simple round-robin algorithm (denoted by RR in the figures). The round-robin algorithm initially sets the value of each q_i to q_{i1}, then it increments the value of the correction factors to their next legal values in a round-robin manner until no further correction factor increasement is possible without violating any instances of Inequality 4.1. That is, the round-robin algorithm is a special case of my heuristic solution in which, in steps 11 and 21, α_i is set to constant 1.

During the simulations, I measured the average time and cost needed for delay correction using each method, as a function of the number of transitions in the TCFMM. For each number of transitions (for each group of TCFMMs), I have run 100 simulations (one for each semi-random TCFMM in the group), using each method. The value of each marker in the figures was calculated by averaging all 100 cost or time values obtained by the corresponding 100 simulations. The worst-case performance correction problem was not solvable for each of the 6100 TCFMMs. Thus, I have only taken into consideration those groups of TCFMMs in which out of the 100 TCFMMs, the problem was solvable for at least 10 TCFMMs.

Since the set of cycles is an input parameter for each method, all cycles in the TCFMMs had to be found before executing any of the methods. For finding the cycles, I used an iterative deepening depth-first search. Thus, each time value plotted in the figures of this section is the average amount of time needed to run the methods plus the amount of time needed to find the cycles of the corresponding group of TCFMMs. The reason why the TCFMMs having the same number of transitions have the same structure is the following: This way, only one cycle search had to be executed for each group of TCFMMs, while all directed cycles of 6100 fully random TCFMMs could not have been found and the simulation results could not have been produced within a reasonable amount of time.

In the headers of the tables in this section, C means cost.

Figure 4.1 and Table 4.1 shows the average costs of correcting the transition delays using the BLP and LP methods, the round-robin method, and my heuristic algorithm, where $CWR_{usr} = 1.0$ that is, CWR_{usr} equals the reciprocal of the mean transition delay. As it can be seen in the figure, and the table, using my heuristic method the cost of delay reduction is by 61% higher than the cost obtained by solving the LP, in average. However, the available costs of the BLP method are halfway between the costs of my heuristics and the costs of the LP and thus, the cost of the LP is a pretty rough lower estimate of the cost of the BLP. Based on the available BLP costs, the cost of my heuristics is by 36% higher than the cost of the BLP, in average. The cost of my method is constantly below the cost of the round-robin method, and it is by 11% lower than it, in average.

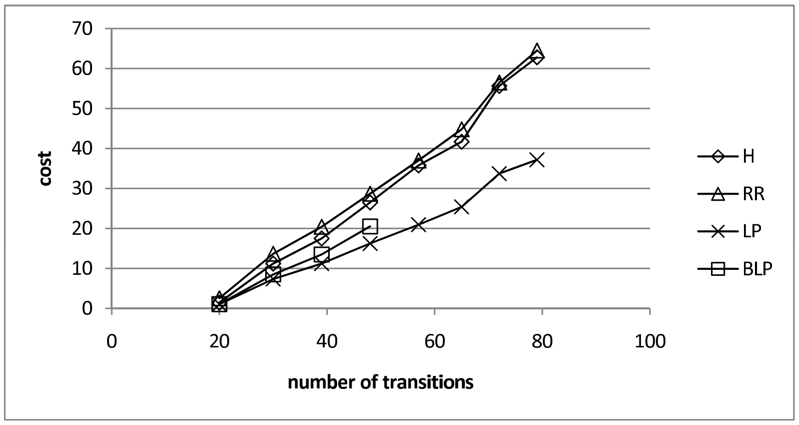

Figure 4.1: Costs of solutions, $CWR_{usr} = 1.0$

$\|T\|$	$C(H)$	$C(RR)$	$C(LP)$	$C(BLP)$
20	1.45	2.53	1.03	1.06
30	11.21	13.67	7.35	8.51
39	17.50	20.40	11.22	13.48
48	26.50	28.67	16.24	20.51
57	35.76	36.98	20.94	
65	41.68	44.78	25.40	
72	55.63	56.47	33.70	
79	62.79	64.48	37.15	

Table 4.1: Costs of solutions, $CWR_{usr} = 1.0$

Figure 4.2 and Table 4.2 show the costs of each method, where $CWR_{usr} = 1.5$. According to the figure and the table, the cost of my heuristic method is closer to the cost of the LP and the BLP than in the previous case. The cost of my method is by 38% higher than the cost of the LP. Based on the available BLP costs, the cost of my method is by only 24% higher than the cost of the BLP, in average. The cost of my heuristics is by 12% lower than the cost of the round-robin method, in average.

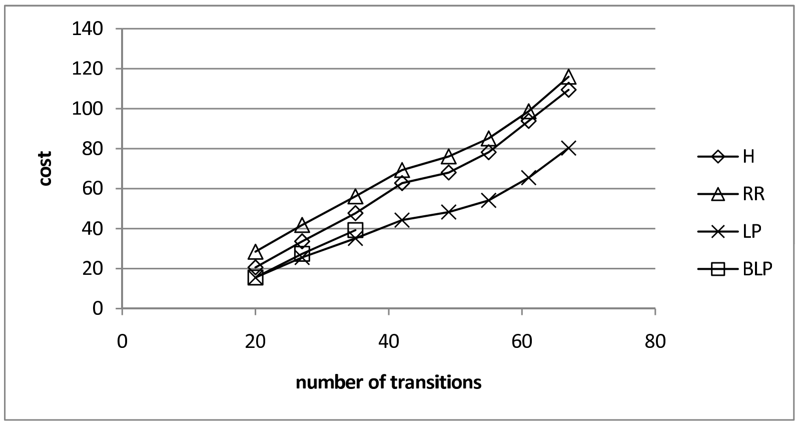

Figure 4.2: Costs of solutions, $CWR_{usr} = 1.5$

$\|T\|$	$C(H)$	$C(RR)$	$C(LP)$	$C(BLP)$
20	20.60	28.44	15.49	15.57
27	33.61	41.81	25.55	27.38
35	47.71	56.12	35.13	39.26
42	62.74	69.29	44.27	
49	68.03	76.10	48.28	
55	78.17	85.11	54.11	
61	93.83	98.64	65.47	
67	109.40	115.95	80.29	

Table 4.2: Costs of solutions, $CWR_{usr} = 1.5$

Figure 4.3 and Table 4.3 show the costs of each method, where $CWR_{usr} = 1.75$. As it can be seen in the figure and the table, the cost of my heuristic algorithm is closer to the cost of the LP and BLP than in the previous two cases. The cost of my heuristic algorithm is by 32% higher than the cost of the LP, and by 20% higher than the cost of the BLP (based on the available BLP costs). The cost of my method is by 13% lower than the cost of the round-robin algorithm, in average.

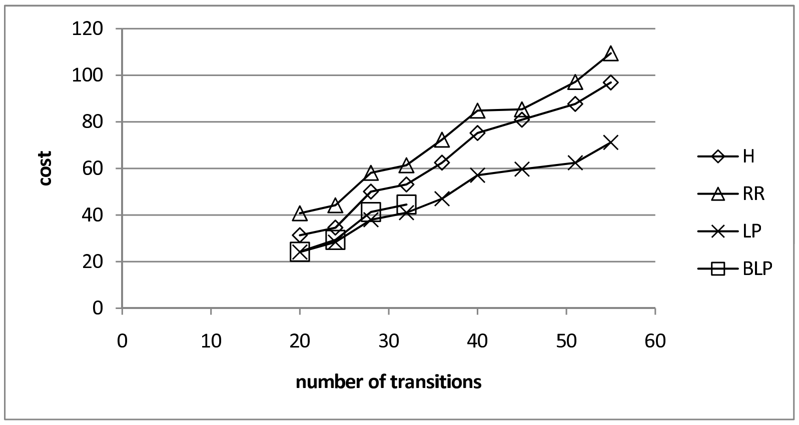

Figure 4.3: Costs of solutions, $CWR_{usr} = 1.75$

$\|T\|$	$C(H)$	$C(RR)$	$C(LP)$	$C(BLP)$
20	31.32	40.77	24.12	24.24
24	34.57	44.20	28.39	29.35
28	50.06	58.13	37.98	41.30
32	53.10	61.39	41.00	44.52
36	62.49	72.35	47.00	
40	75.19	84.82	57.08	
45	80.90	85.38	59.68	
51	87.68	97.13	62.43	
55	96.88	109.42	71.16	

Table 4.3: Costs of solution, $CWR_{usr} = 1.75$

After examining the cost-efficiency of the methods, let us take a look at their time-efficiency. Figures 4.4, 4.5, 4.6, and 4.7 show the amounts of time needed by the BLP, my heuristic algorithm, the round-robin algorithm and the LP, respectively, as a function of the number of transitions in the TCFMM, in different simulation scenarios. As it can be seen in the figures, the running time of the BLP is so high, it cannot be efficiently used for denser TCFMMs, while the running time of the rest of the methods is reasonable and is dominated by the amount of time needed to find all the transition cycles.

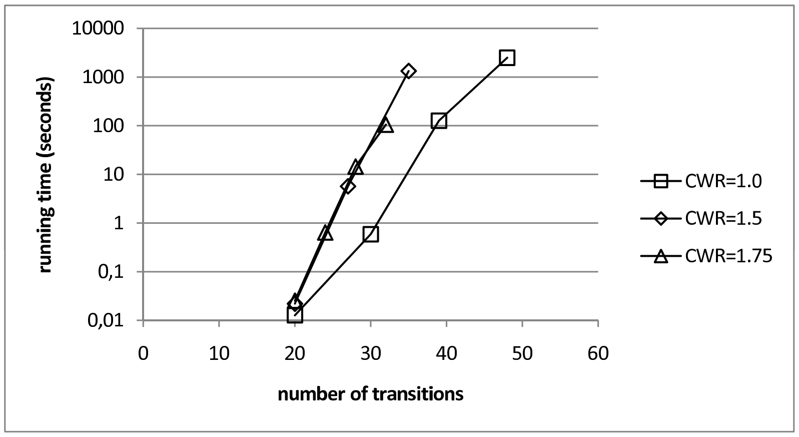

Figure 4.4: Running times of the BLP

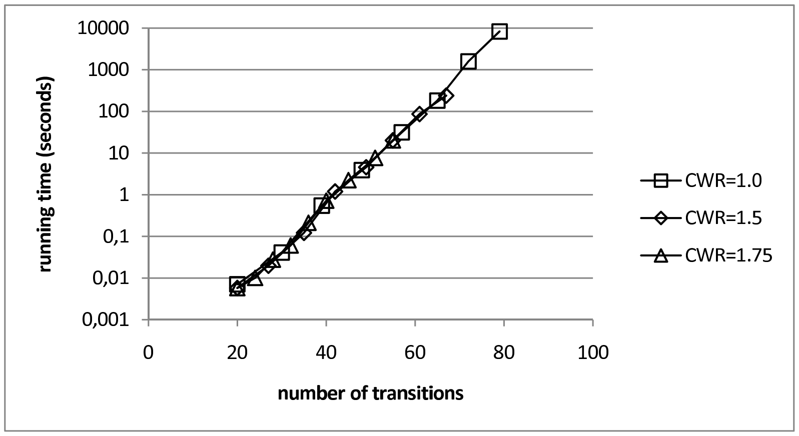

Figure 4.5: Running times of the heuristics

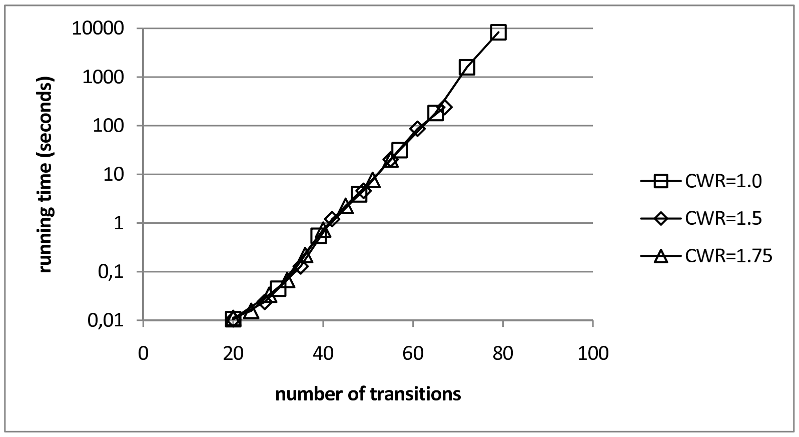

Figure 4.6: Running times of the LP

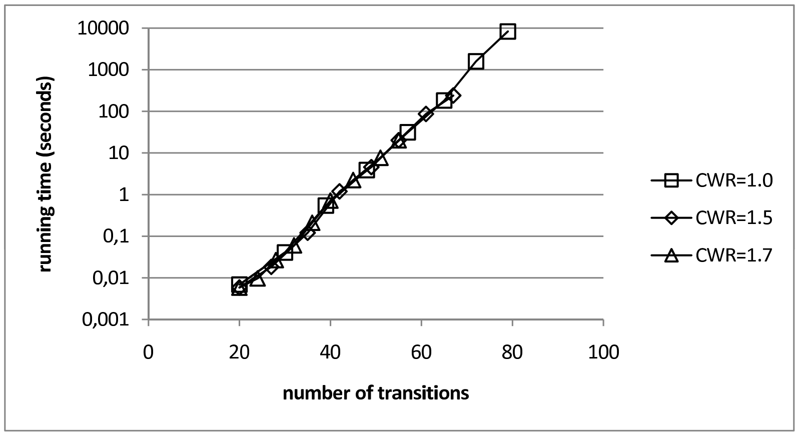

Figure 4.7: Running times of the round-robin algorithm

According to the simulation results, the higher CWR_{usr} is, the more effective my heuristic algorithm gets. The cost obtained by my heuristic algorithm is by $11-13\%$ lower than the cost of the round-robin method. By increasing CWR_{usr}, the delay reduction cost of my algorithm gets closer to the cost of the LP and to that of the BLP. The running time of my heuristics is by orders of magnitude lower than the time needed to solve the BLP, in all three scenarios.

4.6 Conclusions

In this chapter, I have investigated how to improve the performance of the SUT if it has failed the performance test more precisely, if it was found to be incapable of serving the required number of messages within a second while serving a given number of users in parallel. The chapter builds on the performance model presented in Chapter 3. The goal of the methods presented in this section is to increase the number of messages that the SUT is capable of serving within a second to the level required by the performance specifications, and to do it with minimal cost.

In this chapter, I have proven the NP-completeness of the performance correction problem, and formulated it as an integer linear program. Since the time needed to solve this integer linear program is unreasonable in most cases, I have also proposed a heuristic solution. I have evaluated the efficiency of the proposed methods and found that the heuristic solution performs efficiently.

Chapter 5

Summary and Future Work

In this chapter, I briefly review the results achieved, and discuss directions for the further extension of the work.

5.1 Load Distribution in a Performance Testing Environment

In Chapter 2, I have proposed load distribution methods for performance test environments. The methods presented in the chapter aim at solving the problem raised by the fact that while the system under test (SUT) is a system optimized for a specific purpose, the hosts of the test environment, which have to stress the SUT by its maximal specified load, are used for multiple performance tests for budget reasons and thus, they are universal pieces of hardware. In order to be able to stress the SUT with its maximal specified load, multiple THs are used in the test environment usually, and the aggregated output load of all the THs in the test environment is used for stressing the SUT.

The load that stresses the SUT during a performance test is generated by load generator entities (or virtual hosts, VHs), which are executed on THs. The objective of the presented methods is to achieve a maximal utilization of THs by making appropriate VH to TH assignments and to do this without the aggregated capacity of VHs executed on a TH exceeding the total capacity of the TH at any time.

After proving the NP-completeness of the load distribution problem, I formulated it as a binary linear program. Since the solution of the binary linear program can take a huge amount of time, I have proposed an algorithm, which divides up the time axis for time windows and solves the problem as a series of binary linear programs giving sub-optimal solutions for each time window.

Since this latter solution can still require a large amount of time to be solved in some scenarios, I have also proposed a heuristic algorithm for solving the problem. The heuristic algorithm also splits up the time axis for time windows, and considers the sub-problem in each time window separately. Within a single time window, the VH assignments are carried out using a heuristic algorithm similar to the first fit descending algorithm, which is developed for solving the bin packing problem.

I have examined the time-efficiency of and the average utilization achieved by the BLP based method, the heuristic algorithm, and a greedy algorithm. The simulations have shown that the longer the time window gets, the higher average utilization the BLP based heuristic solution achieves, while in the case of the bin packing based heuristic algorithm, the average utilization vs time window size curve is similar to one having a single maximum.

The research objective of creating algorithms which are more efficient than the greedy one has been reached, since the proposed heuristics are more effective than the greedy algorithm in most cases, according to the simulations. In the cases where the greedy algorithm might be the best choice, the average utilization achieved by the bin packing based algorithm is similar to that of the greedy algorithm, but the heuristic algorithm has to be run multiple times in order to find the optimal window size.

The proposed methods can be applied in any case where the test environment has to run multiple load generators againt the SUT. Among many others, one concrete application area is when the emulated users communicating with the SUT are the VHs. As a future extension of my results, further heuristic solutions can be investigated, like simulated annealing or genetic algorithms [63, 64].

5.2 A Model-Driven Performance Testing Method for Communicating Systems

In Chapter 3, I have introduced a performance testing method that automatically checks whether the SUT is able to serve the required number of requests within a second (messages per second), while serving the required number of users in parallel. The required number of messages per second can be the worst-case or the expected number of messages per second.

The presented method uses a formal performance model called the Timed Communicating Finite Multistate Machine (TCFMM) for representing the performance of the SUT and for conducting the performance test.

Before beginning the performance test, the method creates the functional structure of the TCFMM, based on the Finite State Machine (FSM) model to which, the SUT corresponds, according to the conformance test that the SUT has previously passed. The so created functional structure of the TCFMM is then used to conduct the performance test. During the performance test, the transition delays of the SUT are measured in order to create a complete TCFMM model, which models both the performance and conformance characteristics of the SUT. Based on this TCFMM model, the presented method calculates the worst-case and the expected number of messages per second that the SUT is able to process while serving the maximal specified number of users simultaneously.

At the end of the chapter, I compared the accuracy and time-efficiency of the presented performance testing method to those of the ad-hoc performance testing method, which is widely used in the industry nowadays. I have created three test scenarios by manipulating the distribution of the transition delays of the SUT. The experiments have shown that the deviation of performance measurements taken by our method is significantly lower than the deviation of measurements taken by the ad-hoc method, given the same amount of testing time, even in extreme cases. This means that the proposed performance testing method is able to produce performance measurements of a given deviation in a lower amount of time than the ad-hoc method.

The research objective in this section was to create a model-driven performance testing method which is more efficient than the ad-hoc methods used in the industry. This objective has been reached according to the experimental results, since given the same amount of time for testing, the deviation of the measurement results of the testing method proposed in the chapter is by orders of magnitude lower than those of the ad-hoc method. The method presented in this chapter can be used to test any kind of a system implementing a communicating protocol if the performance property to be measured during the test is the number of requests to be processed within a second while simultaneously serving a given number of users. Two examples for the application of this method are a web portal and a SIP proxy. [65] In the first case, the requests are the clicks of users, while in the second case, the requests are the request messages that have to be sent to the proxy, e.g. to initiate a call.

As a future extension of the method proposed in this chapter, the paths on which transition delays are measured could be optimized. The goal of this optimization would be to traverse each transition of the SUT at least I times, where I is the number of iterations given as input of the algorithm, and to minimize the number of traverses over the I^{th} traverse.

5.3 Worst-Case Performance Correction of Communicating Systems

In Chapter 4, I have introduced performance correction methods attempting to determine how to increase the performance of the SUT if it is unable to serve the required number of messages per second in worst-case. The methods proposed in this chapter aim at increasing the performance of the SUT at minimal cost. An input of the presented performance correction methods is a Timed Communicating Finite Multistate Machine (TCFMM) model as a formal performance model of the SUT, which was created according to Section 3.4 and Formula 3.1 in Chapter 3. The methods are also given as an input the number of messages the SUT is required to process within a second, in worst case (CWR_{usr}).

The performance increasement of the SUT is achieved by decreasing some of its transition delays and thus, increasing the number of messages the SUT is able to process within a second. Each transition delay can be decreased by predefined, discrete amounts, which amounts might differ for different transitions. Each delay reduction has a cost. The reduced delay of transition t_i will be d_iq_i, where q_i is the so-called correction factor of transition t_i. By reducing an arbitrary instance of the NP-complete knapsack problem to an instance of this, so-called worst-case performance correction problem, I have proven that the problem is NP-complete and formulated it as a binary linear program. I have also given a heuristic algorithm for solving the problem. The presented heuristic algorithm first chooses the lowest possible (and most expensive) value of each correction factor and then runs iterations in some of which some correction factors are increased (and thus, the cost of the correction is reduced) and some are not. The number of iterations that have to pass between two subsequent increasements of a given correction factor is determined by a weight depending on different attributes of the corresponding transition. This weight is defined and assigned to the transition before running the first iteration and it is redefined upon each correction factor increasement of the transition.

I have compared the time and cost efficiency of the heuristic approach to those of solving the binary linear program and to those of a simple round-robin algorithm. In order to compare the performance of the heuristic method to that of the BLP in extreme cases, where the BLP could not be solved within reasonable time, I have relaxed the BLP to a linear program the cost of which is a rough lower estimate for the cost of the BLP. The simulations have found that the higher CWR_{usr} gets, the better the proposed heuristic method performs.

In this chapter, my research objective was to develop an efficient heuristic algorithm for improving the performance of the SUT after it has failed the performance test. According to the simulations run in the chapter, this goal has been reached. The proposed method however, has a limit; it assumes that the transition delays of the SUT can be decreased independently from each other. There might be cases however, in which, delay reduction of different transitions are not independent. Thus, a future direction to develop the methods presented in this chapter is modeling the delay reduction dependencies between transitions.

Bibliography

[1] **ISO/IEC** 9646-1: Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts, 1994.

[2] **ITU-T** ITU-T Recommendation Z.500 – Framework on formal methods in conformance testing, 1997.

[3] **Utting, M and Legeard, B.** Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.

[4] **Pretschner, A. and Philipps, J.** Methodological Issues in Model-Based Testing In *Model-Based Testing of Reactive Systems*, pp. 281–291, 2004.

[5] **Lee, D. and Yannakakis, M.** Principles and Methods of Testing Finite State Machines – A Survey In *Proceedings of the IEEE vol. 84 issue 8*, pp 1090–1123, 1996.

[6] **Dorofeeva, R. and El-Fakih, K. and Maag, S. and Cavalli, A. R. and Yevtushenko, N.** FSM-based conformance testing methods: A survey annotated with experimental evaluation In *Information and Software Technology vol. 52 issue 12.*, pp. 1286–1297, 2010.

[7] **Tretmans, J.** Specification Based Testing with Formal Methods: A Theory In *FORTE / PSTV 2000 Tutorial Notes*, 2000.

[8] **Chow, T.** Testing Software Design Modelled by Finite-State Machines In *IEEE Transactions on Software Engineering vol. 4 issue 3.*, pp. 178–187, 1978.

[9] **Fujiwara, S. and Khendek, F. and Amalou, M. and Ghedamsi, A.** Test Selection Based on Finite State Models In *IEEE Transactions on Software Engineering vol. 17 issue 6.*, pp. 591–603, 2002.

[10] **Sun, X. and Shen, Y. and Feng, C. and Lombardi, F.** Advanced Series in Electrical and Computer Engineering - Vol. 12, Protocol Conformance Testing Using Unique Input/Output Sequences, World Scientific Publishing, 1997.

[11] **Sabnani, K. and Dahbura, A.** A Protocol Test Generation Procedure In *Computer Networks and ISDN Systems vol. 15 issue 4.*, pp. 285–297, 1988.

[12] **Naito, S. and Tsunoyama, M.** Fault Detection for Sequential Machines by Transition-Tours In *Proc. 11th Ann. IEEE Internat. Symp. Fault-Tolerant Comput.*, pp. 238–243, 1981.

[13] **Kohavi, Z.** Switching and Finite Automata Theory, McGraw-Hill, 1978.

[14] **Chul, K. and Song, J. S.** Test Sequence Generation Methods for Protocol Conformance Testing In *Proc. of the Eighteenth Annual International Computer Software and Applications Conference*, pp. 169–174, 1994.

[15] **Dahbura, A.T. and Sabnani, K.K. and Uyar, M.U.** Formal Methods for Generating Protocol Conformance Test Sequences In *Proceedings of the IEEE vol. 78 issue 8*, pp. 1317–1326, 2002.

[16] **Lee, D. and Yannakakis, M.** Testing Finite-State Machines: State Identification and Verification In *IEEE Transactions on Computing vol. 43 issue 3.*, pp. 306–320, 1994.

[17] **Kovacs, G. and Pap, Z. and Csopaki, G. and Tarnay, K.** Iterative Automatic Test Generation Method for Telecommunication Protocols In *Computer Standards & Interfaces vol. 28 issue 4.*, pp. 412–427, 2006.

[18] **Brinksma, E. and Tretmans, J. and Verhaard, L.** A Framework for Test Selection In *Proc. IFIP WG6.1 11th Int. Symp. on Protocol Specification, Testing, and Verification*, pp. 233–248, 1991.

[19] **Nemeth, G. A. and Pap, Z. and Kovacs, G. and Subramaniam, M.** A Bounded Incremental Test Generation Algorithm for Finite State Machines In *Proc. 19th IFIP Int. Conf. on Testing of Communicating Systems (TestCom/FATES)*, pp. 244–259, 2007.

[20] **El-Fakih, K., Yevtushenko, N., von Bochmann, G.** FSM-based Incremental Conformance Testing Methods In *IEEE Transactions on Software Engineering vol. 30 issue 7.*, pp. 425–436, 2004.

[21] **Ghedamsi, A. and Bochmann, G.V.** Test Result Analysis and Diagnostics for Finite State Machines In *Proc. of the 12th International Conference on Distributed Computing Systems*, pp. 244–251, 1992.

[22] **Ghedamsi, A. and Bochmann, G.V. and Dssouli, R.** Multiple Fault Diagnosis for Finite State Machines In *INFOCOM'93. Proc. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. IEEE*, pp. 782–791, 2002.

[23] **Ufuk Celikkan and Rance Cleaveland** Computing Diagnostic Tests for Incorrect Processes In *Proc. of the IFIP Symposium on Protocol Specification, Testing, and Verification*, pp. 263–278, 1992.

[24] **Tretmans, G. J.** Test Generation with Inputs, Outputs, and Quiescence In *Proc. Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS*, pp. 127–146, 1996.

[25] **du Bousquet, L. and Zuanon, N.** An Overview of Lutess - A Specification-based Tool for Testing Synchronous Software In *Proc. 14th IEEE Intl. Conf. on Automated SW Engineering*, pp. 208–215, 1999.

[26] **Raymond, P. and Nicollin, X. and Halbwachs, N. and Weber, D.** Automatic Testing of Reactive Systems In *Proc. IEEE Real-Time Systems Symposium*, 1999.

[27] **Marre, B. and Arnould, A.** Test Sequences Generation from LUSTRE Descriptions: GATEL In *Proc. 15th IEEE Intl. Conf. on Automated SW Engineering*, 2000.

[28] **Clatin, M. and Groz, R. and Phalippou, M. and and Thummel, R.** Two Approaches Linking Test Generation with Verification Techniques In *Proc. 8th International Workshop on Protocol Test Systems*, 1996.

[29] **Koch, B. and Grabowski, J. and Hogrefe, D. and Schmitt, M.** Autolink- A Tool for Automatic Test Generation from SDL Specifications

In *Proc. Workshop on Industrial Strength Formal Specication Techniques*, pp. 21–23, 1998.

[30] **Elvior TestCast Generator** http://www.elvior.ee/motes/

[31] **SpecExplorer** http://research.microsoft.com/en-us/projects/SpecExplorer/

[32] **Huima, A.** Implementing Conformiq Qtronic In *Testing of Software and Communicating Systems*, pp. 1–12, 2007.

[33] **Bokhari, S. H.** On the Mapping Problem In *IEEE Trans. Comput. vol. 30 issue 3.*, pp. 207–214, 1981.

[34] **Lo, V. M.** Heuristic Algorithms for Task Assignment in Distributed Systems In *IEEE Trans. Comput. vol. 37 issue 11.*, pp. 1384–1397, 1988.

[35] **Wu, S. S. and Sweeting, D.** Heuristic algorithms for task assignment and scheduling in a processor network In *Parallel Comput. vol. 20 issue 1.*, pp. 1–14, 1994.

[36] **Efe, K.** Heuristic models of task assignment scheduling in distributed systems In *Computer vol. 15 issue 6.*, pp. 50–56, 1982.

[37] **Kopidakis, Y. and Lamari, M. and Zissimopoulos, V.** On the Task Assignment Problem: Two New Efficient Heuristic Algorithms In *J. Parallel Distrib. Comput. vol. 42 issue 1.*, pp. 21–29, 1997.

[38] **Coffman, E. G. and Garey, M. R. and Johnson, D. S.** An Application of Bin-Packing to Multiprocessor Scheduling In *Siam Journal on Computing vol. 7*, pp. 1–17, 1978.

[39] **Saledo-Sanz, S. and Xu, Y. and Yao, X.** Hybrid Meta-Heuristics Algorithms for Task Assignment in Heterogenous Computing Systems In *Comput. Oper. Res. vol. 33 issue 3.*, pp. 820–835, 2006.

[40] **Salman, A. and Ahmad, I. and Al-Madani, S.** Particle Swarm Optimization for Task Assignment Problem In *Microprocessors and Microsystems vol. 26 issue 8.*, pp. 363–371, 2002.

[41] **Shen, C. C. and Tsai, W. H.** A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion In *IEEE Trans. Comput. vol. 34 issue 3.*, pp. 197–203, 1985.

[42] **Kafil, M. and Ahmad, I.** Optimal Task Assignment in Heterogeneous Distributed Computing Systems In *IEEE Concurrency vol. 6 issue 3.*, pp. 42–51, 1998.

[43] **Ucar, B. and Aykanat, C. and Kaya, K. and Ikinci, M.** Task Assignment in Heterogeneous Computing Systems In *J. Parallel Distrib. Comput. vol. 66 issue 1.*, pp. 32–46, 2006.

[44] **Kellerer, H. and Pferschy, U. and Pisinger, D.** Knapsack Problems, Springer, pp. 487–491, 2005.

[45] **Martello, S. and Vigo, D.** Exact Solution of the Two-Dimensional Finite Bin Packing Problem In *IEEE Management Science vol. 44 issue 3.*, pp. 388–399, 1998.

[46] **Garey, M. R. and Johnson, D. S.** Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., San Francisco, CA, 1990.

[47] **Coffman,Jr., E. G. and Garey, M. R. and Johnson, D. S.** Approximation algorithms for bin packing: a survey In *Approximation algorithms for NP-hard problems*, PWS Publishing Co., pp. 46–93, 1997.

[48] **Kemper, P. and Kritzinger, P. and Bause, F. and Kabutz, H.** SDL and Petri Net Performance Analysis of Communicating Systems In *Proc. of the 15th International Symposium on Protocol Specification, Testing and Verification*, pp. 269–282, 1995.

[49] **ITU-T** Recommendation Z.100 – Specification and Description Language (SDL), 1994.

[50] **Youness, O. S. and El-Kilani, W. S. and El-Wahed, W. F. A.** A Behavior and Delay Equivalent Petri Net Model for Performance Evaluation of Communication Protocols In *Computer Communications, vol. 31 issue 10.*, pp. 2210–2230, 2008.

[51] **Marsan, M. A.** Stochastic Petri Nets: An Elementary Introduction In *Advances in Petri Nets, Lecture Notes in Computer Science, vol. 424*, pp. 1–29, 1990.

[52] **Murata, T.** Petri Nets: Properties, Analysis and Applications In *Proceedings of the IEEE vol. 77. issue 4.*, pp. 541–580, 1989.

[53] **El-Karaksy, M. R. and Nouh, A. S. and Al-Obaidan, A.** Performance Analysis of Timed Petri Net Models for Communication Protocols: A Methodology and Package In *Computer Communications vol. 13 issue 2.*, pp. 73–82, 1990.

[54] **Marsan, M. A. and Chiola, G. and Fumagalli, A.** Timed Petri Net Model for the Accurate Performance Analysis of CSMA/CD Bus LANs In *Computer Communications vol 10. issue 6.*, pp. 304–312, 1987.

[55] **Schieferdecker, I. and Stepien, B. and Rennoch, A.** PerfTTCN, a TTCN Language Extension for Performance Testing In *Proc. of the IFIP TC6 10th International Workshop on Testing of Communicating Systems*, pp. 21–36, 1997.

[56] **ISO/IEC** 9646-1: Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation, 1995.

[57] **ETSI** ES 201 873-1 ver. 4.2.1: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, 2010.

[58] **Dai, Z. R. and Grabowski, J. and Neukirchen, H.** TimedTTCN-3 - A Real-Time Extension for TTCN-3 In *Testing of Communicating Systems*, pp. 407–424, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[59] **Mingwei, X. and Jianping, W.** A formal approach to protocol performance testing In *Journal of Computer Science and Technology vol. 14 issue 1*, pp. 81–87, 1999.

[60] **ISO/IEC** 9646-3 AM. 1: Information technology - OSI conformance testing methodology and framework - Concurrent TTCN, 1993.

[61] **Szabo, J. Z. and Csondes, T.** TITAN, TTCN-3 test execution environment In *Infocommunications Journal vol. 57. issue 1*, pp. 27–31, 2007.

[62] **Hoffman, K. and Kunze, R** Linear Algebra, Prentice Hall, 2nd Edition, pp. 161–162, 1971.

[63] **Goldberg, D. E.** The Design of Innovation: Lessons from and for Competent Genetic Algorithms, Addison-Wesley, Reading, MA., 2002.

[64] **Otten, R. H. J. M. and van Ginneken, L. P. P. P.** The Annealing Algorithm, Boston, MA: Kluwer, 1989.

[65] **RFC 3261** SIP: Session Initiation Protocol, IETF, 2002.

Publications

Journal papers

[J1] **Erős, L. and Bozóki, F.** Test Component Assignment and Scheduling in a Load Testing Environment In *Periodica Polytechnica vol. 52 issue 3–4*, pp. 145–152, 2008.

[J2] **Erős, L. and Csöndes, T.** Model-Driven Black Box Performance Testing of Communicating Systems In *Journal of Universal Computer Science*, under review, 2011.

[J3] **Erős, L. and Csöndes, T.** Model-Driven Performance Correction of Communicating Systems In *Computing and Informatics*, under review, 2011.

[J4] **Erős, L. and Csöndes, T.** Model-Driven Diagnostics of Underperforming Communicating Systems In *Acta Cybernetica*, under review, 2011.

[J5] **Pernek, Á. and Erős, L. and Csöndes, T.** Kommunikáló rendszerek teljesítménytesztelése *Híradástechnika vol. 65 issue 7–8.*, pp. 28-32, 2010.

[J6] **Erős, L. and Bozóki, F.** Refactorisation Methods for TTCN-3 *Acta Polytechnica, Czech Technical University in Prague vol. 47 issue 4–5.*, pp. 33–37, 2007.

Conference papers

[C1] **Erős, L. and Csöndes, T.** Test Component Assignment in a Performance Testing Environment In *Proc. IEEE SoftCom*, pp. 399-403, Split-Dubrovnik, Croatia, 2008.

[C2] **Erős, L. and Csöndes, T.** An Automatic Performance Testing Method Based on a Formal Model for Communicating Systems In *Proc. 18th IEEE International Workshop on Quality of Service*, Paper No. 1569284789, Beijing, China, 2010.

[C3] **Bozóki, F. and Erős, L.** Refactoring Test Data Structures In *Proc. IEEE ConTel 2007: 9th International Conference on Telecommunications*, pp. 123-130, Zagreb, Croatia, 2007.

[C4] **Bozóki, F. and Erős, L.** Eliminating the Redundancy of TTCN-3 Sources In *Proc. TRANSCOM 2007: 7th European Conference of Young Research and Science Workers*, pp. 47-50, Zilina, Slovakia, 2007.

[C5] **Wu-Hen-Chang, A. and Adamis, G. and Erős, L. and Kovács, G. and Csöndes, T.** A New Approach in Model-Based Testing: Designing Test Models in TTCN-3 In *Proc. SDL 2011: 15th International Conference on System Design Languages*, pp. 90-105, Toulouse, France, 2011.