

POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA E INFORMAZIONE

TECHNICAL REPORT



Performance Evaluation Modelling
with JMT:
learning by examples

G. Serazzi *Ed.*

Technical Report n. 2008.09

Piazza Leonardo da Vinci, 32
20133 Milano (Italia)

*To my students,
for their patience ...*

*To myself,
before the night falls*

Every theory, every project
require experimental validation

Contents

Preface

1. Software Applications

- 1.1. Enhancing Process performance through Dynamic Web Service selection 7
- 1.2. Performance Evaluation of Web Services: bottleneck identification 37
- 1.3. A Queueing Network Model of a Parallel Application 51

2. Capacity Planning of Enterprise Systems

- 2.1. Capacity Planning of an Hospital Intranet with Multiclass Workload 64
- 2.2. Intranet with Web Servers and RAID-0 Storage 74
- 2.3. A Network with Finite Capacity Region and Drop Rule 101

3. Computer System Architectures

- 3.1. Performance Evaluation of Computer Memory Hierarchy 108
- 3.2. Shared-Memory Multiprocessor Systems: Hierarchical Task Queue 135

4. Multimedia

- 4.1. Modelling a Surveillance System 150
- 4.2. Performance Evaluation Report on VoIP Gateway Systems 161

5. Security

- 5.1. Encryption Management in Wireless Personal Area Networks 206
- 5.2. Modeling the Performance of SCADA Field Devices 232

6. Networks

- 6.1. Capacity Planning of a Wireless Lan 244
- 6.2. Queueing Network Model of Ad-Hoc Wireless Networks 268
- 6.3. Peer to Peer File Sharing 283

7. Protocols

- 7.1. Modelling of BitTorrent Peer-to-Peer protocol 290
- 7.2. Performance Evaluation of Tairona VoIP Server 310

8. Various Topics

- 8.1. Modeling a Road Junction for Vehicular Traffic Control 348

This material is intended only for a **free distribution**.

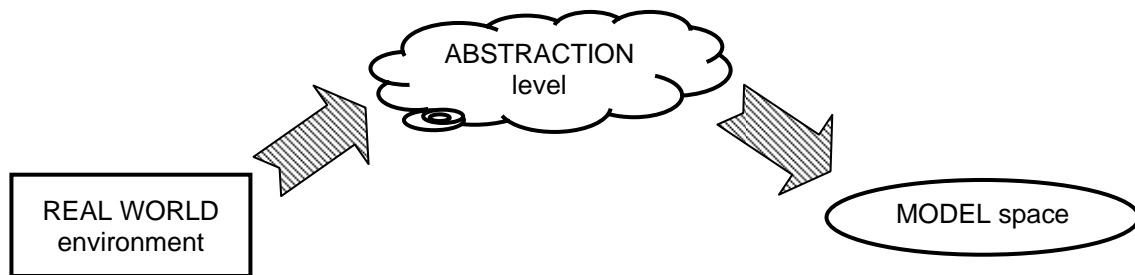
Preface

The goal of this eBook is to improve the performance modeling skills of students and researchers who need to build *accurate*, i.e., *representative*, models of computer systems, networks and applications.

With the rapid growth of computer networking and the increasing demand for complex services there is a strong interest in developing models to evaluate, forecast, and optimize the performance of the digital infrastructures that are implemented.

The construction of a reliable model requires understanding the real world phenomenon, problem, system to be studied. The most important characteristics should be identified, and those not affecting the representativeness of the model should be discarded. Then the model can be designed and parameterized.

The process of conducting a modelling study requires to work at three levels: the real world, the abstraction level, and the modeling space.



The abstraction process brings the real world into the modeling space. It relies upon the *experience* and *creativity* of a person rather than her/his technological background. It neither can be learned with an academic approach nor absorbed passively; it instead builds up on day-by-day trial and error work and experience.

The statements above summarize the rationale behind this eBook. It was felt that readers will benefit from a collection of application oriented projects showing how the abstraction process is conducted in different cases. The basic idea is that of sharing with the readers the experience accumulated by students.

This eBook collects some of the performance evaluation studies carried out by the Master and Ph.D. students of “Computer Systems Performance Evaluation” courses I taught at the Politecnico di Milano, Italy and at the Advanced Learning and Research Institute (ALaRI) of the University of Lugano, Switzerland. The JMT tools (an open source suite that can be downloaded from <http://jmt.sourceforge.net>) were used to construct and solve the models.

The projects are in the form presented by their authors, acknowledged in the front pages, after several interactions. There might be errors (not only from a technical point of view, most of the authors are not English native speakers) and some projects might not even have received a positive evaluation. However, we believe that readers could benefit from their analysis by *concentrating on the abstraction process* and on the *model design* rather than on the actual quantitative results presented in the projects.

I hope you will find this material of some help and apologize in advance for the mistakes you will find. Neither the authors nor myself can be considered responsible for errors that you may introduce in your work due to the content of this eBook. Errors, opinions, conclusions and other information in the projects collected in this eBook shall be understood to be neither given nor endorsed by myself.

Giuseppe Serazzi

Milano, Italy

June 17, 2008

1 – Software Applications

1.1 - Enhancing Process Performance through Dynamic Web Service selection	7
1.2 - Performance Evaluation of Web Services: bottleneck identification	37
1.3 - A Queueing Network Model of a Parallel Application	51

Enhancing Process Performance Through Dynamic Web Service Selection: A Queueing Network Based Approach

Course of Performance Evaluation
December, 12, 2006

Alessandro Bozzon¹, Florian Daniel², Federico M. Facca¹, and Enrico Mussi¹

¹ Politecnico di Milano
Dipartimento di Elettronica ed Informazione
Via Ponzio 34/5 20133 Milano, Italy
[bozzon,facca,mussi]@elet.polimi.it

² Università di Trento
Via Sommarive 14, 38100 Povo (TN), Italy
daniel@disi.unitn.it

Abstract. The dynamic selection of Web services and their integration into composite applications or services is a prominent research challenge. Most of the times, however, the adoption of dynamic selection mechanisms is mainly justified by the increase of *robustness* of the overall system and its higher *availability*. In this paper, we instead investigate another important aspect of the dynamic selection of Web services, i.e. its impact on the *performance* of the resulting system.

Starting from a real-world, process-centric application scenario, we develop a set of possible system architectures that meet the scenario requirements and analyze them by means of proper queue models and simulation techniques. Final goal of the study is to investigate utilization and throughput properties of the different architectures and to identify and eliminate possible bottlenecks by applying dynamic service selection techniques. As the results of this work show, queue modeling techniques provide a powerful means to study the performance of composite applications.

1 Introduction

In service-oriented computing (SOC), developers use services as basic components in their application development processes. In particular, the way in which services are used and the definition of which role an actor may assume during its interaction with a service is specified by the Service Oriented Architecture (SOA), a reference architecture that describes how Service Oriented Computing can be used. The traditional SOA defines three actors: the Service Provider that provides services, the Service Requestor that searches and uses services,

and the Service Broker that provides means to publish and retrieve services [11]. Extensions to the traditional SOA have also been studied [12] in order to introduce advanced management aspects that the traditional SOA does not take into account. Among these problems, the extended SOA offers some interesting hints about dynamic service composition, which is one of the most important capabilities that a service-based application environment must have in order to automate as much as possible the creation and the maintenance of solutions that combine basic services by ordering them to best suit their requirements.

One of the open problems still under investigation by researchers is the automatic selection, aggregation and invocation of services according to real-time changes in the environment that surrounds the applications themselves. In this adaptive scenario, different actors could raise events that may cause the triggering of dynamic mutations in the system (e.g. users changing their functional requirements, availability of new services with better quality and so on...) and, as a response, the system would have to change its internal state accordingly.

Among such events, one of the most tedious problems to cope with is represented by the temporary unavailability of one or more services, or, more in general, by the unacceptable growth of the response time due to the overload of the service provider. Such problem is not of fast practical solution since it depends on facing dynamic changes that can be recognized only at runtime. For this reason, proper formal methods to simulate dynamic behaviors of service networks are needed.

In this work we use queueing networks as a model to analyze the aforementioned problem by representing each tier of a generic service-based application as a queue. This approach has been inspired by the work described in [5] that, besides representing a good approximation for the dynamic service selection, also provides: i) *capacity provisioning*, which enables system architectures to determine how much capacity to allocate to a service in order to support its peak workload, ii) *performance prediction*, which allows the determination of the response time of a service for a given workload and a given hardware and software configuration, iii) *bottleneck identification and tuning*, which allow the identification of system bottlenecks for tuning purposes, and iv) *request policing*, which enables the application to turn away excess requests during transient overloads.

This paper reports our experiences in modeling both static, partially dynamic and completely dynamic Web service publication and consumption by means of queueing networks. The chosen model allows to represent service-oriented applications based on an arbitrary number of layers, each one having significantly different performance characteristic. As proof of concept we used a reference scenario inspired by a European project, the WS-Diamond project ³. Evaluations, consideration and conclusions are performed in an incremental fashion by means of simulations ran with the Java Modeling Tool (JMT) ⁴, an open-source simulation software developed by Politecnico di Milano.

³ <http://wsdiamond.di.unito.it>

⁴ <http://jmt.sourceforge.net/>

This paper is organized as follow: in Section 2 we detail the motivation and the objectives of the work while defining the reference scenario used throughout the paper. In Section 3 we present a brief description of the chosen formal basis and of the simulation tools, while Section 4 reports the results of our model evaluation. Finally, in Section 5 we draw our conclusion.

2 Motivation and Reference Scenario

As already outlined in Section 1, the objective of this work is to show how the dynamic selection of Web services can be used to improve the efficiency of a Web service-based application in order to increase its performances. This section first introduces the motivations behind our works and then describes the scenario we decided to use as a proof of concept for our application of queueing networks models to the problem of dynamic Web service substitution.

2.1 Motivation and Objectives

Usually, Web services are used in a static configuration. The application developer first selects the set of atomic Web services to use for building the application and then, using languages such as WS-BPEL [2], realizes the main application as an orchestration [13] of atomic Web services.

In WS-BPEL, the application is realized as a business process in which operations provided by atomic Web services are combined to create the application logic desired by the developer. Once created, the business process is deployed inside a BPEL engine (e.g., ActiveBPEL [1]) and it is offered to users as an atomic Web service, where the complexity of its implementation is hidden.

The main shortcoming of this approach derives from the fact that, due to both language and engine limitations, Web services used inside the BPEL process cannot be dynamically substituted [9]. The substitution is achievable only by modifying the process to insert the set of atomic Web services first and then redeploying the process again.

Even if for some applications this solution is acceptable (i.e., the cost of redeploy a process is negligible), this may not be true for sophisticated applications where there is the necessity to dynamically select and invoke Web services according to particular conditions that can be evaluated only at runtime.

Analyzing the influence that the dynamic selection of Web services has over applications, it is possible to see that it ensures two types of improvements. First of all, the dynamic selection can be used to increase the satisfiability of users. Each user has its own needs and preferences that cannot be captured by statically defined processes. On the contrary, the dynamic and runtime selection of Web services helps selecting the set of atomic Web services that best fulfill user requirements increasing the user satisfiability. Several work has been done in this area and interested readers are referred to [3, 10].

The second type of improvements that the introduction of the dynamic Web service selection may bring concerns the increase of performances. For example,

through the use of the Web service substitution it is possible to increase the *scalability* (i.e., new Web services can be used without modifying the original structure of the process), the *reliability* (i.e., faulty Web services can be substituted at runtime), and the *throughput* (i.e., Web services are selected at runtime according to their capacity of increasing the system throughput of the main application) of an application.

This paper is mainly focused on performance improvements. In particular, starting from a reference scenario described in the next section, we show how to use the dynamic Web service selection to enhance an existing application that needs to be improved in order to support the growth of its users.

2.2 Reference Scenario

The reference scenario is concerned with a *FoodShop Company* [17] that sells and delivers food using the Web service technology. The scenario is taken from the WS-Diamond Project ⁵, a project of the *Sixth Framework Programme Priority2 - Information Society Technologies* that aims at developing a framework for self-healing Web Services.

The company has an online *Shop* that customers use to select and order food. The Shop does not have a physical counterpart but instead stores and delivers food using either a *Warehouse* or a *Supplier*. The Warehouse is responsible for stocking unperishable goods and physically delivering items to customers, depending on the area each customer lives in. In case of perishable items, that cannot be stocked, or in case of out-of-stock items, the FoodShop Company must interact with the Supplier.

Besides *Customers*, which interact with the FoodShop Company in order to place their orders, pay the bills and receive their goods, also *Employees* use the FoodShop software infrastructure to remotely manage the Shop application and access both Warehouse and Supplier services.

The Shop, the Warehouse, and the Supplier are Web services. They are described by means of WSDL [7, 15] interfaces and electronic interactions among them are carried out exchanging SOAP [16] messages. Both Customers and Employees communicate with the Shop service using the SOAP protocol. Fig. 1 shows how the main actors collaborate:

- C_i represents the generic client of the Shop. A client can be either a Customer or an Employee. As it is possible to see from the picture, the maximum number of clients is unbounded;
- *SHOP* represents the online Shop that provides the operations that both customers and employees use to interact with the system;
- WS_i represents a generic warehouse that may be used during an interaction between the Shop and one of its clients;
- SUP_i represents a generic warehouse that may be used during an interaction between the Shop and one of its clients.

⁵ <http://wsdiamond.di.unito.it/>

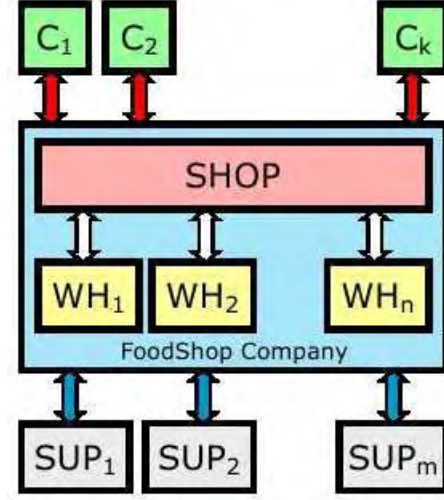


Fig. 1. FoodShop example actors

Fig. 1 also shows how actors are distributed inside the network. The Shop and Warehouses reside inside the same local networks, while Clients (i.e., customers or employees) access the Shop service remotely from the Internet. Suppliers are also accessed by the Shop through the Internet.

2.3 Reference Process Models

In order to better detail the reference scenario, this section briefly describes the workflow of some of the actors. In particular, we take into account the *Customer*, the *Shop*, the *Warehouse*, and the *Supplier* [17]. For sake of brevity and clarity we decided to avoid describing the *Employee* workflow and the operations that the Shop provides for Employees because, apart from specific operations, their informative content is negligible with respect to the performance analysis reported in Section 4.

The Customer

The Customer workflow (see Fig. 2) is abstract and we represent only its interface with the other services, while we do not represent internal activities. The reason is that the customer is an external entity with respect to the company, thus we cannot assume to have its detailed workflow.

The Customer places an order (**sendOrder**) communicating the items he/she is interested in (*items*) and its personal data (*custInfo*). Then it waits for an answer from the Shop: if some of the items are not available the conversation ends (**exit**). Otherwise the user receives the *bill* and decides whether to pay (**replyPay**) sending its *payment* to the Shop. If the Customer decides not to

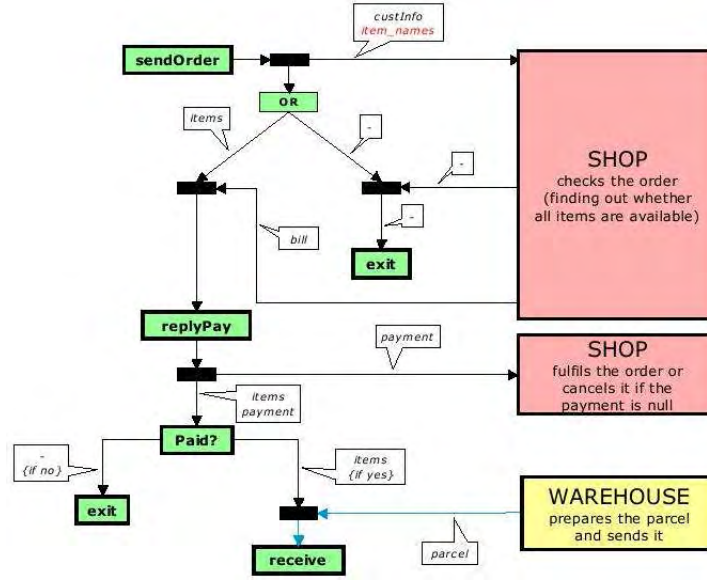


Fig. 2. Customer workflow

pay the conversation ends (**exit**). Otherwise, he/she waits for the parcel sent by one of the companys Warehouse. Notice that the *parcel* shipment is a physical transaction, while the others are all electronic transactions.

The Shop

The Shop workflow (see Fig. 3) is detailed, and contains several internal activities. When the Shop receives an order (**receiveOrder**) with the ordered *items* and the Customer data (*custInfo*), it selects the Warehouse (**selectWH**) and splits (**splitOrder**) the ordered items into the set of perishable items (*ns_items*) and that of unperishable items (*s_items*). It then checks the availability of perishable items (**checkAvail&reserve**) with the Supplier, asking to temporarily reserve them in case they are available. The Shop receives back the set of reserved items (*ns_resitems*), the corresponding reservation codes (*ns_answers*), and the answers on availability (*ns_answers*).

The list of unperishable items is instead sent to the Warehouse (**checkAvail**), that sends back a collective answer (*s_answers*) on availability. If any of the items is unavailable, the order is canceled.

The Shop communicates this to the Customer, and cancels the reservations (**unreserved**) both with the Supplier and the Warehouse.

If on the other hand all the items are available, the Shop asks the Warehouse to compute the ship cost (*shipCost*), which depends on the distance between the Warehouse itself and the user address, as well as the total weight of the ordered

items are out-of-stock, the Warehouse contacts the Supplier in order to check for availability and to reserve them (**findSuppliers**), receiving back the set of reserved items (*s_resitems*), the corresponding reservation codes (*s_rescodes*) and the answers on availability (*s_answers*).

The Warehouse elaborates a collective answer on availability and sends it to the Shop (**collectAnswers**). Then it waits for one of the following things to happen: either the Shop decides to cancel the order, or to proceed.

In the first case the Warehouse has to cancel its own reservations, and, in case some Supplier were contacted, it must also cancel the reservations with the Suppliers (**unreserved**).

In the second case, the Warehouse is asked by the Shop to compute the shipment cost. Then the Shop tells the Warehouse to proceed with the order. In case of out-of-stock items, the Warehouse asks the Suppliers to send the reserved items (**requestSupply**), by providing the reservation codes (*s_rescodes*) and its address (*whInfo*).

At this point the Warehouse must assemble the package. In order to do this, it must wait both for the (unperishable) items it reserved directly from the Suppliers, and for the (perishable) items that were reserved by the Shop.

Once the *parcel* is ready, the Warehouse asks a shipper (**requestShipping**) to send it to the user.

The Supplier

Like the Customer workflow, the Supplier workflow (see Fig. 5) is abstract since each supplier may have a different internal workflow. Of course, it is the same workflow independently from the Web Service that contacts the Supplier. For this reason, the Web Service that buys the goods is generically called Buyer, while the receiver of the products is generically called Receiver. It is clear that in our context the Buyer can be either the Shop or the Warehouse, while the Receiver is always the Warehouse.

The Supplier is first asked by the user to verify the availability of some *items* and reserve them (**verify&reserve**). The Supplier sends back the set of reserved items (*resitems*), the corresponding reservation codes (*rescodes*) and the *answers* on availability. Then the Buyer can either cancel the reservation (**unReserve**) or ask the Supplier to send the items (**supply**) to the address (*sendAddress*) of the Receiver.

2.4 Towards the Dynamic Selection of Web Services

At the beginning, the FoodShop application has been developed defining static links in such a way the the Shop was able to use only one Warehouse and only one Supplier. This means that the first version of the FoodShop application could use only one Warehouse and only one Supplier at a time.

With the growth of Customers, the FoodShop Company needs to increase the number of Employees in order to be able to efficiently manage all the new business transactions. Unfortunately, increasing the Employees is not enough,

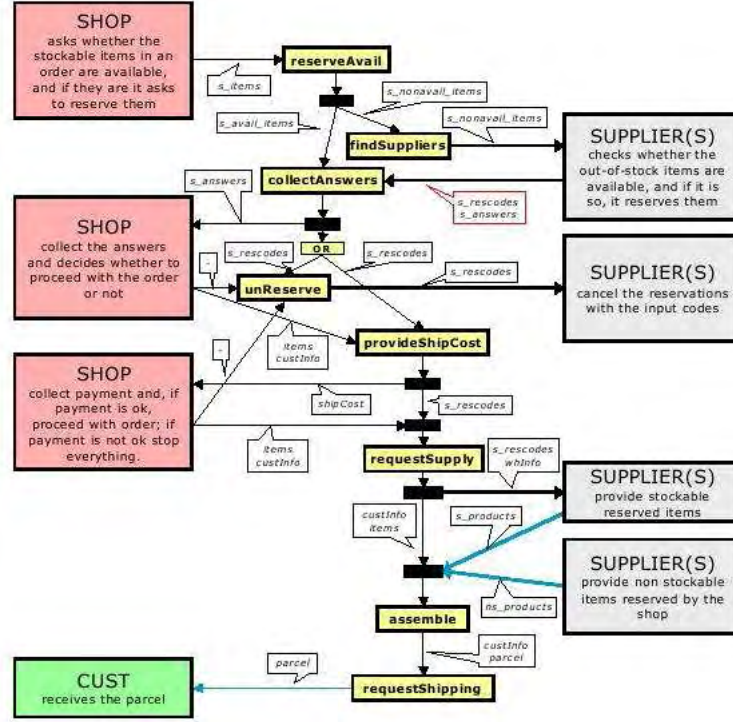


Fig. 4. Warehouse workflow

since the FoodShop Company quickly realizes that the software infrastructure is inadequate to support the new traffic that flows among orchestrated Web services. Moreover, the solution of using just one Warehouse and one Supplier appears as inadequate because they are not able to satisfy the new requests that comes both from new Customers and new Employees. The FoodShop Company decides to introduce the dynamic selection of Web services as a way to solve its infrastructural problems. In that way, the number of Warehouses and Suppliers the the Shop service may use is unbounded and the overall system may benefit of the increased computational capacity.

Section 4 enters into the details and analyzes, using queue modeling techniques described in Section 3, the performance advantages the introduction of dynamic Web service selection may provide.

3 Modeling and simulation instruments

Queueing networks with multiple class models provide estimates for performance measures such as utilization, throughput, and response time. With multiple class models, outputs are given in terms of the individual customer classes and, for

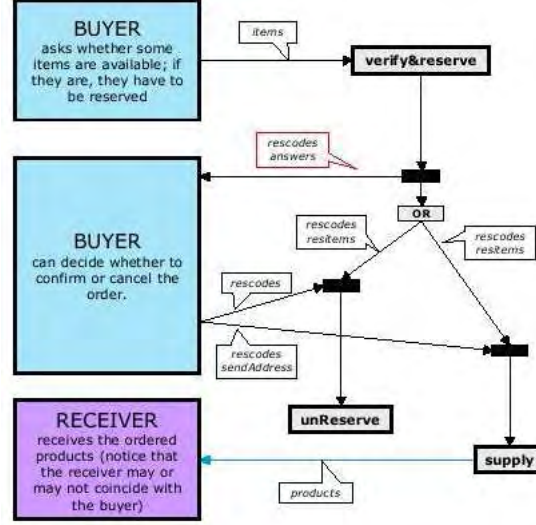


Fig. 5. Supplier workflow

systems in which the jobs being modeled have significantly different behaviors, results are more accurate [8].

3.1 Queueing networks with multiple class modeling

Let C be the number of classes in the model. Each class c is an open class with arrival rate λ_c . We denote the vector of arrival rates by $\bar{\lambda} \equiv (\lambda_1, \lambda_2, \dots, \lambda_C)$. We list below the formulae used to calculate performance measures of interest:

- *processing capacity*: a system is said to have sufficient capacity to process a given offered load $\bar{\lambda}$ if it is capable of doing so when subjected to the workload over a long period of time. For multiple class models, sufficient capacity exists if the following inequality is satisfied:

$$\max_k \left\{ \sum_{c=1}^C \lambda_c D_{c,k} \right\} < 1 \quad (1)$$

- *throughput*: by the forced flow law the throughput of class c at center k as a function of $\bar{\lambda}$ is:

$$X_{c,k}(\bar{\lambda}) = \lambda_c V_{c,k} \quad (2)$$

- *utilization*: from the utilization law:

$$U_{c,k}(\bar{\lambda}) = X_{c,k}(\bar{\lambda}) S_{c,k} = \lambda_c D_{c,k} \quad (3)$$

- *residence time*: residence time is given by:

$$R_{c,k}(\bar{\lambda}) = \frac{D_{c,k}}{1 - \sum_{j=1}^C U_{j,k}(\bar{\lambda})} \quad (\text{queueing centers}) \quad (4)$$

- *queue length*: applying Littles law to the residence time equation above, the queue length of class c at center k , $Q_{c,k}(\bar{\lambda})$, is:

$$Q_{c,k}(\bar{\lambda}) = \lambda_c \cdot R_{c,k}(\bar{\lambda}) = \begin{cases} U_{c,k}(\bar{\lambda}), & (\text{delay centers}) \\ \frac{U_{c,k}(\bar{\lambda})}{1 - \sum_{j=1}^C U_{j,k}(\bar{\lambda})}, & (\text{queueing centers}) \end{cases} \quad (5)$$

- *system response time*: the response time for a class c customer, $R_c(\lambda)$, is the sum of its residence times at all devices:

$$R_c(\bar{\lambda}) = \sum_{k=1}^K R_{c,k}(\bar{\lambda}) \quad (6)$$

- *average number in system*: the average number of class c customers in system can be calculated using Littles law, or by summing the class c queue lengths at all centers :

$$Q_c(\bar{\lambda}) = \lambda_c R_c(\bar{\lambda}) = \sum_{k=1}^K Q_{c,k}(\bar{\lambda}) \quad (7)$$

3.2 The Java Modelling Tool (JMT)

The Java Modelling Tools (JMT) is an open source suite for performance evaluation, capacity planning, and modeling of computer and communication systems. The suite provides the implementation of numerous algorithms specifically design to perform exact, asymptotic and simulative analysis of queueing network models, either with or without product-form solution.

As its authors indicate [4], JMT has been developed with two main objectives: i) to support performance evaluation scientists and practitioners in the analysis of complex systems, and ii) as a didactic tool to help students to understand the basic principles of performance evaluation and modeling. Figure 6 shows a screenshot of the JMT's main window. JMT includes six different tools supporting different analysis frequently used in capacity planning studies:

- **JSIMwiz**: a discrete-event simulator for the analysis of queueing network models. The simulation engine supports several probability distributions for characterizing service and inter-arrival times. JSIMwiz supports state-independent routing strategies as well as state-dependent strategies, and evaluates performance indices like throughput, utilizations, response times, residence times and queue-lengths. *Whatif* analysis, where a sequence of simulations is run for different values of parameters, are also possible;

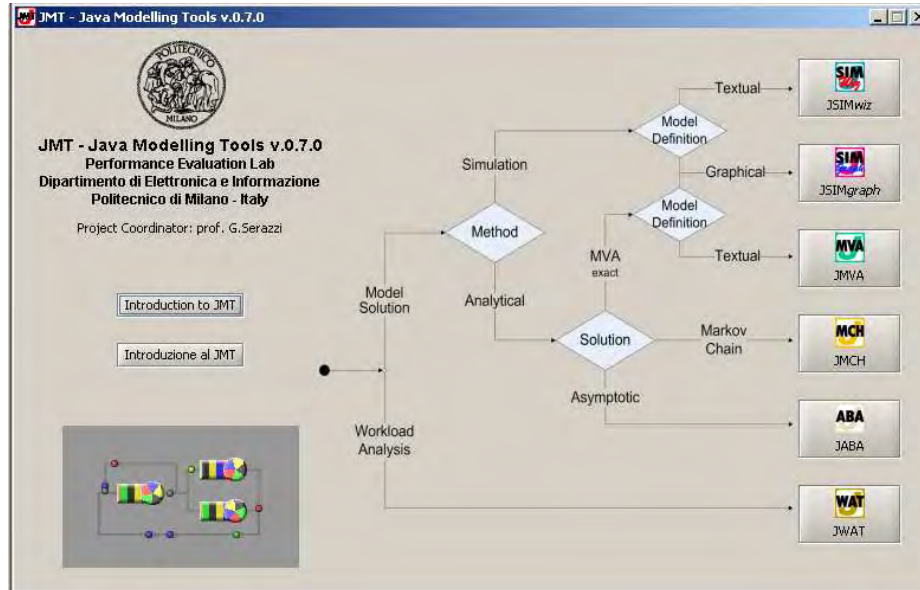


Fig. 6. Screenshot of the Java Modeling Tool (JMT).

- **JSIMgraph**: a graphical user-friendly interface for the simulator engine used by JSIMwiz. It integrates the same functionalities of JSIMwiz with an intuitive graphical workspace;
- **JMVA**: meant for the exact analysis of single or multiclass product-form queueing network models, either processing open, closed or mixed workloads. The computed performance indices are the same of JSIM. What-if analyses are allowed;
- **JABA**: a tool for the identification of bottlenecks in closed product-form networks using efficient convex hull algorithms algorithms.
- **JWAT**: supports the workload characterization phase, with emphasis on Web log data;
- **JMCH** is a didactic tool that applies a simulation technique to solve a simple model and show the underlying Markov Chain;

For sake of brevity we reported only a brief description of each of tool. Interested readers can find more details inside the official JMT website ⁶.

4 System Queue Modeling and Evaluation

Starting from the application scenario presented in Section 2, in this section we discuss the scalability properties of some possible system architectures that could be adopted to support the execution of the described processes. For this

⁶ <http://jmt.sourceforge.net/>

purpose, we will adopt queue models to capture relevant system requirements and to simulate increasing workloads.

We will begin with the case of a *static* architecture, where warehouse and supplier are statically fixed at design time and cannot be changed anymore after the deployment of the system. Next we will show how the *dynamic* selection of the warehouse service, out of a set of three different, equivalent services, allows us to slightly improve the performance of the static system. Then, a third system architecture with dynamic selection of both warehouse and supplier will finally allow us to address all possible bottlenecks in the system and to derive an architecture with excellent scalability capabilities, thus also able to provide support for a large and growing number of process executions.

4.1 Workload Classes

In order to be able to evaluate the performance of the three systems, we need to characterize the expected workloads of the system. As our final goal is to optimize the execution of the shop application, that is the performance of the processes composing the shop application, we will apply two different classes of workload to the systems under investigation, one for each main actor causing the execution of a process:

- *Customers* may connect to the shop service and start the shopping process graphically summarized in Figure 3;
- *Personnel* members may initiate management and maintenance processes (for the sake of simplicity, in this paper we do not discuss the details of the respective process).

Note that the two workload classes, *customers* and *personnel*, represent the execution of a whole process, not just one interaction of a customer or an employee. Both classes are characterized by different service times and visit numbers at the components building up the system; such parameters will be provided when discussing the proposed architectures singularly.

Table 1, shows the initial arrival rates for the two workload classes, which, instead, are the same for all the systems under investigation. Starting from these values, our scalability study will consider arrival rates that are up to ten times higher than these initial values, so as to guarantee a reasonable growth potential of the final architecture.

Table 1. Initial arrival rates of the two workload classes [job/sec].

	Customers	Personnel
Arrival rate	0.6	0.2

4.2 Static Selection

In this section we describe a first system architecture, i.e. the *static* architecture, that will also serve as reference architecture for the discussion of the two proposed dynamic solutions.

Queue Model

Figure 7 shows the queue models that shows the interconnection of *shop*, *warehouse* and *supplier*. Since the object under investigation (the shop) is again a Web service exposed on the Internet, we model the system as open system, which is expressed by the *source* node, which generates incoming requests, and by the *sink* node, which gathers completed jobs.

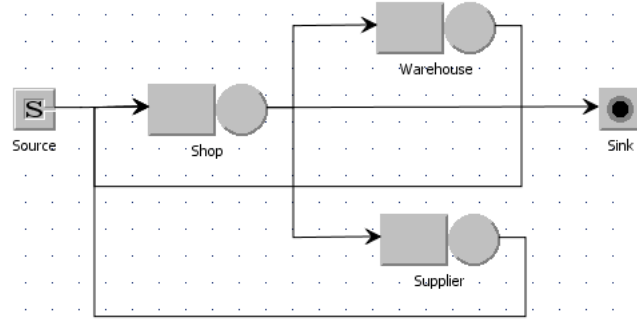


Fig. 7. The static system architecture modeled as queue network.

Table 2 and Table 3 show the service times and the number of visits at the different stations in the queue network for the two workload classes. In general, customer processes have a higher service demand than personnel processes.

Table 2. Service times [msec].

Station	Workload classes	
	Customers	Personnel
Shop	4	3
Warehouse	80	50
Supplier	100	-

Table 3. Number of visits.

Station	Workload classes	
	Customers	Personnel
Shop	1	1
Warehouse	5	2
Supplier	2	0

Evaluation

The evaluation of the scalability properties of the previous queue model is

achieved by means of a so-called “What-if” analysis, which allows us for example to track utilization and throughput values for arrival rates that range from 100% to 1000% of the values listed in Table 1 (in 10 incremental steps).

Utilization Figure 8 shows the dynamics of the utilization of the three stations in the static system architecture in accordance to the increasing workload. The shop (Figure 8(a)) and the supplier (Figure 8(c)) do not present any problem of saturation, while the warehouse (Figure 8(b)) clearly saturates after a 4 times increase of the two workloads, thus limiting the scalability of the overall system.

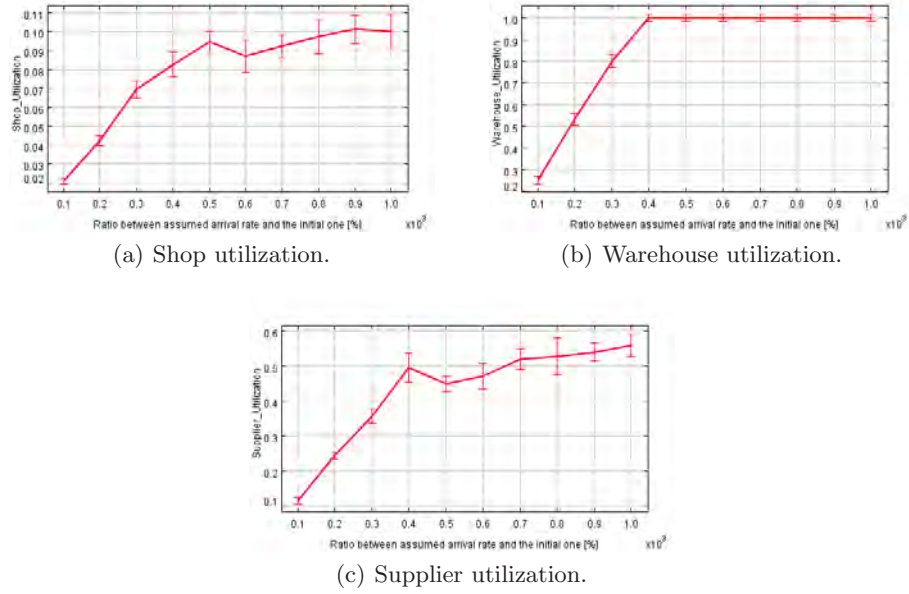


Fig. 8. Utilizations in the static queue network.

System Throughput As can be seen in Figure 9, the slow-down of the system is mainly due to the customers workload class. In fact, while the throughput of the personnel workload class almost linearly scales with the increasing arrival rates (cf. Figure 9(b)), the throughput diagram corresponding to the customer class (Figure 9(a)) shows a clear interruption of the linear growth in correspondence to the saturation of the warehouse, which occurs at arrival rates around 4 times higher than the initial values. The breakdown of the throughput can be seen in both diagrams, but it is slightly more accentuated for customers because they have a higher service demand to the warehouse than the personnel.

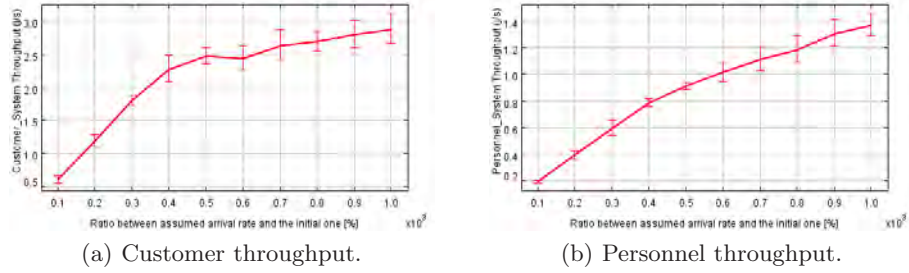


Fig. 9. System throughputs in the static queue network.

4.3 Dynamic Warehouse Selection

To overcome the bottleneck identified in the static system architecture, we propose the introduction of a dynamic Web service selection mechanism, so as to be able to distribute the workload dynamically over a set of warehouses. Each warehouse is represented by its own Web service that can be integrated into our system architecture as described in the following.

Queue Model

Figure 10 represents the system architecture for the execution of our business processes, enriched with a set of three different warehouses and one load balancer representing the dynamic service selection logic⁷. The rest of Figure 10 is analogous to Figure 7.

The dynamic load balancer is in charge of deciding to which warehouse an incoming job should be forwarded. This decision may be based on a variety of different policies, e.g. *random*, *round robin*, *probabilities*, *shortest queue length*, *shortest response time*, *least utilization* or *fastest service*⁸. As our main goal is to enhance the performance of the process executions, in this first step we adopt the *shortest response time* policy for our simulations.

Table 4 shows the new table of service times with respect to the new queue model. In particular, we now have three different warehouses with different performance characteristics also depending on the particular workload class. We do not explicitly take into account the service time of the load balancer.

Evaluation

Analogously to the previous simulation, we perform a “What-if” analysis on

⁷ For the sake of simplicity, in this paper we consider only three different warehouses, but the following considerations also hold in the more general case where we could have an arbitrary large set of warehouses to choose among.

⁸ This set of load balancing policies is already supported by the JMT instrument, but other algorithms could be required and implemented as well.

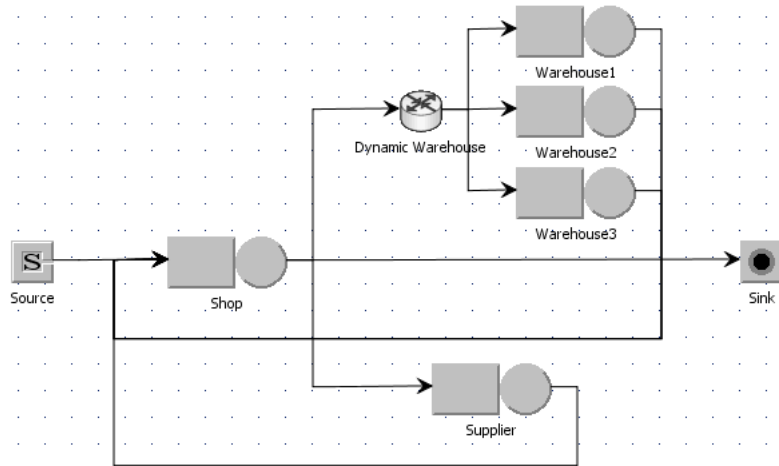


Fig. 10. System with dynamic warehouse selection as queue network.

Table 4. Service times [msec].

Station	Workload classes	
	Customers	Personnel
Shop	4	3
Warehouse1	80	50
Warehouse2	60	40
Warehouse3	100	65
Supplier	100	-

the system with dynamic warehouse selection, varying the arrival rate of both customers and personnel from 100% to 1000% of their initial values.

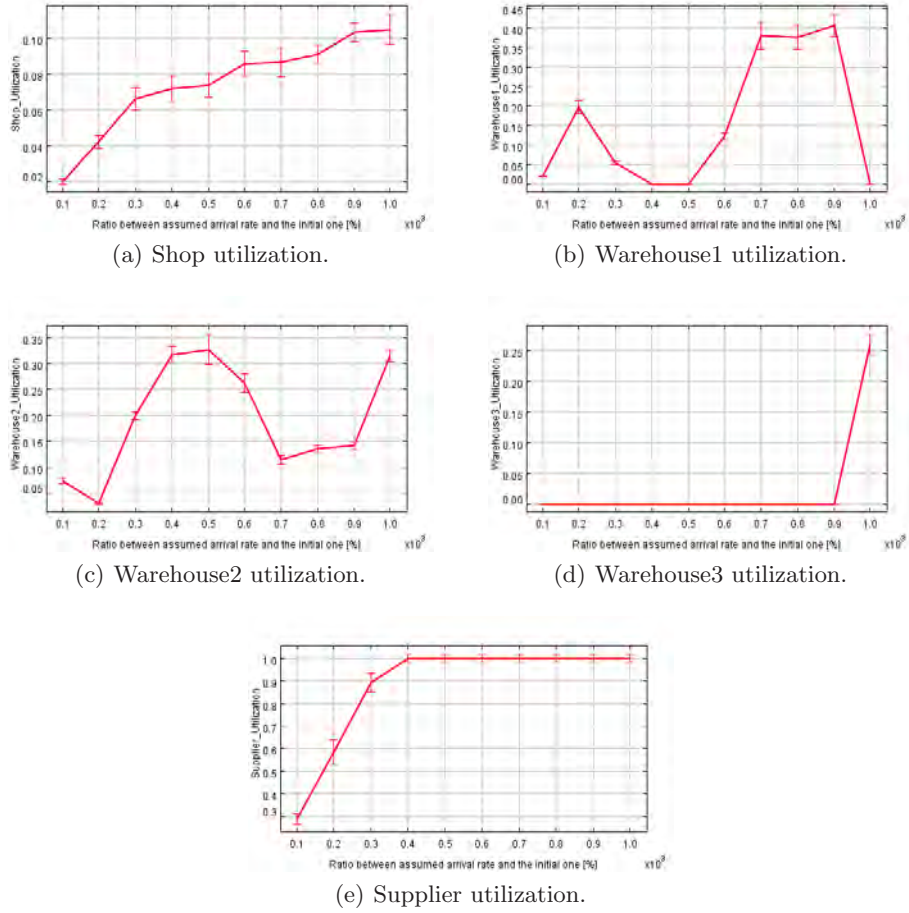


Fig. 11. Utilizations in the queue network with dynamic warehouse selection.

Utilization The utilization diagrams corresponding to the different stations in the queue model and resulting from the performed analysis are depicted in Figure 11.

The utilization of the shop (Figure 11(a)) has not changed from the one that could be observed in the static version of the system architecture (see Figure 8(a)). Based on the observed utilization we can even state that the current shop infrastructure can easily cope with the expected growth of workload.

Figures 11(b) to 11(d) show the dynamics of the utilization for the three warehouses that operate in parallel. Due to our decision to adopt the *shortest response time* load balancing policy, the three warehouses present different dynamics of their utilization. More precisely, the captured dynamics derives from the different service times we have associated to the three warehouses (cf. Table 4). In response to the growing workload, the load balancer first chooses the fastest warehouse (i.e. warehouse 2), then second fastest one (warehouse 1), and only with very high workloads the load balancer also forwards requests to the slowest warehouse (warehouse 3). As can be seen in the diagrams, none of the warehouses saturates, and the more the workload grows, the more the load balancer parallelizes the jobs sent to the warehouses.

However, Figure 11(e) now clearly shows that, once the bottleneck at the warehouse has been resolved through the dynamic service selection, now we have a saturation at the supplier service at arrival rates greater than 4 times the initial values. This behavior can be explained in the following way: if in the static system the bottleneck was represented by the warehouse, now this bottleneck has been resolved, and more jobs per time unit reach the supplier service, which is however not capable of coping with this higher values.

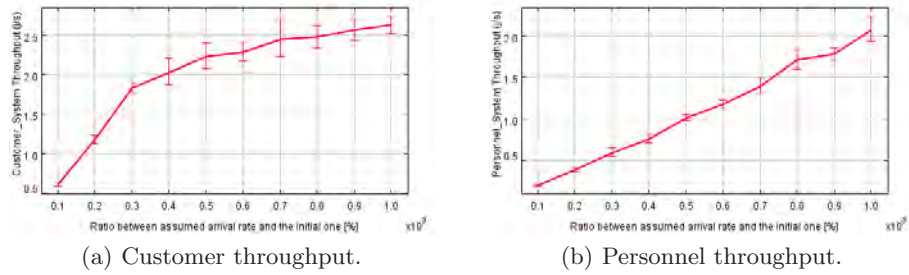


Fig. 12. System throughputs in the queue network with dynamic warehouse selection.

System Throughput Again, the saturation at the supplier service impacts the overall system throughput, as can easily be seen in Figure 12. While in the static architecture the throughput of the personnel was less (but still) affected by the saturation of the warehouse, in this case the personnel is not affected at all by the breakdown of the throughput at the supplier service. In fact, differently from the customer workload class, the personnel workload class makes not visits to the supplier. Therefore, a possible congestion of the supplier service does not impact on the performance of the personnel workflows in execution. To the contrary, we can even say that the more customers are waiting in the queue of the supplier, the more personnel requests can be served at the warehouse.

4.4 Dynamic Warehouse and Supplier Selection

In order to resolve the bottleneck at the supplier service, once again we introduce a dynamic service selection logic and use three different suppliers, instead of just one.

Queue Model

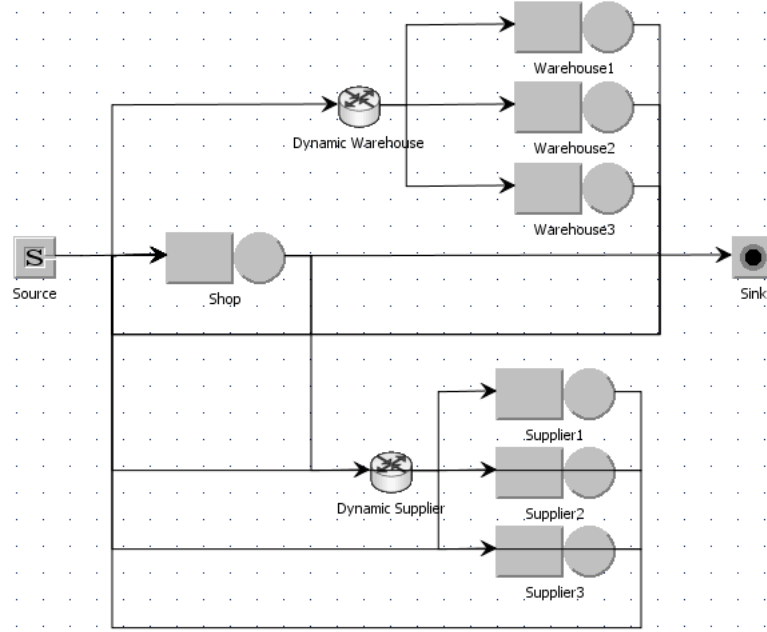
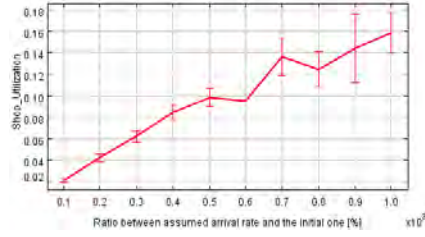


Fig. 13. System with dynamic selection of warehouse and supplier as queue network.

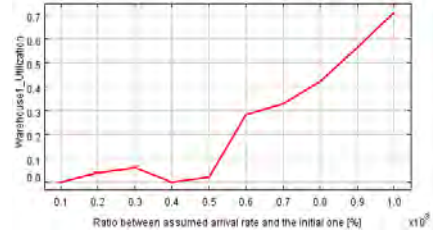
The three supplier services and the dynamic selection logic are introduced into the queue model in the same way as we introduced the dynamic selection of the warehouses. Also to select the appropriate supplier service, we adopt the *shortest response time* load balancing policy. Figure 13 shows the resulting queue model drawn with JSIMgraph, while Table 5 reports the new service times for the three supplier services.

Evaluation

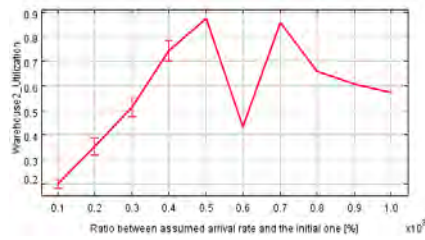
To check the quality of the new system with dynamic selection of both warehouse and supplier, we perform again a “What-if” analysis to track the system behavior in response to the growing arrival rates of customer and personnel requests.



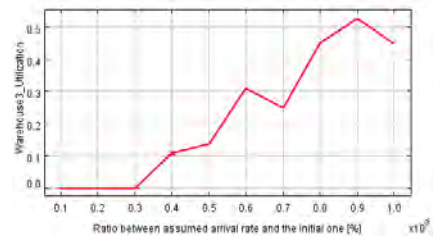
(a) Shop utilization.



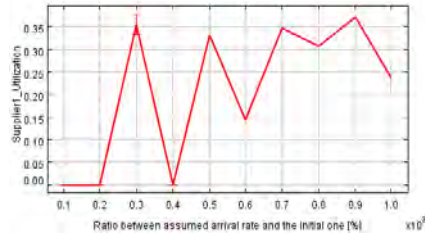
(b) Warehouse1 utilization.



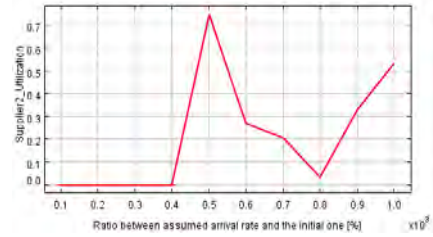
(c) Warehouse2 utilization.



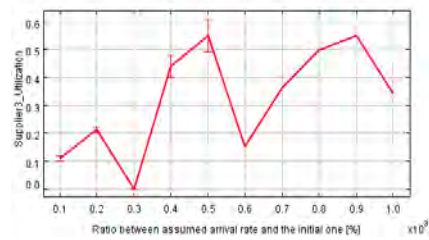
(d) Warehouse3 utilization.



(e) Supplier1 utilization.



(f) Supplier2 utilization.



(g) Supplier3 utilization.

Fig. 14. Utilizations in the queue network with dynamic warehouse and supplier selection.

Table 5. Service times [msec].

Station	Workload classes	
	Customers	Personnel
Shop	4	3
Warehouse1	50	30
Warehouse2	40	20
Warehouse3	60	40
Supplier1	100	-
Supplier2	120	-
Supplier3	90	-

Utilization As expected, also in this new version of the system architecture, the utilization of the shop does not present any saturation problems (cf. Figure 14(a)) and instead scales linearly with the growing workload.

Figures 14(b) to 14(d) and Figures 14(e) to 14(g), respectively, show the utilization of the three warehouses and the three suppliers. Due to the different service times of the Web services and to the adopted load balancing policy, the diagrams again show highly varying behaviors, but none of the Web services saturates.

The two bottlenecks (first the warehouse, then the supplier) have thus been eliminated by introducing suitable dynamic Web service selection mechanisms into the execution of the workflows running in the shop application. The so achieved dynamic system architecture is thus able to cope with the expected workload growth, providing a highly scalable solution for the parallel execution of the business processes described in our reference scenario.

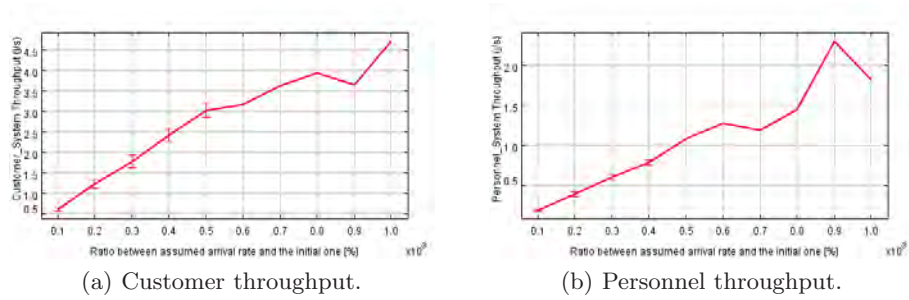


Fig. 15. System throughputs in the queue network with dynamic warehouse and supplier selection.

Throughput The good scalability of the system with dynamic warehouse and supplier selection is further documented by Figure 15, which reports the system

throughput for both customer and personnel processes. In fact, now both curves grow almost linearly with the growing workload.

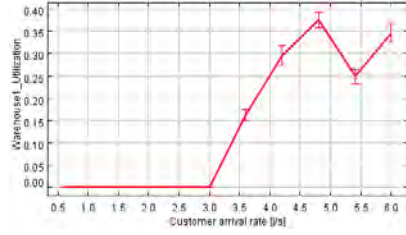
4.5 Advanced Dynamic Warehouse Selection

In the previous experiments with the dynamic selection of Web services we adopted the *shortest response time* strategy for the load balancing activities. If we look at the warehouse or supplier utilizations of the last experiment (cf. Fig. 14(b)- 12(b)), it can easily be seen that this strategy does not always present an optimal behavior. For example, the variance over time of the utilizations of the warehouses or suppliers is large, that is, their utilization grows or decreases rapidly from one time instant to another. Generally speaking, the loads at the warehouses and at the suppliers are not well balanced. A better load balancing policy/strategy is required. In this section, we thus investigate a more sophisticated load balancing algorithm with the aim of optimizing the load distribution over the different warehouses and suppliers.

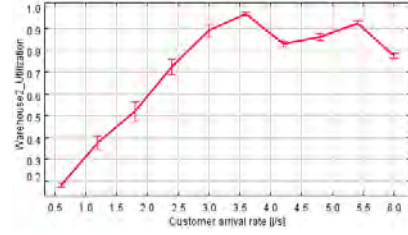
To improve the performance of the system it is possible to derive a routing strategy that is based on optimum loading, according to the available Web Services. However, optimal loading algorithms for multiclass queuing networks are quite complex [14]. Yet, Fig. 11(b) clearly shows that the dominant workload class in the considered problem is the customer class, while the personnel throughput scales up quite linearly (cf. Fig. 11(c)). Based on this observation and for presentation purposes, we decided to simplify the problem and consider the optimization of the workload only with respect to the customer class by switching off the personnel class in the tool simulations. Fig. 16 shows the system behavior when only the customer class is enabled in the dynamic warehouse and supplier selection configuration (cf. Section 4.4). The simulation has been performed using the same arrival growth rate of Section 4.4, i.e. we started from 0.6 job/sec and increased the arrival rate up to 6 job/sec.

In the following, we propose the load balancing algorithm introduced by Chen [6] to study the optimum loading for multilevel storage systems. The *optimum loadings* are defined as the loadings, one for each resource, that maximize the throughput, and hence minimize the mean response time of the subsystem for a given global load.

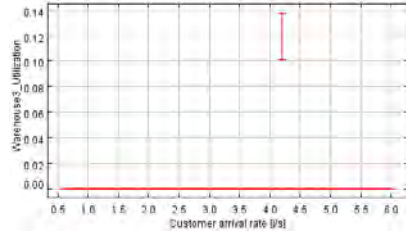
In our case we want to compute the load on each Warehouse Web Service that minimize the response time R of the whole warehouse selection subsystem. We denote λ as the arrival rate of requests to the whole subsystem, and λ_i as the mean arrival rate of the i th Web Service. S_i is the mean service time of the i th Web Service. In the open model proposed we assume that the interarrival times of the λ requests to the Warehouse sub system are exponentially distributed, hence also the interarrival times of requests to the individual Warehouses are exponentially distributed. This allows us to consider each Warehouse as a single independent M/M/1 model with a first come-first served scheduling algorithm. According to these assumptions the mean number N_i of requests in the i th



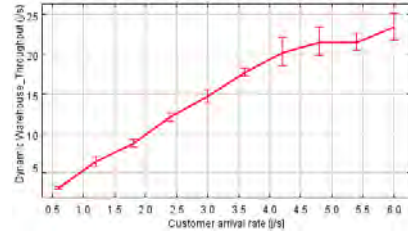
(a) Warehouse1 utilization.



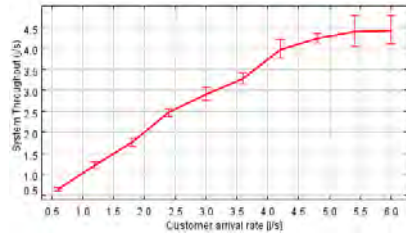
(b) Warehouse2 utilization.



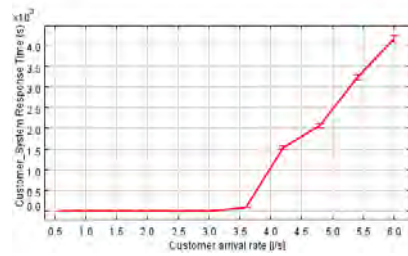
(c) Warehouse3 utilization.



(d) Dynamic Warehouse Selector throughput.



(e) Customer throughput.



(f) System Response Time.

Fig. 16. System behaviour in the queue network with dynamic warehouse and supplier when only the Customer class is enabled (cf. Section 4.4).

Warehouse is:

$$N_i = \frac{U_i}{1 - U_i}$$

where $U_i = \lambda_i \cdot S_i$ is the utilization of i th Warehouse. Thanks to Little's formula ($N_i = \lambda_i \cdot R_i$), we know that for the i th Warehouse the response time is:

$$R_i = \frac{1}{1/S_i - \lambda_i}$$

According to the assumptions, R , the response time of the whole system, is the weighted sum of R_i :

$$R = \sum_{i=1}^n \frac{\lambda_i}{\lambda} R_i = \sum_{i=1}^n \frac{\lambda_i}{\lambda(1/S_i - \lambda_i)}$$

This last formula clearly shows how the response time R of the whole subsystem is related with the load of the single Warehouse, and that to minimize R , we need to compute optimal λ_i according to the total λ load. Then propose an algorithm to solve such problem. The algorithm is valid under the assumption that the devices are ordered by increasing service times (so that $S_1 \leq S_2 \leq \dots \leq S_n$).

Step 1. Compute the quantities $L(k)$ given by:

$$L(k) = \sum_{j=1}^k \frac{1}{S_j} - \sqrt{\frac{1}{S_k}} \sum_{j=1}^k \sqrt{\frac{1}{S_j}} \quad (k = 1, \dots, n)$$

$$L(n+1) = \sum_{j=1}^n \frac{1}{S_j} - \sqrt{\frac{1}{S_k}} \sum_{j=1}^k \sqrt{\frac{1}{S_j}}$$

Step 2. Compute the value of \bar{k} such that:

$$L(\bar{k} + 1) > \lambda \geq L(\bar{k})$$

Step 3. Compute the *optimal values* λ_i^* that are given by:

$$\lambda_i^* = \frac{1}{S_i} - \sqrt{\frac{1}{S_i}} \frac{\sum_{p=1}^{\bar{k}} \frac{1}{S_p} - \lambda}{\sum_{p=1}^{\bar{k}} \sqrt{\frac{1}{S_p}}} \quad (i = 1, \dots, \bar{k})$$

$$\lambda_i^* = 0 \quad (i = \bar{k} + 1, \dots, n)$$

Customer optimum loadings

First of all we order the Warehouses according to their increasing service time: Warehouse2, Warehouse1, Warehouse3. Then, by applying the *step 1* we obtain:

$$L_c(1) = 0$$

$$L_c(2) = 2.23$$

$$L_c(3) = 5.08$$

$$L_c(4) = 39.17$$

According to these results, we can say that: when the number of requests incoming in the Warehouse subsystem are less of 2.23 requests/s the optimal solution is to load only the Warehouse2; when then number of requests incoming is included between 2.23 and 5.08 requests/s the optimal solution is to share the load between Warehouse2 and Warehouse1; when then number of requests incoming is included between 5.08 and 39.17 requests/s the optimal solution is to share the load between all the Warehouses. Finally when the number of incoming requests reach 39.17 requests/s all the components are saturated (i.e., $U_i = 1$ for $i = 1, 2, 3$).

Taking into account *step 2* and *step 3* we can describe the optimal values λ_i^* in function of the total load λ of the considered class:

$$\lambda_1^* = \begin{cases} \lambda, & \text{if } \lambda \leq 2.23 \\ 1.04 + 0.54 \cdot \lambda, & \text{if } 2.23 < \lambda \leq 5.08 \\ 1.83 + 0.38 \cdot \lambda, & \text{if } 5.08 < \lambda \leq 39.17 \end{cases}$$

$$\lambda_2^* = \begin{cases} 0, & \text{if } \lambda \leq 2.23 \\ -1.03 + 0.46 \cdot \lambda, & \text{if } 2.23 < \lambda \leq 5.08 \\ -0.38 + 0.33 \cdot \lambda, & \text{if } 5.08 < \lambda \leq 39.17 \end{cases}$$

$$\lambda_3^* = \begin{cases} 0, & \text{if } \lambda \leq 2.23 \\ 0, & \text{if } 2.23 < \lambda \leq 5.08 \\ -1.47 + 0.29 \cdot \lambda, & \text{if } 5.08 < \lambda \leq 39.17 \end{cases}$$

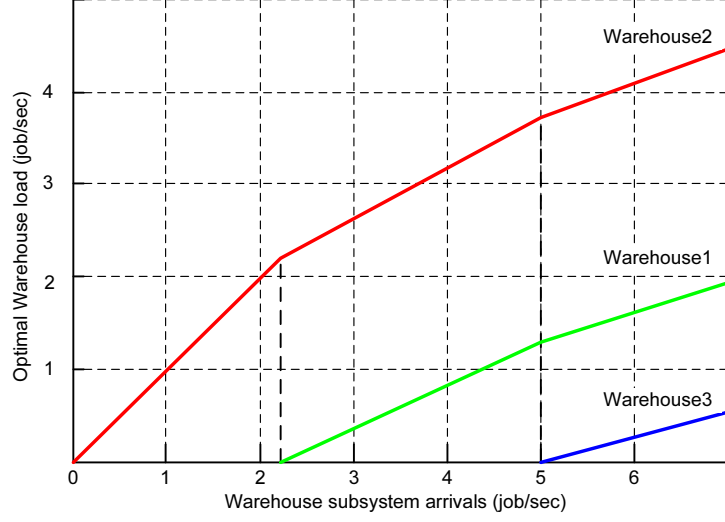
The above equations are graphically presented in Fig. 17(a), while Fig. 17(b) shows the optimal utilization of Warehouses according to the arrivals to the Warehouse subsystem.

To apply the obtained results, we need to know the actual values of the total arrivals to the Warehouse subsystem. To solve the problem we can simply consider the throughput of the Dynamic Warehouse Selector as calculated by the tool (cf. Fig. 16(c)). The Figure shows that the maximum arrival rate to the Warehouse subsystem is 23.367 job/sec when the Customer arrival rate is 6 job/sec. Hence we are still far from saturating the whole warehouse subsystem (39.17 requests/s).

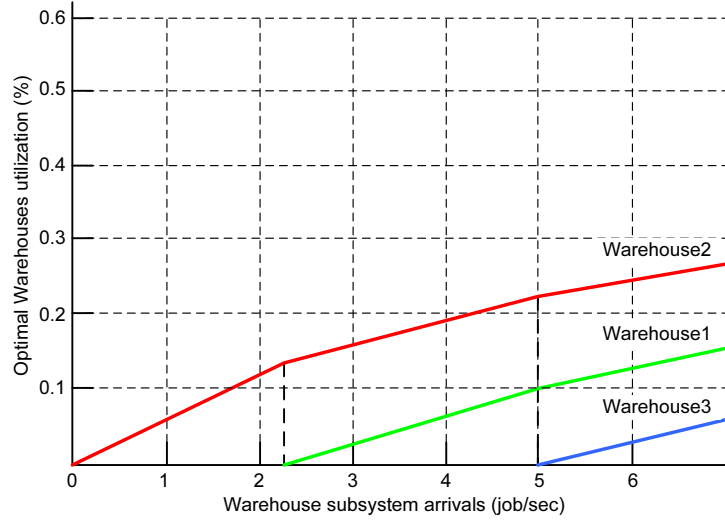
Finally, to compare the whole systems performance with the previous ones, we compute a new simulation in ten steps, corresponding to the growth steps of the Customer arrival rate in the previous experiments. The values used for the experiments are reported in Table 6. Fig. 18 shows the experiment results: while the system throughput gets slightly worse (cf. Fig. 18(a)), the system response time improves definitively (cf. Fig. 18(b)).

5 Conclusions

In this paper we have proposed a queueing networks-based approach to the evaluation of the performance of systems that integrate Web Services and leverage

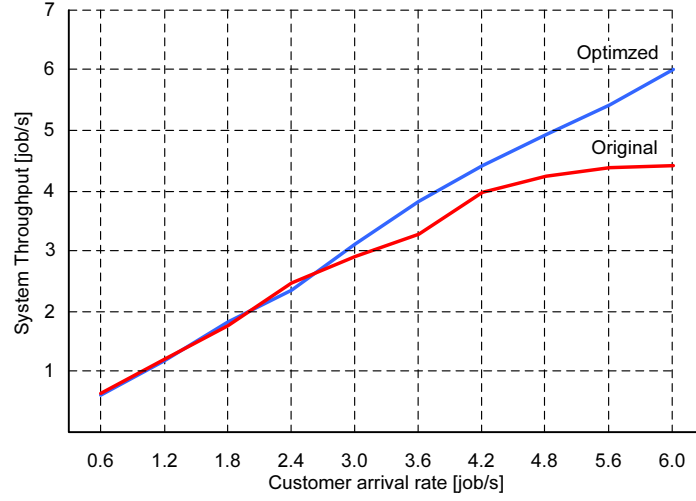


(a) The optimal values λ_i^* in function of the global load λ of the Customer class in the Warehouse subsystem.

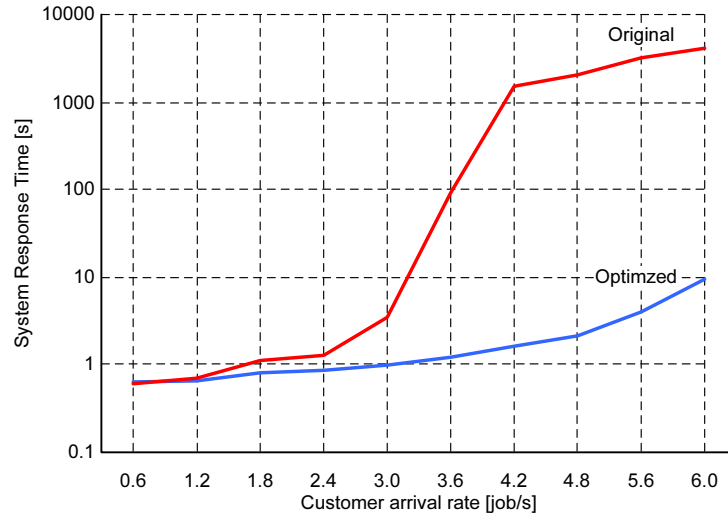


(b) The optimal values U_i^* in function of the global load λ of the Customer class in the Warehouse subsystem.

Fig. 17. The optimal loads and utilization of Warehouses



(a) The comparison between the original system throughput and the modified one.



(b) The comparison between the original system response time and the modified one.

Fig. 18. Advanced selection experiment results

Table 6. Considered arrivals rates [jobs/sec] and associated Warehouses probabilities.

Step	Arrivals		Warehouses probabilities		
	Customers	Warehouses	Warehouse 1	Warehouse 2	Warehouse 3
1	0.6	3.143	0.13	0.87	-
2	1.2	6.419	0.27	0.66	0.07
3	1.8	8.732	0.29	0.59	0.12
4	2.4	12.024	0.30	0.53	0.17
5	3.0	14.632	0.30	0.50	0.20
6	3.6	17.666	0.31	0.48	0.21
7	4.2	20.193	0.31	0.47	0.22
8	4.8	21.446	0.31	0.46	0.23
9	5.4	21.490	0.31	0.46	0.23
10	6.0	23.367	0.31	0.46	0.23

dynamic service selection mechanisms. Based on a food shop scenario, we have shown how the entire process executions can be interpreted as “jobs” in queueing network terms and how this interpretation has allowed us to optimize the shop’s system architecture so as to assure the scalability of the overall system.

In particular, the simulations performed with the JMT instrument on three different possible system architectures has allowed us to demonstrate the effectiveness of dynamic Web Service selection solutions to augment the scalability of service-based systems. Furthermore, we have been able to demonstrate that the dynamic service selection positively affects the overall throughput of the system and, thus, also optimizes the system’s efficiency. The simulation results achieved with the *shortest response time* load balancing strategy however have led to sub-optimal utilizations of the dynamically selected services; hence, we have also shown how an optimal load balancing algorithm could be implemented. We also showed how the used optimal load balancing algorithm contributes to improve the overall response time of the system, reducing the amount of system time required to execute a process, and hence improves the users’ productivity.

References

1. ActiveBPEL. ActiveBPEL Engine. <http://www.activebpel.org/>.
2. T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services Version 1.1, May 2003. <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>.
3. D. Ardagna, A. Avenali, L. Baresi, D. Berardi, D. Bianchini, C. Cappiello, M. Comuzzi, V. D. Antonellis, F. D. Rosa, D. Desideri, C. Francalanci, C. Leporelli, G. Matteucci, A. Maurino, M. Mecella, M. Melchiori, S. Modafferi, E. Mussi, B. Pernici, P. Plebani, and D. Presenza. *Mobile Information Systems. Infrastructure and Design for Adaptivity and Flexibility*, chapter E-Services, pages 47–80. Springer, 2006.

4. M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proceedings of QEST 2006 Conference*, pages 119–120, Riverside, US, Sep 2006. IEEE Press.
5. P. S. M. S. Bhuvan Urgaonkar, Giovanni Pacifici and Asser Tantawi. An analytical model for multitier internet services and its applications. *ACM SIGMETRICS*, pages 291 – 302, 2005.
6. P. P. Chen. Optimal file allocation in multi-level storage systems. In *AFIPS Conference Proceedings*, pages 277–282, Sanibel Island, FL, USA, June 1973.
7. E. Christensen, F. Curbera, G. M. Microsoft, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, March 2001. <http://www.w3.org/TR/wsdl>.
8. G. G. K. S. E.D. Lazowska, J.Zahorjan. *Quantitative System Performance*, Prentice Hall, 1984. Prentice Hall, 1984.
9. D. J. Mandell and S. A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In *2nd International Semantic Web Conference (ISWC '03)*, pages 227–241, Sanibel Island, FL, USA, October 2003.
10. A. Maurino, S. Modafferi, E. Mussi, and B. Pernici. A Framework for Provisioning of Complex e-Services. In *IEEE International Conference on Services Computing (SCC 2004)*, pages 81–90, Shanghai, China, September 2004.
11. OASIS TC. Reference Model for Service Oriented Architecture 1.0. Committee Specification, August 2006. <http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>.
12. M. P. Papazoglou and D. Georgakopoulos. Service Oriented Computing: Introduction. *Communications of the ACM*, 46(10):24–28, 2003.
13. C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.
14. J. Sethuraman and M. S. Squillante. Optimal stochastic scheduling in multiclass parallel queues. In *SIGMETRICS '99: Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 93–102, New York, NY, USA, 1999. ACM Press.
15. W3C Consortium. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. W3C Candidate Recommendation, W3C, January 2006. <http://www.w3.org/TR/wsdl20-primer/>.
16. W3C Working Group. SOAP Version 1.2 Part 0: Primer. W3C Recommendation, June 2003. <http://www.w3.org/TR/soap12-part0/>.
17. WS-Diamond Group. Requirements, application scenarios, overall architecture, test/validation specification, common working environment and standards at Milestone M1. Internal report, 2006.

POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione



PhD course

Advanced topics in computer system performance analysis

09/11/2007

Performance evaluation of Web Services

Authors

Luca Cavallaro

Matteo Miraz

Tutor

prof. Giuseppe Serazzi

1 Introduction

The Service-oriented Architecture (SOA) paradigm foresees the creation of business applications from independently developed services. In this vision, providers offer similar competing services corresponding to a functional description of a service; these offerings can differ significantly in some Quality of Service (QoS) attributes like performance [1]. On the other side, prospective users of services dynamically choose the best offerings for their purposes. Using the SOA paradigm to build applications, services can be dynamically selected and integrated at runtime, so enabling system properties like flexibility, adaptiveness, and reusability.

In this context, the key point is to build applications through the composition of available services. The application can be specified as a process in Business Process Execution Language (BPEL) language in which the composed Web Services (WSs) are specified at an abstract level. The interfaces of individual services are specified using Web Service Description Language (WSDL), the W3C standard to model WSs interfaces, with documented quality properties. Specifically, the agreed performance attributes and levels can be specified by means of appropriate notations that augment the service specifications [2].

Applications built on services face different challenges: on one hand they should ensure that users experience the required performance, and on the other hand they have to maximize the resource utilization, so that provider incomes are maximize. Besides, due to the high dynamism of the applications, a deployment time quality prediction may lose significance during the application life time. In fact the quality of the application depends on the selected services that may be changed at run time.

In this paper we propose an approach to estimate a service oriented application performance. Our approach starts from the description of a service oriented application given using BPEL. This description is translated into a queue network and it is used to run a deployment time analysis of the application performance. This estimation can be used by application clients to monitor QoS. Since performances may change during application run time, due to changes in invoked services, the QoS needs to be monitored, in order to be kept up to date. For this task our approach suggests to annotate the BPEL process with performance indices and to monitor application QoS using WS-CoL (web service constraint language) framework [3]. Using WS-CoL it is possible to become aware of changes in application performance and to react to changes invoking services offering a better QoS.

The rest of the paper is organized as follows: section 2 describes in detail service oriented architectures, section 3 presents a case study to demonstrate our approach, section 4 presents our approach to performance estimation for service compositions, section 5 proposes the results of the approach on our case study, section 6 presents some considerations about the approach and proposes some future developments.

2 SOAs and Web Services

Service Oriented Architecture (SOA) is an evolution of distributed computing and modular programming. SOAs build applications out of software services. Services are relatively large, intrinsically unassociated units of functionality, which have no calls to each other embedded in them. They typically implement pieces of functionality most humans would recognize as a service, such as filling out an on line application for an account, viewing an on line bank statement, or placing an on line book or airline ticket order. Instead of services embedding calls to each other in their source code, protocols are defined which describe how one or more services can talk to each other. This architecture then relies on a business process expert to link and sequence services, in a process known as orchestration, to meet a new or existing business system requirement.

The goal of SOA is to allow fairly large chunks of functionality to be strung together to form ad-hoc applications which are built almost entirely from existing software services. The larger the chunks, the fewer the interface points required to implement any given set of functionality; however, very large chunks of functionality may not be granular enough to be easily reused. Each interface brings with it some amount of processing overhead, so there is a performance consideration in choosing the granularity of services. The great promise of SOA is that the marginal cost of creating the n-th application is zero, as all of the software required already exists to satisfy the requirements of other applications. Only orchestration is required to produce a new application.

A service oriented application usually has to meet the three following requirements:

- Interoperability between different systems and programming languages. The most important basis for a simple integration between applications on different platforms is a communication protocol, which is available for most systems and programming languages.
- Clear and unambiguous description language. To use a service offered by a provider, it is not only necessary to be able to access the provider system, but the syntax of the service interface must also be clearly defined in a platform-independent fashion.
- Retrieval of the service. To allow a convenient integration at design time or even system run time, a search mechanism is required to retrieve suitable services. The services should be classified as computer-accessible, hierarchical or taxonomies based on what the services in each category do and how they can be invoked.

2.1 Web service

A service oriented system is not tied to a specific technology. SOA can be implemented using one or more of these protocols and, for example, might use

a file system mechanism to communicate data conforming to a defined interface specification between processes conforming to the SOA concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without the service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks. For this reasons SOA have been implemented using many technologies.

The most well known SOAs are *web services*. Web services are SOAs that make functional building blocks accessible over standard Internet protocols that are independent from platforms and programming languages, just like HTTP or SMTP. Web services applications meet the three SOA requirements using SOAP, WSDL and UDDI. SOAP (Simple Object Access Protocol) is a protocol similar to RPC that allows to call methods on remote objects residing on different tiers of a network. WSDL (Web Service Description Language) is an interface description language that allows a service to expose information about its interface that may be used by a remote client to invoke a method on the service. UDDI (Universal Description Discovery and Integration) is a registry that allows a remote client to run queries to discover and use web services.

2.2 Service workflow and BPEL

One of the most interesting possibilities of SOA is the possibility for a service to invoke other services using their interfaces. In this way fine-grained services can be composed in more coarse-grained services. This mechanism makes possible incorporating services into business processes and workflows, specified using high level languages.

The most spread workflow language is BPEL. This is an XML language, developed by Microsoft to describe web service compositions. Using BPEL it is possible to define the workflow of a service oriented application. A BPEL workflow can be invoked by a client, just like a remote application, since it exposes to the client information about parameters to provide. BPEL does not provide the possibility to annotate performance indices into the workflow. For this reason an extension of the language is needed. Since BPEL syntax allows annotations the extension can be easily performed.

3 Case Study

A travel agency manager wants to move its company business on the web. He wants to offer a travel booking service. Users accesses to the service through a front end web application. After accessing the web application users can query a map service to plan their travel, then they can book train, plane or taxi to reach the selected destinations. Their tickets can be stored in a cart and, when they are done with their booking, they can pay for their order using a payment service.

The travel agency manager plans that all the customers, in order to access his

on line trip planner, must fill a registration form. Moreover he thinks that there will be two user classes that will use the trip planner: a class that uses the planner to plan trip from a start point to an arrival point, without intermediate stops, and a class that will plan trips featuring intermediate stops.

This process is built using web services. Each step that an user can perform is implemented by a service. Services are then composed in a BPEL workflow, following the schema reported in figure 1. Users access the planner through the front end. This service invokes a map service, called *mappe*, that returns the starting and destination point of the trip. The map service is invoked once by users planning punctual trips and at least twice by users planning trips with intermediate stops. The data retrieved by the map service are returned to the front end that uses them to book tickets on train, plane or tax rides, using three services called respectively *treno aereo taxi*. Tickets data are returned to the front end that uses them for concluding the order. Users booking a punctual itinerary are sent directly to the payment service, called *pagamento*, by the front end, when their ticket is chosen. Users booking an itinerary with intermediate stops are supposed to store their tickets using a shopping cart service, called *carrello*. Finally the reservation is returned to the users.

The travel agency manager chose some existing services to build his composition, and he wants to know which quality of service he can offer to his customers. Since he knows that service oriented applications can be dynamically changed in a cost-effective way he also wants to know how many customers he can serve without make them experiencing a low quality of service and, if any problem arises during the application life time, which services he should change in order to speed up the composition.

4 Our Approach

Our approach is articulated in four main steps:

- Build a service composition, choosing services to invoke
- Build a queue network to model the composition
- Assign performance indices to the invoked services
- Use the *JMT* suite to study system performances [4] [5].

The first step can be performed using BPEL to describe the composition. BPEL is an XML syntax language that allows to specify which services have to be invoked in a composition and how data have to be passed between invocations. The second step consists in modeling the composition as a queue network. Each service in the composition can be modeled as a station with queue. It is necessary to model the composition itself as a station with queue, since the composition can be considered as a service. This station will be called *frontend* and will have service time equals to the time the server hosting composition is busy, excluding the time of other services invocation. If the number of users

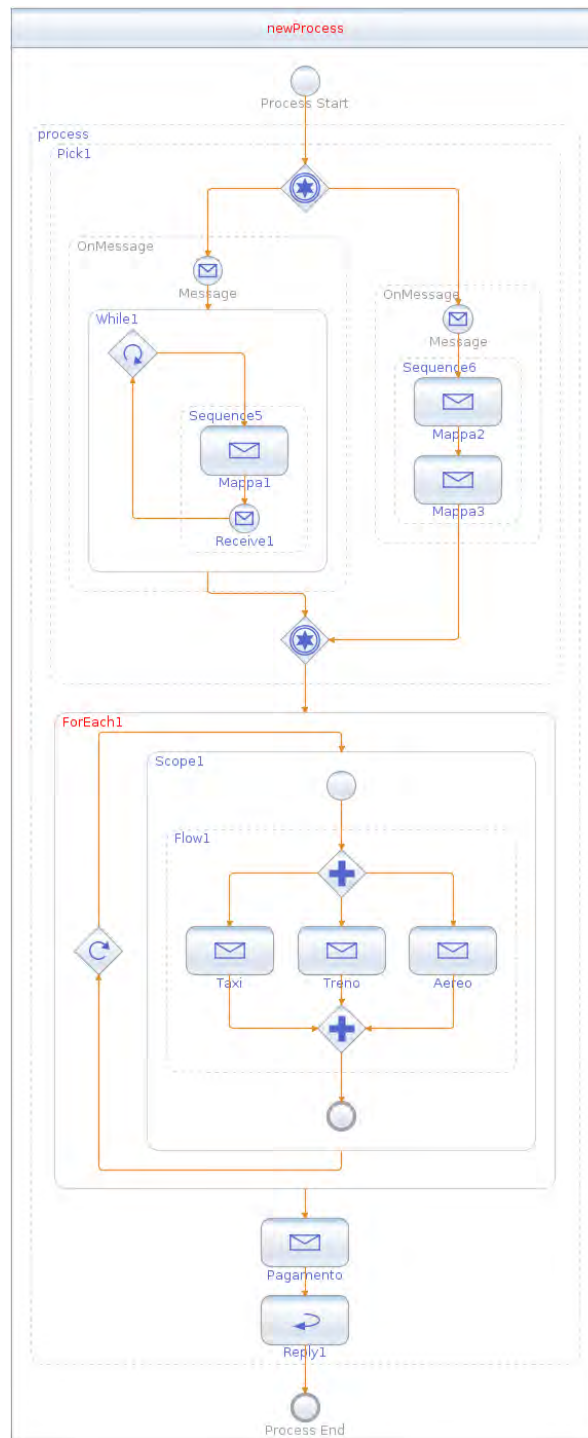


Figure 1: BPEL process

that can use the system is fixed (i.e. users need registration in order to use the composition) then it is necessary to model also users, using a station without queue. The service time of this station will be the think time.

The third step can be performed in two different ways, depending on the goal the performance study has:

- estimating the performances of a web service composition before deploying it
- estimating a web service composition QoS when it is running and deciding if it is necessary to replace a service in the composition.

If we want to estimate performances of a service composition at deploy time the data can be evinced from services logs, if we use existing services, or can be estimated by the system administrator, in case the services are built ad-hoc. In this case performance analysis can be used to estimate system run time performances with a prefixed load and to estimate when the system will become overloaded, using a what-if analysis.

If we want to estimate the composition QoS we need an historical log of service composition performance. This log can be obtained using a service monitor [6] to log residence times and visits for each service.

These data can be used to obtain service demands for each invoked service in the composition. Service demand lattice can be used to estimate bottlenecks in the system and to decide if any invoked service needs to be substituted. The approach is exemplified in section 5 using the case study of section 3.

5 Performance Evaluation

The approach we presented in section 4 was demonstrated on the case study reported in section 3.

The travel agency server composition was translated into a queue network. The network was reported in figure 2. Each service in the composition is modeled as a station with queue, with the same of the service it models. The queue network presents, in addition, two more stations. The station with queue called *frontend* models the BPEL composition itself, as the composition is a service. The station without queue called *users* models the registered users of the system. Since the system is closed we can assume that the number of users is fixed.

We started our analysis assuming that there is a set of services whose service time is known and we have already designed a BPEL workflow that uses those services. In this case we should instrument our application in order to retrieve the services' time requests. Afterwards we should model the service time requested by each service with an exponential process whose lambda is compatible with the measured mean.

Since the goal of this study is to evaluate the validity of the overall approach, we used our experience to propose a set of plausible values for the service time

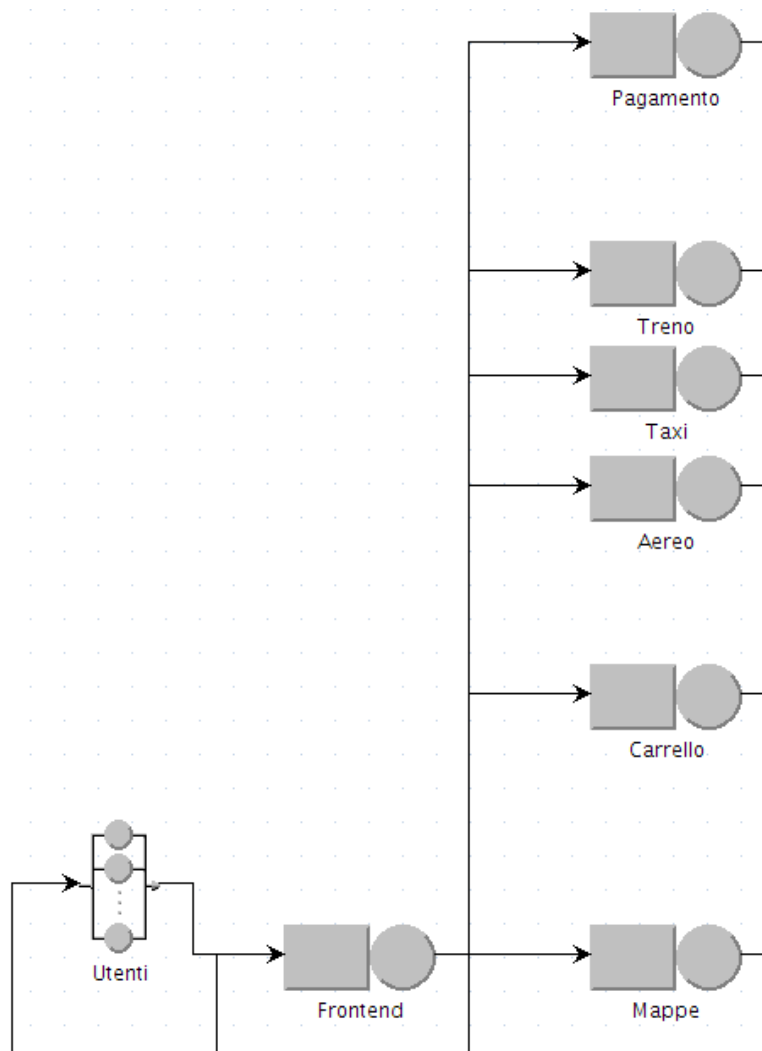


Figure 2: Qn Model of the travel agency service composition

of each web-service. The table 1 reports the mean values we selected for the various services.

	“Puntuale”	“Itinerario”
Frontend	0.0020	0.0066
Carrello		0.0050
Mappe	0.0025	0.0005
Treno	0.0050	0.0010
Aereo	0.0030	0.0008
Taxi	0.0030	0.0008
Pagamento	0.0010	0.0010

Table 1: Service request: mean values.

After selecting the service request, we had to analyze the work-flow and calculate the likelihood of visiting a service. Considering the performance model reported in figure 2, we had to decide the routing of the *fronted* service, and we proposed to use the probabilities listed in table 2.

	“Puntuale”	“Itinerario”
Carrello	0.00	0.20
Mappe	0.37	0.24
Treno	0.09	0.08
Aereo	0.09	0.08
Taxi	0.07	0.12
Pagamento	0.12	0.04

Table 2: Routing probabilities.

In order to retrieve the \mathbf{D} matrix of the system, we run the simulation using JMT (Java Modeling Toolkit [5]) with only one user. In this mode the service requests of the different web-services represent the \mathbf{D} matrix of the system. Table 3 reports the result of that simulation. Looking at the results it is possible to oversee eventual bottlenecks, analyzing the \mathbf{D} for each service. The service with the highest \mathbf{D} value (usually that value is referred with D_{max}) has the greater demand, thus will require more time than other services and will represent the bottleneck of the system. Since the case study has two classes, we can have two different bottleneck, one for each class. In this case we can notice that the *frontend* service represent the bottleneck for both classes, requiring respectively 0.704 s for the “Puntuale” class and 2.148 s for the “Itinerario” class.

In order to improve the system, we propose to enhance the server that host the *pagamento*, doubling its speed. The time required for each visit on that server now is distributed as an exponential process with a lambda factor of 0.0040 for the “Puntuale” class and 0.0110 for the “Itinerario” class. We re-run the simulation with the new values, and we got the new values of the \mathbf{D} matrix of the system, that are reported in 4. Analyzing this table we can notice that the *pagamento* is no longer the bottleneck of the system, so enhancing the server

	“ <i>Puntuale</i> ”	“ <i>Itinerario</i> ”
Frontend	0.704	2.148
Carrello	0.000	0.165
Mappe	0.610	1.951
Treno	0.079	0.252
Aereo	0.122	0.396
Taxi	0.074	0.593
Pagamento	0.510	0.161

Table 3: Initial service residence time.

that hosts it we are able to remove the original bottleneck.

	“ <i>Puntuale</i> ”	“ <i>Itinerario</i> ”
Frontend	0.350	1.027
Carrello	0.000	0.161
Mappe	0.599	1.741
Treno	0.075	0.261
Aereo	0.112	0.404
Taxi	0.074	0.605
Pagamento	0.496	0.167

Table 4: Simulated service residence with enhanced front-end service.

Continuing the analysis on the new system, we can detect a new bottleneck, that is the *mappe* service. That service requires 0.599s for processing a user of the “puntuale” class and 1.741s for a user of the “itinerario” class, that is more than any other service. Again, in order to improve the overall performance, we can enhance the speed of the *mappe* server. For this reason, we model the time requested by the *mappe* service to process each visit as an exponential process with a lambda factor of 0.0011 for the “itinerario” class and of 0.0050 for the “puntuale” class. We re-run the simulation and we got the new **D** matrix, that is reported in table 5.

Analyzing the result of the simulations, we can detect that the enhancement is useful since the bottleneck on the *mappe* service is resolved. The new system has a different bottleneck for each class: the one for the “puntuale” class is the *pagamento* service (imposing the D_{max}^p to 0.5 s), while for the “itinerario” class the bottleneck is the *Frontend* service (imposing the D_{max}^i to 1.041 s). In order to better understand the bottlenecks of the system, we plot the D values in a convex hull diagram, that is reported in figure 3.

Analyzing the convex hull diagram, it is possible to visualize better the bottlenecks of the system. The *frontend* and *pagamento* services are two potential bottlenecks, while all the other services are dominated by this two services. For

	“Puntuale”	“Itinerario”
Frontend	0.352	1.041
Carrello	0.000	0.166
Mappe	0.300	0.900
Treno	0.075	0.266
Aereo	0.112	0.400
Taxi	0.075	0.600
Pagamento	0.500	0.166

Table 5: Simulated service residence with enhanced front-end and map services..

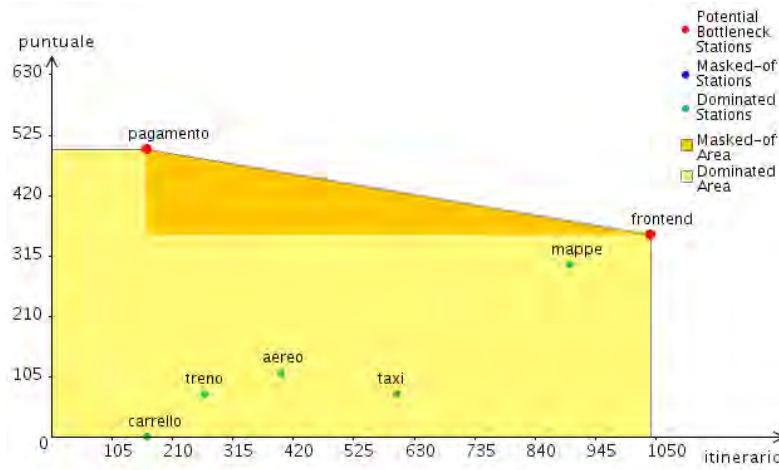


Figure 3: Convex Hull

this reason we can concentrate our analysis only on those two services. Moreover the convex hull shown that the bottlenecks services are used in a different way from the two classes; for this reason we investigate the effect of the population mix on the performance. In multiple workload mixes across multiple resources systems, changes in workload mixes can change the system bottleneck; the points in the workload mix space where the bottlenecks change are called *crossover points*, and the subspaces for which the set of bottlenecks does not change are called *saturation sectors*. The first analysis we performed is the calculation of these saturation sectors.

In figure 4 is reported the saturation sector of the system, calculated analytically with JMT. If the population has only customers of the “itinerario” class, the saturated service is *frontend*; otherwise if the population is composed only by customers of the “puntuale” class, the saturated service is *pagamento*. If the users of the system are composed by a particular mix of these two classes, both servers are saturated. If the system is in this situation, it has the maximum

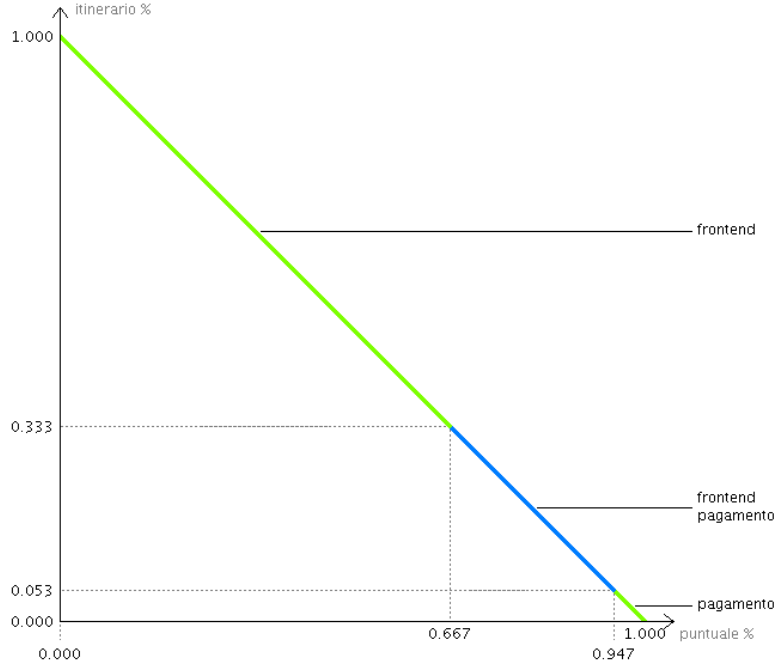


Figure 4: Saturation sectors

throughput since two servers are used at their maximum capacity. The system that we are analyzing has an optimal mix that has customers of the “puntuale” class between the 66% and 95%.

In order to validate this result, we performed a *what-if* analysis with a fixed population size and varying the mix of the two classes. In figure 5 is reported the utilization of the three more used services (*frontend*, *pagamento* and *mappe*) with respect to the population mix. The highlighted region is the common saturation sector: if there are the correct mix of users, there are two servers that are used almost completely. In this case we have the highest throughput, as measured and plotted in figure 6.

Analyzing the results of this analysis, we can state that if we are smart regarding the selection of users, we can serve more people without any server upgrade. If we are able to dynamically monitor the execution of the workflow, understanding the class that the user being served belongs to, we can detect the current population mix. At this point we can understand if the system is saturated, and in this case we should also check if the population mix is within the common saturation sector. If the current mix is not in the saturation set, we should take actions that tries to modify that mix. If the error is only a transient one, we can adopt an intelligent routing strategy that select the job belonging to the right class. Otherwise the problem is persistent, thus we can adopt ad-hoc promotions that helps in filling the gap in the population mix.

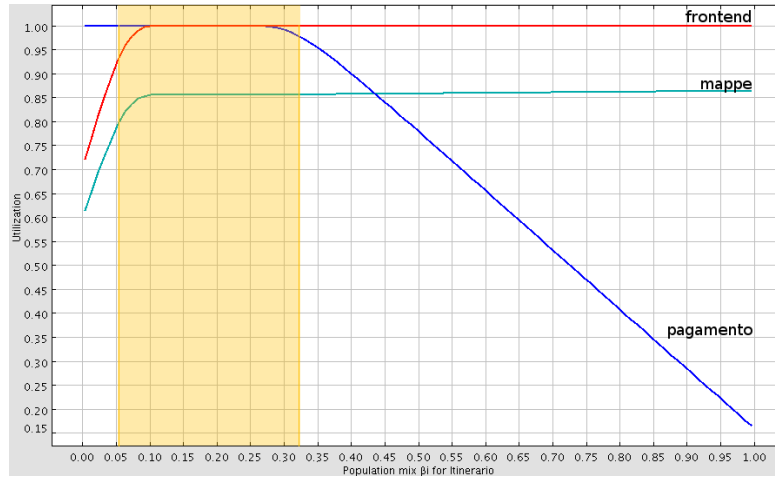


Figure 5: What-if analysis: utilization w.r.t. population mix

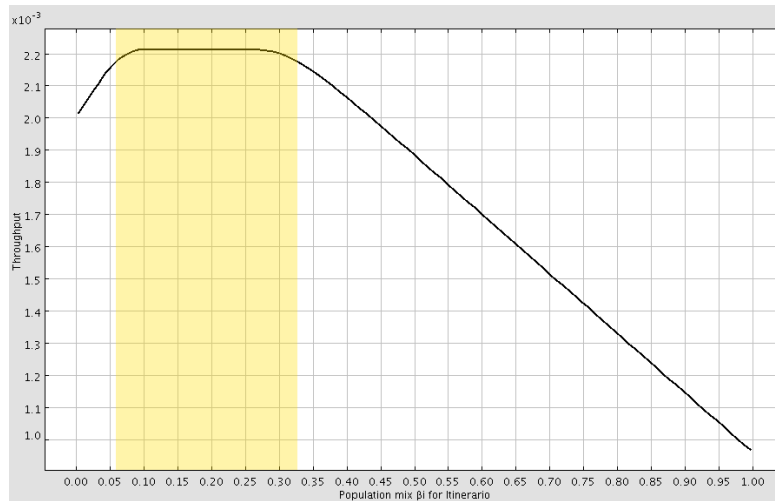


Figure 6: What-if analysis: throughput w.r.t. population mix

6 Conclusion and Future Works

In this paper we described an approach for the performance prediction of service-based applications. The approach can be used both to estimate performances of a service composition at design time and to analyze composition QoS at run time.

We demonstrated our approach on an example composition describing a travel agency application. In our demonstration we could estimate the application performances at deployment time, manually creating a QN model of our composition using as performance indices distributions of the service times of the services invoked in the composition. We also used these data to estimate bottleneck of the system, to study saturation sectors and to plan a speed up of some invoked services.

The approach was run manually, but it can be automatized. As a first step towards the realization of this approach an automatic translation from the BPEL specification to a queue network model is needed. This translation can take as input a BPEL file and give as output a JSymGraph model file. This model allows analyzing the queue network derived from the BPEL composition using JMT. To make possible this translation we need to extend the BPEL language to make it possible to annotate QoS indices directly into the composition description.

References

- [1] Daniel A. Menascé. Qos issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.
- [2] Andrea D'Ambrogio. A model-driven wsdl extension for describing the qos of web services. *icws*, 0:789–796, 2006.
- [3] L. Baresi and S. Guinea. Towards dynamic monitoring of ws-bpel processes. In *ICSOC05, 3rd International Conference On Service Oriented Computing*, 2005.
- [4] Serazzi G Bertoli M., Casale G. The jmt simulator for performance evaluation of non-product-form queueing networks. In *ANSS07 Spring Simulation Multiconference*, 2007.
- [5] G. Serazzi. Java modeling tools. <http://jmt.sourceforge.net/>.
- [6] Luciano Baresi and Sam Guinea. Dynamo and self-healing bpel compositions. In *ICSE COMPANION '07: Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 69–70, Washington, DC, USA, 2007. IEEE Computer Society.

Politecnico di Milano

Dipartimento di Elettronica e Informazione

Via Ponzio, 34/35 - 20133 Milano (MI) - Italy

Course: Advanced Topics in Computer system performance analysis

Teacher: Prof. Giuseppe Serazzi

Date: November, 2007

A Queueing Network Model of a Parallel Application

Alessandro Lazaric and Roberto Turrin

{lazaric, turrin}@elet.polimi.it



1 Introduction

In recent years, large-scale problems have driven the development of high performance computing (HPC) systems and a new class of applications based on parallel computation. In fact, by means of parallel computation, the problem is decomposed into lighter sub-problems faster to be solved. As a drawback, partitioning an application does not have a linear speed-up due to the presence of an intrinsic serial fraction and to the overhead due to communications and synchronization among distributed resources. Indeed, there exists an ideal trade-off between partitioning and resulting overheads. Several performance models have been proposed in the literature to evaluate the impact of such factors on the system performance.

In this report, we describe and simulate a queueing network model to analyze the performance metrics of a real parallel application based on a Monte-Carlo method. The rest of the report is organized as follows. In Section 2 we introduce the performance model. The parallel application is described in Section 3 and its performance is evaluated in Section 4. Finally, we trace the conclusion in Section 5.

2 Parallel application performance models

Parallel applications represent a family of algorithms in which the processing is decomposed in many atomic and independent parts, referred to as *tasks*, which are executed simultaneously using different resources. According to the application characteristics and to the underlying architecture, there exist an optimal way to split an application, which can be obtained by means of suitable performance models. The dependency relationships among the tasks are typically expressed by means of task graphs. A task graph contains complete information about the application inherent parallelism: nodes represent tasks, and arcs represent constraints (e.g., input/output data dependency or control flow) [4].

The first effort to model the performance of parallel programs is due to Amdahl and the well-known homonymous law [2], which states that the maximum improvement of a parallel computation is upper limited by the inverse of the application serial fraction. For instance, Figure 1 shows the behavior of the speed-up of a parallel application as a function of the number of processors allocated. Since the Amdahl's law does not take into account the effect of communication and synchronization among the resources (e.g., processors) on the performance (e.g., execution time, speed-up), different extensions of such model have been proposed [7, 8].

We base our analysis on the model presented in [5]. Let us assume the application under analysis has an average sequential execution time μ , which is the execution time expected when performed on a single node. The average execution time of the whole application on multiple machines is defined as:

$$T(n) = \frac{1}{S(n)}\mu \quad (1)$$

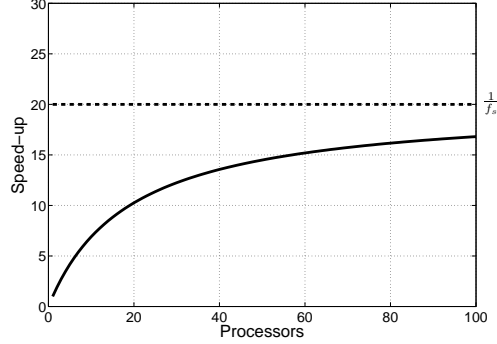


Figure 1: Speed-up with serial fraction equals to 5%.

where $S(n)$ is a speed-up factor depending on the number of nodes n . $S(n)$ can be an aggregation of various factors.

Job partitioning divides a job into *tasks* that can be executed independently and in parallel on a number of nodes to improve performance. The average service time of one task can be defined as:

$$\tilde{\mu} = \mu \frac{1 + (n-1)f_s}{n} \quad (2)$$

where f_s is the serial fraction of the specific job as defined by Amdahl in [2].

However, the execution on a parallel system brings to high variability in computation and communication time. As a result, the synchronization among the nodes smooths the benefits of partitioning. In general, the higher the variability of computation, the greater the impact of synchronization. Such overhead factor is referred to as the *synchronization overhead* $O_n \geq 1$. As a result:

$$S(n) = \frac{n}{1 + (n-1)f_s} \frac{1}{O_n} \quad (3)$$

In order to compute O_n , it can be used a *fork-join queuing network* [3], consisting of n identical delay service centers, each one representing a node. Once the distribution of the task service times is known, it is possible to derive O_n analytically.

3 Matrix Inversion algorithm

In this section we present a real parallel application that will be analyzed by means of a queueing network (QN) model.

The application addresses the problem of inverting diagonally-dominant matrices. A generic matrix A is called diagonally-dominant if $|A_{ii}| \geq \sum_{j \neq i} |A_{ij}|, \forall i$. The algorithm has been implemented with a Monte Carlo Markov Chain (MCMC) method.

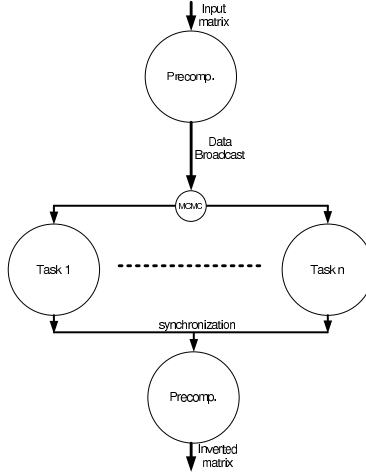


Figure 2: Task graph model of the application.

3.1 Application details

The algorithm has been implemented accordingly to the master-slave paradigm and utilizes the interface for parallel algorithms MPI. The task graph in Figure 2 explains the algorithm workflow. At the beginning the master prepares the data, calculating the intermediate matrices. Then the intermediate matrices are broadcasted to the slaves. As soon as all the data are sent the slaves start the computation. Afterward they send the result of their computation to the master. Once the master has collected all the slave partial results, it assembles them in the inverted matrix, which is the output of the process altogether. The application receives, as input, 8000x8000 matrices.

3.2 Application modeling

We first present a brief analysis of the application assuming that it receives a single matrix as input.

The incoming job (i.e., the matrix to be inverted) enters the master. The master pre-computes the intermediate matrices (pre-processing) which are sent to the slaves (data broadcast). Thereafter, each slave starts computing independently the assigned task (MCMC) and then it sends the result back to master. Finally, when the master has received all the partial results, it elaborates them (post-processing) returning the inverted matrix.

The model parameters are the following:

- The pre-processing service time is pretty constant ($S_{\text{PRE}} = 4.156$ seconds, with coefficient of variation 0.0015446).

- The post-processing is negligible ($S_{\text{POST}} = 0.009771$ seconds, with coefficient of variation 1.5191).
- The local communication network is a 1Gbps switched Ethernet.
 - The amount of data broadcasted to the slaves is $8K(2K+1)$ bytes, where K is the number of rows/columns of the input matrix, in our case $K = 8000$. This is a locking point-to-multipoint communication which uses the MPIBCast function. The service time of the data broadcasting is given by:

$$S_{\text{NET}} = K(2K+1) \cdot (0.2733 + 0.000413n) \mu\text{sec} \quad (4)$$

- The amount of data sent from a slave to the master is $8K^2/n$ bytes, i.e., it decreases linearly with the number of nodes. This is a point-to-point communication which can be neglected.
- The slave service time follows a Weibull distribution with scale B and shape α . The slave service time expectation is expressed by (2), where μ and f_s are, respectively, the sequential time and the serial fraction of the split code. Furthermore:
 - The Weibull shape is assumed to be constant and equals to 1.3.
 - The coefficient of variation γ is related to Weibull shape as:

$$\gamma = \sqrt{\frac{\Gamma(\frac{\alpha+2}{\alpha})}{\Gamma(\frac{\alpha+1}{\alpha})} - 1} \quad (5)$$

- The serial fraction f_s is 3.8%
- The sequential time μ is 721.5 sec

As a consequence, the synchronization overhead O_n in (3) is given by [6]:

$$O_n = \sum_{k=1}^n \binom{n}{k-1} \frac{(-1)^{n-1}}{(n-k-1)^{1/\alpha}} \quad (6)$$

3.3 Queueing Network Model

We extend the previous model to the case in which the system manages a flow of matrices, i.e., we introduce queueing service centers. According to the application workflow, the inversion of a matrix can be divided in one sequential phase (i.e., the pre-processing and the data broadcast) and a parallel phase (i.e., the *MCMC* algorithm). The resulting model is depicted in Figure 3, where the system is composed by two service centers, the *master* and the *MCMC*. While the master performs the pre-processing and the data broadcasting, the *MCMC* models the parallel computation of the Monte-Carlo algorithm. The model assumes that:

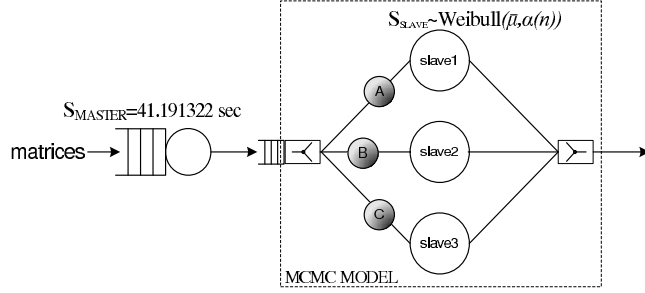


Figure 3: QN model of the matrix inversion algorithm.

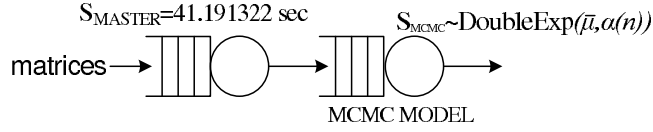


Figure 4: Compact model of the matrix inversion algorithm.

- The queueing discipline is FIFO (First-In First-Out)
- Queues are unbounded
- Communication do not compete for network resources
- Any incoming job waits in the fork queue until the previous one has been completely executed

This model is an accurate approximation of the system for any number of slaves n in the *MCMC* service center. Nonetheless, as n grows, we can simplify the fork-join block by relying on the asymptotic results of Weibull distributions [6]. As a result, when n tends to infinity, the system can be modeled as in Figure 4, where *MCMC* is represented by a single service center with service time equals to $S_{MCMC} \sim DoubleExp(\tilde{\mu}, \alpha(n))$. In the following, we will refer to this model as the *compact model*.

4 Analysis of Simulation Results

In this section, we analyze the results obtained by simulating the system described in the previous section. The service time $S_{master} = 41.191322\text{sec}$. Furthermore, we assume that the inter-arrival time of the incoming matrices are modeled as an exponential distribution with parameter λ .

It is obvious that the bigger the system the higher its capability in tolerate high workloads. This leads to a concept that we refer to as *relative workload*,

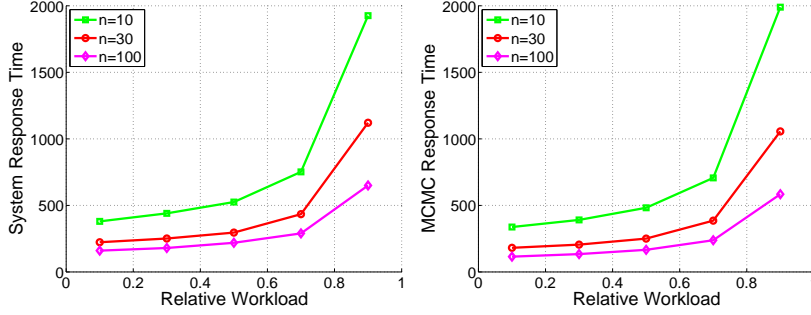


Figure 5: System response time and *MCMC* response time for different relative workloads.

which we express in terms of system utilization U . We indicate with U the maximum utilization among the different service centers and we define the relative workload as the workload which leads to a certain U .

In the following analysis we evaluate the system performance metrics by simulating it. In particular, we want to measure the impact of the main parameters characterizing the system: the number of partitions n , the coefficient of variation γ , the utilization of the system U , and the serial fraction f_s . For each combination of the parameters, we report different measures: queue length, response time, utilization of the service centers, and the system response time. As first, we analyze the performance metrics with different relative workloads. Therefore, we select a suitable relative workload and we vary the other parameters.

Since the queueing network modeling the application does not have a product-form solution, we simulate the network by means of a simulation tool (JMT [1]). In the following, we adopt the compact model when $n \geq 10$.

4.1 Relative Workload

In this simulation we vary the relative workload λ such that the system utilization is $U = 0.1, 0.3, 0.5, 0.7, 0.9$. For different values of U we expect the system performance to follow the classical response time behavior, which is linked to the workload with a non-linear relationship.

As shown in Figure 5, the system response time and the response time of *MCMC* have exactly the same trend. The three lines refer to tests with different level of partitioning. As it can be inferred, the value of U is not critical for the system response time until a utilization of about 50%, while for high workloads the response time increases at a very high rate. Furthermore, the number of slaves n affects only the absolute values of the response time but it does not significantly impact on their trend.

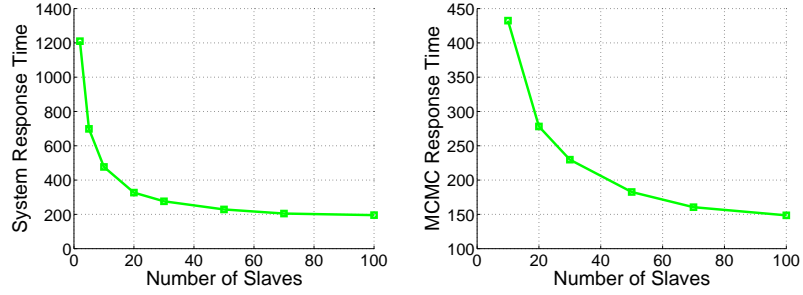


Figure 6: System response time and *MCMC* response time for different number of partitions.

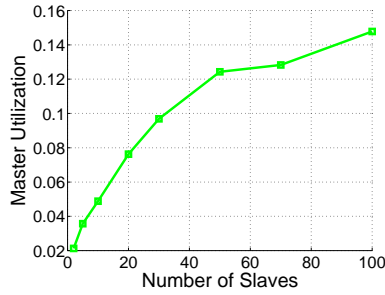


Figure 7: Utilization of the *master* for different number of partitions.

4.2 Number of partitions

Since the application under analysis is nearly embarrassingly parallel, the performance of the system is strictly related to the number n of partitions (i.e., slaves allocated). The more the number of slaves, the more the system benefits from the possibility to run multiple jobs in parallel. Nonetheless, as n becomes very high, the effect of the parallelism becomes less evident, while non-parallel parts of the system gets more and more relevant.

Unlike the previous simulation, in this case we set the system utilization to 40% and accordingly the workload. Figure 6-*left* shows the system response time for different values of n . For instance, the system response time with 40 slaves is about one tenth of the system response time when only 2 slaves are available. On the other hand, by increasing the number of slaves from 40 to 100, the advantage in terms of system response time becomes negligible. As shown in Figure 6-*right*, the decrease in the system response time is mainly due to the reduction of the response time of the *MCMC* service center. In fact, *MCMC* is the only part of the system that can directly benefit from the increase in the number of slaves, while the rest of the system performs serial computations.

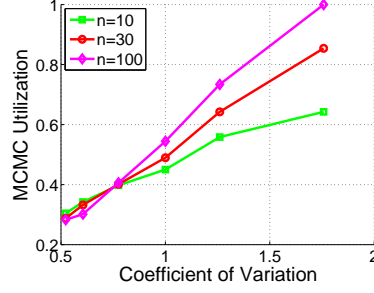


Figure 8: Utilization of the *MCMC* for different coefficients of variation.

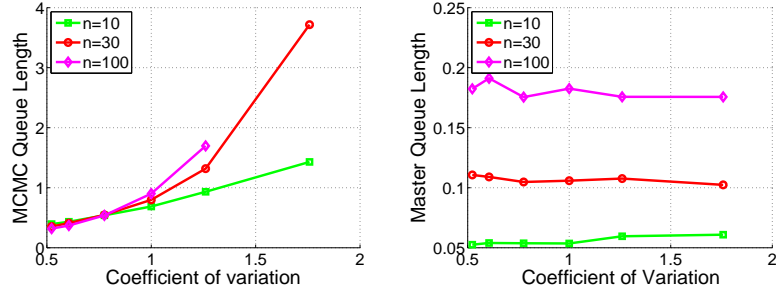


Figure 9: Queue length of the *MCMC* and of the *master* for different coefficients of variation.

As the response time of the *MCMC* decreases, more matrices can be served and, as shown in Figure 7, the utilization of the *master* increases. As a result, the length of the master queue increases as well, but in both cases they do not reach critical values that can represent a bottleneck for the whole system.

4.3 Coefficient of Variation

Another critical parameter for the system performance is the coefficient of variation γ (Equation 5). High values of the coefficient of variation γ indicates that it is more likely for the slaves to have very different completion times. As a result, we expect an increase in the synchronization overhead that causes the performance of the *MCMC* to get worse. Furthermore, we expect this effect to be more and more relevant as the number of slaves increases.

In the simulation we considered values of the coefficient of variation γ from 0.5 to 1.7. In Figure 8, we report the utilization of the *MCMC* service center. While for low values of γ the utilization is low (about 30%) independently from the number of slaves, when γ is high the synchronization overhead becomes

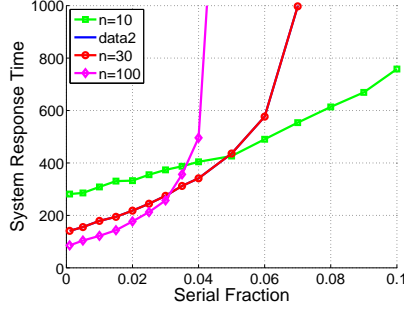


Figure 10: The system response time for different serial fractions.

more and more relevant, so that most of the time is spent waiting for all the tasks to finish and the utilization increases. Furthermore, it can be noticed that when 100 slaves are allocated, the utilization becomes 100% and performance metrics such as the response time and queue length diverge (Figure 9-*left*).

Finally, in Figure 9-*right*, we report the *master* queue length. As it can be noticed, the queue length is constant independently from the value of γ . In fact, in these experiments the system is operating with a constant relative workload.

4.4 Serial Fraction

The serial fraction f_s determines the parallelization capability of an application and it has a significant impact on the system performance. In particular, as f_s increases the application does not take any advantage from being concurrently executed and we expect overhead and partitioning to become more and more relevant in the overall system performance.

Figure 10 shows the system response time for different values of f_s varying from 0.001 to 0.1 and with a number of slaves n from 10 to 100. For low values of f_s there is a good trade-off between the cost of partitioning and overhead, and the advantage of the parallelization of the application. As a result, the system greatly benefits from a huge number of slaves and the system response time for $n = 100$ is about one third of that for $n = 10$. Nonetheless, already for $f_s = 0.03$, the overhead for $n = 100$ becomes relevant and the response time is exactly the same as in a system with only 30 slaves. Finally, for $f_s > 0.05$ there is almost no advantage in parallelizing the application and the system response time for $n = 30$ and $n = 100$ diverges.

5 Conclusions

In this report we analyzed the performance of a parallel application by means of a queueing network model. In particular, we simulated the system in order to evaluate the sensitivity of the performance metrics to different workload

intensities (λ), levels of parallelization (n), application (f_s) and architectural features (γ). The simulation results highlighted that the serial fraction and the variability of the service times significantly affect the system performance, the most important being the response time.

Bibliography

- [1] Java modeling tools. <http://jmt.sourceforge.net>.
- [2] G. M. Amdhal. Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. AFIPS 1967 Spring Joint Computer Conference*, volume 30, pages 483–485, April 1967.
- [3] Francois Baccelli, William A. Massey, and Don Towsley. Acyclic fork-join queuing networks. *J. ACM*, 36(3):615–642, 1989.
- [4] E. G. Coffman and P. J. Denning. *Operating System Theory*. Prentice–Hall, N. J. Henglewood Cliff, 1973.
- [5] Paolo Cremonesi and Roberto Turrin. Performance models for hierarchical grid architectures. In *Proc. of the 7th IEEE/ACM International Conference on Grid Computing*, pages 1–8, Barcelona, Spain, September 2006.
- [6] H. A. David and H. N. Nagaraja. *Order Statistics - Third Edition*. John Wiley & Sons Ltd, 2003.
- [7] Tristan Glatard, Johan Montagnat, and Xavier Pennec. Probabilistic and dynamic optimization of job partitioning on a grid infrastructure. In *PDP '06: Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*, pages 231–238, Washington, DC, USA, 2006. IEEE Computer Society.
- [8] Lorenzo Muttoni, Giuliano Casale, Federico Granata, and Stefano Zanero. Optimal number of nodes for computation in grid environments. In *Proc. of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP04)*, pages 282–289, A Coruna, Spain, February 2004. IEEE Computer Society.

intensities (λ), levels of parallelization (n), application (f_s) and architectural features (γ). The simulation results highlighted that the serial fraction and the variability of the service times significantly affect the system performance, the most important being the response time.

Bibliography

- [1] Java modeling tools. <http://jmt.sourceforge.net>.
- [2] G. M. Amdhal. Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. AFIPS 1967 Spring Joint Computer Conference*, volume 30, pages 483–485, April 1967.
- [3] Francois Baccelli, William A. Massey, and Don Towsley. Acyclic fork-join queuing networks. *J. ACM*, 36(3):615–642, 1989.
- [4] E. G. Coffman and P. J. Denning. *Operating System Theory*. Prentice–Hall, N. J. Henglewood Cliff, 1973.
- [5] Paolo Cremonesi and Roberto Turrin. Performance models for hierarchical grid architectures. In *Proc. of the 7th IEEE/ACM International Conference on Grid Computing*, pages 1–8, Barcelona, Spain, September 2006.
- [6] H. A. David and H. N. Nagaraja. *Order Statistics - Third Edition*. John Wiley & Sons Ltd, 2003.
- [7] Tristan Glatard, Johan Montagnat, and Xavier Pennec. Probabilistic and dynamic optimization of job partitioning on a grid infrastructure. In *PDP '06: Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*, pages 231–238, Washington, DC, USA, 2006. IEEE Computer Society.
- [8] Lorenzo Muttoni, Giuliano Casale, Federico Granata, and Stefano Zanero. Optimal number of nodes for computation in grid environments. In *Proc. of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP04)*, pages 282–289, A Coruna, Spain, February 2004. IEEE Computer Society.

2 – Capacity Planning of Enterprise Systems

2.1 – Capacity Planning of an Hospital Intranet with Multiclass Workload	64
2.2 – Intranet with Web Servers and RAID-0 Storage	74
2.3 – A Network with Finite Capacity Region and Drop Rule	101

Capacity Planning of an Hospital Intranet With a Multiclass Workload

Guido Salvaneschi

guido.salva@gmail.com

Project for the course ‘Computer
Performance Evaluation Techniques and Application’

May 22, 2008

Abstract

The subject of this work is the intranet of an hospital with a 3-tier architecture implemented with Local Area Network. The resulting two class closed model with a fixed number of customers has been analyzed. The values of the more interesting performance indexes are derived. Two different what-if analyses are given: increasing the number of customers and changing the mix of the jobs in execution. We use the following tools of the JMT [1] package: JSIM for simulation, JMVA for exact solutions and what-if analyses, and JABA for bottleneck multiclass identification.

1 Introduction

The paper is organized in the following way. In Section 2 we give a general overview of the context in which the system operates. In Section 3 we describe the technical aspects of the system structure and how they are mapped into our model. We explicitly emphasize the process of realizing a model from a limited slice of reality. Section 4 gives the main performance indices obtained by JMVA and JSIM in the usual working point of the system and provides a first bottleneck identification, while Section 5 is aimed to show the behavior at the increasing customers number. In Section 6 we expose the multiclass analysis, i.e. the changing of performance indices while the ratio between classes changes. We use JSIM for multiclass performance indices analysis and JABA for bottleneck representation in multiclass regime. An introduction in system performance with the fundamental laws partially showed in this paper see [7], while the JMT tool is described in [6] [5] [4].

2 Problem Description

The system under analysis is part of the IT infrastructure of an hospital. The information system is aimed to manage the patient history, e.g. the date of arrive in the hospital, previous diseases, or notes related to his convalescent as daily-measured temperature. We now give a brief idea of the available functionalities.

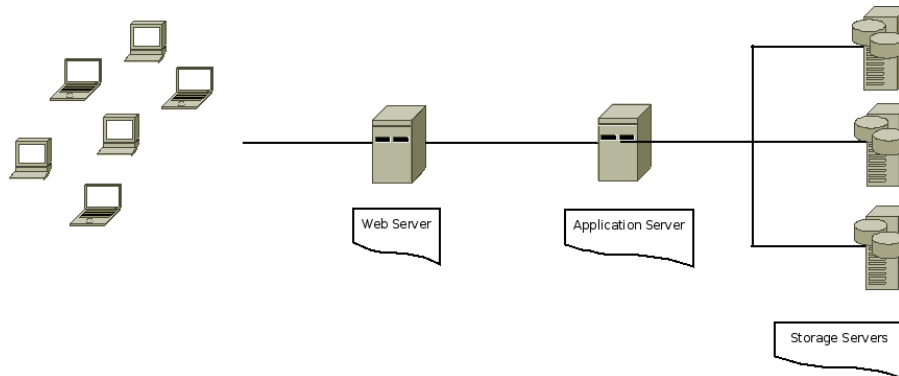


Figure 1: A representation of the system under analysis: a Local Area Network and a 3-tier architecture.

The medicines prescriptions are centrally managed: doctors insert the desired quantity for a certain patient and the pharmacy of the hospital make orders for the whole amount of a needed medicine. All the doctors have a PC in their ambulatory which they use to insert data from visits and prescriptions. The nurses have an instrument similar to a notebook to insert values of temperature, pressure, etc, while they are next to the bed of the patient. They also obtain from the system the quantity of medicine for the patient. Obviously there are functionalities that give the possibility to create a new file for a patient and retrieve a file searching for his name or bed number.

3 Architecture and Model

We suppose that the number of customers remains constant which leads to the choice of a closed model for the representation of our system. The local network has been implemented with a fast 100 Mbit ethernet connection, which let us suppose that the impact of the delay imposed by wireless and wired communications is negligible due to the speed guaranteed by the infrastructure.

Customers wait in mean a few seconds after submitting a new request to the system, which results in a delay station in our model. We suppose that the delay between HTTP requests is 1 second, supposing that several elements are present in a browser page. The interaction of the customers with the system is provided by a web application with a browser on client side and a typical 3-tier architecture on server side. The requests are submitted to a web server; in case of a static page the HTTP response is immediately passed back to the client, otherwise the web server interacts with an application server that performs some queries on a backend database and processed data are passed back to the web server. Finally clients receive the dynamically-generated page.

The storage consists of 3 database servers in parallel. The attribution of a job to one of the storage servers is done by a load-balancer in a random way. The distribution of the requests among the three servers is uniform. We suppose negligible the delay introduced by the load-balancer in the reasonable hypothesis of a minimal processing inside the load-balancer.

	<i>LightLoad</i>	<i>HeavyLoad</i>
<i>Web Server</i>	1.40	1.10
<i>App Server</i>	2.10	1.50
<i>Storage 1</i>	1.10	2.90
<i>Storage 2</i>	1.20	2.70
<i>Storage 3</i>	1.10	2.80

Figure 2: Service demands in milliseconds for each station in the system. The different values of the two classes are due to the different behavior of the two types of users.

We subdivide the requests arriving at the system in two groups: the requests for a database search and the requests for a database modification. The two groups are modelled by classes with different service time for each station. Considering the point of view of the database we refer to the search-class and the modification-class as the *HeavyLoad* class and the *LightLoad* classes. In fig. 2 are reported the service demands for each station in the system and for each class.

4 Working condition analysis

In this section we present the analysis of the system in its usual load condition. We estimate that 1000 customers are present in the system. The ratio between the number of *HeavyLoad* requests and the *LightLoad* requests is about of 3/7, which results in 300 jobs of *LightLoad* and 700 jobs of *HeavyLoad*.

The analysis of the working condition can be performed either with JSIM through a simulation or with JMVA which is an exact solver. Due to the simplicity of our model we chose the second option (the first implies a long-running process and approximate results in a confidence interval; in our tries with JSIM similar results to JMVA has been found).

In fig. 3 we present the results of JMVA analysis. Note that the values of the utilization show that the bottleneck in the usual workload is the application server. In fact the value of the utilization of that station is 1 indicating the complete saturation of the server.

5 What if Analysis for Scalability

In this section we provide an analysis of the system behavior while the number of customers increases. We increase the absolute number of customers from 10 to 1000 while keeping the mix of jobs, i.e. the percentage of the jobs of each class, constant.

In fig. 5 we plotted the values of the system throughput in function of the number of customers in the system. The saturation is reached at about 600 customers. In fig. 6 there is the plot of the system response time. The figure shows the linear rising of the response time after an initial phase. The linear gain could be easily forecast by the well-known asymptotic law:

$$R(N) \geq \max(D, ND_{max} - Z)$$

System Throughput	0.52519
System Resp. Time	1904.06403
	Utilization
<i>Web Server</i>	0.68381
<i>App Server</i>	1.00000
<i>Storage 1</i>	0.88749
<i>Storage 2</i>	0.86927
<i>Storage 3</i>	0.88642

Figure 3: JMVA exact analysis with the following parameters: 1000 customers, 700 *LightLoad*, 300 *HeavyLoad*, delay of 1 sec. and service demands in fig. 2. Values are in milliseconds

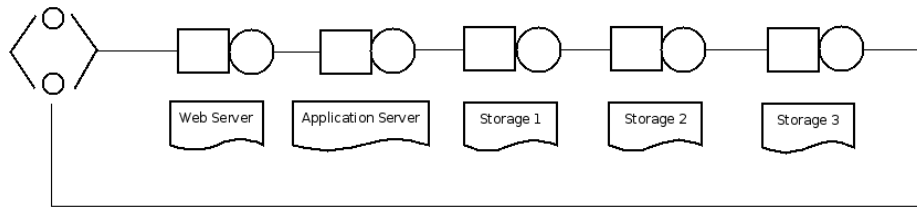


Figure 4: The system model of the JSIM. Note that we are allowed to use this topology since we consider the service demands ($D_i = S_i V_i$) for each resource and not the visits V_i and the service times S_i as parameters of the model.

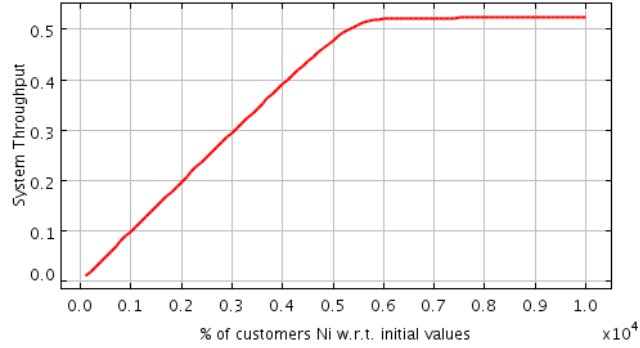


Figure 5: Global system throughput in function of the number of customers. The saturation is reached at about 600 customers.

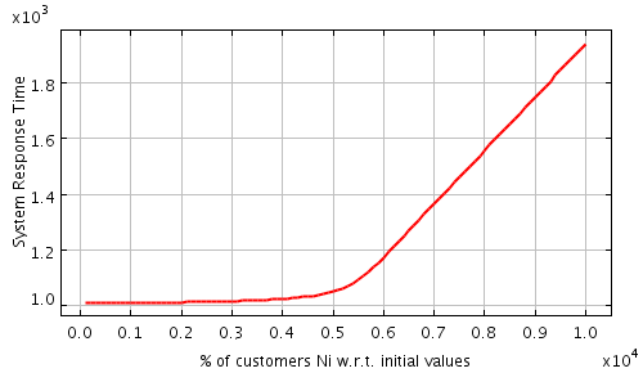


Figure 6: Global response time in function of the number of customers in the system keeping constant the mix of the two classes. Accordingly to the theory for high values of the num. of customers, the response time grows linearly.

where D is the service demand of the whole system, N is the number of customers and Z is the think time, i.e. the interval they wait in mean.

The presence of a bottleneck means that the corresponding station works at his best and the load of the other stations remains limited also while the number of customers increases. In fig. 7 we plot the values of utilization in function of the growing customers. It is evident that the application server reaches saturation as first and constitutes the bottleneck for the whole system. This implies that after the saturation of the application server (at about 600 jobs), the throughput of the system remains constant (see the system behavior in fig. 5).

6 Multiclass Analysis

In this section we provide an analysis of the system behavior with all the possible mixes of jobs. First of all we use JMVA in order to study the behavior of the

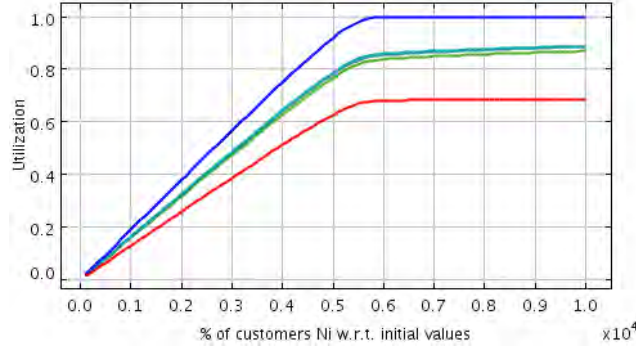


Figure 7: Utilization of each station in function of the rising number of customers in the system. The upper line refers to the application server, the lower line refers to the web server and the other three lines represents the storage servers. Accordingly to the theory the utilization time has an horizontal asymptote.

system while the ratio between the elements of the two classes changes. Than we give some diagrams obtained by JABA which are useful for bottleneck detection in multiclass environment.

6.1 Multiclass with JMVA

We have performed a what-if analysis with the aim to understand the behavior of the system under study with the change of the ratio between the two classes. The goal of this type of work is to establish the ideal mix under which to guarantee the best performance.

In fig. 8 we represent the throughput of the system. The plot shows an initial rising while the *HeavyLoad* increases; in the interval between 0.4 and 0.5 it presents a plateau in which the throughput is maximum and constant. After 0.5 the performance have a degradation while the percentage of *HeavyLoad* strongly increases. We can conclude that the optimal mix between *HeavyLoad* class and *LightLoad* class is in the interval 0.4 - 0.5, which is the range that maximize the throughput.

In the plot in fig. 9 we show the response time of the system in function of the ratio between classes. The behavior of this function can be recognized to be the dual of the global throughput of fig. 8: a progressive reduction of the response time, a plateau in the interval 0.4 - 0.5 and a following rising are evident.

A further analysis gives a more precise reason of the behavior of the system. In particular it explains the presence of a plateau with maximum performance. In fig. 10 there are the utilizations of each station in the system while classes ratio changes. In the plot the utilization of the application server starts at the value of 1, i.e. the application server is bottleneck when the amount of *HeavyLoad* class is low. The utilization of the storage for low values of the class *HeavyLoad* is about 0.4 - 0.5 and rises while the ratio increases, reaching 1 for the ratio value 0.4. This results in the plateau of above: at 0.4 the storage server reaches the saturation while the application server *still is in saturation*. After 0.5 the

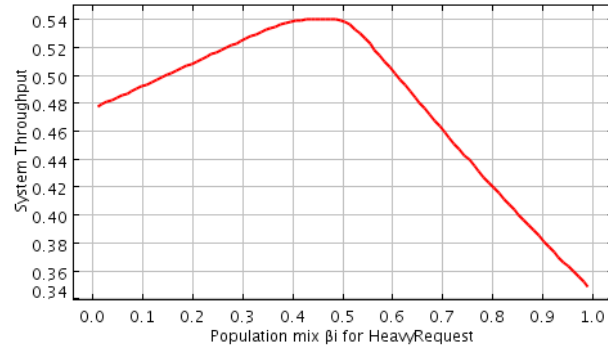


Figure 8: System throughput as a function of the ratio between HeavyRequest and LightRequest.

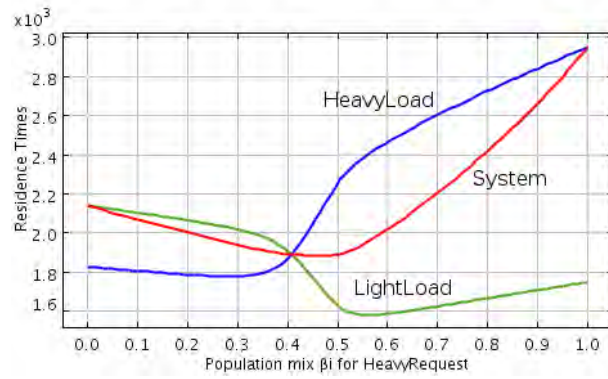


Figure 9: The system response time as a function of the ratio (mix) between HeavyRequest and LightRequest. The line starting from 1.8 represents the response time for the *HeavyLoad* class, while the other line is referred to the *LightLoad* class. Values are in milliseconds.

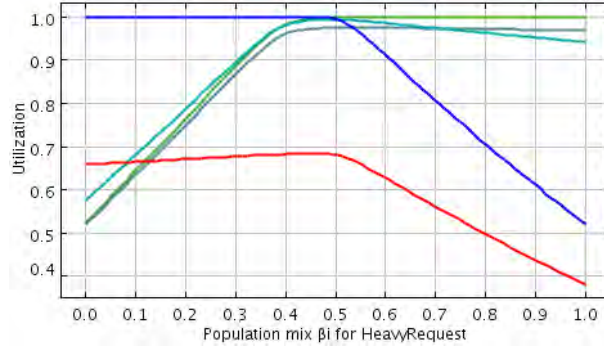


Figure 10: Values of the utilization for the stations of the system as a function of the ratio *HeavyLoad* and *LightLoad*. The line starting at value 1 (100% utilization) represents the application server, the lower line is the web server, and the three overlapping lines are the three storage servers.

utilization of the application server begin to lower and the only bottleneck is the storage server. We evidence that the plateau is indeed an interval of common saturation which means that *two bottlenecks are present in the system at the same time*. This area is the interval of major stress for the servers and this results in the highest throughput showed in fig. 8.

6.2 Multiclass asymptotic analysis with JABA

JABA is a tool specifically designed for the bottleneck identification in multi-class closed systems. In fig. 11 we show the plot of the saturation sectors. In the graph are represented three condition of work of the system. For light load of *HeavyLoad* class the bottleneck turns out to be the application server, while for light load of the *HeavyLoad* class (i.e. heavy load of *LightLoad* class) the bottleneck is the storage server.

The graph provides an additional information regarding the fact that there is an interval of common saturation in which *both the application end the storage server are bottlenecks*. We now give a more rigorous explanation of these phenomena. A *natural bottleneck* of a system is the station whose utilization tends to one when the number of customers in the network grows to infinity, given that the network is used by class- r customers only. The *bottleneck migration* is the fact that the bottleneck of the system can migrate among the stations when the population mix changes. Our study has identified *distinct* natural bottlenecks for different classes. This means that keeping the total population of the network constant (and high) and varying the population mix, we may observe the bottleneck migration phenomenon. In particular, fig. 11 can be interpreted in terms of *saturation sectors* i.e. regions on the line where two stations saturate simultaneously, and *switching points*, where stations change their bottleneck/non-bottleneck status. A more general survey of these topics an an accurate mathematical background is in [3] and [2].

The second type of plot retrieved by JABA is the saturation hall graph shown

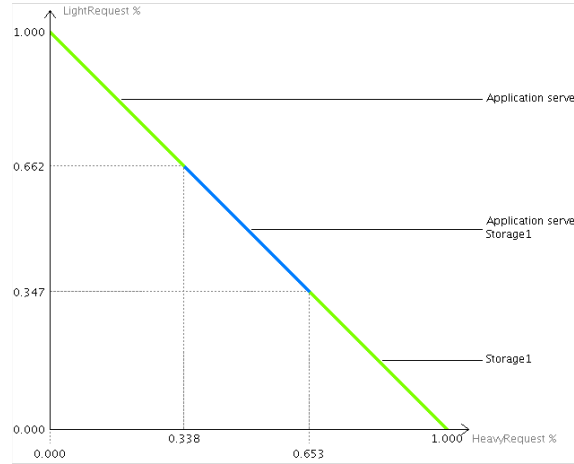


Figure 11: The plot of the saturation sectors. In the central part of the line there is the area of common saturation.

in fig. 12. On the border of the polygon there are the two possible bottlenecks, the application server and the storage. Being in the internal part of the polygon, the web server is never the bottleneck, i.e. for any class mix.

6.3 Conclusions

The performance of an hospital intranet with a workload consisting of two classes of customers has been analyzed. Asymptotic values of performance indices are obtained as a function of the mix of jobs of the two classes in concurrent execution.

Optimal operational condition of the system, i.e. the mix of customers that provides the maximum throughput and correspondingly the minimum response time is between the interval 0.4 - 0.5 as shown in figures 8 - 9 and 10. This optimal operation condition is a subset of the common saturation sector identified in fig. 11 where either the application server and the storage saturate together.

References

- [1] <http://jmt.sourceforge.net>.
- [2] Gianfranco Balbo and Giuseppe Serazzi. Asymptotic analysis of multiclass closed queueing networks: common bottleneck. *Perform. Eval.*, 26(1):51–72, 1996.

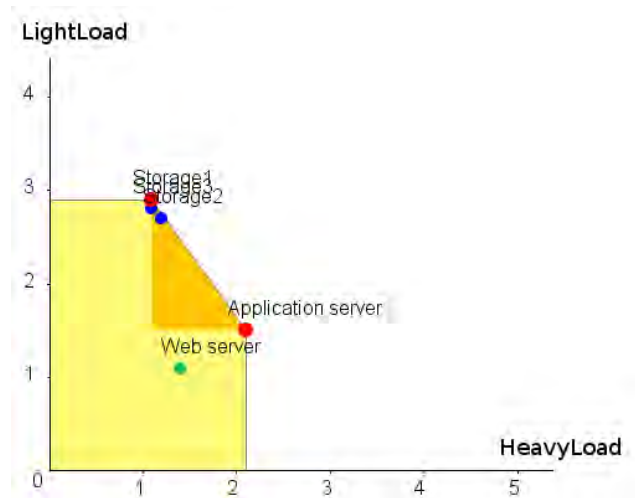


Figure 12: The plot of the convex hull. The two possible bottlenecks, the application server and the storage, are on the border of the polygon. The dark part of the polygon represents the masked-off area, while the light part is the dominated area.

- [3] Gianfranco Balbo and Giuseppe Serazzi. Asymptotic analysis of multiclass closed queueing networks: multiple bottlenecks. *Perform. Eval.*, 30(3):115–152, 1997.
- [4] M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proceedings of QEST 2006 Conference*, pages 119–120, Riverside, US, Sep 2006. IEEE Press.
- [5] M. Bertoli, G. Casale, and G. Serazzi. An overview of the jmt queueing network simulator. Technical Report TR 2007.2, Politecnico di Milano - DEI, 2007.
- [6] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. The jmt simulator for performance evaluation of non-product-form queueing networks. In *Annual Simulation Symposium*, pages 3–10, Norfolk, VA, US, 2007. IEEE Computer Society.
- [7] G. Scott Graham Kenneth C. Sevcik Edward D. Lazowska, John Zahorjan. *Quantitative System Performance Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.

Luca De Fulgentis
708268



Intranet with Web Servers and RAID-0 storage

February 2008

Project for the course :

"Performance Evaluation : Techniques and Models"
Prof. G. Serazzi

Contents

1 Introduction	3
2 Problem Definition	4
3 Model Description	5
4 Model Simulation	9
4.1 JSIMGraph Simulation	9
4.2 What-If Analysys	12
4.2.1 Increment of the Visitors Arrival Rate	13
4.2.2 Increment of the Players Arrival Rate	21

Section 1

Introduction

This paper discusses the modelling and analysis of an entertainment system based on a cluster of web servers and a storage server consisting of RAID (type 0, striping mode) temporary data storage. The entertainment system modelled is characterized by two classes of users : the first class is represented by users that navigate the system web pages while the second class describes users gaming (fast online game) utilizing the system as temporary data repository (i.e., a client will be installed on the user station when the game starts and typically submits a sequence of web request to save temporary game status). We simulated and evaluated the model performance using JMT (*Java Modelling Tools*) version 0.7.3.

Section 2

Problem Definition

The system considered in the study is composed by two web servers running *Apache* and a *RAID-0* storage server used to store temporary data. The system receives requests from two classes of users. The first class (referred to as *Visitors*) represents users that navigate system web pages, while the second class represents users that store temporary data through web server on a storage server (referred to as *Players*, i.e. users that are playing a game). In Figure 1.1 we can see a high-level system representation. Requests arriving from internet visit first a *Load Balancer*, a traffic balancer used to distribute the traffic between two web servers. The purpose of the load balancer is to increase web server scalability. The objective is to handle high volumes of incoming traffic by using a cluster of web servers behind the load balancer itself.

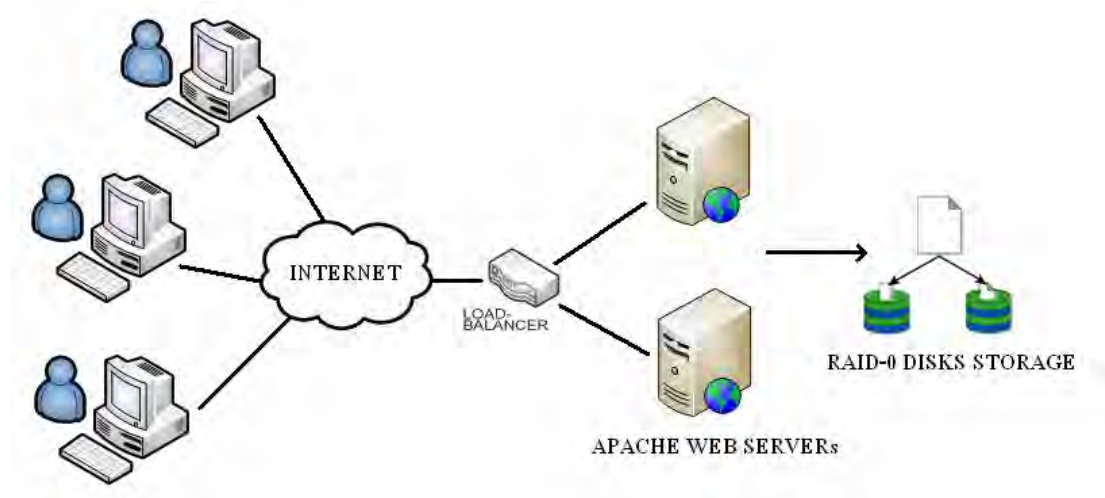


Figure 1.1 – System to model

The two web servers use Apache as *HTTP Daemon*, and we established a limit on the number of users (including both players and visitors) in concurrent execution that is set to 30. This limitation has been set in order to guarantee an acceptable level of performance, i.e. to avoid the congestion of the resources. The RAID-0 (Redundant Array of Independent Disks) storage system is a set of disks that in our specific example are used in RAID-0 mode : this means that a block of data (temporary data) is striped between two or more disks. Because of this, data is segmented in *strip unit* (i.e., a fix length data unit) that are then distributed on various disks (in our example 3 disks). RAID-0 configuration ensures high performance but has a very low *reliability* (the probability of the failure of a disk is equal to $1/\text{number of disks}$) due to lack of data redundancy. We decide to use the RAID-0 model because of its high level of performance and because this is a critical factor in entertainment systems.

Section 3

Model Description

In this section we will describe the architecture of the model developed. We used the tool *JSIMgraph* to represent and solve this model (see Figure 2.1). We modelled our system as an *open model*. A job enter the model, it visit several service stations, and then exit the model. An important characteristic of this model is that a job should *not* be understood as a request for a single resource but as a complete user interaction with the system. Because of this we can see a job as a *user* that cycles through system's resources. We used several *feedbacks* to represent the user behaviour of looping inside model stations during its interaction (i.e., during the execution of a game or during a navigation session). A first feedback was introduced between the web servers and the *Delay_1* element to simulate that most of the time (70% in our model) user requests more than a single web page during its navigation. We also introduced a feedback between the *Join_Storage* (at the end of the storage system) and web servers' *Load Balancer*. This feedback connection represents the cycle followed by users that to submit continuously data while they playing online game. Even in this case most of the jobs (96%) return back to *Load Balancer* element..

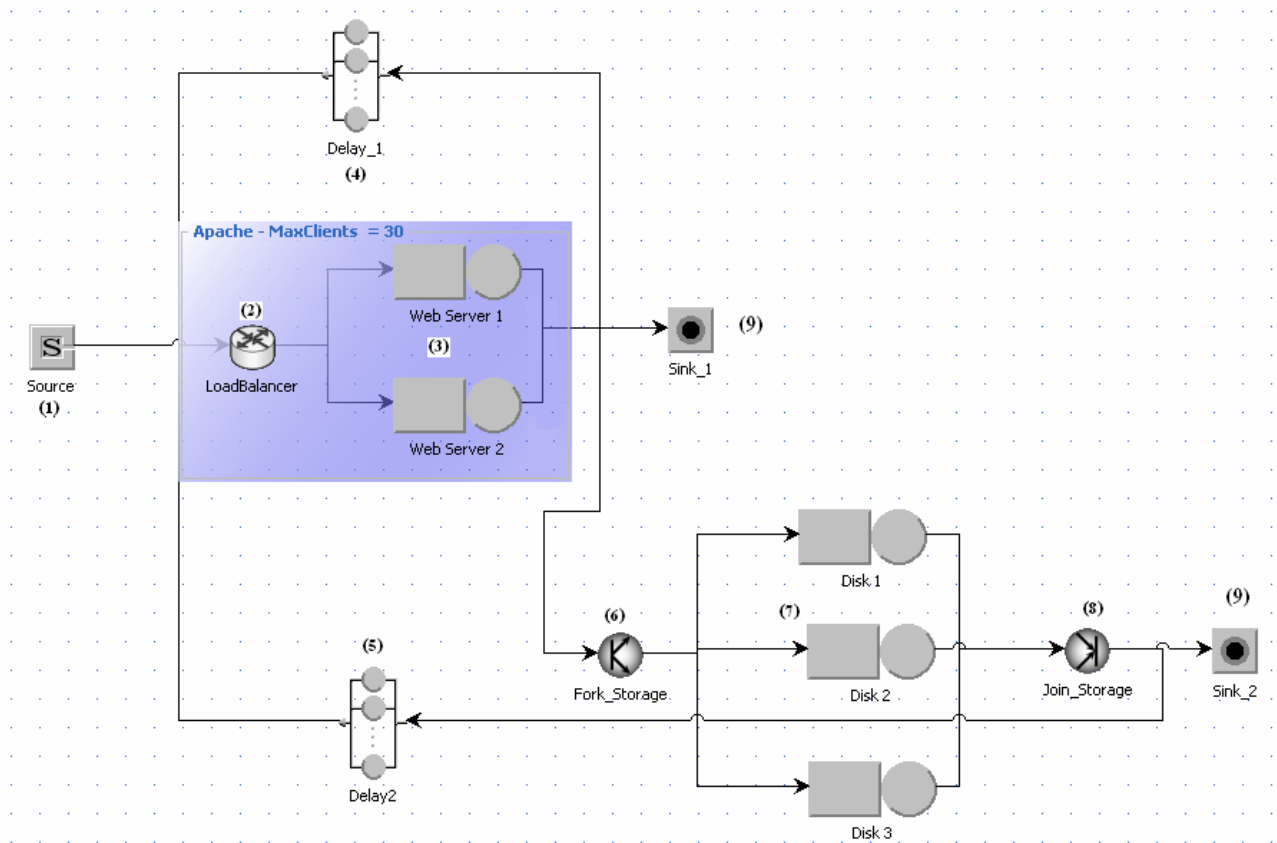


Figure 2.1 – System Model

Tables 2.1 and 2.2 shows job class routing probabilities.

From/To	Visitors Class		
	Sink_1	Delay	Fork_Storage
Web Server 1	0.3	0.7	0
Web Server 2	0.3	0.7	0

Table 2.1 – Class *Visitors* Routing Probabilities

From/To	Players Class			
	Sink_1	Load Balancer	Fork_Storage	Sink_2
Web Server 1	0	0	1	0
Web Server 2	0	0	1	0
Join_Storage	---	0.96	---	0.04

Table 2.2 – Class *Players* Routing Probabilities

Another important characteristic of the model is the workload characterization. We defined two classes of users (jobs) :

- *Visitors* : represent a system interaction where a user navigates system web pages, but this operation doesn't involve the RAID system. Because of this a job belonging to this class will spend all of its time in the *Web Server* (and delay) stations;
- *Players* : *Players* jobs represent a user system interaction that involves the RAID System. In this case jobs remain most of their time in the storage stations than in web server stations.

We model the Visitors/Players traffic with an exponential distribution of interarrival times with arrival rate : see Table 2.3

Class Arrival Rate	jobs/seconds	users/day
λ VISITORS	1	86400
λ PLAYERS	2	172800

Table 2.3 – Job Class Arrival Rates

A detailed description of model elements follows :

- **Source (1)** : This element is used to generate customers for the open classes. We need of this element because open classes are characterized by an infinite stream of jobs (here web/storage request as mentioned above). We use the traffic generator to introduce jobs in the model;
- **Load Balancer (2)** : The load balancer mentioned in the introduction section is represented by a *routing station* that distributes the arriving jobs to the connected output stations (two web servers) according to a routing strategy. We established that *Load Balancer* distributes 50% of the job's population on the first web server and 50% on the second one, regardless of job class;
- **Web Server 1 & 2 (3)** : Apache web servers are modelled with two same *Service Stations*. The queue policy used for both web stations is First Come First Server (FCFS). Under this queuing discipline customers are served in the order in which they arrive at the station. Another system's important characteristic is that a service station takes different time to serve a job depending on the job's class. The meaning of this behaviour is that a *Visitor* must be processed in a longer time respect to a *Player* because the second one will not request any computation, but must only pass through the station. Details about web servers' Service Time are reported on Table 2.5.

We used two different (but very close) values for the web servers' service time because we wanted to simulate two realistic web servers which, while having the same hardware, have obviously different performance.

The two web servers are situated inside a *Finite Capacity Region* (FCR). A FCR is a region of the model where the number of customers is controlled. The maximum number of users connected concurrently is limited in order to guarantee an acceptable *Quality of Service* (QoS) in terms of performance : we fixed a maximum users number equal to 30. We used a *Shared FCR* because we set an upper bound for the number of customers that are in the region, regardless of the classes they belongs to.

- **Delay_1 (4) Delay_2 (5)** : Customers that arrive at this station are delayed for the amount of time that defines the station service time. We used the *Delay_1* Station to simulate user *think-time*, and so the average time that a user spends to read a web page. We defined it as a high value because we also would to represent the networking infrastructure delay. *Delay_2* Station was used to simulate the periodically sending of game status from players game client . Table 2.4 shows delay station service times.

	Visitors Job Class [sec.]	Players Job Class [sec.]
Delay Station Service Time	20	10

Table 2.4 – Delay Station Service Time

- **RAID Sub-system (6) (7) (8)** : The RAID subsystem models the usage of three disks organized in RAID-0 mode. When a job arrives at the fork element called *Fork_Storage* (6), a fixed number of tasks are generated on its forward link. The three new tasks corresponding to the original job are then processed by three different service stations called *Disk 1/2/3* (details in Table 2.5). A *Disk* element represents a single disk of the RAID system. The fork element is used to simulate the striping process that involves the data to write.

Such as Web Server service time, we used three different but very close values for disks' service time because we would simulate a RAID disks configuration where we have the same disks but that obviously have small performance differences. In our model, disks' service time represents time needed to store a single strip unit on disks. When the three tasks are ended, the model ensures that the three jobs collapse in the original one by using the *Join_Storage* (8) element. The join element waits for the complete execution of all the tasks of the fork and recreates the original job back;

- **Sink_1 and Sink_2 (9)** : Two *Sink Station* were used to model customers leaving the system. The first element only involves *Visitors* jobs while the second one only *Players* jobs.

Service Station	Visitors Job Class [sec.]	Players Job Class [sec.]
Disk 1	---	0.006
Disk 2	---	0.0058
Disk 3	---	0.0062
Web Server 1	0.200	0.015
Web Server 2	0.198	0.018

Table 2.5 – Stations Service Time

A note on Table 2.5 : Disks 1/2/3 Service Time values for *Visitors* jobs are not indicated because these jobs don't utilize disk stations.

Section 4

Model Simulation

4.1 JSIMGraph Simulation

In this Section we present the model simulation results obtained using *JSIMGraph* tool. The results refer to the parametric values reported in Section 3. Model simulation results follow :

- *Utilization :*

Table 4.1.1 shows simulation results about web server's utilization :

	Visitors Class			Players Class			All Classes		
	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
WS 1	0.293	0.311	0.329	0.181	0.202	0.222	0.596	0.624	0.652
WS 2	0.289	0.299	0.308	0.261	0.285	0.308	0.567	0.608	0.649
D1	---	---	---	0.641	0.658	0.674	---	---	---
D2				0.643	0.660	0.677			
D3				0.678	0.691	0.703			

Table 4.1.1 –Utilization for System Resources

Table 4.1.1 shows utilization results for system resources. As we can see there are no value for Visitor Class disk utilization, because these jobs don't use these resources.

- *System Throughput :*

System throughput simulation tells how many jobs (users) leaves the system per second :

	Visitors Class			Players Class			All Classes		
	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
X	0.941	0.972	1.006	1.483	1.516	1.550	2.387	2.471	2.562

Table 4.1.2 – System Throughput

Table 4.1.2 shows that not all jobs that enter the system, exit it. This is because part of the Players jobs (about 0.5 jobs/sec, referred to as $\lambda_{PLAYERS} = 2$ jobs/sec.) are dropped by system because of the FCR control.

- **Response Time :**

With this simulation we want to know the average time a job/user spends in the system, both if it is referred to as Visitors or Players users. Because a job exits the system only when the user terminates the interaction, the response time measures the session duration. Simulations results show that Visitors jobs navigate for an average time of 40 seconds, while Players jobs play online game for an average time of about 180 seconds (~3 minutes) (See Table 4.1.3).

	Visitors Class [sec.]			Players Class [sec.]		
	Min	Avg.	Max	Min	Avg.	Max
R	42.272	43.973	45.674	174.323	180.894	187.465

Table 4.1.3 – Global System Response Time

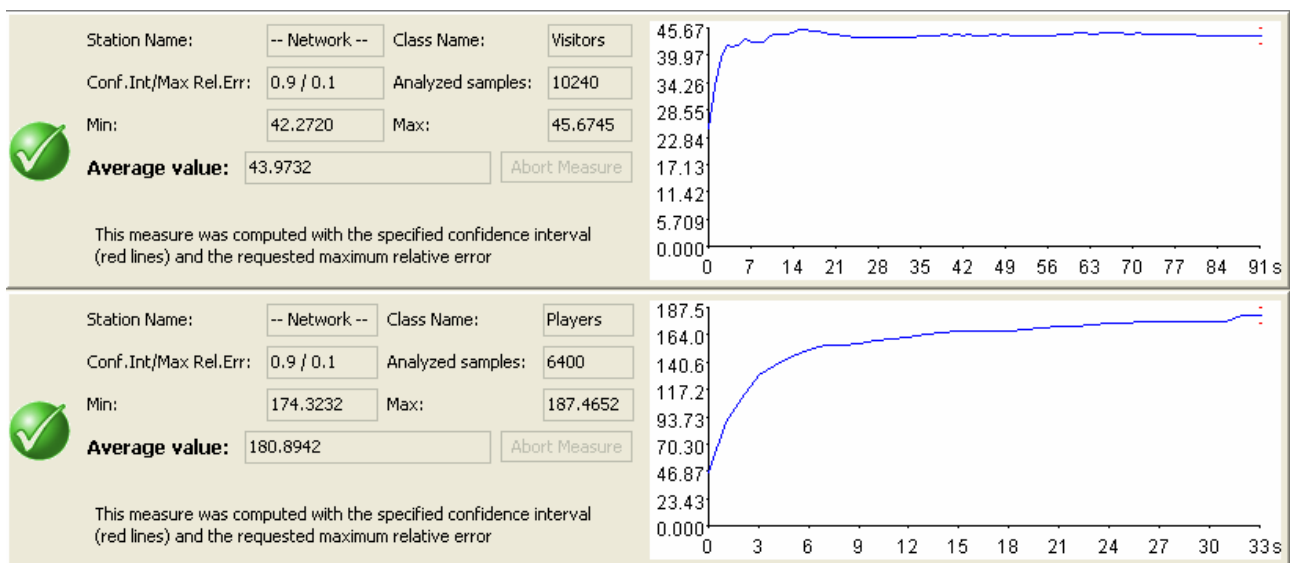


Figure 4.1.1 – System Response Time

- **Single Station Response Time :**

This index refers to the time a user of type Visitors is forced to wait for the download a complete web page. Table 4.1.4 shows an average time of about 0.36 seconds.

	Visitors Class		
	Min	Avg.	Max
Web Server 1	0.346	0.368	0.389
Web Server 2	0.347	0.367	0.387

Table 4.1.4 – Single Web Server Response Time

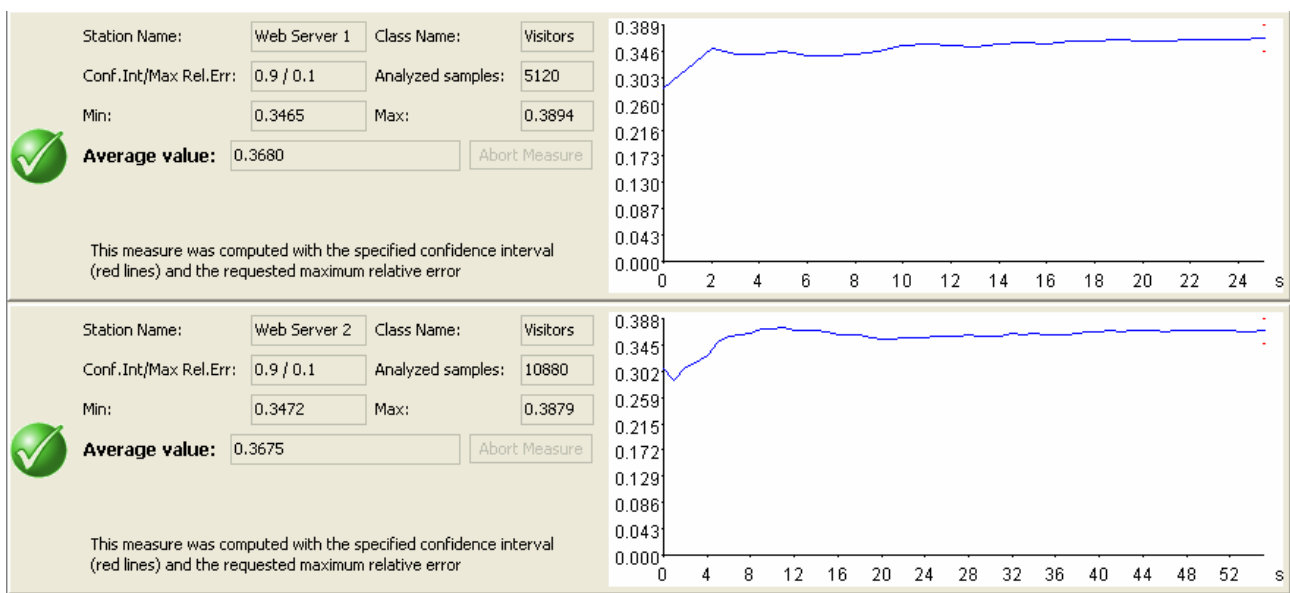


Figure 4.1.2 – Single Web Servers Response Time

- FCR Drop Rate :**

Drop Rate simulation shows how many users are dropped by the control of the FCR. Table 4.1.5 shows these values :

	Visitors Class			Players Class		
	Min	Avg.	Max	Min	Avg.	Max
Drop	---	0.044	---	0.499	0.521	0.546

Table 4.1.5 – FCR Drop Rate

Table 4.1.5 shows that the large part of the dropped jobs belongs to Players class.

4.2 What-If Analysis

A *What-If Analysis* consists of a series of simulations in which one or more parameters are varied over a specified range. This allows the observation of system behaviour under a spectrum of conditions, unlike the single *JSIMGraph* simulation run where the system is observed under a specific set of configuration parameters. During our study we varied the value of the job's arrival rates and we studied the performance of our model.

Table 4.2.1 shows Players/Visitors jobs arrival rate increments realized during the what-if analysis :

<i>Class Arrival Rate</i>	<i>jobs/seconds</i>	<i>new jobs/sec</i>	<i>new users/day</i>	<i>% increment</i>
$\lambda_{\text{VISITORS}}$	1	2	172800	+ 100%
λ_{PLAYERS}	2	4	345600	+ 100%

Table 4.2.1 – Arrival Rate Increment of the two classes of users

Section 4.2.1 will present Players jobs class arrival rate increment and section 4.2.2 the Visitors one.

4.2.1 Increment of the Visitors Arrival Rate

This subsection will presents What-If analysis results when the Visitors arrival is incremented by 100%. This simulation permits to study the system performance variation with an increment of visitors that correspond to an increment of users during week-ends.

- *Web Servers Utilization :*

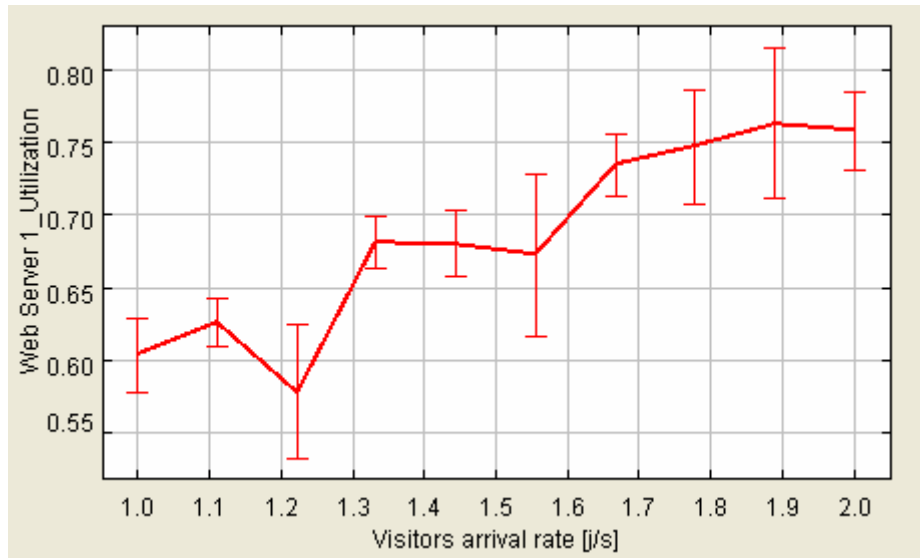


Figure 4.2.1.1 - Web Server 1 Utilization

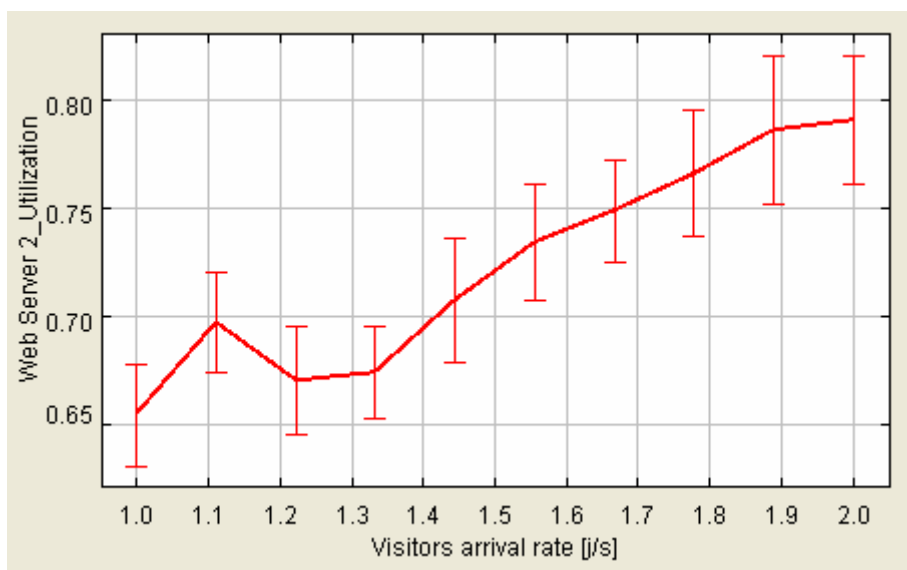


Figure 4.2.1.2 – Web Server 2 Utilization

Figures 4.2.1.1 and 4.2.1.2 show how the web servers utilization is increased of an average value of about 15% with the Visitors arrival rate increment. We will better investigate the impact of this utilization increment studying the single web server response time.

- *Single Web Server Response Time for Visitors Job Class :*

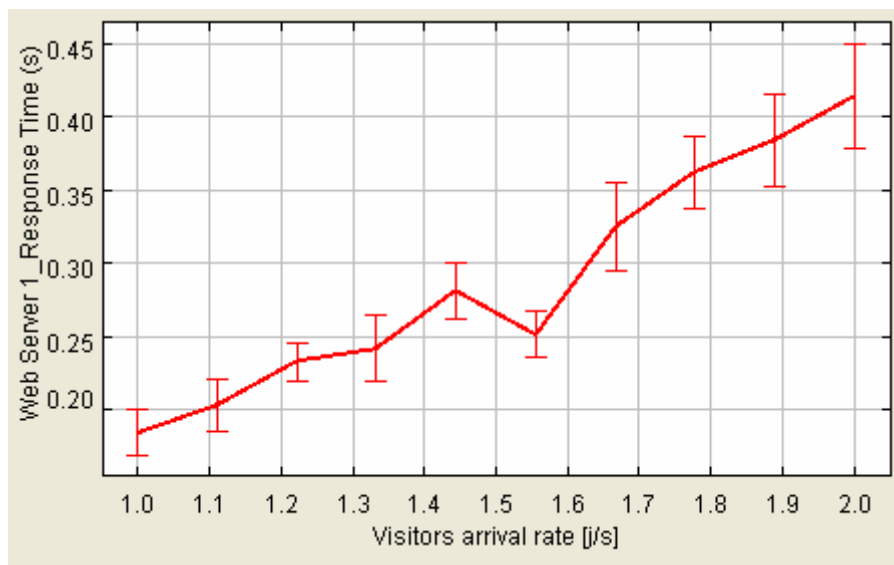


Figure 4.2.1.3 – Web Server 1 Response Time

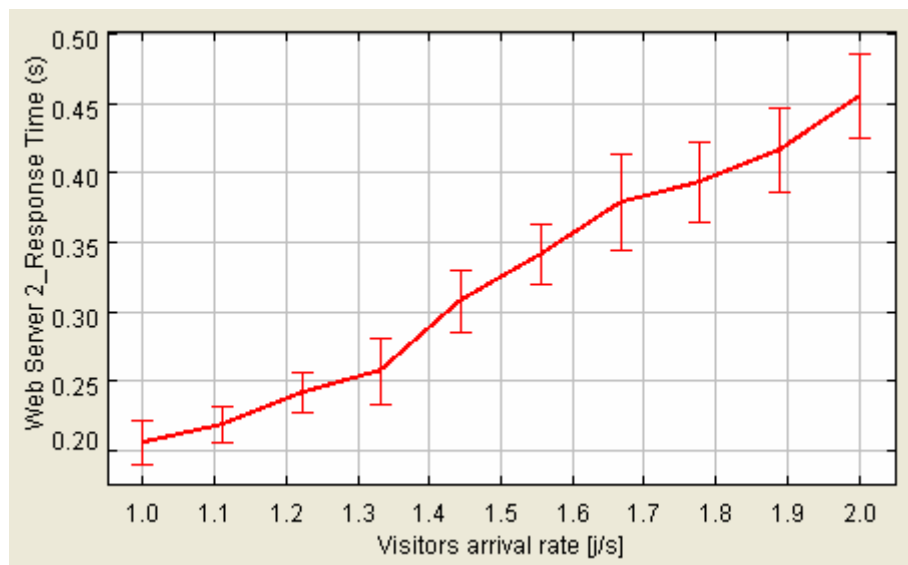


Figure 4.2.1.4 – Web Server 2 Response Time

Figures 4.2.1.3 and 4.2.1.4 show how the increment of the 15% of the web server utilization causes a performance worsening resulting in a doubled web server response time. This means that users are forced to longer wait for the download of an entire web page. However the new response time (about 0.45 seconds) is still an acceptable waiting time value.

- ***Storage Disks Utilization :***

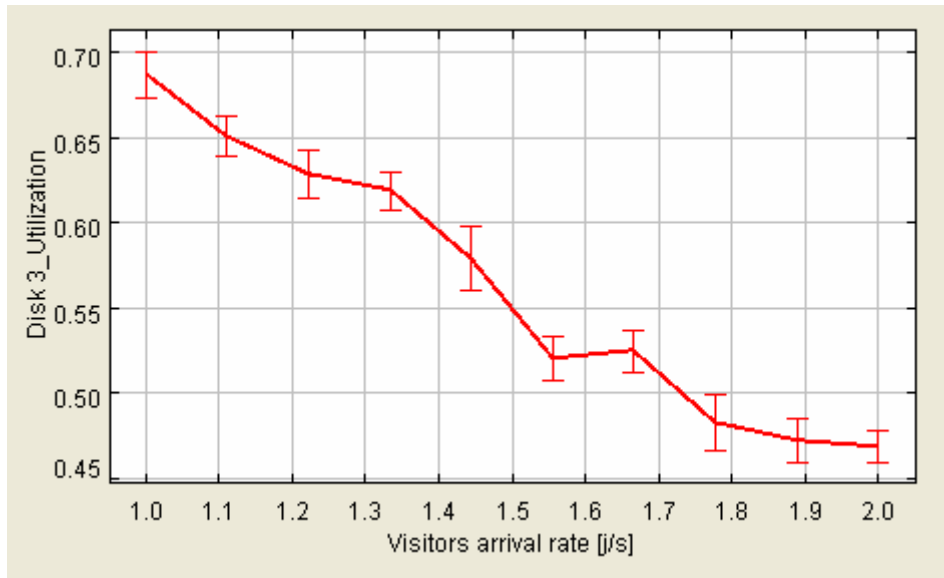


Figure 4.2.1.5 – Disk 3 Utilization : Disk 1 and 2 have similar behaviour

If we consider an initial average utilization of 60-70% for all the three disks, we can see that the Visitors jobs increment causes a reduction of the utilization of the disks that is about 45%. This situation can be explained by the higher number of Visitors jobs (we incremented Visitors arrival rate) which spend more time in the web server than Players ones. Because of this Players jobs are delayed and this causes two effects : first of all Players jobs have an higher probability to be dropped in the FCR (because of their accumulation, will be demonstrated with the FCR jobs mix graph) and then a lower number of Players exit FCR and enter the storage servers resulting in disks utilization reduction.

- **Finite Capacity Region Drop Rate :**

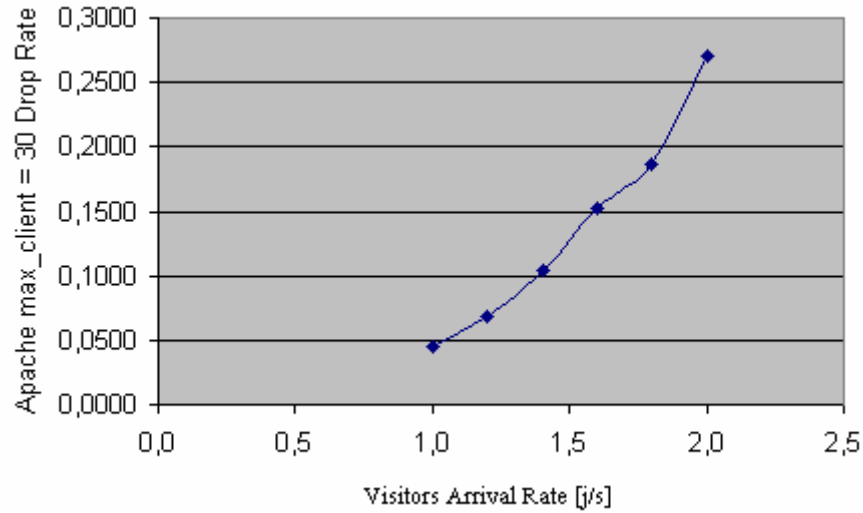


Figure 4.2.1.6 – Class *Visitors* FCR Drop Rate

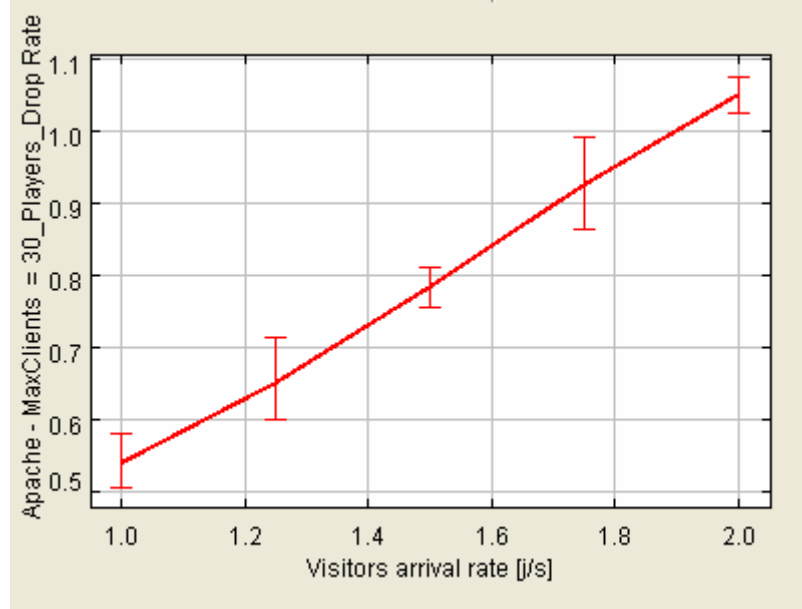


Figure 4.2.1.7 – Class *Players* FCR Drop Rate

Figures 4.2.1.6, 4.2.1.7 show that the most dropped jobs belong to Players class. This justifies the disks utilization reduction shown in Figures 4.2.1.5. We can also see that even in the worst situation almost all Visitors jobs are served by web servers.

- *System Throughput* :



Figure 4.2.1.8 – Class *Visitors* System Throughput

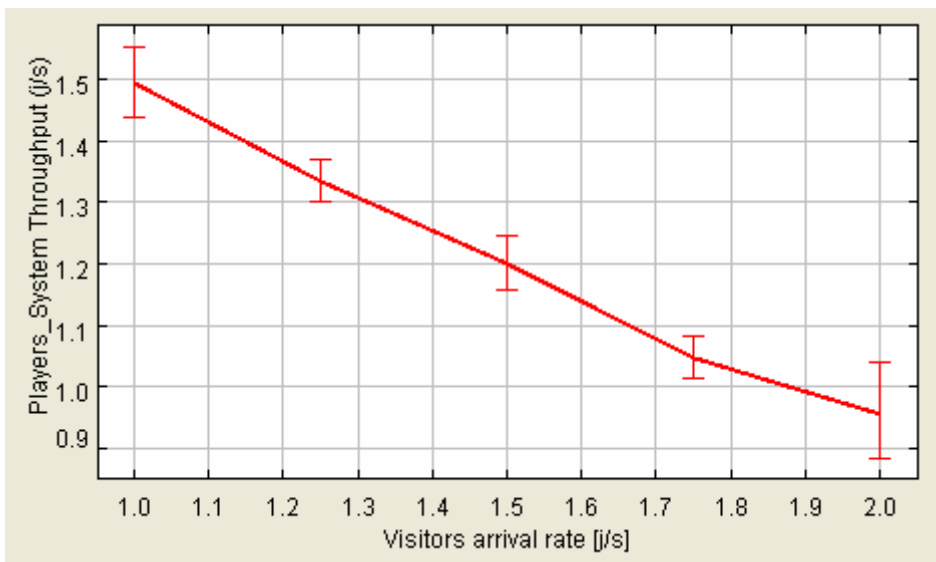


Figure 4.2.1.9 – Class *Players* System Throughput

Because of the high number of dropped Players in FCR, the system throughput for this class slows decreasing behaviour. See Figure 4.2.1.9.

- **Single Disk Response Time :**

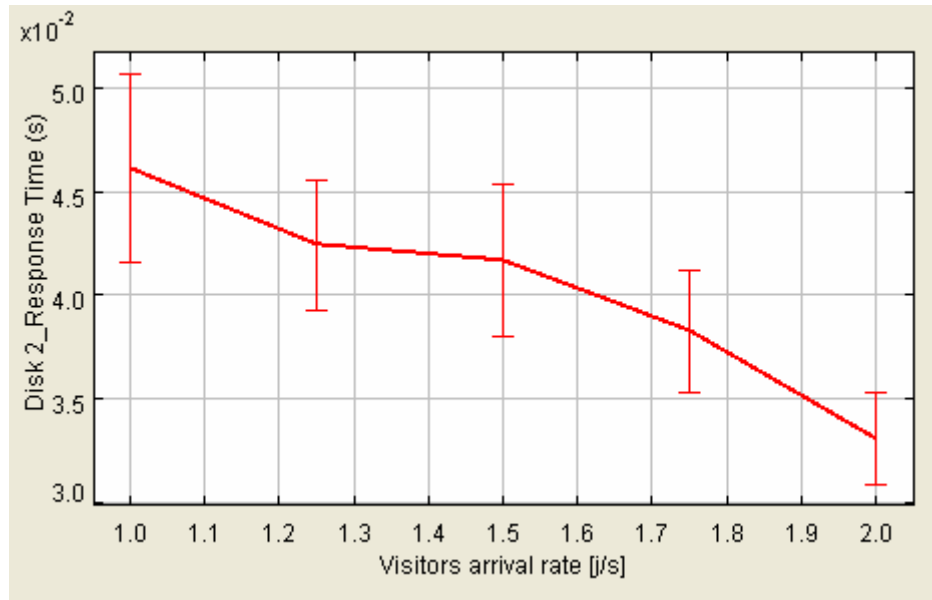


Figure 4.2.1.10 – Disk 2 Response Time : Disk 1 and 2 have a similar behaviour

Because of the Players jobs reduction the disk utilization was considerably reduced (as shown in Figure 4.2.1.5). The same happened to single disk response time as show in Figures 4.2.1.10 because of the lighter disk load.

- **Global System Response Time**

In this subsection we present the simulation results for the Global System Response Time. It represents the average duration of a user session.

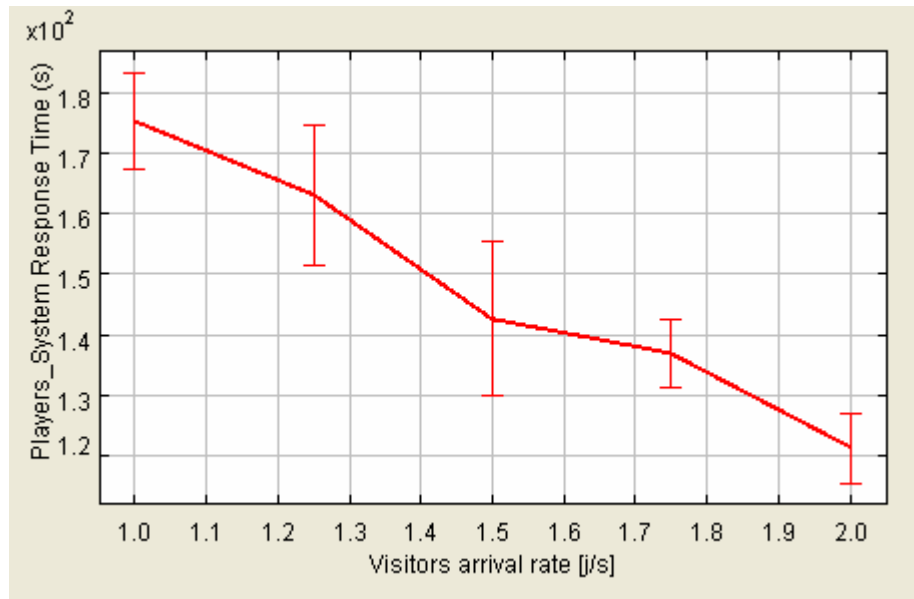


Figure 4.2.1.11 – Class *Players* Global System Response Time

Previously we introduced the disks utilization and response time reduction. These factors caused the Players Global System Response Time reduction shown in Figure 4.2.1.12. That's because there are less jobs that faster loop within the system.

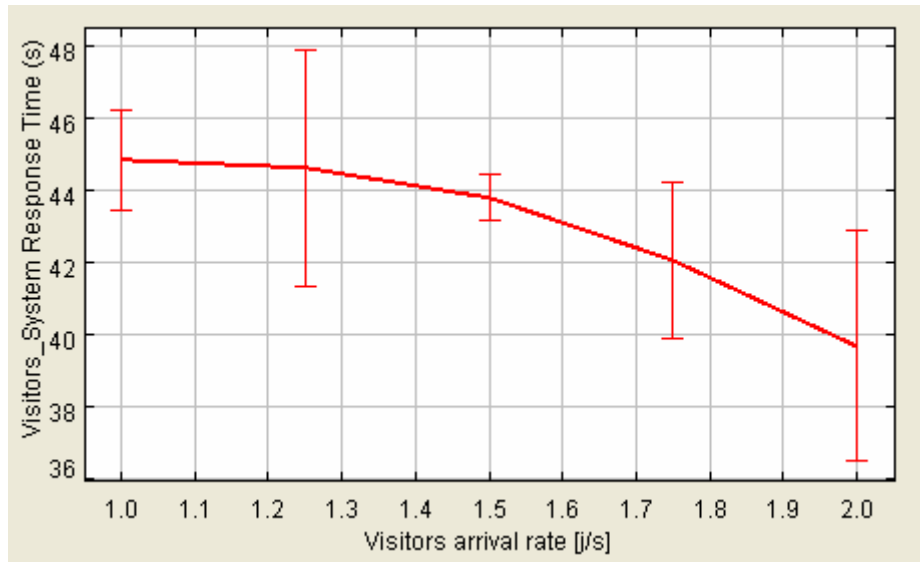


Figure 4.2.1.12 – Class *Visitors* Global System Response Time

In the case of Visitors jobs we also have a system response time reduction, that corresponds to a reduction of the visitors web navigation session duration. If a job doesn't loop inside the FCR (the faster ones) are the ones who have the higher probability to exit the system and contribute to the calculus of system response time.

- **Visitors/Players Mix in the FCR :**

We investigated the results obtained with Global System Response Time by calculating the mix of Visitors/Players in FCR region. Figure 4.2.1.13 was obtained through a series of what-if analysis where we obtained the queues length inside the FCR region. These values were then added to obtain the number of jobs in the region.

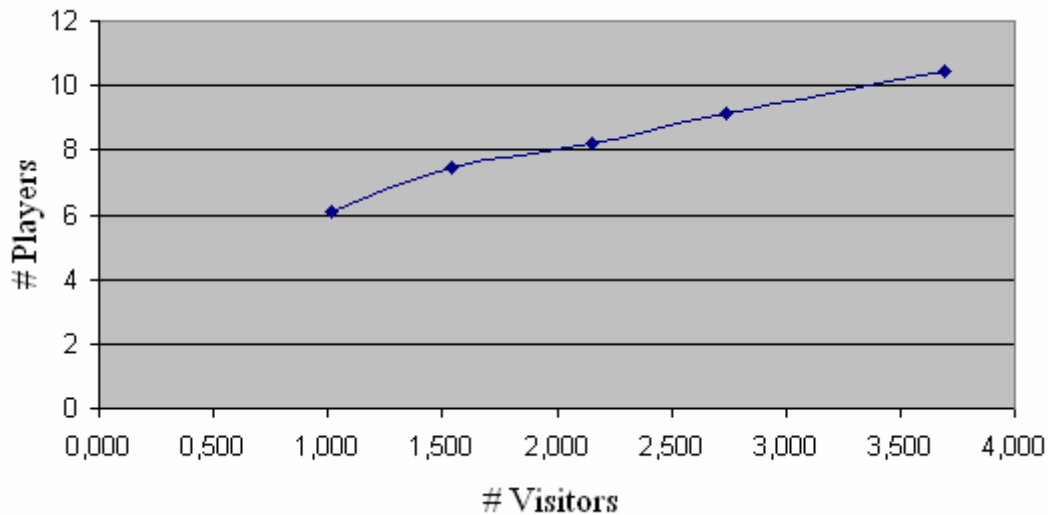


Figure 4.2.1.13 – Jobs Mix in FCR

Figure 4.2.1.13 shows how the progressive increment of Visitors also causes an increment of the number of Players in web servers queue. That's because Visitors spend more time in the stations (they have an higher service time) than Players one, and this mean that Players must wait a longer time before being served. An higher number of Players in FCR also means that these jobs have an higher probability to be dropped. The more they loop in the system (96%) the more they have an higher probability to be dropped in FCR. And so jobs that contribute to Global Response Time are the ones who loop as little as possible.

4.2.2 Players Arrival Rate Increment Simulation

This subsection will presents What-If analysis results when the Visitors arrival is incremented by 100%. Even in this case, the jobs arrival rate increment could correspond to an increment of gamers during week-ends.

- *Web Servers Utilization :*

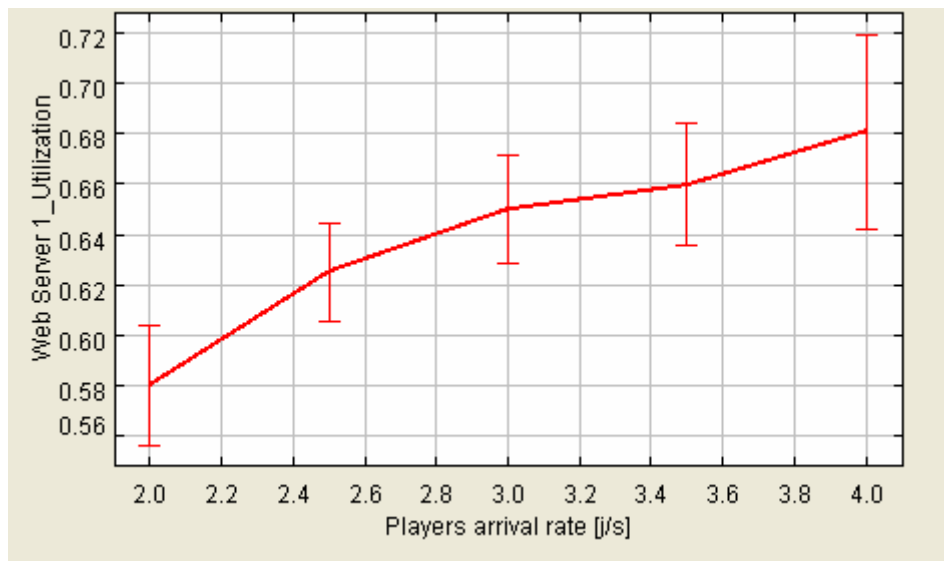


Figure 4.2.2.1 – Web Server 1 Utilization, Disk 2 have similar behaviour

Figure 4.2.2.1 shows how the increase of 100% of the arrival rate number of Players, causes and increment of about 10% of the web servers utilization. That's because these jobs spend few time in these stations.

- ***Storage Disks Utilization :***

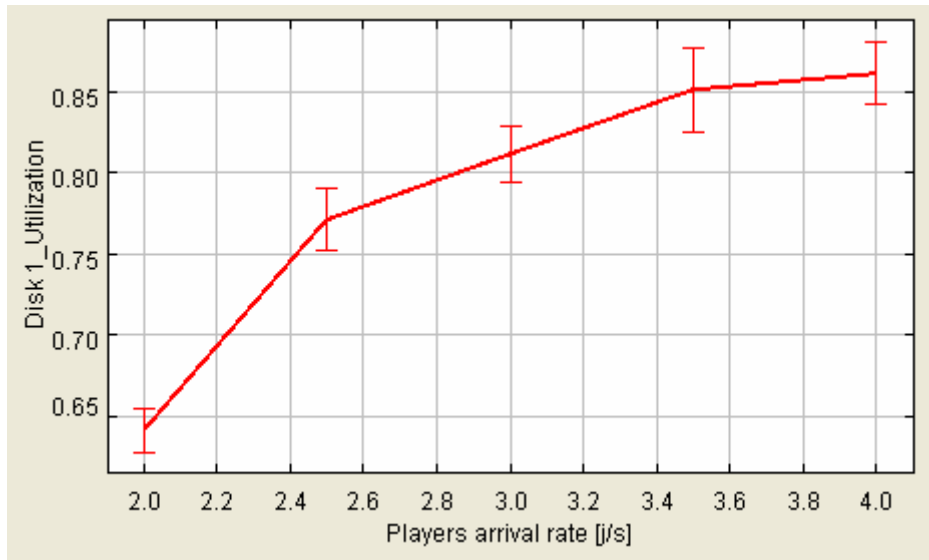


Figure 4.2.2.2 – Disk 1 Utilization : Disk 2 and 3 have similar behaviour

Figure 4.2.2.2 shows that the Players arrival rate increment introduce a considerable disks utilization increment of about 20% that brings the disks utilization near to saturation.

- ***Web Server Response Time for Visitors Job Class:***

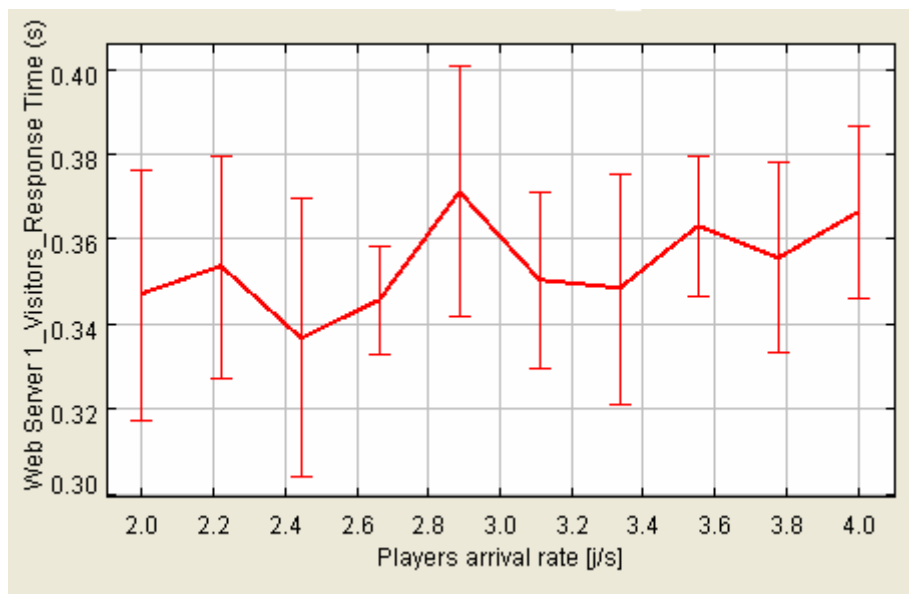


Figure 4.2.2.3 – Web Server 1 Response Time

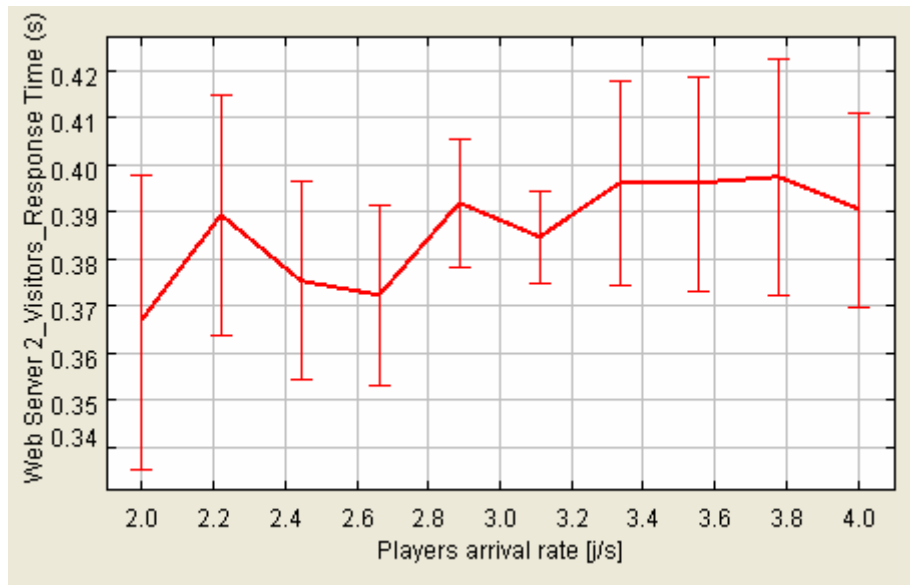


Figure 4.2.2.4 – Web Server 2 Response Time

Figures 4.2.2.3 and 4.2.2.4 show that the response time of a single web server is increased of about 0.02 second. Because these response times must be understood as the time a user must wait to download a entire web page, we can conclude that the Players jobs increment doesn't negatively affect on waiting time of web pages download.

- ***Disks Utilization :***

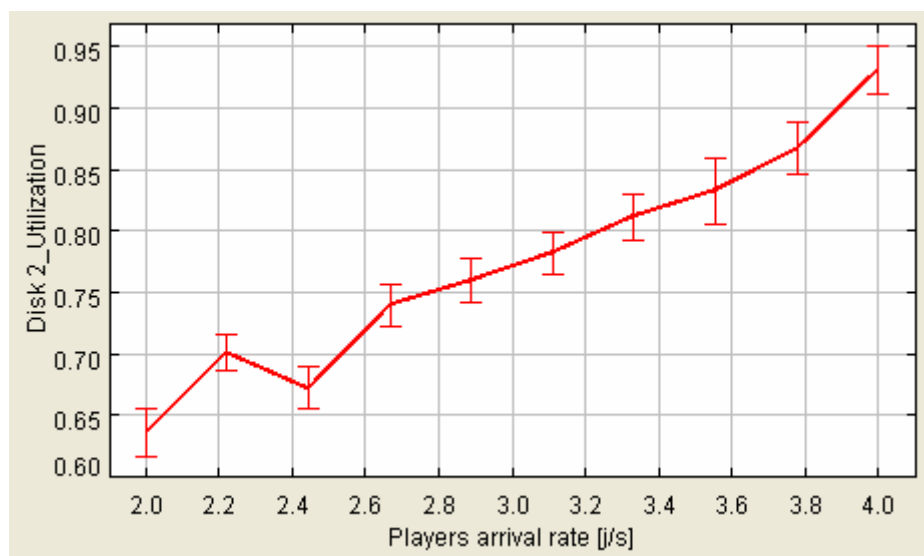


Figure 4.2.2.5 – Disk 2 Utilization, Disk 1 and 3 have similar behaviour

Because an high number of Players loop inside system, we have a considerable increment in disks utilization. With an arrival rate equal to 4 jobs/second we have a disk utilization close to saturation.

- ***Finite Capacity Region (FCR) Drop Rate :***

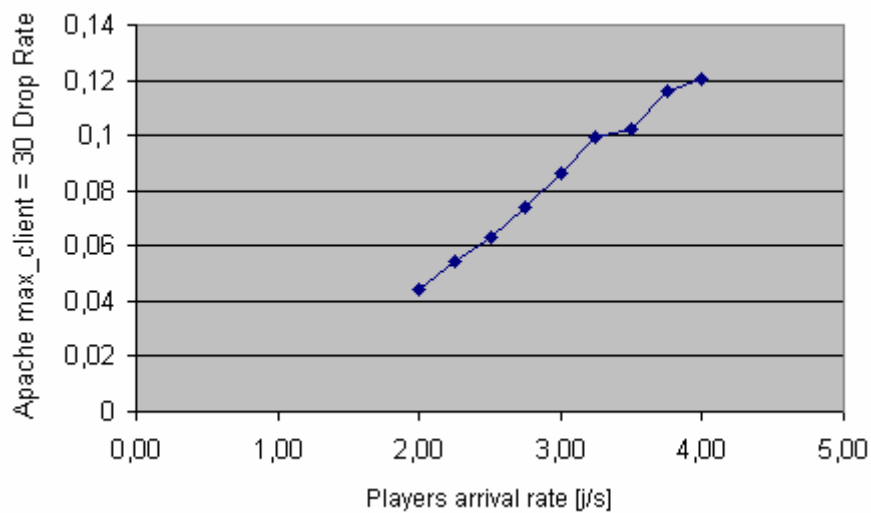


Figure 4.2.2.6 – Class *Visitors* FCR Drop Rate

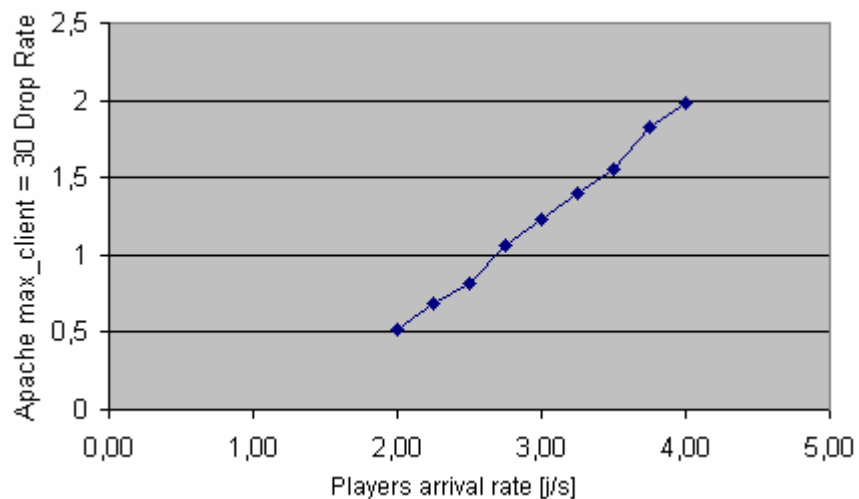


Figure 4.2.2.7 – Class *Players* FCR Drop Rate

Figure 4.2.2.7 shows that with the increment of the Players arrival rate we also had an increment in the number of dropped Players jobs. We can notice that when the arrival rate is equal to 4 jobs/second about half of the incoming jobs are dropped by FCR.

- **System Throughput :**

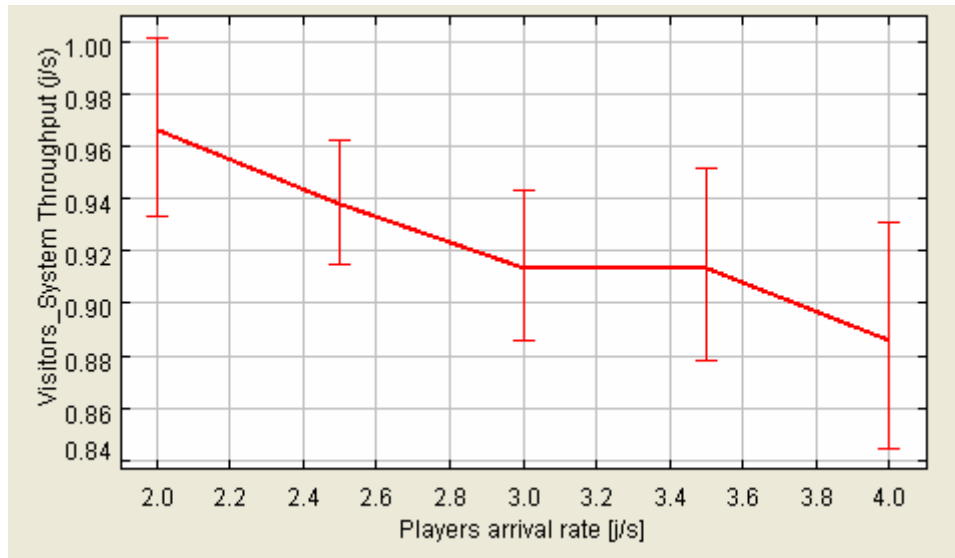


Figure 4.2.2.8 – Class *Visitors* System Throughput

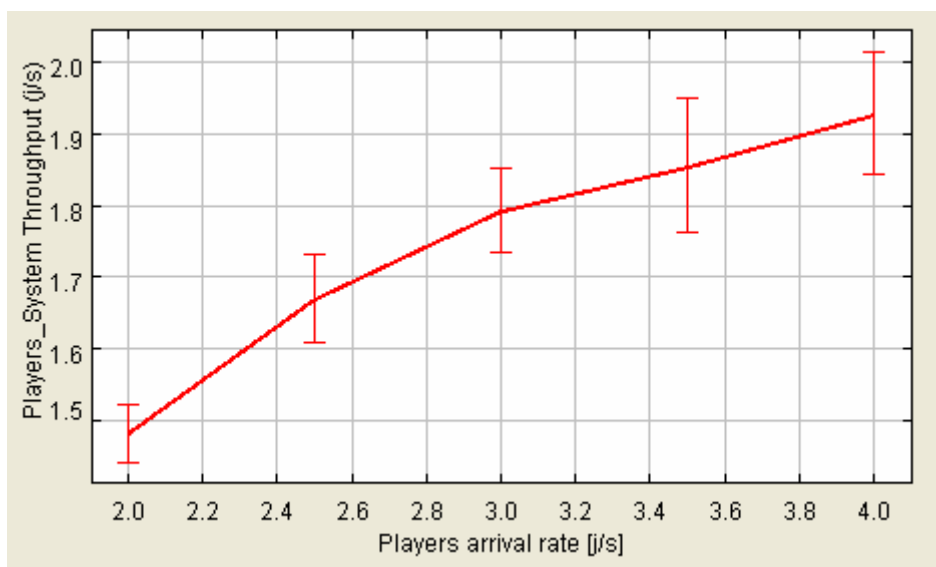


Figure 4.2.2.9 – Class *Players* System Throughput

Because of the dropping effect of the FCR region we can see in Figures 4.2.2.8 and 4.2.2.9 that System Throughput has always a value lower than the corresponding arrival rate.

- **Global System Response Time :**

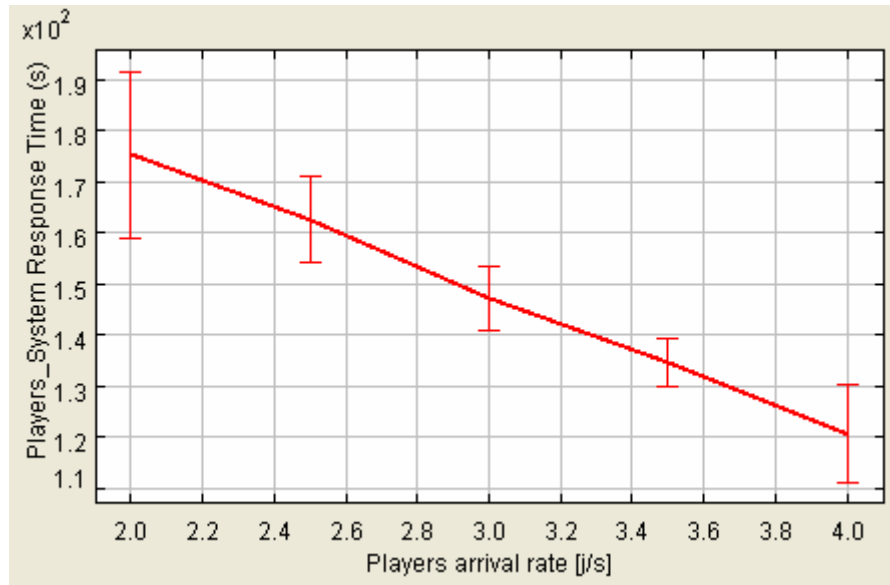


Figure 4.2.2.10 – Class *Players* Global Response Time

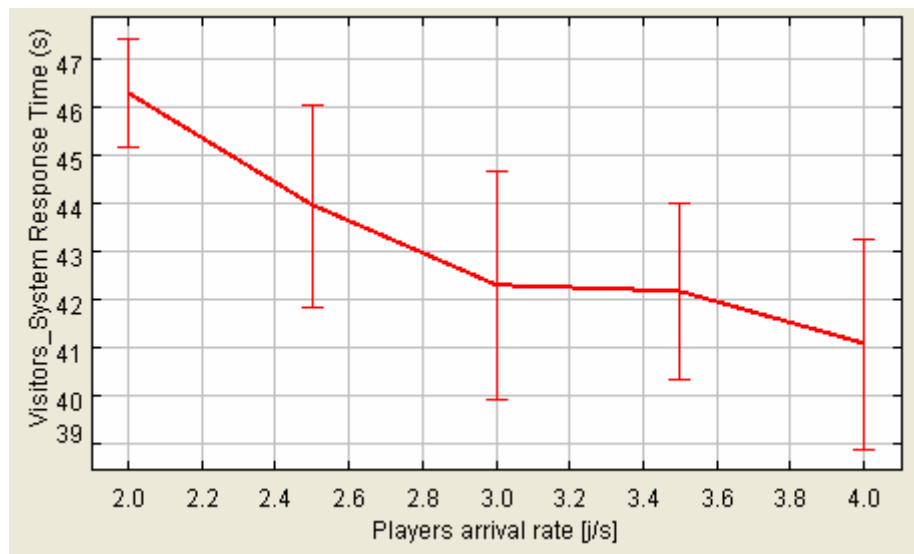


Figure 4.2.2.11 – Class *Visitors* Global Response Time

The Global System Response Time reduction will be better investigate with the FCR jobs mix graph (Figure 4.2.2.12) in the section below.

- **Visitors/Players Mix in the FCR :**

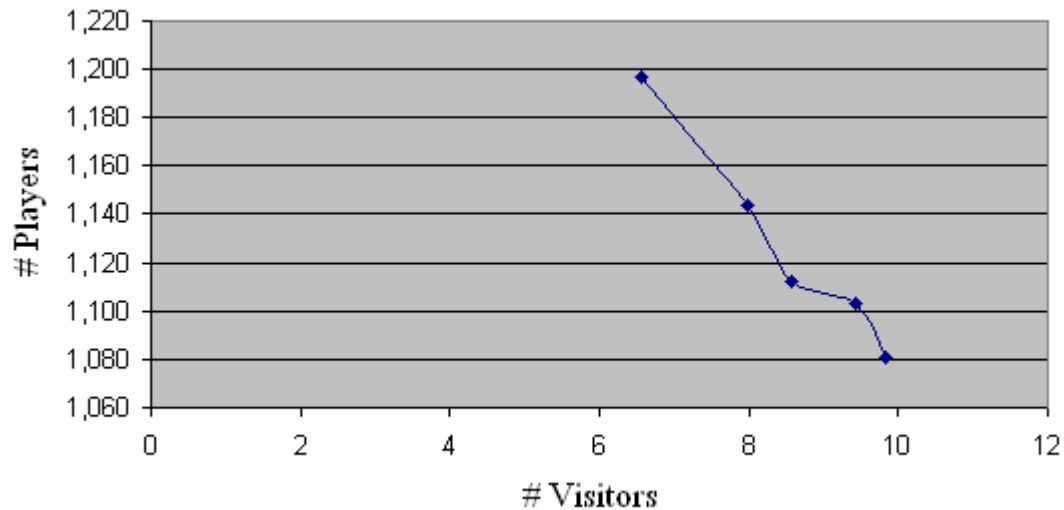


Figure 4.2.2.12 – Jobs Mix in FCR

There are conclusions similar to the first what-if analysis. The Players arrival rate increment determinates a greater number of Players in the FCR region. Because of the greater number of Players jobs, Visitors one have an higher probability to being dropped and so only the faster ones contribute to the Global System Response Time. Similar considerations could be done for Players jobs.

A Queueing Network Model with a Finite Capacity Region and Drop Rule

Project for the doctoral course: “Advanced Techniques of
Performance Evaluation of Computer Systems”

Jonatha Anselmi
anselmi@elet.polimi.it

March 26, 2008

1 Introduction

In this case study we evaluate the performance of an open single-class queueing network model with a finite capacity region and drop rule. Queueing networks with finite capacity regions impose upper bounds on the number of jobs that can simultaneously reside in a set of stations [3, 4], and can be used to model application constraints. Such networks more accurately model real systems behavior since in reality they do have a finite capacity. For instance, consider a web-based multi-level application having a pool of threads to process HTTP requests. The pool size is chosen in order to either have a compromise between response time and number of rejected requests and model the finite capacity of the web server queue. If this upper bound is ever reached, clients will be locked out. The web-based application can be described with a set of resources modeling the architecture (e.g., web, application and database servers) with a finite capacity region bounded by the size of the HTTP threads pool.

In general, such constraints make the analysis of finite capacity models more difficult than in the product-form case, thus, now we solve such networks with simulation.

In Section 2 we discuss the queueing network model under investigation, give notation and model parameters. The performance evaluation, performed using the JMT simulation engine [2], is shown in Section 3.

2 The Model

The open queueing network under investigation models the behavior of a web-site. The architecture is characterized by a web server, an application server and a database server. We assume that the queueing network is separable [4], i.e. it satisfies the product-form assumptions. However, since in practise the web-server handles a finite number of connections, we assume that all the servers belong to a single finite capacity region. The size of the region is implicitly defined in the web-server settings: the *Apache HTTP Server Project* [1], for instance, relies on the *httpd.conf* configuration file which lets the user bound the maximum number of clients simultaneously handled. The drop rule is meant to reject requests arriving when the number of requests inside the region has already reached the maximum capacity allowed.

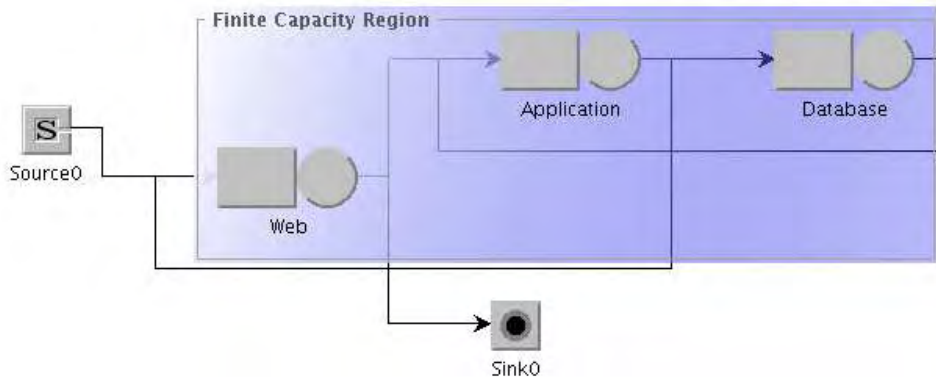


Figure 1: The queueing network model with a finite capacity region.

(a) Service times		(b) Routing prob. inside the region			
Station	S_i	$p_{i,j}$	1	2	3
1 (Web)	0.067	1 (Web)	0	0.4	0
2 (Application)	0.050	2 (Application)	0.5	0	0.5
3 (Database)	0.055	3 (Database)	0	1	0

Table 1: Service time and routing probabilities

2.1 Notation

We denote the mean arrival rate by λ , the region capacity by B , the mean service time at station m by S_m , the mean number of visit at station m by V_m and the probability that a job is transferred to station j after completion at station i by $p_{i,j}$. Loadings D_m are computed as $D_m = V_m S_m$ (see, i.e., [4]). The total number of station is denoted by M . We assume that both service and think times are exponentially distributed. The performance indices of interest are:

- $R(\lambda)$: system response time
- $Q_i(\lambda)$: queue length of station i
- $X(\lambda)$: system throughput.

In Figure 1 is depicted the queueing network model under investigation.

2.2 Model Parameters

After a log files analysis, we assume that the network is characterized by the service times and the routing probabilities shown in Table 1. In-going and out-going probabilities are, respectively, $p_{in,1} = 1$ and $p_{1,out} = 0.6$.

3 Performance Evaluation

In this section we evaluate the system performance by varying λ and B ; we assume that service times are fixed. In order to better understand the improved accuracy of the model depicted above, we also make a parallel evaluation considering the network without the finite capacity region.

First, we compute the maximum allowed arrival rate and derive the system stability condition. Exploiting the values of routing probabilities shown in Table 1 we compute V_i , $i = 1, \dots, M$ and,

consequently, D_i , $i = 1, \dots, M$. Recall that for an open model the number of visits is given by the linear system [3]

$$V_i = p_{\text{in},i} + \sum_{j=0}^M p_{j,i} V_j, \quad i = 1, \dots, M \quad (1)$$

Substituting numerical values of $p_{i,j}$ in (1), we have

$$\begin{cases} V_1 = 1 + 0.5V_2 \\ V_2 = 0.4V_2 + V_3 \\ V_3 = 0.5V_2 \end{cases} \quad (2)$$

Therefore, we get visits $V_1 = 1.\bar{6}$, $V_2 = 1.\bar{3}$, $V_3 = 0.\bar{6}$ which lead to loadings $D_1 = 0.0\bar{4}$, $D_2 = 0.08\bar{3}$, $D_3 = 0.07\bar{3}$. The stability condition is verified if

$$\lambda < \frac{1}{\max_{1 \leq m \leq M} D_m} = \frac{1}{D_2} = 12 = \lambda_{\text{Max}} \text{ job/sec} \quad (3)$$

In our model, since the drop rule forces the number of requests to be less than or equal to B , we are able to evaluate performance indices also for $\lambda > \lambda_{\text{Max}}$.

Now we simulate the system using the JModel module of JMT choosing 95% level confidence intervals. Using the JModel *What-If* option, we perform a numerical evaluation by varying λ from 2 to 18 job/sec with step 1 and B from 80 to 120 units with step 20. In the following figures, we show performance indices of interest obtained by both simulation and assuming that the finite capacity constraint and the drop rule are removed. Recall that in this latter case, formulas for $R(\lambda)$, $X(\lambda)$ and $Q(\lambda)$ are given by

$$X(\lambda) = \lambda \quad (4)$$

$$R(\lambda) = \sum_{i=1}^3 \frac{D_i}{1 - \lambda D_i}. \quad (5)$$

$$Q(\lambda) = X(\lambda)R(\lambda) \quad (6)$$

The former is known as *flow balance assumption* and the latter as Little's Law (see, e.g., [4]). As shown in the following figures, analytic solutions (the red dashed lines) are not given for $\lambda \geq 12$ since the stability condition is no more verified.

We first consider the case in which $B = 80$. As shown in Figure 2, the finite capacity region drop rule does not let the response time grow to infinity. System throughput saturates where the stability condition is not verified, i.e. for $\lambda \geq 12$ (see Figure 3). However, the important aspect is that when the stability condition is no more verified, as λ increases, the response time continues to grow and saturates only when the finite capacity constraint is reached. This is obviously due to the fact that the number of jobs in the region for $\lambda = 12$ (i.e. $Q(12)$) is still strictly less than B (see Figure 4).

System response time begins to saturate approximately for $\lambda \simeq 14$. For $\lambda \geq 14$ we note the number of customers in the network approaches to the capacity constraint, i.e. $Q = \sum_{i=1}^M Q_i \simeq B$ (see Figure 4).

We now consider the case in which $B = 100$ (see Figures 5, 6, 7). Analogously, we note the response time saturation is reached for $\lambda \simeq 14$. In this case, the saturation value is trivially greater than the previous case since in the region we allow more jobs. As a consequence, increasing the finite capacity constraint we proportionally increase the response time and drop less jobs¹.

We now consider the case in which $B = 120$. Analogously, we note the response time saturation is reached for $\lambda \simeq 14$ (see Figures 8, 9, 10). Also for this case the previous considerations hold.

In any case we note that when the capacity constraint is active (i.e., the number of requests inside the region reaches the upper bound B) the system throughput saturates to the same constant value,

¹The value of dropped jobs ratio is not available with the current release of JMT (0.6.2)

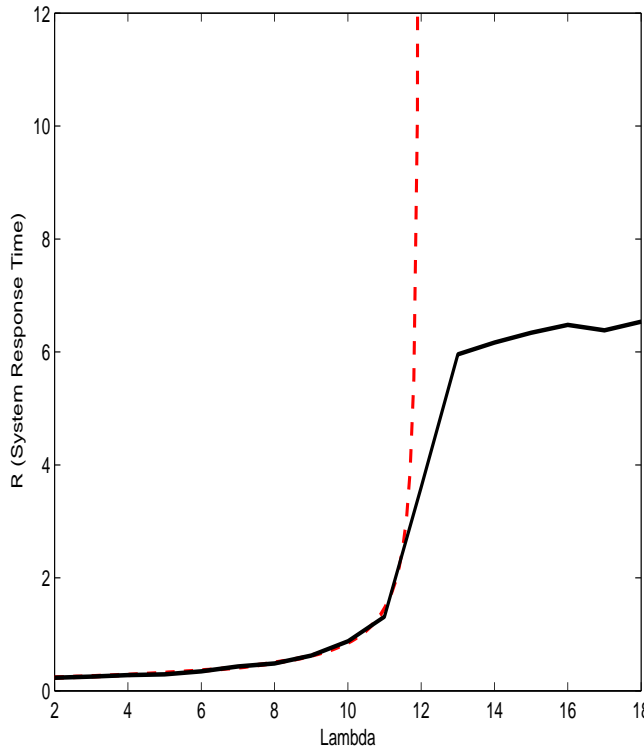


Figure 2: Response time (B=80)

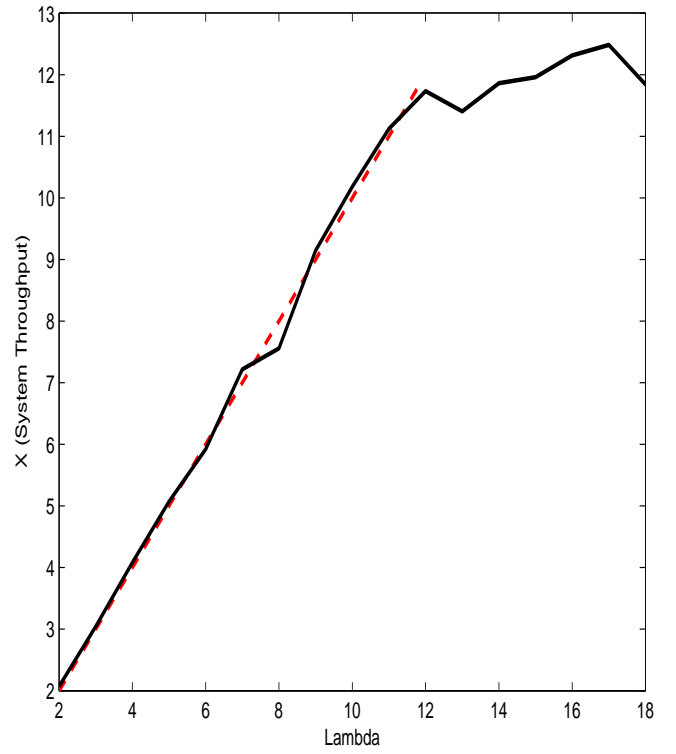


Figure 3: Throughput (B=80)

$X(\lambda) = \lambda_{max} = 12$. For $\lambda \geq \lambda_{max}$, the way in which $R(\lambda)$ and $Q(\lambda)$ are related is explained by (6), i.e. Little's Law. Thus, keeping fixed $X(\lambda)$ to λ_{max} , R and Q are directly proportional. In this scenario, Little's Law can be used to choose the capacity size which guarantees a given response time (note the different response time saturation values for the three cases).

References

- [1] *Apache http server project*, <http://httpd.apache.org/>.
- [2] *Java modelling tools*, <http://jmt.sourceforge.net>.
- [3] G. BOLCH, S. GREINER, H. DE MEER, AND K. S. TRIVEDI, *Queueing Networks and Markov Chains*, Wiley-Interscience, 2005.
- [4] E. D. LAZOWSKA, J. ZAHORJAN, G. S. GRAHAM, AND K. C. SEVCIK, *Quantitative System Performance*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

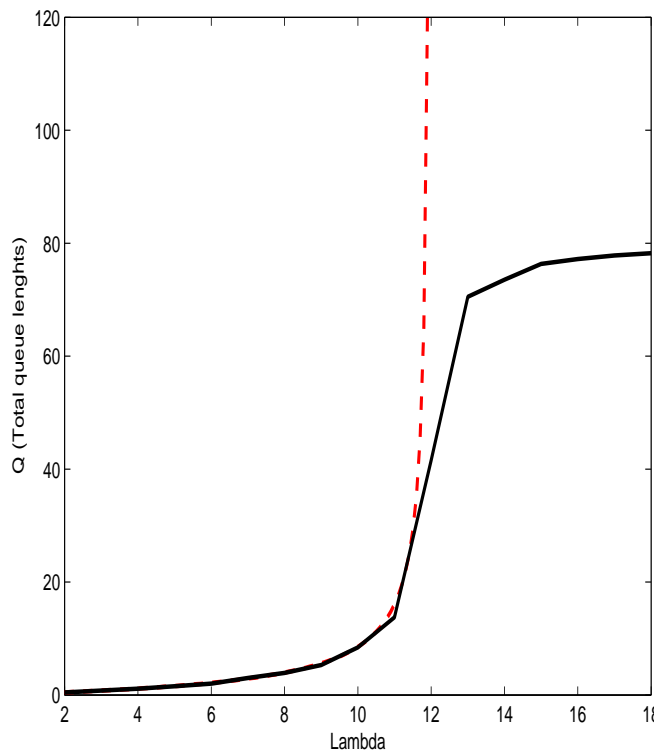


Figure 4: Number of jobs in the region ($B=80$)

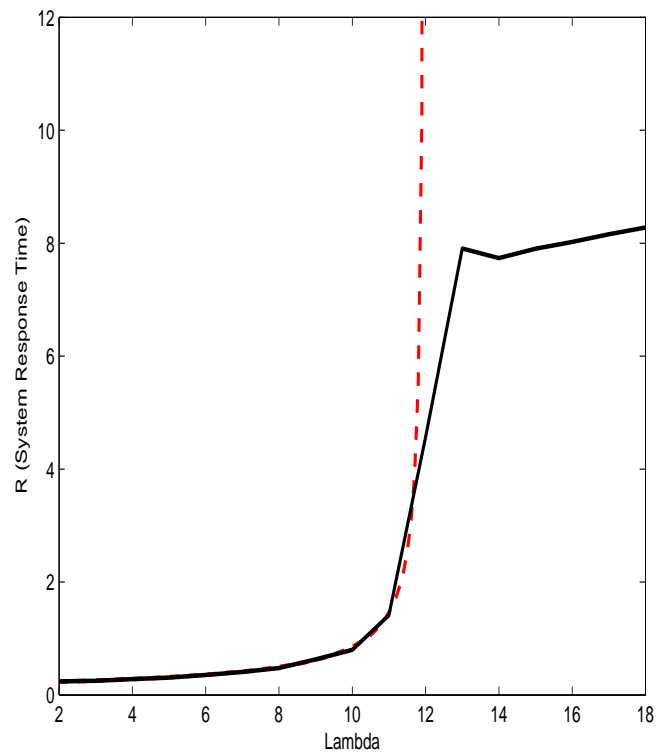


Figure 5: Response time ($B=100$)

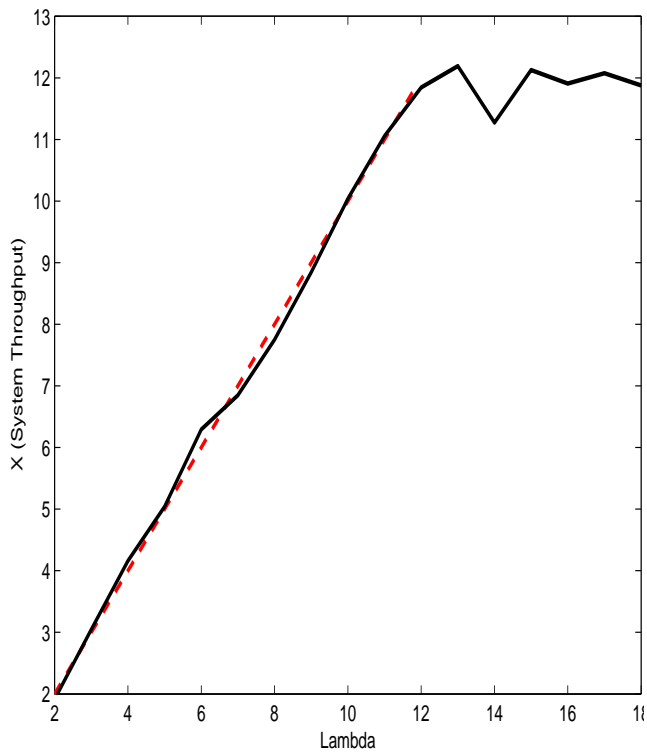


Figure 6: Throughput ($B=100$)

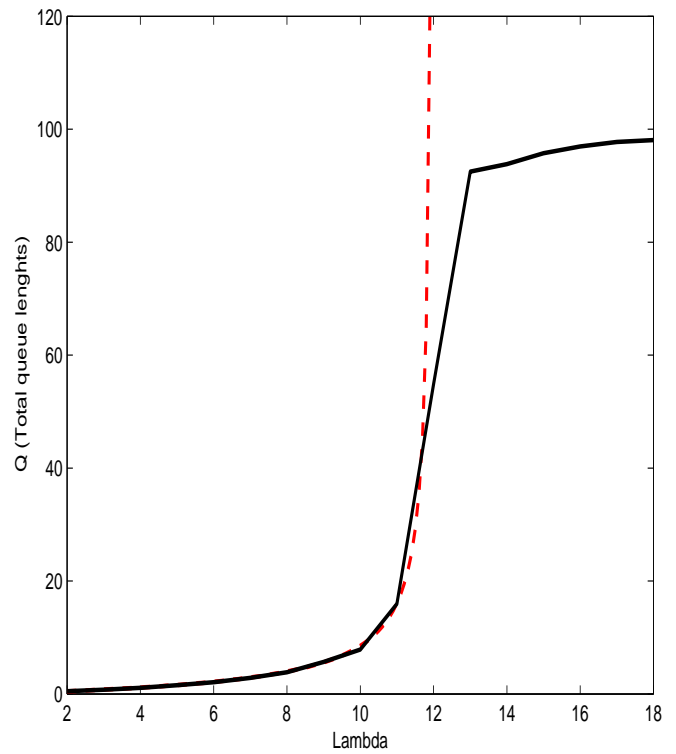


Figure 7: Number of jobs in the region ($B=100$)

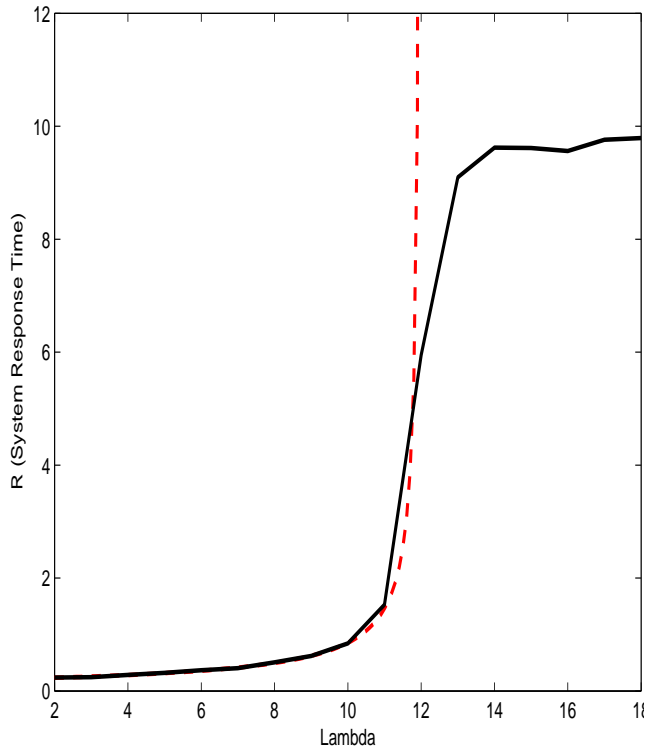


Figure 8: Response time ($B=120$)

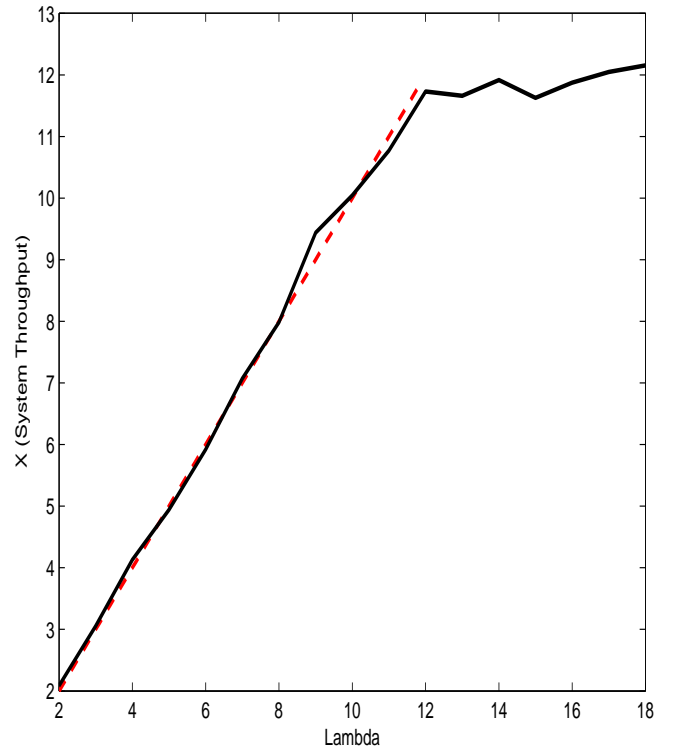


Figure 9: Throughput ($B=120$)

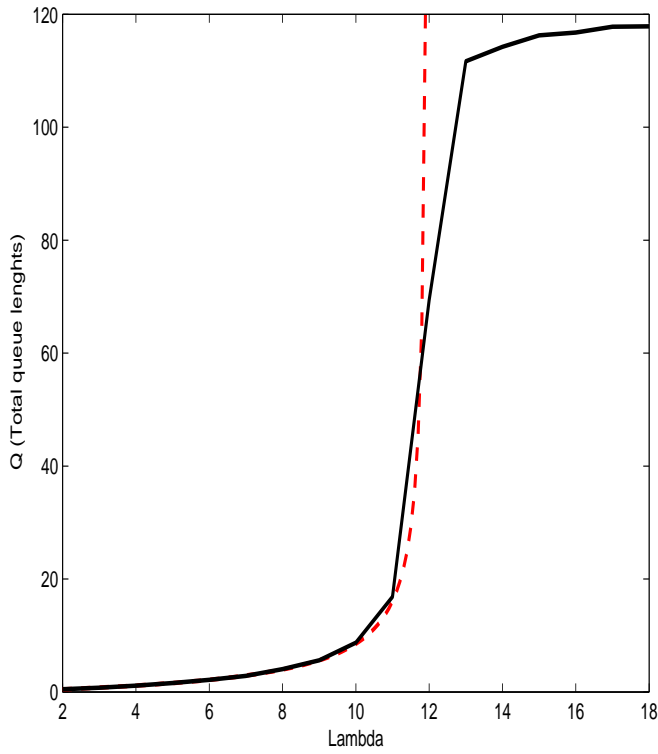


Figure 10: Number of jobs in the region ($B=120$)

3 – Computer System Architectures

3.1 – Performance Evaluation of Computer Memory Hierarchy	108
3.2 – Shared-Memory Multiprocessor Systems: Hierarchical Task Queue	135

Performance Evaluation of Computer Memory Hierarchy

Anirban Dutta Choudhury

duttacha@alari.ch

Alie El-Din Mady

madya@alari.ch

Advanced Learning and Research Institute

ALaRI - USI

Lugano, Switzerland

Final version

Jan. 27, 2008

Contents

1	Introduction to Computer Memory Hierarchy	5
1.1	General Purpose Computer Memory Architecture Model	5
1.2	Network-on-Chip Memory Architecture Model	7
2	Modeling Approaches	8
2.1	MATLAB Modeling	8
2.1.1	Model not considering Hard disk	9
2.1.2	Model considering Hard disk	9
2.1.3	Final Cost Function	10
2.2	Queuing Network Modeling	10
2.2.1	Introduction	10
2.2.2	General Purpose Computer Memory Architecture Model	11
2.2.3	Network-on-Chip Memory Architecture Model	20
3	Results	23
3.1	MATLAB Simulation Result	23
3.1.1	Simulation Not Considering Hard disk	24
3.1.2	Simulation Considering Hard disk	24
3.1.3	Conclusion	26

List of Figures

1.1 General Purpose Computer Architecture Hierarchy	6
2.1 General Purpose Computer Architecture Queuing Network Model	11
2.2 QL of CPU	12
2.3 QL of L1	12
2.4 QL of L2	12
2.5 QL of DRAM	12
2.6 QL of HD	12
2.7 TP of CPU	12
2.8 TP of L1	13
2.9 TP of L2	13
2.10 TP of Dram	13
2.11 TP of HD	13
2.12 System Response Time	13
2.13 QL of DRAM (Normal Distribution)	13
2.14 QL of HD (Normal Distribution)	13
2.15 General Purpose Computer Architecture Queuing Network Model (with Fork & Join)	14
2.16 General Purpose Computer Architecture Queuing Network Model (NOT Working !)	15
2.17 Processor Queue Length	16
2.18 Processor Queue Time	16
2.19 Processor Throughput	16
2.20 L1 Queue Length	16
2.21 L1 Queue Time	16
2.22 L1 Throughput	16
2.23 L2 Queue Length	17
2.24 L2 Queue Time	17
2.25 L2 Throughput	17
2.26 DRAM Queue Length	17
2.27 DRAM Queue Time	17
2.28 DRAM Throughput	17
2.29 System Response Time	17

2.30 Processor Queue Length	18
2.31 Processor Queue Time	18
2.32 Processor Throughput	18
2.33 L1 Queue Length	18
2.34 L1 Queue Time	18
2.35 L1 Throughput	18
2.36 L2 Queue Length	19
2.37 L2 Queue Time	19
2.38 L2 Throughput	19
2.39 DRAM Queue Length	19
2.40 DRAM Queue Time	19
2.41 DRAM Throughput	19
2.42 System Response Time	19
2.43 NoC Queuing Network Model	20
2.44 NoC QL of CPU1	21
2.45 NoC QL of CPU2	21
2.46 NoC QL of 1st L1	21
2.47 NoC QL of 2nd L1	21
2.48 NoC QL of L2	21
2.49 NoC QL of DRAM	21
2.50 NoC QT of CPU1	22
2.51 NoC QT of CPU2	22
2.52 NoC QL of L1 of CPU1	22
2.53 NoC QL of L1 of CPU2	22
2.54 NoC QT of L2	22
2.55 NoC QT of DRAM	22
2.56 NoC System Response Time	22
3.1 Normalized Cost Function	23
3.2 Cost function NOT Considering Hard disk	24
3.3 Cost function Considering Hard disk	25

Chapter 1

Introduction to Computer Memory Hierarchy

1.1 General Purpose Computer Memory Architecture Model

In most cases, the Computer Memory Hierarchy for General Purpose Computers may be summarized as given below [5, 6]:

1. **CPU Registers:** In computer architecture, a processor register is a small amount of storage available on the CPU whose contents can be accessed more quickly than storage available elsewhere. Most, but not all, modern computer architectures operate on the principle of moving data from main memory into registers, operating on them, then moving the result back into main memory.
2. **Level 1 Cache:** A CPU cache may be defined as a volatile memory used by the central processing unit of a computer to reduce the average time to access memory. The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. As long as most memory accesses are to cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory. L1 cache is the nearest to the processor (most of the times on the same chip) and it is also second fastest memory after the CPU registers.
3. **Level 2 Cache:** L2 cache is slower and bigger cache.
4. **Main Memory (RAM):** This is the Random Access Memory and this is the biggest volatile memory devices, since it loses its data when the power supply is removed.
5. **Hard Disk:** This is nonvolatile memory or non-volatile storage, the slowest and biggest in the computer memory hierarchy that can retain the stored information even when not powered.

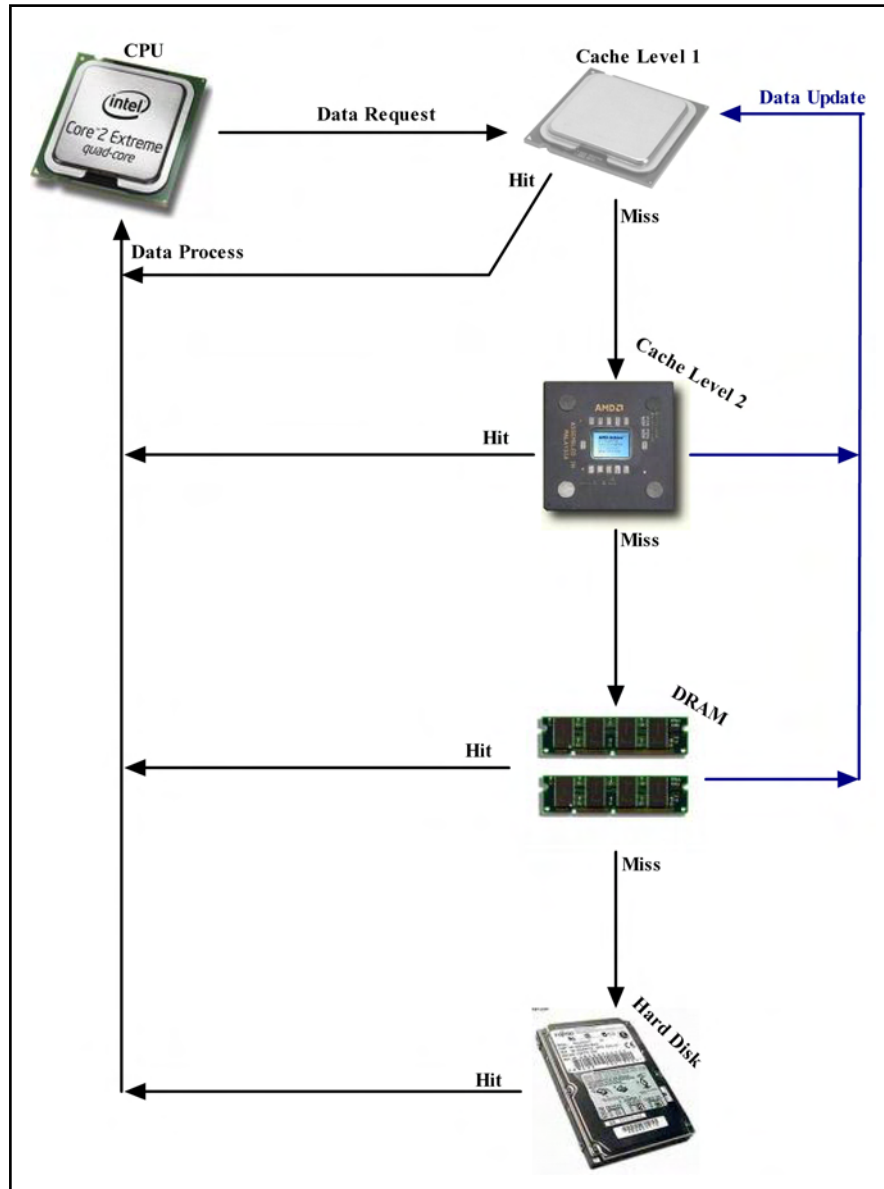


Figure 1.1: General Purpose Computer Architecture Hierarchy

We explained the aforementioned memory structure in the figure 1.1. Nowadays, thanks to top level micro & nano electronics researches and implementation, in general purpose computers we can see different improvisations, like:

1. A large sized L1 and L2 on-chip cache

2. Multi-Processor Personal computers.

1.2 Network-on-Chip Memory Architecture Model

During the last decade, Network on Chip has become a dense area of research in Embedded Systems Research field. Due to tremendous advancement in VLSI, several functional blocks in a single chip has become a reality. Network-on-Chip (NoC) is an emerging paradigm for communications within large VLSI systems implemented on a single silicon chip. In a NoC system, modules such as processor cores, memories and specialized IP blocks exchange data using a network as a *public transportation sub-system* for the information traffic. A NoC is constructed from multiple point-to-point data links interconnected by switches (or routers), such that messages can be relayed from any source module to any destination module over several links, by making routing decisions at the switches.

Chapter 2

Modeling Approaches

2.1 MATLAB Modeling

In our mathematical model, we used the miss rates of different memory components as the input to the system. The output function, also called cost function is a nonlinear function of all the input parameters. The basic form of cost function in our project is taken from the CACTI Tool manual [1].

We used a wide range of possible values of miss rates (0.02 to 0.3 in steps of 0.01) in our project.

```
M11 = (0.02:0.01:0.3); % L1 miss rate
M12 = (0.02:0.01:0.3); % L2 miss rate
MD  = (0.02:0.01:0.3); % DRAM miss rate
```

As mentioned below, The standard values of power and time costs are taken from different research publications [3, 4].

```
P11 = 3; % L1 Power in nJ
P12 = 200; % L2 Power in nJ
PD  = 3000; % DRAM Power in nJ
PHD = 1500000000; % disk Power in nJ
```

```
T11 = 2.2; % L1 access Time in ns
T12 = 100; % L2 access Time in ns
TD  = 2000; % DRAM access Time in ns
THD = 2000000; % disk access Time in ns
```

```
A11 = 2.*exp(3 - (M11.*100)); % L1 Area in MB
A12 = A11 .* (2.*exp(7));      % L2 Area in MB
AD  = A11 .* (2.*exp(8));      % disk Area in MB
```

2.1.1 Model not considering Hard disk

Following is the power and time cost function defined for such systems. These equations along with other following equations (equations having functions of the parameters $Ml1$, $Ml2$, & MD) are directly influenced by the widely known cache hit-miss formulas [5].

$$Power\ Cost = \frac{(Pl1 + Ml1 \times Pl2 + Ml1 \times Ml2 \times PD)}{(Pl1 + Pl2 + PD)} \quad (2.1)$$

where,

$Pl1$ = Cache Level 1 Power Consumption,
 $Pl2$ = Cache Level 2 Power Consumption,
 PD = DRAM Power Consumption,

$$Time\ Cost = \frac{(Tl1 + Ml1 \times Tl2 + Ml1 \times Ml2 \times TD)}{(Tl1 + Tl2 + TD)} \quad (2.2)$$

where,

$Tl1$ = Cache Level 1 Time Consumption,
 $Tl2$ = Cache Level 2 Time Consumption,
 TD = DRAM Time Consumption,

Following is the Area Cost function used in our project[2]. As shown in the following equation, all the different components of Area Cost *except Hard disk* are considered.

$$Area\ Cost = \frac{1}{Memory\ Area\ Efficiency} = \frac{Al1 + Al2 + AD}{IPC} \quad (2.3)$$

where,

$Al1$ = Cache Level 1 Time Consumption,
 $Al2$ = Cache Level 2 Time Consumption,
 AD = DRAM Time Consumption,
 IPC = Instruction per Cycle

We derived an empirical formula from the standard results published in research papers for modeling IPC using miss rate parameters. IPC is defined as:

$$IPC = (1 - Ml1) \times 1.45 \quad (2.4)$$

2.1.2 Model considering Hard disk

Now, we will define the individual components of the aforementioned Cost Function.

$$Power\ Cost = \frac{(Pl1 + Ml1 \times Pl2 + Ml1 \times Ml2 \times PD + Ml1 \times Ml2 \times MD \times PHD)}{(Pl1 + Pl2 + PD + PHD)} \quad (2.5)$$

where,

P11 = Cache Level 1 Power Consumption,
 P12 = Cache Level 2 Power Consumption,
 PD = DRAM Power Consumption,
 PHD = Hard disk Power Consumption

$$Time\ Cost = \frac{(Tl1 + Ml1 \times Tl2 + Ml1 \times Ml2 \times TD + Ml1 \times Ml2 \times MD \times THD)}{(Tl1 + Tl2 + TD + THD)} \quad (2.6)$$

where,

Tl1 = Cache Level 1 Time Consumption,
 Tl2 = Cache Level 2 Time Consumption,
 TD = DRAM Time Consumption,
 THD = Hard disk Time Consumption

The Area Cost function in this model is same as defined in the model not considering Hard disk (vide equation 2.3).

2.1.3 Final Cost Function

In the final cost function, we normalized each component to make sure each of them affects the final value in a similar way. So the cost function doesn't have any unit of different components mentioned earlier (i.e. nJ, ns and MB) and the *final cost function* is defined as following:

$$Cost\ Function = \frac{Power\ Cost}{Max.\ Power\ Cost} + \frac{Time\ Cost}{Max.\ Time\ Cost} + \frac{Area\ Cost}{Max.\ Area\ Cost} \quad (2.7)$$

In our project we considered two models and compared the cost functions. One model considers the presence of nonvolatile memory(i.e. Hard disk) in the architecture while the other model doesn't take into account Hard disk. In the following two sections, we will represent the different components of the cost functions for both the cases.

2.2 Queuing Network Modeling

2.2.1 Introduction

Queuing Network Modeling Approach is a particularly applicable for the kind of system we are discussing. We used a special modeling tool named *JMT version 0.7.3* developed in *Politecnico the Milano* [8]. The Java Modeling Tools (JMT) is a free open source suite for performance evaluation, capacity planning and modeling of computer and communication systems. The suite implements numerous state-of-the-art algorithms for the exact, asymptotic and simulative analysis of queueing network models, either with or without product-form solution.

2.2.2 General Purpose Computer Memory Architecture Model

Here, we started with the model in 2.1. In this model, the miss rates (constant) are :

$MI1 = 0.05$

$MI2 = 0.05$

$MD = 0.1$

The model is a closed model with instruction buffer queue length = 8.

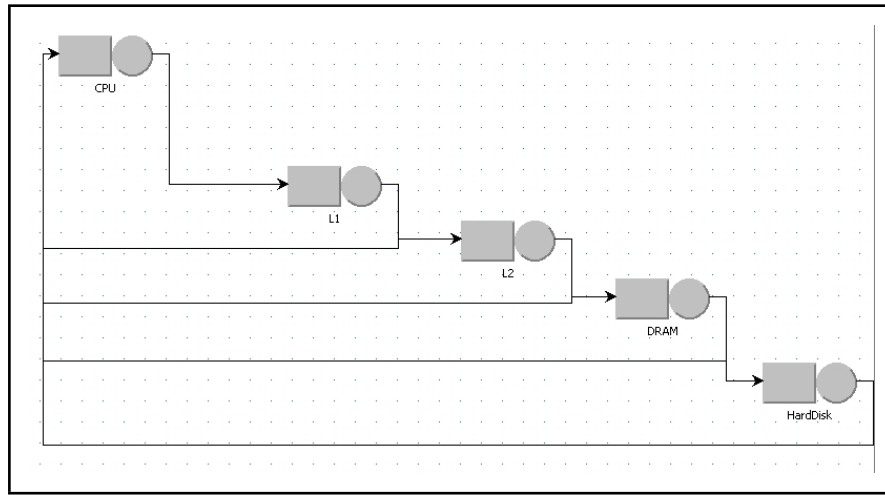


Figure 2.1: General Purpose Computer Architecture Queuing Network Model

Result

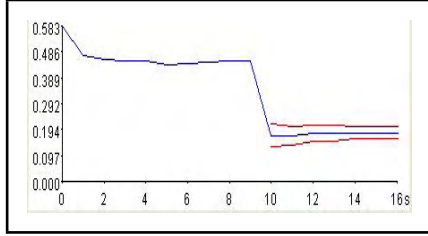


Figure 2.2: QL of CPU

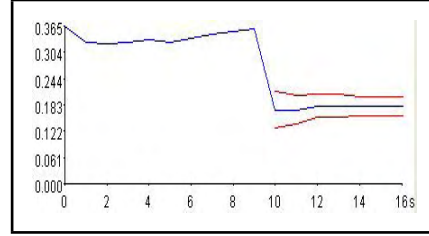


Figure 2.3: QL of L1

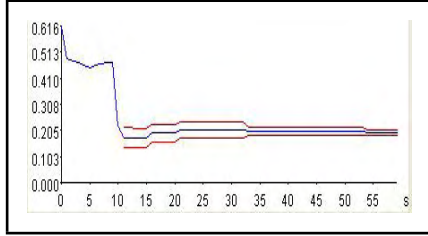


Figure 2.4: QL of L2

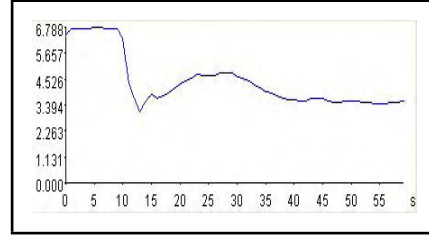


Figure 2.5: QL of DRAM

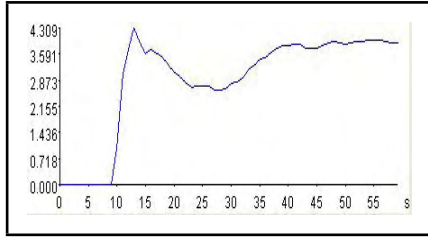


Figure 2.6: QL of HD

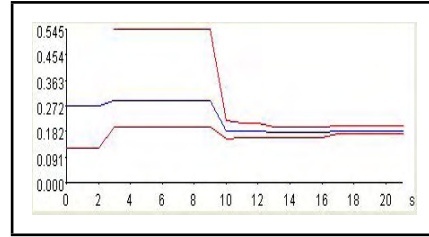


Figure 2.7: TP of CPU

To reflect the variations of the nature of the big (and slow) memory components (e.g. Hard disk), we also checked the same model with Hard disk having a normal distribution with a mean value 2×10^4 ns (same as the previous model) and a standard deviation of 100 ns (refer to figure 2.13 and 2.14).

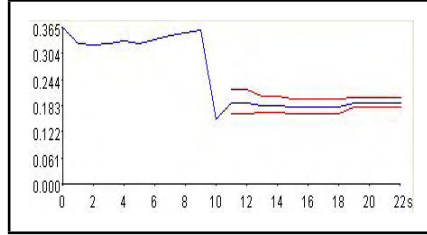


Figure 2.8: TP of L1

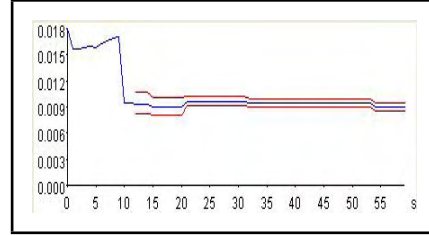


Figure 2.9: TP of L2

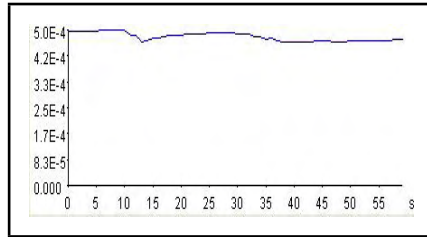


Figure 2.10: TP of Dram

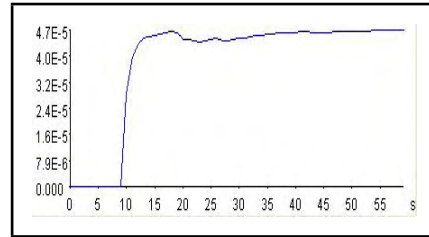


Figure 2.11: TP of HD

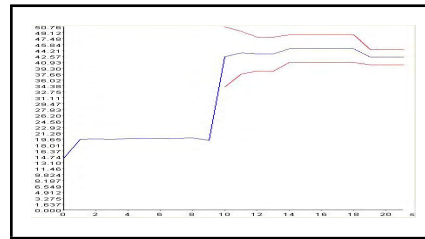


Figure 2.12: System Response Time

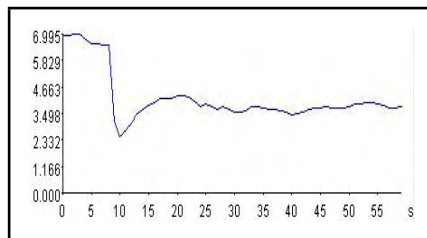


Figure 2.13: QL of DRAM (Normal Distribution)

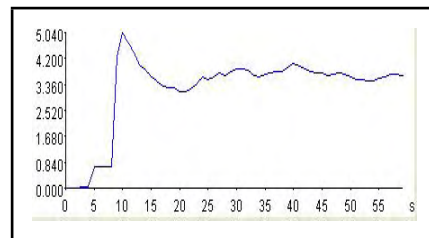


Figure 2.14: QL of HD (Normal Distribution)

In this part of the implementation, we are modeling General Purpose Computers with only one processor and consequent memory structure dedicated to that processor. As shown in the figure 2.15, this system is explained as following. If there is a hit after a cache or other memory search, the value goes back to CPU and L1 cache values are updated (Fork 0 and Fork 1). In case of a miss, the search proceeds to the next level of memory. Processor and L1 cache have more than one inputs. And we use joins (Join 0 and Join 1) to constrain the exponential increment of the number of tokens in the system. Otherwise, it would become an unstable system with 2 forks increasing the number of jobs exponentially.

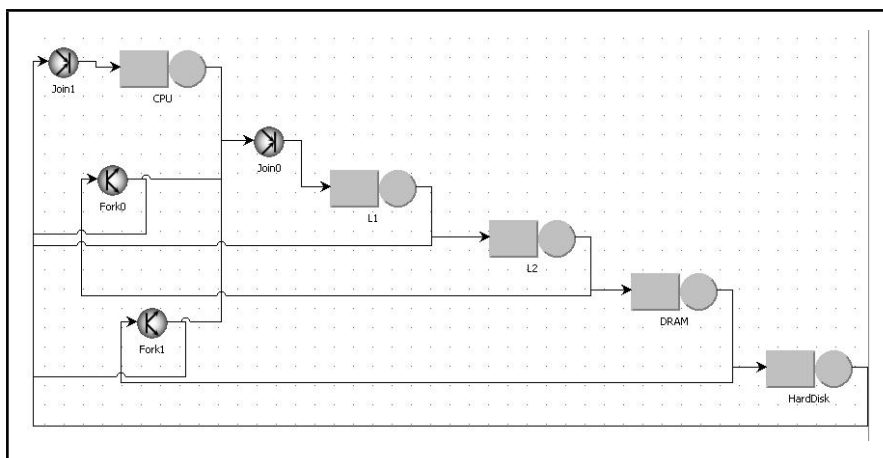


Figure 2.15: General Purpose Computer Architecture Queuing Network Model (with Fork & Join)

Here when we tried to run the system with instruction buffer length = 8, the JMT result window was blank (figure 2.16).

The initial number of instructions (number of jobs in the model) was 10000. The model was run with 2 different set of parameters, namely:

1. **Modeling with High Miss Rate:** L1 miss rate = 0.1, L2 miss rate = 0.05 and DRAM miss rate = 0.01.
2. **Modeling with Low Miss Rate:** L1 miss rate = 0.05, L2 miss rate = 0.005 and DRAM miss rate = 0.001.

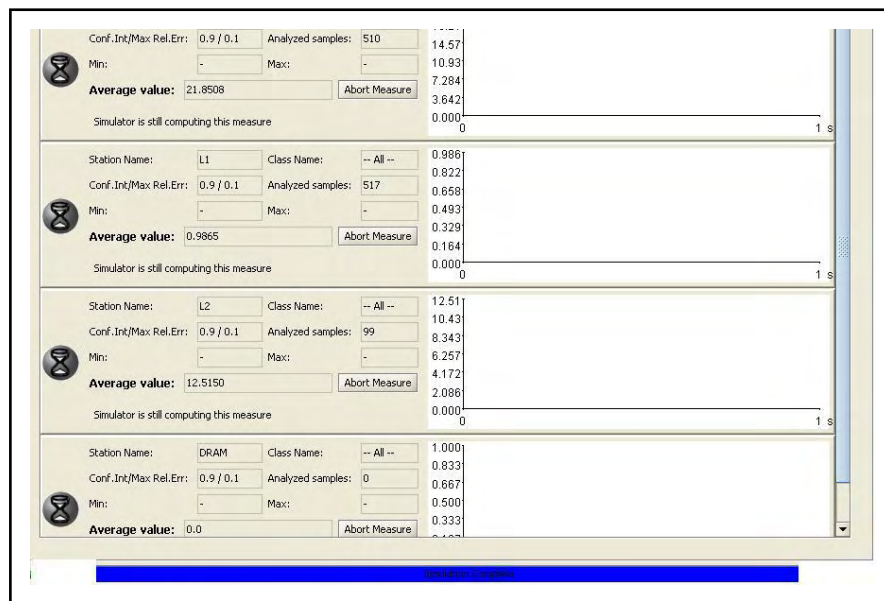


Figure 2.16: General Purpose Computer Architecture Queuing Network Model (NOT Working !)

Result (with $N = 10000$)

1. Low Miss Rate

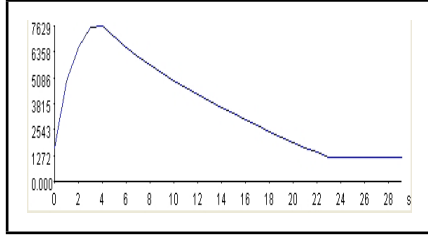


Figure 2.17: Processor Queue Length

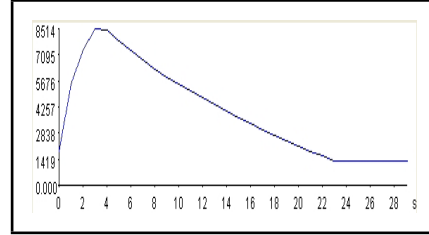


Figure 2.18: Processor Queue Time

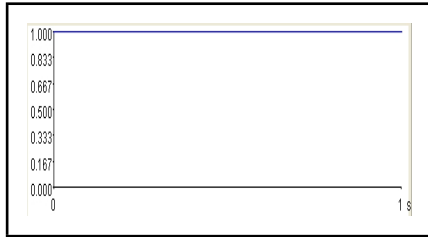


Figure 2.19: Processor Throughput

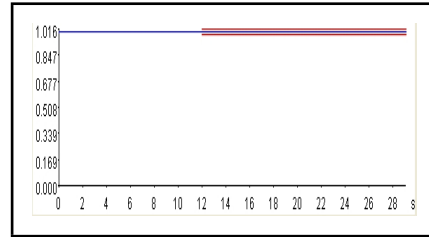


Figure 2.20: L1 Queue Length

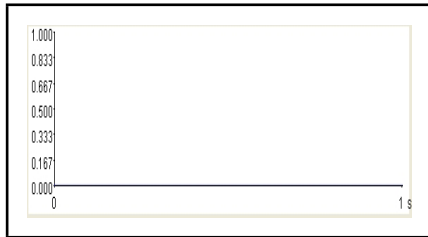


Figure 2.21: L1 Queue Time

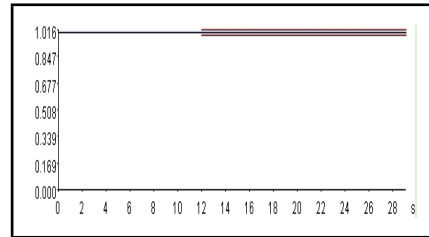


Figure 2.22: L1 Throughput

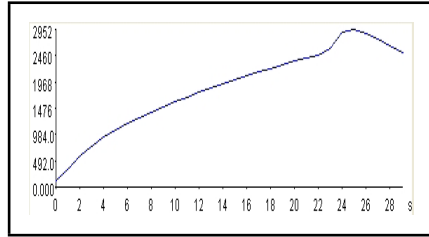


Figure 2.23: L2 Queue Length

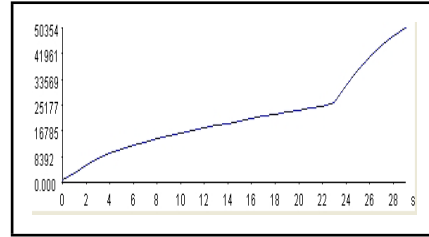


Figure 2.24: L2 Queue Time

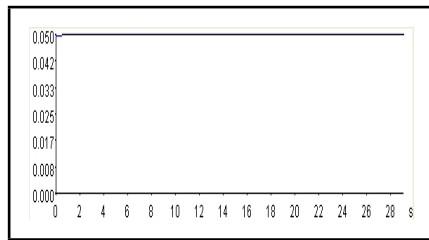


Figure 2.25: L2 Throughput

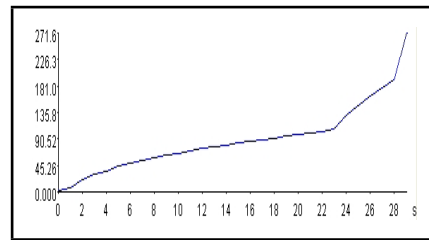


Figure 2.26: DRAM Queue Length

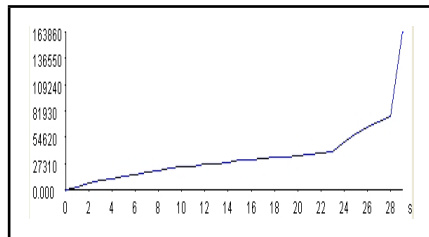


Figure 2.27: DRAM Queue Time

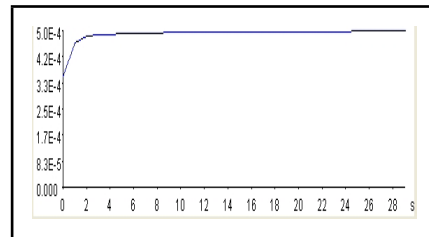


Figure 2.28: DRAM Throughput

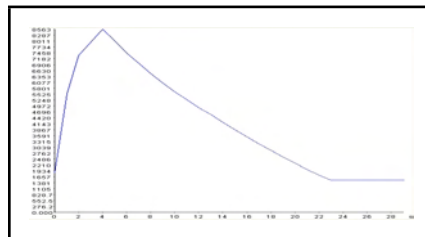


Figure 2.29: System Response Time

2. High Miss Rate

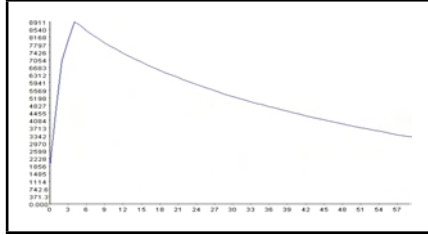


Figure 2.30: Processor Queue Length

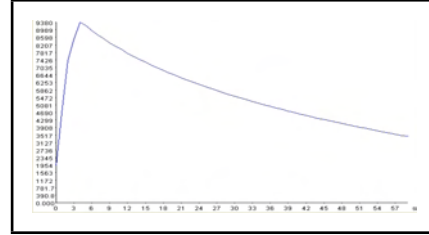


Figure 2.31: Processor Queue Time



Figure 2.32: Processor Throughput

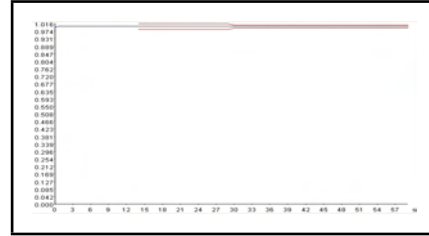


Figure 2.33: L1 Queue Length



Figure 2.34: L1 Queue Time

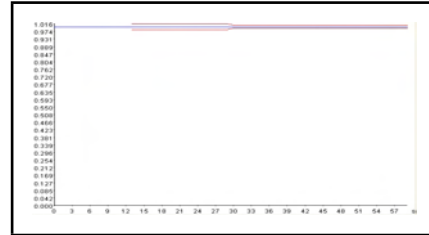


Figure 2.35: L1 Throughput

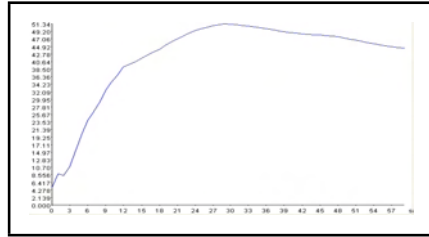


Figure 2.36: L2 Queue Length

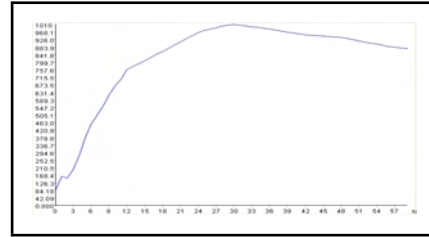


Figure 2.37: L2 Queue Time

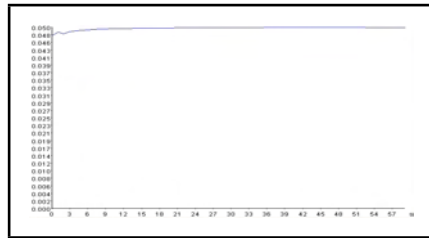


Figure 2.38: L2 Throughput

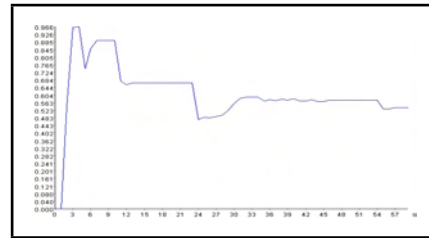


Figure 2.39: DRAM Queue Length

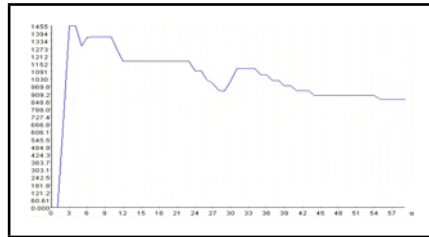


Figure 2.40: DRAM Queue Time

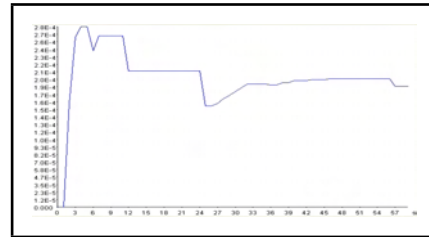


Figure 2.41: DRAM Throughput

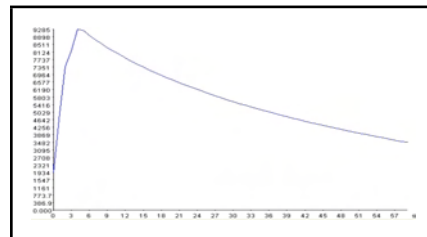


Figure 2.42: System Response Time

2.2.3 Network-on-Chip Memory Architecture Model

In the Network-on-Chip Architecture environment, the systems have typically several resources for the same job (like more than one processor in a single system). As shown in 2.43, we have two processors having dedicated L1 cache memory but sharing L2 cache and main memory. This is an open source model with 2 sources (CP schedulers) and one sink (jobs reaching this point are served, in other words the memory execution is finished).

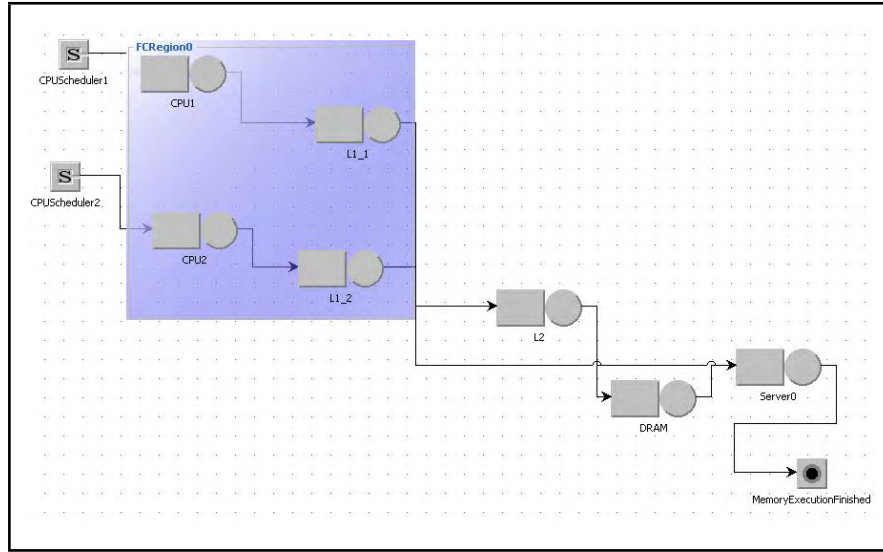


Figure 2.43: NoC Queuing Network Model

In NoC architecture, we used the following parameters - L1 miss rate = 0.05 and L2 miss rate = 0.3.

Result : NoC

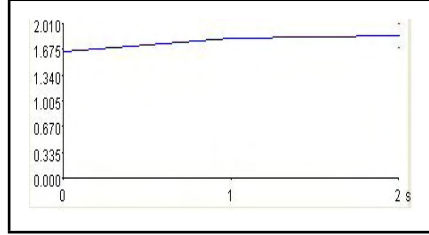


Figure 2.44: NoC QL of CPU1

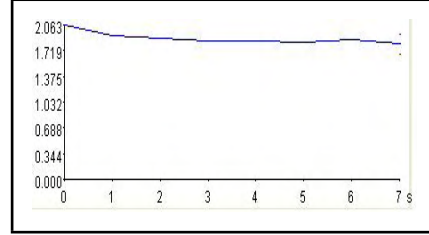


Figure 2.45: NoC QL of CPU2

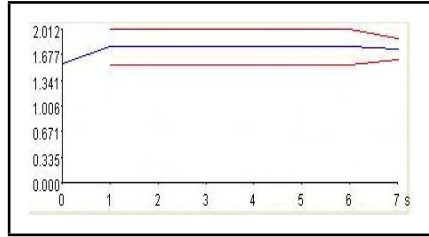


Figure 2.46: NoC QL of 1st L1

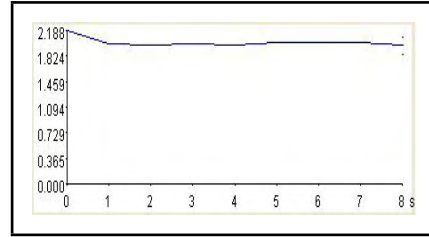


Figure 2.47: NoC QL of 2nd L1

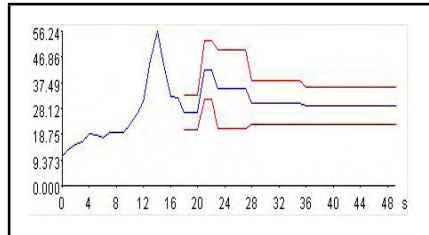


Figure 2.48: NoC QL of L2

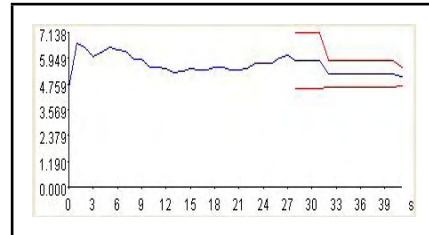


Figure 2.49: NoC QL of DRAM

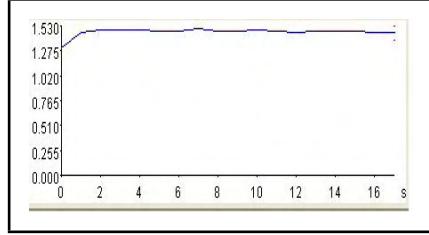


Figure 2.50: NoC QT of CPU1

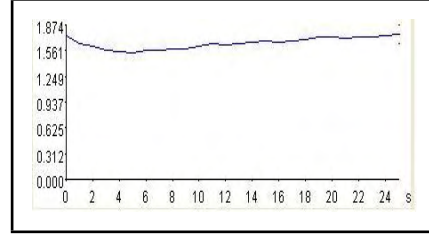


Figure 2.51: NoC QT of CPU2

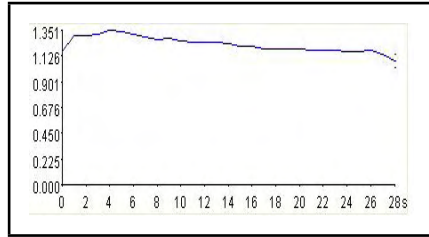


Figure 2.52: NoC QL of L1 of CPU1

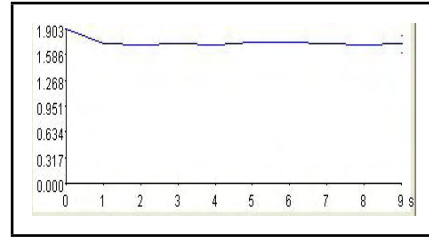


Figure 2.53: NoC QL of L1 of CPU2

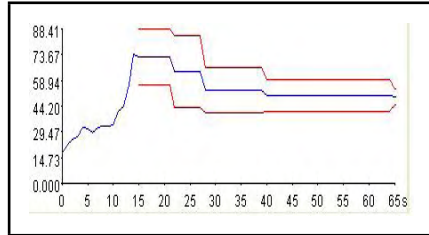


Figure 2.54: NoC QT of L2

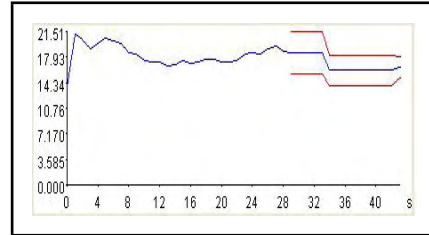


Figure 2.55: NoC QT of DRAM

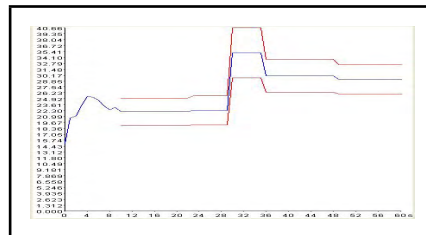


Figure 2.56: NoC System Response Time

Chapter 3

Results

3.1 MATLAB Simulation Result

As shown in chapter 2.7, all the individual components of the final cost function are normalized before adding up. In figure 3.1, we can see the variation of each component over a set of design space points of MI1, MI2, MD and MHD. And then in the 4th segment of 3.1, we can see the normalized components added up to generate the *Final Cost Function*.

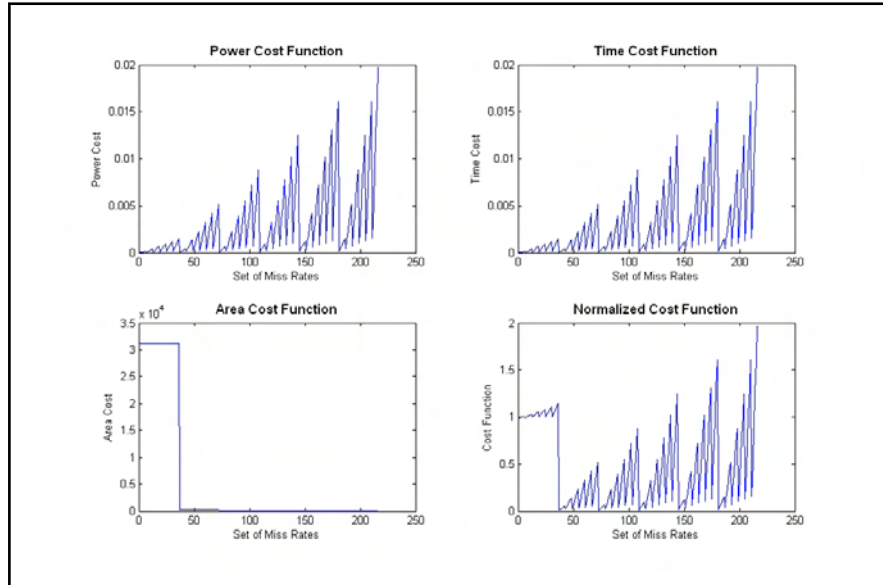


Figure 3.1: Normalized Cost Function

3.1.1 Simulation Not Considering Hard disk

Here in 3.2 we can see the results of MATLAB simulation not considering Hard disk in the system. The color of the graph changes with the height of the 3-D plane. As expected from the basic equations discussed, Power and Time cost functions are increasing with L1 and L2 miss rate. On the other hand, Area cost function is decreasing abruptly with increasing miss rates. The *Final Cost Function* reflects the nature of all the graphs. The design points with the lowest heights are the optimum combination for our system.

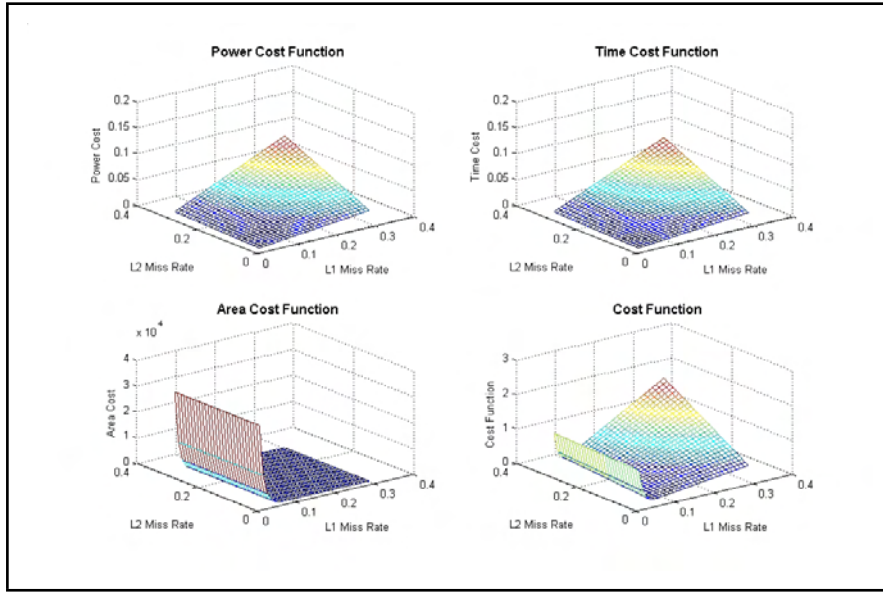


Figure 3.2: Cost function NOT Considering Hard disk

3.1.2 Simulation Considering Hard disk

In 3.3, we have 4 dimensions, namely Ml1, Ml2, MD and MHD. So we have drawn slices along axes of the miss rates to show the *Final Cost Function*. The red end of the colors shown in the graph have a higher value of final cost function, while the black end of the spectrum have a low final cost function value. The intensity of the color changes with the value. The trade off is clear as the lighter region in Area Cost Graph is just opposite of Power and Time cost graph.

As a result, we can say the most optimum points are somewhere middle in the 3-D region (Neither in the extreme left as in Power/Time Cost graph, nor in the extreme right as in Area Cost graph).

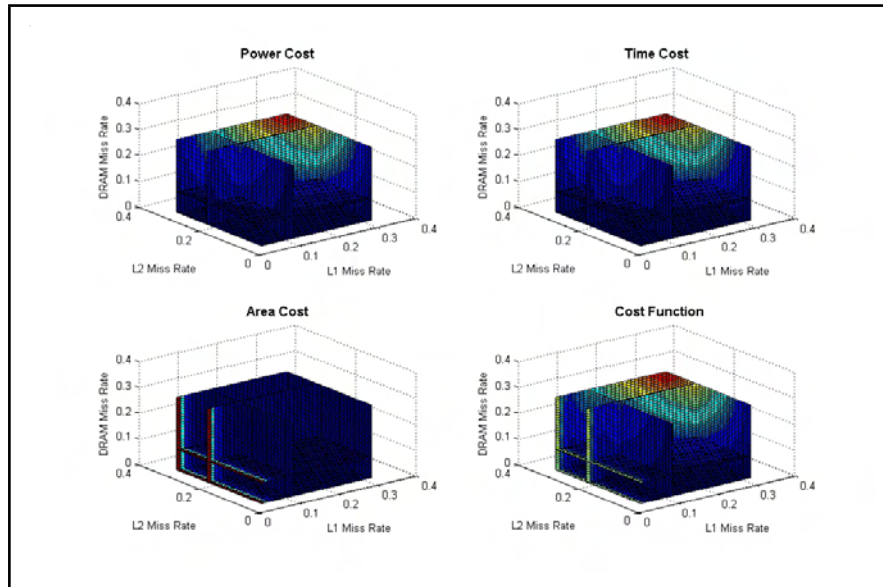


Figure 3.3: Cost function Considering Hard disk

3.1.3 Conclusion

The findings and explanations from this queuing network model is given below:

1. In General Purpose Computer Memory Architecture,
 - (a) In low miss rate systems, the CPU queue length and time is staying a bit in the maximum value and then coming back. While in the high miss rate system there is a sharp maximum point and then the curve is coming down with a steep slope. This can be explained as following. The low miss rate system have hit hit rates. As a result the jobs are driven to CPU more fast and the system is holding the maximum value for a longer time.
 - (b) In our system, the level 1 cache is considered on-chip and as fast as CPU, so as expected, the L1 queue time is constant at 0, as there will no queue formed at all. Following the same logic, L1 throughput is also 1 all the time.
 - (c) L2 and DRAM having an exponentially large service time, the queue time and queue length of both the servers are going up.
2. In Network-on-Chip Memory Architecture,
 - (a) In NoC, there are L1 cache dedicated to 2 processors. In both of them, the queue length is stable around 2.
 - (b) One very important observation is that DRAM is not following any specific pattern. This is because in our simulation, L2 is modeled with a high miss rate. So a good percent of jobs (which are again mixed bunch of jobs) are reaching DRAM. And while DRAM is working on that some miss happens in L2 and that again makes an increment of the queue length of DRAM.
 - (c) System Response Time is having a stable value initially, but then it goes up reflecting the queues in L1, L2 and DRAM. And when the queue lengths of different components become stable, the System Response Time decrease again to a stable value.

Bibliography

- [1] Premkishore Shivakumar and Norman P. Jouppi , “CACTI 3.0: An Integrated Cache Timing, Power, and Area Model”
- [2] J. Huh, D. Burger, Stephen W. Keckler, “Maximizing Area Efficiency for Single-Chip Server Processors”
- [3] Smail MAR, Samy MEFTALI, Jean-Luc DEKEYSER, “Power Consumption Awareness in CacheMemory Design with SystemC,” *The 16th International Conference on Microelectronics, 2004. ICM 2004 Proceedings.*
- [4] Giovanni De Micheli, Yung-Hsiang L, “Adaptive Hard Disk Power Management on Personal Computers,” *IEEE Great Lakes Symposium on VLSI, 1999*
- [5] John L. Hennessy, David A. Patterson, “Computer Architecture: A Quantitative Approach, Third Edition,” *The Morgan Kaufmann Series in Computer Architecture and Design*
- [6] www.wikipedia.org
- [7] ALaRI Classnote, Prof. G. Serazzi
- [8] JMT 0.7.3 Manual

UNIVERSITY OF LUGANO
Advanced Learning and Research Institute -ALaRI

PROJECT

COURSE: PERFORMANCE EVALUATION

Shared-Memory Multiprocessor Systems – Hierarchical Task Queue

Mentor: Giuseppe Serazzi

Candidates:

Ana Jankovic, MAS
jankovia@alari.ch

Elena Zamsha, MAS
zamshae@alari.ch

Ghazal Haghani, MSc 2nd
haghanig@alari.ch

Lugano, February 2007.

CONTENTS

1. Description of the system	4
2. Introduction	5
3. Propose of the new technique	5
4. The modelling approach	6
4.1. Java Modelling Tools.....	6
4.2. Input parameters.....	7
4.3. Performance analysis	7
5. Simulations	8
5.1. Impact of access contention	9
5.1.1. Centralized organization	9
5.1.2. Distributed organization.....	10
5.1.3. Hierarchical organization.....	10
5.2. Impact of system size.....	11
5.2.1. Centralized organization	11
5.2.2. Distributed organization.....	11
5.2.3. Hierarchical organization.....	12
6. Results & Conclusions.....	13
7. REFERENCES:.....	14

LIST OF FIGURES

Figure 1 Shared-memory multiprocessors system	4
Figure 2 Hierarchical organization for N=8 processors with branching factor B=2	5
Figure 3 Task transfer process in the hierarchical organization for N=64 processors with branching factor B=4 and transfer factor Tr=1	6
Figure 4 Centralized organization with N=4 sinks	8
Figure 5 Distributed organization with N=4 sinks.....	8
Figure 6 Hierarchical organization with N=4 sinks, Br=2, Tr=1.....	9
Figure 7 Centralized, what-if analysis, N=4 sinks.....	9
Figure 8 Distributed, simple analysis, N=4 sinks	10
Figure 9 Hierarchical, simple analysis, N=4 sinks	10
Figure 10 Centralized, what-if analysis, N=8 sinks.....	11
Figure 11 Distributed, simple analysis, N=8 sinks	11
Figure 12 Hierarchical organization with N=8 sinks, Br=2, Tr=1.....	12
Figure 13 Hierarchical, simple analysis, N=8 sinks	12

1. Description of the system

Shared-memory multiprocessors are an important class of parallel processing systems.

Shared memory is a large block of Random Access Memory (RAM) which can be accessed by several different Central Processing Units (CPUs) in a multiple-processor computer system. Processors can access one or more shared memory modules. A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. The processors can be physically connected to the memory modules in a variety of ways, but logically every processor is connected to every module. Figure 1 presents basic concept of shared-memory multiprocessors system.

Processor scheduling is an important factor that influences the overall system performance and has received considerable attention from several researchers. On the other hand, sharing-memory multiprocessors system will cause some problems. The first issue is to decide how to dedicate memory between processors to obtain best performances of the system which can be represented by measuring its response time and utilization.

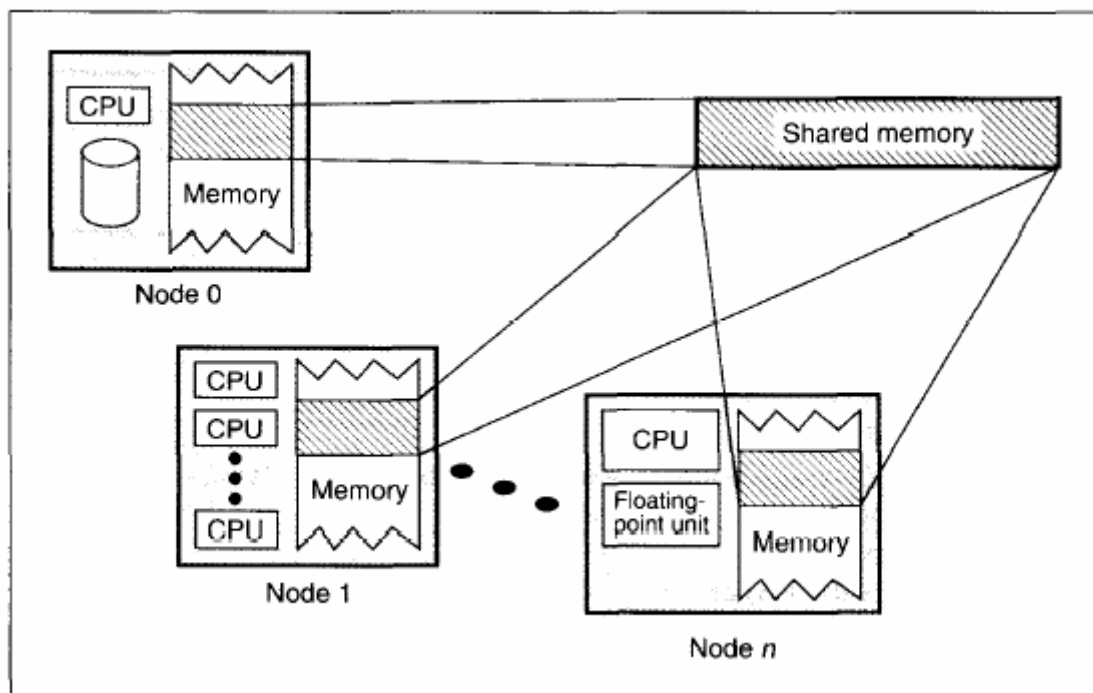


Figure 1 Shared-memory multiprocessors system

Several commercial and research shared-memory machines have been developed including BBN's Butterfly, NYU's Ultracomputer, IBM's RP3, Sequent's Balance and Symmetry, DEC's Firefly, and the Stanford Dash.

2. Introduction

Waiting ready tasks in shared-memory multiprocessors (the main topic of this course work) can be organized in two basic ways – centralized or distributed.

In the **centralized organization** there is a single global queue of ready tasks that is accessible to all processors in the system. However, due to mutually exclusive access, the single global task ready queue becomes the system bottleneck as the number of processors increases, so it seems that centralized organization is not suitable for large parallel systems.

In the **distributed organization**, on the other hand, local ready queues are associated with the processors; this avoids the problem of ready queue access contention but introduces additional problems. The main problem with this organization is to find an appropriate ready task queue for the arriving task (the task assignment problem). A simple random assignment strategy, in which an arriving task is simply assigned to a random ready task queue, causes load imbalance resulting in performance degradation. In the absence of ready task queue access contention, the centralized organization provides better performance mainly due to its load sharing characteristic.

3. Propose of the new technique

The goal is to present a superior technique – **hierarchical organization**, which provides performance similar to the centralized organization even when there is no access contention like in the distributed organization. In this new organization (example is shown in Figure 2), a set of ready task queues is organized as a tree with all the processors with their local queues attached to the bottom level of the tree (as leaf nodes).

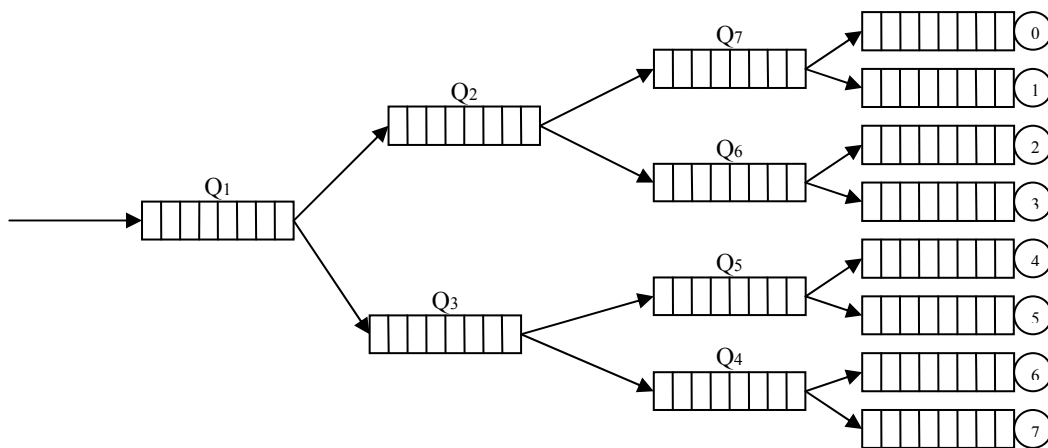


Figure 2 Hierarchical organization for N=8 processors with branching factor B=2

Each ready queue can be viewed as a ready queue in the centralized organization serving only the tree nodes directly below it; these nodes can themselves be ready queues or processors local queues. All incoming tasks are added to the root task queue. If L is the leaf node level, when processor is looking for work, it first checks its associated task queue at level (L-1). If that queue is empty it checks the parent node of this node at level (L-2) and the process is repeated up the tree until it finds a task to be scheduled (unless the root queue is empty).

In order to reduce access contention at higher levels, when a task queue is accessed, a set of tasks is moved one level down the tree – the size of set decreases progressively (at the bottom of the tree it is reduced just to one task). Parameter Tr is transfer factor which indicates the number of tasks transferred from a parent queue to its child queue.

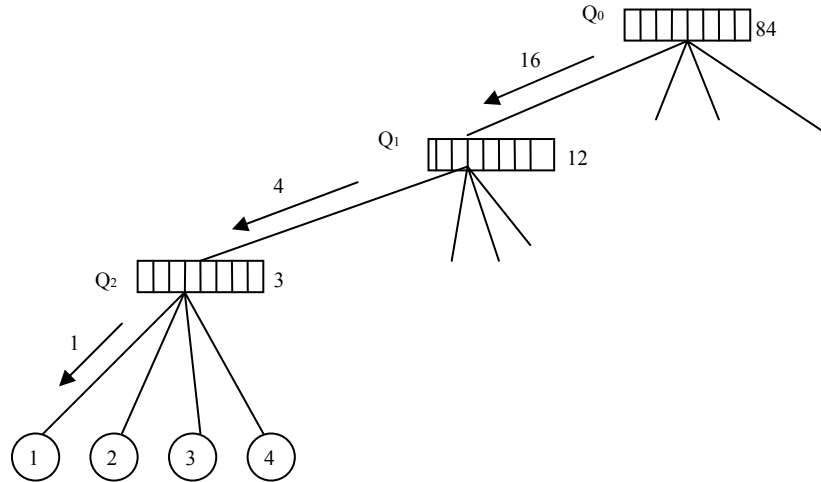


Figure 3 Task transfer process in the hierarchical organization for $N=64$ processors with branching factor $B=4$ and transfer factor $Tr=1$

Hierarchical organization avoids ready queue bottleneck because both the branching and transfer factors can be adjusted. It can also be seen that the set of task queues that form the tree can all be distributed to different memory modules so that concurrent access can be permitted (provided the interconnection network allows the particular permutation).

4. The modelling approach

In the following course work will be presented and analyzed performances of three different organizations for the shared-memory multiprocessor systems, mentioned upper.

4.1. Java Modelling Tools

JMT Simulator, also called as **JSIM**, is simulation module of the **Java Modelling Tools (JMT)**, an open-source fully-portable Java suite for performance evaluation, capacity planning and modelling of computer and communication systems. The suite implements numerous state-of-the-art algorithms for the exact, asymptotic and simulative analysis of queuing network models. JSIM is a discrete-event simulator for the analysis of queuing network models. An intuitive sequence of wizard windows helps specifying network properties. The simulation engine supports several probability distributions for characterizing service and inter-arrival times. Load-dependent strategies using arbitrary functions of the current queue-length can be specified. JSIM supports state-independent routing strategies, e.g., Markovian or round robin, as well as state-dependent strategies, e.g., routing to the server with minimum utilization, or with the shortest response time, or with minimum queue-length. The simulation engine supports several extended features not allowed in product-form models, namely, finite capacity regions (i.e., blocking), fork-join servers, and priority classes. The analysis of simulation results employs on-line transient detection techniques based on spectral analysis. What-if analyses, where a sequence of simulations is run for different values of parameters, are also possible.

4.2. Input parameters

We will consider a parallel system with N identical processors and model this system with an abstraction consisting of a set of queues and N sinks. For most results reported in this course work, we fixed N on 4 or 8 (partly because of the time needed to run simulation experiments and partly because many commercial multiprocessor systems use similar number of processors). Concentration is on the scheduling aspect of the system.

Generation of the jobs is a common feature among this three task queue organizations. However, contention for the task queue varies depending on the task queue organization. We will consider a simple **fork type of job structure**, assumed to be composed of a set of independent tasks that can be run on system concurrently. The job completes when all of its component tasks are completed. Tasks within the job do not communicate with each other. The **job arrival process** is assumed to be **exponentially distributed** with parameter $\lambda = 0.75$, and for what-if analysis we used range $[0.75, 1.50]$. Tasks are characterized by processor service demand with **mean $1/\mu$** , and $\mu = 1$.

We model the amount of time needed to access (not including the waiting time to get exclusive access to the ready queue) the ready queue as a **fraction f** of the average task execution time. We further assume that each time an idle processor accesses a ready queue it spends a constant amount of time (i.e., f/μ) independent of the actual number of entries in the ready queue i.e., ready queue access time is deterministic. This is true for simple scheduling algorithms such as the **first-come-first-served (FCFS)** where it simply involves removing the task at the head of the queue. Fraction $f=10\%$ causes the increasing of the service time, which we model with **constant distribution** with **coefficient 1.1**.

We determine our models also as FCFS, and **load independent**. **Routing of tasks** in the system is chosen to be **random**. **Queue length is finite** and fixed on **20**, and described models use **waiting queue, without dropping the tasks**.

Model is described as **one-open-class system**, which means that we will have only one class of the tasks arriving randomly into our system.

4.3. Performance analysis

To compare these three different organizations, we decided to make analysis over two important output parameters – **Response time** and **Utilization**, presenting the results in two sections:

- impact of access contention, and
- impact of system size.

Response time means average time spent by job before leaving the centre (queuing + in service). Utilization means average number of jobs in service.

Impact of access contention will be analysed and the results will be presented with the diagram describing the impact of the ready queue access time as a function of utilization. The analysis for the centralized organization will be done by varying the parameter λ implementing this in what-if analysis, and for the other two organizations, we will fix this parameter on 0.75, expecting the best result.

Impact of system size, will be analysed with the double size of the processors ($N=8$), so we will compare results for $N=4$ and $N=8$ processors in all three distributions. Results will be also presented with the diagrams describing job response time and utilization when the average task service time is doubled.

5. Simulations

First step is to build three different models for centralized, distributed and hierarchical organization in JSIM tool. They are presented in Figures 4, 5 and 6, respectively.

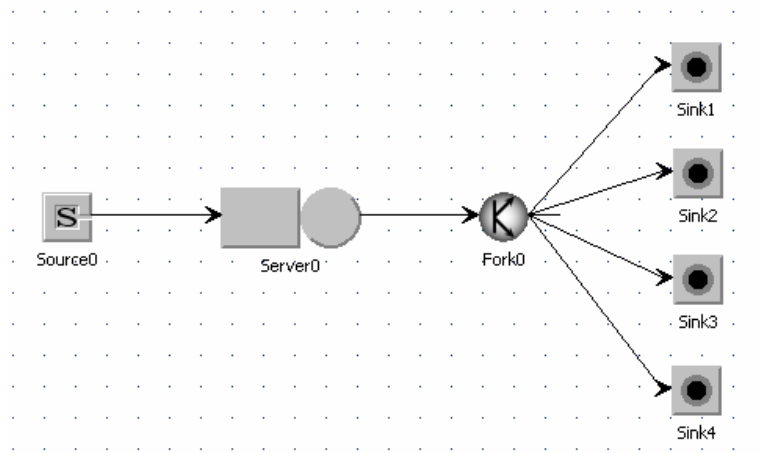


Figure 4 Centralized organization with $N=4$ sinks

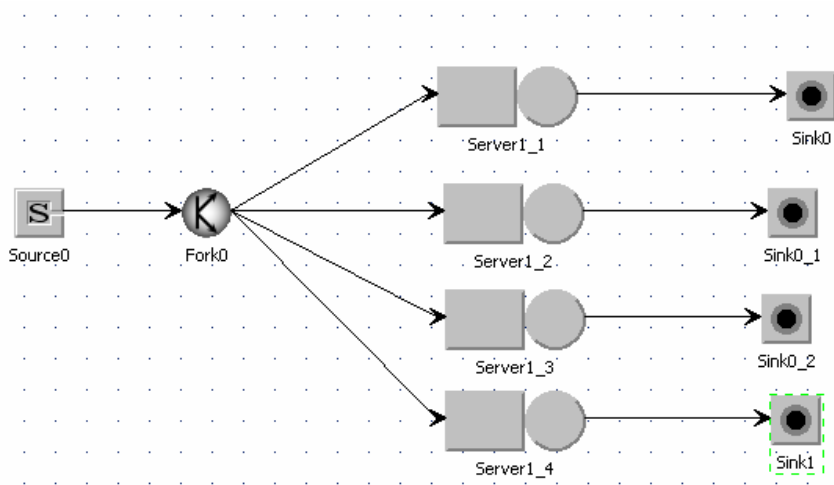


Figure 5 Distributed organization with $N=4$ sinks

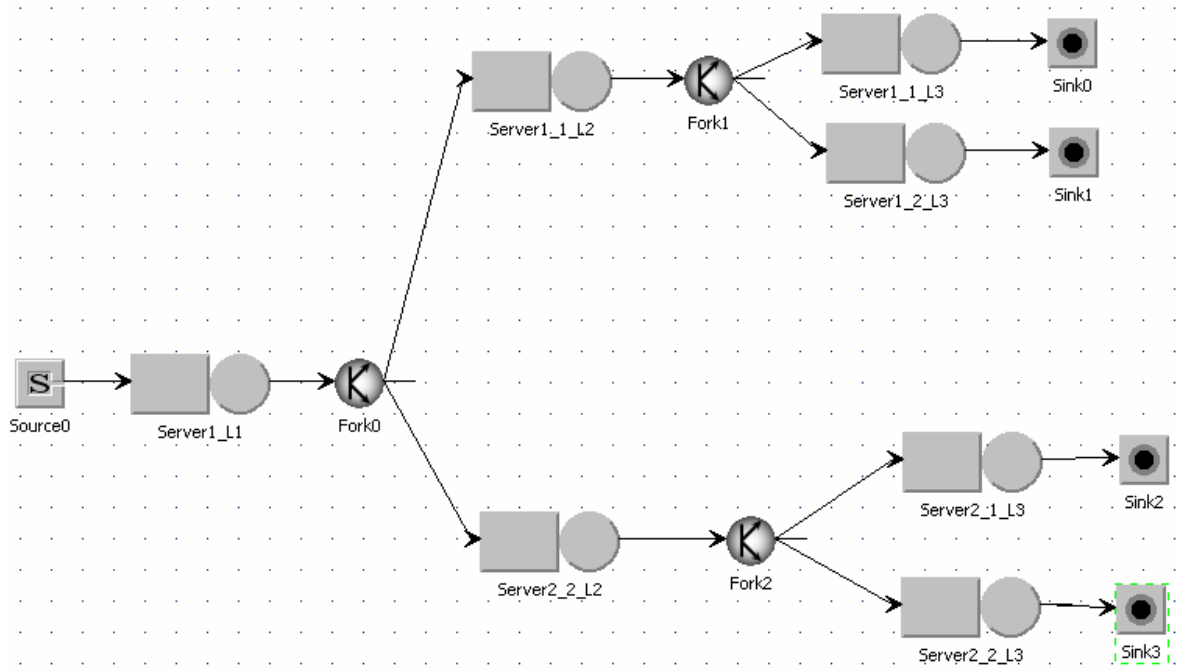


Figure 6 Hierarchical organization with N=4 sinks, Br=2, Tr=1

Second step is to give to all used blocks (source, servers and forks) needed values for the **job arrival distribution parameters** (exponential: $\lambda = 0.75$), **queue lengths** (20 tasks), **service time distribution parameters** (constant: $c=1.1$) and **simulation time** (6 seconds).

5.1. Impact of access contention

5.1.1. Centralized organization

For the centralized model we used **what-if analysis** choosing the **Arrival time** as the relevant parameter with λ in the range of [0.75,1.5] tasks/second (exponential distribution of the jobs). In this range we run simulations in four steps. Area of interest was to analyze response of the system from 4 to 8 tasks, regarding the number of sinks at the output of the model, because it was interesting to watch the performance for the same, or double number of tasks comparing how much we have sinks in the model (N=4).

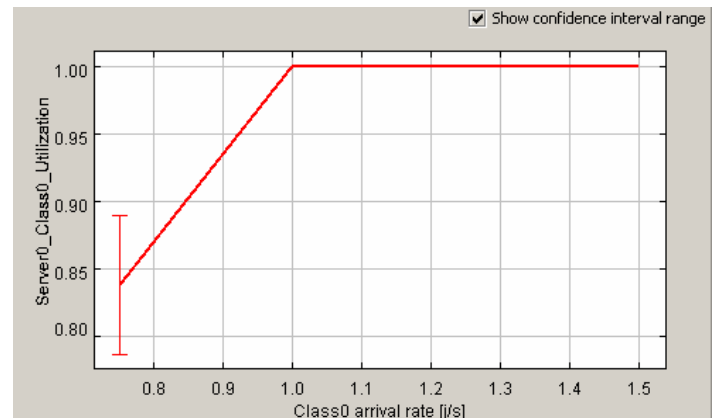
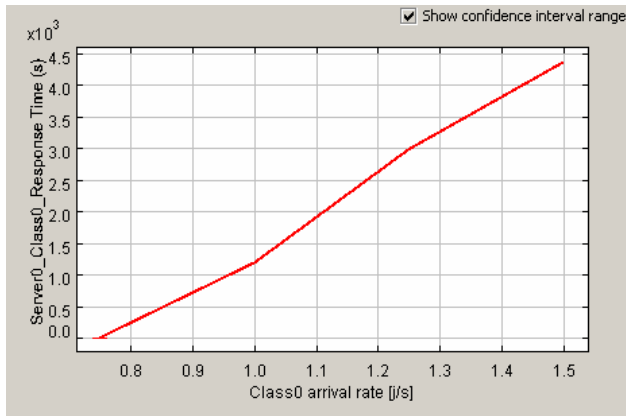


Figure 7 Centralized, what-if analysis, N=4 sinks

5.1.2. Distributed organization

For the distributed model we used simple analysis and simulated the model. Duration of the simulation was also 6 seconds. In Figure 8 on the left side is represented Response time, and on the right side is Utilization.

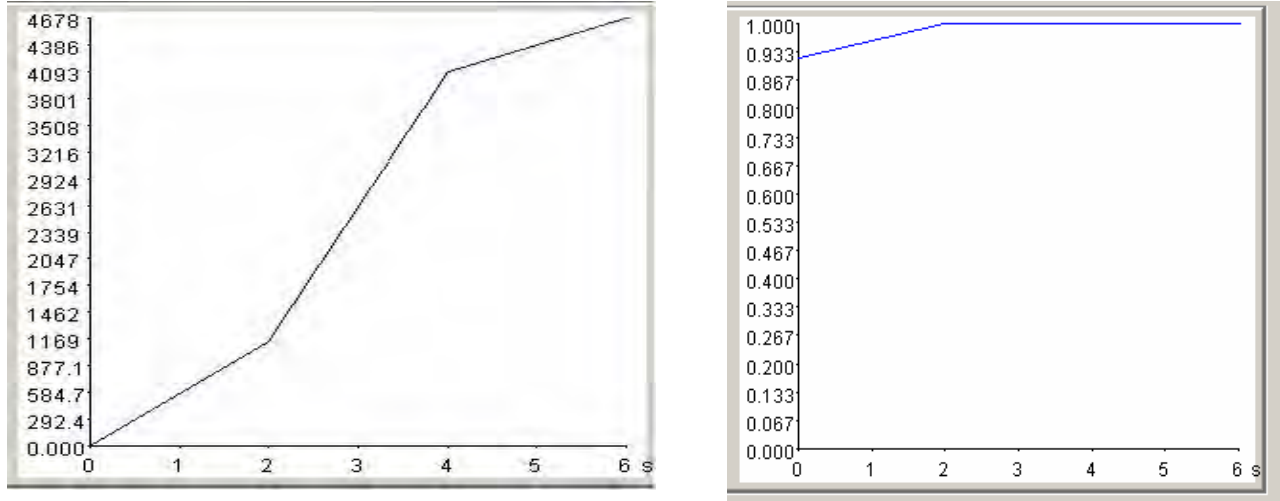


Figure 8 Distributed, simple analysis, N=4 sinks

5.1.3. Hierarchical organization

For the hierarchical model we used simple analysis and simulated the model. Duration of the simulation was 4 seconds long. We wanted to see how will model work for the similar number of the tasks, as sinks we have at the output; $4\text{sec} / (0.75\text{tasks/sec}) = 5.3$ tasks. In Figure 9 on the left side is represented Response time, and on the right side is Utilization.

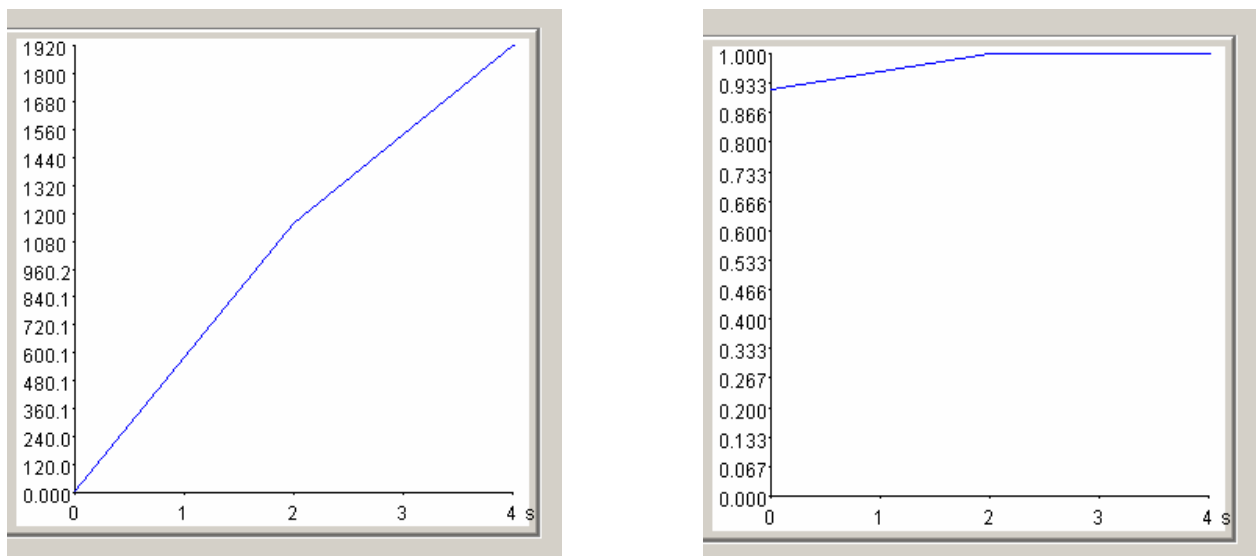


Figure 9 Hierarchical, simple analysis, N=4 sinks

5.2. Impact of system size

5.2.1. Centralized organization

For the centralized model with $N=8$ sinks at the output of the model, we used again **what-if analysis** choosing the **Arrival time** as the relevant parameter with λ in the range of $[0.75, 1.5]$ tasks/second, and run simulations in four steps. Area of interest was to analyze response of the system from 4 to 8 tasks, regarding the number of sinks at the output of the model, because now it was interesting to watch the performance for the half or the same number of tasks comparing how much we have sinks in the model ($N=8$).

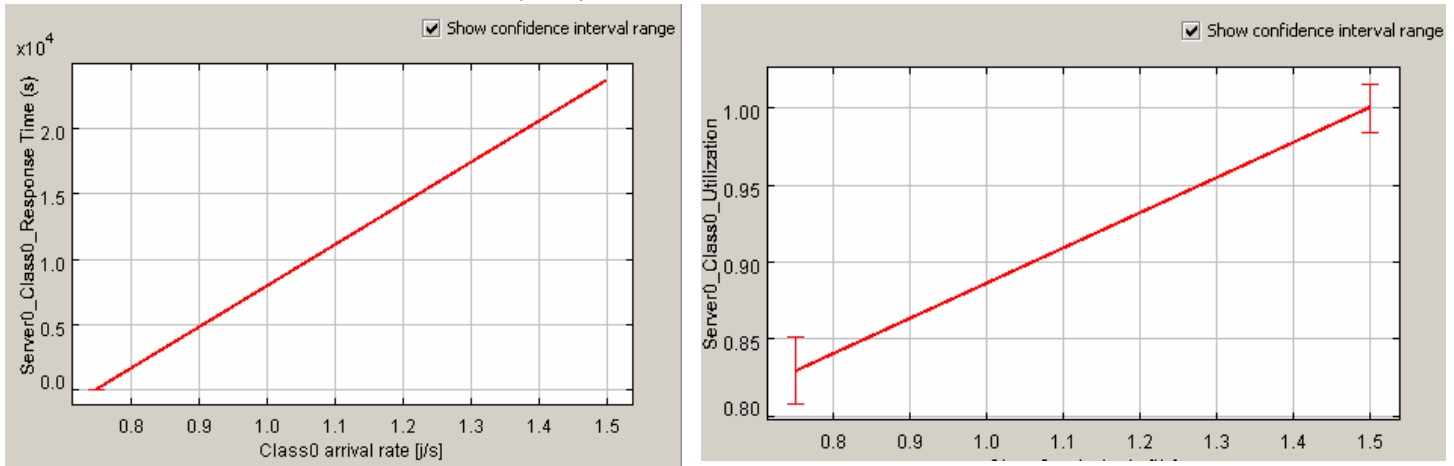


Figure 10 Centralized, what-if analysis, $N=8$ sinks

5.2.2. Distributed organization

For the distributed model with $N=8$ sinks at the output we used simple analysis and simulated the model. Duration of the simulation was 6 seconds, regarding the number of the tasks in the system, because we wanted to have the same number of tasks as sinks at the output; $6\text{sec} / (0.75\text{tasks/sec}) = 8$ tasks ($N=8$). In Figure 11 on the left side is represented Response time, and on the right side is Utilization.

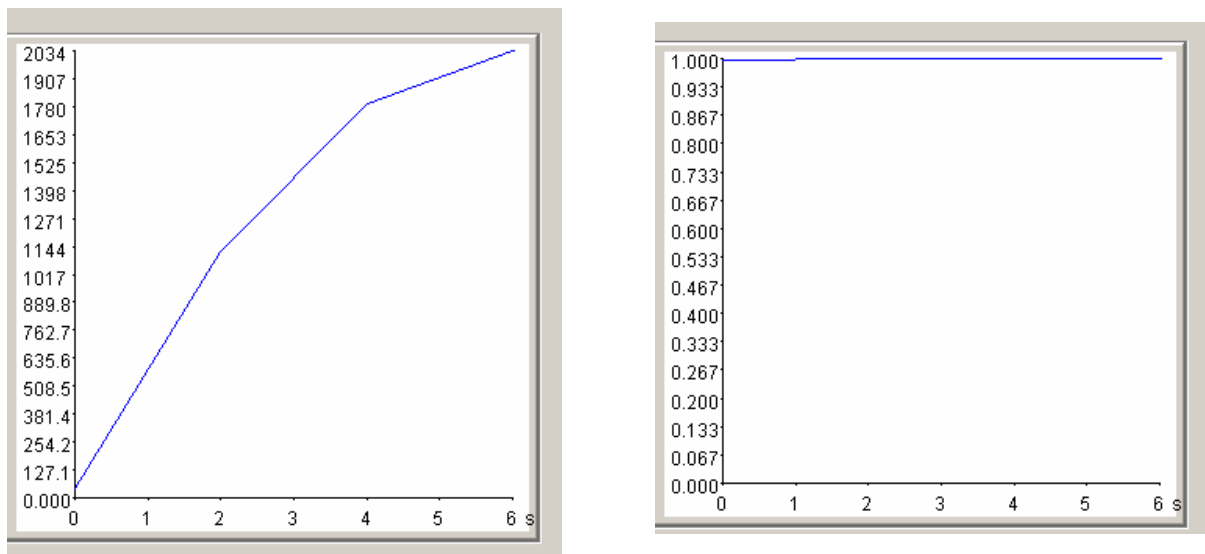


Figure 11 Distributed, simple analysis, $N=8$ sinks

5.2.3. Hierarchical organization

In Figure 12 is presented model of the hierarchical distribution with N=8 sinks. On this picture it is also possible to see the work-sheet of the JSIM tool.

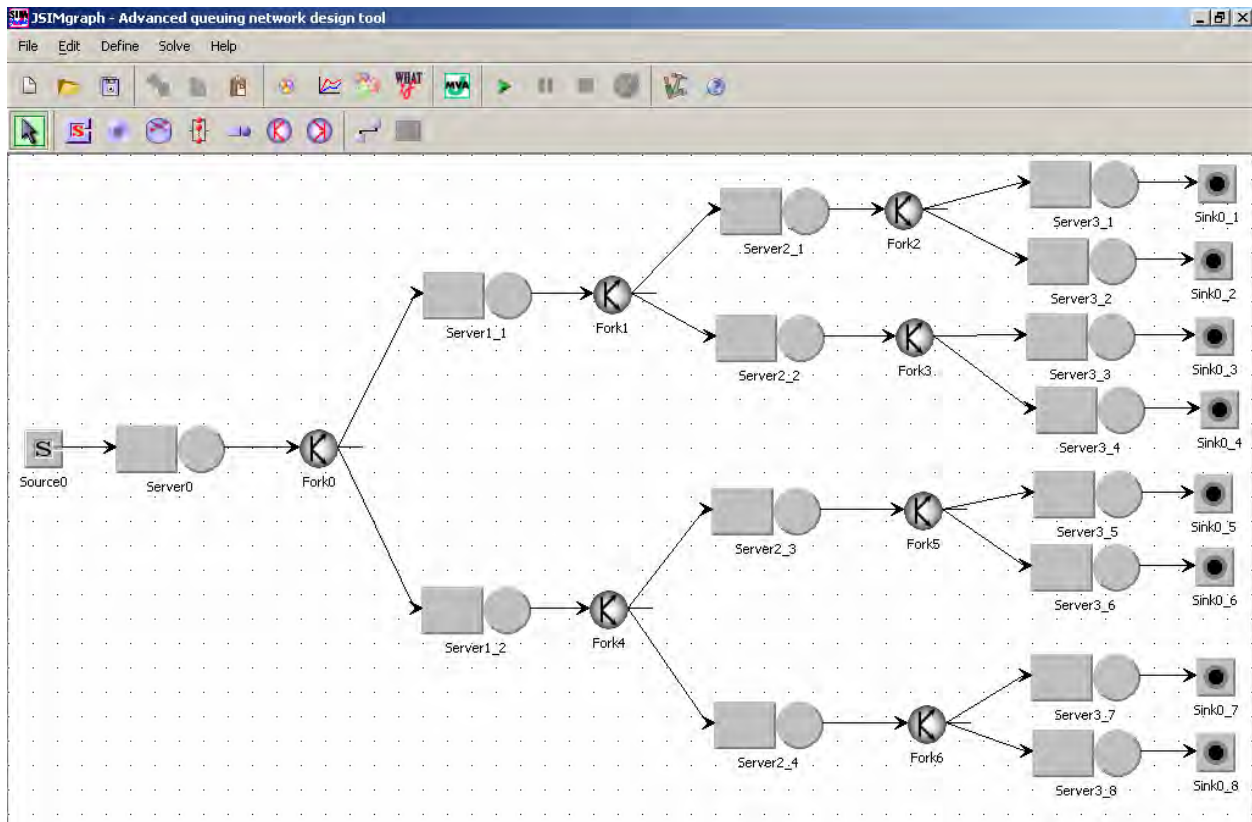


Figure 12 Hierarchical organization with N=8 sinks, Br=2, Tr=1

For this analysis we have chosen again simple analysis, 12 seconds long, because we wanted to compare how will this model work with double number of the tasks, comparing with the number of sinks in the model; $12\text{sec} / (0.75\text{tasks/sec}) = 16$ tasks, that means $2*N$. In Figure 13 on the left side is represented Response time, and on the right side is Utilization.

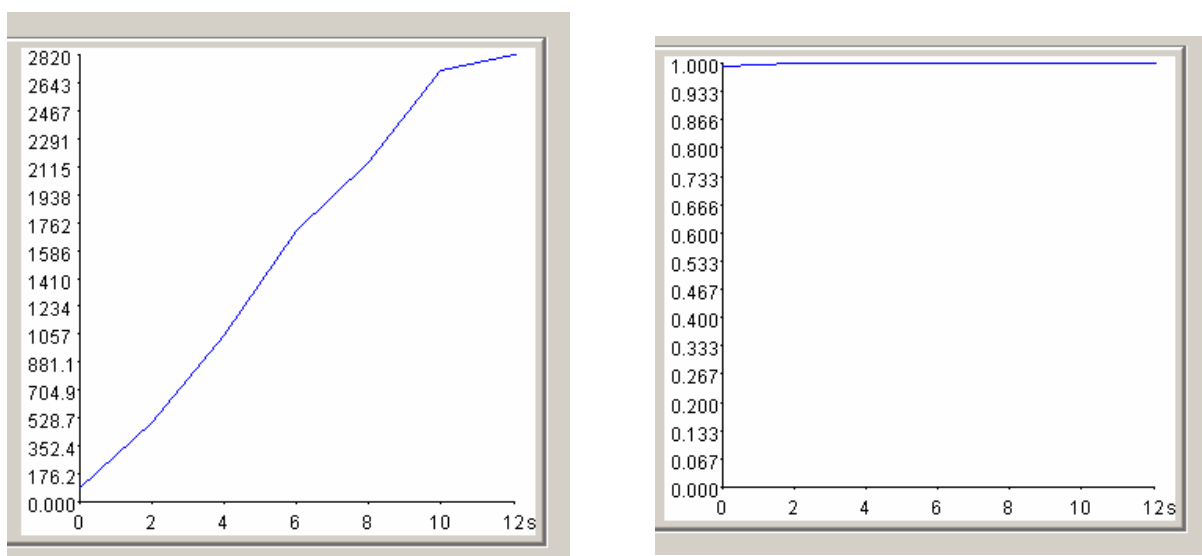


Figure 13 Hierarchical, simple analysis, N=8 sinks

6. Results & Conclusions

Comparing three different techniques for modelling waiting ready tasks in shared-memory multiprocessors, which were described, analyzed and simulated in this course work, we can conclude that centralized organization showed very good results for small number of processors. Response time was growing rapidly and utilization reached value of 1 in the short period. In organizing ready tasks, we can say that centralized task queue organization is preferred in the absence of access contention. However, the major drawback is that the central task queue becomes a bottleneck for large system sizes and therefore it is not suitable for large parallel systems or fine granularity task scheduling.

Introducing more processors in the system, centralized model was not good representative anymore, the utilization couldn't reach value of 1 in provided range for job arrival distribution, but on the other hand, distributed organization brought better results. With distributed organization we eliminated the task queue bottleneck disadvantage of the centralized organization by associating a local task queue for each processor. A main disadvantage of this organization is that there may be a load imbalance in the sense that some task queues may be empty while there are tasks waiting to be scheduled in other queues.

In the moment of making last simulations, and providing detailed performance analysis, we could conclude, finally, that in the both situations, with small and doubled number of processors (in the case of the JSIM model – sinks), **hierarchical approach is the best choice**. For $N = 4$ sinks at the output of the presented system, hierarchical model was good enough like the centralized, providing similar performances while eliminating the ready queue bottleneck. The properly designed hierarchical organization inherits the load sharing property of the centralized organization, while distributing the task queues as in the distributed organization. Mainly due to its load sharing property, the hierarchical organization exhibits less sensitivity to task service time variance like centralized organization in absence of access contention.

On the other hand, for $N = 8$ sinks hierarchical model was giving the best performances, even comparing with the distributed model. The results showed that both, hierarchical and distributed, organizations scale nicely for all system utilizations without creating system bottlenecks, but hierarchical approach retains its performance advantage over distributed one.

This new approach, hierarchical organization, presented in our course work, has also good characteristics in the case of larger number of the incoming jobs (or tasks) in the system, comparing with the number of the processors (in fact sinks) in the model.

Queue problems are optimized very well - implementing hierarchical organisation for modelling waiting ready tasks in shared-memory multiprocessors, problem of ready queue access contention and load imbalance problem now became negligible.

7. REFERENCES:

- [1] E. Lazowska, J. Zahorjan, *Quantitative System Performance - Computer System Analysis Using Queuing Network Models*, Prentice-Hall, Inc., New Jersey
- [2] S. Dandamudi, P. Cheng, *A Hierarchical Task Queue Organization for Shared-Memory Multiprocessor Systems*, IEEE transactions and distributed systems, vol. 6, No 1, January 1995
- [3] B. Nitzberg, V. Lo, *Distributed Shared Memory: A Survey of Issues and Algorithms*, University of Oregon, Oregon
- [4] B. Sinclair, *Analysis of Shared Memory Multiprocessors*, Version 2.3, United States, Jun 2005
- [5] M. Bertoli, G. Casale, G. Serazzi, *The JMT Simulator for Performance Evaluation of Non-Product-Form Queuing Networks*, Politecnico di Milano, Italy
- [6] *Tutored by G. Serazzi, Java Modelling Tools, users manual*, Version 0.3, Performance Evaluation Lab, Politecnico di Milano, Italy, June 2006
- [7] *Tutored by G. Serazzi, Java Modelling Tools, system manual*, Version 0.1, Performance Evaluation Lab, Politecnico di Milano, Italy, June 2006

4 – Multimedia

4.1 – Modelling a Surveillance System	150
4.2 – Performance Evaluation Report on VoIP Gateway Systems	161

POLITECNICO DI MILANO

DIPARTIMENTO DI ELETTRONICA E INFORMAZIONE



**MODELING OF A SURVEILLANCE SYSTEM
USING JMT ENVIRONMENT**

LUCA BERTOSSI
LAURA FRIGERIO

October 25, 2006

Doctoral course on Advanced Topics of Performance
Evaluation

Tutor: Prof. G. Serazzi

1 Problem definition

Aim of the work is to describe a surveillance system composed of several videocameras and service stations. System should monitor some critical points of a city, in order to identify possible dangerous situations.

Each critical point is equipped with videocameras and a local service station that could analyze data coming from videocameras and transmit results to a central DB server. Videocameras are triggered each time a motion is detected in their field of sight and record a stream of 5-seconds length. Streams are encoded in MPEG and transmitted to the local service station that decompresses the video and analyzes the sequence. Results of the analysis are sent with the compressed video to a central DB server. The local service stations have a limited amount of space to queue data streams coming from videocameras.

Five city critical points are examined: Dome area, Central Station area, University area, Airport area and Stadium area. Several videocameras are placed in each area and classified according to the average number of motions detected in a time interval.

Videocameras that monitor crowded areas usually produce more complex streams requiring more time to be analyzed by the local service stations. For each area three videocameras groups are considered: those which monitor a crowded space, those which monitor a quite crowded one and those which monitor a slightly crowded one.

2 Numerical data

Average times between the sending of two data streams from a group of videocameras to the local service stations are classified in table 1. Average times are tabled with reference to the area and to the videocamera class.

	Dome	Station	University	Airport	Stadium
Crowded	18	21	17	15	20
Quite Crowded	35	37	30	28	30
Slightly Crowded	100	80	110	75	75

Table 1: Average times between the sending of two data streams (in seconds)

The elaboration time at the local service station is composed of two parts: the time for the MPEG decompression and the time for the image analysis. As explained before the elaboration time is dependent on the videocamera class. Table 2 shows the average times. There are light differences according to the hardware installed at each station.

The number of streams that could be stored in each local service station are limited. Their values are shown in table 3.

Service time at the DB service station is similar for all requests, with an average value of 2 seconds. The Central DB is composed of two servers.

	Dome	Station	University	Airport	Stadium
Crowded	13	13	8	13	10
Quite Crowded	8	8	7	8	7
Slightly Crowded	5.5	5.5	5.2	5.5	5.2

Table 2: Times of elaboration at local service station (in seconds)

	Number of Data Streams
Dome	500
Station	300
University	200
Airport	400
Stadium	600

Table 3: Local service station capacity

3 Model definition with JMT

The JModel Tool can be used to model the described situation. A tree structure can be identified: 5 local service stations collect requests and then transmit them to the Central DB Server.

Focusing the attention on a local service station we can see that it collects data from several videocameras grouped into 3 classes. Each class has its own average time between the sending of two data streams and its own average elaboration time. Therefore we can model this behaviour considering a local source composed of three classes that sends request to the local service station. As no other information except time between two data streams is available, we can model the situation with an exponential distribution with medium interarrival time equal to the time between the sending of two data streams.

Each service station can be modeled as a server that can store only a finite number of data streams. To model this situation we insert a finite capacity region for the local service station. Each stream that causes the queue to exceed the local capacity is dropped. Server services each class with a different mean time. Also in this case an exponential distribution is used.

These considerations could be extended to every local service station. Mean values used for the exponential distributions of requests and server elaborations are inserted according to tables 1 and 2. Server capacities are those shown in table 3.

Finally a Central DB Server is included to collect all the requests coming from the Local Service Stations. The Central DB Server has a service time described with an exponential distribution with a mean value equal to 2 seconds.

Figure 1 shows the graphical representation of the model.

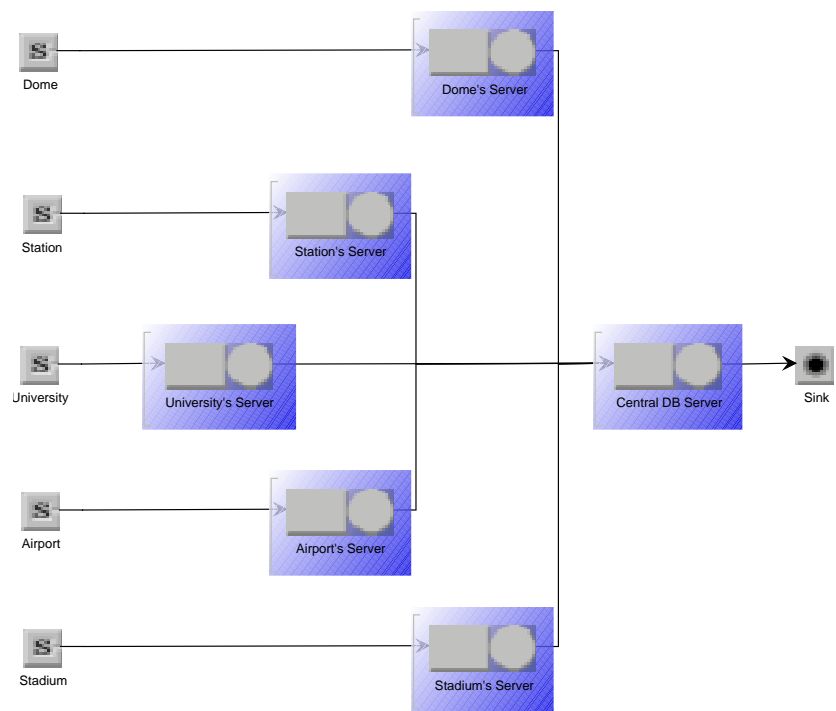


Figure 1: JModel System Model graph

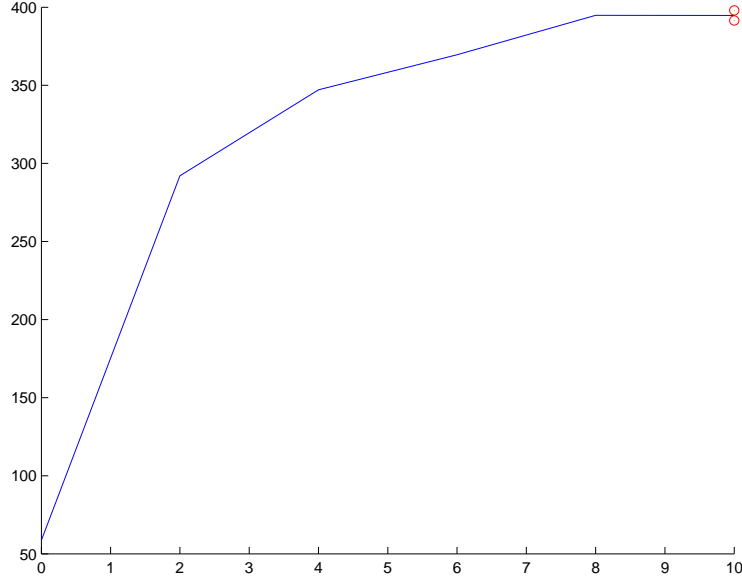


Figure 2: Airport Server Queue length

4 Model analysis

In order to understand the model behaviour we investigate some simulation results. Queue length, Response time, Utilization and Throughput are examined for each local service station. Data for all stations are available from the software model. In the following sections we report results for some of the stations.

4.1 Queue length

Figures 2, 3 and 4 show queue dimensions for three local service stations (Airport, Stadium, DB). As it can be noticed, queue lengths for Stadium and DB servers are quite limited (maximum 5 requests queued) while the Airport server queue is near to its upper bound, i.e. 400 requests. This effect is due to the service time at the local station. Indeed, while average time between two requests is similar for both servers (ref. Table 1), service time for the Airport Server is greater (reg. Table 2). Very small differences of this values impact greatly on queue length.

4.2 Response Time

Figures 5, 6 and 7 show response time for three local service stations (Airport, Stadium, DB). Also in this case it can be noticed that response times for Stadium and DB servers are quite limited (maximum 55 seconds) while the Airport server response time is very high. This effect is due to the queue saturation that impacts on the average time for the request to be dispatched.

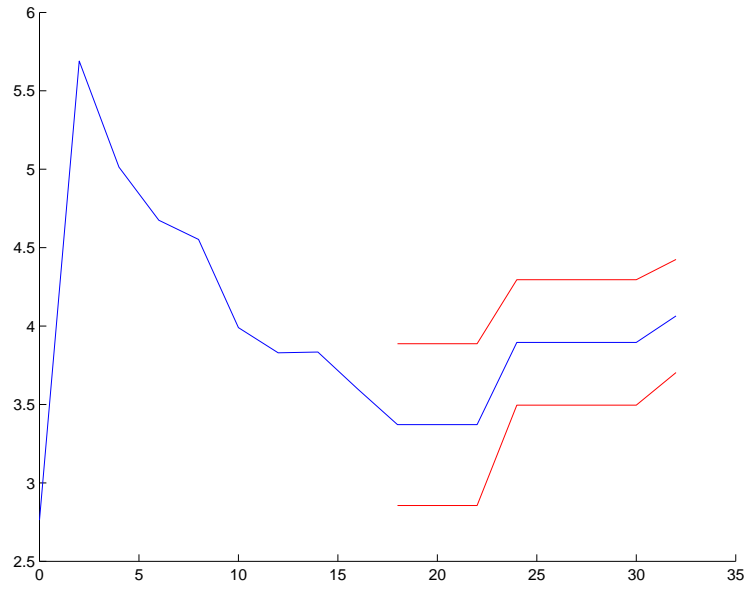


Figure 3: Stadium Server Queue length

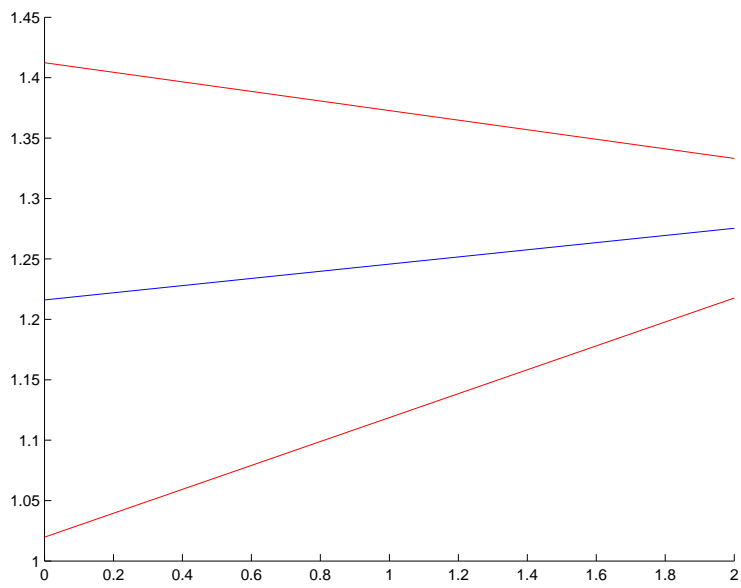


Figure 4: DB Central Server Queue length

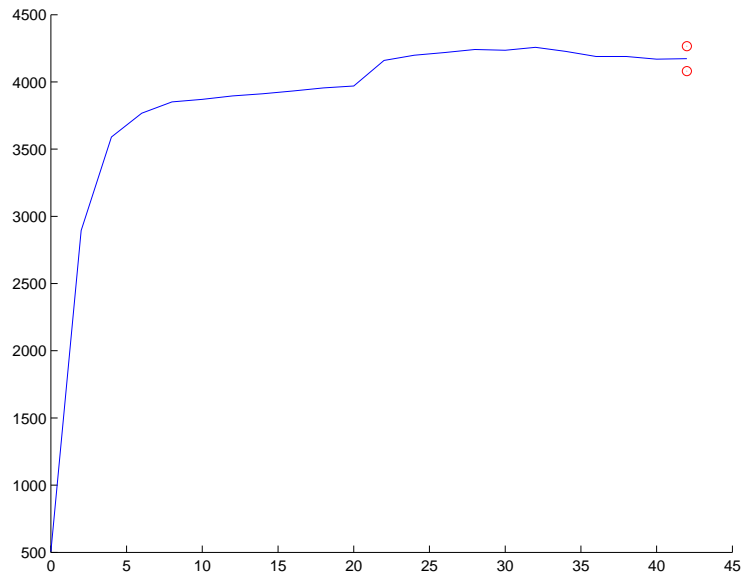


Figure 5: Airport Server Response Time

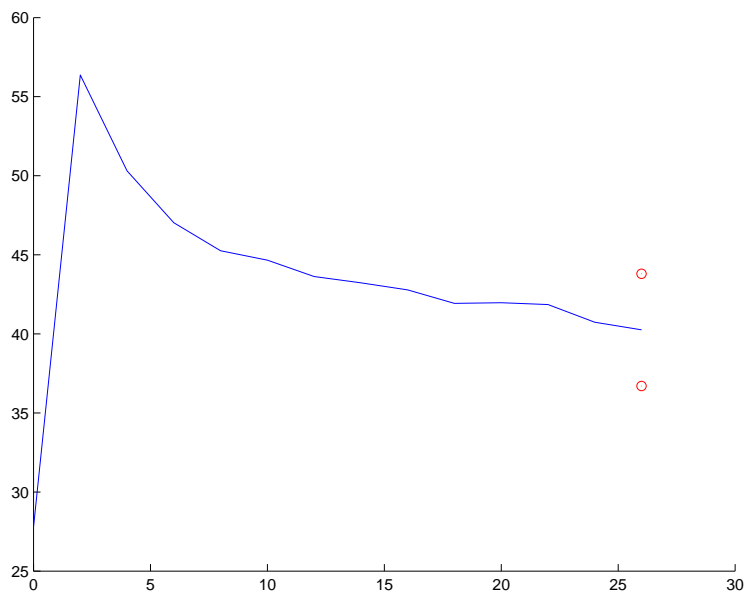


Figure 6: Stadium Server Response Time

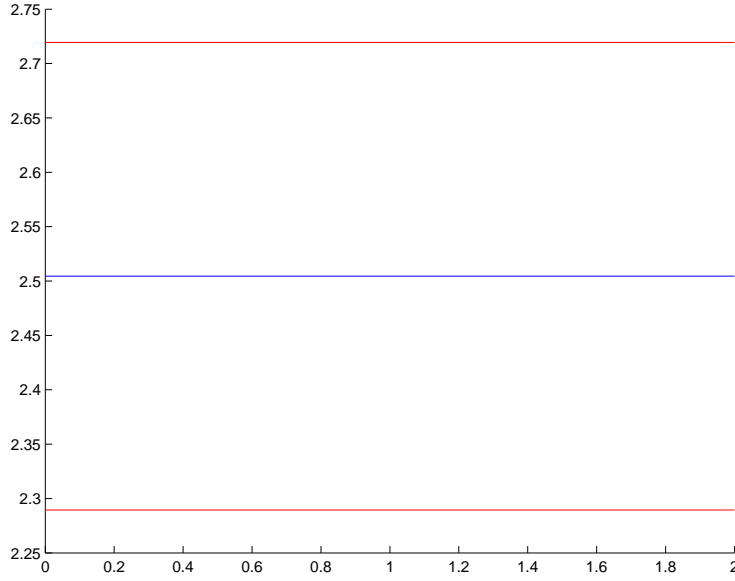


Figure 7: DB Server Response Time

	Dome	Station	University	Airport	Stadium
Sum of arrival rates	0.095	0.087	0.101	0.116	0.096
Average Throughput	0.095	0.092	0.100	0.096	0.101

Table 4: Comparison between arrival rate and throughput

4.3 Utilization

Utilization results show how the resources are exploited. As it can be guessed, the airport server station is always busy. For this reason it cannot dispatch all requests, leading the queue length to its maximum value and the response time to high values, as shown in the previous sections. Figure 8 shows the trend of the Utilization for the Airport server compared with the mean values of Stadium and DB Server.

4.4 Throughput

Throughputs of Airport, Stadium and DB servers are plotted in figure 9. We can compare the throughput to the arrival rate in order to understand the behaviour of the system. Summing up lambdas of the exponential distributions for the three source classes of a server ($\sum_{i=0}^3 \lambda_i$) and comparing this value with the medium throughput resulting from simulations, we should obtain similar values to be in equilibrium; results are shown in table 4.

We can notice that sum of lambda and average throughput are similar for the most of servers. Therefore at the equilibrium all requests are dispatched.

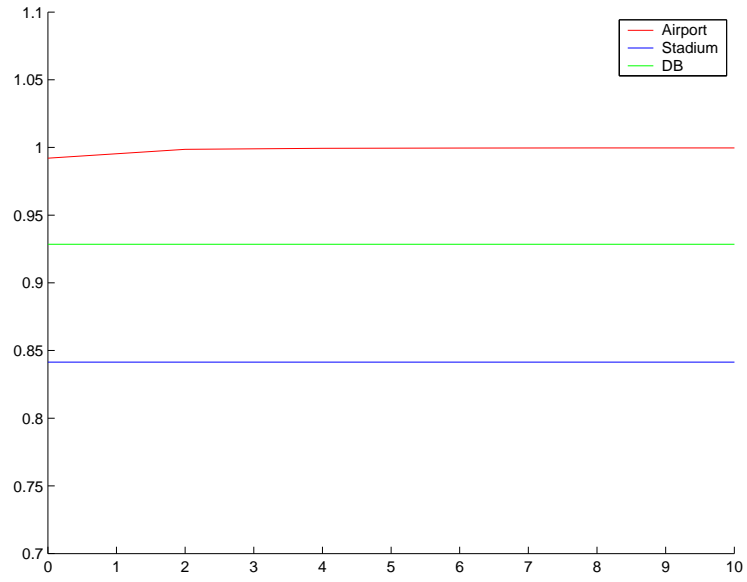


Figure 8: Comparison of the utilization of Airport server with the mean values of stadium and DB servers

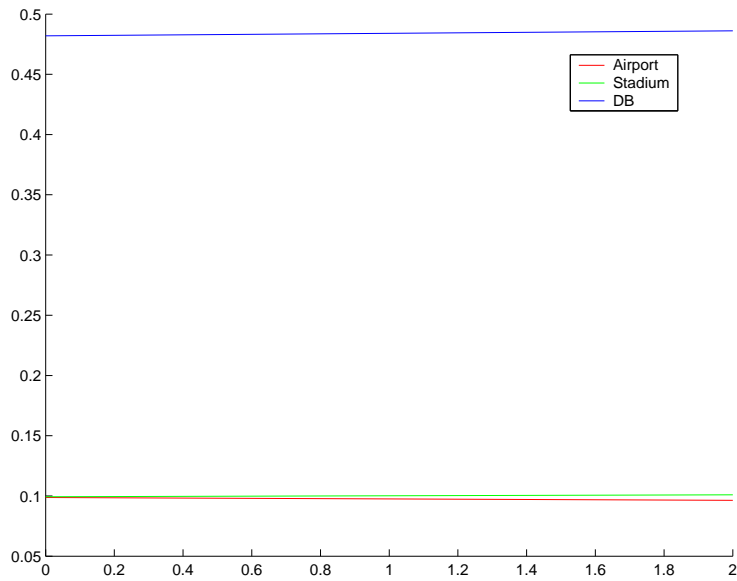


Figure 9: Throughput comparison

	Dome	Station	University	Airport	Stadium
Crowded	13	13	8	9	10
Quite Crowded	8	8	7	7	7
Slightly Crowded	5.5	5.5	5.2	5.2	5.2

Table 5: Updated times of elaboration at local service station (in seconds)

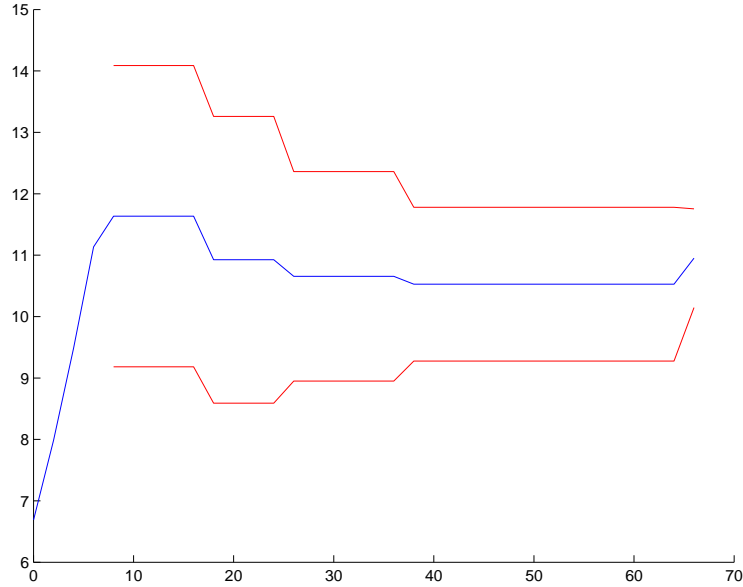


Figure 10: Updated Airport Server Queue Lenght

For the Airport server, we can notice that the first value is greater than the second. In this case the server saturates and some requests are dropped as they overcame queue capacity.

5 Model modification

As shown in the previous section, the Airport Server, could not manage all received requests. In order to improve the system performance we slightly changed service response times of the airport server. This corresponds to a hypothetical upgrade of the server hardware. In table 5 updated values are reported.

With this new configuration the server is able to manage all requests. We obtain an average Utilization of 0.931 and an average Throughput of 0.1085. Figure 10 and 11 shows the improvement obtained for queue length and response time.

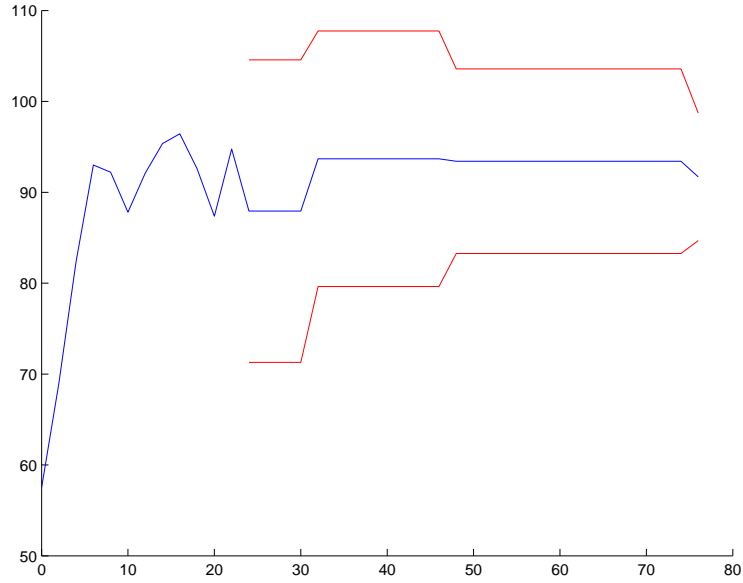


Figure 11: Updated Airport Server Response Time

A Support Matlab functions for graphics generation

JMT deal with data structures building a XML file that defines the model architecture, the parameters and the simulation results. In order to easily manage results obtained from simulations we developed a set of Matlab functions to extract values from the XML file with `jmodel` extension. Functions are based on GPL XMLTree toolbox freely downloadable from Matlab center site.

Three functions have been developed:

- JMTDraw: it plots and/or save to file simulation results.
- JMTRread: it reads simulations results and plot them using JMTDraw function
- JMTLoad: it loads from `jmodel` file simulations results and creates a Matlab structure array to store data

Functions are provided with the source code and are fully commented.

PERFORMANCE EVALUATION REPORT ON VoIP GATEWAY SYSTEMS

Submitted on 05, February, 2005

Presented To:

**Prof. Giuseppe Serazzi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano
Italy**

Presented By:

**Dorosh Oleksander
Kaveripakam Sathish Chandra Kumar
Xu Liu Fang**

**Advanced Learning & Research Institute(AlaRI)
University of Lugano
Via Lambertenghi 10A, 6904 – Lugano
Switzerland**

CONTENTS

1. Introduction:.....	4
1.1 Analog-Digital:	4
1.2 Compression:	5
1.3 IP Conversion:	5
1.4 Extraction of Digitized Voice from IP:.....	5
1.5 De-compression:	5
1.6 Digital – Analog:.....	5
2. ARCHITECTURE 1 :.....	6
2.1 CODEC :.....	6
2.2 DSPx(DSP1,DSP2,DSP3) :	7
2.3 Packet Processor :	7
2.4 Network MicroController:	7
3. ARCHITECTURE 2 :.....	8
3.1 CODEC :.....	8
3.2 HIGH END NETWORK PROCESSOR :	8
4. PERFORMANCE EVALUATION PARAMETERS :	9
4.1 Description of Architecture 1 :.....	9
4.1.1 Queue 1 , 1.1 , 1.2 :.....	9
4.1.2 Queue 2:.....	9
4.1.3 Queue 3:.....	11
4.1.5 Queue 4	11
4.1.6 Queue 5:.....	11
4.1.7 Queue 6:.....	11
4.1.8 Queue 7:.....	11
4.1.9 Queue 8,8.1,8.2:.....	11
4.2 Description of Architecture 2 :.....	11
4.2.1 Queue 1 :.....	11
4.2.2 Queue 2:.....	12
4.2.3 Queue 3:.....	12
4.3 Parameters – Architecture 1:.....	14
4.3.1 CALL 1:.....	14
4.3.2 DSP 1,DSP 2,DSP 3 :	14
4.3.3 Processor :	15
4.3.4 Netcontrol:	15
4.3.5 MEASUREMENT CRITERIA.....	15
4.4 Parameters – Architecture 2:.....	15
4.4.1 CALL :.....	16
4.4.2 CPU:	16
4.4.3 MEASUREMENT CRITERIA:.....	16
5.Conclusion	16
6. Limitations & Assumptions	17
7.References	17
APPENDIX A:.....	17
MATLAB.....	17
8.1 MATLAB CODE – Architecture 1.....	17
8.1.1 Equal Routing Probability , $s11=s12=s13>s2>s3$	17
8.1.2 Equal Routing Probability , $s11=s12=s13>s2>s3$ (Service Times are halved).....	18
8.1.3 $s11 = s12 = s13 > s2 > s3$, Routing : $r1=2* r2=3* r3$	19
8.1.4 Equal Routing Probability, Routing : $s11>s12>s13$	20
8.1.5 $r1=2 r2= 4r3$, $s11>s12>s13$	21
8.1.6 $r1=2 r2=3r3$, $s11>s12>s13$	22

8.1.7 $r1=2$ $r2=3r3$, $s11<s12<s13$	23
8.1.8 $r1= r2= r3$, $s13>s12>s11$	24
8.1.9 $s11=s12=s13$, $s11 >s2>s3$, $r1=r2=r3$	25
8.1.10 $s11=s12=s13$, $s11<s2>s3$ $r1=r2=r3$	26
8.1.11 $r1=2$ $r2=3r3$, $s11=2s12=3s13$	27
8.1.12 $r1=2$ $r2=3r3$, $s11=2s12=3s13$	28
8.2 MATLAB CODE – Architecture 2.....	29
8.2.1 $S = 0.255$, $x=1:0.1:4$	29
8.2.2 $S = 0.255/2$, $x=1:0.1:9$	30
8.2.3 $s3=0.255$, $x=1:0.1:7.5$	31
9. TABULAR RESULTS :	32
9.1 Win Modelling Tool - ARCHITECTURE 1.....	32
9.1.1 Equal Routing Probability , $s11=s12=s13>s2>s3$	32
9.1.2 Equal Routing Probability , $s11=s12=s13>s2>s3$ (Service Time halved).....	33
9.1.3 $s11 = s12 = s13 > s2 > s3$, Routing : $r1=2* r2=3* r3$	34
9.1.4 Equal Routing Probability, Routing : $s11>s12>s13$	35
9.1.5 $r1=2$ $r2= 4r3$, $s11>s12>s13$	36
9.1.6 $r1=2$ $r2=3r3$, $s11>s12>s13$	36
9.1.7 $r1=2$ $r2=3r3$, $s11<s12<s13$	37
9.1.8 $r1= r2=r3$, $s11<s12<s13$	38
9.1.9 $s11=s12=s13$, $s11 >s2>s3$, $r1=r2=r3$	39
9.1.10 $s11=s12=s13$, $s11<s2>s3$ $r1=r2=r3$	40
9.1.11 $r1=2$ $r2=3r3$, $s11=2s12=3s13$	41
9.1.12 $r1=2$ $r2=3r3$, $s11=2s12=3s13$	42
9.2 Win Modelling Tool - ARCHITECTURE 2.....	43
9.2.1 $S = 0.255$, $x=1:0.1:4$	43
9.2.2 $S = 0.255/2$, $x=1:0.1:9$	43
9.2.3 $s3=0.255$, $x=1:0.1:7.5$	44

1. Introduction:

In principle, two basic technologies are used for building high-capacity networks: circuit switching and packet switching.

In circuit-switched networks, network resources are reserved all the way from sender to receiver before the start of the transfer, thereby creating a circuit. The resources are dedicated to the circuit during the whole transfer. Control signaling and payload data transfers are separated in circuit-switched networks. Processing of control information and control signaling such as routing is performed mainly at circuit setup and termination. Consequently, the transfer of payload data within the circuit does not contain any overhead in the form of headers or the like. Traditional voice telephone service is an example of circuit switching. An advantage of circuit-switched networks is that they allow for large amounts of data to be transferred with guaranteed transmission capacity, thus providing support for real-time traffic. A disadvantage of circuit switching, however, is that if connections are short-lived—when transferring short messages.

On the other hand, Packet switching was developed to cope more effectively with the data-transmission limitations of the circuit-switched networks during bursts of random traffic. In packet switching, a data stream is divided into standardized packets. Each contains address, size, sequence, and error-checking information, in addition to the payload data. The packets are then sent through the network, where specific packet switches or routers sort and direct each single packet.

In Packet switching, the IP, packets are treated independently of each other inside the network, because complete information concerning the packet destination is contained in each packet. This means that packet order is not always preserved, because packets destined for the same receiver may take different paths through the network.

In many aspects, a packet-switched network is a network of queues. Each network node contains queues where incoming packets are queued before they are sent out on an outgoing link. If the rate at which packets arrive at a switch point exceeds the rate at which packets can be transmitted, the queues grow. The queuing causes delay, and if the queues overflow, packets are lost, which is called congestion. Loss of data generally causes retransmissions that may either add to the congestion or result in less-effective utilization of the network. The ability to support real-time traffic in packet-switched networks thus calls for advanced control mechanisms for buffer handling and direction. As a result, the complexity and necessary ability to process information, and therefore the need for computer power, increases sharply when striving for high transmission capacity.

Voice-Over-IP, (VoIP) is the latest buzz word these days. It involves digitizing the voice samples and sending over IP medium. This technology threatens the very existence of a century old circuit-switched traditional telephony system. It involves several steps to convert Analog voice to IP Packets and also to convert IP Packets to Analog voice.

1.1 Analog-Digital:

The Analog voice coming from various sources are digitized using CODEC (Coder/Decoder) and the resultant digital samples are sampled using Pulse Code Modulation (PCM). The Sampling is followed by Quantization. This process allows us to determine the amplitude range and within this range each value is assigned a distinct value.

The bottom line is that the process of sampling gives us amplitudes and the process of Quantization allows us to depict the amplitude values in bits.

1.2 Compression:

The Analog voice is digitized and is represented in the forms of bits. This data is further compressed to decrease the length of payload. To compress the digitized data, there are different compression algorithms like G.723, G.729, G.728, G.711 etc. Added to these compression techniques, different noise reduction, echo cancellation algorithms could be inserted to have better voice quality

1.3 IP Conversion:

The voice data ie. compressed digitized data, is converted into RTP (Real Time Protocol) & RTCP (Real Time Control Protocol) packets and uses TCP/IP protocol, to send the data into IP cloud.

To convert an IP Packet into Analog samples, the following steps are employed.

1.4 Extraction of Digitized Voice from IP:

The IP packets are received over IP cloud and the voice information encapsulated in IP packet is defragmented to extract the RTP Packet. Depending on source and destination address, the voice packets are extracted to send for de-compression.

1.5 De-compression:

The Digitized sample which is compressed using any of the compression techniques are decompressed accordingly and the extract voice information is made free from noise and echo before sending it for Digital-Analog conversion.

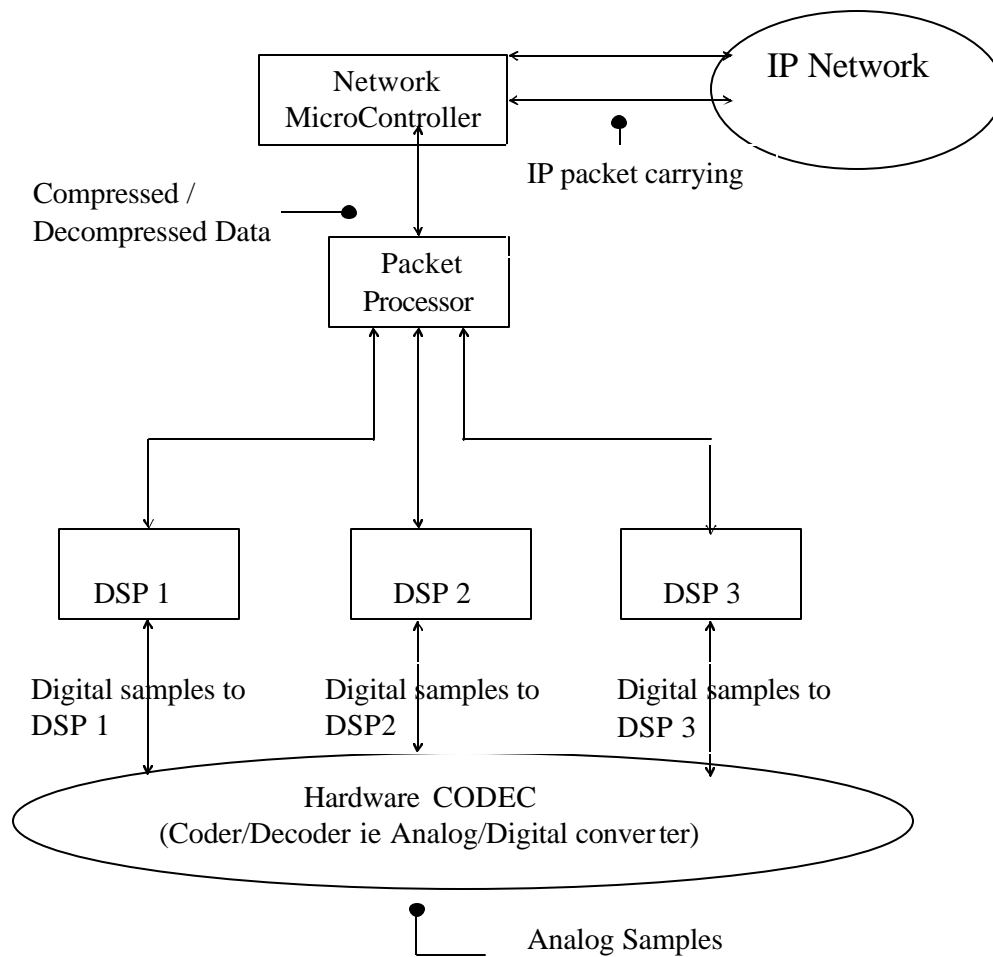
1.6 Digital – Analog:

The Digital information of voice encoded in zeros and ones is converted back to analog signals of sine waves using CODEC(Coder/Decoder) and is passed to different Analog listeners.

The above techniques are the traditional entities of VoIP Gateway which employs the steps 1.1, 1.2 & 1.3 are used in converting Analog signal – Digital packet – IP packet and the steps 1.4, 1.5 & 1.6 are used in converting IP packet – Digital packet – Analog signal. The VoIP Gateway will be designed to do the above conversion for numerous telephonic users. The number of subscribers, it can support depends on various factors and parameters. Given the above steps a closer look, it involves many queues, packets, service time, arrival time, resident time at each node of conversion.

In this report, we are evaluating the VoIP Gateway, which is available in two flavours are discussed. Also different performance evaluating variables, constraints and other influencing factors are dealt. Hence this report is an attempt to introduce the VoIP Gateway as a testing entity and to evaluate the performance of it.

2. ARCHITECTURE 1 :



The above picture shows our first flavour of VoIP Gateway. It consists of pool of DSPs (DSP1,DSP2,DSP3), Packet Processor(DSP), Network Micro-Controller(Low End Processor) and Hardware components as CODECs.

2.1 CODEC :

This section of Hardware contains CODECS to convert Analog signals to Digital samples and vice-versa. It receives Analog signals from different sources(Telephones) and converts into Digital samples and gives it as input to DSPs

On the other way, it receives Digital signals as input and converts into Analog signals and then sends it to listeners

Hence this section of hardware takes care of sections 1.1 & 1.6 as explained above.

2.2 DSPx(DSP1,DSP2,DSP3) :

The three DSPs shown above receives digital samples, checks for error correction/detection and performs echo cancellation on the received signal and later it compresses the signal depending on the compression algorithm. After the completion of compression, it submits a packet output which is fed as input to the packet processor. During the execution of this path, it executes the functionalities of section 1.2 as explained above.

On the other hand, it receives voice samples in digitized format from packet processor and it decompresses the signal(depending on the compression used) and later it checks for error correction/detection and also echo cancellation, which inturn is fed as input to Hardware CODEC. During the execution of this path, it executes the functionalities of section 1.5 as explained above.

2.3 Packet Processor :

This Processor acts as Multiplexer / Demultiplexer between DSPs and Network Microcontroller. Also, the main objective of using this processor is to lessen the burden on Network Microcontroller in segregating and desegregating the voice samples.

On one end, it receives voice samples from 3 DSPs i.e information content of X channels $\times 3 = 3X$ channel information and sends it to Network MicroController. During this operation,it acts as traffic integrator (De-Multiplexer) and it feeds input to Network Micro-Controller.

On the other end, it receives voice samples from Network MicroController(information corresponding to Y channels for 3 DSPs). During this operation, it acts as traffic disintegrator (Multiplexer) and it sends the samples to the corresponding DSP by employing channel checking mechanisms.

2.4 Network MicroController:

This MicroController takes voice samples from packet processor and employs RTP & RTCP packetization and in turn frames IP packet and outputs into IP network. Hence it executes the section 1.3 as explained above.

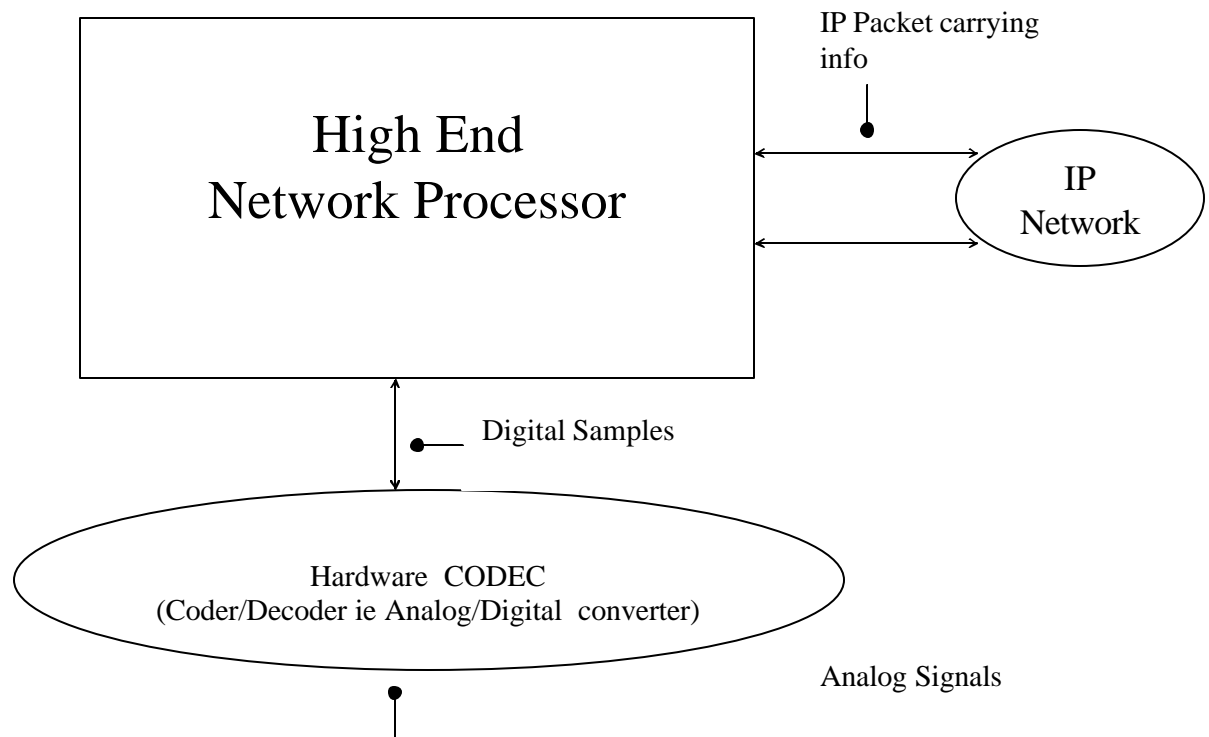
On the other hand, it receives IP packets from the network, extracts RTP & RTCP packets and in turn extracts the voice sample information in RTP packet and it feeds to packet processor as input. Hence it executes the section 1.4 as explained above.

NOTES :

It is assumed for evaluation that, Digital samples for X channels is given as input to each DSP and the Network MicroController outputs IP packets for X channels

Also, the other consideration is that, the Network MicroController receives IP information of X channels and each DSP outputs digital data corresponding to X channels to CODEC.

3. ARCHITECTURE 2 :



The above picture shows our second flavour of VoIP Gateway. It consists of High End Network processor and Hardware components as CODECs.

3.1 CODEC :

This section of CODEC is similar to the model described in Architecture 1. The only difference is the CODEC ip/op is connected to Network Processor and it sends/receives digital samples corresponding to Y channels.

Also, it is implied that this section of hardware takes care of sections 1.1 & 1.6 as explained above.

3.2 HIGH END NETWORK PROCESSOR :

This Processor acts as both DSP and Network Processor. Hence the name High End Network Processor. It contains the modules like Echo Cancellation, Compression/De-Compression, RTP – RTCP Handling, IP Handling. Basically this processor could be defined as a system which replaces

Network MicroController, Packet Processor and DSPs, because of its high end capabilities that includes the speed of execution and better architecture of the processor.

On one end, it receives digital samples related to Y channels, does error correction/error detection, Echo Cancellation and also runs compression algorithm. The compressed voice packet is packetized into RTP , RTCP packets and subsequently into IP packet. Hence it performs the functions listed in the sections 1.2 & 1.3.

On other end, it receives IP packet for Y channels, defragments the IP packet and extracts the RTP & RTCP information and then takes the voice sample. The voice sample is sent for decompression and later it is checked for errors and echo cancellation. After the refinement of packet, it employs channel identification mechanism and finally writes into codec.Hence it performs the functions listed in the sections 1.4 & 1.5.

NOTES :

It is assumed for evaluation that, Digital samples for Y channels s is given as input to High End Network Processor and it outputs IP packets for Y channels

Also, the other consideration is that, the High End Network Micro Processor receives IP packet information of Y channels and digital data corresponding to Y channels to CODEC.

4. PERFORMANCE EVALUATION PARAMETERS :

4.1 Description of Architecture 1 :

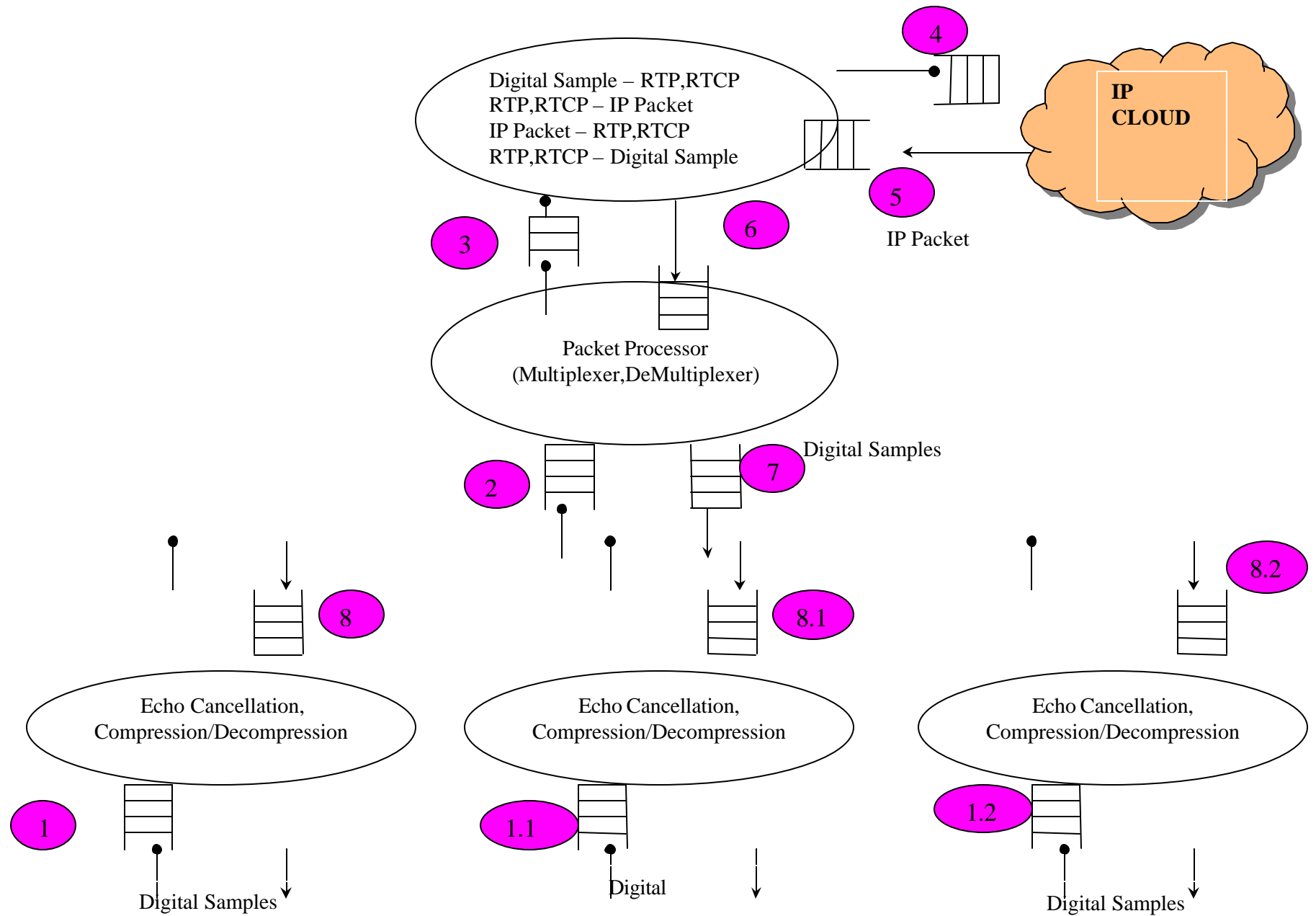
As shown in the below diagram, it represents the practical representation of Voice Gateway of Architecture 1. This architecture contains 12 queues in which 6 queues(No:1,1.1,1.2,2,3,4,) are used for Digital sample to IP conversion and other 6 queues(No: 5,6,7,8,8.1,8.2) are used to convert IP to Digital sample. The description of each queue is described below:

4.1.1 Queue 1 , 1.1 , 1.2 :

These queues are identical to each other because of the data it carries. But it operates on different DSPs and gets data from different CODECs. Each queue contains Digital samples for X channels. This queue is populated after getting the Digital data from CODEC. This data is given as input to DSP which does Echo cancellation, Compression of the signal.

4.1.2 Queue 2:

This queue takes input from the queues numbered 2 and is fed to Packet Processor where it does error checking . In this way of operation, it acts as multiplexer by taking Digital samples from



each DSP, and routing to the queue 4. This contains samples from $3 * X = 3X$ (i.e Y) channels information.

4.1.3 Queue 3:

This queue is the output from Packet Processor which is echo free, compressed and error free packet. This queue is given as input to Network MicroController for RTP, RTCP and subsequently IP processing.

4.1.5 Queue 4:

This queue is the output from Network MicroController and represents the IP packet which contains information about RTP, RTCP, Voice(Digital) samples to be routed to IP cloud.

4.1.6 Queue 5:

This queue is populated from the IP input received from IP cloud which contains information as stated in Queue 5 and is fed as input to Network MicroController for further processing.

4.1.7 Queue 6:

This queue is populated by Network MicroController after extracting RTP, RTCP information from the IP packet(Queue 5) and subsequently Digital Samples which are compressed, error free. This is fed as input to Packet Processor.

4.1.8 Queue 7:

This queue contains information for $3 * X = 3X$ (i.e Y) channels contains voice packet after undergoing error checking by Packet Processor and this information is passed to the queues numbered 8, 8.1, 8.2 depending on the channel numbers. In this scenario Packet Processor acts as De-Multiplexer taking inputs from queue 7 and passing it on to queues 8, 8.1, 8.2.

4.1.9 Queue 8, 8.1, 8.2:

This queue contains the data received from packet processor and contains data for X channels each to give input to DSPs for De-Compression & Echo Cancellation.

4.2 Description of Architecture 2 :

As shown in the below diagram, it represents the practical representation of Voice Gateway of Architecture 2. This architecture contains 3 queues in which 2 queues (No:1 & 2) are used for Digital sample to IP conversion and the other queue (No: 3) are used to convert IP to Digital sample. The description of each queue is described below :

4.2.1 Queue 1 :

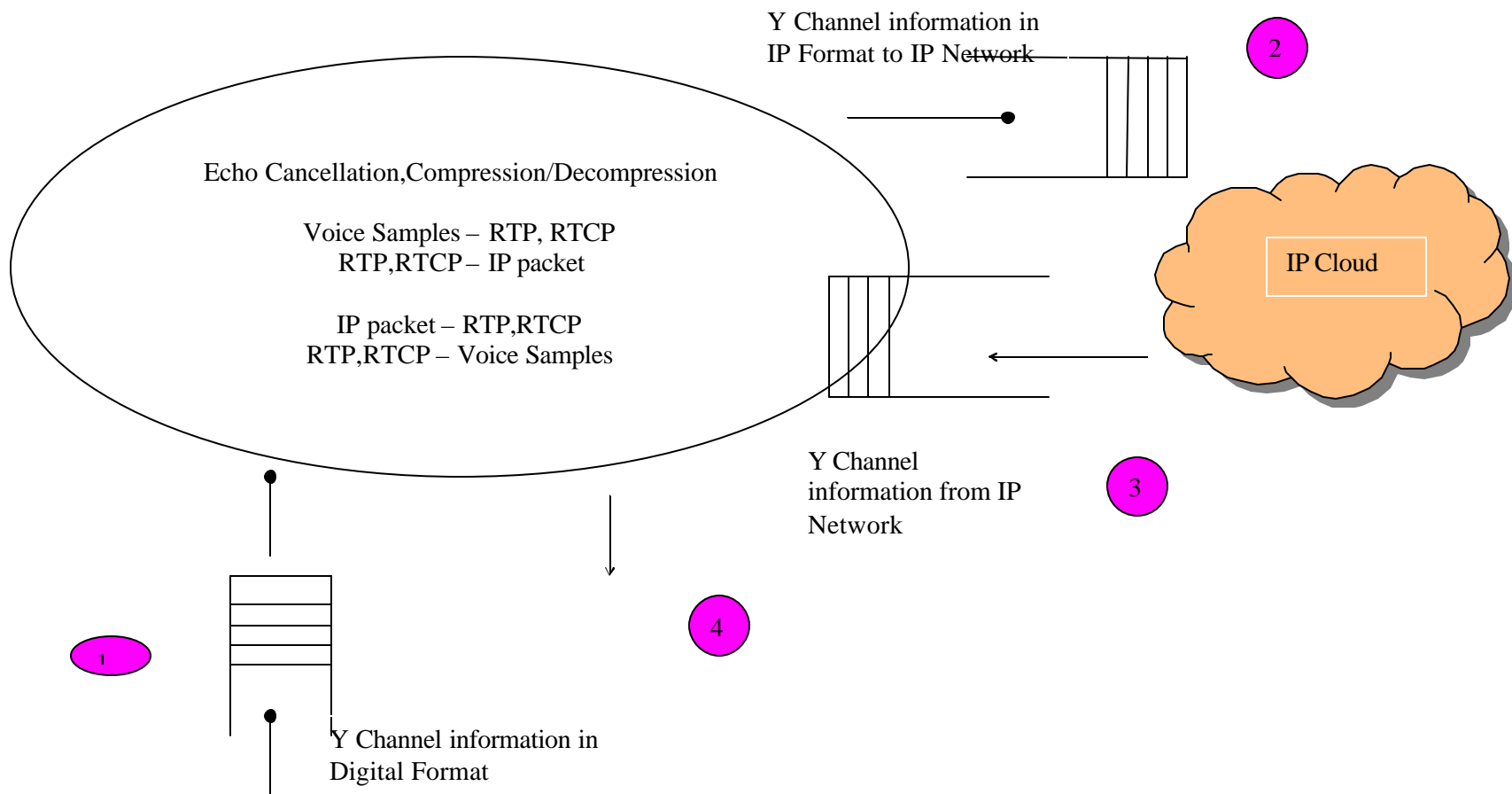
This queue contains Digital Sample received from different CODECs and contains samples for Y channels. This is fed as input to High End Network Processor for subsequent processing.

4.2.2 Queue 2:

The High End Network Processor received Digital samples , it checks for echo cancellation and compresses it using different compression algorithms and subsequently frames RTP,RTCP packets before encapsulating in IP format. This queue contains IP Packet from High End Network Processor and it pumps the packet to IP Cloud.

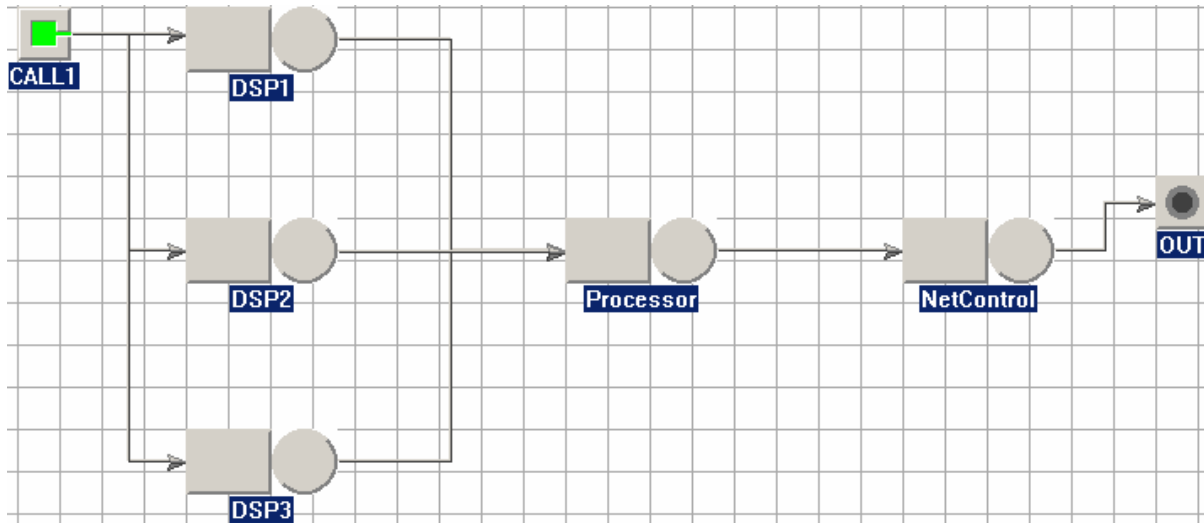
4.2.3 Queue 3:

This queue does exactly the opposite of Queue 2. It receives IP packet from IP cloud. It collects the information, executes RTP & RTCP , extracts the voice (Digital) sample and runs echo cancellation and De-Compression algorithm before feeding it to CODECs



4.3 Parameters – Architecture 1:

The Architecture – 1 could be represented in the format as shown below using Win Modelling Tool. Each entity is described below:



4.3.1 CALL 1:

This is the input to the system ie the output of CODEC which carries Digital Samples. The Arrival Rate of CALL 1 describes total customers in the system which conveys the total number of channels ($Y = X1 + X2 + X3$), where $X1, X2, X3$ represents the arrival rate for DSP1, DSP2 & DSP3 respectively. The load could be distributed indifferent ways depending upon routing used. The routing probability used will decide the input to each DSP ie X as described above.

1) If arrival rate for DSP1, DSP2, DSP3 is equal, ie $DSP1 = DSP2 = DSP3 = 0.333333$ then $X1, X2, X3$ are equal (hence $Y = 3 * X$)

2) If Routing follows $DSP1 > DSP2 > DSP3$ then $X1 > X2 > X3$

3) If Routing follows $DSP1 < DSP2 < DSP3$ then $X1 < X2 < X3$

so for a given Arrival Rate Y , whatever may be the routing formula, always it follows $Y = X1 + X2 + X3$

4.3.2 DSP 1, DSP 2, DSP 3:

The Basic idea of using these DSPs is for Compression and error detection. The DSPs could be selected in different ways wrt performance, frequency of operation which is directly correlated with service time of the Win Modelling Tool.

Eg:- It could be assumed for a given execution of application that for a DSP whose frequency of operation = 75 MHZ corresponds to a service time of 0.25 Sec, it is always implied for other DSP whose frequency of operation < 75 MHZ then service time is always > 0.25 Sec. Hence frequency of operation is indirectly proportional to service time. More the frequency lesser the service time.

Hence the DSPs could be analyzed as FCFS queues with variant service times

4.3.3 Processor :

This processor receives the samples from 3 DSPs, hence the queue is designed as PS (processor sharing) and the frequency of execution of this processor could be correlated with service time.

4.3.4 Netcontrol:

This processor is a Network Controller which accepts samples from Packet Processor and outputs to IP Domain. This system could be analyzed as FCFS queue with variable service time which decides frequency of operation of the Controller.

4.3.5 MEASUREMENT CRITERIA :

As described above the system is studied considering the following parameters:-

1. Variable Arrival Rate.
2. Routing probability (among DSP1, DSP2, DSP3) (i.e Equal probability, lesser & greater probability when comparing DSP1, DSP2, DSP3)
3. Variable Service times (DSP1, DSP2, DSP3, Processor, Netcontroller)

Also, assuming s_{11}, s_{12}, s_{13} to be service time for DSP1, DSP2, DSP3; x_1, x_2, x_3 being arrival rates for DSP1, DSP2, DSP3. The Response time at the input of processor is given by

$R(i) =$

$$(((s_{11}/(1-x_1*s_{11})) * x_1/(x_1+x_2+x_3)) + ((s_{12}/(1-x_2*s_{12})) * x_2/(x_1+x_2+x_3)) + ((s_{13}/(1-x_3*s_{13})) * x_3/(x_1+x_2+x_3)))$$

Hence we used matlab tool to get the Response Time which employs above equation and also for different Routing probabilities. The same variables are considered to get the output parameters like Queue Length, Throughput, Response Time, utilization, Waiting time etc using Win Modelling Tool and the results are tabulated.

Also the results of the graphs, tabulated values gives ideal values for the system parameters like X, X_1, X_2, X_3, Y . for given constraints of the system.

4.4 Parameters – Architecture 2:



4.4.1 CALL :

This is the input to the system ie the output of CODEC which carries Digital Samples. The Arrival Rate of CALL 1 describes total customers in the systems which conveys the total number of channels (Y).

4.4.2 CPU:

The service time of CPU decides the frequency of operation of the Processor. The frequency of operation of the CPU could be studied properly to understand whether the CPU can sustain to the Arrival Rate so that there will be no packet loss.

4.4.3 MEASUREMENT CRITERIA:

1.Variable Arrival Rate.

2.Variable Service times(CPU)

Also the Response time

$$R(i) = S/(1-x*S)$$

Where S – Service Time, X – Arrival Rate,

we used matlab tool to get the Response Time to plot the graph between arrival rate and service time which employs above equation and also for different Routing probabilities. The same variables are considered to get the output parameters like Queue Length, Throughput, Response Time, utilization, Waiting time etc using Win Modelling Tool and the results are tabulated.

5.Conclusion

This project makes a niche to all the designers of VoIP Gateway. The VoIP Gateway, available in two flavours is analyzed & studied for different system parameters like Arrival Rate, Response Time, Queue Lengths, Throughput, Utilization, Waiting Time etc. The values and graphs related to the above parameters are tabulated.

The results mentioned in Appendix A & B gives an input framework for any person who would like to select the design of VoIP Gateway. The results tabulated gives an indication about the performance of the system taking different real time design components like frequency of operation of Processors(Service Time) , Performance(Throughput) , Input (Arrival Rate), Output (Utilization). These values could be taken as base reference by any customer who wish to evaluate the better architecture that suits the demands from the Client(Cost of the project, Design time, Man Power etc.,)

To conclude VoIP Gateway offers lucrative advantages to customers and service providers alike. However, as with any new technology, it brings its own sets of network design and optimization issues. By understanding the important parameters, and acquiring the knowledge over performance evaluation parameters, customers & clients can reap the benefits of voice over packet services.

6. Limitations & Assumptions

1. It is assumed that sections described in 1.1 & 1.6 will be carried out by hardware and the DSPx(DSP1,DSP2,DSP3) always gets Digital Samples as input & output.

2. The VoIP Gateway is analyzed only for Digital to IP conversion. It doesn't consider any parameters involving IP – Digital conversion. The parameters & other system parameters considering IP – Digital conversion (section 1.4 to 1.6) in coherence with Digital – IP conversion(section 1.1 to 1.3) is beyond the scope of this current project.

7. References

[1] William Stallings ,*“Queuing Analysis”*

[2] <http://www.protocols.com/papers/voip2.htm> “Fine - Tuning Voice Over Packe t Services”

[3] http://www.cisco.com/en/US/tech/tk652/tk701/tech_white_papers_list.html “IP Telephony / VoIP White Papers ”

APPENDIX A:

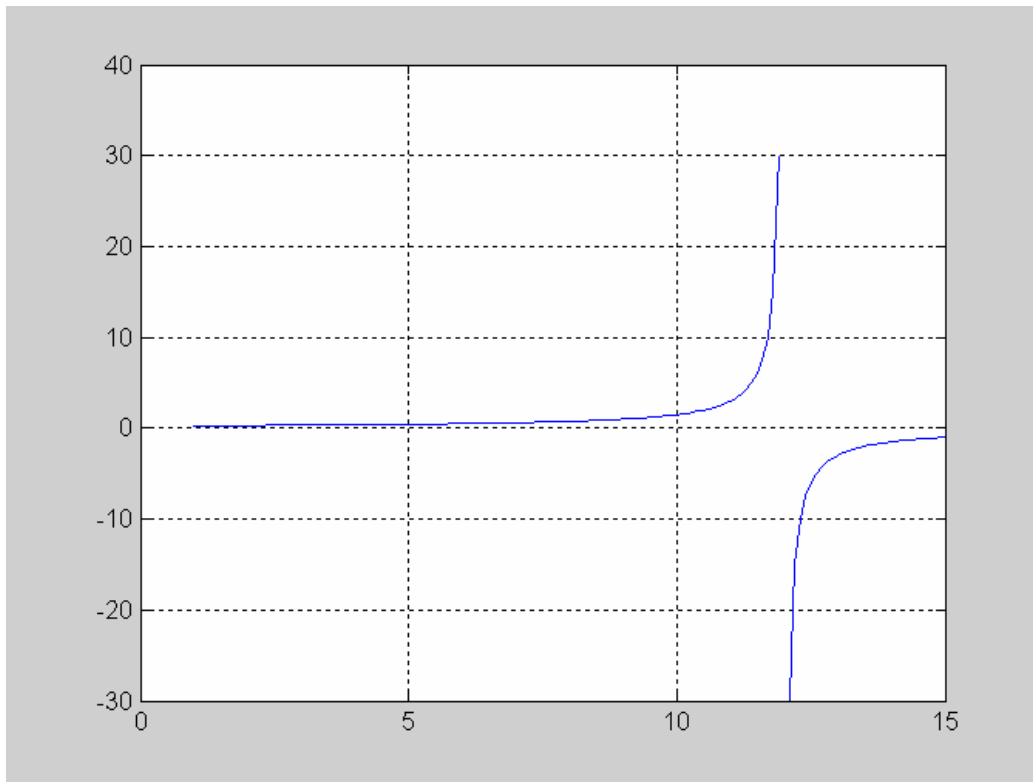
MATLAB

8.1 MATLAB CODE – Architecture 1

In this section, we would like to present the Matlab code and the graph we obtained for different conditions and parameters.

8.1.1 Equal Routing Probability , $s11=s12=s13>s2>s3$

```
x=1:0.1:15;
s11=0.25;
s12=0.25;
s13=0.25
s2=0.003
s3=0.002
r1=x./3
r2=x./3
r3=x./3
y=(1/3)*(s11./(1-r1.*s11))+(1/3)*(s12./(1-r1.*s12))+(1/3)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```

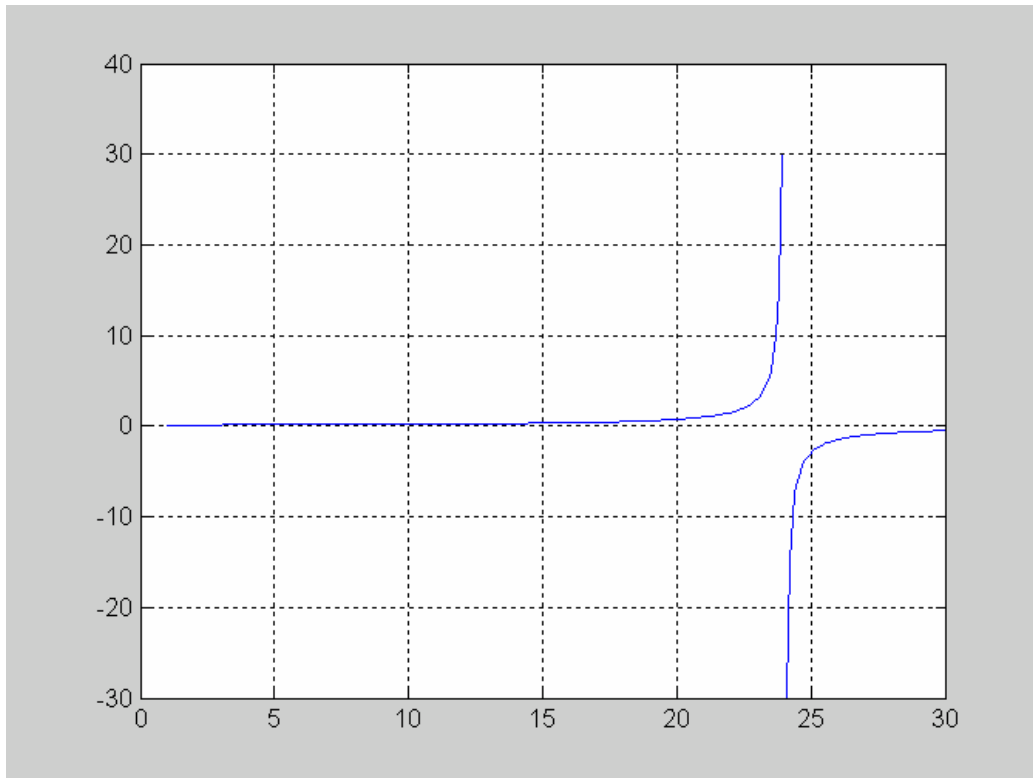


8.1.2 Equal Routing Probability , $s_{11}=s_{12}=s_{13}>s_2>s_3$ (Service Times are halved)

```

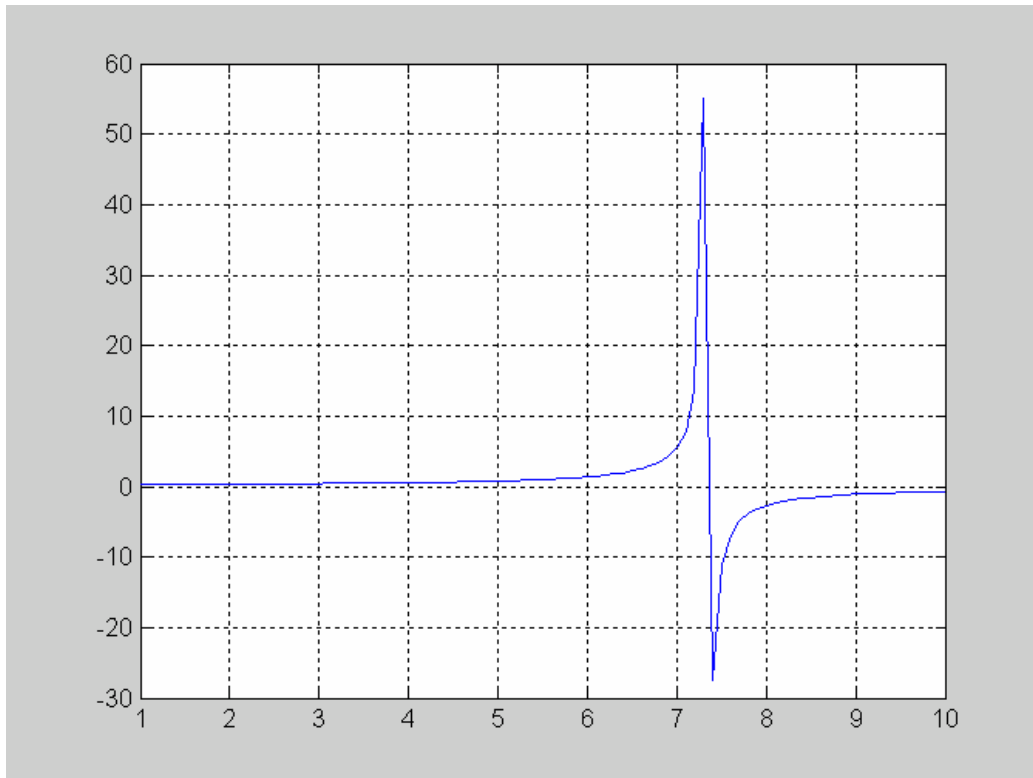
x=1:0.1:30;
s11=0.25/2;
s12=0.25/2;
s13=0.25/2
s2=0.003/2
s3=0.002/2
r1=x./3
r2=x./3
r3=x./3
y=(1/3)*(s11./(1-r1.*s11))+(1/3)*(s12./(1-r1.*s12))+(1/3)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;

```



8.1.3 $s_{11} = s_{12} = s_{13} > s_2 > s_3$, Routing : $r_1=2* r_2=3* r_3$

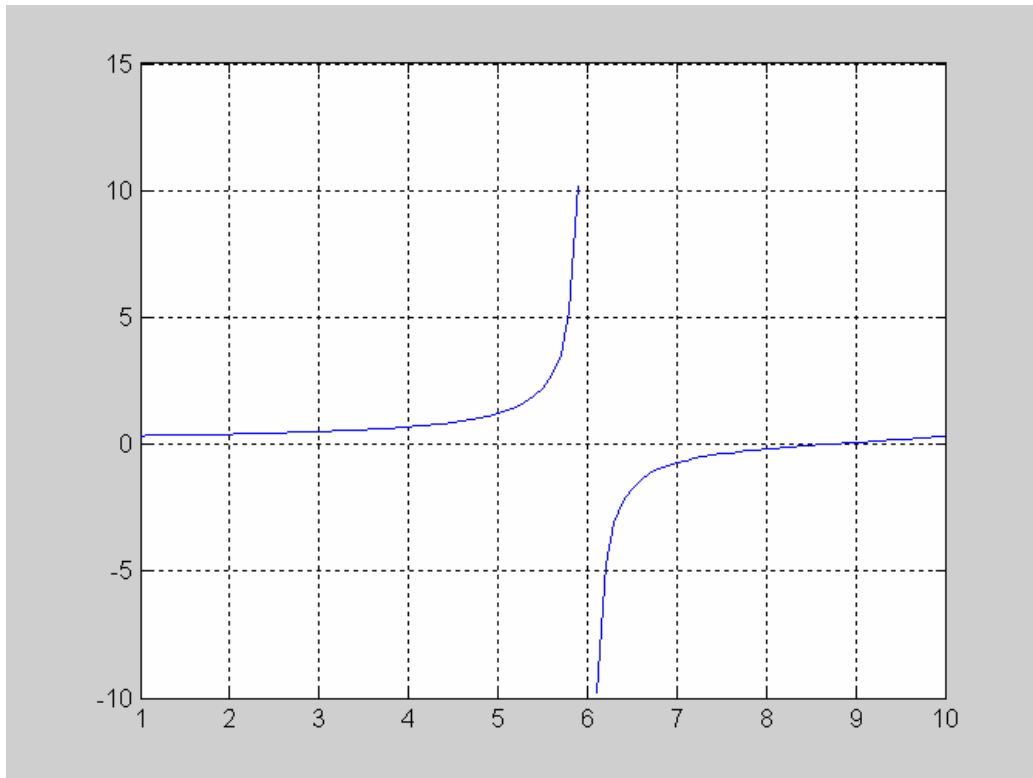
```
x=1:0.1:10;
s11=0.25;
s12=0.25;
s13=0.25
s2=0.003
s3=0.002
r1=x.*6/11
r2=x.*3/11
r3=x.*2/11
y=(1/3)*(s11./(1-r1.*s11))+(1/3)*(s12./(1-r1.*s12))+(1/3)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```



8.1.4 Equal Routing Probability, Routing : $s_{11} > s_{12} > s_{13}$

```
% r1= r2= r3
% s11>s12>s13

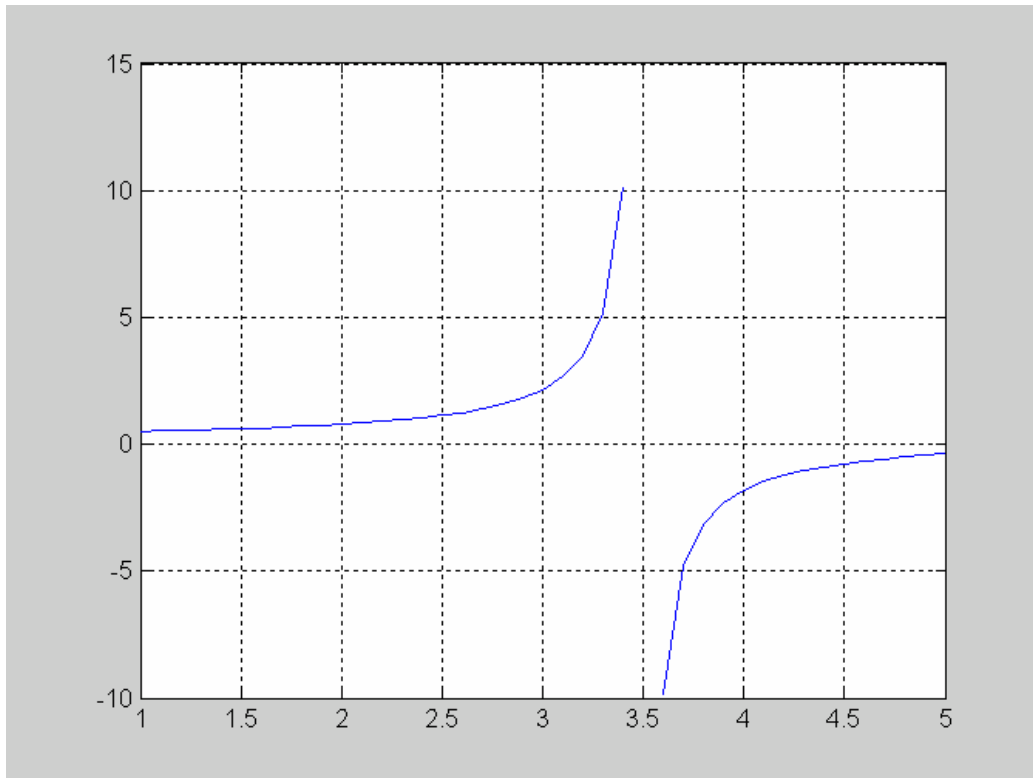
x=1:0.1:10;
s11=0.5;
s12=0.25;
s13=0.25/2
s2=0.003
s3=0.002
r1=x./3
r2=x./3
r3=x./3
y=(1/3)*(s11./(1-r1.*s11))+(1/3)*(s12./(1-r1.*s12))+(1/3)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```

8.1.5 $r_1=2$ $r_2=4r_3$, $s_{11}>s_{12}>s_{13}$

```
% r1=2 r2= 4r3
% s11>s12>s13
```

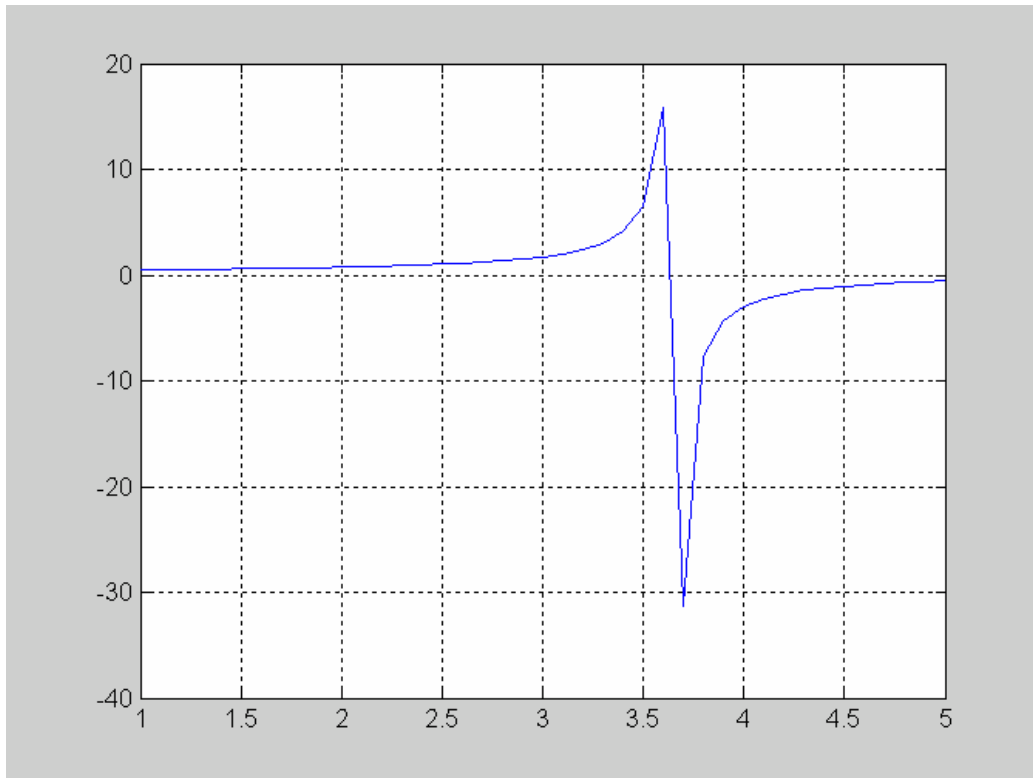
```
x=1:0.1:5
s11=0.5;
s12=0.25;
s13=0.25/2
s2=0.003
s3=0.002
r1=x.*4/7
r2=x.*2/7
r3=x.*1/7
y=(4/7)*(s11./(1-r1.*s11))+(2/7)*(s12./(1-r1.*s12))+(1/7)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```



8.1.6 $r_1=2$ $r_2=3r_3$, $s_{11}>s_{12}>s_{13}$

```
% r1=2 r2=3r3
% s11>s12>s13
```

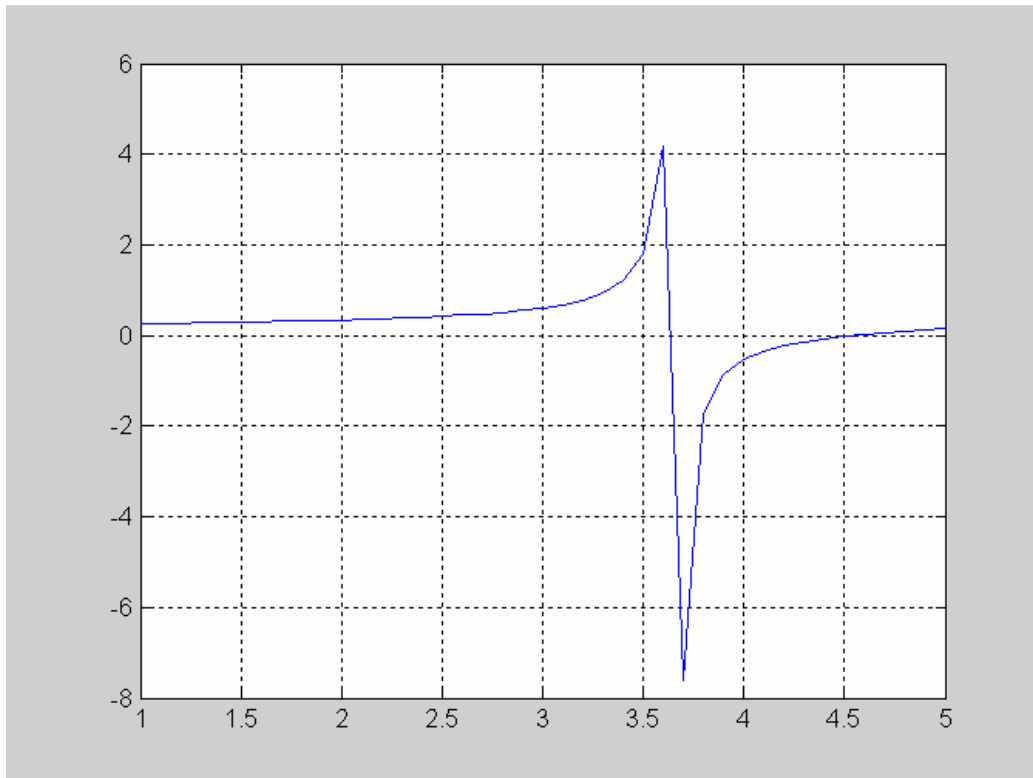
```
x=1:0.1:5
s11=0.5;
s12=0.25;
s13=0.25/2
s2=0.003
s3=0.002
r1=x.*6/11
r2=x.*3/11
r3=x.*2/11
y=(4/7)*(s11./(1-r1.*s11))+(2/7)*(s12./(1-r1.*s12))+(1/7)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```



8.1.7 $r_1=2$ $r_2=3r_3$, $s_{11}<s_{12}<s_{13}$

```
% r1=2 r2=3r3
% s11<s12<s13
```

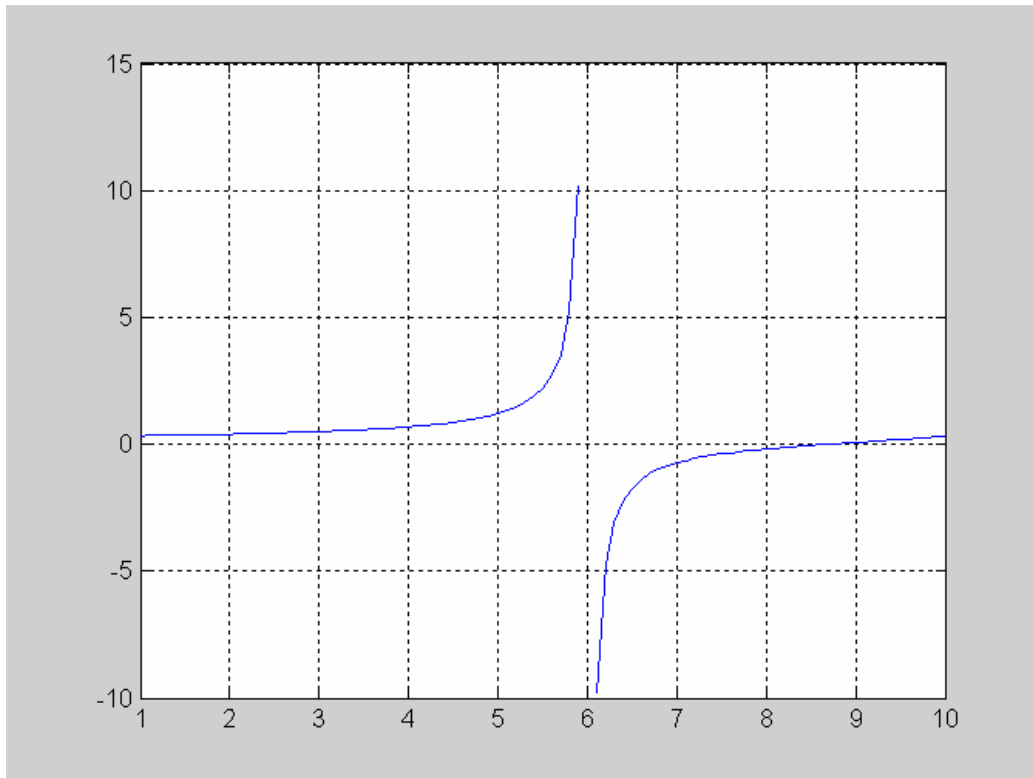
```
x=1:0.1:5
s11=0.25/2;
s12=0.25;
s13=0.25*2
s2=0.003
s3=0.002
r1=x.*6/11
r2=x.*3/11
r3=x.*2/11
y=(4/7)*(s11./(1-r1.*s11))+(2/7)*(s12./(1-r1.*s12))+(1/7)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```



8.1.8 $r1 = r2 = r3$, $s13 > s12 > s11$

```
% r1= r2= r3
% s13>s12>s11

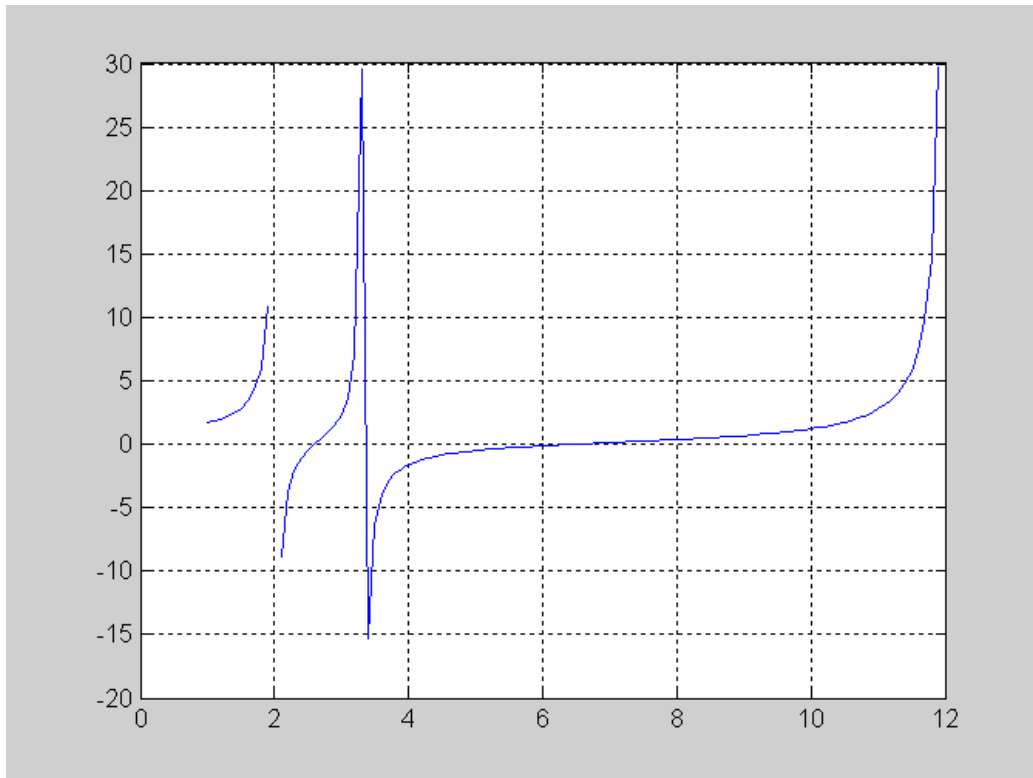
x=1:0.1:10;
s11=0.25/2;
s12=0.25;
s13=0.5
s2=0.003
s3=0.002
r1=x./3
r2=x./3
r3=x./3
y=(1/3)*(s11./(1-r1.*s11))+(1/3)*(s12./(1-r1.*s12))+(1/3)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```



8.1.9 $s_{11}=s_{12}=s_{13}, s_{11} > s_2 > s_3, r_1=r_2=r_3$

```
% s11=s12=s13,s11 > s2>s3
%r1=r2=r3
```

```
x=1:0.1:12;
s11=0.25;
s12=0.25;
s13=0.25
s2=0.3
s3=0.5
r1=x./3
r2=x./3
r3=x./3
y=(1/3)*(s11./(1-r1.*s11))+(1/3)*(s12./(1-r1.*s12))+(1/3)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```



8.1.10 $s_{11}=s_{12}=s_{13}, s_{11}<s_2<s_3$ $r_1=r_2=r_3$

```
% s11<s2<s3, s11 = s12=s13
```

```
%r1=r2=r3
```

```
x=1:0.01:5;
```

```
s11=0.25;
```

```
s12=0.25;
```

```
s13=0.25
```

```
s2=0.3
```

```
s3=0.27
```

```
r1=x./3
```

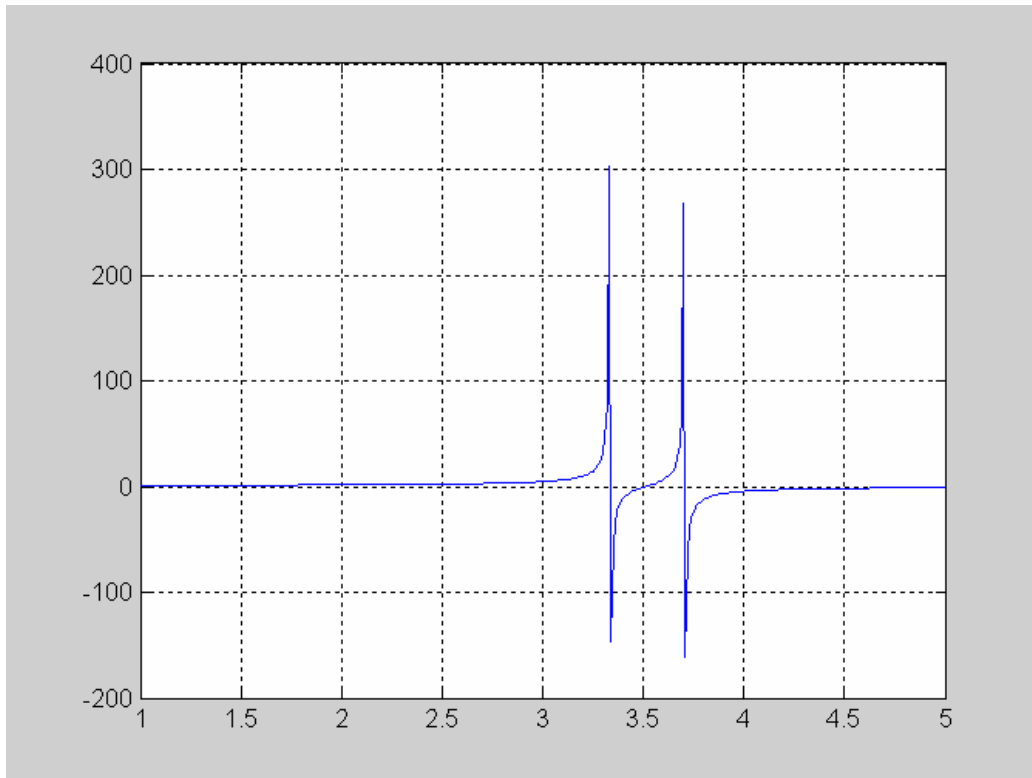
```
r2=x./3
```

```
r3=x./3
```

```
y=(1/3)*(s11./(1-r1.*s11))+(1/3)*(s12./(1-r1.*s12))+(1/3)*(s13./(1-r1.*s13))+(s2./(1-x.*s2))+(s3./(1-x.*s3));
```

```
title('multiple server');
```

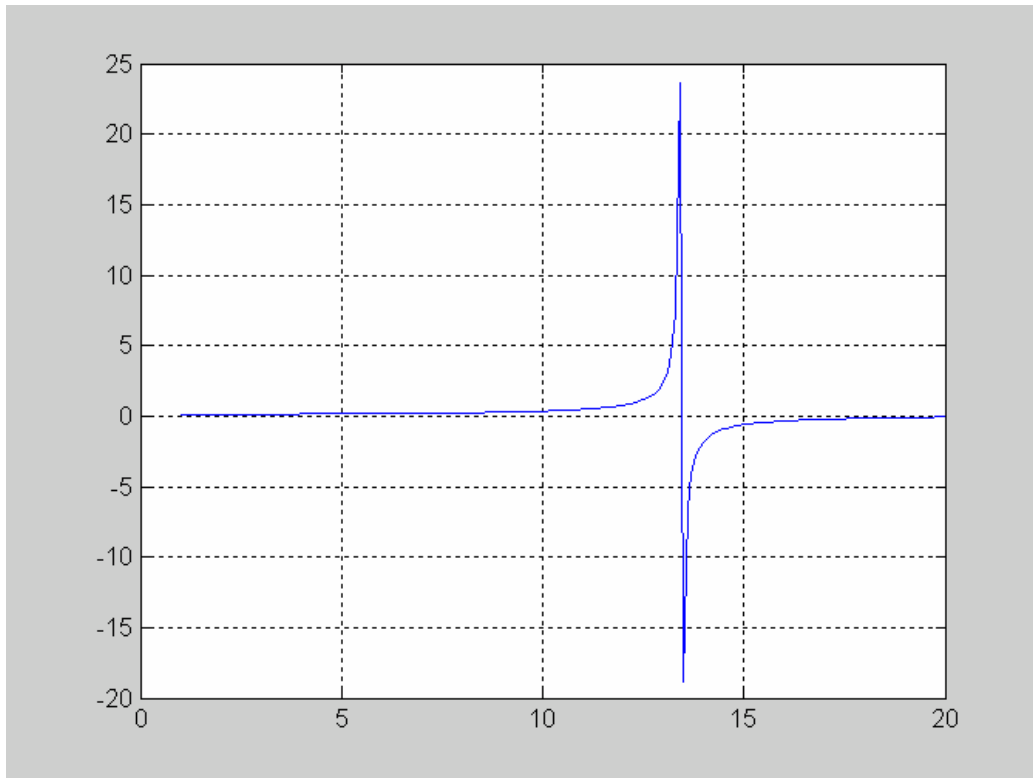
```
plot(x,y),grid;
```



8.1.11 $r1=2$ $r2=3r3$, $s11=2s12=3s13$

```
% r1=2 r2=3r3
% s11=2s12=3s13

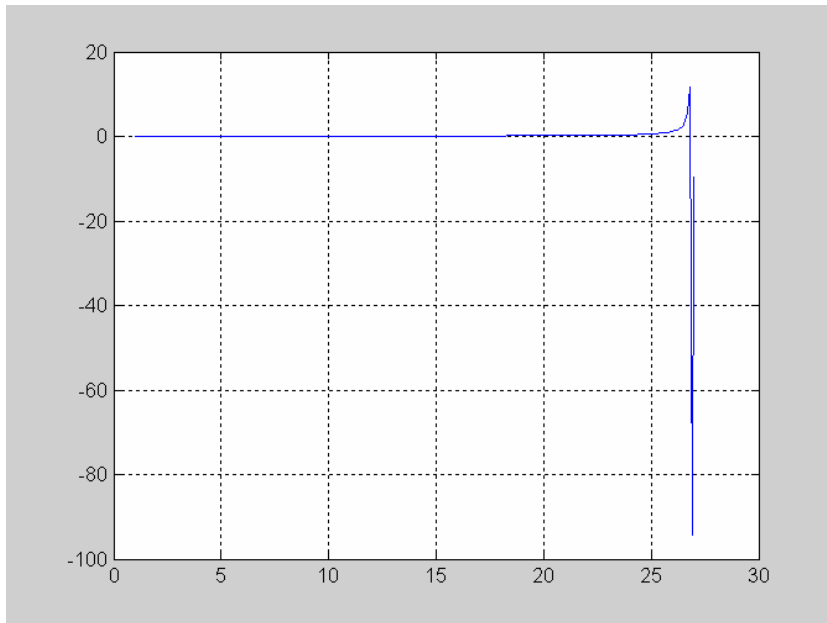
x=1:0.1:20
s11=0.25*6/11;
s12=0.25*3/11;
s13=0.25*2/11
s2=0.003
s3=0.002
r1=x.*6/11
r2=x.*3/11
r3=x.*2/11
y=(4/7)*(s11./(1-r1.*s11))+(2/7)*(s12./(1-r1.*s12))+(1/7)*(s13./(1-r1.*s13))+(s2./(1-
x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```



8.1.12 $r1=2$ $r2=3r3$, $s11=2s12=3s13$

```
% r1=2 r2=3r3
% s11=2s12=3s13

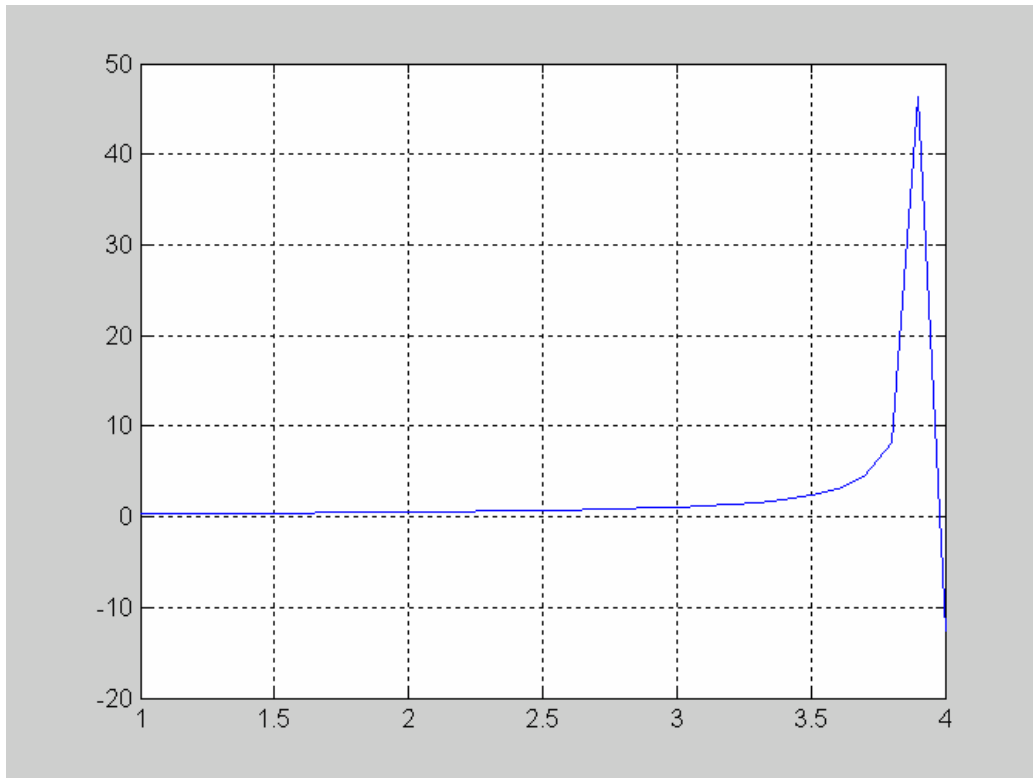
x=1:0.1:27
s11=0.25/2*6/11;
s12=0.25/2*3/11;
s13=0.25/2*2/11
s2=0.003
s3=0.002
r1=x.*6/11
r2=x.*3/11
r3=x.*2/11
y=(4/7)*(s11./(1-r1.*s11))+(2/7)*(s12./(1-r1.*s12))+(1/7)*(s13./(1-r1.*s13))+(s2./(1-x.*s2))+(s3./(1-x.*s3));
title('multiple server');
plot(x,y),grid;
```

8.2 MATLAB CODE – Architecture 2

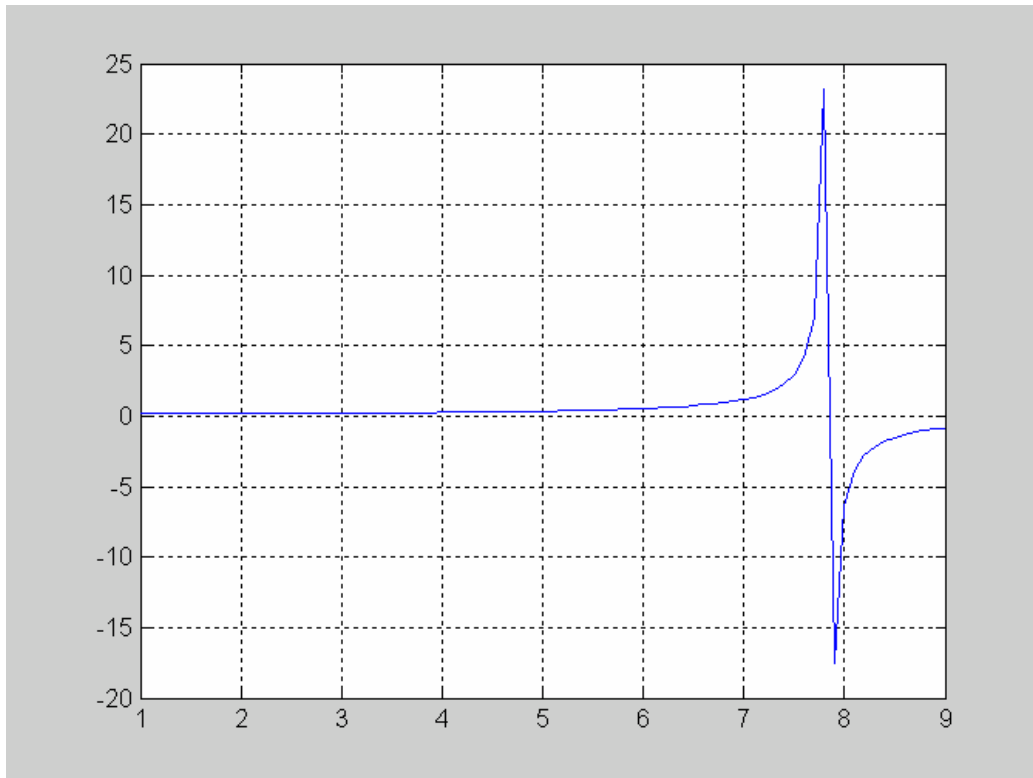
8.2.1 $S = 0.255$, $x=1:0.1:4$

```
x=1:0.1:4;  
s3=0.255;  
y=s3./(1-x.*s3);  
title('single server');  
plot(x,y),grid;
```



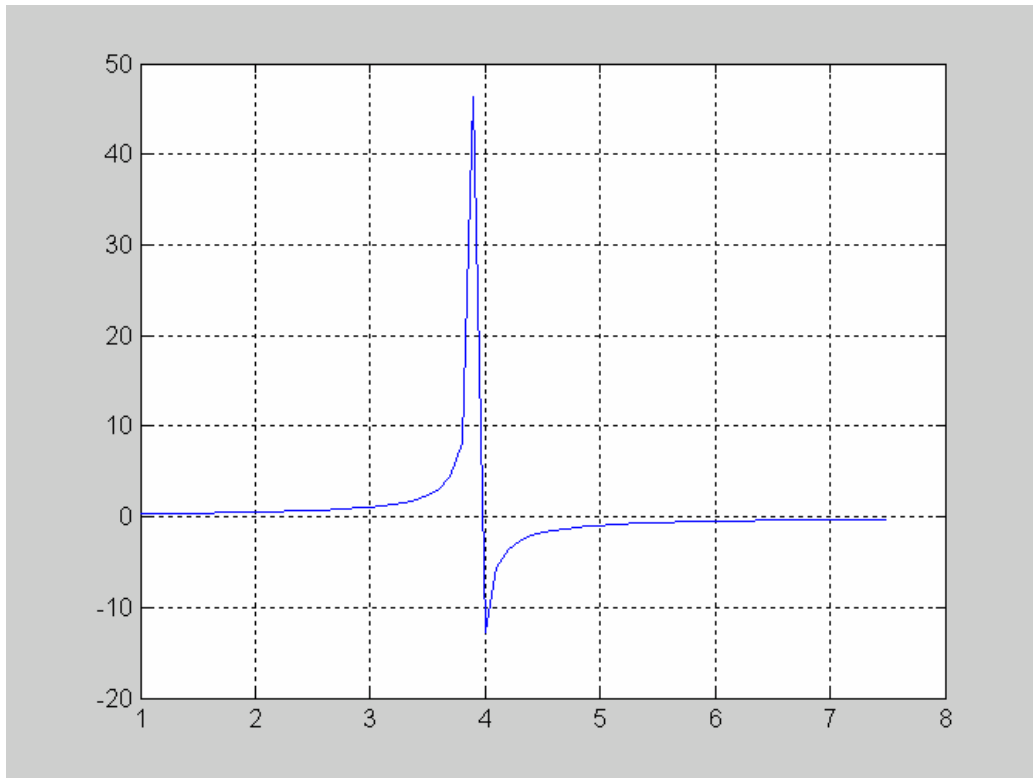
8.2.2 $S = 0.255/2$, $x=1:0.1:9$

```
x=1:0.1:9;  
s3=0.255/2;  
y=s3./(1-x.*s3);  
title('single server');  
plot(x,y),grid;
```



8.2.3 $s3=0.255$, $x=1:0.1:7.5$

```
x=1:0.1:7.5;  
s3=0.255;  
y=s3./(1-x.*s3);  
title('single server');  
plot(x,y),grid;
```



9. TABULAR RESULTS :

9.1 Win Modelling Tool - ARCHITECTURE 1

9.1.1 Equal Routing Probability , $s11=s12=s13>s2>s3$

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	26.03	26.03
DSP2	35.56	35.56
DSP3	43.35	43.35
Processor	0.03631	0.03631
NetControl	0.02411	0.02411
GLOBALS	105	105

Throughput (X)

	VOIP1	GLOBALS
DSP1	3.977	3.977
DSP2	3.985	3.985
DSP3	3.994	3.994
Processor	11.95	11.95
NetControl	11.95	11.95
GLOBALS	12	12

Response Time (R)		
	VOIP1	GLOBALS
DSP1	2.177	2.169
DSP2	2.974	2.963
DSP3	3.626	3.613
Processor	0.003037	0.003026
NetControl	0.002016	0.002009
GLOBALS	8.783	8.749

Utilization (U)		
	VOIP1	GLOBALS
DSP1	0.9942	0.9942
DSP2	0.9963	0.9963
DSP3	0.9984	0.9984
Processor	0.01793	0.01793
NetControl	0.01195	0.01195
GLOBALS	0	0

Waiting Time (W)		
	VOIP1	GLOBALS
DSP1	6.545	6.545
DSP2	8.922	8.922
DSP3	10.86	10.86
Processor	0.003037	0.003037
NetControl	0.002016	0.002016
GLOBALS	0	0

9.1.2 Equal Routing Probability , $s_{11}=s_{12}=s_{13}>s_2>s_3$ (Service Time halved)

Analysis results for Simulation:

Queue Length (n)		
	VOIP1	GLOBALS
DSP1	7.59	7.59
DSP2	7.943	7.943
DSP3	6.846	6.846
Processor	0.03453	0.03453
NetControl	0.02301	0.02301
GLOBALS	22.44	22.44

Throughput (X)		
	VOIP1	GLOBALS
DSP1	7.673	7.673
DSP2	7.681	7.681
DSP3	7.636	7.636
Processor	22.99	22.99
NetControl	22.99	22.99
GLOBALS	23	23

Response Time (R)		
	VOIP1	GLOBALS
DSP1	0.3302	0.33
DSP2	0.3455	0.3453
DSP3	0.2978	0.2976
Processor	0.001502	0.001501
NetControl	0.001001	0.001
GLOBALS	0.9759	0.9755

Utilization (U)		
	VOIP1	GLOBALS

DSP1	0.9592	0.9592
DSP2	0.9601	0.9601
DSP3	0.9545	0.9545
Processor	0.01724	0.01724
NetControl	0.01149	0.01149
GLOBALS	0	0

Waiting Time (W)

	VOIP1	GLOBALS
DSP1	0.9891	0.9891
DSP2	1.034	1.034
DSP3	0.8965	0.8965
Processor	0.001502	0.001502
NetControl	0.001001	0.001001
GLOBALS	0	0

9.1.3 $s11 = s12 = s13 > s2 > s3$, Routing : $r1=2*r2=3*r3$

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	4.672	4.672
DSP2	0.5743	0.5743
DSP3	0.3579	0.3579
Processor	0.021	0.021
NetControl	0.014	0.014
GLOBALS	5.639	5.639

Throughput (X)

	VOIP1	GLOBALS
DSP1	3.823	3.823
DSP2	1.903	1.903
DSP3	1.273	1.273
Processor	6.998	6.998
NetControl	6.998	6.998
GLOBALS	7	7

Response Time (R)

	VOIP1	GLOBALS
DSP1	0.6676	0.6674
DSP2	0.08206	0.08204
DSP3	0.05114	0.05112
Processor	0.003001	0.003
NetControl	0.002	0.002
GLOBALS	0.8058	0.8056

Utilization (U)

	VOIP1	GLOBALS
DSP1	0.9557	0.9557
DSP2	0.4757	0.4757
DSP3	0.3182	0.3182
Processor	0.0105	0.0105
NetControl	0.006998	0.006998
GLOBALS	0	0

Waiting Time (W)

	VOIP1	GLOBALS
DSP1	1.222	1.222
DSP2	0.3018	0.3018
DSP3	0.2812	0.2812

Processor	0.003001	0.003001
NetControl	0.002	0.002
GLOBALS	0	0

9.1.4 Equal Routing Probability, Routing : s11>s12>s13

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	1.962	1.962
DSP2	0.4591	0.4591
DSP3	0.2078	0.2078
Processor	0.015	0.015
NetControl	0.009998	0.009998
GLOBALS	2.654	2.654

Throughput (X)

	VOIP1	GLOBALS
DSP1	1.669	1.669
DSP2	1.668	1.668
DSP3	1.662	1.662
Processor	4.999	4.999
NetControl	4.999	4.999
GLOBALS	5	5

Response Time (R)

	VOIP1	GLOBALS
DSP1	0.3925	0.3924
DSP2	0.09184	0.09182
DSP3	0.04157	0.04156
Processor	0.003	0.002999
NetControl	0.002	0.002
GLOBALS	0.5309	0.5308

Utilization (U)

	VOIP1	GLOBALS
DSP1	0.8347	0.8347
DSP2	0.4169	0.4169
DSP3	0.2078	0.2078
Processor	0.007498	0.007498
NetControl	0.004999	0.004999
GLOBALS	0	0

Waiting Time (W)

	VOIP1	GLOBALS
DSP1	1.175	1.175
DSP2	0.2753	0.2753
DSP3	0.125	0.125
Processor	0.003	0.003
NetControl	0.002	0.002
GLOBALS	0	0

9.1.5 $r1=2$ $r2=4r3$, $s11>s12>s13$

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	12.41	12.41
DSP2	0.2069	0.2069
DSP3	0.05429	0.05429
Processor	0.008983	0.008983
NetControl	0.005989	0.005989
GLOBALS	12.68	12.68

Throughput (X)

	VOIP1	GLOBALS
DSP1	1.732	1.732
DSP2	0.8277	0.8277
DSP3	0.4343	0.4343
Processor	2.994	2.994
NetControl	2.994	2.994
GLOBALS	3	3

Response Time (R)

	VOIP1	GLOBALS
DSP1	4.143	4.135
DSP2	0.0691	0.06897
DSP3	0.01813	0.0181
Processor	0.003	0.002994
NetControl	0.002	0.001996
GLOBALS	4.235	4.227

Utilization (U)

	VOIP1	GLOBALS
DSP1	0.99	0.99
DSP2	0.2069	0.2069
DSP3	0.05429	0.05429
Processor	0.004492	0.004492
NetControl	0.002994	0.002994
GLOBALS	0	0

Waiting Time (W)

	VOIP1	GLOBALS
DSP1	7.161	7.161
DSP2	0.25	0.25

DSP3	0.125	0.125
Processor	0.003	0.003
NetControl	0.002	0.002
GLOBALS	0	0

9.1.6 $r1=2$ $r2=3r3$, $s11>s12>s13$

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	6.633	6.633
DSP2	0.2385	0.2385
DSP3	0.07866	0.07866
Processor	0.01049	0.01049
NetControl	0.006991	0.006991
GLOBALS	6.968	6.968

Throughput (X)

	VOIP1	GLOBALS
DSP1	1.912	1.912
DSP2	0.9539	0.9539
DSP3	0.6293	0.6293
Processor	3.495	3.495
NetControl	3.495	3.495
GLOBALS	3.5	3.5

Response Time (R)

	VOIP1	GLOBALS
DSP1	1.898	1.895
DSP2	0.06823	0.06814
DSP3	0.0225	0.02247
Processor	0.003	0.002996
NetControl	0.002	0.001997
GLOBALS	1.993	1.991

Utilization (U)

	VOIP1	GLOBALS
DSP1	0.9562	0.9562
DSP2	0.2385	0.2385
DSP3	0.07866	0.07866
Processor	0.005243	0.005243
NetControl	0.003495	0.003495
GLOBALS	0	0

Waiting Time (W)

	VOIP1	GLOBALS
DSP1	3.469	3.469
DSP2	0.25	0.25
DSP3	0.125	0.125
Processor	0.003	0.003
NetControl	0.002	0.002
GLOBALS	0	0

9.1.7 $r1=2$ $r2=3r3$, $s11<s12<s13$

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	0.2396	0.2396
DSP2	0.2387	0.2387
DSP3	0.3469	0.3469
Processor	0.0105	0.0105
NetControl	0.006999	0.006999
GLOBALS	0.8427	0.8427

Throughput (X)

	VOIP1	GLOBALS
DSP1	1.916	1.916
DSP2	0.9549	0.9549
DSP3	0.6279	0.6279
Processor	3.499	3.499
NetControl	3.499	3.499
GLOBALS	3.5	3.5

Response Time (R)

	VOIP1	GLOBALS
DSP1	0.06846	0.06845
DSP2	0.06822	0.06821
DSP3	0.09913	0.09911
Processor	0.003	0.002999
NetControl	0.002	0.002
GLOBALS	0.2408	0.2408

Utilization (U)

	VOIP1	GLOBALS
DSP1	0.2396	0.2396
DSP2	0.2387	0.2387
DSP3	0.3141	0.3141
Processor	0.005249	0.005249
NetControl	0.003499	0.003499
GLOBALS	0	0

Waiting Time (W)

	VOIP1	GLOBALS
DSP1	0.125	0.125
DSP2	0.25	0.25

DSP3	0.5524	0.5524
Processor	0.003	0.003
NetControl	0.002	0.002
GLOBALS	0	0

9.1.8 $r1=r2=r3, s11<s12<s13$

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	0.2514	0.2514
DSP2	0.6006	0.6006
DSP3	32.52	32.52
Processor	0.01796	0.01796
NetControl	0.01197	0.01197
GLOBALS	33.4	33.4

Throughput (X)

	VOIP1	GLOBALS
DSP1	2.011	2.011
DSP2	1.993	1.993
DSP3	1.983	1.983
Processor	5.987	5.987
NetControl	5.987	5.987
GLOBALS	6	6

Response Time (R)		
	VOIP1	GLOBALS
DSP1	0.04199	0.0419
DSP2	0.1003	0.1001
DSP3	5.431	5.419
Processor	0.003	0.002993
NetControl	0.002	0.001996
GLOBALS	5.579	5.566

Utilization (U)		
	VOIP1	GLOBALS
DSP1	0.2514	0.2514
DSP2	0.4981	0.4981
DSP3	0.9916	0.9916
Processor	0.00898	0.00898
NetControl	0.005987	0.005987
GLOBALS	0	0

Waiting Time (W)		
	VOIP1	GLOBALS
DSP1	0.125	0.125
DSP2	0.3014	0.3014
DSP3	16.4	16.4
Processor	0.003	0.003
NetControl	0.002	0.002
GLOBALS	0	0

9.1.9 $s11=s12=s13, s11 > s2 > s3, r1=r2=r3$

Analysis results for Simulation:

Queue Length (n)		
	VOIP1	GLOBALS
DSP1	0.6033	0.6033
DSP2	0.5892	0.5892
DSP3	0.5998	0.5998
Processor	2.128	2.128
NetControl	3000	3000
GLOBALS	3004	3004

Throughput (X)		
	VOIP1	GLOBALS
DSP1	2.011	2.011
DSP2	1.986	1.986
DSP3	2.002	2.002
Processor	5.998	5.998
NetControl	3.999	3.999
GLOBALS	6	6

Response Time (R)		
	VOIP1	GLOBALS
DSP1	0.1509	0.1005
DSP2	0.1474	0.0982
DSP3	0.15	0.09997
Processor	0.5322	0.3547
NetControl	750.2	500
GLOBALS	751.2	500.6

Utilization (U)		
	VOIP1	GLOBALS
DSP1	0.5028	0.5028
DSP2	0.4964	0.4964
DSP3	0.5005	0.5005
Processor	0.8997	0.8997
NetControl	0.9997	0.9997
GLOBALS	0	0

Waiting Time (W)		
	VOIP1	GLOBALS
DSP1	0.3	0.3
DSP2	0.2967	0.2967
DSP3	0.2996	0.2996
Processor	0.3548	0.3548
NetControl	750.2	750.2
GLOBALS	0	0

9.1.10 $s11=s12=s13, S11<s2>s3 r1=r2=r3$

Analysis results for Simulation:

Queue Length (n)		
	VOIP1	GLOBALS
DSP1	0.3334	0.3334
DSP2	0.3327	0.3327
DSP3	0.3337	0.3337
Processor	1.2	1.2
NetControl	1.08	1.08
GLOBALS	3.279	3.279

Throughput (X)		
	VOIP1	GLOBALS
DSP1	1.333	1.333
DSP2	1.331	1.331
DSP3	1.335	1.335
Processor	3.999	3.999
NetControl	3.999	3.999
GLOBALS	4	4

Response Time (R)		
	VOIP1	GLOBALS
DSP1	0.08338	0.08335
DSP2	0.08322	0.08319
DSP3	0.08347	0.08344
Processor	0.3	0.2999
NetControl	0.27	0.2699
GLOBALS	0.8201	0.8199

Utilization (U)		
	VOIP1	GLOBALS
DSP1	0.3334	0.3334
DSP2	0.3327	0.3327
DSP3	0.3337	0.3337
Processor	0.5999	0.5999
NetControl	0.5399	0.5399
GLOBALS	0	0

Waiting Time (W)		
	VOIP1	GLOBALS
DSP1	0.2501	0.2501
DSP2	0.25	0.25
DSP3	0.25	0.25
Processor	0.3	0.3
NetControl	0.27	0.27
GLOBALS	0	0

9.1.11 $r1=2$ $r2=3r3$, $s11=2s12=3s13$

Analysis results for Simulation:

Queue Length (n)		
	VOIP1	GLOBALS
DSP1	429.6	429.6
DSP2	0.2616	0.2616
DSP3	0.1155	0.1155
Processor	0.04113	0.04113
NetControl	0.03701	0.03701
GLOBALS	430	430

Throughput (X)		
	VOIP1	GLOBALS
DSP1	7.331	7.331
DSP2	3.837	3.837
DSP3	2.541	2.541
Processor	13.71	13.71
NetControl	13.71	13.71
GLOBALS	14	14

Response Time (R)		
	VOIP1	GLOBALS
DSP1	31.33	30.68
DSP2	0.01908	0.01869
DSP3	0.008426	0.008251
Processor	0.003	0.002938
NetControl	0.0027	0.002644
GLOBALS	31.37	30.72

Utilization (U)		
	VOIP1	GLOBALS
DSP1	0.9997	0.9997
DSP2	0.2616	0.2616
DSP3	0.1155	0.1155
Processor	0.02056	0.02056
NetControl	0.01851	0.01851
GLOBALS	0	0

Waiting Time (W)		
	VOIP1	GLOBALS
DSP1	58.59	58.59
DSP2	0.06818	0.06818
DSP3	0.04545	0.04545
Processor	0.003	0.003
NetControl	0.0027	0.0027
GLOBALS	0	0

9.1.12 $r1=2$ $r2=3r3$, $s11=2s12=3s13$

Analysis results for Simulation:

Queue Length (n)

	VOIP1	GLOBALS
DSP1	2.433	2.433
DSP2	0.2226	0.2226
DSP3	0.09915	0.09915
Processor	0.07199	0.07199
NetControl	0.0648	0.0648
GLOBALS	2.891	2.891

Throughput (X)

	VOIP1	GLOBALS
DSP1	13.11	13.11
DSP2	6.529	6.529
DSP3	4.362	4.362
Processor	24	24
NetControl	24	24
GLOBALS	24	24

Response Time (R)

	VOIP1	GLOBALS
DSP1	0.1014	0.1014
DSP2	0.009275	0.009274
DSP3	0.004131	0.004131
Processor	0.003	0.003
NetControl	0.0027	0.0027
GLOBALS	0.1205	0.1205

Utilization (U)

	VOIP1	GLOBALS
DSP1	0.8937	0.8937
DSP2	0.2226	0.2226
DSP3	0.09915	0.09915
Processor	0.036	0.036
NetControl	0.0324	0.0324
GLOBALS	0	0

Waiting Time (W)

	VOIP1	GLOBALS
DSP1	0.1856	0.1856
DSP2	0.03409	0.03409
DSP3	0.02273	0.02273
Processor	0.003	0.003
NetControl	0.0027	0.0027
GLOBALS	0	0

9.2 Win Modelling Tool - ARCHITECTURE 2

9.2.1 $S = 0.255$, $x=1:0.1:4$

Analysis results for Simulation:

Queue Length (n)		
	VOIP2	GLOBALS
CPU	28.64	28.64
GLOBALS	28.64	28.64

Throughput (X)		
	VOIP2	GLOBALS
CPU	3.921	3.921
GLOBALS	3.94	3.94

Response Time (R)		
	VOIP2	GLOBALS
CPU	7.305	7.27
GLOBALS	7.305	7.27

Utilization (U)		
	VOIP2	GLOBALS
CPU	0.9999	0.9999
GLOBALS	0	0

Waiting Time (W)		
	VOIP2	GLOBALS
CPU	7.305	7.305
GLOBALS	0	0

9.2.2 $S = 0.255/2$, $x=1:0.1:9$

Analysis results for Simulation:

Queue Length (n)		
	VOIP2	GLOBALS
CPU	1736	1736
GLOBALS	1736	1736

Throughput (X)		
	VOIP2	GLOBALS
CPU	7.843	7.843
GLOBALS	9	9

Response Time (R)		
	VOIP2	GLOBALS
CPU	221.4	192.9
GLOBALS	221.4	192.9

Utilization (U)		
	VOIP2	GLOBALS
CPU	1	1
GLOBALS	0	0

Waiting Time (W)		
	VOIP2	GLOBALS

CPU	221.4	221.4
GLOBALS	0	0

9.2.3 $s3=0.255, x=1:0.1:7.5$

Analysis results for Simulation:

Queue Length (n)

	VOIP2	GLOBALS
CPU	118.6	118.6
GLOBALS	118.6	118.6

Throughput (X)

	VOIP2	GLOBALS
CPU	3.921	3.921
GLOBALS	4	4

Response Time (R)

	VOIP2	GLOBALS
CPU	30.25	29.66
GLOBALS	30.25	29.66

Utilization (U)

	VOIP2	GLOBALS
CPU	0.9999	0.9999
GLOBALS	0	0

Waiting Time (W)

	VOIP2	GLOBALS
CPU	30.25	30.25
GLOBALS	0	0

5 – Security

5.1 – Encryption Management in Wireless Personal Area Networks 206

5.2 – Modeling the Performance of SCADA Field Devices 232

Università della Svizzera Italiana

Advanced Learning and Research Institute

MAS 2006/07

Performance Evaluation (teacher: Giuseppe SERAZZI)

Julien CAMISANI – Fabio DE RICCARDIS

Performance evaluation of encryption management in a beacon enabled low-rate wireless personal area network (LR-WPAN) with star topology through queuing network modelling

1 summary

1	summary	1
2	scope.....	1
3	IEEE Std 802.15.4-2003.....	2
3.1	overview	2
3.2	general description	2
3.3	components of the IEEE 802.15.4 WPAN	2
3.4	network topologies	2
3.5	functional overview	3
4	literature exploration	7
5	queuing network model	11
5.1	simulation assumptions	11
5.2	channel	11
5.3	microcontroller unit.....	13
6	simulation results.....	14
6.1	M/M/n model, uplink mode.....	14
6.2	M/M/n model, downlink mode.....	17
6.3	M/M/n model, joint uplink and downlink mode.....	20
7	conclusions	23
8	further work.....	23
9	acronyms and abbreviations	24
10	references	24
11	appendix	25

2 scope

Scope of this simulation is to assess the performance of an encryption management in a beacon enabled low-rate wireless personal area network (LR-WPAN) with star topology by means of queuing networks theory, and possibly evaluate the maximum level of encryption of the channel possible for given a microcontroller unit.

The simulation will initially assess the performance of a LR-WPAN under unencrypted channel conditions.

Afterwards, a *what-if analysis* will be carried out to explore the performance of a LR-WPAN under managed encrypted channel conditions.

3 IEEE Std 802.15.4-2003

IEEE Std 802.15.4 defines the protocol and interconnection of devices via radio communication in a personal area network (PAN). The standard uses carrier sense multiple access with a collision avoidance medium access (CSMA/CA) mechanism and supports star as well as peer-to-peer topologies. The media access is contention based; however, using the optional superframe structure, time slots can be allocated by the PAN coordinator to devices with time critical data. Connectivity to higher performance networks is provided through a PAN coordinator.

This standard specifies two PHYs: an 868/915 MHz direct sequence spread spectrum (DSSS) PHY and a 2450 MHz DSSS PHY. The 2450 MHz PHY supports an over-the-air data rate of 250 kb/s, and the 868/915 MHz PHY supports over-the-air data rates of 20 kb/s and 40 kb/s. The PHY chosen depends on local regulations and user preference.

3.1 overview

Wireless personal area networks (WPANs) are used to convey information over relatively short distances. Unlike wireless local area networks (WLANs), connections effected via WPANs involve little or no infrastructure. This feature allows small, power-efficient, inexpensive solutions to be implemented for a wide range of devices.

IEEE Std 802.15.4 defines a standard for a low-rate WPAN (LR-WPAN).

IEEE Std 802.15.4 defines the physical layer (PHY) and medium access control (MAC) sublayer specifications for low data rate wireless connectivity with fixed, portable, and moving devices with no battery or very limited battery consumption requirements typically operating in the personal operating space (POS) of 10 m. It is foreseen that, depending on the application, a longer range at a lower data rate may be an acceptable trade-off.

IEEE Std 802.15.4 provides a standard for ultra-low complexity, ultra-low cost, ultra-low power consumption, and low data rate wireless connectivity among inexpensive devices. The raw data rate will be high enough (maximum of 250 kb/s) to satisfy a set of simple needs such as interactive toys, but scalable down to the needs of sensor and automation needs (20 kb/s or below) for wireless communications.

3.2 general description

A LR-WPAN is a simple, low-cost communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements. The main objectives of an LR-WPAN are ease of installation, reliable data transfer, short-range operation, extremely low cost, and a reasonable battery life, while maintaining a simple and flexible protocol.

Two different device types can participate in an LR-WPAN network:

- a full-function device (FFD) and
- a reduced-function device (RFD).

The FFD can operate in three modes serving as

- 1) a personal area network (PAN) coordinator,
- 2) a coordinator, or
- 3) a device.

An FFD can talk to

- RFDs or
- other FFDs,

while an RFD can

- talk only to an FFD.

An RFD is intended for applications that are extremely simple, such as a light switch or a passive infrared sensor; they do not have the need to send large amounts of data and may only associate with a single FFD at a time. Consequently, the RFD can be implemented using minimal resources and memory capacity.

3.3 components of the IEEE 802.15.4 WPAN

A system conforming to IEEE 802.15.4 consists of several components. The most basic is the device. A device can be an RFD or an FFD. Two or more devices within a POS communicating on the same physical channel constitute a WPAN. However, a network shall include at least one FFD, operating as the PAN coordinator.

A well-defined coverage area does not exist for wireless media because propagation characteristics are dynamic and uncertain. Small changes in position or direction may result in drastic differences in the signal strength or quality of the communication link. These effects occur whether a device is stationary or mobile as moving objects may impact station-to-station propagation.

3.4 network topologies

Depending on the application requirements, the LR-WPAN may operate in either of two topologies: the *star topology* or the *peer-to-peer topology*.

All devices operating on a network of *either topology* shall have unique 64 bit extended addresses.

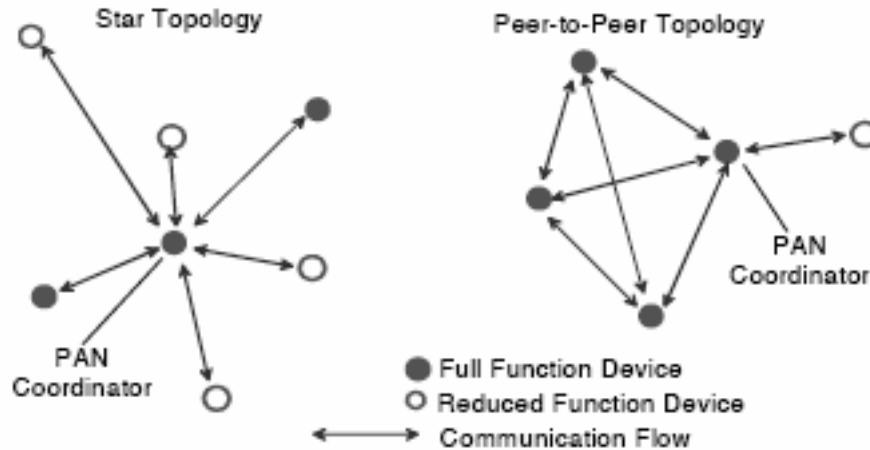


Figure 1: star and peer-to-peer topology examples [2].

In the *star topology* the communication is established between devices and a single central controller, called the PAN coordinator. A device typically has some associated application and is either the initiation point or the termination point for network communications. A PAN coordinator may also have a specific application, but it can be used to initiate, terminate, or route communication around the network. The PAN coordinator is the primary controller of the PAN. The address can be used for direct communication within the PAN, or it can be exchanged for a short address allocated by the PAN coordinator when a device associates. The PAN coordinator may be mains powered, while the devices will most likely be battery powered. Applications that benefit from a star topology include home automation, personal computer (PC) peripherals, toys and games, and personal health care.

The *peer-to-peer topology* also has a PAN coordinator; however, it differs from the star topology in that any device can communicate with any other device as long as they are in range of one another. Peer-to-peer topology allows more complex network formations to be implemented, such as mesh networking topology. Applications such as industrial control and monitoring, wireless sensor networks, asset and inventory tracking, intelligent agriculture, and security would benefit from such a network topology. A peer-to-peer network can be ad hoc, self-organizing and self-healing. It may also allow multiple hops to route messages from any device to any other device on the network. Such functions can be added at the network layer, but are not part of this standard.

3.5 functional overview

3.5.1 superframe structure

The LR-WPAN standard allows the optional use of a superframe structure. The format of the superframe is defined by the coordinator. The superframe is bounded by network beacons, is sent by the coordinator, and is divided into 16 equally sized slots. The beacon frame is transmitted in the first slot of each superframe. If a coordinator does not wish to use a superframe structure, it may turn off the beacon transmissions. The beacons are used to synchronize the attached devices, to identify the PAN, and to describe the structure of the superframes. Any device wishing to communicate during the contention access period (CAP) between two beacons shall compete with other devices using a slotted CSMA-CA mechanism. All transactions shall be completed by the time of the next network beacon.

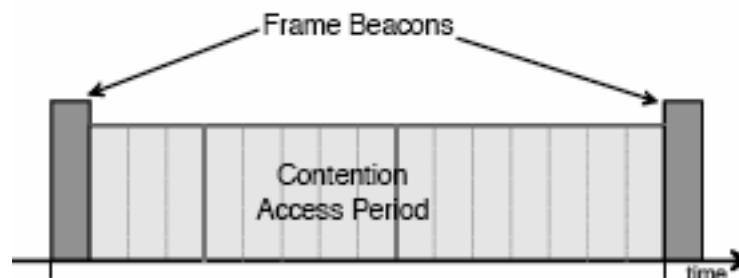


Figure 2: superframe structure without GTSS [2].

The superframe can have an active and an inactive portion. During the inactive portion, the coordinator shall not interact with its PAN and may enter a low-power mode.

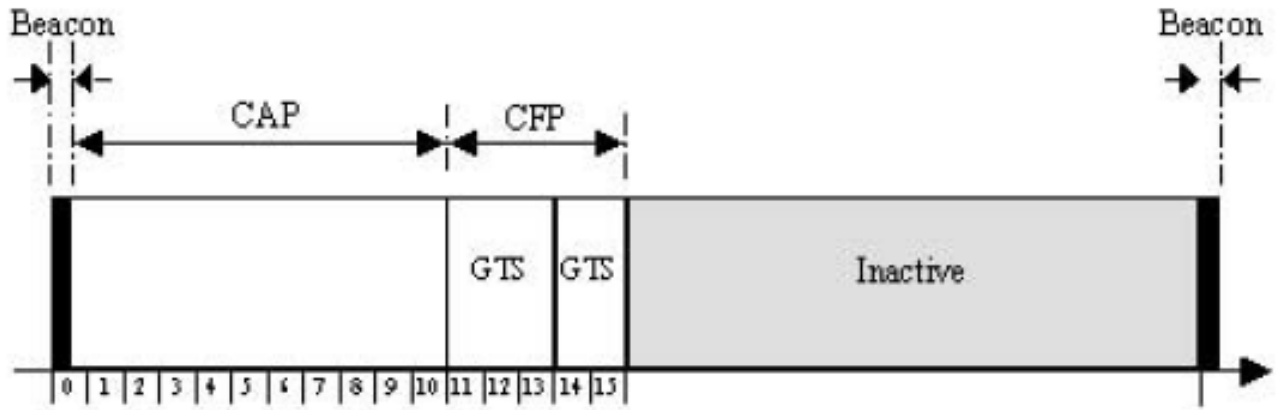


Figure 3: an example of the superframe structure [2].

3.5.2 data transfer model

Three types of data transfer transactions exist.

- 1) The first one is the data transfer to a coordinator in which a device transmits the data.
- 2) The second transaction is the data transfer from a coordinator in which the device receives the data.
- 3) The third transaction is the data transfer between two peer devices.

In star topology only two of these transactions are used, because data may be exchanged only between the coordinator and a device. In a peer-to-peer topology data may be exchanged between any two devices on the network; consequently all three transactions may be used in this topology.

The mechanisms for each transfer type depend on whether the network supports the transmission of beacons. A beacon-enabled network is used for supporting low-latency devices, such as PC peripherals. If the network does not need to support such devices, it can elect not to use the beacon for normal transfers. However, the beacon is still required for network association.

3.5.2.1 data transfer to a coordinator in a beacon-enabled network

This data transfer transaction is the mechanism to transfer data from a device to a coordinator.

When a device wishes to transfer data to a coordinator in a beacon-enabled network, it first listens for the network beacon. When the beacon is found, the device synchronizes to the superframe structure. At the appropriate point, the device transmits its data frame, using slotted CSMA-CA, to the coordinator. The coordinator acknowledges the successful reception of the data by transmitting an optional acknowledgment frame. The transaction is now complete.

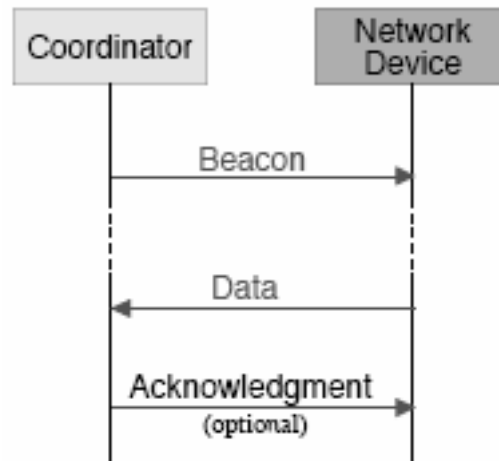


Figure 4: communication to a coordinator (*uplink*) in a beacon-enabled network [2].

3.5.2.2 data transfer from a coordinator in a beacon-enabled network

This data transfer transaction is the mechanism for transferring data from a coordinator to a device.

When the coordinator wishes to transfer data to a device in a beacon-enabled network, it indicates in the network beacon that the data message is pending. The device periodically listens to the network beacon and, if a message is pending, transmits a MAC command requesting the data, using slotted CSMA-CA. The coordinator acknowledges the

successful reception of the data request by transmitting an optional acknowledgment frame. The pending data frame is then sent using slotted CSMA-CA. The device acknowledges the successful reception of the data by transmitting an acknowledgment frame. The transaction is now complete. Upon receiving the acknowledgement, the message is removed from the list of pending messages in the beacon.

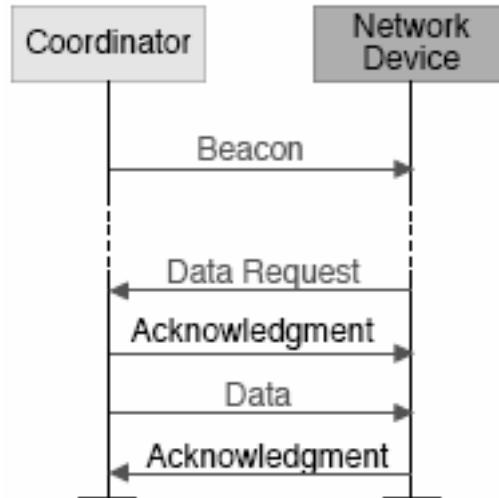


Figure 5: communication from a coordinator (*downlink*) in a beacon-enabled network [2].

3.5.3 frame structure

The frame structures have been designed to keep the complexity to a minimum while at the same time making them sufficiently robust for transmission on a noisy channel. Each successive protocol layer adds to the structure with layer-specific headers and footers. The LR-WPAN defines four frame structures:

- A beacon frame, used by a coordinator to transmit beacons
- A data frame, used for all transfers of data
- An acknowledgment frame, used for confirming successful frame reception
- A MAC command frame, used for handling all MAC peer entity control transfers

3.5.4 CSMA-CA mechanism

The LR-WPAN uses two types of channel access mechanism, depending on the network configuration.

Beacon-enabled networks use a slotted CSMA-CA channel access mechanism, where the backoff slots are aligned with the start of the beacon transmission. Each time a device wishes to transmit data frames during the CAP, it shall locate the boundary of the next backoff slot and then wait for a random number of backoff slots. If the channel is busy, following this random backoff, the device shall wait for another random number of backoff slots before trying to access the channel again. If the channel is idle, the device can begin transmitting on the next available backoff slot boundary. Acknowledgment and beacon frames shall be sent without using a CSMA-CA mechanism.

3.5.5 frame acknowledgment

A successful reception and validation of a data or MAC command frame can be *optionally* confirmed with an acknowledgment. If the receiving device is unable to handle the received data frame for any reason, the message is not acknowledged.

If the originator does not receive an acknowledgment after some period, it assumes that the transmission was unsuccessful and retries the frame transmission. If an acknowledgment is still not received after several retries, the originator can choose either to terminate the transaction or to try again. When the acknowledgment is not required, the originator assumes the transmission was successful.

3.5.6 power consumption considerations

In many applications that use this standard, the devices will be battery powered where their replacement or recharging in relatively short intervals is impractical; therefore the power consumption is of significant concern. This standard was developed with the limited power supply availability in mind. However, the physical implementation of this standard will require additional power management considerations that are beyond the scope of this standard.

The protocol has been developed to favor battery-powered devices. However, in certain applications some of these devices could potentially be mains powered. Battery-powered devices will require duty-cycling to reduce power consumption. These devices will spend most of their operational life in a sleep state; however, each device shall

periodically listen to the RF channel in order to determine whether a message is pending. This mechanism allows the application designer to decide on the balance between battery consumption and message latency. Mains-powered devices have the option of listening to the RF channel continuously.

3.5.7 data encryption

In this standard data encryption is a security service that uses a symmetric cipher to protect data from being read by parties without the cryptographic key. Data may be encrypted using a key shared by a group of devices (typically stored as the default key) or using a key shared between two peers (typically stored in an individual ACL entry). In this standard, data encryption may be provided on beacon payloads, command payloads, and data payloads.

4 literature exploration

In this paragraph, some reference papers dealing with performance evaluation of IEEE Std 802.15.4 have been selected and reported hereunder by approach: analytical, simulative and experimental.

4.1.1 analytical approach

The performance of the slotted CSMA/CA mechanism in IEEE 802.15.4 has recently been analytically investigated using discrete time Markov chain (DTMC) models in [3], [4], [6]. Those works have provided analytic models of the slotted CSMA/CA mechanism in both saturation and non saturation modes, and also provided steady state solutions. These analytical models are interesting for capturing the behaviour of the protocol in terms of throughput and access delays

[3] proposed a new discrete time Markov chain model of 802.15.4, and analysed the throughput and energy consumption in saturation conditions. The proposed model utilises the probability of a device in the channel sensing states instead of the channel accessing states.

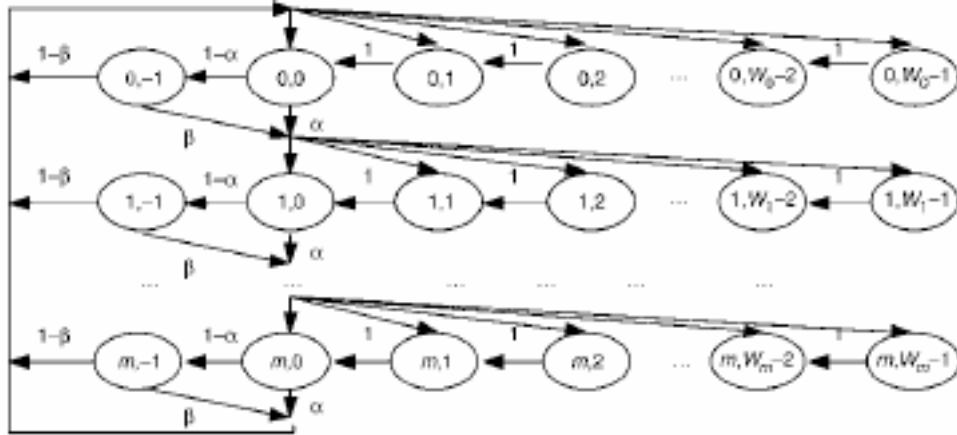


Figure 6: Markov chain model of IEEE 802.15.4 slotted CSMA/CA [3].

The key approximation is that the busy probability of the channel at the first CCA and at the second CCA are α and β , respectively, regardless of the stages. Then, the saturation throughput is derived as

$$S = \frac{n\tau(1-\alpha)(1-\beta)\gamma l_p}{(1-\tau) + \tau\alpha + 2\tau(1-\alpha)\beta + \tau(1-\alpha)(1-\beta)[\gamma T_s + (1-\gamma)T_c]}$$

where

- n : devices in the backoff states
- τ : conditional probability that a device is at one of the first CCA states
- γ : probability that the given device transmits a packet successfully after performing CCAs twice
- l_p : payload length in the number of slots
- T_s and T_c : number of occupied slots for successful transmission and collision, respectively

Incidentally, [3] utilises the channel sensing probabilities and criticises the approach of a reference work of the authors of [6] utilising instead the channel accessing probabilities.

In [4], the key approximation in the model is the independence of the carrier sensing probability that determines when the nodes become active to listen to the channel.

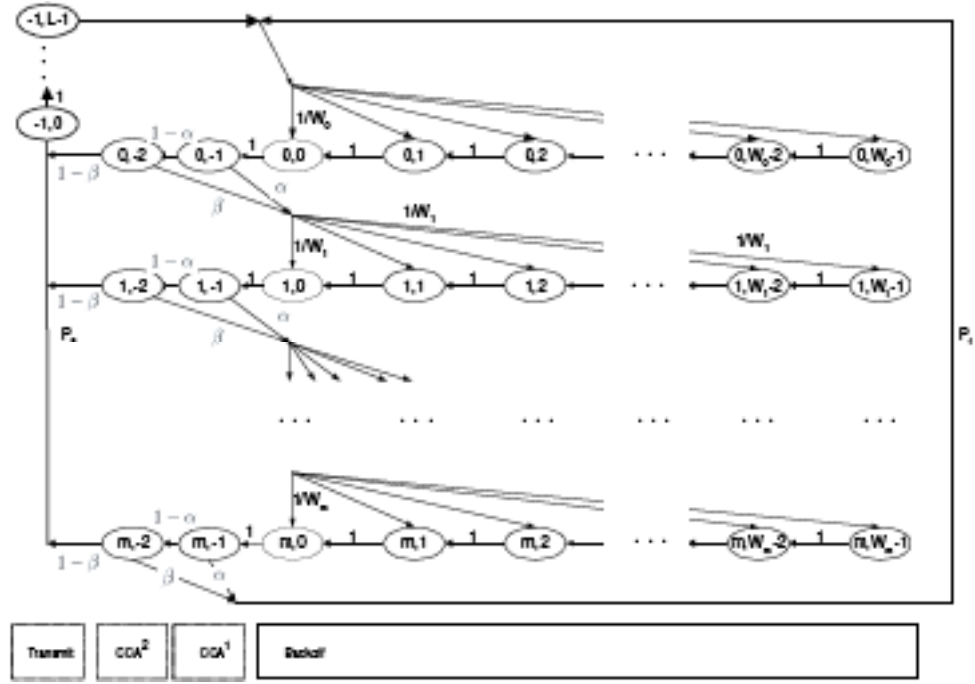


Figure 7: Markov chain model of IEEE 802.15.4 slotted CSMA/CA [4].

The saturation throughput S is expressed as the number of slots occupied for a successful packet transmission:

$$S = LN\tau(1 - \tau)^{N-1}(1 - \alpha)(1 - \beta)$$

where

- L : packet size
- N : number of devices (each device always having a packet available for transmission)
- τ : stationary probability that the device attempts carrier channel assessment (CCA) for the first time within a slot
- α : probability of assessing channel busy during CCA1
- β : probability of assessing channel busy during CCA2, given that it was idle in CCA1

Incidentally, [4] criticises a reference work of the authors of [6] underlining that their analytical model seems to fail to match the simulation results.

[6] combines the theory of discrete-time Markov chains and the theory of M/G/1 queues to derive the probability distributions of packet service times, with both downlink and uplink traffic.

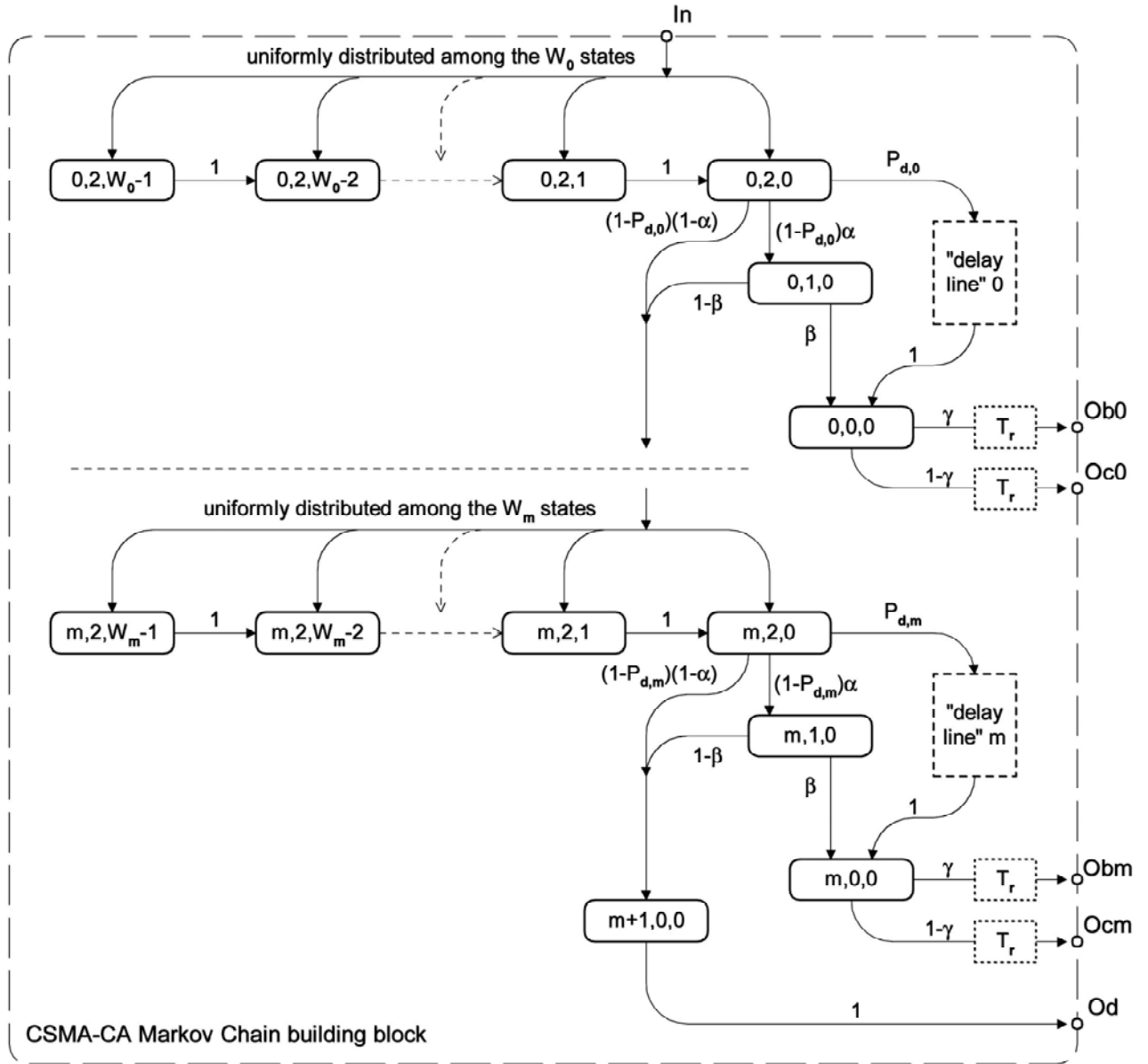


Figure 8: general Markov chain model of the slotted CSMA-CA algorithm representing a nonidle state of the node (uplink data transmission, uplink request transmission, or downlink data transmission) [6].

The packet queues in the device data buffer and request buffer are modelled as a M/G/1 queuing system, in which the packet request queue has non-preemptive priority over the data queue at the device. (It may be argued that the M/G/1/K system would be more accurate, but the increase in complexity would be unjustifiably high.) **Both uplink and downlink packet arrivals follow a Poisson process with the average arrival rate of λ_{iu} and λ_{id} , respectively.**

The service time for uplink data packets T_{ud} is obtained as

$$T_{ud}(z) = \frac{\gamma_u P(z)}{1 - R_{ep}(z) - (1 - \gamma_u)P(z)}$$

where:

- γ_u : uplink success probability; denotes the probability that an uplink transmission will not experience a collision
- $P(z)$: PGF that describes the time needed for the backoff countdown before a transmission attempt and the transmission attempt itself
- $R_{ep}(z)$: function that represents $m + 1$ unsuccessful backoff countdown iterations without a transmission attempt and requires that the backoff procedure is repeated

The service time for uplink request packets T_{ur} is obtained as

$$T_{ur}(z) = \frac{\gamma_u(1 - P_B)P^{ur}(z)}{1 - R_{epu}(z) - (1 - \gamma_u(1 - P_B))P^{ur}(z)}$$

where:

- γ_u : uplink success probability; denotes the probability that an uplink transmission will not experience a collision
- P_B : blocking probability
- P^{ur} : PGF-like function that describes the time for the backoff procedure and subsequent transmission attempt
- $R_{epu}(z)$: PGF-like function

The service time for downlink packets T_{dd} is obtained as

$$T_{dd}(z) = \frac{T_{ur}(z)\gamma_d P^{dd}(z)}{1 - R_{epd}(z) - T_{ur}(z)(1 - \gamma_d)P^{dd}(z)}$$

where:

- T_{ur} : service time for uplink request packets
- γ_d : downlink success probability; denotes the probability that a downlink transmission will not experience a collision
- $P^{dd}(z)$: PGF-like function to describe the time needed to backoff countdown and the subsequent transmission attempt
- $R_{epd}(z)$: function which represents $m + 1$ unsuccessful backoffs without the transmission attempt

4.1.2 simulative approach

In [24] the authors have

- evaluated the performance of slotted CSMA/CA using simulations and
- presented results
 - without doing restrictive assumptions and
 - taking into account some realistic features of the physical layer (propagation delays, fading, noise effect, etc.).

4.1.3 experimental approach

In [5], after an overview of the IEEE 802.15.4 wireless networks, it has been made a preliminary performance study via several sets of practical experiments.

[7] shows experimental results of a reference implementation.

5 queuing network model

The implementation of IEEE Std 802.15.4 is complex, and an attempt to incorporate a detailed representation in a queuing network model would be ill-advised [1]. Therefore, a simplified queuing network model is presented in the following under some representative operating conditions extracted from [9].

5.1 simulation assumptions

The performance evaluation of the LR_WPAN have been simulated under the following assumptions:

- number of WPANs: 1
- operation topology: star
- number of FFDs: 1, operating as a PAN coordinator
- number of RDDs: $\geq 1, \leq 2^{16}-1$
- network configuration: beacon-enabled
- channel access mechanism: slotted CSMA/CA
- BO=0, SO=0 (\Rightarrow duty cycle = 100%, active portion = 0.01536s = 15,36 ms \approx 16 ms = 16 x 1 ms)
- number of channel slots: 16
- media access: contention based
- physical layer: 2450 MHz direct sequence spread spectrum (DSSS)
- allocation of guaranteed time slots (GTSs): no
- allocated address bits: 16 (short address)
- acknowledgment: not required
- duty cycling: not required

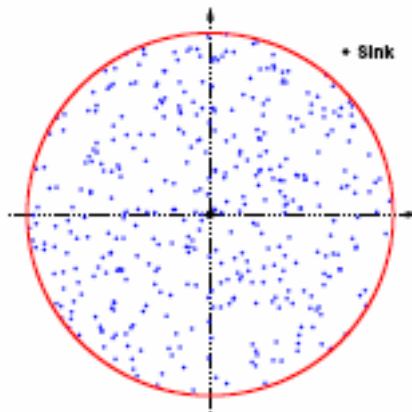


Figure 9: network topology of the reference scenario.

5.2 channel

We propose two queuing network models for the channel:

- M/M/n model:
the channel is modelled as an array of servers, one server for each packet allotted every frame;
- load dependent:
the channel is modelled as a server with service time increasing along with its corresponding service load.

5.2.1 M/M/n model

- inter-arrival times

The literature suggests that the packet inter-arrival times distribution is Poisson (see e.g. [3-8]).

- service times

Unfortunately, to the best of our knowledge, the literature does not provide a common distribution for the service times. We will assume they also follow a Poisson distribution.

- number of servers

In this model, we assume that packets are exactly one slot long (i.e. packet length = 1 slot). Under this assumption, the channel (16 slot wide) can be modelled as an array of 16 servers with a single queue serving one packet per frame.

In case the system is better modelled by assuming packet lengths of 2/4/8/16 slots, the channel shall be modelled as an array of 8/4/2/1 server(s), respectively.

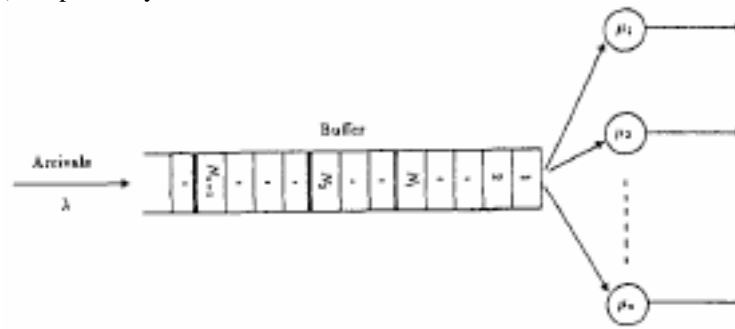


Figure 10: M/M/n model of the channel.

- queue capacity

As the requests come from an undetermined number of RFDs, we will assume the queue capacity infinite.

To sum up:

- inter-arrival times distribution: Poisson
- service times distribution: Poisson
- number of servers: 16
- queue capacity: infinite

5.2.2 load dependent model

- inter-arrival times

The literature suggests that the packet inter-arrival times distribution is Poisson (see e.g. [3-8]).

- service times

Unfortunately, to the best of our knowledge, the literature does not provide a common distribution for the service times. We will assume they also follow a Poisson distribution, with a load dependent strategy.



Figure 11: load dependent model of the channel (FESC: flow equivalent service centre).

In this model, we assume that packets are exactly one slot long (i.e. packet length = 1 slot).

If the number of packets inside the channel is only 1, the channel serves the 1 packet at a rate of 16 slot/frame (1 x 16 slot/frame = 16 slot/frame).

If the number of packets inside the channel is 2, the channel serves the 2 packets at a rate of 8 slot/frame each (2 x 8 slot/frame = 16 slot/frame).

...

If the number of packets inside the channel is 16, the channel serves the 16 packets at a rate of 1 slot/frame each (16 x 1 slot/frame = 16 slot/frame).

As the number of packets inside the channel cannot be greater than 16, we shall define the channel as a finite capacity (16 slots) resource with finite queuing capacity.

In case the system is better modelled by assuming packet lengths of 2/4/8/16 slots, the channel shall be modelled as reported in appendix.

- number of servers

Under the previous assumptions, the channel (16 slot wide) can be modelled as a single server.

- queue capacity

As the request come from an undetermined number of RFDs, we will assume the queue capacity infinite.

To sum up:

- inter-arrival times distribution: Poisson
- service times distribution: Poisson
- number of servers: 1
- queue capacity: infinite

5.3 microcontroller unit

Unfortunately, to the best of our knowledge, the literature does not provide an explicit queuing network model for the FFD microcontroller unit. For the sake of simplicity, we will make the following assumptions.

- inter-arrival times

We will assume that the packet inter-arrival times distribution is Poisson.

- service times

We will assume that the packet service time distribution is Poisson. The *mean value* will increase with increasing *encryption level*.

<i>encryption level</i>	<i>mean service time</i>
No encryption	0.3 ms
...	...
Maximum simulated encryption	2.5 ms

- number of servers

We will assume a single microcontroller unit.

- queue capacity

The queue capacity is implementation dependent, but we will assume this capacity equal to 7 (the maximum number of addresses contained in a beacon frame).

To sum up:

- inter-arrival times distribution: Poisson
- service times distribution: Poisson
- number of servers 1
- queue capacity: 7

6 simulation results

Now that we have defined which models will be used for simulating the 802.15.4 protocol, the next step is to integrate them under the JMT - JAVA MODELLING TOOL v.0.7.0 and to set parameters in order to simulate this wireless network under the conditions explained in the previous chapter.

As described in the previous chapter, we have chosen three topologies with two models each ('load dependent' and 'load independent') for simulating the 802.15.4 protocol. They are summarized below:

- uplink communication when a RFD wants to send a message to the FFD:
 - Load dependent mode
 - Load independent mode
- downlink communication when a FFD wants to send a message to a RFD:
 - Load dependent mode
 - Load independent mode
- joint uplink and downlink communication when both FFD and RFD want to exchange messages:
 - Load dependent mode
 - Load independent mode

6.1 M/M/n model, uplink mode

The first simulation represents a reduced functional device sending packets to the full functional device. Data have to be sent through the channel and then processed by the MCU unit. The following scheme is used for simulating the uplink mode:

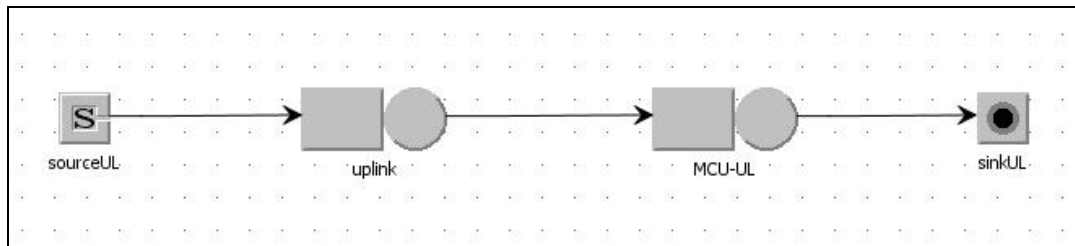


Figure 12: Representation of the Uplink model.

Assuming the fact that a the channel is the bottleneck of the system, we try to figure out what is the maximum encryption processing time possible before slowing down the entire system.

The following parameters are used for the simulations:

- Source distribution: Poisson with mean of 0.9 ms
- uplink service time distribution, load independent:
 - 16 servers
 - Poisson distribution with mean of 0.0625ms ($16 \cdot 0.0625 = 1\text{ms}$)
- uplink service time distribution, load dependent:
 - 1 server
 - Poisson distribution with mean values of Chapter 6.1.2
- uplink queue: finite of 16 packets with waiting queue (no drop) rule
- service time MCU distribution:
 - 1 server
 - Poisson distribution with variable mean from 0.3ms (without encryption) to 2.5ms (full encryption)
- MCU queue: finite of 7 packets with drop rule

Uplink channel simulations, Load Independent and Load Dependent models:

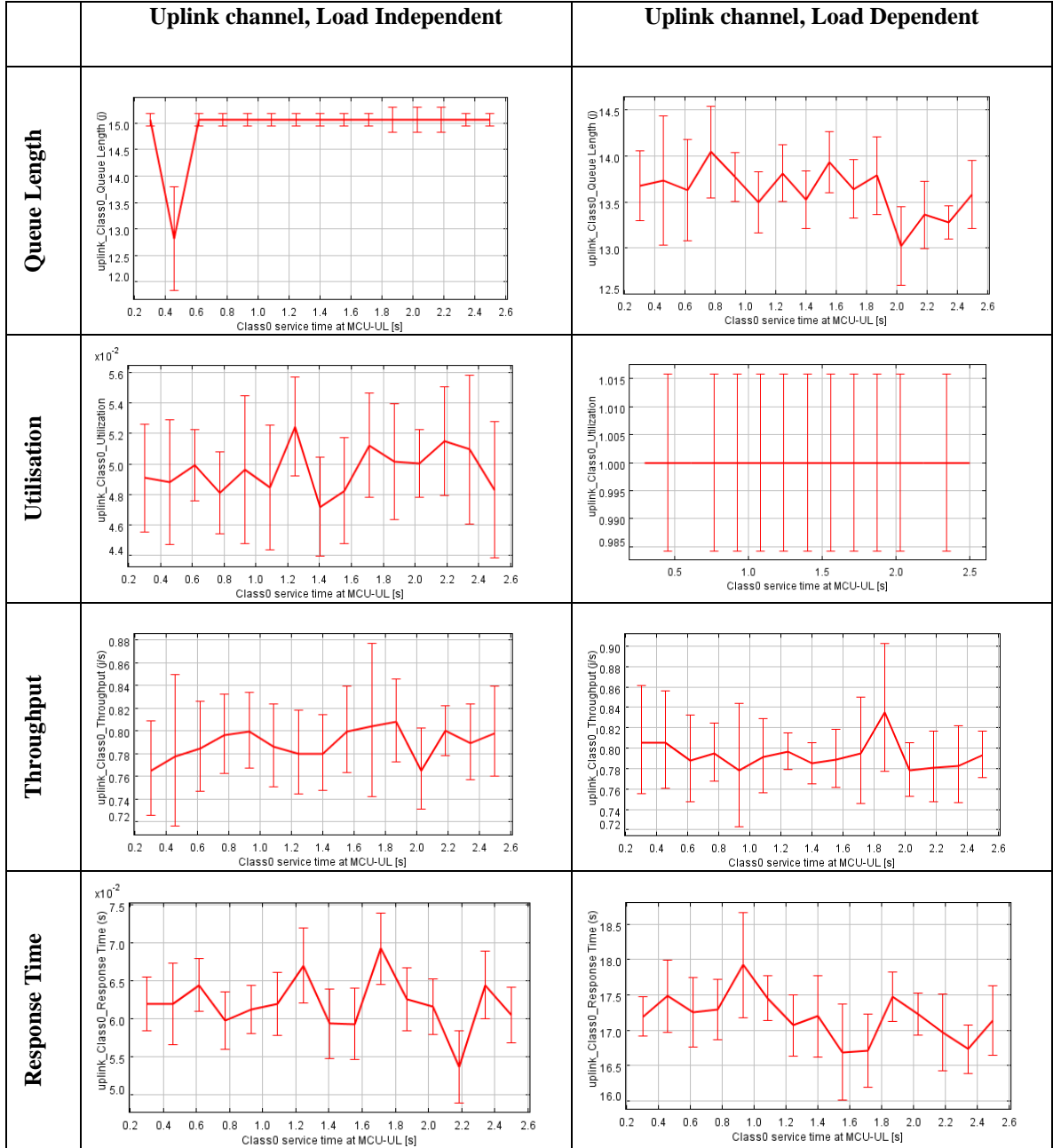


Figure 13: Uplink channel simulations.

Uplink MCU simulations, Load Independent and Load Dependent models:

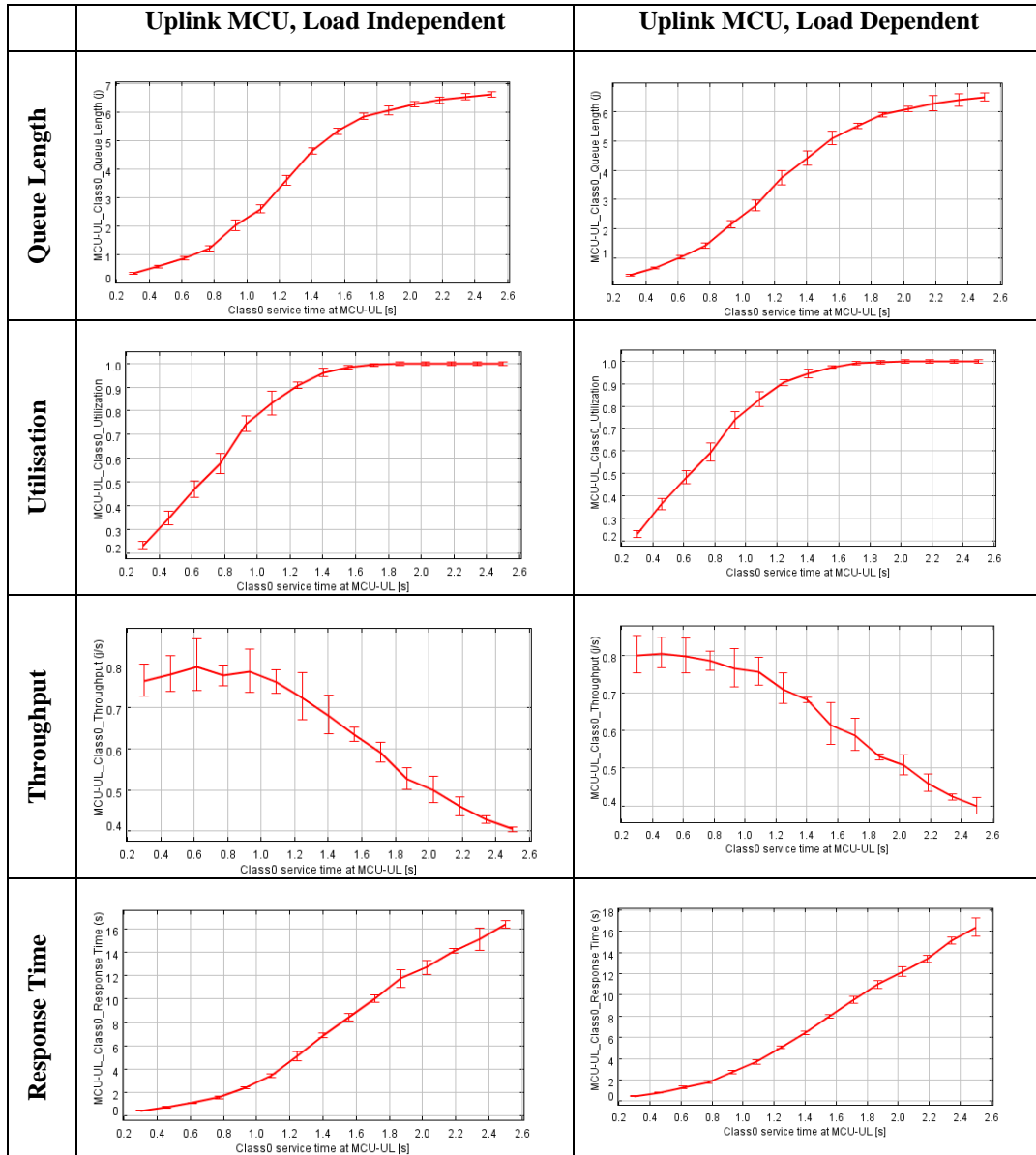


Figure 14: Uplink MCU simulations.

Observations:

- **Uplink channel**
 The first observation to make is that the arrival distribution is a bit higher than the channel service time, which explain the fact that the channel utilisation is 100% for the load dependent mode and near 100%/16 for the independent mode which possesses 16 servers. Due to the higher arrival distribution time, the Queue Length is also logically nearly or totally full depending on the mode used.
 In this case, the channel is not related to the MCU and his behaviour remains the same even with a changing processing time of the MCU.
- **Uplink MCU**
 Concerning the MCU, we try to determine what is the maximum encryption level that it is possible to use before reaching an MCU utilisation of 100%. We clearly see that when the MCU has a distribution higher than 1ms, the throughput starts to decrease, the response time increase and the utilisation is closed to saturation. The system starts to have lower performances.

6.2 M/M/n model, downlink mode

The second simulation represents the full functional device sending packets to a reduced functional device. Data are firstly processed by the MCU and then sent to the RFD via the channel. The following scheme is used for simulating the downlink mode:

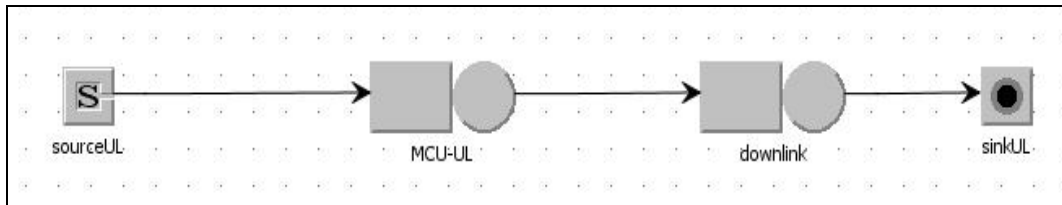


Figure 15: Representation of the Downlink model.

As already explained in the previous paragraph, we try also to figure out what is the maximum encryption processing time possible before slowing down the entire system and if this result is the same than the one determined during an uplink communication.

The following parameters are used for this simulation:

- Source distribution: Poisson with mean of 0.9 ms
- MCU service time distribution:
 - 1 server
 - Poisson distribution with variable mean from 0.3ms (without encryption) to 2.5ms (full encryption)
- MCU queue: finite of 7 packets with drop rule
- uplink service time distribution, load independent:
 - 16 servers
 - Poisson distribution with mean of 0.0625ms ($16 \times 0.0625 = 1\text{ms}$)
- uplink service time distribution, load dependent:
 - 1 servers
 - Poisson distribution with mean values of paragraph 6.1.2
- uplink queue: finite of 16 packets with waiting queue (no drop) rule

Downlink MCU simulations, Load Independent and Load Dependent models:

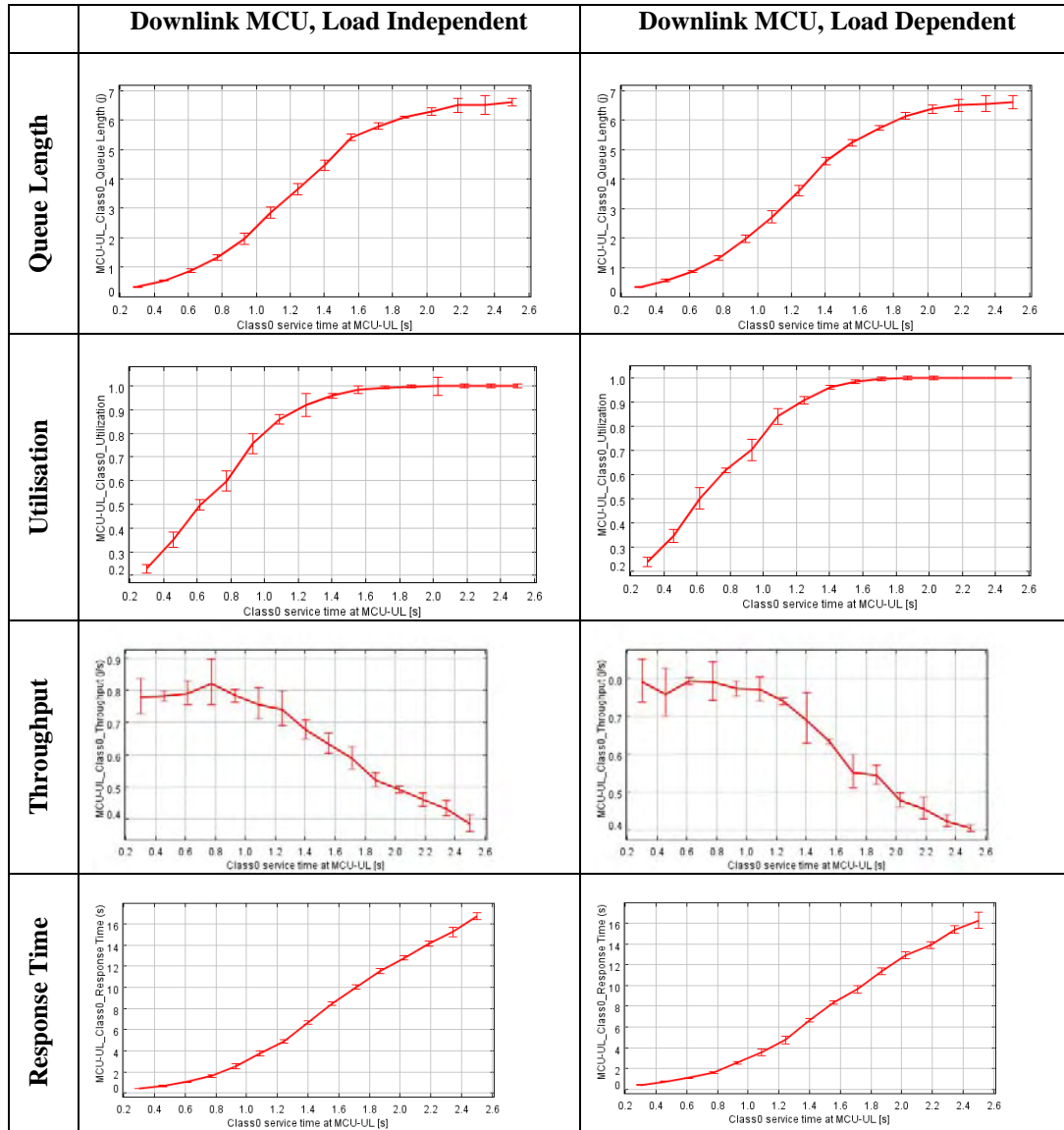


Figure 16: Downlink MCU simulations.

Downlink channel simulations, Load Independent and Load Dependent models:

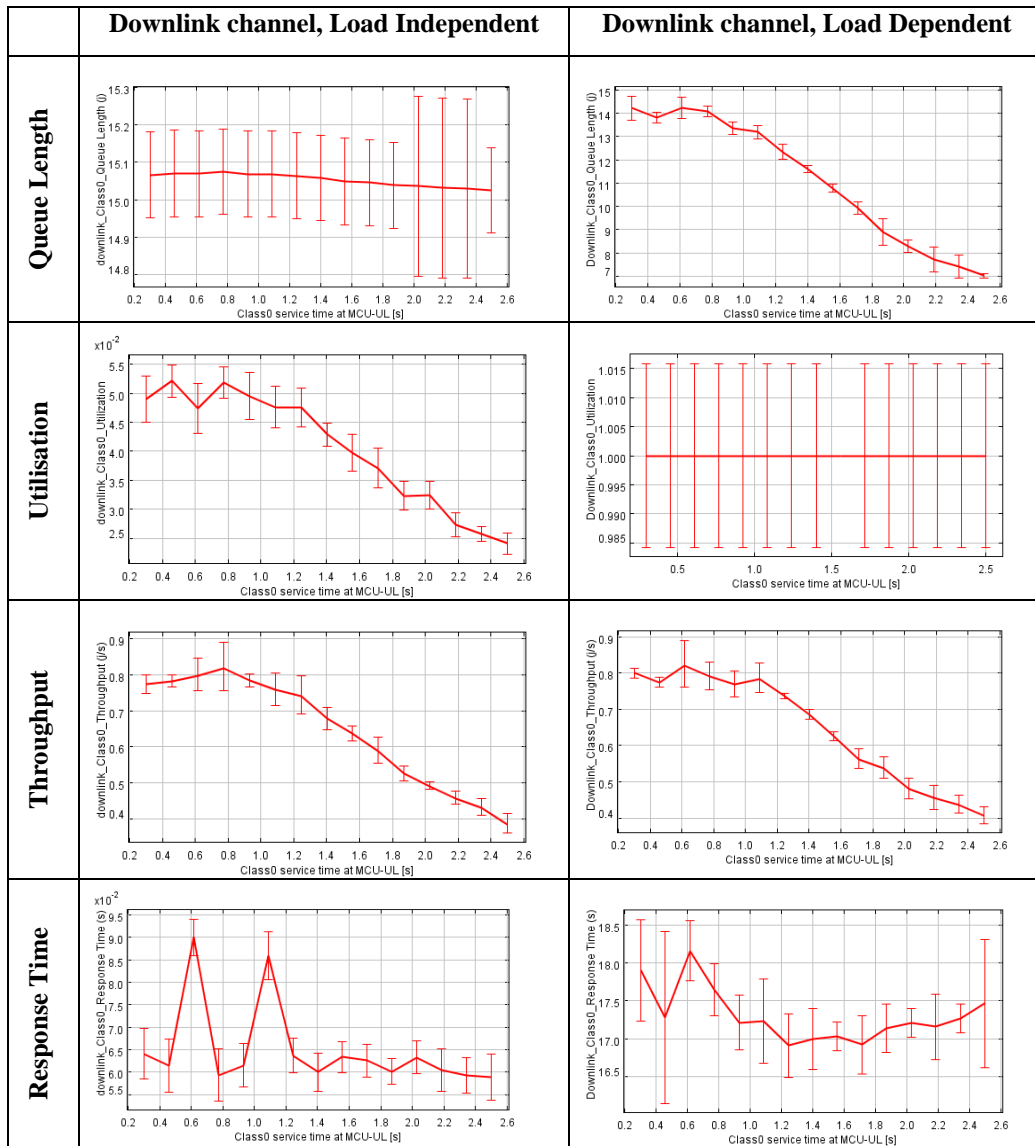


Figure 17: Downlink channel simulations.

Observations:

- **Downlink channel**
 In this case, the MCU is not related to the channel and will depend directly on the distribution arrival rate. The behaviour of the MCU is logically the same than for the uplink mode and the system performance starts to decrease as soon as the processing time is slower than the arrival rate.
- **Downlink MCU**
 When the MCU processing time is quicker than the speed of the channel, the latter remains the bottleneck of the system and overall performances remain constant. Once the mean value of the MCU service time distribution reaches 1ms, then performances are going down and the maximum encryption level accepted is reached.

6.3 M/M/n model, joint uplink and downlink mode

Now that the Uplink and Downlink modes have been simulated separately, we can try to integrate these two models in one in order to simulate a more realistic case, when packets are exchanged between FFD and RFD. The following scheme will be used for simulations:

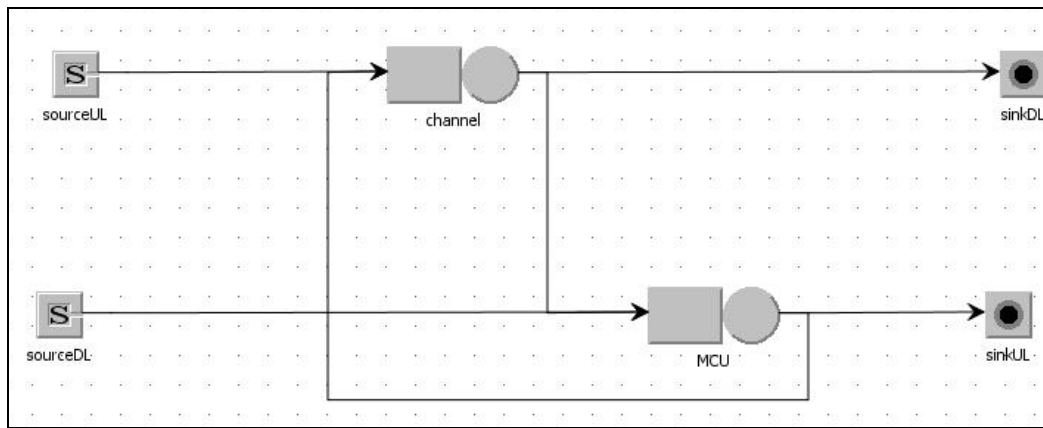


Figure 18: Representation of the joint uplink and downlink model.

In this model, packets leave from the uplink source and reach the uplink sink passing through the MCU followed by the channel and vice versa, packets leave from the downlink source and reach the downlink sink passing through the channel followed by the MCU. Routing probabilities have been introduced accordingly.

As we are trying to simulate the up and down communication at the same time, each arrival time distribution has to be modified in order to be two times slower than in the previous configurations. Concerning the channel, it is configured with the same constant service time as before. As we still try to figure out what is the maximum encryption level allowed, the MCU service time will remain the variable parameter.

The following parameters are used for this simulation:

- uplink source distribution: Poisson distribution with mean of 1.8 ms
- downlink source distribution: Poisson distribution with mean of 1.8 ms
- MCU service time distribution:
 - 1 server
 - Poisson distribution with variable mean from 0.3ms (without encryption) to 2.5ms (full encryption)
- MCU queue: finite of 7 packets with drop rule
- Channel service time distribution, load independent:
 - 16 servers
 - Poisson distribution with mean of 0.0625ms ($16 \cdot 0.0625 = 1\text{ms}$)
- Channel service time distribution, load dependent:
 - 1 servers
 - Poisson distribution with mean values of paragraphs 6.1.2
- Channel queue: finite of 16 packets with waiting queue (no drop) rule

Up-Downlink MCU and channel simulations, Load Independent model:

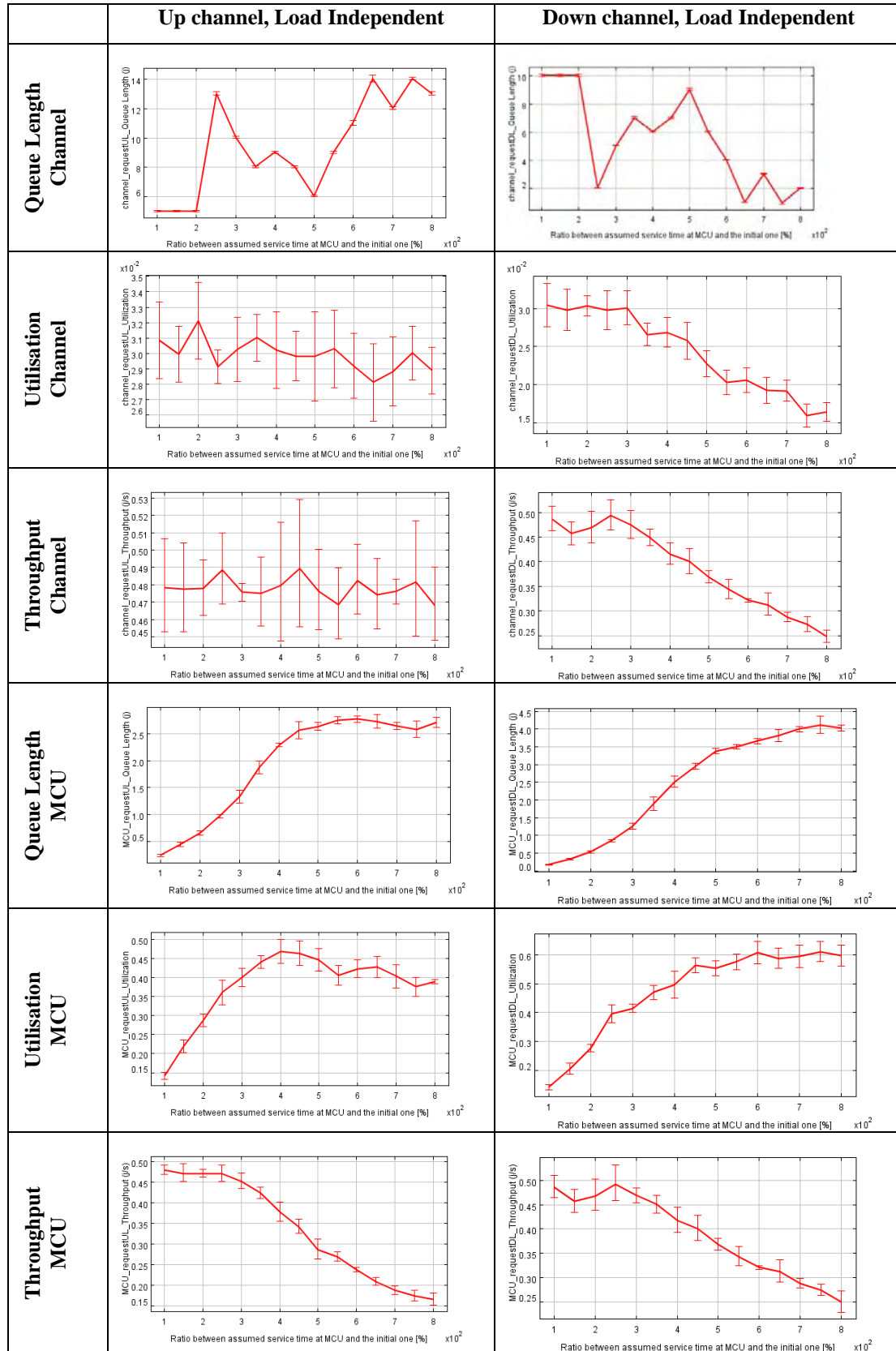


Figure 19: joint uplink and downlink simulations, Load Independent model

Up-Downlink MCU and channel simulations, Load Dependent model:

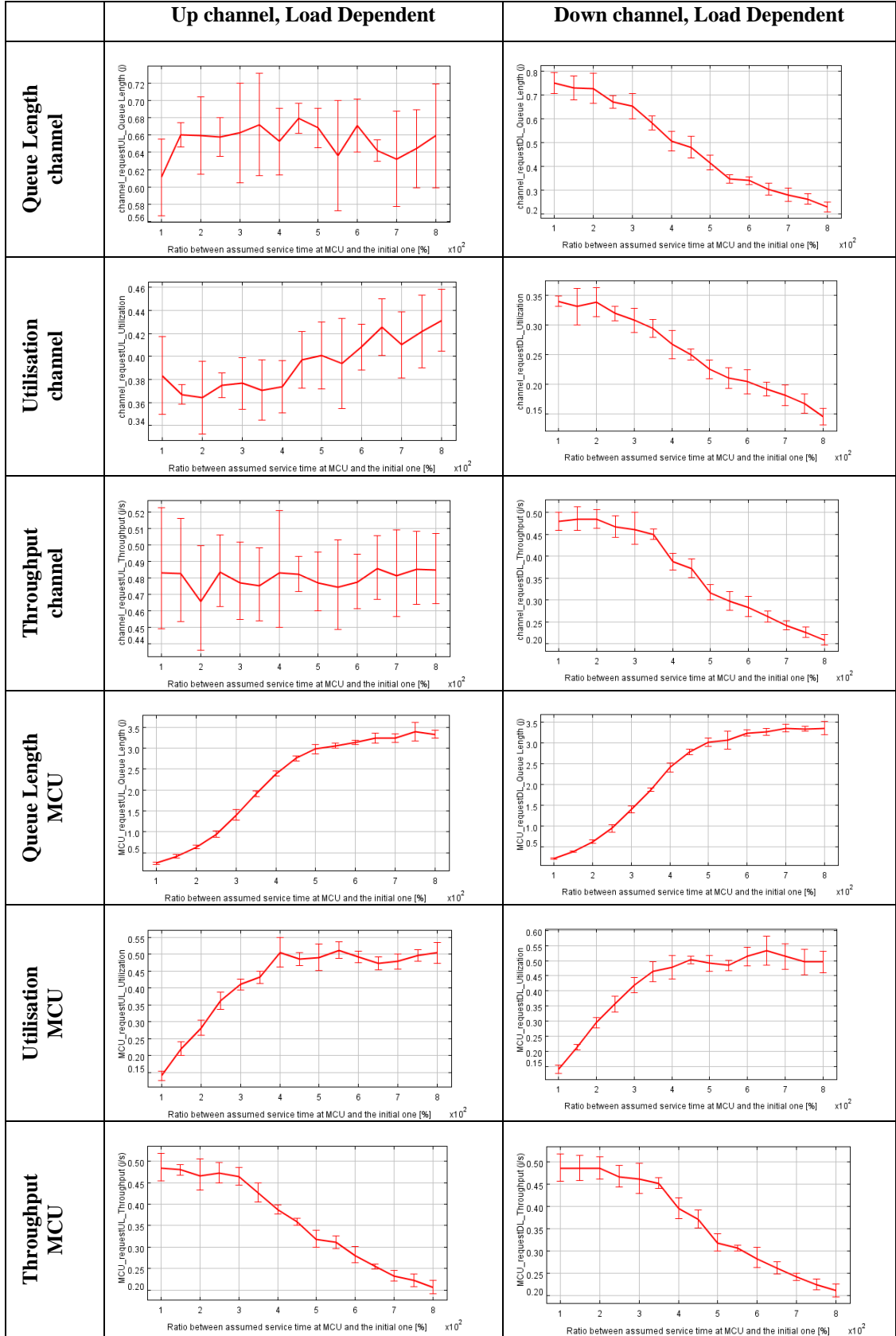


Figure 20: Up-Downlink simulations, Load Dependent model.

Observations:

- Up-downlink channel, Load Independent
Considering figure 15, we analyze these results in order to find similarities with the uplink or downlink model. Attention should be given at the x line which represents the ratio in % regarding to the 0.3ms original. When analyzing the channel during uplink mode, we see that the encryption level does not influence it; during the downlink communication, the latter is affected and starts to decrease as soon as the MCU become the bottleneck of the system. Regarding the MCU, its throughput starts to decrease after 1 ms as expected and the addition of both up and down utilisation tends to 100%. These results corresponds to those obtained in the single up and down mode.
- Up-downlink channel, Load Dependent
The results obtained are in general way very similar to those obtained with the Load Independent mode, which corresponds perfectly to our expectation. The same comments can be made regarding the different parameters of the system.

7 conclusions

The goal of this project was to study the low power wireless protocol IEEE 802.15.4, determine a model which represents a real case application of a sensor network, and finally determine the maximum level of encryption supported by a microcontroller before starting to slow down the speed of the entire system.

The first step has been to study the theory of this protocol, mainly by exploring the literature available today and by understanding the different models that have been modelled and simulated. Then, three models representing the uplink communication mode, the downlink communication mode and finally the joint uplink and downlink mode came out and were accepted as a representation of our system. According to some research under investigation, some assumptions have been made in order to represent a realistic case with consistent values. Finally, The JMT – Java modelling tool has been used for performing the simulations of this low power protocol.

Then, the last step of this project has been to run several simulations. As mentioned above, the Java Modelling Tool has been used for realizing our simulations. It appeared that some bugs still exist in the simulator and plenty of time was spent in order to understand and fix them. But eventually, after setting up the correct parameters, we obtained some results consistent with our predictions.

Behavioural differences between the uplink and downlink models have been reported. They are mainly due to

- deep differences in the queuing policies
the MCU has been modelled with a very limited queuing capacity whereas the RFD cloud has been modelled with an unlimited capacity, as the distributed individual queuing capacity is normally able to sustain the individual distributed communication needs
- the assumption that the encryption management is mainly impacting on the MCU station performance and not on the channel station performance (see "further work").

Concerning simulations, we can make the following conclusions. The results obtained correspond well with the theoretical predictions and the behaviour of the two systems (load dependent and load independent) gave the same results as expected.

It is also possible to determine the maximum level of encryption, which corresponds to three times the execution time of a packet without encryption.

8 further work

In this work, several interdependencies have been considered and analysed.

The most important interdependency not considered in this work is the dependency of the packet duration on the encryption level.

Furthermore, IEEE Std. 802.15.4 could have been investigated more accurately in case the JMT simulator could support the Probabilities Generating Functions for the service times reported in the "literature exploration" paragraph.

9 acronyms and abbreviations

CSMA/CA	carrier sense multiple access with collision avoidance
CSMA/CD	carrier sense multiple access with collision detection
DSSS	direct sequence spread spectrum
FFD	full-function device
MAC	medium access control sublayer specifications
PGF	probability generating function
PHY	physical sublayer specifications
POS	personal operating space
RFD	reduced-function device

10 references

- [1] Lazowska et al.,
Quantitative System Performance - Computer System Analysis Using Queuing Network Models
Prentice-Hall, 1984
- [2] IEEE Std 802.15.4-2003
IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements— Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)
- [3] Park TR, Kim T, Choi J, Choi S, and Kwon W
Throughput and energy consumption analysis of IEEE 802.15.4 slotted CSMA/CA
IEEE Electronics Letters, vol. 41, issue 18, pp. 1017-1019, Sept. 2005.
- [4] Pollin S, et al.
Performance analysis of slotted IEEE 802.15.4 medium access layer.
Technical Report, DAWN Project, Sep. 2005
- [5] Jin-Shyan Lee
An experiment on performance study of IEEE 802.15.4 wireless networks
2005/09/19
- [6] Jelena Mišić, Shairmina Shafi, and Vojislav B. Mišić
Performance of a Beacon Enabled IEEE 802.15.4 Cluster with Downlink and Uplink Traffic
IEEE transactions on parallel and distributed systems, vol. 17, no. 4, April 2006
- [7] Tony Sun et al.
Measuring effective capacity of IEEE 802.15.4 beaconless mode
2006/04/03
- [8] Koubâa A, Alves M, Tovar E, and Song YQ
On the performance limits of slotted CSMA/CA in IEEE 802.15.4 for broadcast transmissions in wireless sensor networks.
Hurray Technical Report, 06 April 2006
- [9] Fabio FABBRI
Power modelling: procedures, techniques and proposals for wireless sensor networks
ALaRI.ch, Master of Science in Embedded Systems Design, July 2006

11 appendix

The following table represents the load dependency of a 16-slotted CSMA/CA channel efficiency for different packet lengths.

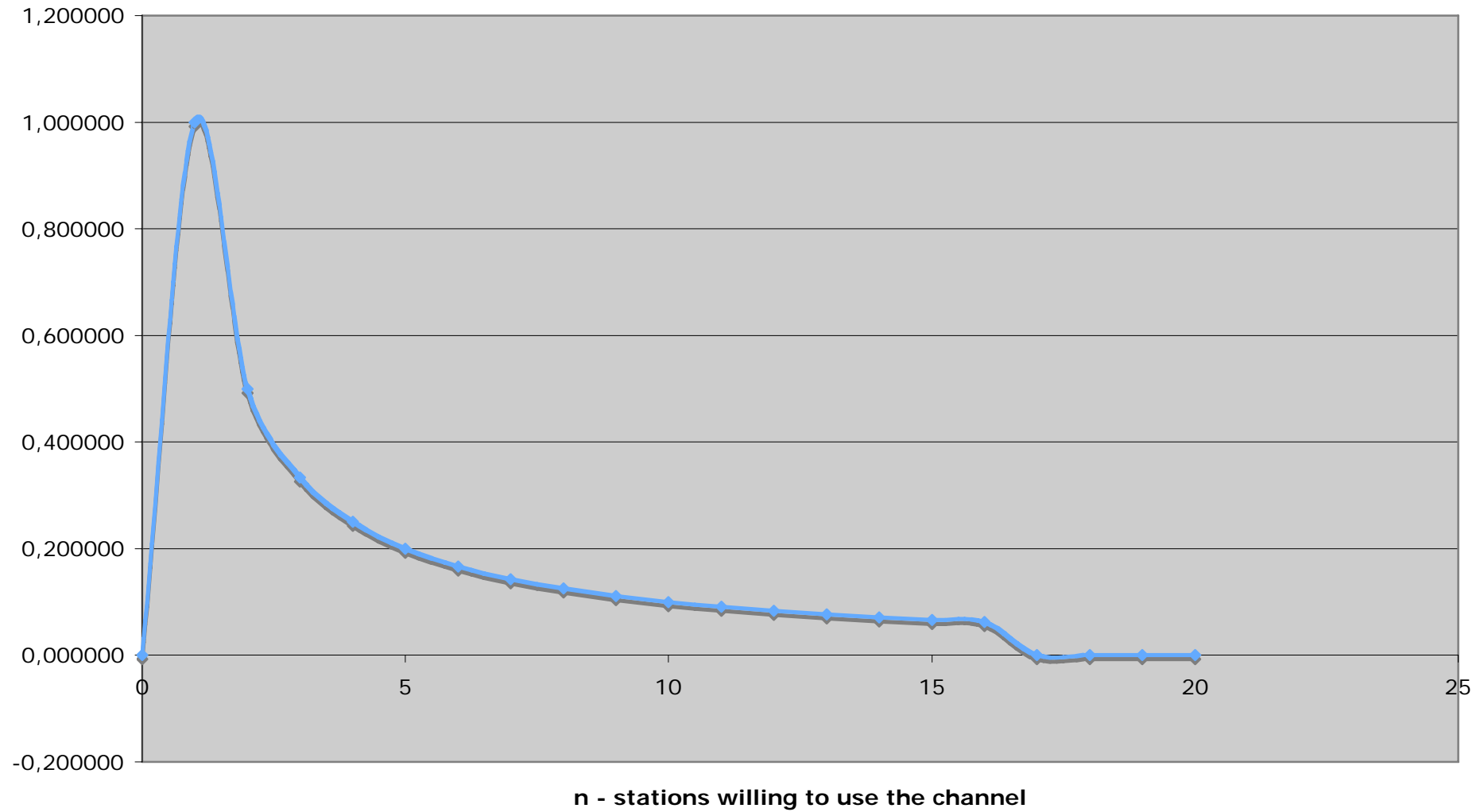
packet length jobs	service rates (slots/frame)				
	1	2	4	8	16
1	16/1=16	8/1=8	4/1=4	2/1=2	1/1=1
2	16/2=8	8/2=4	4/2=2	2/2=1	
3	16/3	8/3	4/3		
4	16/4=4	8/4=2	4/4=1		
5	16/5	8/5			
6	16/6	8/6			
7	16/7	8/7			
8	16/8=2	8/8=1			
...	...				
15	16/15				
16	16/16=1				

The following table and chart represent the load dependency of a 16-slotted CSMA/CA channel efficiency for packet lengths = 1.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
E(n)	0,000	1,000	0,500	0,333	0,250	0,200	0,167	0,143	0,125	0,111	0,100	0,091	0,083	0,077	0,071	0,067	0,063	0	0	0	0
B(n)	0,000	16,000	8,000	5,333	4,000	3,200	2,667	2,286	2,000	1,778	1,600	1,455	1,333	1,231	1,143	1,067	1,000	0	0	0	0

n: stations willing to use the channel (frame by frame)
 E(n): load dependent channel efficiency (frame by frame)
 B(n): load dependent channel capacity (slots/frame)

16-slotted CSMA/CA load dependent channel efficiency



A Mean-value Modeling of the Performance of SCADA Field Devices

Doctoral course on Advanced Topics of Performance Evaluation

Tutor: Prof. Giuseppe Serazzi

Submission date: April 2007

PhD Student: Julian L. Rrushi

University ID: R06567

Institution: Università degli Studi di Milano

1. Motivations

Embedded operating systems and communication protocols devised and implemented for use in SCADA were not designed with security in mind. Employment of proprietary technology and some kind of relative isolation of SCADA networks from Internet have been the motivations behind design paths oriented exclusively towards functionality rather than functionality and security. Nevertheless, with the advent of high connectivity of SCADA networks to the enterprise network and some times even to Internet, and a drastic switch to standard open protocols and operating systems, security has become one of the most critical issues in the control of critical infrastructures. Designing and implementing security mechanisms for SCADA field devices, i.e. remote terminal units, programmable logic controllers, or intelligent electronic devices, is quite a challenge due to their limited memory and computational power. Thus, a given host-based defensive approach for a SCADA field device should be sound, i.e. be efficient in protecting the device from cyber attacks, and it should be affordable by the device where that approach is to be deployed.

Determining the feasibility of a given defensive approach for a SCADA field device requires not only a careful evaluation of the performance overhead induced by such a defensive capability on that specific device, but also an estimation of the System Response Time of that device in a specific configuration of a SCADA environment where that device is meant to operate. Such a consideration is quite important since SCADA communications are real-time, and in some cases hard real-time. This means that when a master station sends a request to a field device, there exists a maximum value of delay within which the response should arrive from the field device to the master station. If the majority of replies does not respect such a time off value, a critical infrastructure cannot be controlled and monitored correctly, which in other words means there is a real danger for explosions and similar disasters. In support to the statement given above we can take as an example a field device running a cryptographic algorithm. In a SCADA environment where a master station communicates directly with a field device that device may be able to authenticate a request that it receives from the master station, take action, build and sign a reply, and send it to the master station before the timer at the master station goes off.

The same field device running the same cryptographic algorithm as in the previous example, in a SCADA environment where the master station communicates with substations which in turn communicate with field devices through one or more gateways, may not be able to respect the expiration time set up at the master station. This is the point where performance modeling appears as a viable solution for the estimation of the feasibility of a given implementation of a defensive approach in a determined SCADA field device deployed in a determined configuration of a SCADA environment. Actually little has been done to model the performance of SCADA field devices equipped with defensive capabilities. Isolated examples include the work by Lambert presented in [1]. Lambert describes the results of several empirical evaluations of the performance of an ARM-based field device running cryptographic algorithms one at a time. From the measurements performed by Lambert results that the device tested along with the respective cryptographic algorithms is usable in certain critical infrastructure environments such as an electrical power generation plant, and not usable in other ones such as gas facilities.

Nevertheless, even in this case Lambert does not take into account all factors which directly or indirectly affect the system response time of a field device running a cryptographic algorithm. In fact he couldn't since what he did are empirical measurements on an isolated field device. It may be useful to consider a modeling tool such as Java modeling tool (JMT) for a more thorough estimation of the feasibility of a given defensive technique such as signing SCADA protocol frames in a field device. Currently most SCADA field devices are not protected by any defensive mechanism, therefore generally most of the focus is put on inventing them. Nevertheless, at the moment of transferring such defensive techniques from a prototype form to fully operational ones performance evaluation becomes crucial to their success. Examples include code mutation [3] and FireBuff [4] devised to counter memory corruption vulnerabilities, or the suite B cryptography specified by NSA to protect sensitive but unclassified communications such as SCADA ones. In industrial control systems more than in any other computerized environments security is tied to performance evaluation.

2. Modeling of the Performance of a Basic Industrial Control System

The purpose of this document is to provide a first demonstration of the usefulness of JMT in modeling the performance of field devices possibly equipped with defensive capabilities such as mitigation of memory corruptions and/or authentication of SCADA protocol frames. For the sake of simplicity a very basic configuration of a SCADA system was chosen for modeling. The work described in this paper is ongoing and aims at providing modeling of the performance of arbitrary field devices along with arbitrary configurations of SCADA environments. In this paper is described the use of JMVA to model the performance of a field device in an environment composed of a single master terminal unit (MTU) communicating directly with a programmable logic controller (PLC). The PLC in turn communicates directly with a sensor to gather monitoring data, and with an actuator to perform mechanical operations such as opening or closing valves. For more information on SCADA see [2]. The topology of our model is depicted in Figure 1.

Our model contains a single class which we refer to as PLC class. The PLC class is closed and its population consists of $N=1$ customer. Assuming that MTU communicates with field devices according to the MODBUS protocol [5, 6], in our model a customer is a MODBUS request message. A field device in a SCADA environment may receive jobs either from a MTU or from a substation if any. Other field devices may contact a given field device, but they will do so by passing through the MTU or a substation. At any point in time the number of devices which may contact a field device is well known in SCADA. There are 5 stations in our model, namely MTU/HMI, CPU, main memory, sensor, and actuator. MTU/HMI represents user's think time which varies from SCADA environment to SCADA environment. Furthermore, in most SCADA environments during certain time intervals a set of predefined commands are sent periodically by a program. That program is conceptually similar to Cron in Unix and its operation is controlled by a configuration file. In our model we set think time $Z=6$ s.

The four remaining stations are load independent. The types of the stations of our model are summarized in Table 1. The MTU/HMI and the PLC are located in the control network of SCADA, while the sensor and actuator are located in the fieldbus network. Thus, in our model the service demand for MTU/HMI is $D_k=6$ s. If our field device is running an implementation of ECDSA cryptographic algorithm, the service demand for the CPU is equal to the number of visits, i.e. the number of instructions executed for verifying the MODBUS request received from MTU and signing the MODBUS reply sent to MTU, times the service time of the CPU, i.e. the time it takes to the CPU to execute an instruction.

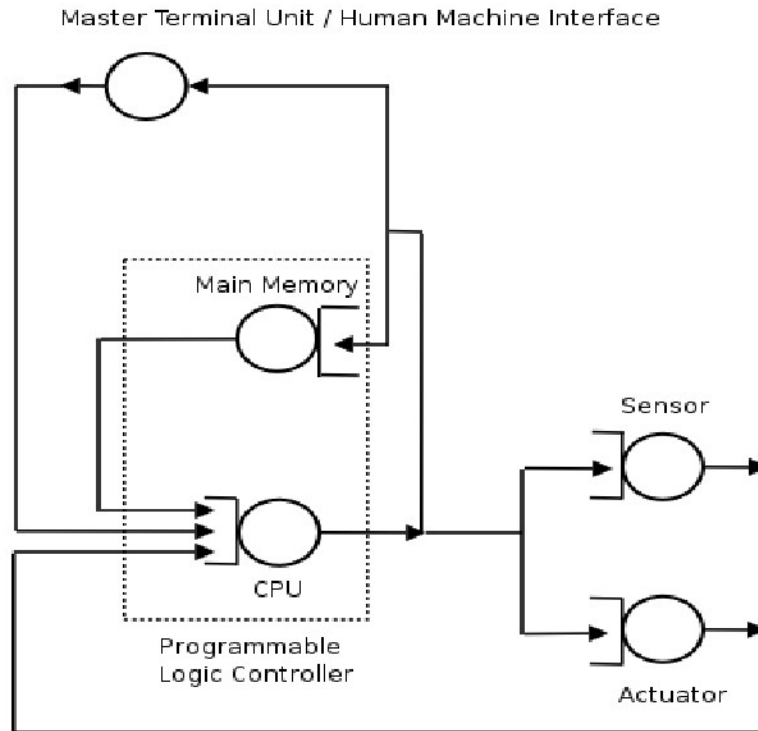


Figure 1: Network topology of our single closed class model

For a CPU whose clock cycle is 50 MHz we consider $D_k=0.035$ s. The service demand for the memory necessary to support verification of the MODBUS request received from MTU and signature of the MODBUS reply sent to MTU is $D_k=0.015$ s. Furthermore, the number of visits to the sensor and actuator is 1 and their service time is $S_k=0.001$ s, thus $D_k=0.001$ s. After running JMT under such a configuration we get the response time of the entire system $R=6.052$ s. The entire system response time in the case of a SCADA environment where the PLC runs a more powerful or less powerful processor may be determined by performing an What-if analysis. In such an analysis the control parameter would be a lower or higher service demand for the CPU, respectively. As a matter of fact the number of visits to CPU does not change. What changes is the service time S_k , i.e. the time required to execute an instruction. Similarly, the entire system response time in the case of a SCADA environment where the

PLC is equipped with more main memory and larger cache, or less main memory and little cache, could be determined by performing an What-if analysis where the control parameter would be a lower or higher service demand for the memory, respectively.

<i>Station</i>	<i>Type</i>	<i>Location in SCADA</i>
MTU/HMI	Delay	Control network
CPU	Load independent	Programmable logic controller
Main memory	Load independent	Programmable logic controller
Sensor	Load independent	Fieldbus network
Actuator	Load independent	Fieldbus network

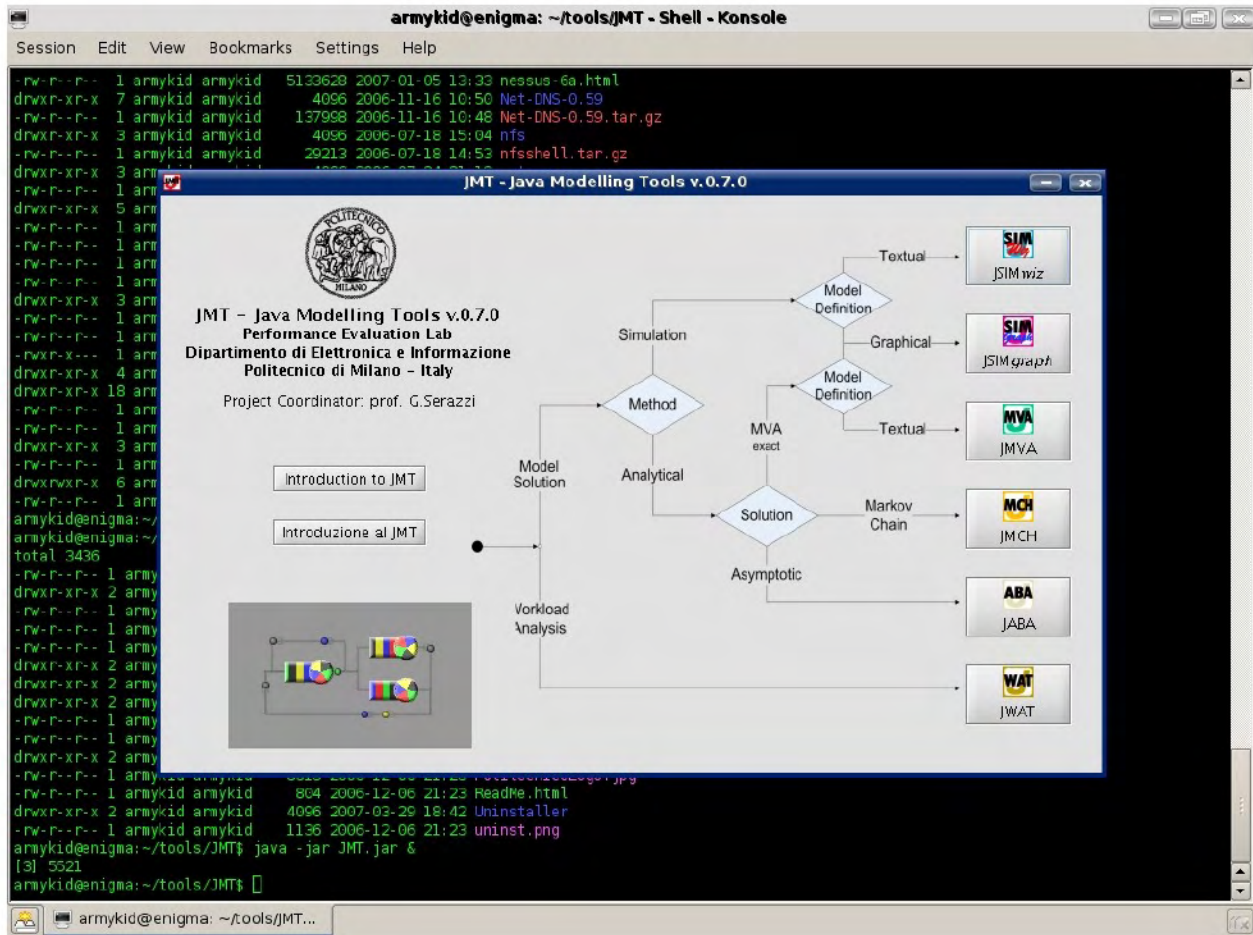
Table1: Stations of our model and their locations in SCADA

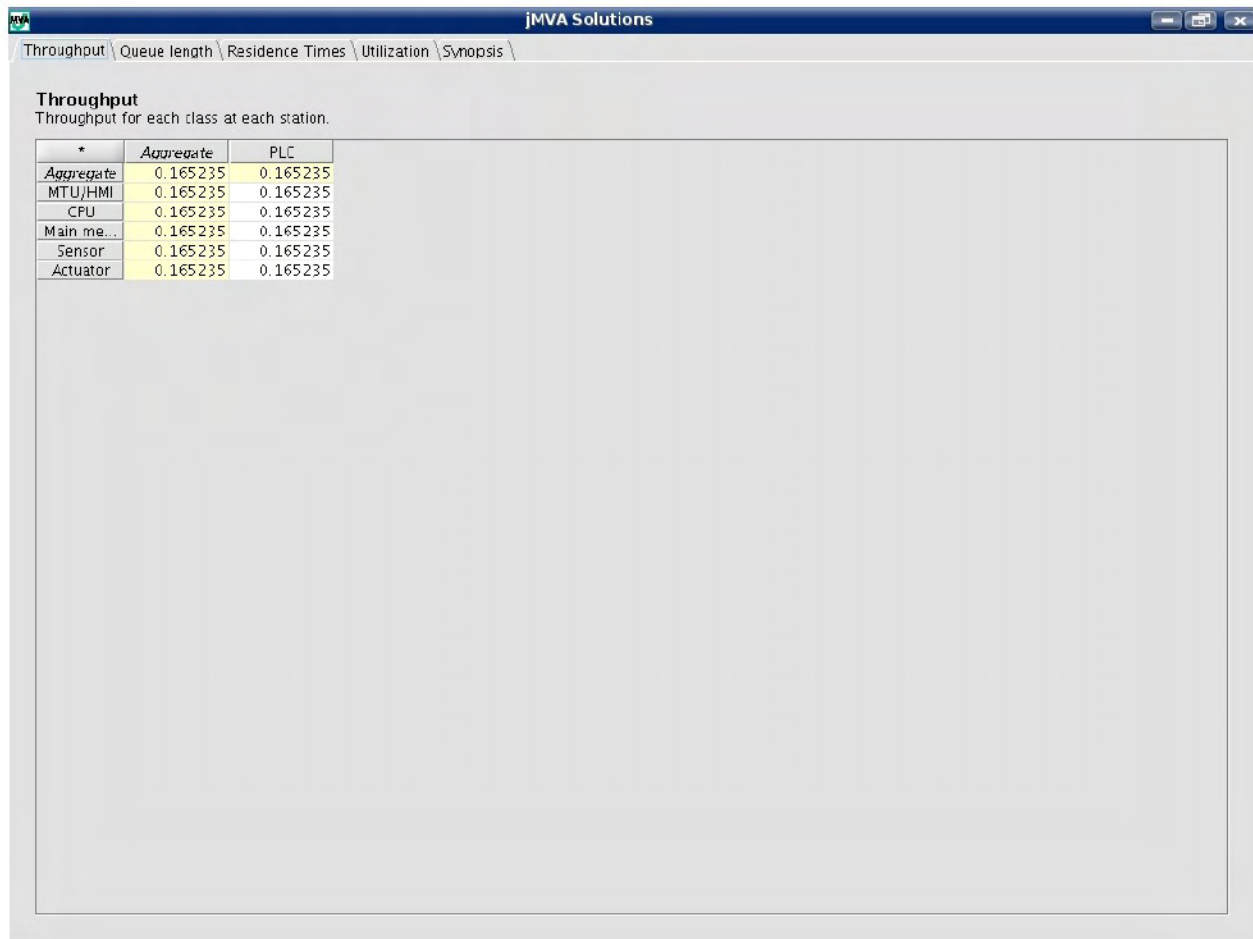
References

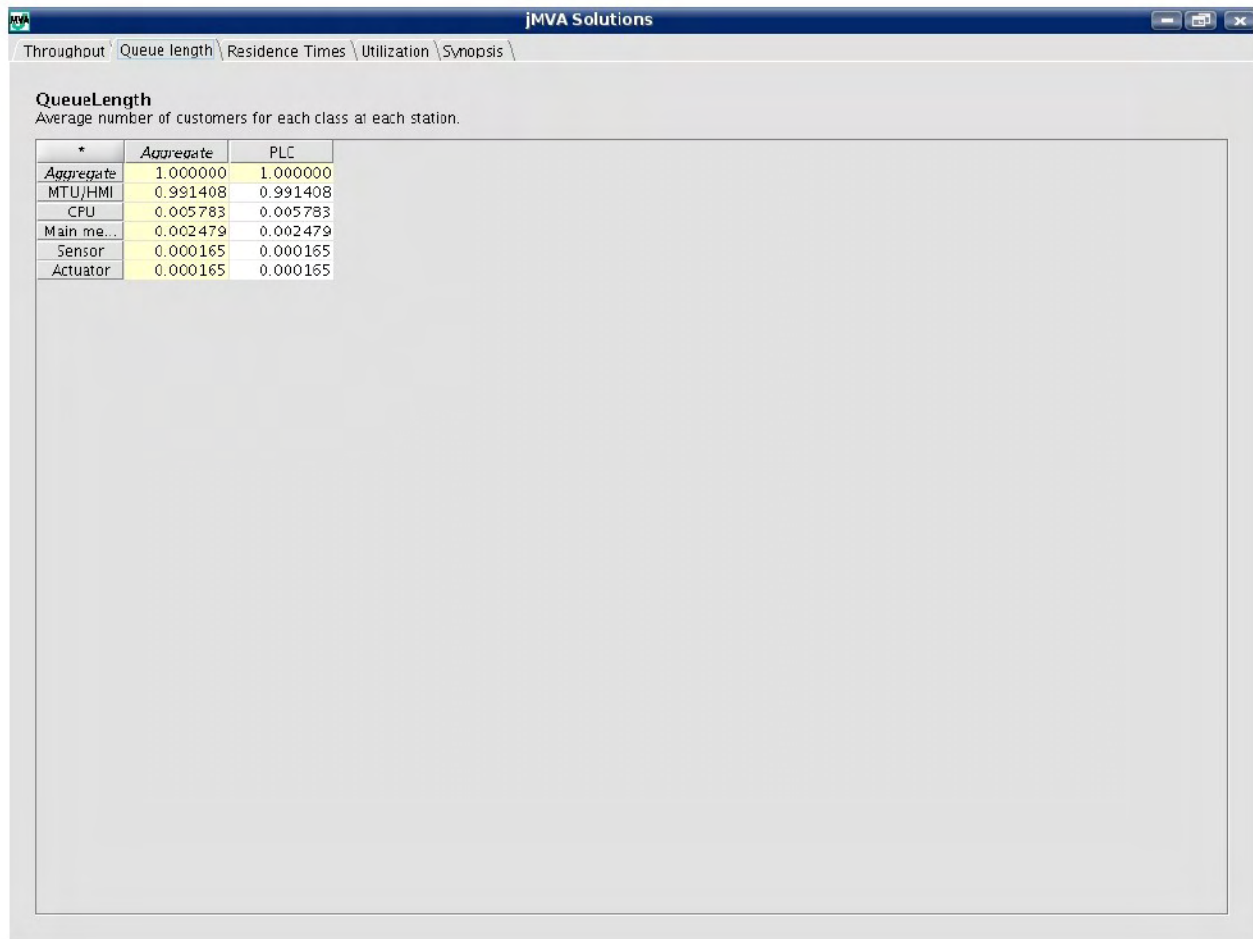
- [1] Lambert, R., “ECC and SCADA Key Management”, Presented in SCADA Security Scientific Symposium, Miami, USA, January 2007.
- [2] Stouffer, B.K., Falco, J., and Kent, J., “Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security”, National Institute of Standards and Technology, Special Publication 80082, September 2006.
- [3] Simmons, Sh., Edwards, D., and Wilde, N., “Securing Control Systems with MultiLayer Static Mutation”, presented in the 2007 meeting of the process control forum, USA, March 2007.
- [4] Bellettini, C., and Rrushi, J.L., “FireBuff: A Defensive Approach Against Control-data and Pure-data Attacks”, paper submitted for peer review.
- [5] MODBUS Organization, “MODBUS Application Protocol Specification V 1.1a.”, Retrieved December 1, 2006, from http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1a.pdf
- [6] Schneider Automation. “MODBUS Messaging on TCP/IP Implementation Guide V1.0a”, Retrieved December 1, 2006, from http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0a.pdf

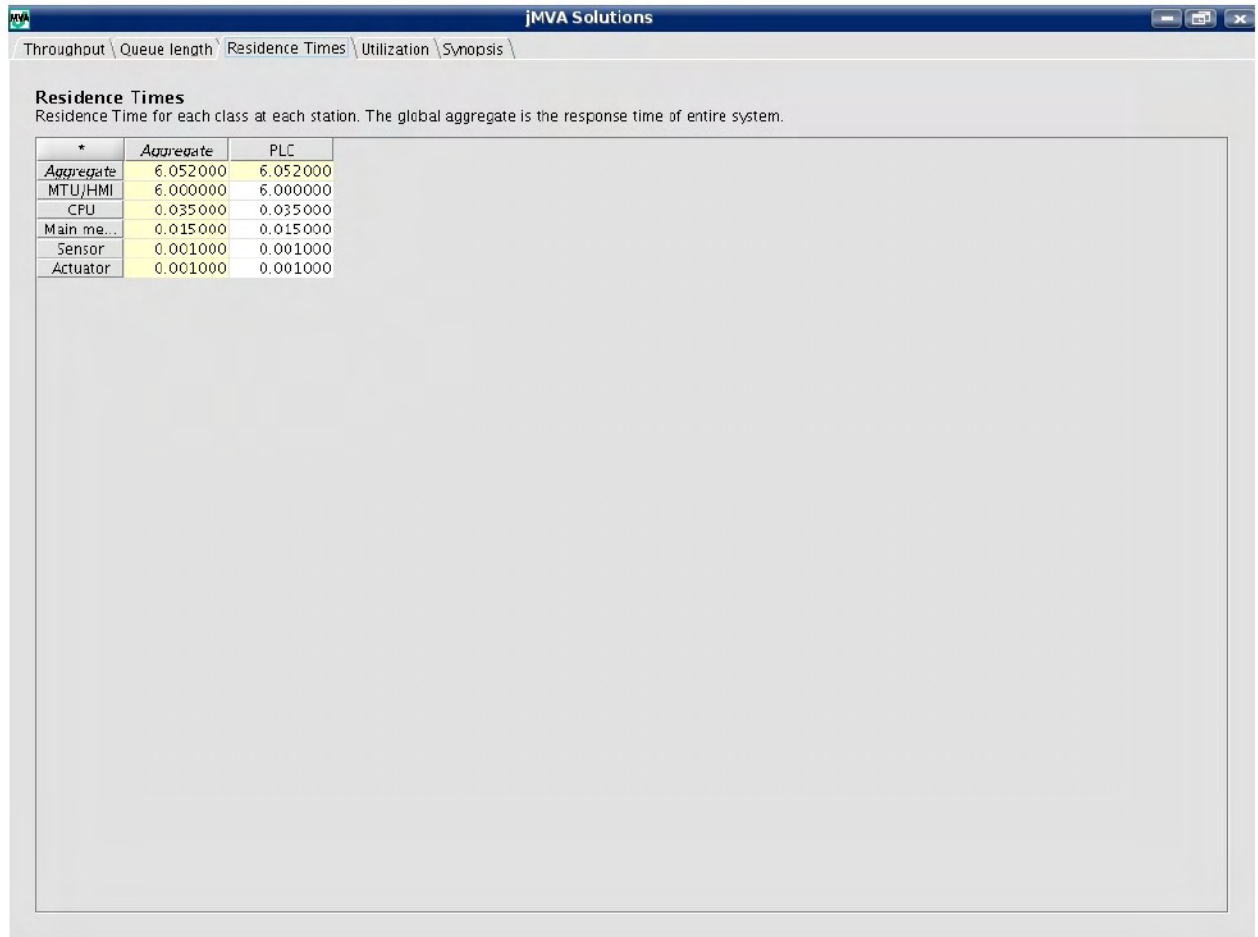
Appendix

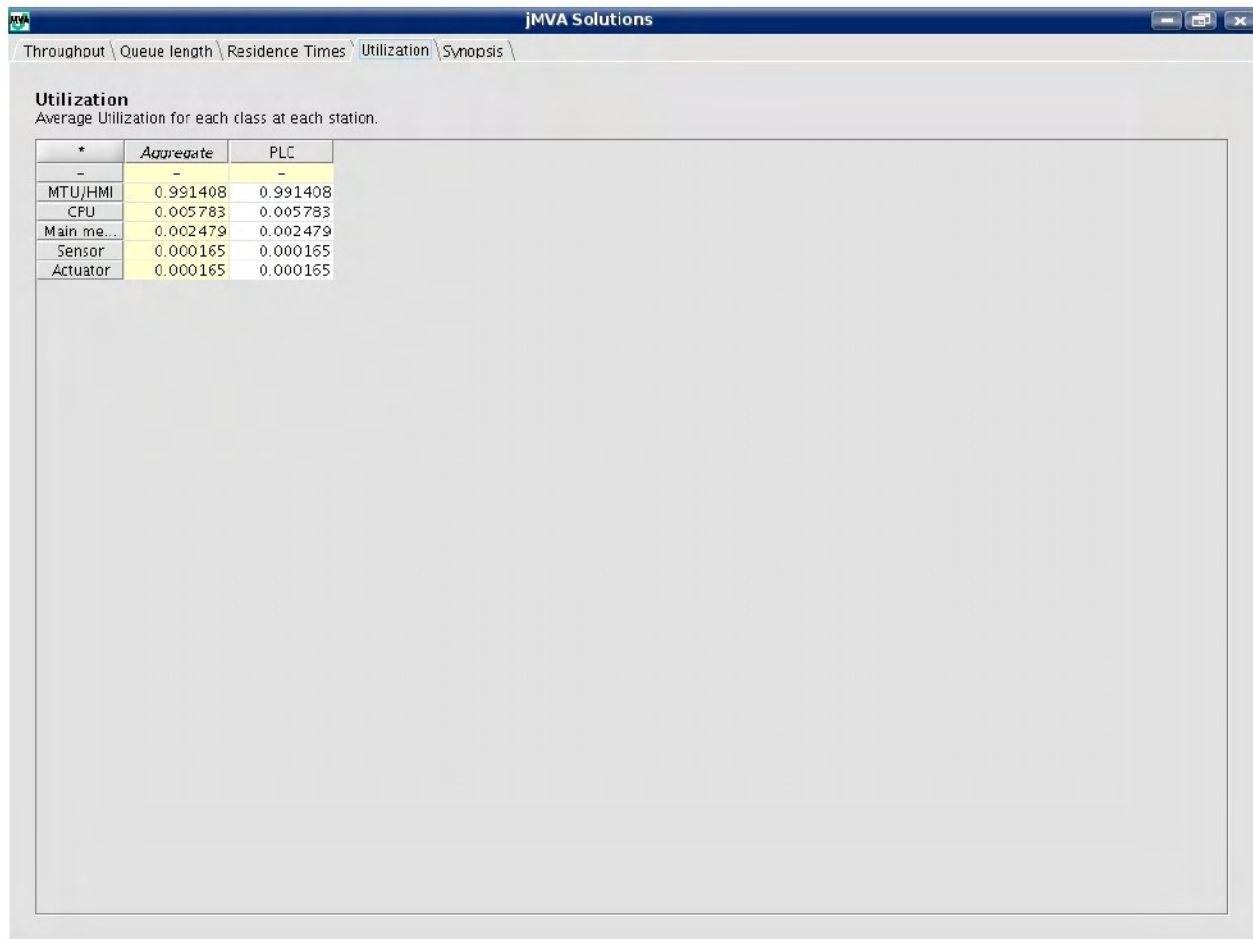
JMT was run under a Linux operating system. In what follows are given some screen shots which document how JMT was run to model a basic industrial control system.

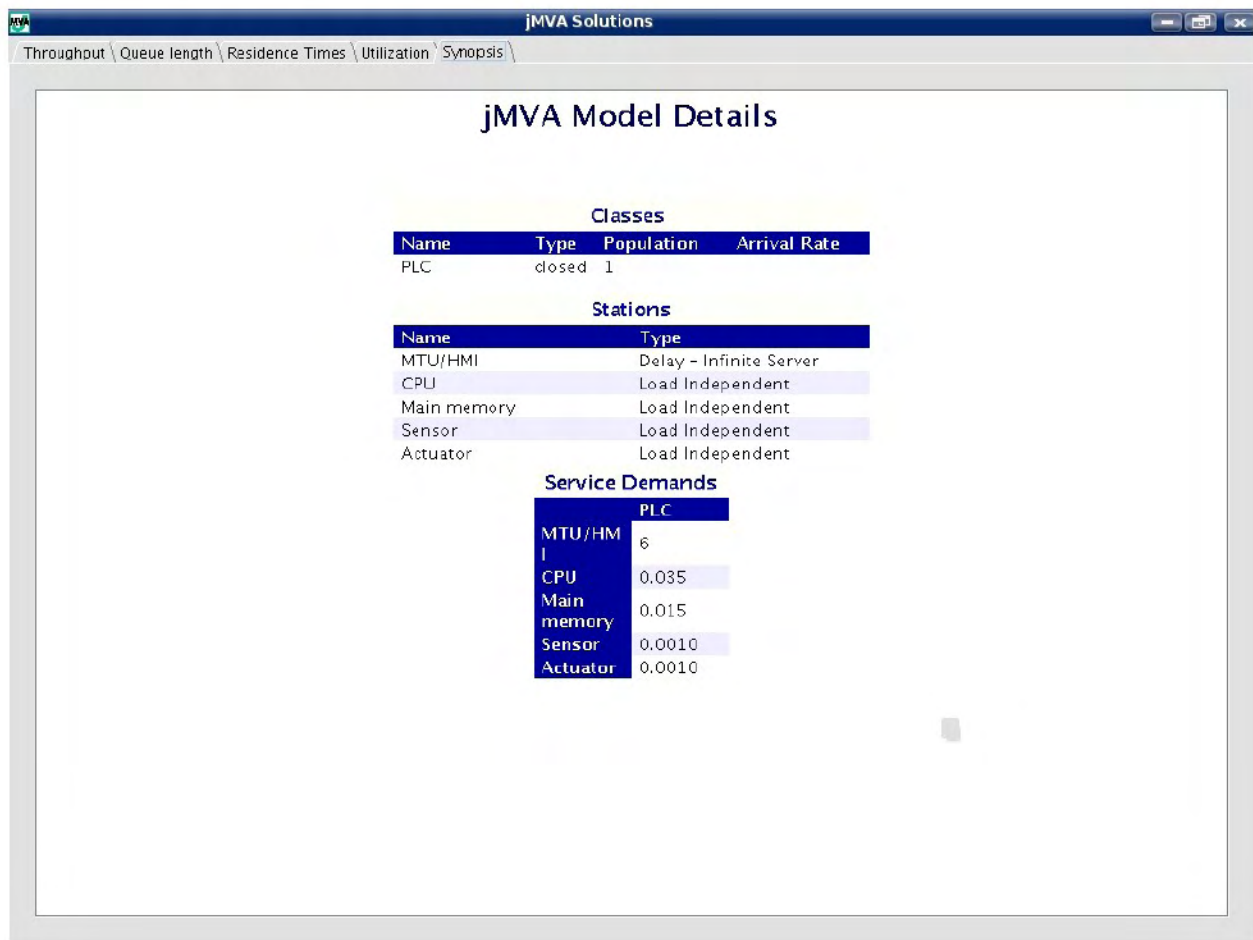












6 – Networks

6.1 – Capacity Planning of a Wireless Lan	244
6.2 – Queueing Network Model of Ad-Hoc Wireless Networks	268
6.3 – Peer to Peer File Sharing	283

Capacity Planning of a Wireless LAN

Sergio Vavassori

March 10, 2008

Project course: Performance Evaluation
Professor: Giuseppe Serazzi

Table of Contents

1 Introduction.....	4
1.1 Why use a Wireless LAN Network Model.....	4
1.1.1 Today's Technology.....	4
1.2 So why to use this Model?.....	4
1.3 The Scenario.....	5
2 The Model.....	6
2.1 Modeling the Scenario.....	6
2.2 The Basic Components.....	7
2.3 The Simulation.....	10
3 The Results.....	11
3.1 What if clients grow from 15 to 80.....	11
3.1.1 The Throughput.....	16
3.2 What if proxy service time grows from 0.18 up to 0.36.....	18
4 Conclusions.....	23
5 Bibliography.....	24

List of Figures

Figure 1: Our client-side wireless network model.....	7
Figure 2: Clients parameters.....	8
Figure 3: Proxy Server parameters.....	8
Figure 4: WLAN parameters.....	8
Figure 5: Outgoing & Incoming link parameters.....	9
Figure 6: Internet parameters.....	9
Figure 7: Finite Capacity Region parameters.....	9
Figure 8: The JMT simulator.....	10
Figure 9: Finite Capacity Region queue length.....	11
Figure 10: Outgoing link queue length.....	12
Figure 11: Proxy Server queue length.....	12
Figure 12: WLAN queue length.....	13
Figure 13: System drop rate.....	13
Figure 14: Finite Capacity Region residence time.....	14
Figure 15: Outgoing link residence time.....	14
Figure 16: Proxy Server residence time.....	15
Figure 17: WLAN residence time.....	15
Figure 18: Incoming link throughput.....	16
Figure 19: Proxy Server throughput.....	16
Figure 20: WLAN throughput.....	17
Figure 21: Finite Capacity Region queue length.....	18
Figure 22: WLAN queue length.....	18
Figure 23: Proxy Server queue length.....	19
Figure 24: Incoming link queue time.....	19
Figure 25: WLAN queue time.....	20
Figure 26: Proxy Server throughput.....	20
Figure 27: WLAN throughput.....	21
Figure 28: Proxy Server utilization.....	22
Figure 29: WLAN utilization.....	22

1 INTRODUCTION

1.1 *Why use a Wireless LAN Network Model*

With this brief introduction we explain why we choose a wireless LAN Network Model. It could seem that network architecture are quite closer one each other but we will show that this assumption isn't always true.

1.1.1 Today's Technology

In today's networks we use heavily wireless networks since they are easy to build and can cover a wide area than standard wired networks. In addition we don't need to take care about how many cable to distribute, how long and where put the plug to go on line, just turn on the wireless card and authenticate the user, if required.

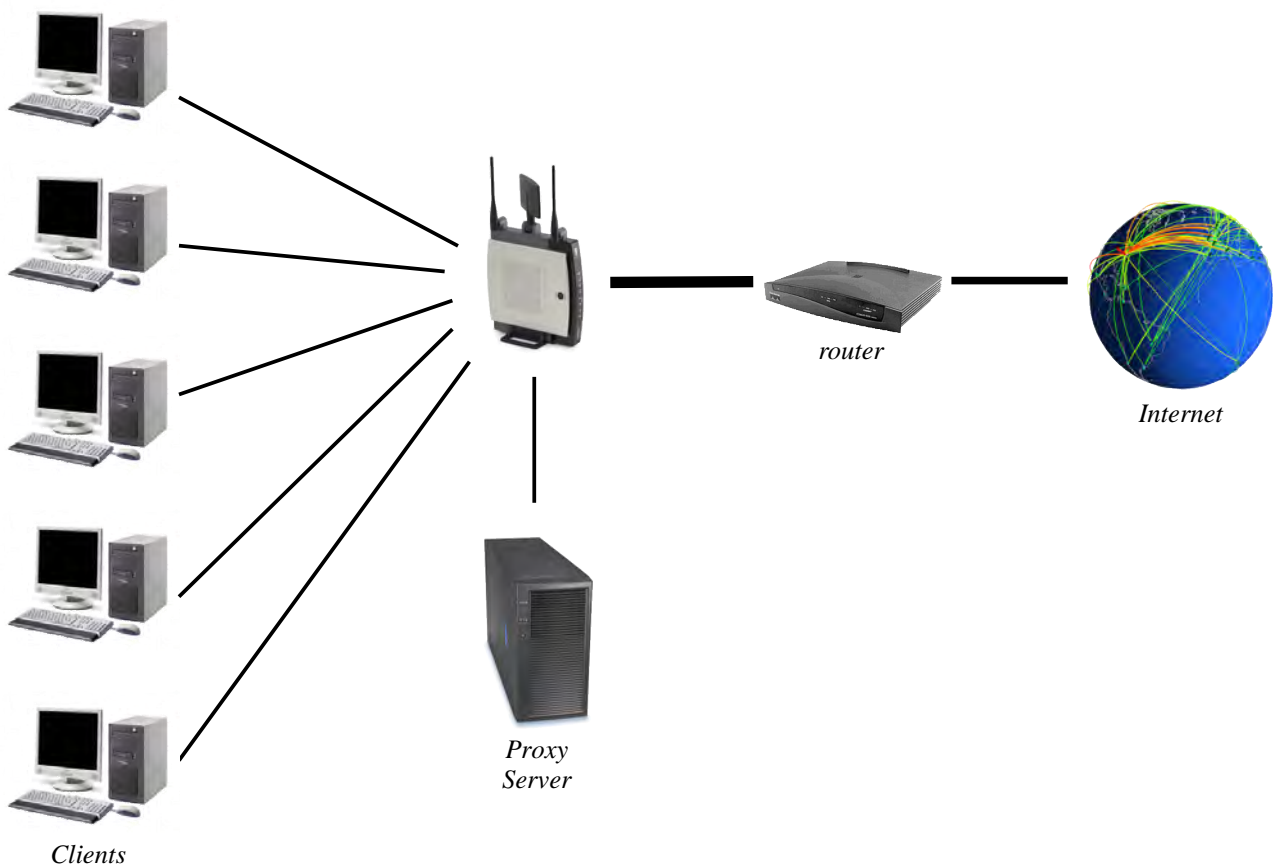
However there are some important differences that most of time we don't see: the first is that wired LAN are switched, this means that we can use all the bandwidth and we shouldn't compete for medium access; the second is that is more difficult to sniff data over a cable than over the air, but the most important one is that today's protocols are developed to work over a reliable channel and the most famous protocol, TCP, must be reviewed to take care of the differences between the two environment.

Sending data over air can be seen as an easy task but the protocol must take care about topology changing and transient faults, as thunder or others natural phenomena that can disturb the communication channel and we must still remember that the bandwidth we have on a cable is much more than the one we can have with a wireless link.

1.2 *So why to use this Model?*

Even though there are so many obstacles on the way, the advantages introduced by using WLAN are so big that cover most of them limitations. In addition there are fields and applications into which there isn't the possibility to use wired networks: we can think, for instance, on a telecommunications between spaceships, airplanes, cargo ships, cars or at instable environments like war scenarios, flooding or landslides.

1.3 The Scenario



This is a typical scenario of open spaces or conference rooms where the laptop (*clients*) are connected at an access point and all the network is behind a Proxy Server that allow (or deny) the request generated by clients. The presence of the router is necessary because usually there are few lines that connect directly to Internet, there can be few IP address available and it could be that your request doesn't need to go outside but can be satisfy by others host connected at others subnetworks.

In addition we assume that even the Proxy Server is connected at a wireless LAN and not by wire thus it competes with clients to transmit over the medium. This assumption can be seen as a not so often used but if we consider that for each service like HTTP or FTP there is a Proxy and that each Proxy could be on a different machine it turns out the convenience to use this type of linking as a generalization of each machine.

2 THE MODEL

2.1 Modeling the Scenario



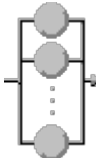





First of all we have to find the equivalent component that models the real one and substitute each component with its model; then we can “wire” the various components to take care about the constraints and the relationship as showed into the table here below (Tab. 1).

As we can see, four different types of components are modeled with the same item and what differs one to the other are the parameters that we will set. In addition we have split the link between the router and the Internet into two links because there could be a difference on bandwidth into uploading and downloading links.

Here below follows a graphical schema that shows the notation used to build the model.

<i>Real Component</i>	<i>Model's Component</i>
Laptop (<i>Clients</i>)	Delay Station
Access Point	Server Station
Proxy Server	Server Station
Router	Routing Station
Outgoing link	Server Station
Incoming link	Server Station
Internet	Delay Station

Table 1: Components and their Models

 <i>Clients</i>  <i>Internet</i>	 <i>Delay Station</i>
 <i>router</i>	 <i>Router Station</i>
 <i>Access Point</i>  <i>Proxy Server</i>	 <i>Server Station</i>

The model that we build, according to the elements described previously, is showed in Fig. 1.

One thing to notice is that the Proxy Server has the same medium as the clients to access the Internet: thus it has to compete with others clients. This is represented with the *Finite Capacity Region* that limits the number of jobs in the WLAN and into the Proxy Server.

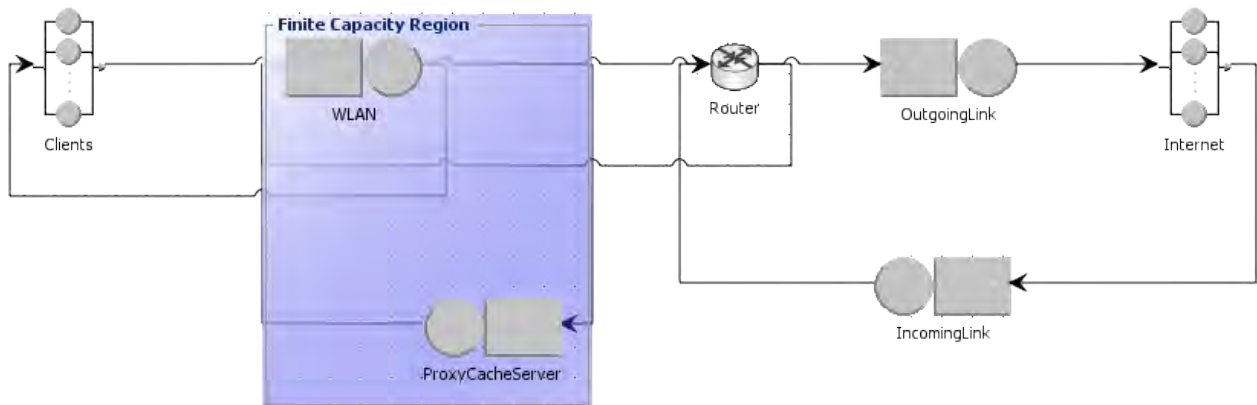


Figure 1: Our client-side wireless network model.

In this model we used a Proxy Cache server to improve responsiveness and reduce response time of web pages asked by clients; a router gives the connectivity to the Internet. The link between the router and Internet is represented by the Outgoing link and the Incoming link: we choose this type of representation because most of connection aren't symmetric and so there is a difference between the upload bandwidth and the download bandwidth; this is common in European Country where the service is carried out by the telephone cable infrastructure.

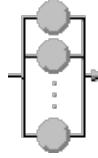
We could use a source and a sink to represent the two links but into this way we will lose the correlation between requests and responses done by clients. The clients represents all people connected to the access point that carry out the signal: this configuration is typical of open space, university classes or airport lounges and allow an easily scaling of the whole infrastructure by adding more access point where needed. Internet is represented with a *Delay Station* because we couldn't see what happens inside: thus we must consider it as a black box on which we can measure the delay between one job enter and the same job exit.

2.2 The Basic Components

This Model is parametrized assuming a starting condition of 15 clients that are connected to the access point. As usual an access point has a maximum limit of clients that can be simultaneously connected, typical 30 clients, and a policy to drop other's connection attempts. We will see that this parameter is fundamental to have a maximum response time and to guarantee an upper bound.

Here follow a resume of all parameters used into the model:

CLIENTS:



- ✓ load independent strategy
- ✓ 15 starting clients
- ✓ exponential distribution with $mean=8$ ($\lambda=0.125$)
- ✓ random routing strategy (since there is only one link)

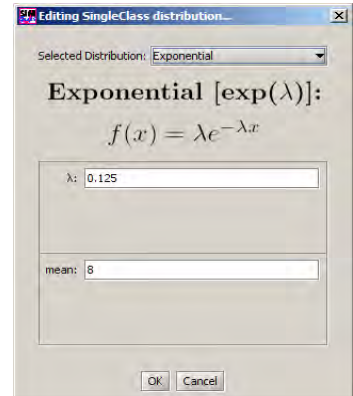


Figure 2: Clients parameters

PROXY CACHE SERVER:



- ✓ FCFS policy
- ✓ waiting queue (no drop)
- ✓ a maximum number of 25 job in queue (finite queue capacity)
- ✓ load independent strategy
- ✓ exponential distribution with $mean=0.18$ ($\lambda=5.5$)
- ✓ probabilities routing strategy equal to 1.0 to WLAN

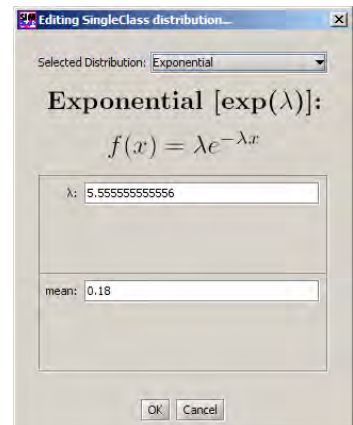


Figure 3: Proxy Server parameters

WLAN:



- ✓ FCFS policy
- ✓ Drop rule enable
- ✓ load independent strategy
- ✓ a maximum number of 30 job in queue (finite queue capacity)
- ✓ Pareto distribution with $mean=0.1$ and $C=3$
- ✓ probabilities routing strategy:

clients	0.5
proxy server	0.23
router	0.12

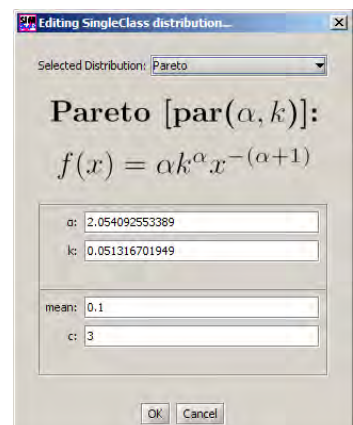


Figure 4: WLAN parameters

ROUTER:

- ✓ only probabilities routing strategy

WLAN 0.35
Outgoing link 0.65

OUTGOING LINK AND
INCOMING LINK:

- ✓ infinite capacity
- ✓ FCFS policy
- ✓ load independent strategy
- ✓ exponential distribution with $mean=0.13$ ($\lambda=7.69$)
- ✓ random routing strategy (since there is only one link)

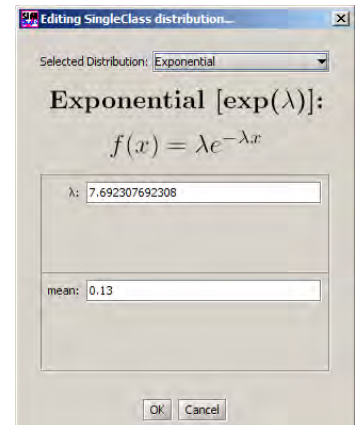
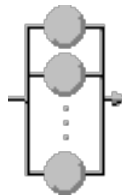


Figure 5: Outgoing & Incoming link parameters

INTERNET:



- ✓ load independent strategy
- ✓ Pareto distribution with $mean=2.3$ $C=3.5$
- ✓ random routing strategy (since there is only one link)

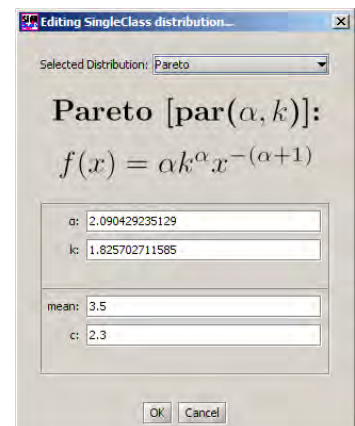


Figure 6: Internet parameters

FINITE CAPACITY REGION:

- ✓ region capacity 45
- ✓ Drop enable
- ✓ class capacity 45

to create a finite capacity region you have to select the wlan and the proxy server and after click on the "finite capacity region button".

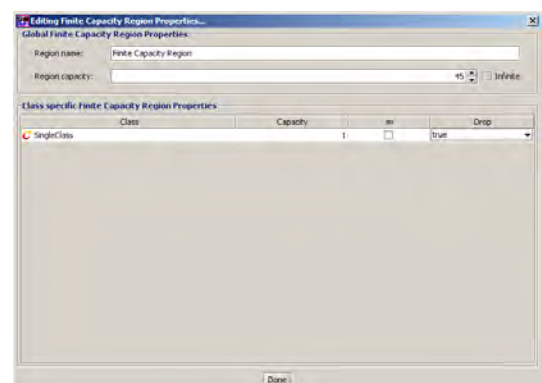


Figure 7: Finite Capacity Region parameters

2.3 The Simulation

We used as network simulator the Java Modelling Tools created at Politecnico di Milano with its JSIMgraph utility that calculates models' behaviour using MVA algorithms.

This simulator is available at <http://jmt.sourceforge.net/>

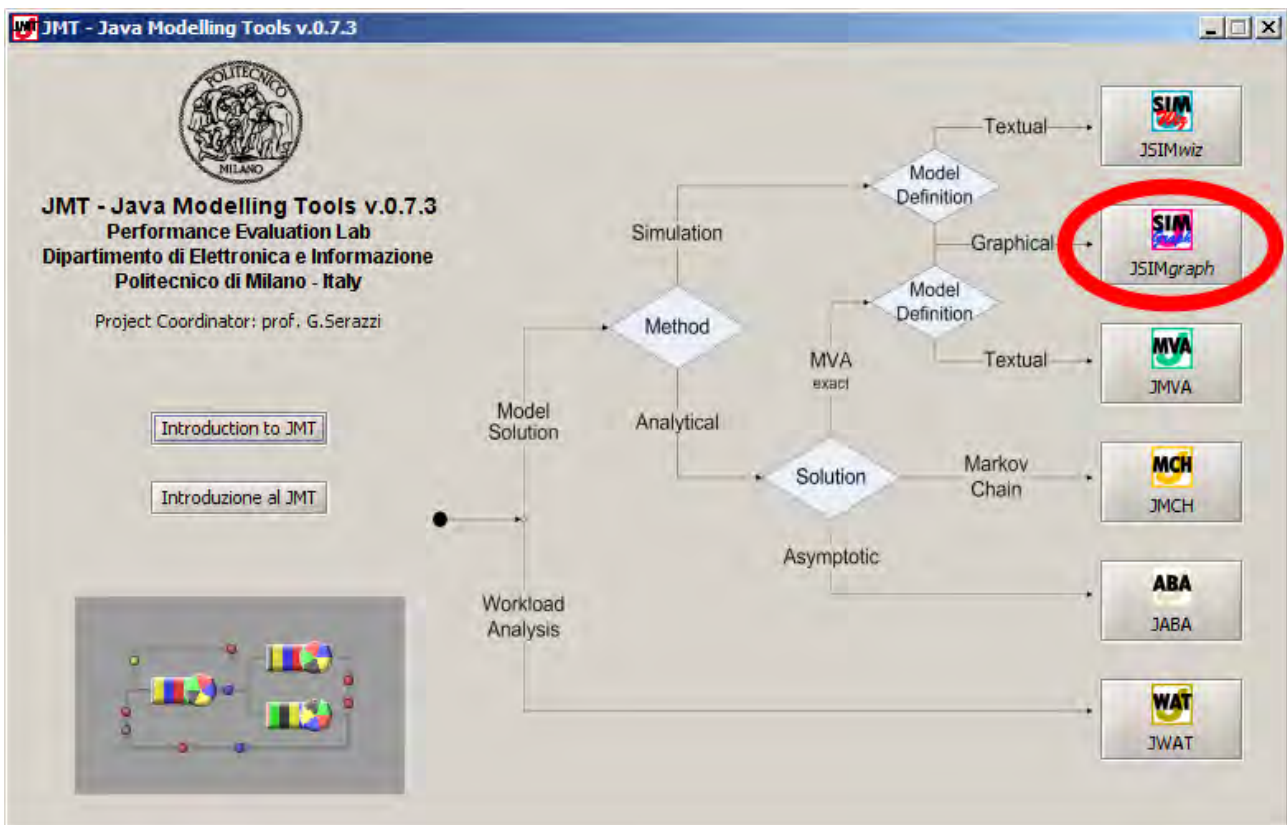


Figure 8: The JMT simulator.

With this simulator we run a first *what-if* simulation that calculates what happens if the number of clients that try to attempt new connections or others activity with the wireless network, grows up to 80.

Thanks to this simulation we can examine the possible bottlenecks and undesired behaviours of our system, without to set up all clients, an access point and all other stuff to do the work. In addition we can easily change one or two parameters and re-run again the simulation to see the different responsiveness of the system. Of course our model must be checked over a real model at the end to be sure that our results are correct.

3 THE RESULTS

3.1 What if clients grow from 15 to 80

Here below are reported some graphs that show how some commons indices of a computer networks system. These indices like *queue length*, *queue time*, *response time*, *residence time*, *utilization* and *throughput* measures different aspects of our system and allow us to comprehend in a deeper way what's going on into that system and to predict with a reasonable uncertain how this system will evolve or discover some bottlenecks.

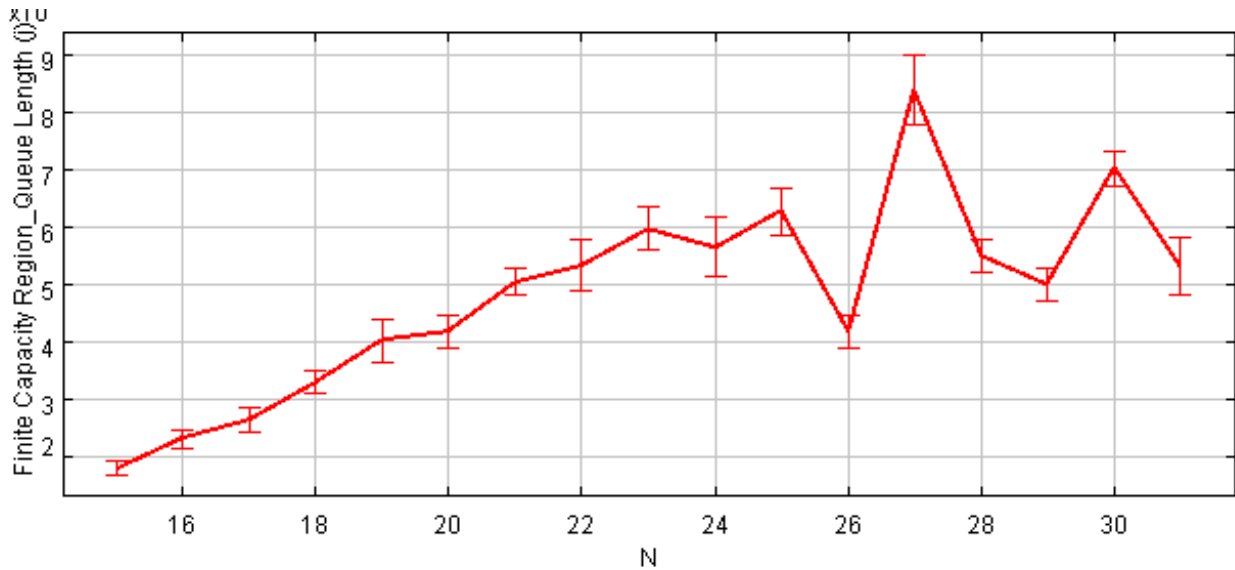


Figure 9: Finite Capacity Region queue length

As we can see (Fig. 9) the queue of the finite capacity region grows but when the clients are 26 or more there is an initial decrease.

As the matter of fact (Fig. 13) the *system drop rate* is increasing linearly and this effect is found in all queue length monitored on **WLAN**, **Proxy Server**, **Outgoing link** and **Incoming link** (the last one graph isn't reported because it's very closer to the outgoing one). We can easily see that around 24-28 clients the queues in these stations decreases (Fig. 10, Fig. 11, Fig. 12).

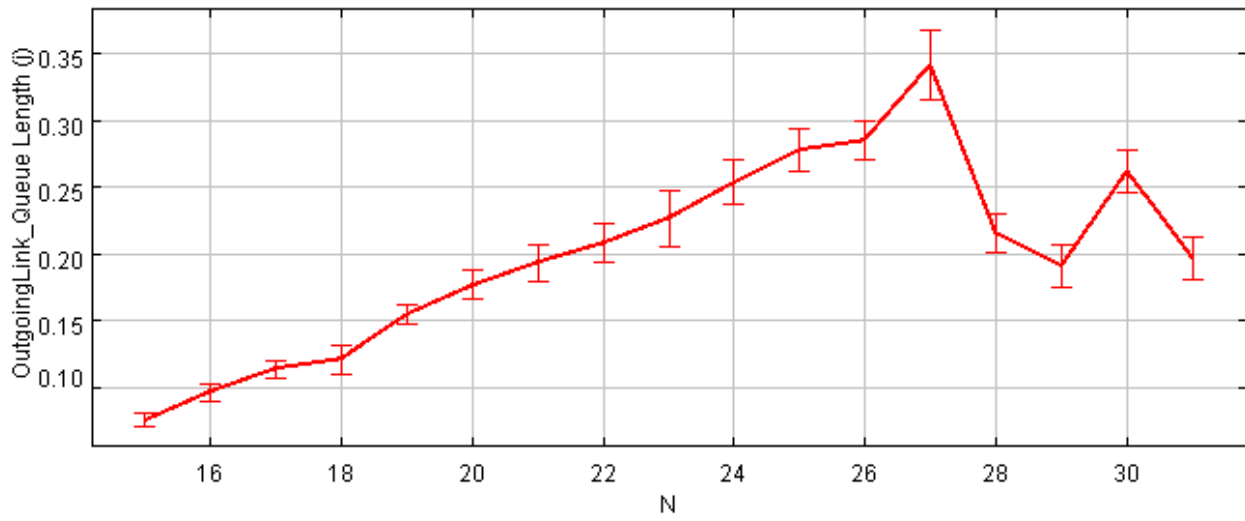


Figure 10: Outgoing link queue length

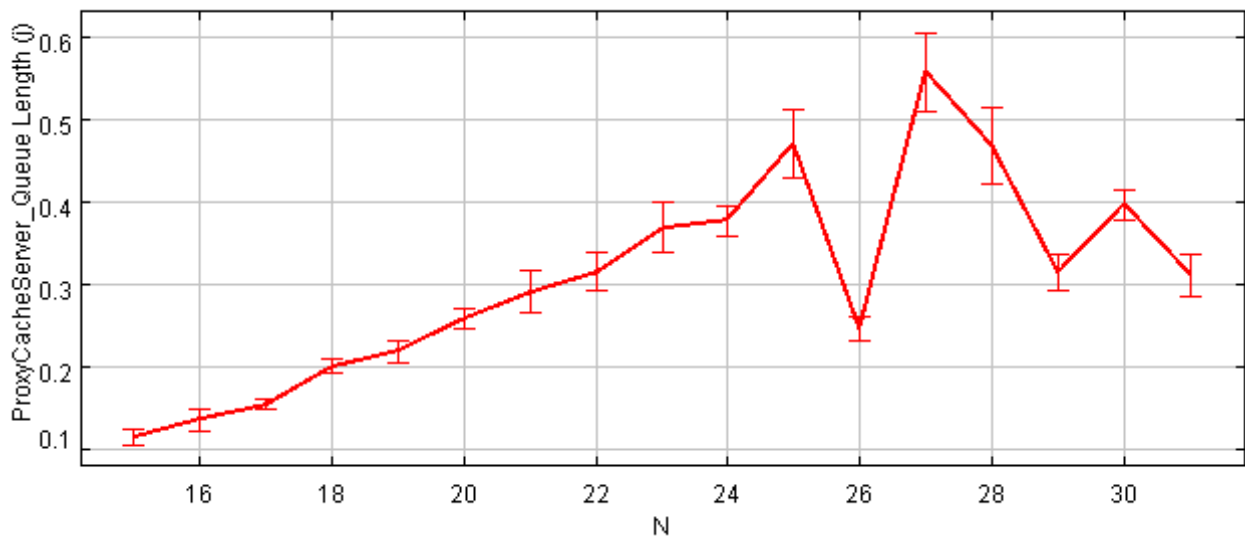


Figure 11: Proxy Server queue length

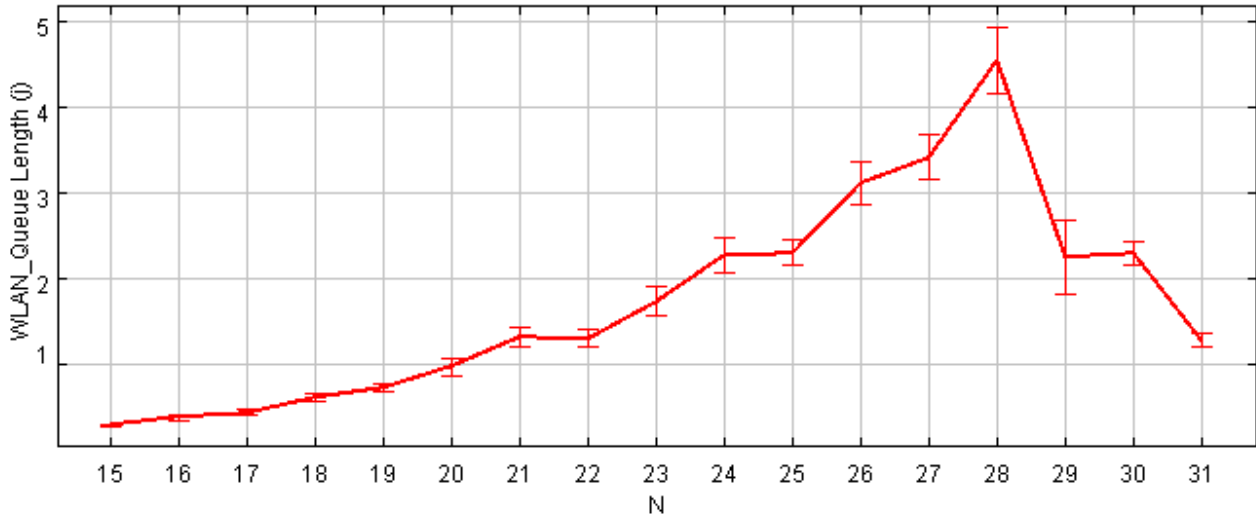


Figure 12: WLAN queue length

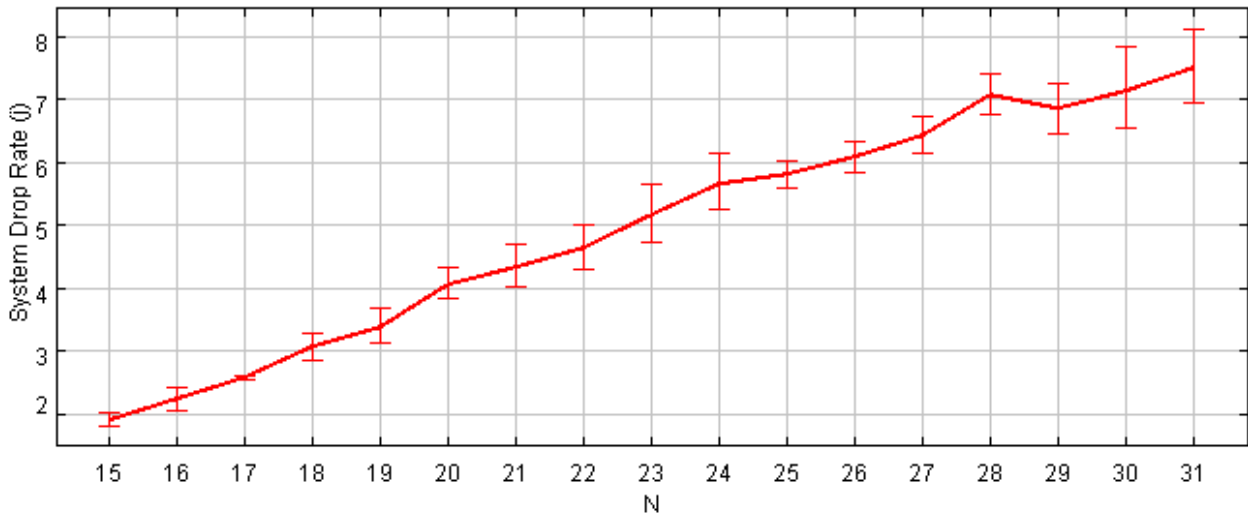


Figure 13: System drop rate

If we take a look at the responsiveness of the system, that is measured by the *response time* and *residence time*, we can observe that more or less all the stations have the same behaviour (see Fig. 14, Fig. 15, Fig. 16 and Fig. 17) but the variance is increasing as shown both by the *confidence interval range* (straight vertical lines on graphs) and by the non-linear shape of the line.

A non surprising fact is that the greater variance is on the WLAN (see Fig 17). This can be explained by looking at the number of link per component: the WLAN is the component with the most number of link, thus, since every link as a variance and a part of jobs aren't correlated (the ones that came from different clients) and the remaining part of are correlated (the ones that came from the *same* client) its variance is very closer to a weighed average between sum of the variance that came from every uncorrelated job and sum of the variance that came from every uncorrelated job minus their covariance, taken by a groups of two jobs.

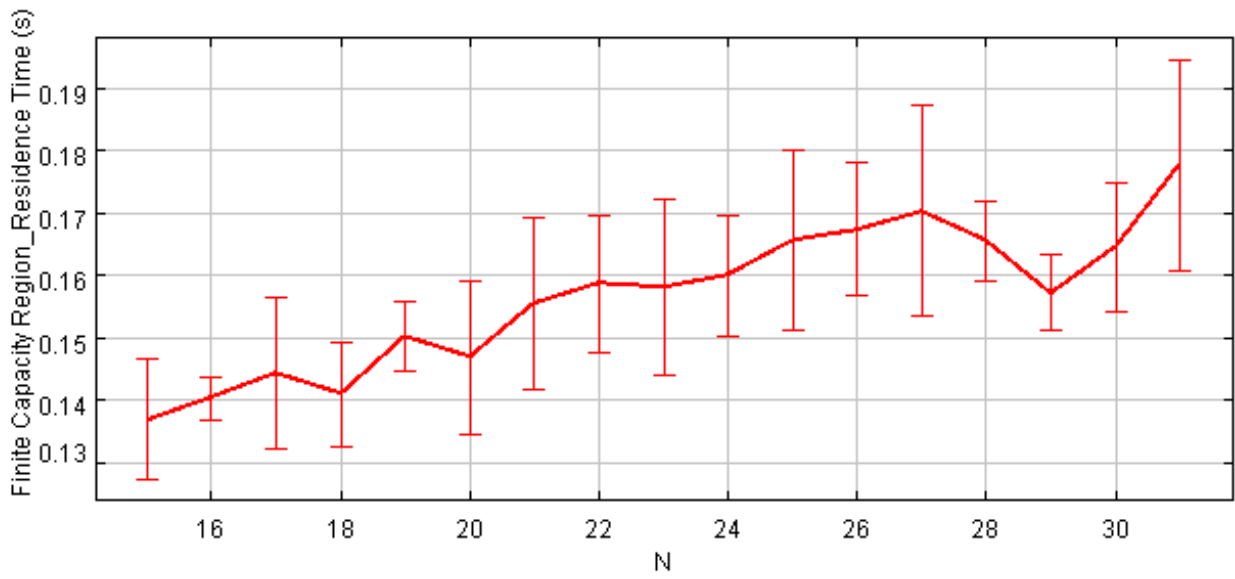


Figure 14: Finite Capacity Region residence time

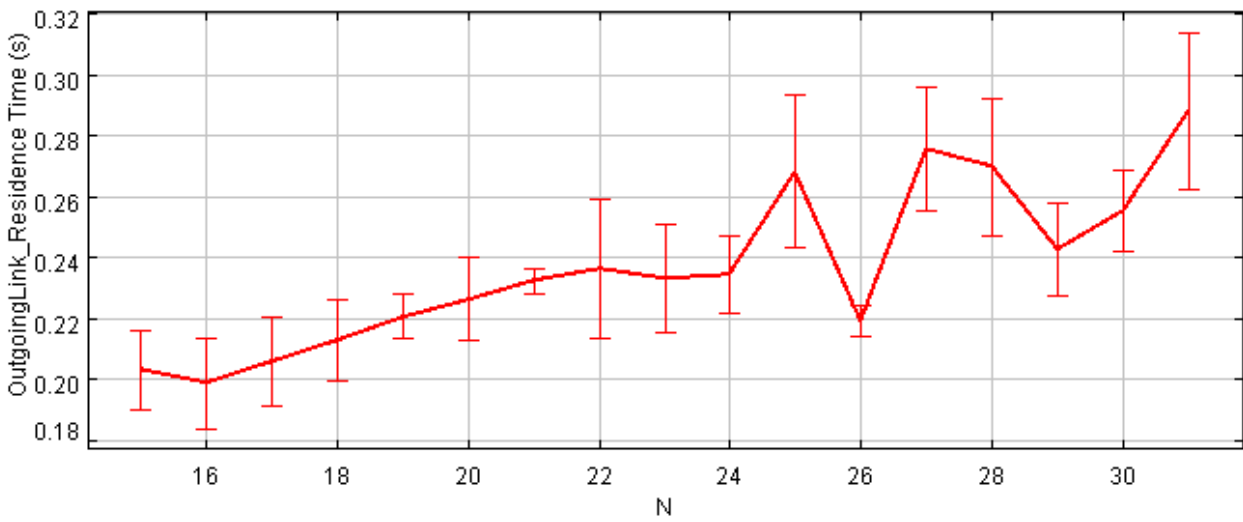


Figure 15: Outgoing link residence time

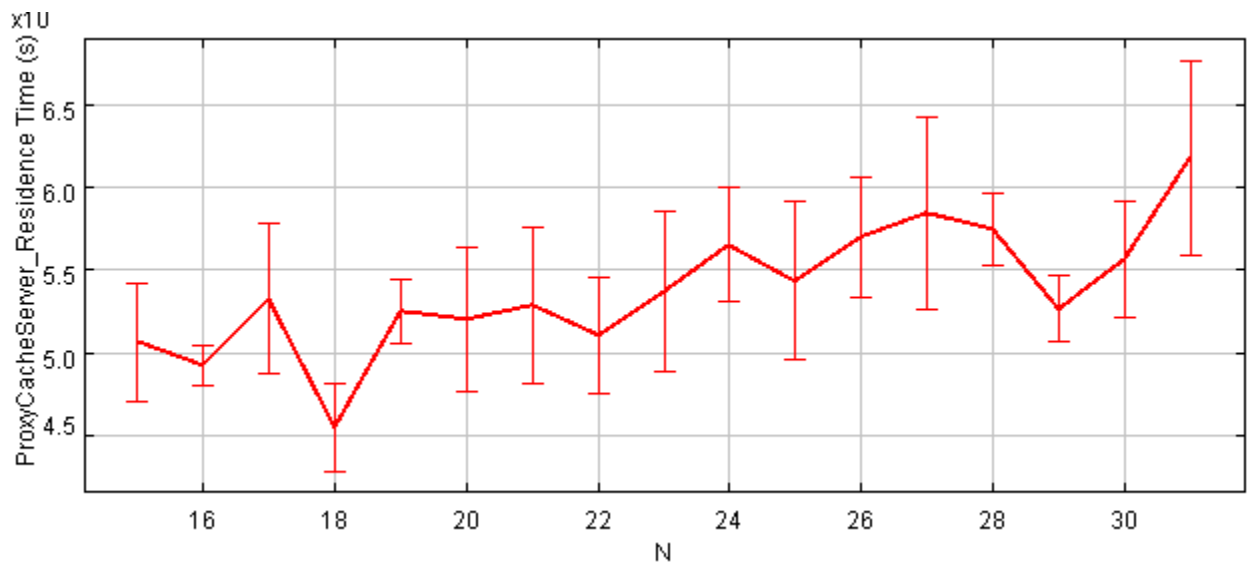


Figure 16: Proxy Server residence time

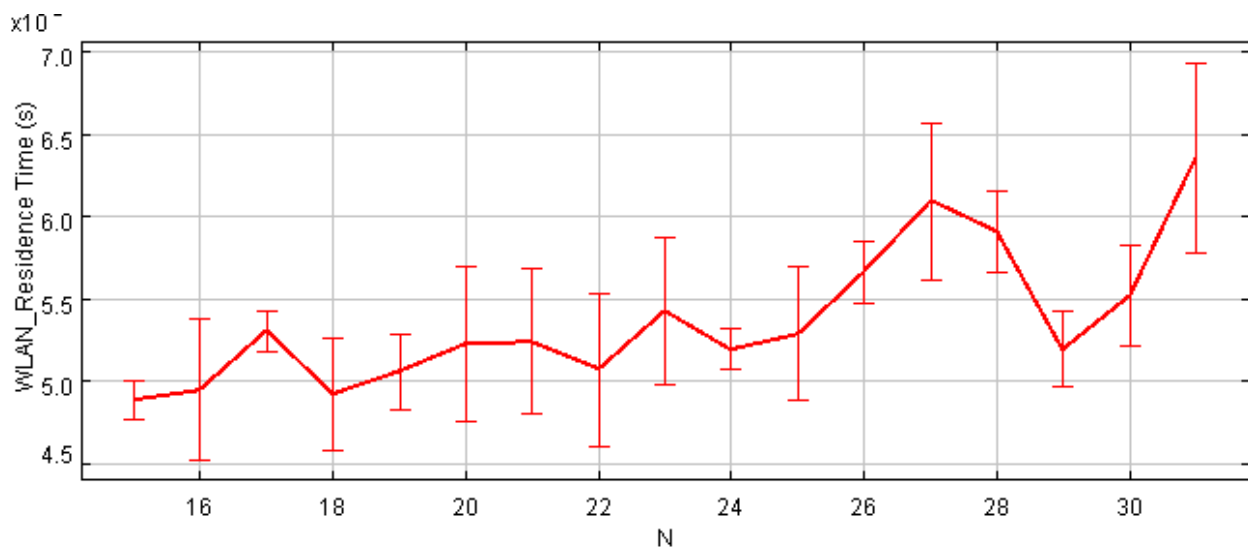


Figure 17: WLAN residence time

3.1.1 The Throughput

The throughput measures how many job are carried out in a second by a station. This measure gives a meter on how “parallelizing power” has the server not on how much time it takes to complete a single job. For instance, if our server station model is composed by N servers the time token to do one job is the same for every server but if we had up to N jobs the time to do all the jobs its the same because it is balanced (if possible) on every n -server.

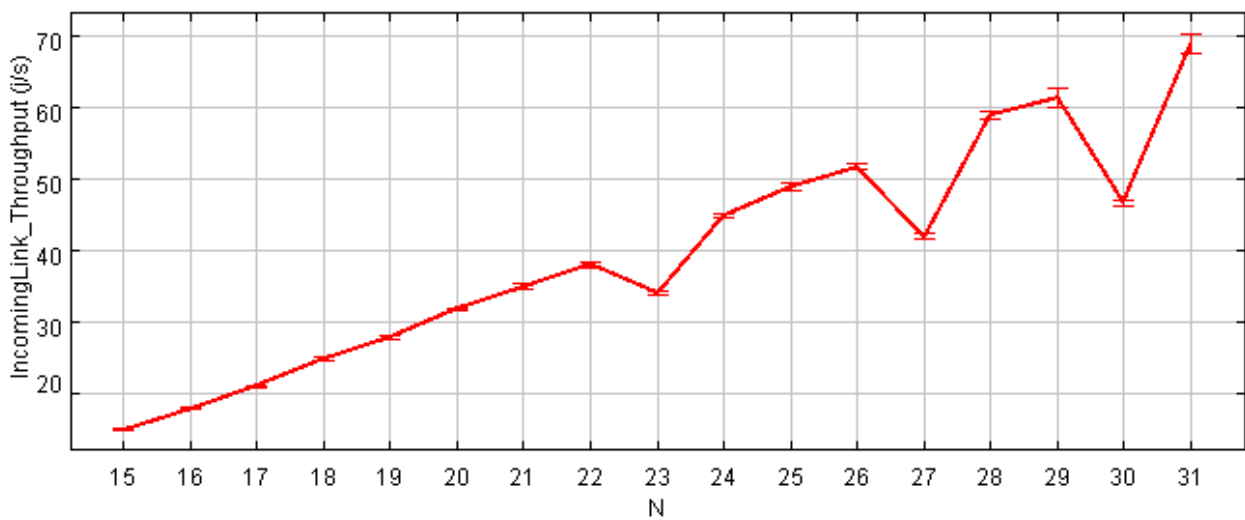


Figure 18: Incoming link throughput

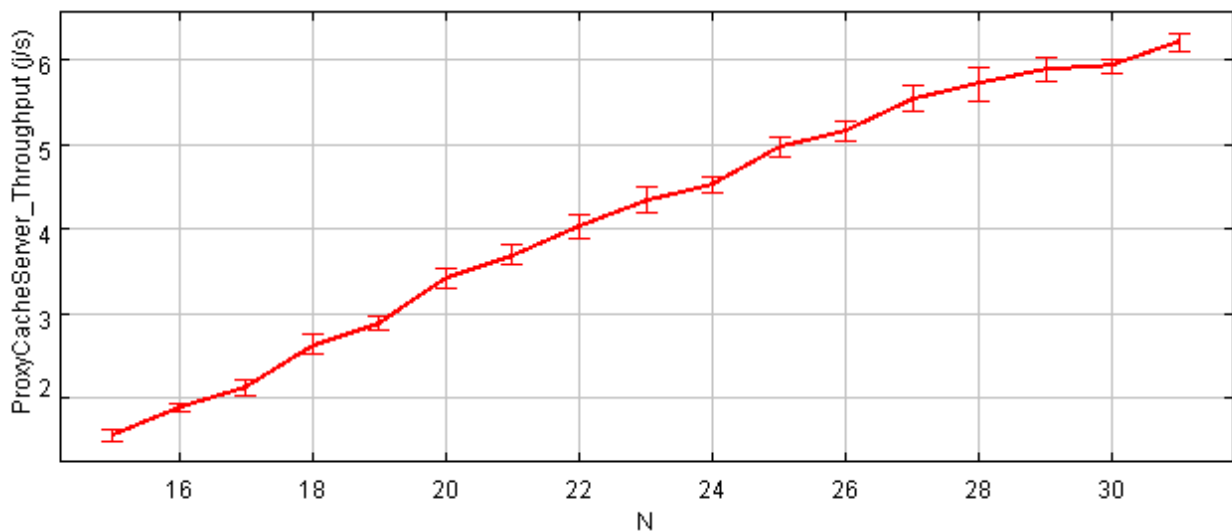


Figure 19: Proxy Server throughput

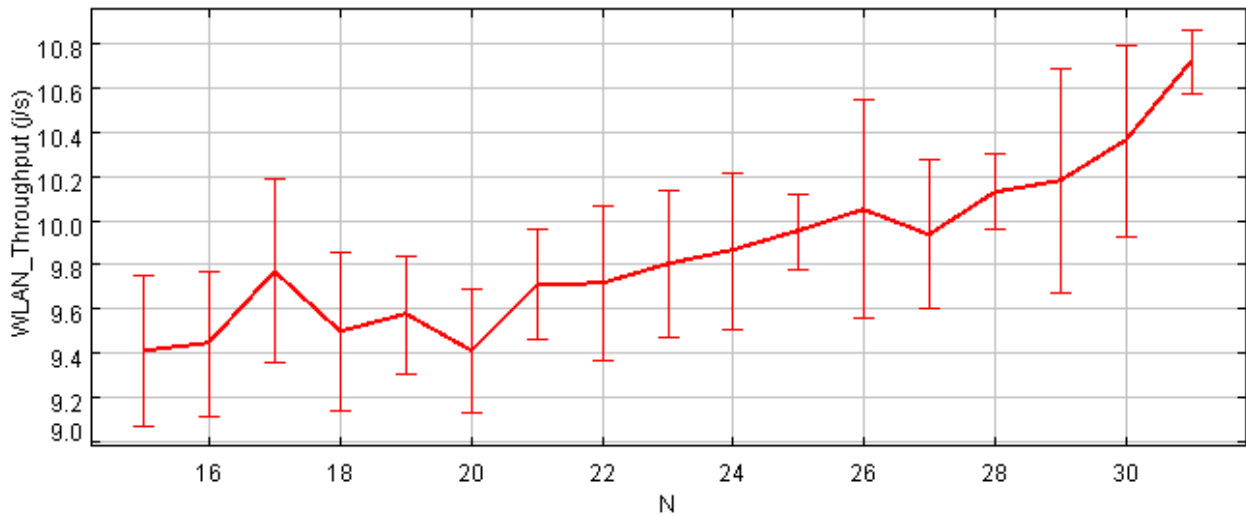


Figure 20: WLAN throughput

The graph of Incoming link station (Fig. 18) show how linear can be its throughput, this is a good behaviour of a system because allow us to predict in an easy way how it evolves with different loads. We can notice, in addition, that even the proxy server has a similar behaviour; instead the WLAN show a non linear trace. That means if we duplicate the workload the responsiveness of the station isn't the double time but can be greater or smaller, depending on the concavity of the curve. Typically these type of curve have a concavity that generate a greater response time because there are some long tail effects with Pareto distribution and this is in according to our assumptions done on WLAN (see Fig. 4).

3.2 What if proxy service time grows from 0.18 up to 0.36

In this new *what if* analysis we changed the response time of the proxy server from 0.18 s. to 0.36 s. As the matter of fact we have duplicated its response time to investigate how the entire system react and to see if the proxy server can be considered as a possible bottleneck.

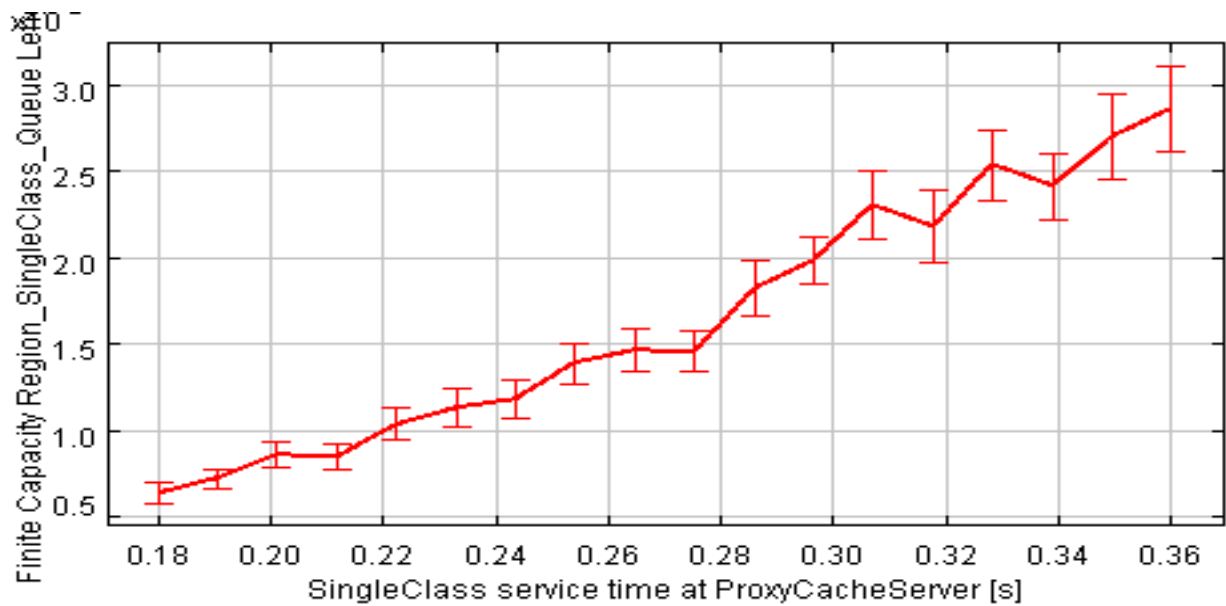


Figure 21: Finite Capacity Region queue length

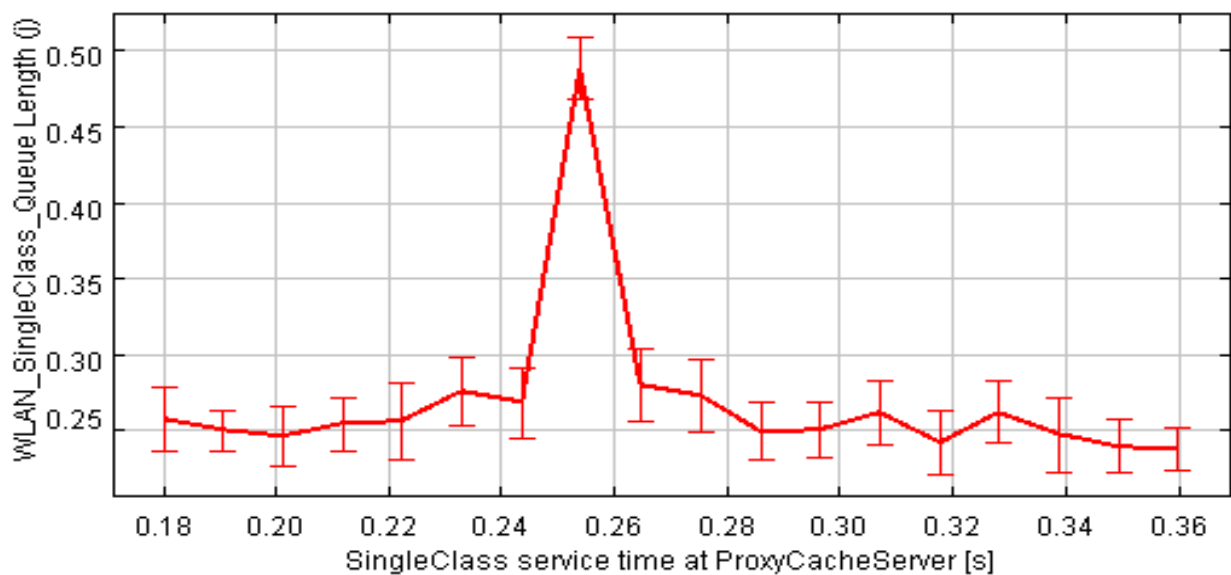


Figure 22: WLAN queue length

An interesting behaviour that we can observe (see Fig. 21) is the queue length of finite capacity region that's quite closer to the proxy server one (Fig.23). This implies there is a direct correlation between the two queues but this correlation there isn't on the WLAN queue (Fig. 22). Thus we can presuppose that when the service time of the proxy server grows, it dominates the WLAN inside the finite capacity region.

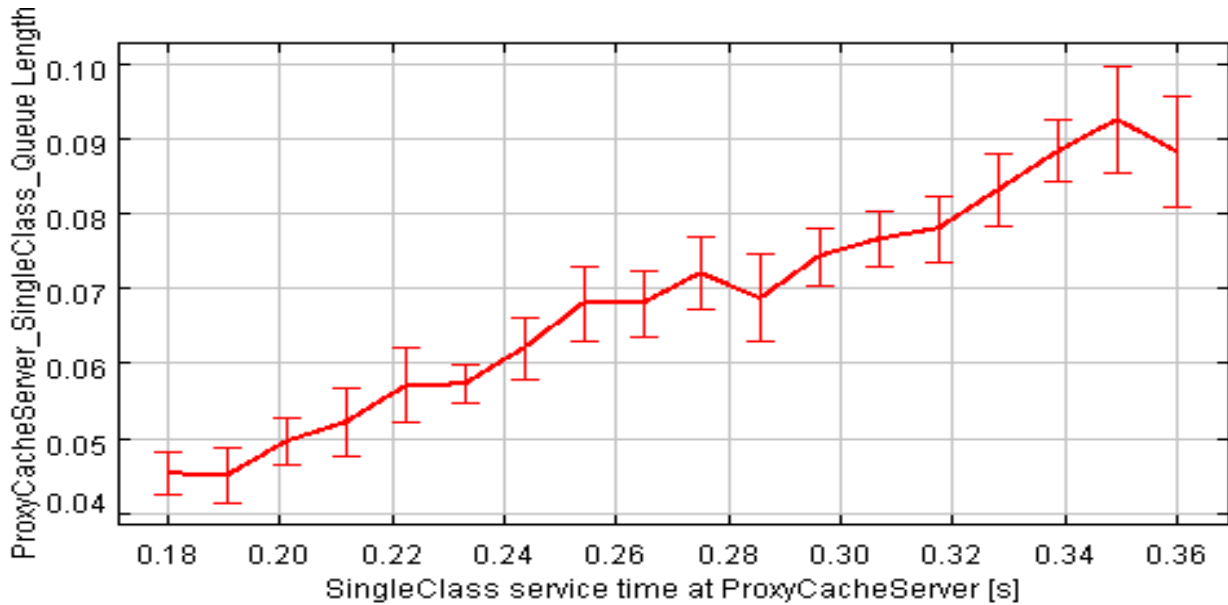


Figure 23: Proxy Server queue length

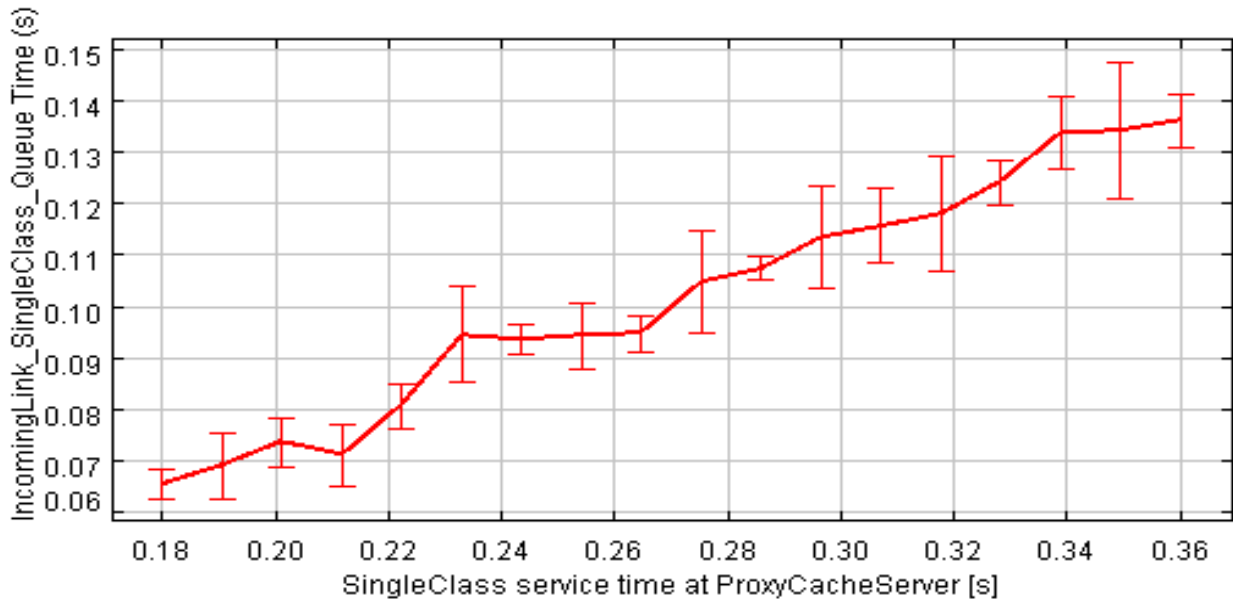


Figure 24: Incoming link queue time

In addition must be notice that even the incoming link queue time (Fig. 24) is quite closer to the first and the third one. An explanation of this can be found looking at the linking: the job that arrive in the incoming link (coming from Internet) go to the WLAN but after this they go, with a probability of 0.23, to the proxy server. Increase the service time is very similar to increase the probability with whom the job goes to the proxy server without modifying its service time.

As the matter of fact, it can be seen looking at the WLAN queue time (Fig. 25); despite the remarkable variance we see that the average is between 3.2 and 4 but doesn't increase according to the change of the proxy's service time.



Figure 25: WLAN queue time

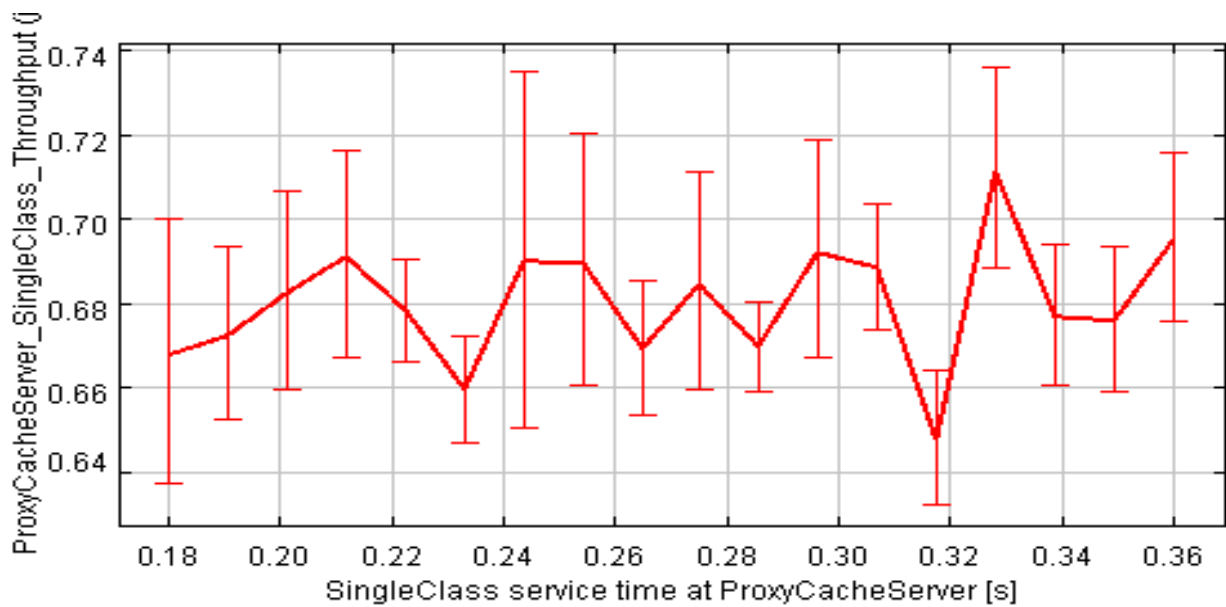


Figure 26: Proxy Server throughput

Instead, if we look at the proxy server throughput (Fig. 26) is easy to see that doesn't change so much but its not so high since it goes from 0.65 up to 0.71.

A possible reason to explain it is to look what happens at the throughput of WLAN and at its utilization (Fig. 27 and Fig. 29): the WLAN throughput is very high and the utilization is pretty much one.

This means that increasing proxy service time incline to overload the WLAN. Thus we can comprehend in a better way what happens in Fig. 25: since the utilization can't be more that one (the higher value are due to a numerical errors of the simulation) once we reach the maximum we cannot get over and so there be two cases: the queue grow or not.

The first case can't be because we have a drop rule enable on the WLAN. This feature that can be seen as a little and now so useful behaviour instead brings great advantages on worst case scenarios and allow the system to have a infinite response time.

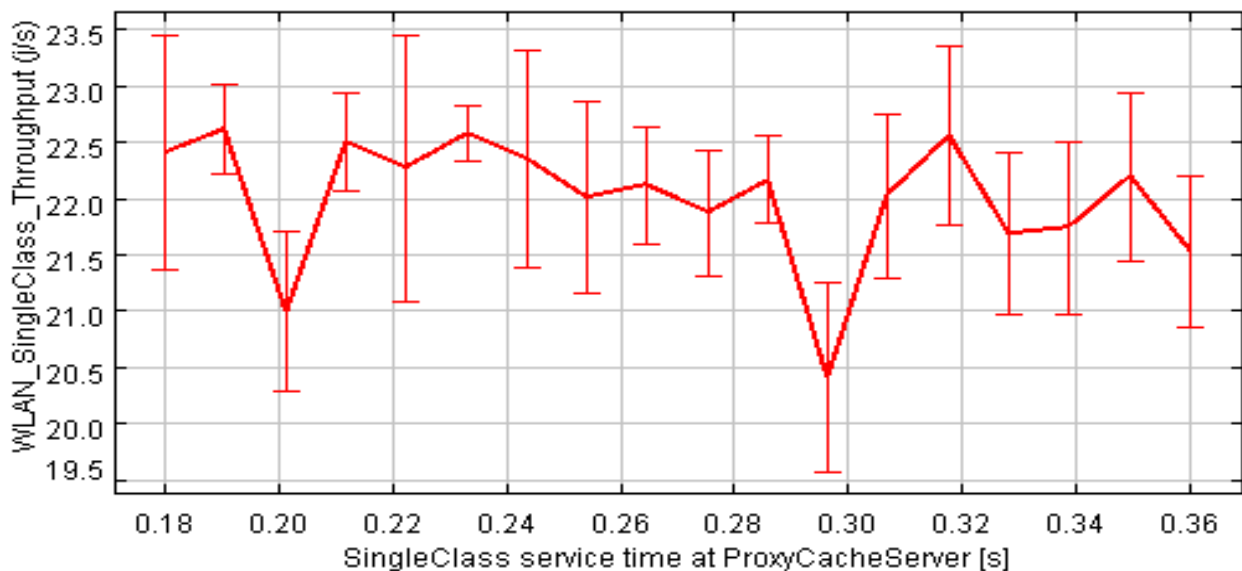


Figure 27: WLAN throughput

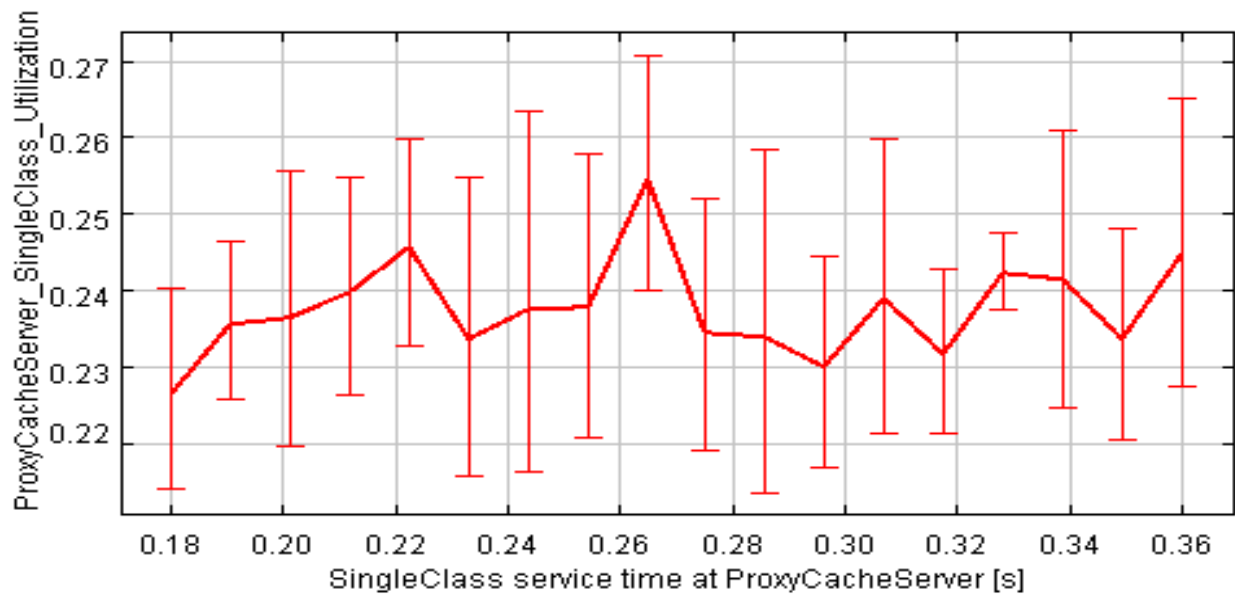


Figure 28: Proxy Server utilization

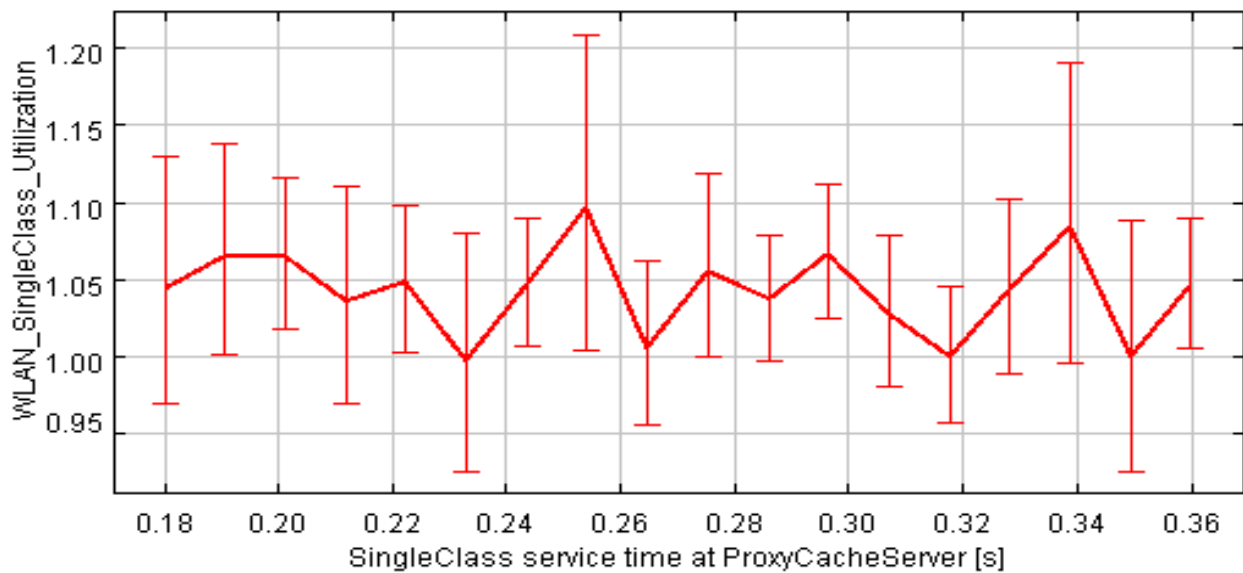


Figure 29: WLAN utilization

4 CONCLUSIONS

As we can see the performance of the system can vary a lot even changing by a little quantity some parameter. This happens because the effect of one component are propagated to the others components: if the propagation has a factor greater than one the effects on other components are amplified otherwise they are attenuated.

The good or bad of this effects depends on which side we are looking at the system: if the system is on heavy load conditions and we move reducing its load then a factor greater than one is positive but if we are in a light load condition, the same factor is negative.

In addition if we have to guarantee that our constraints are satisfied, like for instance, assume that customer's satisfaction is reached if the response time is less than a α value, we have to put inside the system some *features* like *dropping* that allow us to have boundaries on which rely on.

We can prevent some other type of disadvantages like service with heavy tail, as Pareto distribution, that presents a different increment into response time depending to the length of queue with respect to exponential distributions but before changing some component we must predict and possibly ensure, with a certain degree of confidence, how the system will act.

Another important aspect is that if we must guarantee a certain type of availability of our service could be useful to use some filter (like limiting to maximum number some resources) and rounding the customers that try to access the service. As a matter of fact if our responsiveness is too low could be that most of *new* customers aren't motivated to request access to the system.

5 BIBLIOGRAPHY

- ♦ Edward D. Lazowska, John Zahorjan, G. Scott Graham, Kenneth C. Sevcik
Quantitative System Performance - Computer System Analysis Using Queueing Network Models.
Prentice-Hall, 1984.
- ♦ M.Bertoli, G.Casale, G.Serazzi.
The JMT Simulator for Performance Evaluation of Non-Product-Form Queueing Networks.
SCS Annual Simulation Symposium 2007, Norfolk,VA, US, 3-10, IEEE Press.
- ♦ M.Bertoli, G.Casale, G.Serazzi.
An Overview of the JMT Queueing Network Simulator.
Technical Report, Politecnico di Milano - DEI, TR 2007.2, 2007.
- ♦ M.Bertoli, G.Casale, G.Serazzi.
Java Modelling Tools: an Open Source Suite for Queueing Network Modelling and Workload Analysis.
Proceedings of QEST 2006 Conference, Riverside, US, Sep 2006, 119-120, IEEE Press.

Performance Evaluation Project

Queuing model for Ad-Hoc wireless networks

by

Erick Amador, Gerardo Garcia and Sebastian Moreno

Lugano, February 6th, 2007

CONTENTS

1	Introduction	2
2	Problem Description	3
3	Queuing Network Model	4
3.1	Parameters of the model	4
3.2	Simulation settings	5
4	Simulation Results	8
4.1	System Performance	8
4.2	<i>What if</i> analysis	9
5	Conclusions	12
	List of Figures	13
	Bibliography	14

Introduction

In this work the queuing delay and throughput of a multihop wireless ad hoc network is investigated. This kind of network is characterized by a collection of nodes that communicate with each other without any established infrastructure or centralized control, hence the term *ad hoc*.

Because of power limitation issues the transmission of a packet goes through several intermediate nodes that forward the packets until their final destination. In this sense each node may be a source, destination and relay.

The wireless channel is shared, therefore requiring an access control protocol besides scheduling, accounting for an efficient MAC (Medium Access Control) protocol. Since ad hoc networks lack a centralized control the MAC protocol should be distributed, so that random access MAC protocols are suitable for such networks. The delay and throughput of wireless ad hoc networks depend on the number of nodes, the transmission ranges and traffic pattern, all specified by the behavior of the MAC protocol.

In this work the end-to-end delay and throughput in a random access based MAC multihop wireless network with stationary nodes is investigated. By means of a queuing network model the system is simulated with the Java Modelling Tools[1] for performance evaluation of queueing networks.

The packet delay is defined as the time taken by a packet to reach its destination node after it is generated. The average end-to-end delay is the expectation of the packet delay over all packets and all possible network topologies. The average delay is investigated as a function of the degree of locality of traffic, which implies the number of nodes involved (the network density).

The maximum achievable throughput is defined as the maximum packet arrival rate at each node for which the average end-to-end delay remains finite. The simulations do not necessarily aim to evaluate the performance of the MAC protocol but to investigate the overall delay according to network traffic and topology.

In the following sections the network model that is simulated is described, as well approximations that are taken to evaluate the model and actually port it to the JMT simulation tool.

Problem Description

A sample wireless ad hoc network is shown on figure 2.1. Several technologies have been standardized to support either ad hoc networking or mesh (multihop) networking, or a combination of both, some are shown on table 2.1.

The average delay for a packet in a wireless network is the primary characteristic for properly assessing the network performance as a whole. This specific quantity provides insights into more detailed aspects such as network traffic, protocols performance and general achievable transmission rate, characterizing the channel capacity as a functional specification after considering the information theoretic aspects.

Test results are required to verify theoretical ones, so that all aspects of a design are simulated and results are assessed before the respective implementation. In this case the JMT tool for performance evaluation is used. With this tool a queuing network is simulated and different parameters can be changed in order to check for *what if* scenarios, an important aspect of the modelling for communications systems, so that different results can be obtained depending on different network scenarios, topologies and traffic/congestion conditions. Capacity modelling is for communication systems a primary objective, since system capacity and performance need to be specifically design in order to meet tight performance specifications in order to offer a reliable quality of service (QoS) and other different aspects of communication systems.



Figure 2.1: Wireless ad hoc network

Standard	Known as	Operation
IEEE802.11	Wi-Fi	Ad hoc
IEEE802.11s	Wi-Fi	Mesh
IEEE802.15.4	ZigBee	Mesh
IEEE802.20	MBWA	Mesh

Table 2.1: Wireless ad hoc networking standards

Queuing Network Model

The network consists of $n + 1$ nodes, which are distributed uniformly and independently. Each node is assumed to have an equal transmission range $r(n)$, so that r_{ij} denotes the distance between nodes i and j . The traffic model is as follows: each node could be a source, destination and/or relay of packets. Each node generates packets with rate λ packets/sec, the packet generation process is assumed to be an independently identically distributed Poisson process. The packet size is set to be L bits. There is a probability $p(n)$ for which the packet is absorbed or a probability $(1 - p(n))$ for which the packet is forwarded to a neighboring node. When a node forwards a packet, each of its neighbors is equally likely to receive the packet. In this sense the network traffic is localized or unlocalized for high or small values of $p(n)$ respectively.

The packets are served by the nodes on first come first serve basis. No packets are dropped, assuming an infinite buffer from each node.

This multihop wireless ad hoc network is modeled as a queuing network as shown on figure 3.1.

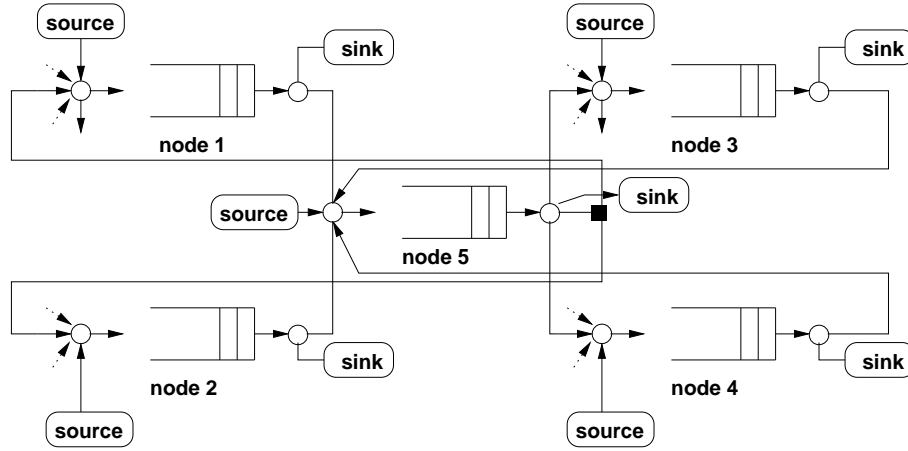


Figure 3.1: Network model

The stations correspond to the nodes of the wireless network.

3.1 Parameters of the model

The following parameters are defined for the wireless network model based on the work of [2]:

- Absorption probability $p(n)$, probability that a packet is absorbed by a node.

$$p(n) = \sqrt{\log(n)/n} \quad (3.1)$$

- Forward probability $(1 - p(n))$, probability that a packet is forwarded by a node.
- Time required to transmit a packet L/W , where L is the packet length and W is the transmission rate.
- Packet generation rate per node λ , which is an i.i.d Poisson process.
- Effective packet arrival rate at a node is $\lambda_i = \lambda/p(n)$.
- Expected number of hops traversed by a packet: $\bar{s} = 1/p(n)$.
- Service time per node:

$$X_i = \frac{L}{W} \cdot \frac{\lambda}{p(n)} + \frac{L}{W} \quad (3.2)$$

As shown on the expression for the service time per node, the MAC protocol operation is considered since the stations have a back-off timer (with exponential distribution) that prevents all stations from transmitting at the same time instant. For this case the MAC operation is simplified as the service time considers the number of interfering nodes that may prevent a transmission to happen at a desired time instant. In this sense the expected number of hops traversed by a packet is considered for the service time on each node. For this case there is no contention for the channel as the average queuing delay is considered for each node's service time, simplifying the the MAC model for simulation purposes.

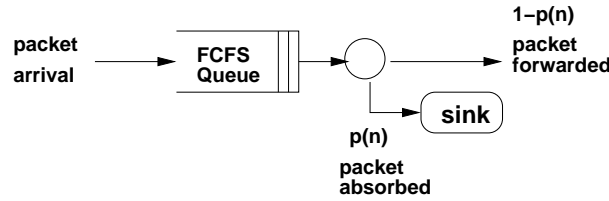


Figure 3.2: Representation of a node

Figure 3.2 shows the representation of a node, with the parameters that characterize the activities of packet generation, absorption and forwarding.

3.2 Simulation settings

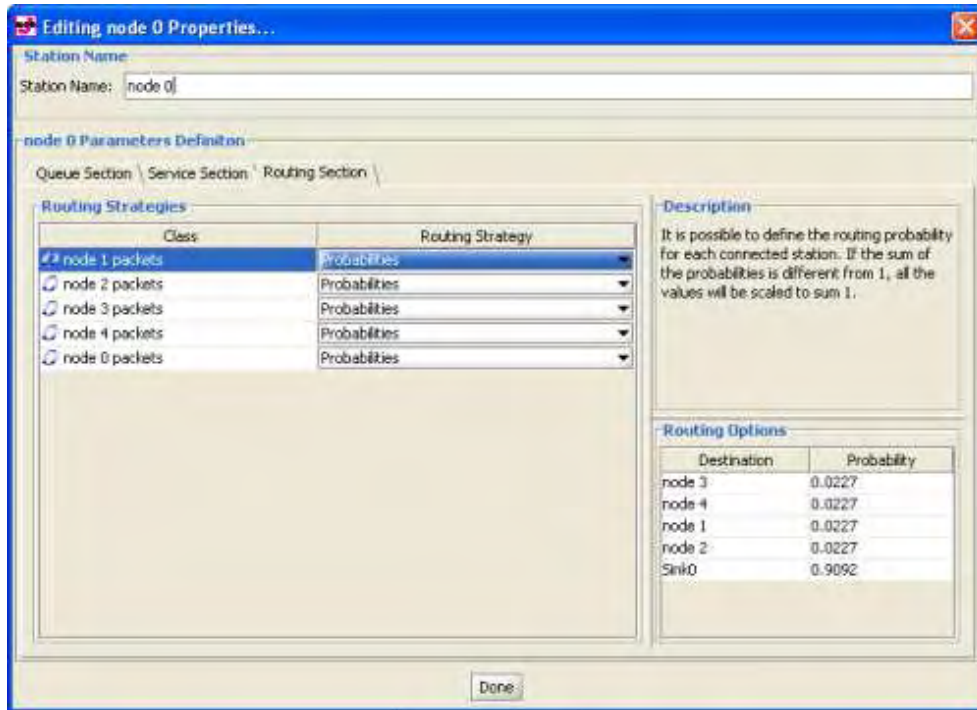
Several settings serve as input for the simulation tool, as for example the number of nodes use would determine the different probabilities of packet absorption or forwarding. The number of peers also has an impact on the service time per node, table 3.1 shows these settings for a fixed value for $\lambda = 0.5$.

Number of nodes	Absortion probability $p(n)$	Service time
1200	0.0506	0.01088
1000	0.0547	0.01014
800	0.0602	0.00931
500	0.0734	0.00781
300	0.0908	0.00651
100	0.14142	0.00454

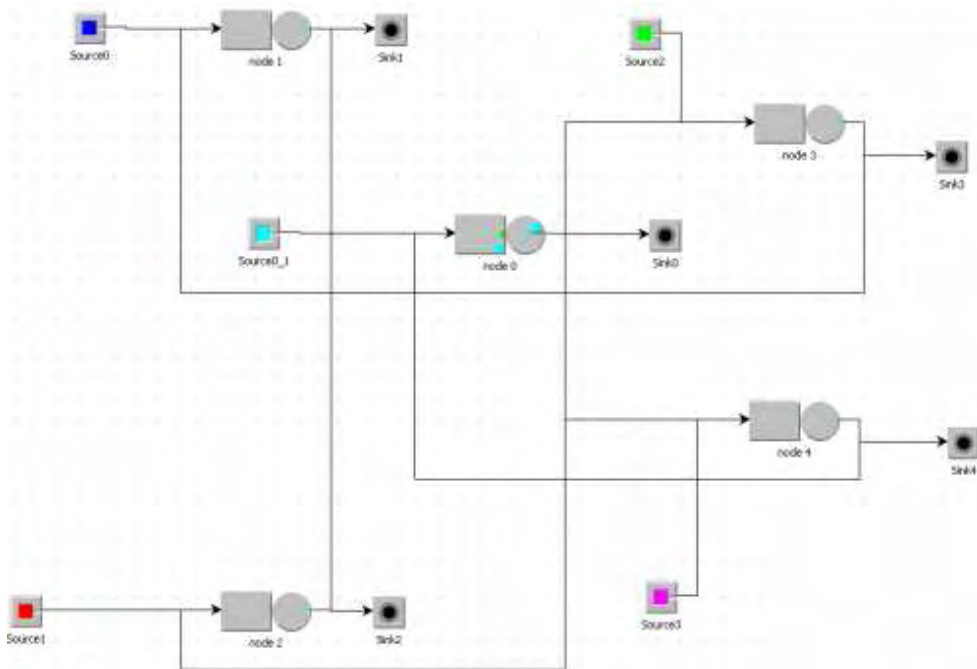
Table 3.1: Simulation parameters according to number of nodes

Under the JMT simulation tool the *System response time* is taken as the average end-to-end delay within the wireless network. This time considers the time between job arrival and departure from the network.

The simulation tool was configured primarily with a graphical interface, such that figure 3.3(b) shows the input queuing model used for simulations. Each server corresponds to a node in the ad hoc wireless network. Figure 3.3(a) shows the configuration screen from the JMT tool in order to configure a server (a node) and the multiclass option used since different classes were used to show the different packets that are generated on each node.



(a) Node configuration



(b) System model

Figure 3.3: JMT configuration

Simulation Results

4.1 System Performance

The queuing network model was successfully simulated on the JMT tool. The system behavior was observed on different levels. System response time is taken as the primary performance metric since it shows the average end-to-end delay on the network.

The service time as shown on 3.2 is a function of the packet arrival rate, in figure 4.1 the service time is observed for different values of λ and different number of nodes.

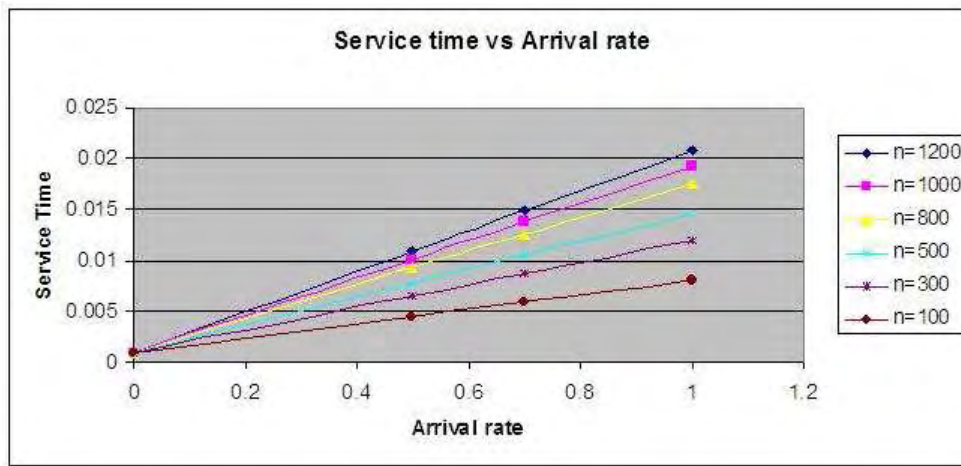


Figure 4.1: Service time

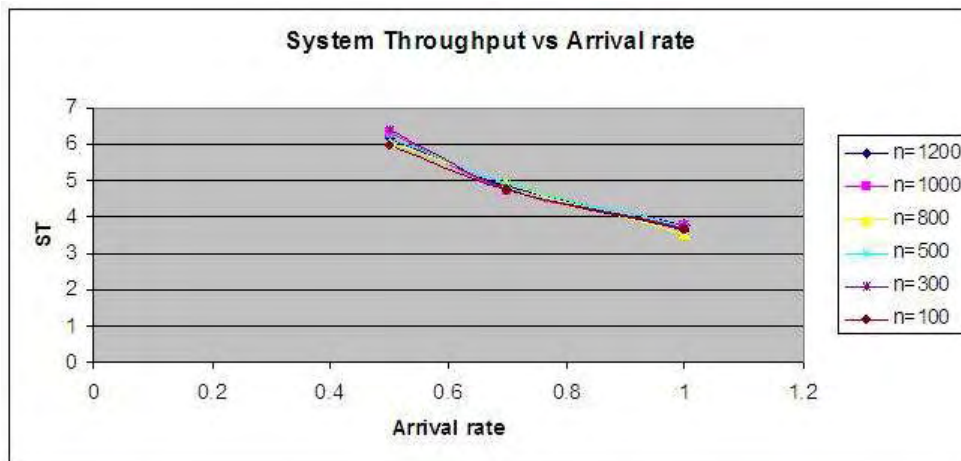
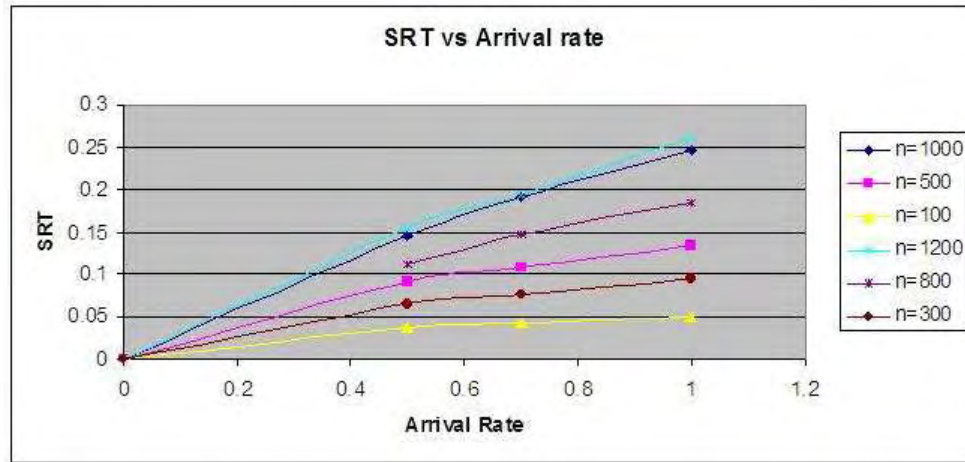


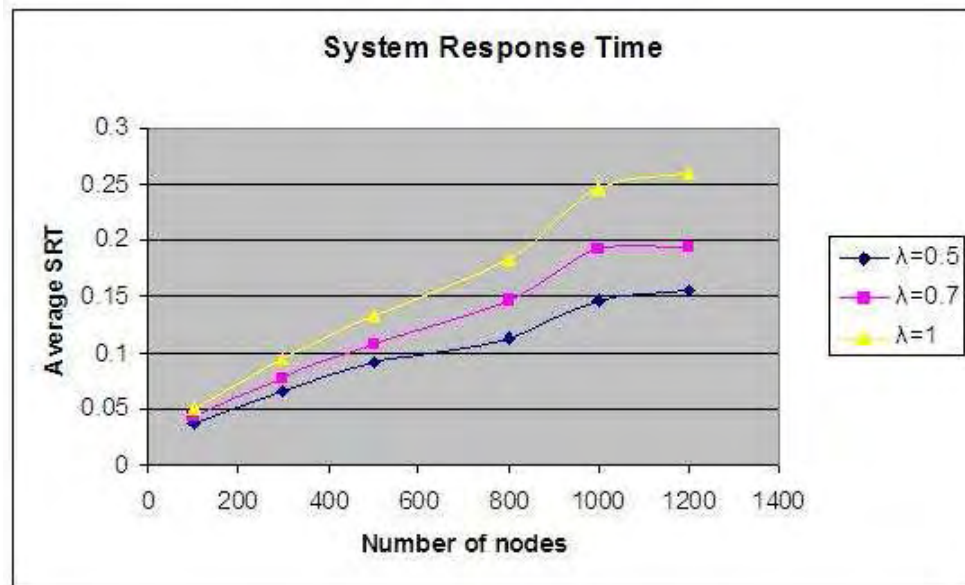
Figure 4.2: System throughput

Figure 4.3 shows how the system response time varies with the number of nodes for different values of λ , $\{\lambda = 0.5, \dots, 1\}$, with $p(n)$ according to 3.1. This value characterizes

the average end-to-end delay on the ad hoc wireless network. As the network grows each packet has to traverse a longer path over all possible topologies in order to reach its destination, hence the delay grows with the number of peers.



(a) As a function of λ



(b) As a function of n

Figure 4.3: System response time

4.2 What if analysis

The JMT tool allows to foresee several scenarios depending on the change of one or several parameters. In this case a *what-if* analysis was performed to observe the systems response after changing key parameters as λ and $p(n)$ (service time affected). Figure 4.2 shows how the system response time (network average delay) changes according to several values of

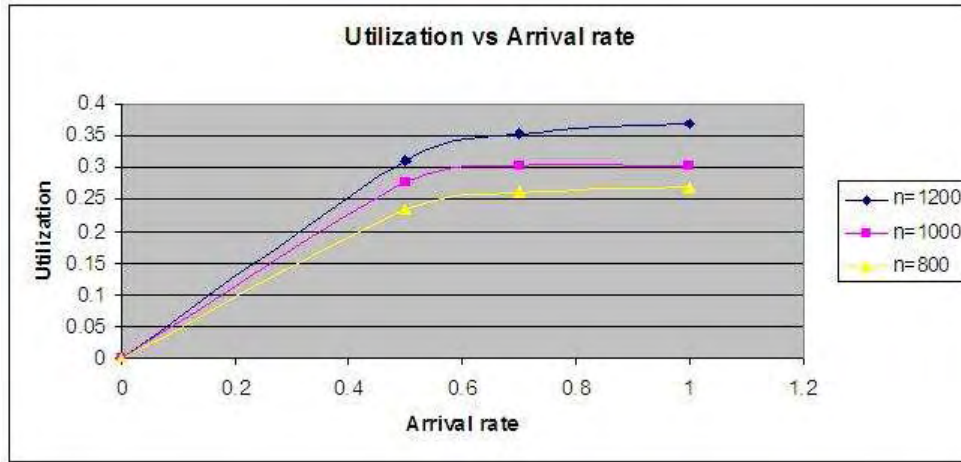


Figure 4.4: Routing node utilization graph

the service time spent for every packet on each node. In figure 4.2 the system response time is observed for a range of values for λ , ranging from 100% to a 200%. Figure 4.2 shows the same curve for an extended range for λ .

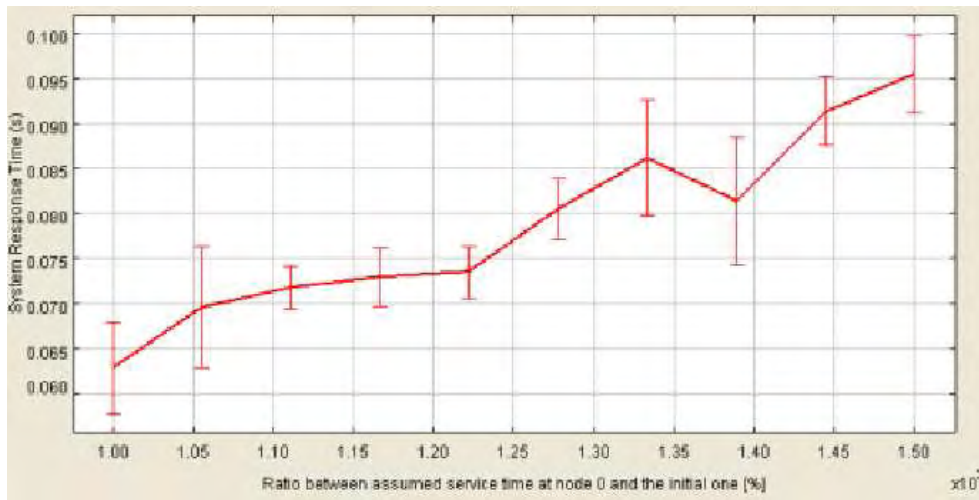


Figure 4.5: What if analysis for service time

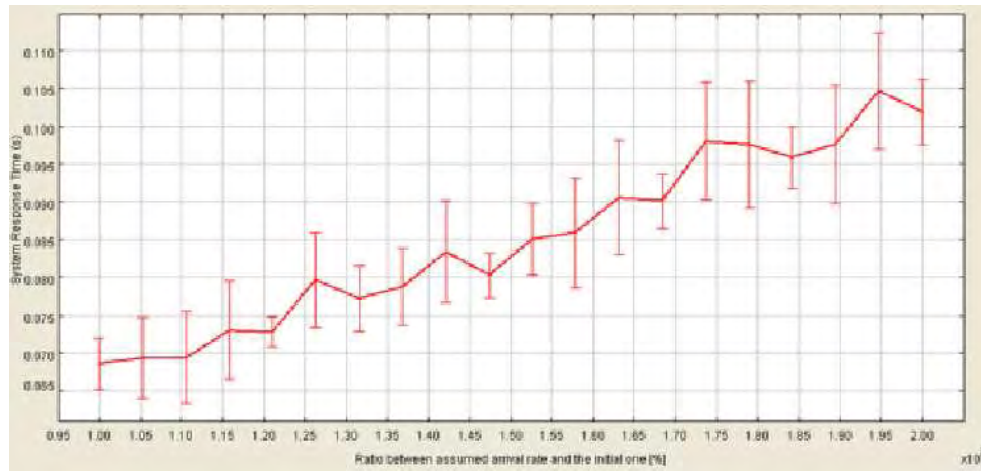


Figure 4.6: What if analysis for a range of λ

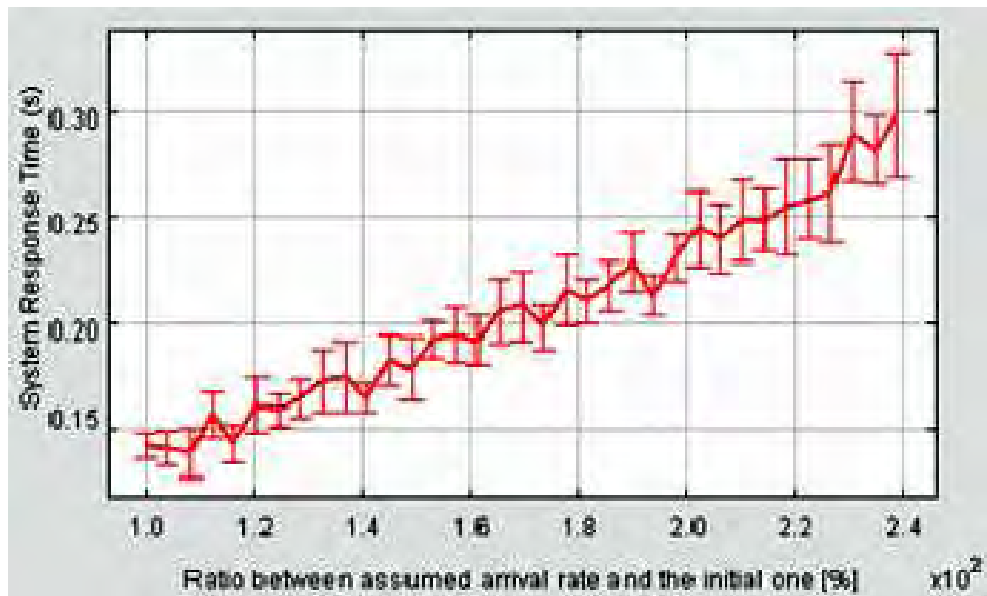


Figure 4.7: What if analysis for another range of λ

Conclusions

The capacity and performance of wireless networks have been a focus of research as communication technologies grow in demand, deployment and cost. Ad hoc networks are gaining considerable attention since sensor networks and other decentralized architectures and topologies offer different advantages for specific applications.

The average end-to-end delay was explored after using a open queuing network model with the JMT simulation tool. A multiclass simulation was used to be able to establish the difference between packets that are generated from different nodes, and also different parameters were setup according to the topology and dimension of the network.

The dimensions of the network have an impact on the overall performance, as seen on the different relations from equations 3.1 and 3.2. The system response time was observed on the *what-if* scenarios for different values of packet arrival time (λ) and service time per node.

The JMT tool allowed us to simulate an ad hoc network that uses a random access MAC protocol after assuming crucial facts as the inclusion on the service time of the delay provided by other interfering nodes (the channel is indeed not accessed by all nodes at the same time).

LIST OF FIGURES

2.1	Wireless ad hoc network	3
3.1	Network model	4
3.2	Representation of a node	5
3.3	JMT configuration	7
4.1	Service time	8
4.2	System throughput	8
4.3	System response time	9
4.4	Routing node utilization graph	10
4.5	What if analysis for service time	10
4.6	What if analysis for a range of λ	11
4.7	What if analysis for another range of λ	11

BIBLIOGRAPHY

- [1] M. Bertoli, G. Casale, and G. Serazzi, “Java Modelling Tools: an Open Source Suite for Queueing Network Modelling and Workload Analysis,” *QEST 2006. Third International Conference on Quantitative Evaluation of Systems*, 2006. [2](#)
- [2] N. Bisnik and A. Abouzeid, “Queueing delay and achievable throughput in random access wireless ad hoc networks,” *SECON '06. 2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, 2006. [4](#)

Peer to Peer file sharing
(Anastasia Stulova)
(Ramy Gad)

Project for the course on
“Performance evaluation”,
Master ALaRI,
January 2008

Abstract

Peer-to-peer file sharing has emerged in many network applications, from downloading games, songs and movies, to even sending updates to programs. In this paper we build a model for a file sharing network, we get the parameter of the model from [1], we propose three types of users: first that contributes to the capacity of the systems by uploading files; second that only downloads files after stays offline for a long time then returns back and third type users who just enter the system to download and exit immediately.

1 Introduction

Peer-to-peer (P2P) file sharing is now too attractive for files like songs, movies and games. The approach of P2P file sharing (figure 1) differs from the traditional client server approach (figure 2), as in a P2P file sharing a peer can be a client and server at the same time. A peer can request files from its peers and store, serve files to its peers. A peer generates a workload for a P2P application as well as providing the capacity to process the workload request of others. The result of that as the number of peers in the network increase the workload increase but also the ability to process this workload increase. While in case of a traditional client server approach clients generate work load which is processed by the servers. In peer-to-peer file sharing networks the life time of the peer is transitory a peer can be active for some time doing uploading and downloading, then go off-line, also sometimes we can have download request

more than the available download bandwidth. This happen in case of congestion when there is file which has few replications in the system and it is requested by many peers.

Peer-to-peer file sharing is very attractive for people who like to watch movies, listen to songs, and play games and use software without buying them. In this way they share copyright materials, they attack the intellectual property and this is illegal in most countries.

Currently a lot of Internet Service Providers (ISP) prevent peer-to-peer file sharing in their networks due to copyright problem and because peer to peer file sharing can slow down the network. Some file sharing software encrypts their traffic so that they can't be detected by the ISP.

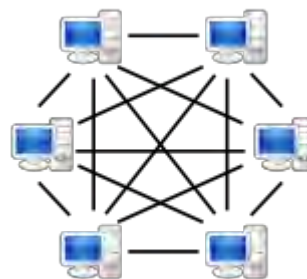


Figure 1: Peer-to-peer topology



Figure 2 : Client/server topology

In this report we explore performance of peer-to-peer file sharing. In section 2 we present how we construct queuing network model of our system and the classes of customers we have. Section 3 describes simulation with Java Modeling

Tool (JMT). In the last two chapters we show the result of our simulation and the conclusion of our work

2 Modeling peer-to-peer system

In this chapter we would like to explain how we model the system in queuing network theory. As it was previously said basic modeling approach is explained in [1] [2]. We have performed some changes of the model in order to create complete picture of the problem. The way we modeled the system is shown on the figure below (see figure 3). Peer has dual influence on the system. First of all it interacts with the system posing queries and performing some downloads. This behavior is modeled in the following form. Peers pose queries to the Indexing Server, which duty is to route queries to the destination file. In different architectures of peer to peer file sharing system such server organized in different way. We have chosen the easiest structure for our project "Centralized Indexing Architecture" (CIA). It is described in more details in [1] [2]. In such type of architecture there is one server that is responsible for routing and keeps information about all the locations of the files. That's why this architecture is named centralized (all the information in centralized in one place). From the indexing server queries are routed to one of the copy of the file in the system. For simplification we said that in our model there are 3 different types of files with ratio between numbers of copies of each of them equals to 5:2:3 for the first second and third files correspondently. Thus we see that capacity of the first file of the system is dominated from others. It is easy to see also that capacity depends on the number of copies (replications) that are proportional to the number of peers in the system. This fact introduces another property of the peers to increase the capacity of the system. Each type of the

file in the system is modeled as independent server and has as much service rate as much capacity it has. Thus we have 3 servers in our models correspondent to the file1, file2 and file3. After posing query peer can either continue to pose the queries or go to off-line or be invisible for the certain amount of time in the system. Moreover between 2 consecutive queries that peer sends to the system without to go to off-line state it might be various amount of time during that user is not active. Such time in [3] is named "think time", and such behavior of the system can be modeled as following. We have 2 delay stations that correspond to "on-line" and "off-line" state of the system. After query was processed there are certain probabilities that peer will go to one of those two states. Each of the station has different think time, this is obvious that user in off-line state has bigger delay than who stays on-line.

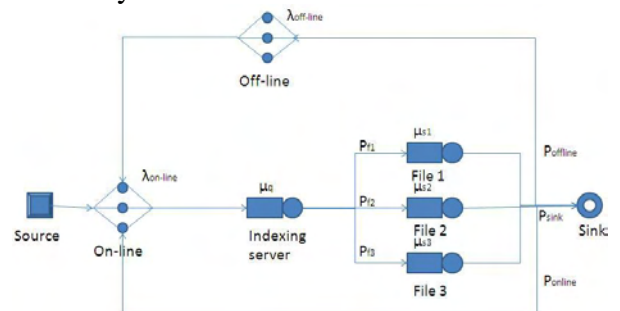


Figure 3 : System model in queuing network

2.1 Classes of user

We have decided that we have 2 closed model classes of users. We name them "freeloader" and "non freeloaders" as it is also defined in [1]. First class corresponds to the peers that only use resources of the system, others also share file that allows increasing capacity of the system. It is observed that ratio between first and second one in the population is 7:3. It was also mentioned in [1] that non free-loaders have shorter think time, and smaller probability to go off-line.

We have decided that to model peer to peer system with constant number of the population will not be complete, because

during life time its number varies. Such thing we are going to model in the following way. We add open class of user that represents variation of the system population. And we would like to know how it is affected behavior of the system. We said that those users that belong to the open classes don't increase the capacity of the system; they can stay for the certain amount of time in the system and their behavior is very similar to that of the freeloaders.

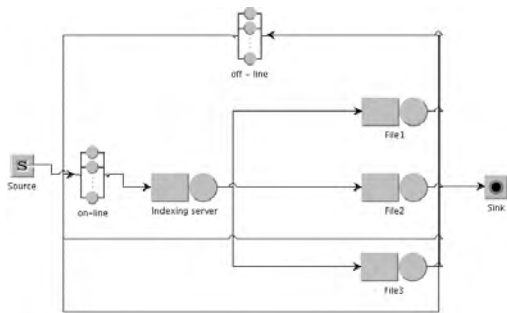


Figure 4 : JMT model of the system

3 Simulations with JMT

Here, it will be explained how we simulate our model with JMT [4] and how we parameterized our model. The model of the system in JMT is represented on figure 4. You can find all the components that were described in the previous chapter. Most of the parameters of the system were taken from the numerical experiment that is described in [1]. List of all the parameters used in the simulation is represented table 1.

Table 1 : Model Parameters

Symbol	Name	value
N_1	number of the population of the non free loaders	30
N_2	number of the population of the free loaders class	70
$\lambda_{on-line1}$	think time for on-line station for the non free loaders class	1/30
$\lambda_{on-line2}$	think time for on-line station for the free loaders class	1/300
$\lambda_{on-line3}$	think time for on-line station for the open model class	19/3000
$\lambda_{off-line1}$	think time for off-line station for the non free loaders class	1/43200
$\lambda_{off-line2}$	think time for off-line station	1/43200

	for the free loaders class	
$\lambda_{off-line3}$	think time for off-line station for the open model class	1/43200
μ_q	Service rate of the indexing server	100
μ_{s1}	Service rate of downloading of the file 1	0.0988
μ_{s2}	Service rate of downloading of the file 2	0.242
μ_{s3}	Service rate of downloading of the file 3	0.1613
P_{f1}	probability of that request is associated with first file type	0.5
P_{f2}	probability of that request is associated with second file type	0.2
P_{f3}	probability of that request is associated with third file type	0.3
P_{off1}	probability of going to off-line for the non free loader class	0.1
P_{off2}	probability of going to off-line for the free loader class	0.019
P_{off3}	probability of going to off-line for the open model class	0.181
P_{sink}	probability of leaving the system for the open model class	0.8

Now we will explain how we obtained those parameters in more details. We have chosen number of the population will follow ratio that was explained before (it was found that for each 3 of non free loaders in file sharing system there are 7 free loaders) and for the simplification we assume to have population of 30 of non freeloaders and 70 freeloaders. Thinking time for the closed classes was taken from the experiment [1]. Thinking time of the open model classes is assumed to repeat behavior of the freeloaders with 0.9 probability and non freeloaders with 0.1 probability.

$$\lambda_{on-line3} = 0.1 * \lambda_{on-line1} + 0.9 * \lambda_{on-line2} \\ = 1/3 * 1/10 + 1/3000 * 9/10 = 19/3000$$

The same way we compute $\lambda_{off-line3}$ and probability to go in off-line or on-line state for open model class. Service rate for the indexing server with CIA architecture equals to the capacity of the central server, and was taken from the same experiment. We supposed that popularity of routing queries to the files in the system follow

ratio 5:2:3 for the first, second and third files. Thus it is clear that we have 50% of request that are posed to the file one, 20% to the second and 30% to the third. From here probability P_{f1} , P_{f2} , P_{f3} that query is associated with file1, file2, file3 is equal to the 0.5, 0.2, 0.3 respectively. We suppose also for simplicity that our shuffle cluster of the system equal to 0. This means that file popularity and file replication fractions are equals for any file. We know that among 10 request 5 goes to file1, 2 to the file2 and 3 to the file 3. Thus the same holds for the replications of the file: among 10 copies of files 5 belong to the file 1, 2 to the file 2, 3 to the file 3. We use the same equations that are defined in [1] to derive service rate for each file server.

$$\mu = \frac{N1 H K}{i}$$

Where H is a constant, in our system we set it to 1/20. This corresponds to a file download time of approximately 150 sec. In our simulations we keep this constant the same for any file, so that we said that files size are more or less equals. K is normalization factor

$$K$$

Here j is number of requests for the file. With fraction of the request defined earlier $K=1/(1/5+1/2+1/3)=0,968$.

i is the number of copies of the file. With fractions for file's replication defined before: $i_{file1}=0.5*N1$, $i_{file2} = 0.2*N1$, $i_{file3}=0.3*N3$

Thus

$$\mu_{s1}=N1*H*K/0.5*N1=H*K/0.5$$

$$\mu_{s2}=N1*H*K/0.2*N1=H*K/0.2$$

$$\mu_{s3}=N1*H*K/0.3*N1=H*K/0.3$$

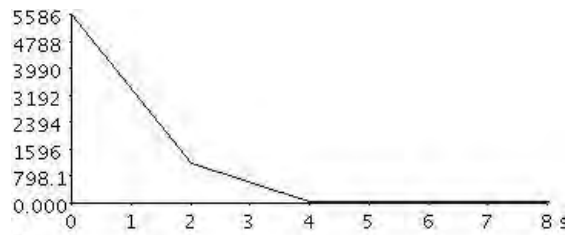
Probability to go off-line for the closed classes was also defined in the experiment. For the open class we have said that in the most cases after downloading customer leave the system (with 0.8 probability) With 0.181 it goes off-line and 0.019 to on-line state.

4 Simulation results

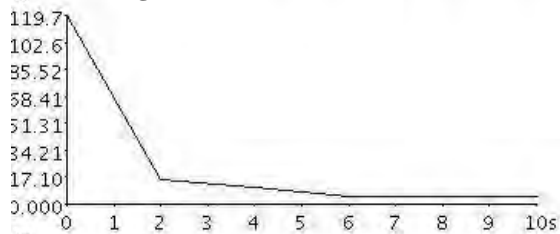
We have made 2 different experiments: first we monitored the performance of the free different file servers, second was to make "what if" analysis for the system exploring influence of variation of arrival rate of the open model customers.

4.1 Performance of the file servers

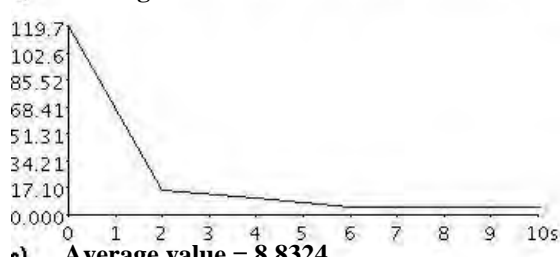
First we were looking at the performance of each file server by monitoring their queue length, utilization and response time. The results are represented on the figure 5, 6 and 7 correspondently. From those plots we can see that performance is scaled with probability of routing to the servers even if their capacity is bigger (remember, that the size of the files are equalized). As we can see also utilization of the server correspondent to the most popular file is almost 100% that means that this file is the bottleneck of the system. The most free resource is file 2, in spite of it has less copies. The reason is it has less probability of routing.



a) Average value = 53.2078



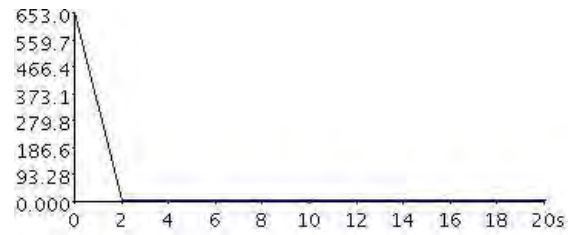
b) Average value = 4.7551



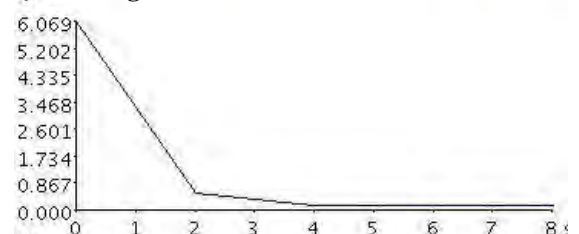
c) Average value = 8.8324

Figure 5 : Response time of the file servers

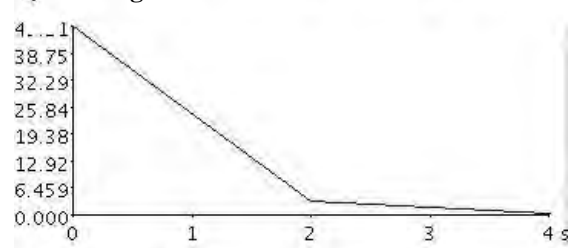
a) file1 b) file2 c) file3



a) Average value = 4.0119



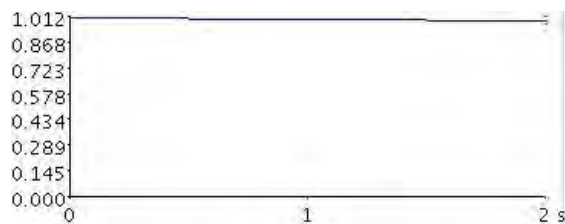
b) Average value = 0.1446



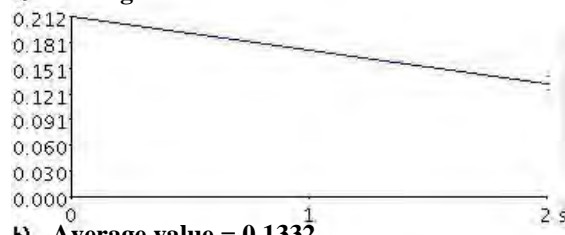
c) Average value = 0.4123

Figure 7 : Queue length of the file servers

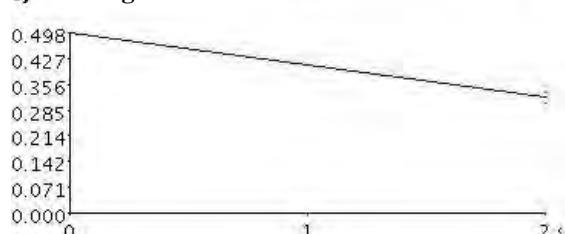
a) file1 b) file2 c) file3



a) Average value = 0.9894



b) Average value = 0.1332



c) Average value = 0.3192

Figure 6 : Utilization of the file servers

a) file1 b) file2 c) file3

4.2 "What if" analysis

Because we introduced additional open model class in order to represent variability of the population of the whole system. We decided that it could be interesting how this variability affects our system. We performed "what if" analysis in order to measure different performance indexes of the system depending on the different arrival rate of the open model class, and that means different variability of the population. We made simulations with JMT, record all the results and then plot it using Microsoft Excel.

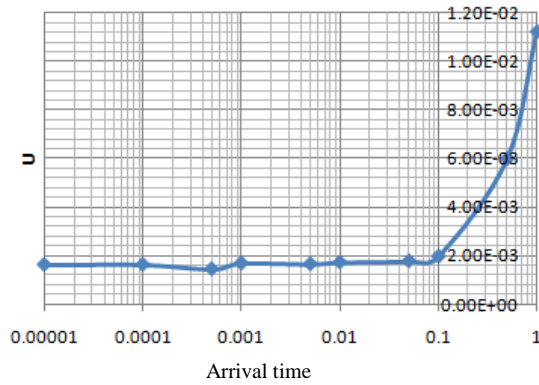


Figure 8 : Utilization of the indexing server

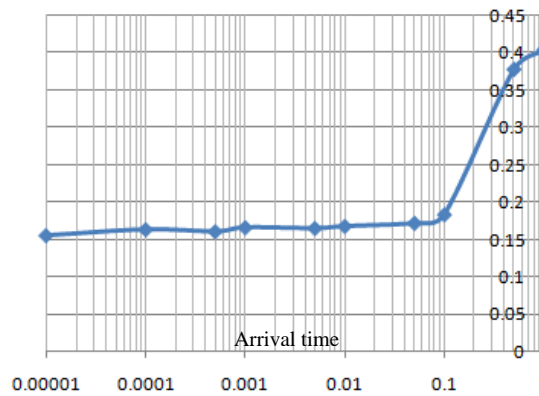


Figure 9 : Throughput of the system

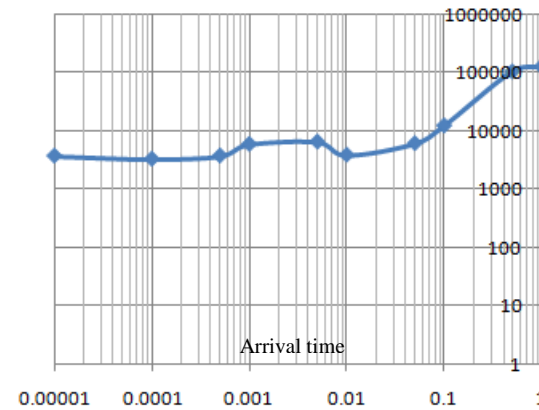


Figure 10 : Response time of the system

We first simulate the system without the open class (this is equivalent to arrival rate equal to 0)

We notice that the system response time (figure 10) and the system throughput (figure 9) increase as the arrival rate increase. But we notice that it increases tremendously after reaching an arrival rate of 0.1. It is good to increase the throughput of the system but the peers will suffer from a very long response time and that is not

good for the peers. Also that the increase is not linear, there is a point where the increase runs up very fast.

We can see in figures 8, 9 and 10 that performance indexes change is not monotonic. Look at the figures, where we plot response time analysis. We can find some discrepancies. For example for arrival rate of 0.001 the response time increases by factor of 2, and when returned to the previous values. As outcome of that we can tune the arrival rate to find the optimal performance point.

5 Conclusions

In this work we create a queuing network model for peer to peer file sharing system. We investigate the effect of changing the system capacity and work load on the performance indexes of the system using JMT.

6 References

- [1] [Modeling Peer to Peer file sharing systems](#), Zihui Ge, Daniel R. Figueiredo, Sharad Jaiswal, Jim Kurose, Don Towsley, IEEE INFOCOM 2003
- [2] [New Scheduling Policies for Multiclass Queuing Networks: Applications to Peer-to-Peer Systems](#), Ioannis. Ch. Paschalidis, Chang Sut. Michael. Proceeding of the 42nd IEEE Conference on Decision and Control Maui, Hawaii USA, December 2003
- [3] [Quantitative System Performance, Computer System Analysis Using Queuing Network Models](#), Edward D. Lazowska, John Zahorjan, G. Scott Graham, Kenneth C. Sevcik. Prentice-Hall, Inc., in 1984
- [4] Java modeling tools: <http://jmt.sourceforge.net>

7 – Protocols

7.1 – Modelling of BitTorrent Peer-to-Peer protocol	290
7.2 – Performance Evaluation of Tairona VoIP Server	310

An Empirical Performance Evaluation and Modelling of BitTorrent peer-to-peer File Sharing System using Queuing Network Models

Prabhat Saraswat & Prashant Batra

5th Feb 2007

Final version

Advanced Learning and Research Institute
ALaRI - USI
Lugano, Switzerland

Contents

1	A Brief Introduction to P2P File Sharing Systems	4
2	Queuing Network Models	4
3	The BitTorrent Protocol - A brief description	5
4	Experiment Methodology	7
4.1	Description of the log file structure	8
4.2	Text Scanner for the log files	10
5	Results and Analysis	10
6	Proposed Queuing Model	16
7	Analysis and Modelling using JMT	17
8	Conclusions	19

List of Figures

1	Graph showing the bandwidth usage of various files while downloading, X axis corresponds to the time, Y axis corresponds to the Instantaneous bandwidth for every piece . . .	11
2	Graph showing the bandwidth usage of various files while uploading, X axis corresponds to the time, Y axis corresponds to the Instantaneous bandwidth for every piece	12
3	Graph showing the bandwidth usage while downloading and seeding, notice that seeding continues well after the downloading stops, seeding - red, downloading - green	13
4	Graph showing the total data downloaded and seeded, green - downloaded data, red - seeded data	14
5	Implemented model in JMT	17
6	Simulation results for the sizefactor = 1.8	18
7	Simulation results for the sizefactor = 1.5	19
8	Simulation results for the sizefactor = 1.4	19
9	Simulation results for the sizefactor = 1.0	20

Abstract

It has been observed that the maximum bandwidth usage incurred during the file transfers on the internet mainly corresponds to the large chunks of data, which typically consists of continuous media, for example videos, movies or the sound files. Data packages like bulky software executables also contribute to the bandwidth usage.

Peer to peer file transfer systems have now become enormously popular on internet due to several advantages. One of the main advantages is the reduced overhead on one centralized entity as the file transfer process is distributed. However some of the traditional file transfer systems do not scale well with the number of clients.

BitTorrent is an improved peer-to-peer(P2P) file distribution protocol. Several free implementations of this protocol exist. This protocol has various mechanisms to award the peers who are actively participating in the file transfer process. These incentive mechanisms and various other mechanisms make BitTorrent scalable and robust. We would investigate these aspects in detail in the later chapters.

In this project we have tried to understand the BitTorrent protocol and have performed an experiment to understand the performance and incentive mechanism imbibed in the protocol. The characteristic graphs are studied to find out the patterns of similarity and relationships between seemingly disjoint parameters. Through the medium of empirical observation we have tried to characterize various parameters of the BitTorrent system so that it could be modelled using a queuing network model. We have also tried to find the asymptotic bounds on the performance. The study is further purported by the comparison between the simulation and experimental results mentioned towards the end.

1 A Brief Introduction to P2P File Sharing Systems

It has been an age old adage that *the sum is greater than its parts*, p2p model supports this point of view and relies primarily on the computing power and the bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. P2P networks are typically used for connecting nodes via largely ad hoc connections.

The motivation behind basing applications on peer-to-peer architectures derives to a large extent from their ability to function, scale and self-organize in the presence of a highly transient population of nodes, network and computer failures, without the need of a central server and the overhead of its administration. Such architectures typically have inherent characteristics like scalability, resistance to censorship, centralized control, and increased access to resources. Administration, maintenance, responsibility for the operation and even the notion of "ownership" of peer-to-peer systems are also distributed among the users, instead of being handled by a single company, institution or person. Finally peer-to-peer architectures have the potential to accelerate communication processes and reduce collaboration costs through the ad hoc administration of working groups (SADS02).

The popularity of peer-to-peer multimedia file sharing applications such as Gnutella and Napster has created a flurry of recent research activity into peer-to-peer architectures. However it should not be believed that all the file sharing protocols are equally resistant to the complications caused by the increased access of resources. Not all protocols scale well. A very important consideration in p2p file sharing systems is the presence of free riders. Due to the adhoc nature of the connections, it is possible for some of the peers to only download a file without themselves contributing files to the network. This hampers the performance of the network, as this leads to less number of copies of the file, which is not really desired. Protocols like BitTorrent provide various incentive mechanisms, that persuades the peers to share in order to get better performance from the network.

2 Queuing Network Models

It is often imperative to understand the behavior of certain systems under certain conditions and certain assumptions. As it is not always possible to perform live tests on the systems, we model the system as an abstraction of the important details, leaving out the mass of irrelevant details. Models are very helpful as when once they are made, they can be parameterized to reflect any of the alternatives under study. The parameters can be tuned to any

chosen value and the models can be simulated under pre-fixed assumptions. These assumptions can also be tuned to other values.

Modelling using the queuing networks is a typical approach to model various systems as a network of queues, each of which can be analyzed analytically. It is also referred to as the mathematical study of waiting lines (or more simply queues). The theory enables mathematical analysis of several related processes, including arriving at the (back of the) queue, waiting in the queue (essentially a storage process), and being served by the server(s) at the front of the queue. The theory permits the derivation and calculation of several performance measures including the average waiting time in the queue or the system, the expected number waiting or receiving service and the probability of encountering the system in certain states, such as empty, full, having an available server or having to wait a certain time to be served.¹

However, it should be noted that all the prior attempts to model the p2p file sharing systems, have modelled the p2p systems as a simple fluid model(QUSRI04), as it is considered to be very amenable for analysis. However, we are using the Java Modelling Tools designed at the performance evaluation lab at Politecnico di Milano(JMT07), which provides quite a lot of functionalities to design load dependent classes. This enables us to make a queuing network model for BitTorrent. Following sections would explain the model in detail.

3 The BitTorrent Protocol - A brief description

Before moving to the description of experiment and the methodology used, it would be better to discuss about the BitTorrent protocol. This would be a brief description as the complete protocol is very complex and long. A first hand overview would be provided which would enable the reader to be able to appreciate various decisions taken during the experimentation. It would also be helpful to understand and appreciate the results and the analysis presented in the next section.

BitTorrent is a P2P application whose goal is to facilitate fast downloads of popular files. We are trying to describe here how BitTorrent operates when a single file is downloaded by many users. Typically the number of simultaneous downloaders for popular files could be of the order of a few hundreds while the total number of downloaders during the lifetime of a file could be of the order of several tens or sometimes even hundreds of thousands. The basic idea in BitTorrent is to divide a single large file (typically a few 100 MBytes long) into pieces of size 256 KB each. The big files are

¹Source - Wikipedia

divided into much bigger pieces, typically of 1 MB. The set of peers attempting to download the file do so by connecting to several other peers simultaneously and download different pieces of the file from different peers. To facilitate this process, BitTorrent uses a centralized software called the tracker.

In a BitTorrent network, a peer that wants to download a file first connects to the tracker of the file. The tracker then returns a random list of peers that have the file. The downloader then establishes a connection to these other peers and finds out what pieces reside in each of the other peers. A downloader then requests pieces which it does not have from all the peers to which it is connected. But each peer is allowed to upload only to a fixed number (default is four) at a given time. Uploading is called unchoking in BitTorrent. Which peers to unchoke is determined by the current downloading rate from these peers, i.e., each peer uploads to the four peers that provide it with the best downloading rate even though it may have received requests from more than four downloaders. The downloading rate is strictly governed by the amount of data which has been uploaded until now. Thus in order to get a good bandwidth, one should contribute to the network by seeding.

This mechanism is intended to deter free-riding. BitTorrent distinguishes between two types of peers, namely downloaders and seeds. Downloaders are peers who only have a part (or none) of the file while seeds are peers who have all the pieces of the file but stay in the system to allow other peers to download from them. Thus, seeds only perform uploading while downloaders download pieces that they do not have and upload pieces that they have. Ideally, one would like an incentive mechanism to encourage seeds to stay in the system. BitTorrent currently has such a feature. In practice, a BitTorrent network is a very complicated system. There may be hundreds of peers in the system. Each peer may have different parts of the file. Each peer may also have different uploading/downloading bandwidth. Further, each peer only has partial information of the whole network and can only make decisions based on local information. In addition, BitTorrent has a protocol (called the rarest-first policy) to ensure a uniform distribution of pieces among the peers and protocols (call the endgame mode) to prevent users who have all but a few of the pieces from waiting too long to finish their download(QUSRI04).

We have tried to experiment with the varying load and tried to characterize this behavior of the system. The results and analysis are mentioned in the later sections. A queue based model is also explained later.

4 Experiment Methodology

Various BitTorrent clients are available on the internet to download the files from the .torrent files. Since BitTorrent is a p2p program, you actually start uploading as soon as your first chunk of data arrives. There are clients for various platforms, some of them are freewares while some of them charge small licensing fee for their usage.

After checking the log files of various clients, we found out that Azureus Java client(AZR07), enables the user to obtain detailed low level logs, down to the protocol level. This is what was required, so we chose to use this tool. The downloading and installing of the tool was easy. It is a java based tool, thus making it platform independent.

It allows the user to choose various parameters to be monitored. Various levels of logging are mentioned. We chose the debugging-expert user mode, which allowed the tool to dump all the information it could generate, in the log files. The next decision that we had to take was concerned with the files to download. This is an important criteria as BitTorrent fragments the files into pieces of different sizes according to their total file size. Their are typically two piece sizes, the most common fragment size is 256 K for the files of typically sizes around 700 -1400 MBs, while for the files of the order of 4 GBs or so, the fragment size is around 1 MB. We tried to download the torrent information files of two files of around 4 GB's (typically the DVD versions of movies). The other files were of mixed sizes. The log files allowed tracking of the data of each file separately. It also allowed us to evaluate the complete system response.

Thus we had 7 files to be downloaded. The files to be downloaded were:

1. File Name: Motorcycle.Diaries.DVD.Seeders.torrent
File Size: 4.36 GB (4681369600 bytes)
2. File Name: The Fifth Element HD-DVD.torrent
File Size: 4.33 GB (4644930112 bytes)
3. File Name: Forrest.Gump.1994.DVDRip.XviD.torrent
File Size: 1.36 GB (1463422408 bytes)
4. File Name: The.Amazing.Race.S10E01.PDTV.X.torrent
File Size: 685.8 M (719116288 bytes)
5. File Name: Pearl Jam - Complete Discography.torrent
File Size: 624.48 M (654815785 bytes)

6. File Name: Big Fish_Movie_English.torrent
File Size: 348.27 M (365183033 bytes)
7. File Name: Smashing Pumpkins - 1993 - Siamese Dream.torrent
File Size: 142.57 M (149495088 bytes)

The torrent files for all the aforementioned media files were downloaded before hand and the download for all the files was started simultaneously. The maximum size of the log file was kept as 250 MBs, as we had suspected that the process of downloading will take alot of time, some days typically. After one file of 250 MBs was made, the tool automatically copied it in the form of a backup file and started logging into another file in the same directory.

We kept the process running for 6 days, even though most of the files were completely downloaded much before, we wanted to observe the seeding behavior of BitTorrent. There was an inevitable and unprecedented shut-down of the machine after 3 days of download, which led to the redoing of the whole process again. However we were able to obtain two log files (one normal and the other backup containing old logs).

4.1 Description of the log file structure

The log file dumped by azureus when all the debugging options were turned on, can be seen below: (a part of it)

```
16:14:22.674 0 net      Received [BT_PIECE data for piece
#723:0->16383] message . . . . .
.TorrentDLM: 'Smashing Pumpkins'; Peer: L: 24.80.122.132: 5216
[Torrent 1.6.0]
```

Some of the fields are self explanatory. The first field is the time stamp, which precedes every message. The time stamp has a resolution of the order of 100th of the milisecs, that allows for the better accuracy in the calculations. The **Received [BT_PIECE** refers to the fact that a fragment is received. The number **16323** refers to the piece's sequence number, which allows for the recombination when all the the pieces are downloaded. The name of the torrent file is also mentioned. Following the name of the torrent, is peer's IP address which supplied that particular fragment. The BitTorrent client used by the peer is also mentioned.

There are few other messages which are also important. They are shown below:

```
16:14:23.034 0 net      Received [BT_REQUEST piece
#1093:458752->475135] message . . . . .
TorrentDLM: 'Smashing Pumpkins'; Peer: L: 24.253.30.246: 19906
[Azureus 2.5.0.2]
```

This indicates that the peer is requesting for the pieces with the sequence numbers in the given range, here the range is 458752->475135. If a piece is available with the client, it replies back with the BT_HAVE message as described below.

```
16:14:22.846 0 net      Sent [BT_HAVE piece #363] message . . . . .
. . . . .
TorrentDLM: 'Smashing Pumpkins'; Peer: L: 67.182.240.99: 60048
[Azureus 2.5.0.2]
```

This means that the client is answering the query of the peer and replies that it has a piece (with sequence number in the desired range) with it. Thus it can be downloaded from this client. As soon as the peer receives this message, it starts downloading the piece.

```
16:14:22.721 0 net      Sent [BT_REQUEST piece #580:81920->98303]
message . . . . .
TorrentDLM: 'Smashing Pumpkins'; Peer: L: 67.182.240.99: 60048
[Azureus 2.5.0.2]
```

This means that the client is requesting for the pieces with the sequence numbers in the given range from the Peer.

```
16:14:23.221 0 net      Received [BT_HAVE piece #458] message . . .
. . . . .
TorrentDLM: 'Smashing Pumpkins'; Peer: L: 24.239.217.250: 24960
[Torrent 1.6.0]
```

After a request for a piece is received, the peer replies using the BT_HAVE message indicating the identification number of the piece it is having and thus it is available for download. As soon as the client receives this message, it begins to download the piece.

4.2 Text Scanner for the log files

A text scanner using awk and sed was implemented to extract the relevant details from the log files. Since the log files size was huge, of the order of around half of GB, it was impossible to do the text analysis as the process itself was taking a lot of time. The log file was stripped of the useless details.

Since we had to investigate for the bandwidth performance and the incentive mechanisms, the packet and the timing information was of the critical importance to us. An awk script was written for calculating the instantaneous bandwidth consumed for downloading each piece. Another awk script was written to calculate the cumulative size of the data that was downloaded and seeded. A text scanner was written to differentiate between the data concerning with different load files.

We are not showing the awk files here, they could be found in the attached appendix for reference. A sample test file is also provided. The scripts to plot the data into graphs and then finally exporting them as an image file are also attached. We have used GNUPLOT in our analysis.²

5 Results and Analysis

The results from the aforementioned experiments are shown in Figure 1. Figure 1 shows very clearly the incentive mechanisms in the BitTorrent protocol. The graph shows the values of instantaneous bandwidth while the files are being downloaded. As it can be seen clearly, the top two lines correspond to the bulky files (refer to the file list mentioned in the previous section, the top two lines correspond to file 1 and file 2 respectively). One possible reason can be speculated for this kind of behavior, which could be an explanation for this spectacular high jump in the graphs. Each torrent is defined by a factor known as *torrent health*, which denotes the number of seeders present for that particular file. Whenever the health factor is high, more number of seeders are present to download from, thus leads to higher bandwidth. It should be noted that this is actually an arbitrary function of time and it changes quite randomly over time. It is related to the peers

²We encountered a problem due to the huge size of our data. We were not able to use the *open office* or *Microsoft EXCEL* to plot the graphs as the maximum samples allowed are 65000, while in our implementation the total samples were around 20 times of that. Thus we resorted to the tried and tested GNUPLOT.

shutting down and coming up, which obviously is unpredictable. Since the measurements were taken over a period of 6 days, the sudden changes, because of this, should not be visible in the graph.

Thus the torrent health factor (seemingly) is not responsible for the high bandwidths of downloads as seen in the graph. The only other factor which differentiates between the other cases, is the size of the fragments. Thus it is possible that the size of the fragment plays a role in the high speed downloads. We have defined a factor, **bandwidth factor** which corresponds to the high bandwidth of the downloads. This directly comes from the slope of the line. This could be put down as:

$$bFactor \propto SoF \quad (1)$$

where SoF refers to the size of fragment

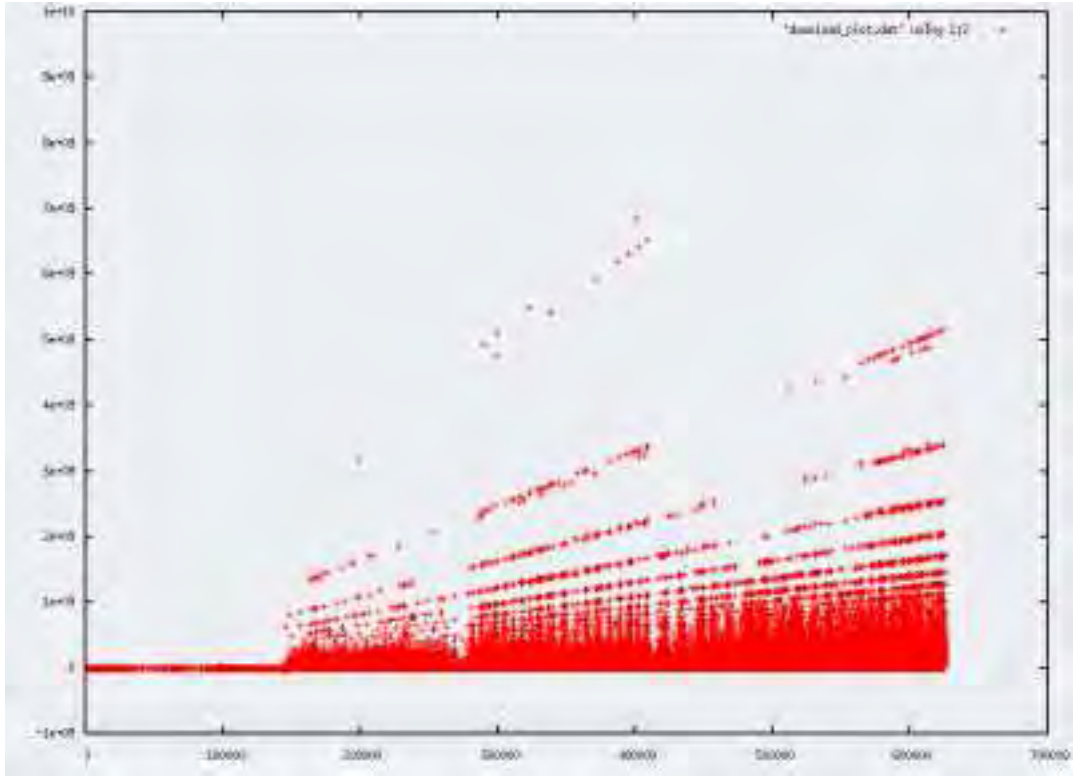


Figure 1: Graph showing the bandwidth usage of various files while downloading, X axis corresponds to the time, Y axis corresponds to the Instantaneous bandwidth for every piece

It should also be seen that there are some outliers which corresponds to the meteoric momentary increase due to increase in the numbers of peers and seeders. Thus the bandwidth is also dependent on the health factor. But this is momentary and it should be a weak function of the same. But for the sake of mathematical tractability we can write it as:

$$bFactor \propto SoF \times NofSeeders \quad (2)$$

where `NofSeeders` refers to the number of seeders present at that point of time

It would be also interesting to see the uploading statistics/graph. Figure 2 shows the instantaneous bandwidth values when the client is seeding the files it is downloading.

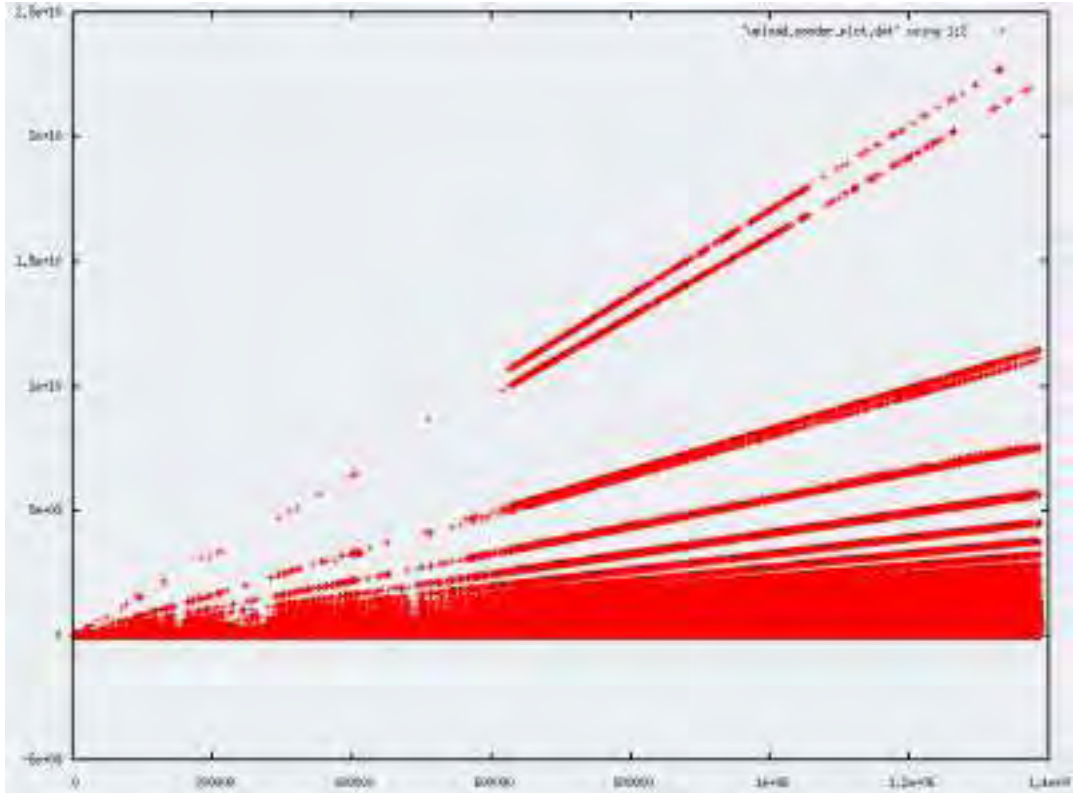


Figure 2: Graph showing the bandwidth usage of various files while uploading, X axis corresponds to the time, Y axis corresponds to the Instantaneous bandwidth for every piece

It can be clearly seen that the value of the slope of the lines is much more than in the upload curve. The only difference between the upload case

and the seeding case is that, in the seeding case one keeps on uploading even after the download has been finished, thus the size of the total uploaded data increases with time, linearly (as seen from the graph). There are not any other subtle differences between both of the graphs. The common regions in the graph can be shown when both of them are plotted together. Figure 3 elucidates this concept:

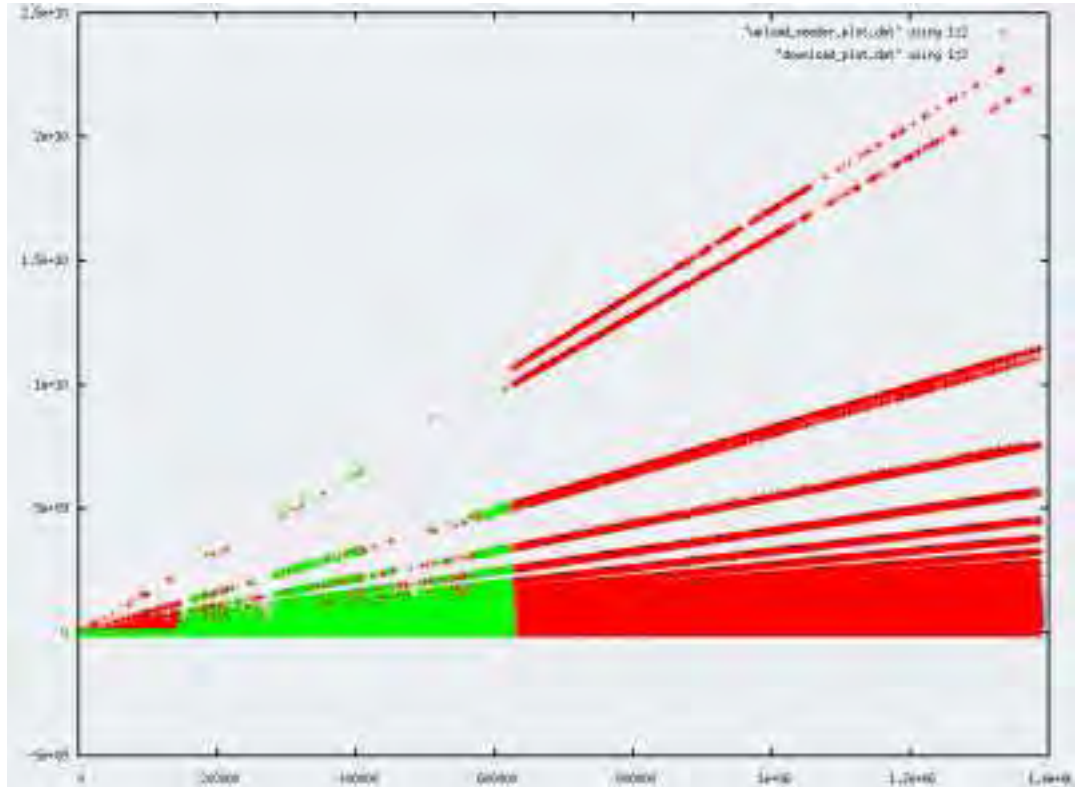


Figure 3: Graph showing the bandwidth usage while downloading and seeding, notice that seeding continues well after the downloading stops, seeding - red, downloading - green

We can also look at the plots of the cumulative growth of the size of data downloaded and uploaded by the peer. Figure 4 shows the downloaded and seeded data. The downloaded data is represented by the green worm and the seeded data is indicated by the red worm. The curve as shown in Figure 4 can be broken down into two linear segments. Thus it can be expressed in terms of a piecewise linear function. As it can be seen that until the data is being downloaded the slope of both the uploaded and seeded are the same. After the data is downloaded, the seeding rate increases as can be seen from the graph. This can be attributed to the fact that since there are multiple

downloads happening from the same seeder, the total size of files uploaded may quite well exceed the total size of files stored at the client (duplicate uploads).

If we try to correlate between the graphs of bandwidth and the cumulative uploaded size of data , it can be seen that the slope of the bandwidth is also a function of the amount of data uploaded. If the amount of data uploaded is more, one will get both good uploading speed and downloading speed. Thus the equation contains three parameters now. Equation 2 can be rewritten as.

$$bFactor \propto SoF \times NofSeeders \times dataSeeded \quad (3)$$

where dataSeeded is the total data seeded by the client

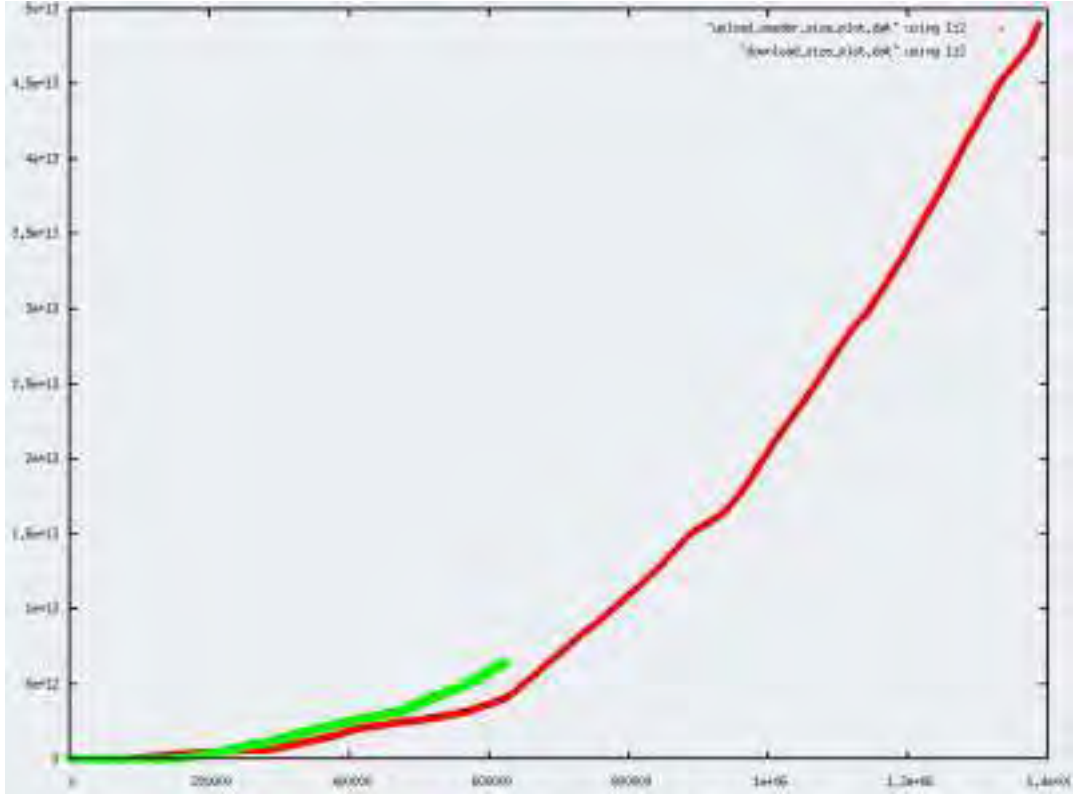


Figure 4: Graph showing the total data downloaded and seeded, green - downloaded data, red - seeded data

We also did measurements on the number of requests made by the client for the pieces of data. It was interesting to observe and as it is also consistent with the protocol, the order in which a BitTorrent client receives the

pieces. This totally depends upon the other peers, all the peers when they have finished downloading a piece they send this `BT_HAVE` message to all the peers, thus if another peer needs those pieces, they get it from those peers.

It was observed that for a large number of peers (as typical of bit torrent systems), the number of advertisements messages encountered by each client are random. However, we were not able to measure the number of peers who were downloading the same file due to unavailability of the required support from the log file description. However it is purported that there should be relation between the number of advertisement messages and number of peers downloading/uploading the file. Also an inverse relation is seen between the no of advertisement messages and the average downloading time of different peers.

This can be put in mathematical terms as:

$$noOfAdvts \propto \frac{NofPeers}{avgDownloadTime} \quad (4)$$

where

`noOfAdvts` - Number of Advertisements of pieces per second
`NofPeers` - Number of peers downloading or uploading the file
`avgDownloadTime` - Average time each peers take to download a piece.

It should be understood that the quantities `NofPeers` and `avgDownloadTime` are dynamic and change with time, also causing the `noOfAdvts` to change with time.

This quantity `noOfAdvts` is very important as it initiates the download process of various pieces. However, when the advertisement message for an already present piece is obtained, the message is discarded and the piece is not downloaded again.

Through the measurements from the log file, we tried to find an upper bound for the number of such advertisements for the files. The plot of the number of advertisements is not shown here, as the behaviour is very random. However we were able to find the bounds of the statistics using *MATLAB*.

The results are shown below. The granularity of the measurement is one second.

```
Min[noOfAdvts] = 0
Max[noOfAdvts] = 7
Avg[noOfAdvts] = 2.324
```

The measurements were done for an interval of one second.

On the basis of the aforementioned observations, we have tried to propose a queuing model. The details are explained in the next section.

6 Proposed Queuing Model

Since BitTorrent is a very complex protocol and it would be very difficult to model all the characteristics of the same, We have decided to model a very basic model for BitTorrent. The properties of BitTorrent that we would want to investigate are the download behavior and the incentive mechanism of the protocol.

We have defined various parameters to model a simple BitTorrent client. We are modelling the each BitTorrent client as a load dependent station. The throughput of which depends upon the number of advertisements made. Number of advertisements is modelled as the number of requests coming to the station. The station's service request time corresponds to the time taken to download a particular piece. The service time is corresponds to the inverse of the downloading bandwidth, thus it is inverse of the bandwidth factor.

Number of jobs in the queue represents the total number of pieces downloaded. The download of each piece represents the processing of a task. However it should be seen that the download times of the pieces depends upon the number of pieces already downloaded, since they are seeded simultaneously. Thus in our queuing model we have to keep the processing time of a single job as a function of number of jobs executed. It can be shown mathematically as:

$$serviceTime = f\left(\frac{1}{nofJobsExecuted}\right) \quad (5)$$

As discussed in the other section, we have calculated the minimum and maximum bounds on the number of advertisement messages. These bounds would correspond to the min-max bounds on the number of requests arriving on the station. Since it was impossible to find the distribution of the incoming advertisement messages. We were able to find out only the average. Thus we are keeping the incoming jobs distribution to be exponential.

The job classe type in our case is multiple class because it was observed from the discussions in the previous section that for large files the size of individual fragments to be downloaded are different. The size of pieces has a huge impact on the download and the upload bandwidth.

However, we encountered few problems while using the JMT tool to simulate our model. The problems are discussed in the next section. We have not been able to solve them.

7 Analysis and Modelling using JMT

As we have proposed a queuing model in the section above, We have implemented the model using the *Java Modelling Toolkit*(JMT07). We have resolved the problems by the solutions aforementioned in the section above.

A simplistic model implemented in JMT is shown below.

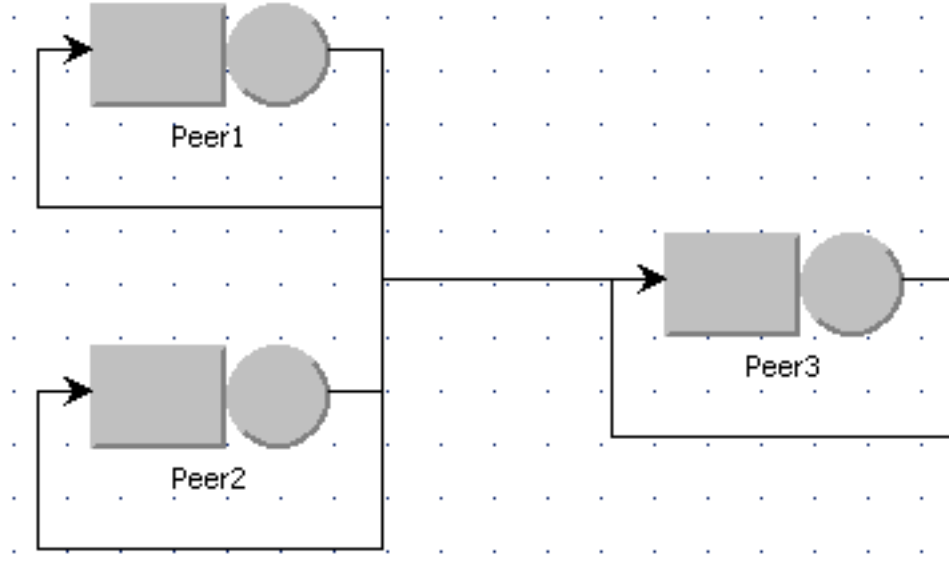


Figure 5: Implemented model in JMT

We have considered a network where a single peer is downloading data pieces from two different peers. The peers are identical, wrt to their configurations. The peers are load dependent stations. The load dependent service time distribution is taken to be exponential with the mean as:

$$mean = sizefactor \times n \quad (6)$$

The peers are designed as closed loop because of the incentive mechanism of the protocol. The feedback ensures that the no of jobs inside the station increases wrt. the number of jobs processed and coming out of the

station. We also have ensured that the service time is also dependent upon the number of jobs in the station. This ensures that the incentive mechanism is implemented well.

There are two producers, as represented by the peers 1 and peers 2. The input to the peer3 which we are monitoring for the download behavior is the combined output of peer1 and peer2. We run the whatif analysis simulation for various values of **sizefactor**. We chose the values of 1, 1.4, 1.5, 1.8 which is the simple ratio of the various file sizes used in the actual experiments. This has been done to correlate with the actual experimental results.

The graphs for the system response time obtained by various simulations are shown below. They have been normalized on the same scale. This has been done to see the similarities between the experimental and the simulation results. The graphs are shown below. It is interesting to observe how the response time changes wrt the various **sizefactor** values.

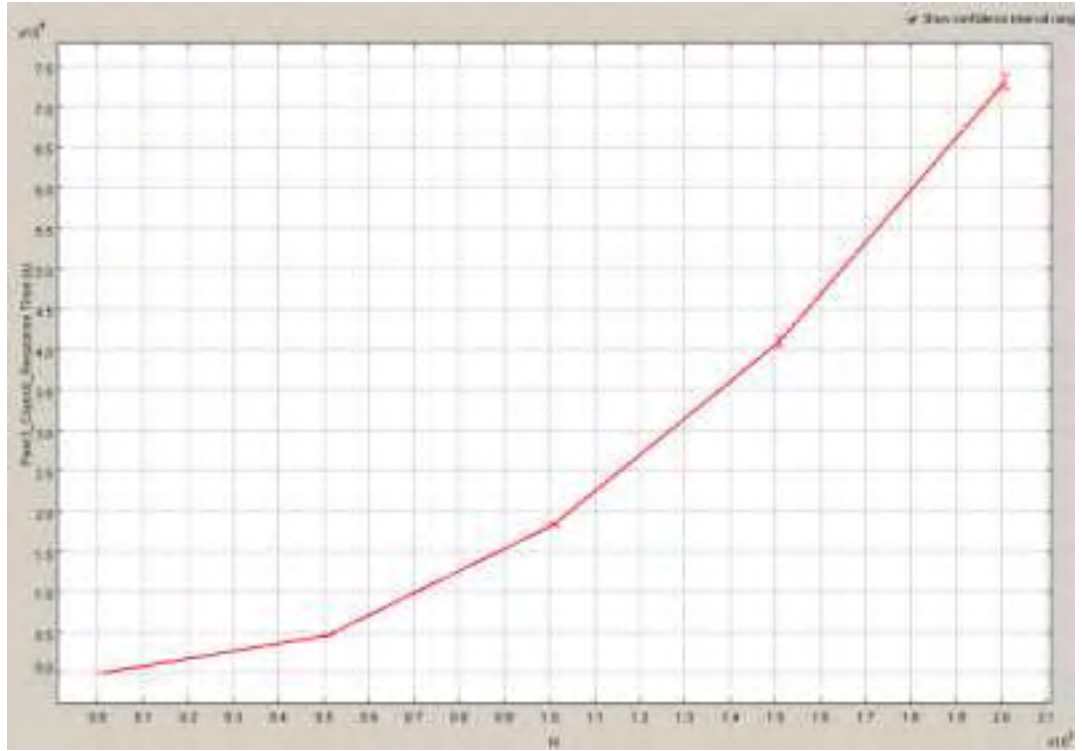


Figure 6: Simulation results for the sizefactor = 1.8

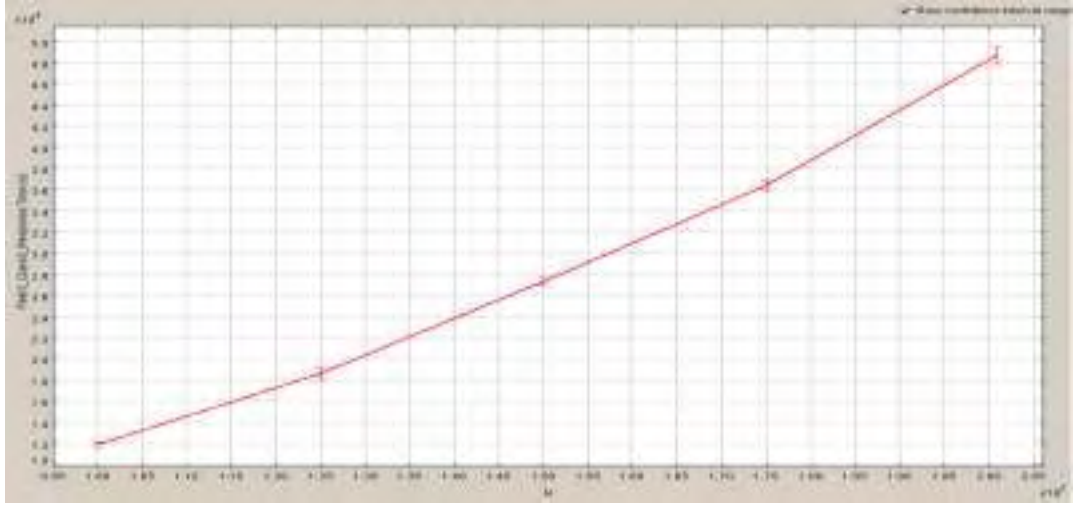


Figure 7: Simulation results for the sizefactor = 1.5

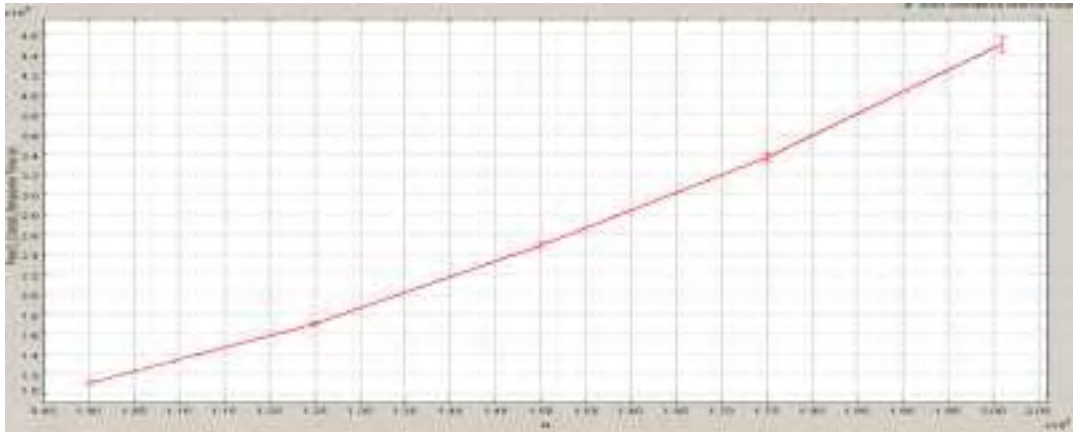


Figure 8: Simulation results for the sizefactor = 1.4

8 Conclusions

We have been able to perform effective experimentations and simulations with the BitTorrent peer-to-peer file sharing protocol. We were able to understand the functioning of the protocol and various incentive mechanisms imbibed in it.

We were able to find various performance bounds of the protocol and were able to reason how various parameters affect the performance of BitTorrent. Thus we have amassed enough knowledge to model certain char-

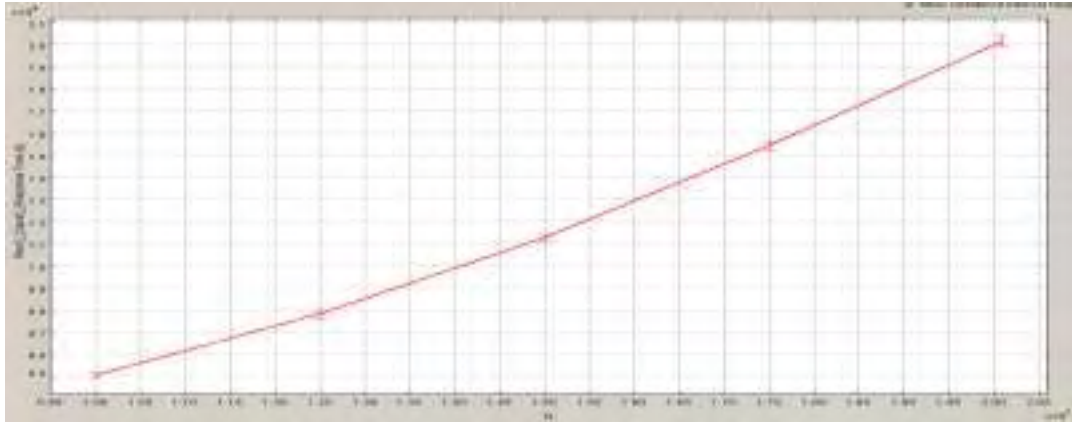


Figure 9: Simulation results for the sizefactor = 1.0

acteristics of the protocol. We have defined various parameters to define a queuing model. The implementation was done on Java Modelling Tool (JMT07), but was hampered by certain glitches. However we have been able to implement a simplistic model and the simulation results concur well with the experimental results.

This project has been a very good learning experience as it allowed us to understand intricacies of the peer to peer file sharing.

We now know what all happens behind a click.

References

- [SADS02] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, “A Survey of Peer-to-Peer File Sharing Technologies. White paper,” *Electronic Trading Research Unit (ELTRUN), Athens University of Economics and Business* December 2002.
- [QUSRI04] Qiu D. and Srikant R., “Modeling and performance analysis of BitTorrent-like peer-to-peer networks,” *In Proceedings of ACM Sigcomm* August 2004.
- [JMT07] M.Bertoli, G.Casale and G.Serazzi., “An Overview of the JMT Queueing Network Simulator,” *Technical Report, Politecnico di Milano* January 2007.
- [AZR07] HTTP Online Document: Azureus - Java BitTorrent Client “<http://azureus.sourceforge.net/>” as on Jan 2007

Performance Evaluation of Tairona VoIP Server

Marco Paolieri

Ivano Bonesana

February 6, 2007
ALaRI MSc 2005-2007

Contents

1	Introduction	2
1.1	Overview of Tairona	2
1.2	MjSIP	3
1.3	Goals	4
2	SIP Protocol	4
3	Server Traffic Analysis	6
3.1	SCI	6
3.2	EMA	7
3.2.1	Parser	8
3.2.2	Java Analyzer	8
3.2.3	Automatic Measurements Environment Scripts	8
4	Models	8
4.1	Performance Parameters	9
4.2	Measurements	9
4.3	Load Independent	11
4.4	Load Dependent Model	14
5	Measured Data	17
6	Conclusions	19
A	Source Code	22
A.1	Call Scripts	22
A.2	Parser Flex	23
A.3	Java SIP Analyzer	24
A.4	SIPServ Data Analyzer	32
B	JMT Screenshots	34

1 Introduction

This introduction aims to provide an overview of the system analyzed: *Tairona*. Section 2 describes the SIP protocol in order to allow the reader to understand how it works and which are the main information that allow us to model the system.

Section 3 presents our methodology and the scripts we used to perform our measurements. Data acquisition is fundamental because it allows us to develop a procedure to collect automatically a significant amount of data measured on the real working system.

Section 4 shows the results of our measurements and the simulation of the system we have built with JMT tool.

Section 5 collects the measured data of our experiment on the SIP server.

Section 6, the last one, reports our observations and some final comments on the results of this work.

1.1 Overview of Tairona

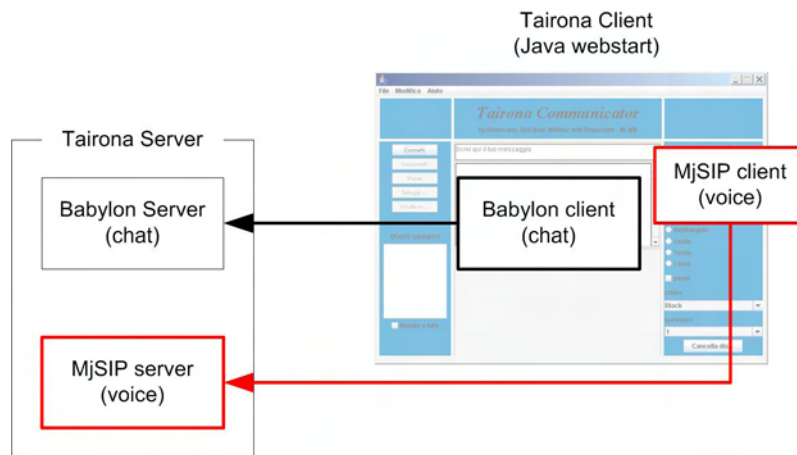


Figure 1: The Tairona system with client and server.

Tairona Communicator is a web-based platform for real time meeting and tutoring. It aims to provide a solution for face to face synchronous communication between the tutor and the students in remote faculties and similar environments where a live meeting is not possible. In particular the application is tailored on needs of a scenario that is very unique: in the considered institution in fact, teachers and students meet themselves only for the week necessary to complete the course. [1]

Figure 1 shows the components of the system. The client side is a java application launched on the user machine through webstart. It works as a completely stand-alone application.

Tairona supports chat, graphics exchange and VoIP communication between clients. As shown in the Figure 1, the server (but also the client) is based on

two projects:

- Babylon Java Chat [9] by Andy McLaughlin;
- MjSIP [8] by Luca Veltri, University of Parma.

The first one allows clients to be registered on the chat server, so that they can select users and interact with them by sending text and graphics elements. Chat in client side is located in the center of the window. The second one, marked in red in the figure, is the service that allows clients to use VoIP to talk. During this work we will focus our modeling on this component of Tairona. The Babylon Java Chat is part of another independent analysis.

Tairona uses the stack of MjSIP to support voice talks between users. Future developments will exploit MjSIP to establish multimedia sessions supporting chat (Babylon protocol will be removed), video-conferences and video-calls.

1.2 MjSIP

MjSIP [8] is an open source Java library developed by Luca Veltri at the department of Information Engineering of the University of Parma [11]. It has been used also by the department of Electronic Engineering of the University of Roma "Tor Vergata" [10] and is currently commercially exploited by PointerCom.

It implements a SIP¹ stack protocol fully compliant with RFC 3261, well used in VoIP and video communication through Internet.

The Java sources of MjSIP have been also ported successfully on cell phones with J2ME.

Since we didn't develop MjSIP, this project gave us the possibility to investigate the source code better understanding the way it works.

Starting from the small documentation available [7] and our source code analysis we could describe generically the structure. MjSIP has two distinct software: the client and the server. Both are based on the same SIP stack, but with different implementations. We will focus on the server side to evaluate its performances under real stress conditions, and to do so we will use the command line client provided with the framework. It has to be noticed that this command line application is fully compatible with the Tairona web graphical application. The server has a single thread architecture that processes requests and messages compliant to SIP protocol. In according to the protocol users have to register before they can call or talk, information about users are stored by the server in a file called "user.db".

The implementation support both UDP and TCP protocols. In the Tairona project we are always using the UDP datagrams, allowing the one-to-many connection (broadcasting). The transport protocol does not affect the system except for the network latency and packet loss. Since in this project we are not considering the QoS but only the performance of the server², we can safely ignore this characteristic of the system.

¹Session Initiation Protocol.

²Notice that when two clients are talking together using SIP, they are using a *peer-to-peer* topology that does not involve the server.

1.3 Goals

We want to investigate the performances of the MjSIP server with an increasing number of users. Our main goal is to describe the behavior of this system in order to evaluate how performances decrease under stress conditions. The method followed to achieve this target is the simulation of a queuing network model with the Java Modelling Tool [13] based on real values measured on a Tairona-test server under stress conditions.

Another important target is to develop a method to measure, acquire and analyze data of the server traffic. The procedure should be autonomous and configurable as much as possible so that we can repeat it at any time and under any condition.

It could be possible that the analysis will be adapted to the next version of the Tairona server, and the measures repeated to create a new model. Thus, this project should be reused in future works.

2 SIP Protocol

Nowadays Internet Telephony is gaining importance because it guarantees the possibility of low-cost calls. From a technology point of view it requires the establishment of a session between end-users; IETF exploited this by standardizing the Session Initiation Protocol (SIP) as RFC3261 [6]. The use of such protocol is not limited to Internet Telephony but it can be used for other applications like: instant messaging, multimedia conferences, etc.

As already specified in [7] the protocol can be transmitted over TCP as well as UDP; obviously being more reliable and increasing the transmission-load in the former case, unreliable but with a reduced transmission-load in the latter.

In order to understand how we measured the Tairona Server performance and how we built up the model in JMT tool, it is necessary to analyze the messages exchanged over SIP protocol in a classic VoIP call.

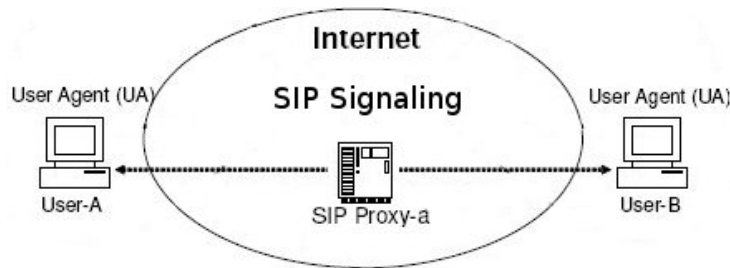


Figure 2: Example of a SIP scenario

Referring to the ISO/OSI protocol stack, SIP is an application-layer control protocol designed to establish multimedia sessions. In a typical scenario, as shown in Figure 2, clients are connected to a SIP-Proxy that is in charge of routing messages between clients. All standard SIP signaling messages are specified in RFC3261.

It has to be noticed that once the session has been established the communication between the caller (user A) and the callee (user B) does not involve SIP

signaling anymore, in fact VoIP phone calls or IM messages are not encapsulated into SIP protocol but a direct peer-to-peer connection is set up between A and B.

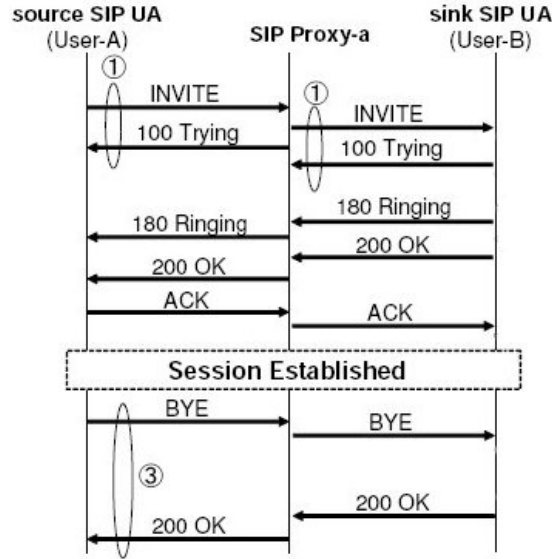


Figure 3: Messages exchanged establishing a SIP session

Sequential diagram of SIP signaling messages exchanged among the users in the classical topology of Figure 2, is shown in Figure 3. In case the user A wants to establish a session with the user B they exchange the following messages:

- The user A (caller) sends the INVITE packet containing the callee user-name *user B*, and not specifying the destination IP address, to the Proxy-a that looks for user B in the current registered users-database and then forwards the INVITE message to his IP address.
- The callee replies with a TRYING packet that is sent back to the proxy and forwarded then to the caller. This message represents the receipt of the invite call process.
- If the callee processing the INVITE considers it valid, it sends back to the caller (through the proxy) the RINGING packet, then he waits for user reply.
- When the user B answers the call sending an OK packet, the session is established.
- When the communication has to be stopped a BYE packet is sent by the user who wants to clear up the session.
- The receiver of BYE packet confirms the reception of such SIP signaling sending back an OK message.

A session can be established only when both users are registered on the server, so before all these messages can be exchanged it is necessary for a client to register

himself on a proxy sending a REGISTER packet. Collecting all packets of this type the server is able to build up a database to keep track of the usernames and the corresponding IP addresses. The server doing so can easily forward messages addressed to a certain username, not requiring that the user A knows the IP address of user B. Other information kept in the database are the live-time, after which an user is supposed to expire.

As already explained in the previous list a packet to go from the caller to the callee is sent to the proxy and then it is in charge of forwarding it to the final destination.

3 Server Traffic Analysis

To evaluate the performances of the SIP server we need to analyze the total time exploited by the server to process the packets used to establish a multimedia session (messages that have been described in the previous section, in the Figure 3). In order to achieve this goal we proceeded in two orthogonal ways:

- *SCI* (Source Code Instrumentation): we analyzed the Java source code of the server application and instrumented it, retrieving information about functions execution time.
- *EMA* (External Monitor Application): we used an external application to monitor the service time required by the server.

Both techniques have different advantages and disadvantages which will be described providing more technical details in the following subsections. All the source code that has been written for this project is attached to this report as Appendix A, obviously due to the big size the Java instrumented code of the server has not been included.

3.1 SCI

Before being able to instrument the server application we proceeded analyzing the source code to understand how it works. The application is composed of a single-thread and all the connections are processed sequentially: when a message reach the server it is dispatched by a method that calls the right function to elaborate it. It can distinguish between different messages, by recognizing them. We modified the source code in such a way that the time required for handling each of the packets composing the SIP signaling was printed on the standard output. Then, redirecting this output to a file, it is simple with a small parser to collect information about the service time of each message and, of course, of the whole transaction. The time is shown in nanoseconds and the difference between the time stamps taken at the entry point and at the end of the elaboration is computed. The current time stamp is taken using the method

```
public static long java.System.nanoTime()
```

that returns the current value of the system timer in nanoseconds.

This kind of approach has the advantage that loss of time due to the execution of other applications are less influencing the measurements as well as the time required to pass a packet from the lower layers of the ISO/OSI protocol stack

to the application one is not taken into account. The disadvantage is that all the time the software of the server is upgraded is necessary to patch the source code again and analyzing another equivalent application means to go through all the source code again instrumenting it.

3.2 EMA

The orthogonal way we proceeded was to use an external application to monitor the service time of the server. We used a TCP/UDP sniffer, directly interacting with the OS, to be able of analyzing and processing the network traffic without the need of modifying internally the server application.

The open source tool *tcpdump*³ captures all the packets addressed to the network interface. This tool has also the feature of filtering packets that match some filter conditions.

In our case we filtered the packets over UDP transport protocol and addressed to the port *5060* saving them on an output file. It contains a simply list of packets specifying the following fields: source and destination IP address, the time stamp and the ASCII content. Differently from the previous kind of analysis (where the time is intrinsically computed instrumenting the code) we had to develop a program able to process the file, filtering in subsequent steps the different types of SIP signaling and computing the time spent in processing a particular message on the server side by using the information provided by the time stamps of related packets.

Considering the advantages and disadvantages of such solution we can state that in comparison to the previous one this is much better because being an external program it works with upgraded version without the need of modifying the Java source code. It has to be noticed that, as it has been designed it could be used to monitor other SIP-compliant application. This is possible because it exploits the traffic filtering at the network layer and not directly inside the application. From this point of view it could be possible for instance to compare the performances of two different server application without even knowing how they are programmed and how they works. This could be very useful considering the case in which a server application has to be chosen rather than another one, and there is not enough time to spend for comparing the programming style.

The disadvantages that are present in this kind of approach are: the concurrent execution (with the Java server application) of the sniffer that is not computationally free for the PC, so this could lower the measured break down point of the server.

Moreover the measured time is accounting also the time spent for a packet to pass from lower layers (the measurements are exploited at network layer) to higher one (the application layer).

We developed a complete automatic measurements environment composed of several bash scripts that are in charge of executing different tests increasing the number of users who are concurrent making VoIP calls. Even if a bigger design effort wasn't necessary we developed a system that is much more scalable and let us test a lot of different scenarios just changing few parameters.

The software we developed in Java, the parser and all the scripts designed to be able to build a complete automatic measurements environment are described in

³<http://www.tcpdump.org>

the following subsection:

3.2.1 Parser

We built a parser realized in Flex in order to filter [15] the output of the Tcp-dump sniffer extracting only the information that are important for our analysis. Once compiled the flex (.lex) file we got a C source code ready to be compiled. The complete code of the Flex parser is shown in the appendix A.

3.2.2 Java Analyzer

One of the main tasks of the system analysis is exploited by the Java Analyzer. The description of how it is working is the following: it loads in memory, reading from a file, the output of the parser and then it processes the data coupling information about packets that are related to the same SIP signaling message. Consider the simple scenario in which two different clients that are trying to make two simultaneously calls. The server receives the packets from the clients in a wrong order because of the concurrency. It is then clear that to compute the time needed by the server to process the packets we need to isolate the information and look at the time stamps.

In a case in which the VoIP calls were managed in a sequential manner, then the task of this tool would be just to compute differences among subsequent entries. But considering that, different SIP signaling messages are processed sequentially we designed an algorithm based on the following information: source and destination IP, address and ASCII content of the packet. To know how to filter it we studied the standard looking at the reference RFC3261.

3.2.3 Automatic Measurements Environment Scripts

All the measurements are performed in a complete automatic way by several different bash scripts. All the source code is attached to this documents in Appendix A.

There are two main scripts, one of those is running on the server while the other on a client. On the client side the script is in charge of launching the client applications that should wait for the incoming calls and after that these have been setup, it executes the instances of clients that start to make VoIP calls simultaneously. On the server the script is waiting for a synchronization signal sent by the client-side and then it executes the server program in order to handle the VoIP calls.

Since the program could not be ran without the console to automatize the test and let several instances run concurrently we used a program called *screen* that is able to create virtual consoles.

4 Models

In our simulations we model the system in two different ways. The first one is a load independent model based on an average response time of the server, computed on the measurements of a single client request, the second one uses a load dependent model designed on the measurements reported in next subsection.

4.1 Performance Parameters

This section describes the parameters used to create the model of the Tairona SIP server. The system can be modeled as a single service center, characterized upon the models of [2] and [3]. We measured the service time of the server with an increasing number of concurrent SIP calls. It is possible to model this system as a closed model in which the number N of customers is equal to the number of concurrent calls for each measure.

In our measurements, we started from 10 clients connecting to the server, i.e. we started with a base number of jobs in the system of $N=10$. Then we performed a *what-if* analysis using JMT to evaluate the model.

We are interested in throughput (X), utilization (U) and response time (R) of the server in order to find in which conditions the response time becomes critical (e.g. server crashes). MjSIP server has a single thread architecture and it doesn't implement any queue structure other than what is provided by the Operating System and by the Java Virtual Machine. For these reasons we assumed an infinite queue size for the load independent model.

Throughput can be defined using the *Little's law* as

$$X = \frac{N}{R}$$

The *utilization law* define how to compute U

$$U = X \cdot S$$

where S is the service time of the station and it has to be computed using the measurements reported in next subsections. Details on the two models we present (load dependent and load independent) are reported in the next subsections.

4.2 Measurements

Using the tools we developed, and that have been described in the previous sections we measured the performance of the server in an experimental way before proceeding building the models and simulate them using JMT.

The measurements were structured in such a way we were automatically increasing the number of concurrent calls, monitoring the network traffic handled by the server. Processing the data collected we were able to account the time spent by Tairona server to manage the communication among different clients. The automatic measurements environment was built in such a way that a specified amount of clients were performing simultaneously calls and then, after a fixed amount of time, they started again; increasing the number of calls by a factor of 10.

The experimental results showed us that the crash-point of the server is identified with 110 calls where a dramatically increase of service time is measured and consequently the server is not able to accept any further incoming calls.

Even though a set of measures (i.e. from 10 to 110 concurrent calls), take almost an hour, since we had a complete automatic tool, we were able to easily perform several measurements in order to use more realistic values in the modeling phase (using the average among all the measured series).

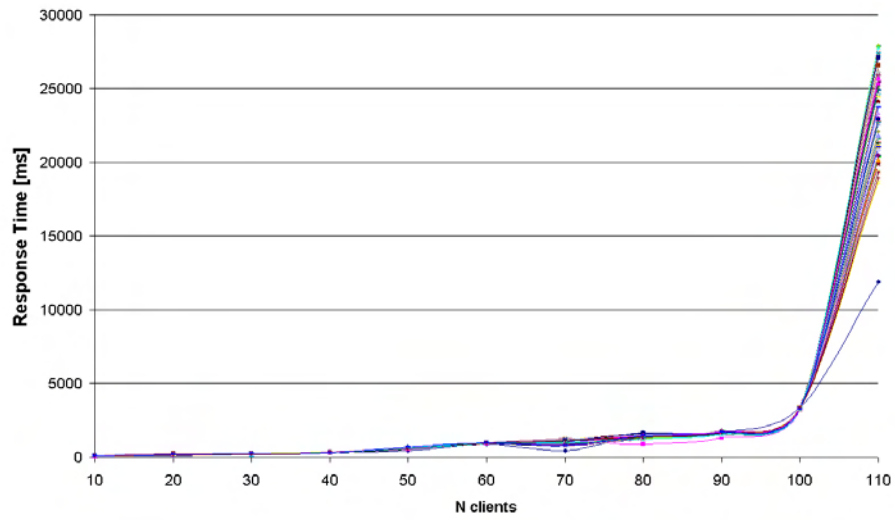


Figure 4: All data measured from the server.

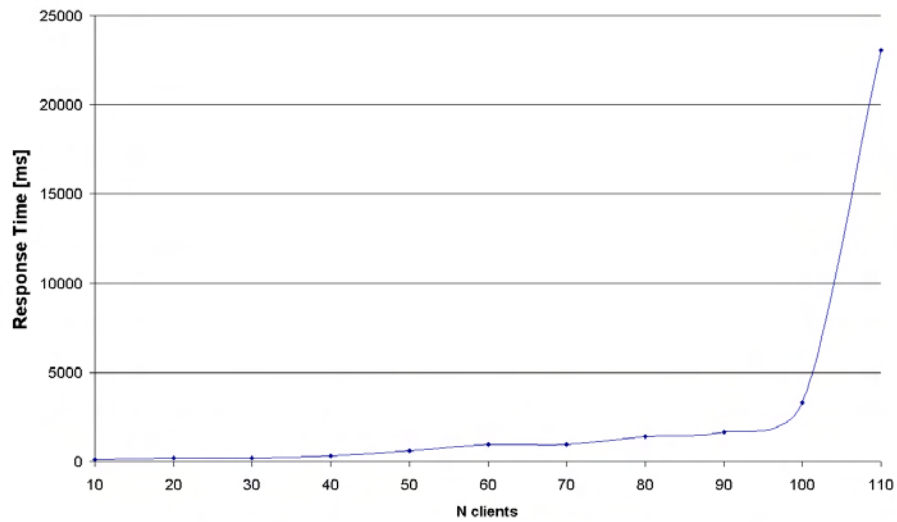


Figure 5: Average of the measured service time.

We scheduled around 60 sets of measures, processing the results we noticed that the measured values were congruous.

The overall pictures including the whole set of measurements series is shown in Figure 4 while the average function is shown in Figure 5 and it is build using the data listed in Table 1. As expected the trend of the total service time is following an *exponential*-curve.

We were able to achieve a low level of granularity measuring the time spent by the server processing each of the SIP signaling standard messages exchanged during a VoIP call. We summed up all the different values and collect the total time required to handle a VoIP call. An example of data collected from a set of measurements is shown in the Section 5. It is clear that different kinds of packets require different amount of computation time to be processed but it is important to conduct a quantitative analysis in order to determine where to focus possible optimizations of the software code and eventually change protocol policies. A result of this is shown in the Figure 6, where a pie-chart is used in order to emphasize the percentage of time required by different SIP messages.

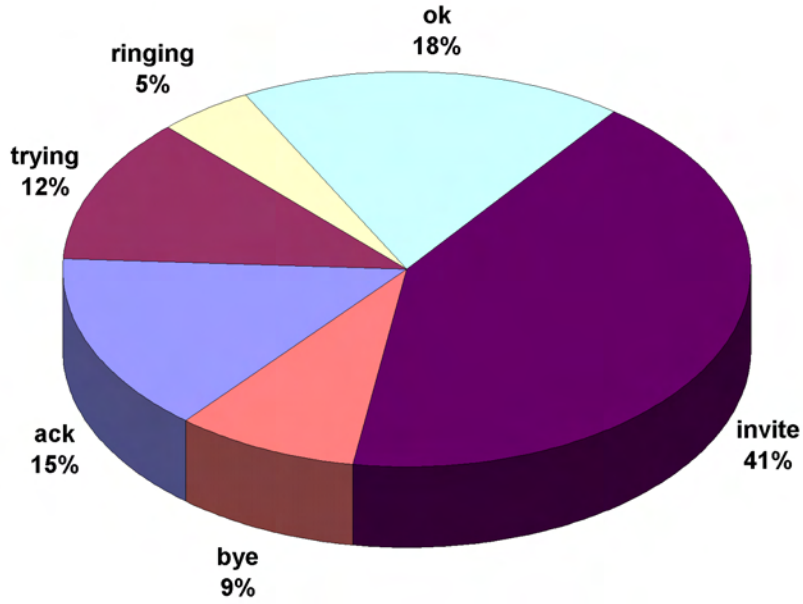


Figure 6: Different percentage of computation time required by different packets of SIP signaling.

4.3 Load Independent

Let us assume that server has a fixed mean service time for any arrival rate. We can compute this value by taking the value measured for 10 client, i.e. when the server is working well, and take the average (obviously over all the series,

i.e. 60 measurements). We assume a value of

$$S = 0.8863674s$$

to set-up a call between two users.

To simulate the system using JMT, we run a *what-if* analysis based on an increasing number of customers. Starting from 10 job, arriving to 110 jobs by 100 sampling steps.

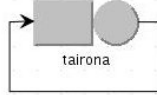


Figure 7: Network queue model for Load Independent model.

The independent model was the first model we created, starting from the assumption that we wanted to study the behavior of the system considering a single station with queue and a service time based on values computed from the experiments.

The model as in the case of the load dependent one, was closed, because it is then possible to model the concurrency of the incoming calls and perform a what-if analysis to collect the trend of the chosen performance indexes.

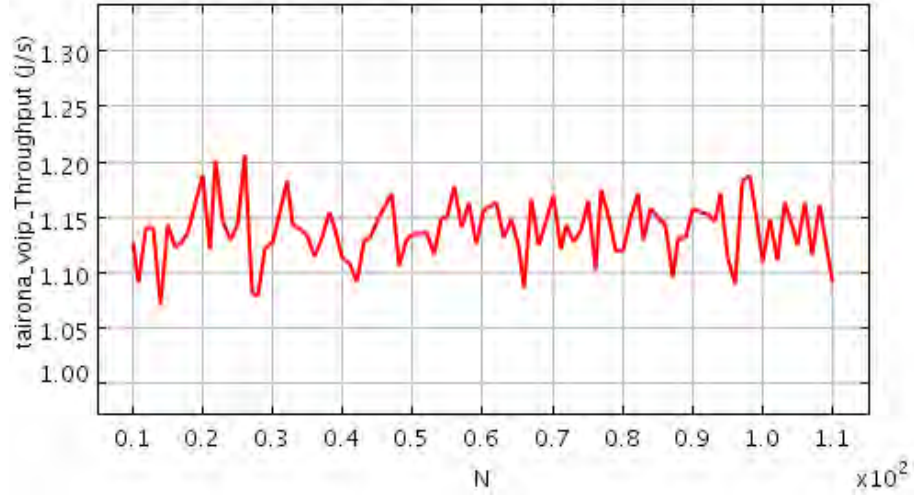


Figure 8: Throughput of server plotted by JMT with Load Independent model.

The overall picture of the model is Figure 7, as we could expect, the response time has a linear trend (please refer to in Figure 9). It is so because of not load dependent characteristics of the service time, moreover the queue was considered to have an infinite length. Even though this result was expected without simulation, it is important to underline and notice how parallelism in the management of different incoming calls helps reducing the user-perceived service

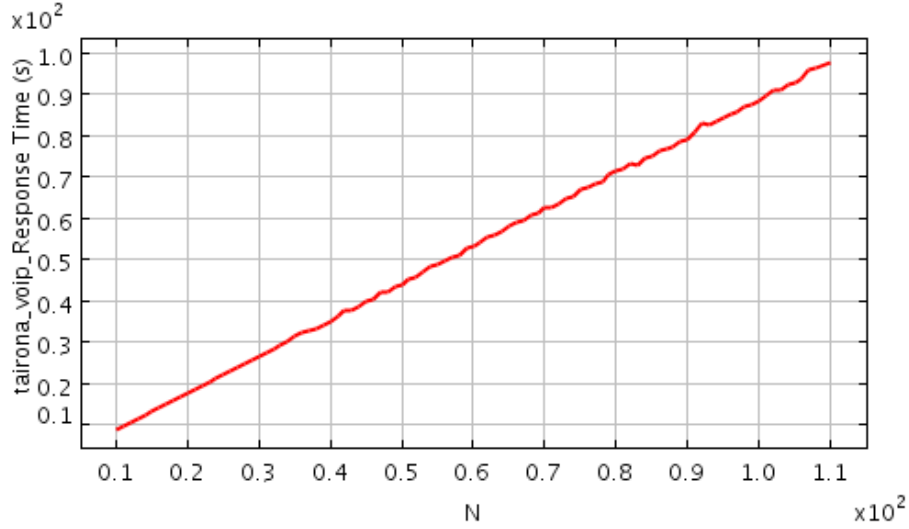


Figure 9: Response time of server plotted by JMT with Load Independent model.

time.

In a simple scenario where user A is calling user B, between processing two subsequent messages (e.g. the INVITE and RINGING messages) the server can start to handle other call messages. Our model does not take into account this peculiarity of the server application but as if all the calls were treated in a pure sequential way.

A more realistic model is proposed by [2] and represented in the network queuing diagram of Figure 10: a call is described as a sequence of processes each one representing a message (or a set of message) identified in the image by their standard [6] names⁴. Moreover this open model describes also the probability for each message to produce a positive answer or an error. The service time is parametrized on the *invite* message processing time, called μ . This model allows some kind of parallelism: a message has to be processed by each queue to establish a SIP call. Now, let us suppose that an invite message is processed by the first station and so it produces a 200-OK message that enters in the second queue. In the meantime another invite message can enter in the first station. It is clear that, in order to better describe the parallelism of the system we should be able to measure some characteristics of the server such as disk accesses (used in invite execution).

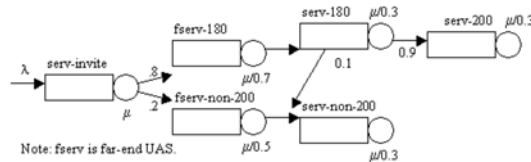


Figure 10: Model of a SIP server without network delay proposed in [2].

⁴ringing is “180-Ringing”, ok is “200-OK”, ... please refer to [6] and [2].

Overall observations can be referred to Figure 15 where the comparison of the two model is shown. At the crash-point in real experiments the server starts to increase the latency of a session establishment respect to this model where the calls are processed sequentially (avoiding parallelism). It is then clear that the processor is not able to manage this workload.

Table 1: Average data computed on the whole sets of measurements)

clients	avg	avg/call
10	104.5433	10.44488
20	195.3769	9.848709
30	220.345	7.360848
40	311.5608	7.800255
50	613.1075	12.31878
60	959.3244	16.00799
70	983.01	14.1743
80	1401.079	17.49014
90	1632.022	18.12077
100	3307.473	33.07164
110	23054.87	211.226

4.4 Load Dependent Model

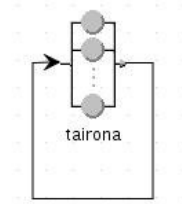


Figure 11: Network Queue model of server plotted by JMT using a load dependent delay station.

In the case of load dependent we designed a model as the one of Figure 11 where the server is represented as a Load Dependent Delay because the actual data we have measured are representing not only the service time of the server - as in the case of independent one - but the total response time R (i.e. queue time summed with service time). In according to these observations it would not be correct to model it as a queue station.

The typology of the model we chose was a closed one, because it is best to represent the concurrency of the incoming calls (i.e. since in the tool there is not the possibility of specifying incoming burst calls, it was the consequent solution to satisfy our requirements). Obviously using a closed-model it is not possible to assume a certain distribution of incoming requests, so what we used to analyze our server was a *what-if* analysis varying the number of incoming requests.

We loaded on the model the values corresponding to average time required to process a call with different numbers of customers from 10 to 110. This is shown

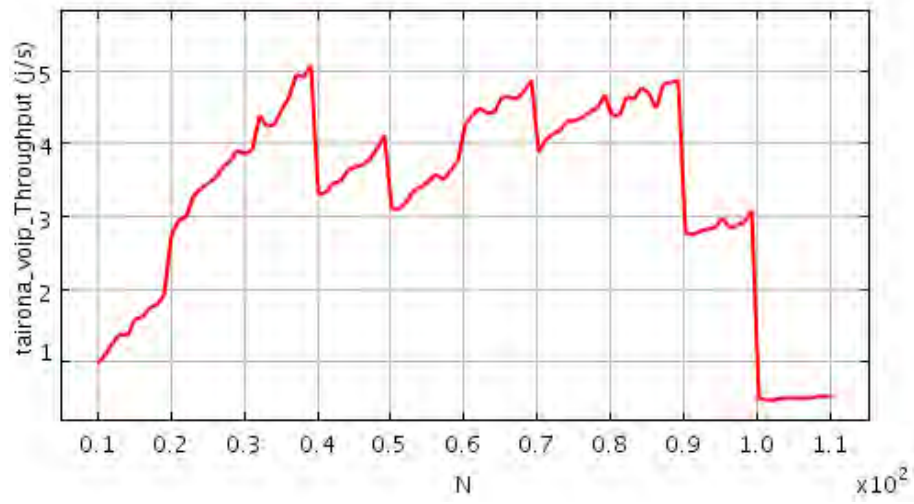


Figure 12: Throughput of the server in load dependent model

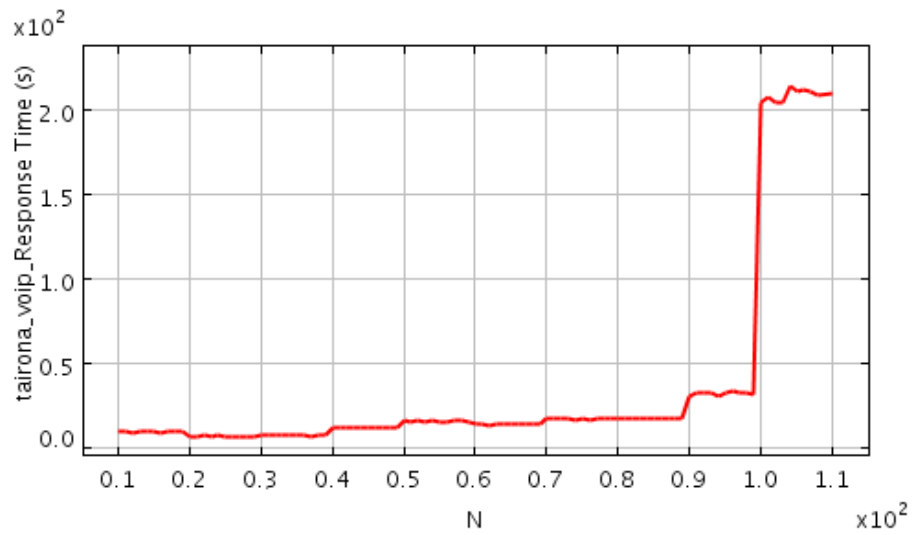


Figure 13: Response time in case of Load Dependent Model.

in the corresponding window of the tool, Figure 14.

Once we selected our performance indexes we were able to carry out the performance analysis on the server. For our goals we chose the Throughput X, and Response time R. What we got as the result is a response time that is closed to the experimental measurements that have been carried out. This is described by the Figure 15 where a complete comparison among the experimental, load dependent and load independent data is shown.

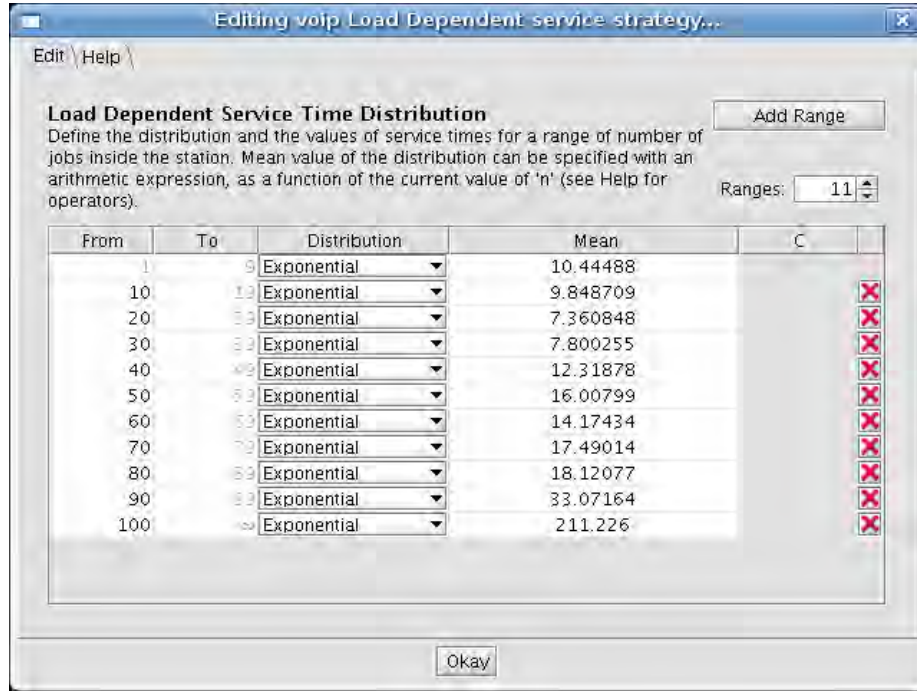


Figure 14: Load Dependent Classes.

The throughput of the system is sketched in Figure 12 while for the response time refer to the Figure 13. From the result of this analysis we can observe that the throughput increases for the first measures (10, 20 clients) and then it reaches some kind of stability around to 4 jobs/s. We notice also some variations between 3 jobs/s and 5 jobs/s. The same happens with the response time with a variation of 5 ms from measures at 40 clients and 60 clients.

A progressive degradation of the response time starts after 50 clients, although the throughput does not seem to get extremely worse. The crash of the system is visible at the end of the simulation with the collapse of the throughput, which arrives to values under 1 job/s, and the explosion of the response time.

In terms of workload we can say that also the registrations of clients can affect the measurements, although we have tried to limit as much as possible their effect by using delays between the two kinds of requests. In fact, the different behavior of the server with a different number of client registrations can suggest a possible explanation for the variations of the throughput.

5 Measured Data

clients	meas 1	meas 2	meas 3	meas 4	meas 5	meas 6	meas 7	meas 8
10	110.4	74.67218	113.0501	115.4991	116.3286	103.1635	106.1888	108.2843
20	96.34717	212.3843	200.8816	228.3345	224.1001	155.9871	177.4995	169.892
30	190.556	224.9111	218.8307	221.4251	217.726	230.6322	223.2188	221.486
40	283.698	316.9695	304.4281	308.6199	304.966	321.8973	318.58	323.5611
50	437.56	645.691	573.4215	545.0339	638.4256	670.1203	677.4926	672.3742
60	887.7038	972.8739	977.9167	934.0315	963.092	963.1704	949.9783	932.6076
70	413.1586	1122.361	955.3392	1149.033	795.7074	1037.211	837.1495	836.6302
80	1516.919	862.5897	1477.918	1589.158	1216.48	1420.24	1455.131	1265.801
90	1703.481	1313.178	1711.465	1726.616	1770.057	1733.911	1657.495	1588.444
100	3326.649	3197.926	3303.943	3340.425	3274.701	3320.67	3321.898	3270.889
110	11895.67	25253.32	24560.28	19566.28	22780.56	24081.35	19740.57	24901.54

clients	meas 9	meas 10	meas 11	meas 12	meas 13	meas 14	meas 15	meas 16
10	98.50408	98.43562	108.2271	92.72369	96.04245	99.84576	112.8527	96.91904
20	205.0943	165.3653	219.253	223.5976	226.1161	180.9466	218.8782	235.2719
30	208.831	229.8707	226.3	212.6701	231.4384	221.8572	209.8604	208.7393
40	308.081	305.8792	303.5922	310.3795	317.8322	319.6501	317.0006	315.9967
50	589.9387	577.6773	544.1794	635.6895	583.519	666.722	563.2809	588.2278
60	950.7927	981.622	948.5217	960.9545	988.4754	984.4116	942.2538	952.6936
70	1253.319	965.2223	923.8401	1210.233	1068.693	1107.186	1029.001	944.4614
80	1375.485	1462.727	1375.689	1615.308	1248.61	1421.863	1199.93	1284.455
90	1531.525	1677.619	1617.961	1634.874	1589.307	1726.659	1637.724	1709.729
100	3295.316	3345.654	3291.782	3325.133	3281.812	3345.043	3305.577	3268.866
110	27441.41	26487.49	27774.42	19442.4	24608.42	20040.46	23248.29	27290.56

clients	meas 17	meas 18	meas 19	meas 20	meas 21	meas 22	meas 23	meas 24
10	94.01212	92.75118	113.4297	104.7904	92.85312	104.0294	96.95444	113.2414
20	172.2593	234.0146	213.7327	162.5034	174.4294	212.9961	231.2259	231.2034
30	227.2957	225.9651	228.1	230.0167	227.5443	214.6129	216.6891	225.6698
40	314.4324	314.2599	314.0359	302.523	302.5109	301.7821	319.6866	322.2049
50	552.9955	682.0175	666.696	602.2102	563.6143	625.4555	594.0001	581.3167
60	980.7292	973.1266	985.2634	937.3938	980.882	967.4744	942.0529	967.091
70	1134.05	1085.93	890.2402	1201.206	1175.029	849.0668	1030.938	912.8275
80	1348.824	1496.998	1587.299	1473.842	1343.764	1306.84	1418.726	1386.409
90	1512.729	1718.515	1693.234	1519.021	1684.776	1530.68	1655.885	1782.344
100	3266.576	3352.346	3303.655	3331.579	3322.403	3321.09	3350.383	3315.096
110	27870.16	24610.18	27869.79	21018.24	20168.94	19342.38	24617.66	25774.9

clients	meas 25	meas 26	meas 27	meas 28	meas 29	meas 30	meas 31	meas 32
10	99.84643	94.15967	101.8981	108.2311	106.2143	105.7981	110.9952	95.22515
20	174.6209	206.5023	208.9266	222.7963	233.5694	190.0874	227.4863	181.393
30	218.464	214.6953	224.7525	221.9124	217.6544	207.8638	228.5556	210.2403
40	304.6375	318.7971	315.1746	310.0965	314.4108	313.6212	323.5468	308.532
50	671.2202	589.1713	546.8795	662.5348	648.7509	674.0736	649.7374	649.4021
60	970.618	979.9816	977.6376	963.3296	947.6565	955.6882	941.3395	946.9926
70	874.6439	981.7546	807.5163	1169.493	1134.694	886.775	768.8217	771.9412
80	1396.677	1382.07	1350.325	1430.863	1209.772	1245.143	1610.723	1565.624
90	1518.7	1568.1	1546.529	1653.709	1663.134	1771.44	1518.618	1591.609
100	3284.256	3319.246	3347.139	3316.424	3308.931	3265.538	3352.237	3319.504
110	18935.25	21623.29	21288.76	22836.59	26558.49	26081.11	22607.02	19901.61

clients	meas 33	meas 34	meas 35	meas 36	meas 37	meas 38	meas 39	meas 40
10	106.8761	113.2305	94.10254	96.59198	115.6225	115.9665	108.7624	95.44687
20	172.4206	176.875	190.1422	191.5284	184.4453	201.0867	190.4779	205.8144
30	223.5965	213.7234	227.0408	225.3842	231.5582	231.8185	213.6745	208.3148
40	312.7194	312.358	316.1034	303.7524	306.9604	311.0108	307.1733	308.028
50	594.3834	666.375	687.6807	654.1791	561.93	566.5811	622.5273	681.5601
60	939.7901	967.444	944.7784	934.5694	945.8009	979.0918	971.8096	939.5407
70	817.9467	802.7115	1120.818	1014.895	820.2447	804.7509	1090.451	1161.61
80	1433.092	1363.016	1611.344	1650.561	1385.186	1420.584	1588.538	1611.353
90	1704.652	1609.421	1661.688	1645.373	1761.06	1689.988	1654.126	1613.505
100	3337.408	3323.811	3310.48	3272.791	3314.115	3297.321	3294.658	3348.482
110	19683.31	19890.13	20331.65	21032.93	22783.32	22922.26	27340.52	27414.34

clients	meas 41	meas 42	meas 43	meas 44	meas 45	meas 46	meas 47	meas 48
10	111.7435	117.4319	100.2985	112.9966	95.68382	114.5575	115.4025	102.9793
20	181.062	203.6658	166.2939	231.9972	175.4405	214.67	178.4876	192.2862
30	212.3924	224.5931	211.471	212.3851	225.1962	232.0039	211.9372	223.5222
40	303.4891	309.7392	311.0651	322.7267	300.6001	309.2541	322.8402	309.2264
50	589.3473	657.1184	582.5026	544.385	594.274	622.0844	599.5488	622.3902
60	987.3128	949.5479	981.9486	963.3478	945.6636	955.8904	939.2816	971.6484
70	951.3233	824.6087	1037.785	956.328	830.0525	843.0062	1186.062	883.9125
80	1279.961	1631.1	1348.571	1331.728	1547.274	1204.791	1366.686	1564.006
90	1752.839	1539.208	1513.233	1700.97	1601.319	1570.653	1509.645	1515.637
100	3309.139	3302.331	3341.188	3309.725	3321.48	3265.298	3271.067	3315.756
110	25105.93	22936.15	22068.04	25439.41	20435.07	26111.39	27150.49	21856.85

clients	meas 49	meas 50	meas 51	meas 52	meas 53	meas 54	meas 55	meas 56
10	116.917	96.82751	104.5459	116.7071	106.3654	99.59541	104.7041	103.8946
20	160.7321	173.4994	187.1127	223.3145	235.6912	187.4098	187.1332	178.4718
30	212.8234	230.1731	208.9239	218.5594	229.0911	222.5469	230.5338	228.9231
40	309.603	314.399	315.713	305.0184	318.3366	309.0484	302.3562	311.0429
50	659.5508	630.0251	584.9477	569.6692	633.0445	656.571	652.233	581.3384
60	982.387	932.8207	935.7172	955.9883	937.0365	964.7517	941.6917	973.7311
70	1240.718	1073.684	979.588	1092.549	1204.104	1046.132	1194.461	1126.276
80	1273.513	1232.765	1261.898	1449.262	1465.653	1423.544	1515.939	1608.31
90	1573.295	1509.478	1528.913	1692.628	1639.819	1586.642	1769.628	1656.836
100	3303.922	3314.063	3330.185	3285.016	3316.284	3301.956	3298.249	3303.731
110	19850.17	24578.04	23899.54	19298.76	21348.38	22680.2	20848.72	27093.18

clients	meas 57	meas 58	meas 59	meas 60	meas 61	meas 62	meas 63
10	112.0697	112.1207	107.7257	95.04184	100.3183	105.4215	100.7129
20	201.5546	191.4587	156.4083	211.1588	156.0383	187.5194	196.854
30	223.9015	221.37	210.6528	217.2714	211.6431	226.2292	222.0934
40	316.0355	319.7072	312.8856	321.826	300.6647	308.9131	314.3506
50	590.208	574.019	673.4379	588.0194	566.2029	587.6845	662.5265
60	960.5786	970.2932	982.0238	979.4993	938.4972	977.0445	979.555
70	961.1397	946.4893	990.8966	825.7377	1093.54	910.0089	774.3253
80	1467.377	1243.123	1198.804	1331.331	1358.932	1292.805	1464.751
90	1736.859	1635.29	1536.307	1723.064	1660.502	1588.515	1677.232
100	3344.136	3319.181	3263.133	3304.785	3320.155	3264.835	3273.432
110	25716.2	18640.62	27775.48	20445.88	18949.05	24861.73	23751.8

6 Conclusions

At the end of this work we can summarize our results. Basically we developed a set of tools to analyze the Tairona SIP server under stress situations. These are composed by scripts, Flex parsers and Java algorithms to analyze data, they are listed in the Appendix A. This environment is highly automatized and can be reused for others measures in other studies.

The results we get from the *what-if* analysis of our two models are reported in the Figure 15. As described in Section 4 we followed two methods to create a model that should represent our measurements: a load dependent model with a delay station and a load independent model with a queuing station. The Figure 15 shows that the load dependent model (blue line) has a best fitting on the measured data (red points). This is because a load dependent model changes its service time (and in our case also the response time) in function of the number of clients that are inside the system. On the other side, the load independent model (purple line) does not seem to fit data over than 10-20 clients. We hypothesize that this model is not able to describe the intrinsically parallelism (i.e. the concurrency of messages) of this kind of network service.

Our goals were to improve the knowledge on the MjSIP framework and to

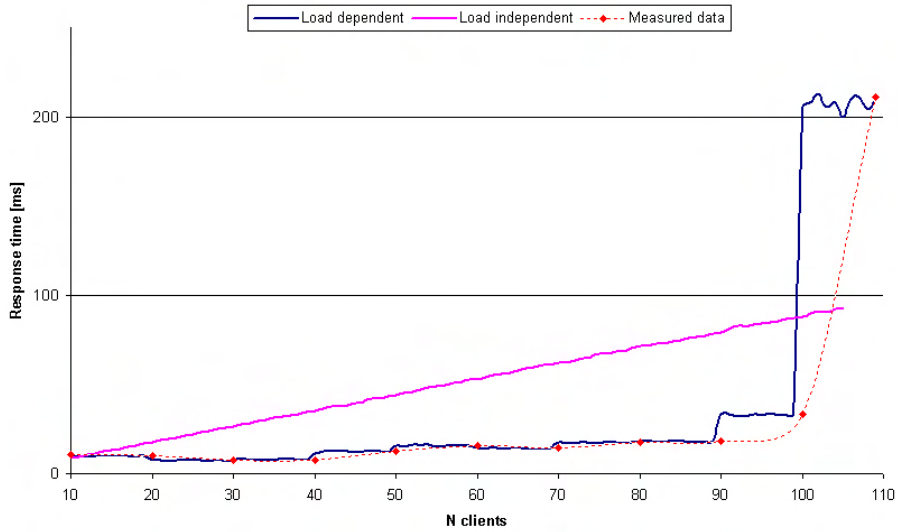


Figure 15: Overall picture of load dependent and load independent models, experimental results

find a critical number of concurrent calls to avoid a crash of the server. We determined that closing to 100 concurrent SIP calls the server collapses.

We focused our analysis only on the behavior of the server when processing SIP call requests. We did not take care of the role of clients, the perception of users and also any kind of Quality of Service because we measured essentially what the MjSIP software does on server side. For these reasons a future development of this study could be the model of the system including clients and network description (i.e. network delay and human reaction time to answer to an incoming call).

Anyway for this kind of server we don't consider the network delay to be a primary parameter of the analysis because the number of the users necessary to crash the system is so limited that with current state of the art network it does not represent a problem. Moreover further details of the measures can be applied. For example, to test how many time is required by the access to the disk stored database it is necessary to find the methods of MjSIP that use it. This means a more extended code analysis and instrumentation.

References

- [1] Amanda Mattiuz, Maksim Djaekov, Ivano Bonesana, Francesco Regazzoni, *Tairona, an Open Source Platform for Worldwide Meeting and Tutoring*, ALaRI, University of Lugano, 2006
- [2] Vijay K. Gurbani, Lalita J. Jagadeesan, and Veena B. Mendiratta, *Characterizing Session Initiation Protocol (SIP) Network Performance and Reliability*, Bell Laboratories, Lucent Technologies Naperville, Illinois, 2005
- [3] Masataka Ohta, *Overload Control in a SIP Signaling Network* in Transaction on Engineering, Computing and Technology v12, march 2006, ISSN 1305-5313
- [4] Edward D. Lazowska, John Zahorjan, G. Scott Graham and Kenneth C. Sevcik, *Quantitative System Performance, Computer System Analysis Using Queueing Network Models*, Prentice-Hall, inc., 1984
- [5] Giuseppe Serazzi, *Performance Evaluation*, lecture notes ALaRI, 2006
- [6] RFC3261, www.ietf.org
- [7] Luca Veltri, *MkSip-Mini-Tutorial*, v.1.5, Luca Veltri, april 24 2005
- [8] <http://www.mjsip.org>
- [9] <http://www.visopsys.org/andy/babylon>
- [10] <http://www.eln.uniroma2.it>
- [11] <http://www.unipr.it>
- [12] <http://www.pointercom.com>
- [13] <http://jmt.sourceforge.net>
- [14] *Java SDK API*, <http://java.sun.com>
- [15] http://www.gnu.org/software/flex/manual/html_mono/flex.html
- [16] http://www.gnu.org/software/bison/manual/html_mono/bison.html

A Source Code

A.1 Call Scripts

```
1  #!/bin/bash
2  # SIPcallBomb.sh script
3
4  echo "number_of_calls_time_of_registration
5  time_of_auto_response_duration_of_a_call"
6
7  port=3000
8  limit=$((3000 + $1))
9
10 # listeners
11 while [ $port -le $limit ]
12 do
13     echo "screen -d -m ./uac.sh -p ${port}"
14     screen --from-url sip:pippo${port}@192.168.68.123 -g $2 -z
15     screen --via-addr 192.168.71.18 -y $3 -t $4"
16     screen -d -m ./uac.sh "-p ${port}"
17     screen --from-url sip:pippo${port}@192.168.68.123 -g $2 -z
18     screen --via-addr 192.168.71.18 -y $3 -t $4"
19
20     port=$((port + 1))
21 done
22
23 sleep 10
24
25 port=4000
26 limit=$((4000 + $1))
27
28 # callers
29 while [ $port -le $limit ]
30 do
31     echo "screen -d -m ./uac.sh -p ${port}"
32     screen --from-url sip:pippo${port}@192.168.68.123 -g $2
33     screen --via-addr 192.168.71.18 -z
34     screen -c sip:pippo$((port-1000))@192.168.68.123 ""
35     screen -d -m ./uac.sh "-p ${port}"
36     screen --from-url sip:pippo${port}@192.168.68.123 -g $2
37     screen --via-addr 192.168.71.18
38     screen -c sip:pippo$((port-1000))@192.168.68.123 "
39     port=$((port + 1))
40 done
41
42 #!/bin/bash
43 # launch calls script
44
45 counter=1
46
47 while [ $counter -le 100 ]
48 do
49     echo "tentativo_n."$counter "chiamate" $((counter * 10))
50     ./SIPcallBomb.sh $((counter * 10)) 1000 2 2
51     sleep 120
52     ssh ivano@192.168.68.123 touch lock.txt < passwd.txt
53     killall screen
54     counter=$((counter + 1))
55     sleep 1
56 done
```


A.2 Parser Flex

```

1  %x TIME
2  %x IP_SOURCE
3  %x IP_DEST
4  %x LENGTH_PKT
5  %x PKT
6
7  %option noyywrap
8
9  NUMBER [0-9]*
10 TIME_IP_SEPARATOR "_" IP "_"
11 IP_ADDRESS ("-"|[0-9]|[a-z]|".")*
12 IP_SRC_DEST_SEPARATOR ">"
13 IP_LENGTH_SEPARATOR "length:"
14 NEW_LINE "\n"
15 CH_13 "\r"
16
17 %%
18
19 <INITIAL>{NUMBER}:"{"NUMBER}":"{NUMBER}"."{NUMBER} {
20     /* getting time */
21     printf ("%s\t", yytext);
22     BEGIN(TIME);
23 }
24 <TIME>{TIME_IP_SEPARATOR} {BEGIN(IP_SOURCE);}
25
26 <IP_SOURCE>{IP_ADDRESS} {
27     /* getting source ip address */
28     printf ("%s\t", yytext); /* src ip*/
29 }
30 <IP_SOURCE>{IP_SRC_DEST_SEPARATOR} {
31     BEGIN(IP_DEST);
32 }
33
34 <IP_DEST>{CH_13} {
35 }
36
37 <IP_DEST>{IP_ADDRESS} {
38     /* getting source ip address */
39     printf ("%s\t", yytext); /* dest ip*/
40     BEGIN(LENGTH_PKT);
41 }
42
43 <LENGTH_PKT>{CH_13} {
44 }
45
46 <LENGTH_PKT>.*{IP_LENGTH_SEPARATOR} {
47 }
48
49
50 <LENGTH_PKT>{NUMBER} {
51     printf ("%s\t", yytext);
52 }
53
54 <LENGTH_PKT>{NEW_LINE} {
55     BEGIN(PKT);
56 }
57
58 <PKT>{CH_13} {
59 }
60

```

```

61 <PKT>. {
62     printf("%s", yytext);
63 }
64
65 <PKT>{NEW_LINE} {
66     printf("\n");
67     BEGIN(INITIAL);
68 }
69
70 <INITIAL>.|\\n      /* eat up any unmatched character */
71
72 %%
73
74 int main () {
75     yylex();
76 }
77

```

A.3 Java SIP Analyzer

```

1  import java.util.StringTokenizer;
2
3  public class SIPRecord {
4
5      public static enum types {
6          REGISTER, INVITE, SIPOK, TRYING, RINGING, ACK, BYE, SIPNOTFOUND, UNKNOWN
7      };
8
9      private String time;
10     private long microsec;
11     private String srcIp;
12     private String destIp;
13     private String packetLen;
14     private String packetData;
15     private types packetType;
16
17     public SIPRecord(String line) {
18         StringTokenizer st = new StringTokenizer(line, "\\t");
19         if (st.hasMoreTokens())
20             setTime(st.nextToken());
21         if (st.hasMoreTokens())
22             setSrcIp(st.nextToken());
23         if (st.hasMoreTokens())
24             setDestIp(st.nextToken());
25         if (st.hasMoreTokens())
26             setPacketLen(st.nextToken());
27         if (st.hasMoreTokens())
28             setPacketData(st.nextToken());
29     }
30
31     public String getDestIp() {
32         return destIp;
33     }
34
35     public void setDestIp(String destIp) {
36         this.destIp = destIp;
37     }
38
39     public String getPacketData() {
40         return packetData;
41     }
42

```

```

43     public void setPacketData(String packetData) {
44         this.packetData = packetData;
45
46         if (packetData.contains("REGISTER"))
47             packetType = types.REGISTER;
48         else if (packetData.contains("INVITE"))
49             packetType = types.INVITE;
50         else if (packetData.contains("Ringing"))
51             packetType = types.RINGING;
52         else if (packetData.contains("Trying"))
53             packetType = types.TRYING;
54         else if (packetData.contains("ACK"))
55             packetType = types.ACK;
56         else if (packetData.contains("BYE"))
57             packetType = types.BYE;
58         else if (packetData.contains("200 OK"))
59             packetType = types.SIPOK;
60         else if (packetData.contains("400 Not Found"))
61             packetType = types.SIPNOTFOUND;
62         else
63             packetType = types.UNKNOWN;
64     }
65
66     public String getPacketLen() {
67         return packetLen;
68     }
69
70     public void setPacketLen(String packetLen) {
71         this.packetLen = packetLen;
72     }
73
74     public String getSrcIp() {
75         return srcIp;
76     }
77
78     public void setSrcIp(String srcIp) {
79         this.srcIp = srcIp;
80     }
81
82     public String getTime() {
83         return time;
84     }
85
86     public void setTime(String time) {
87         this.time = time;
88         StringTokenizer st = new StringTokenizer(time, ":");
89         String token = st.nextToken();
90         // System.out.println("token: "+token);
91         microsec = (long) (Integer.parseInt(token)) * 3600 * 1000000;
92         token = st.nextToken();
93         // System.out.println("token: "+token);
94         microsec += (long) (Integer.parseInt(token)) * 60 * 1000000;
95         token = st.nextToken();
96         // System.out.println("token: "+token);
97         microsec += (long) (Double.parseDouble(token) * 1000000);
98     }
99
100     public long getMicrosec() {
101         return microsec;
102     }
103
104     public void setMicrosec(long time) {

```

```

105         this.microsec = time;
106     }
107
108     public types getPacketType() {
109         return packetType;
110     }
111
112 }

```

```

1  import java.util.Vector;
2
3  public class SIPList {
4
5      private Vector list;
6
7      public SIPList() {
8          list = new Vector();
9      }
10
11     public void add(SIPRecord item) {
12         list.add(list.size(), item);
13         list.trimToSize();
14     }
15
16     public SIPRecord get() {
17         SIPRecord tmp = (SIPRecord) (list.elementAt(0));
18         list.removeElementAt(0);
19         return tmp;
20     }
21
22     public SIPRecord get(String source, String dest) {
23         for (int i = 0; i < list.size(); i++) {
24             SIPRecord tmp = (SIPRecord) list.elementAt(i);
25             if (tmp.getSrcIp().equals(source) && tmp.getDestIp().equals(dest)) {
26                 list.removeElementAt(i);
27                 return tmp;
28             }
29         }
30         return null;
31     }
32
33     public SIPRecord get(String source, String dest, String regex) {
34         for (int i = 0; i < list.size(); i++) {
35             SIPRecord tmp = (SIPRecord) list.elementAt(i);
36             if ((tmp.getSrcIp().equals(source) || source.equals("*"))
37                 && (tmp.getDestIp().equals(dest) || dest.equals("*"))
38                 && tmp.getPacketData().matches(regex)) {
39                 list.removeElementAt(i);
40                 return tmp;
41             }
42         }
43         return null;
44     }
45
46     public SIPRecord get(long microsec) {
47         for (int i = 0; i < list.size(); i++) {
48             SIPRecord tmp = (SIPRecord) list.elementAt(i);
49             if (tmp.getMicrosec() > microsec) {
50                 list.removeElementAt(i);
51                 return tmp;
52             }
53         }
54         return null;

```

```

55     }
56
57     public int getNumberOfElements() {
58         return list.size();
59     }
60
61 }

1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5  import java.util.Vector;
6
7  import org.w3c.dom.css.Counter;
8
9  public class SIPAnalyzer {
10
11     private SIPList Register;
12     private SIPList Ok;
13     private SIPList Invite;
14     private SIPList Trying;
15     private SIPList Ringing;
16     private SIPList Ack;
17     private SIPList Bye;
18     public static final int InviteTime = 0;
19     public static final int TryingTime = 1;
20     public static final int RingingTime = 2;
21     public static final int RegisterTime = 3;
22     public static final int AckTime = 4;
23     public static final int OkTime = 5;
24     public static final int ByeTime = 6;
25     private Vector[] totalTimes = new Vector[7];
26     private static long[] counters = new long[7];
27     public boolean verbose = false;
28     public SIPAnalyzer() {
29
30         Register = new SIPList();
31         Ok = new SIPList();
32         Invite = new SIPList();
33         Trying = new SIPList();
34         Ringing = new SIPList();
35         Ack = new SIPList();
36         Bye = new SIPList();
37
38         for (int i = 0; i < 7; i++)
39             totalTimes[i] = new Vector();
40     }
41
42     public void readLine(String line) {
43         // System.out.println(line);
44         SIPRecord s = new SIPRecord(line);
45         if (s != null && s.getPacketType() != null) {
46             switch (s.getPacketType()) {
47                 case REGISTER:
48                     Register.add(s);
49                     break;
50                 case SIPOK:
51                     Ok.add(s);
52                     break;
53                 case INVITE:
54                     Invite.add(s);
55                     break;

```

```

56         case TRYING:
57             Trying.add(s);
58             break;
59         case RINGING:
60             Ringing.add(s);
61             break;
62         case ACK:
63             Ack.add(s);
64             break;
65         case BYE:
66             Bye.add(s);
67             break;
68         default:
69             System.err.println("Packet unknown");
70     }
71 } else
72     System.err.println("Packet error");
73 }
74
75 public void sipClientProfiler() {
76
77     while (Register.getNumberOfElements() != 0) {
78         SIPRecord ask = Register.get();
79         SIPRecord ok = Ok.get(ask.getDestIp(), ask.getSrcIp());
80
81         if (ok != null) {
82             totalTimes[RegisterTime].add(new Long(ok.getMicrosec()
83                 - ask.getMicrosec()));
84             counters[RegisterTime]++;
85             if (verbose)
86                 System.out.println("REGISTER: ok: " + ok.getMicrosec()
87                     + " ask: " + ask.getMicrosec()
88                     + " totalRegisterTime="
89                     + totalTimes[RegisterTime] + " unRegisterTime="
90                     + counters[RegisterTime]);
91         }
92     }
93
94     while (Invite.getNumberOfElements() != 0) {
95
96         SIPRecord from = Invite.get(" ", " ", ".*(?:sip:user)).*");
97
98         if (from == null)
99             break;
100
101         SIPRecord to = Ack.get(from.getSrcIp(), " ", ".*");
102
103         if (to != null) {
104             totalTimes[InviteTime].add(new Long(to.getMicrosec()
105                 - from.getMicrosec()));
106             counters[InviteTime]++;
107             if (verbose)
108                 System.out.println("INVITE: to: " + to.getMicrosec()
109                     + " from: " + from.getMicrosec()
110                     + " totalInviteTime=" + totalTimes[InviteTime]
111                     + " unInviteTime=" + counters[InviteTime]);
112         }
113     }
114
115     while (Bye.getNumberOfElements() != 0) {
116         SIPRecord from = Bye.get();
117         SIPRecord to = Ok.get(from.getMicrosec());

```

```

118
119         if (to != null) {
120             totalTime[ByeTime].add(new Long(to.getMicrosec()
121                 - from.getMicrosec()));
122             counters[ByeTime]++;
123
124             if (verbose)
125                 System.out.println("BYE: " + to.getMicrosec()
126                     + " from: " + from.getMicrosec()
127                     + " totalByeTime=" + totalTime[ByeTime]
128                     + " ByeTime=" + counters[ByeTime]);
129         }
130     }
131 }
132
133
134 public void sipServerProfiler() {
135
136     // register invite ringing bye
137     while (Register.getNumberOfElements() != 0) {
138         SIPRecord ask = Register.get();
139         SIPRecord ok = Ok.get(ask.getDestIp(), ask.getSrcIp());
140         if (ok != null) {
141             totalTime[RegisterTime].add(new Long(ok.getMicrosec()
142                 - ask.getMicrosec()));
143             counters[RegisterTime]++;
144             if (verbose)
145                 System.out.println("REGISTER: " + ok.getMicrosec()
146                     + " ask: " + ask.getMicrosec()
147                     + " totalRegisterTime="
148                     + totalTime[RegisterTime] + " RegisterTime="
149                     + counters[RegisterTime]);
150         }
151     }
152
153     while (Invite.getNumberOfElements() != 0) {
154         SIPRecord from = Invite.get();
155         SIPRecord to = Invite.get(from.getDestIp(), "*", ".*sip:user.*.*");
156         if (to != null) {
157             totalTime[InviteTime].add(new Long(to.getMicrosec()
158                 - from.getMicrosec()));
159             counters[InviteTime]++;
160             if (verbose)
161                 System.out.println("INVITE: " + to.getMicrosec()
162                     + " from: " + from.getMicrosec()
163                     + " totalInviteTime=" + totalTime[InviteTime]
164                     + " InviteTime=" + counters[InviteTime]);
165         }
166     }
167
168     while (Trying.getNumberOfElements() != 0) {
169         SIPRecord from = Trying.get();
170         SIPRecord to = Trying.get(from.getDestIp(), "*", ".*");
171         if (to != null) {
172             totalTime[TryingTime].add(new Long(to.getMicrosec()
173                 - from.getMicrosec()));
174             counters[TryingTime]++;
175             if (verbose)
176                 System.out.println("TRYING: " + to.getMicrosec()
177                     + " from: " + from.getMicrosec()
178                     + " totalTryingTime=" + totalTime[TryingTime]
179                     + " TryingTime=" + counters[TryingTime]);

```

```

180     }
181 }
182
183 while (Ringing.getNumberOfElements() != 0) {
184     SIPRecord from = Ringing.get();
185     SIPRecord to = Ringing.get(from.getDestIp(), "*", ".*");
186     if (to != null) {
187         totalTimes[RingingTime].add(new Long(to.getMicrosec()
188             - from.getMicrosec()));
189         counters[RingingTime]++;
190         if (verbose)
191             System.out.println("RINGING_:" + to.getMicrosec()
192                 + "from:" + from.getMicrosec()
193                 + "totalRingingTime=" + totalTimes[RingingTime]
194                 + "nRingingTime=" + counters[RingingTime]);
195     }
196 }
197
198 while (Ack.getNumberOfElements() != 0) {
199     SIPRecord from = Ack.get();
200     SIPRecord to = Ack.get(from.getDestIp(), "*", ".*");
201     if (to != null) {
202         totalTimes[AckTime].add(new Long(to.getMicrosec()
203             - from.getMicrosec()));
204         counters[AckTime]++;
205         if (verbose)
206             System.out.println("ACK:" + to.getMicrosec()
207                 + "from:" + from.getMicrosec()
208                 + "totalAckTime=" + totalTimes[AckTime]
209                 + "nAckTime=" + counters[AckTime]);
210     }
211 }
212
213 while (Bye.getNumberOfElements() != 0) {
214     SIPRecord from = Bye.get();
215     SIPRecord to = Bye.get(from.getDestIp(), "*", ".*");
216     if (to != null) {
217         totalTimes[ByeTime].add(new Long(to.getMicrosec()
218             - from.getMicrosec()));
219         counters[ByeTime]++;
220         if (verbose)
221             System.out.println("BYE:" + to.getMicrosec()
222                 + "from:" + from.getMicrosec()
223                 + "totalByeTime=" + totalTimes[ByeTime]
224                 + "nByeTime=" + counters[ByeTime]);
225     }
226 }
227
228 while (Ok.getNumberOfElements() != 0) {
229     SIPRecord from = Ok.get();
230     SIPRecord to = Ok.get(from.getDestIp(), "*", ".*");
231     if (to != null) {
232         totalTimes[OkTime].add(new Long(to.getMicrosec()
233             - from.getMicrosec()));
234         counters[OkTime]++;
235         if (verbose)
236             System.out.println("OK:" + to.getMicrosec()
237                 + "from:" + from.getMicrosec()
238                 + "totalOkTime=" + totalTimes[OkTime]
239                 + "nOkTime=" + counters[OkTime]);
240     }
241 }

```



```

242     }
243
244
245     public double computeAverage(int what) {
246         long totTime = 0;
247
248         for (int i = 0; i < totalTimes[what].size(); i++)
249             totTime += ((Long) (totalTimes[what].elementAt(i))).longValue();
250
251         return (double) totTime / (double) counters[what];
252     }
253
254     public double computeVariance(int what) {
255         double mean = computeAverage(what);
256         double totTime = 0.0;
257
258         for (int i = 0; i < totalTimes[what].size(); i++)
259             totTime += Math.pow(((Long) (totalTimes[what].elementAt(i)))
260                 .longValue()
261                 - mean, 2);
262
263         return (double) totTime / (double) counters[what];
264     }
265
266     public static void main(String[] args) {
267
268         SIPAnalyzer sipa = new SIPAnalyzer();
269
270         String filename = args[0];
271
272         String mode = null;
273         if (args.length > 1)
274             mode = args[1];
275
276         String verbose = null;
277         if (args.length > 2)
278             verbose = args[2];
279
280         try {
281             BufferedReader fileReader = new BufferedReader(new FileReader(
282                 filename));
283
284             String line;
285             while ((line = fileReader.readLine()) != null) {
286                 sipa.readLine(line);
287             }
288
289         } catch (FileNotFoundException e) {
290             e.printStackTrace();
291         } catch (IOException e) {
292             e.printStackTrace();
293         }
294
295         if ((mode != null && mode.equals("-v"))
296             || (verbose != null && verbose.equals("-v")))
297             sipa.verbose = true;
298
299         if ((mode != null && mode.equals("-c"))
300             || (verbose != null && verbose.equals("-c"))) {
301             // CLIENT CODE
302             sipa.sipClientProfiler();
303         } else {

```

```

304         // SERVER CODE
305         sipa.sipServerProfiler();
306     }
307
308     System.out.println(counters[RegisterTime]
309         + "Register average time: "
310         + sipa.computeAverage(SIPAnalyzer.RegisterTime) + " variance "
311         + sipa.computeVariance(SIPAnalyzer.RegisterTime));
312     System.out.println(counters[InviteTime] + "Invite average time: "
313         + sipa.computeAverage(SIPAnalyzer.InviteTime) + " variance "
314         + sipa.computeVariance(SIPAnalyzer.InviteTime));
315     System.out.println(counters[ByeTime] + "Bye average time: "
316         + sipa.computeAverage(SIPAnalyzer.ByeTime) + " variance "
317         + sipa.computeVariance(SIPAnalyzer.ByeTime));
318     System.out.println(counters[TryingTime] + "TryingTime average time: "
319         + sipa.computeAverage(SIPAnalyzer.TryingTime) + " variance "
320         + sipa.computeVariance(SIPAnalyzer.TryingTime));
321     System.out.println(counters[RingingTime]
322         + "RingingTime average time: "
323         + sipa.computeAverage(SIPAnalyzer.RingingTime) + " variance "
324         + sipa.computeVariance(SIPAnalyzer.RingingTime));
325     System.out.println(counters[AckTime] + "AckTime average time: "
326         + sipa.computeAverage(SIPAnalyzer.AckTime) + " variance "
327         + sipa.computeVariance(SIPAnalyzer.AckTime));
328     System.out.println(counters[OkTime] + "OkTime average time: "
329         + sipa.computeAverage(SIPAnalyzer.OkTime) + " variance "
330         + sipa.computeVariance(SIPAnalyzer.OkTime));
331 }
332
333 }

```

A.4 SIPServ Data Analyzer

```

1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5  import java.util.StringTokenizer;
6
7  public class SIPServ {
8
9      public static void main(String args[]) {
10
11          double reg = 0.0;
12          double inv = 0.0;
13          double resp = 0.0;
14          double ack = 0.0;
15          double bye = 0.0;
16
17          int regCnt = 0;
18          int invCnt = 0;
19          int respCnt = 0;
20          int ackCnt = 0;
21          int byeCnt = 0;
22
23          String status = null;
24
25          try {
26              BufferedReader bir = new BufferedReader(new FileReader(args[0]));
27              String line = null;
28              while ((line = bir.readLine()) != null) {
29                  StringTokenizer st = new StringTokenizer(line, " ");

```

```

30         if (line.startsWith(":"))
31             continue;
32         while (st.hasMoreTokens()) {
33             String tmp = st.nextToken();
34             // System.out.println("tmp: "+tmp);
35             if (!tmp.startsWith("_")) {
36                 if (tmp.equals("REGISTER")) {
37                     regCnt++;
38                     status = tmp;
39                 } else if (tmp.equals("INVITE")) {
40                     invCnt++;
41                     status = tmp;
42                 } else if (tmp.equals("RESPONSE")) {
43                     respCnt++;
44                     status = tmp;
45                 } else if (tmp.equals("BYE")) {
46                     byeCnt++;
47                     status = tmp;
48                 } else if (tmp.equals("ACK")) {
49                     ackCnt++;
50                     status = tmp;
51                 } else {
52                     // A number!!!
53                     if (status.equals("REGISTER")) {
54                         reg += Long.parseLong(tmp);
55                     } else if (status.equals("INVITE")) {
56                         inv += Long.parseLong(tmp);
57                     } else if (status.equals("RESPONSE")) {
58                         resp += Long.parseLong(tmp);
59                     } else if (status.equals("BYE")) {
60                         bye += Long.parseLong(tmp);
61                     } else if (status.equals("ACK")) {
62                         ack += Long.parseLong(tmp);
63                     }
64                     status = null;
65                 }
66             }
67         }
68     }
69     catch (FileNotFoundException e) {
70         e.printStackTrace();
71     } catch (IOException e) {
72         e.printStackTrace();
73     }
74
75     if (args.length > 1)
76         System.out.println("REGISTER_ INVITE_ RESPONSE_ BYE_ ACK");
77     System.out.println(reg / regCnt + "_" + inv / invCnt + "_" + resp
78         / respCnt + "_" + bye / byeCnt + "_" + ack / ackCnt);
79
80 }
81 }

```

B JMT Screenshots

The screenshots of the JMT tool simulating the models as shown in the following figures:

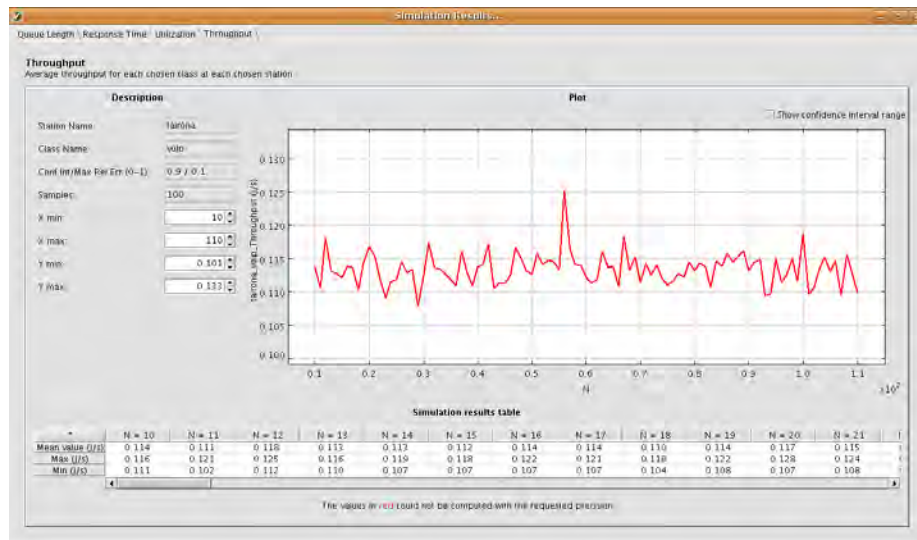


Figure 16: The screenshot of the throughput for the load independent model.

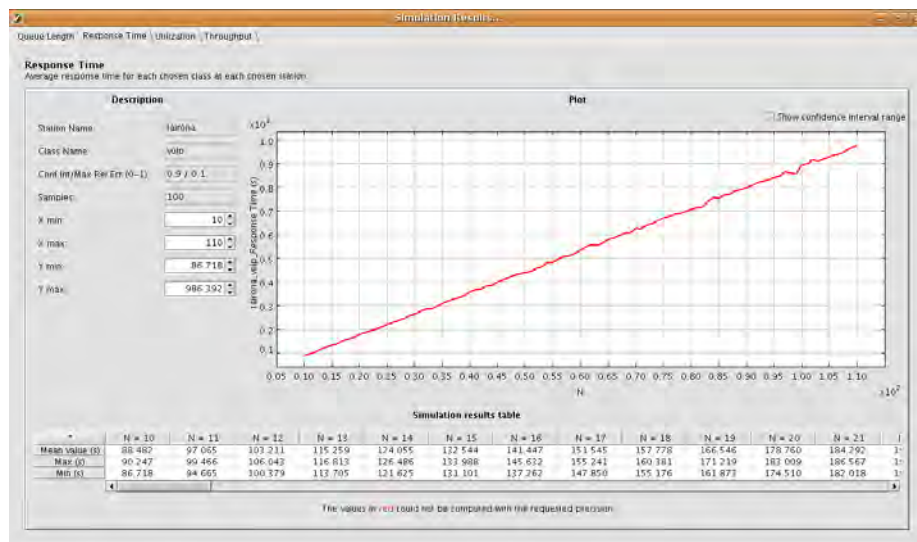


Figure 17: The screenshot of the response time for the load independent model.

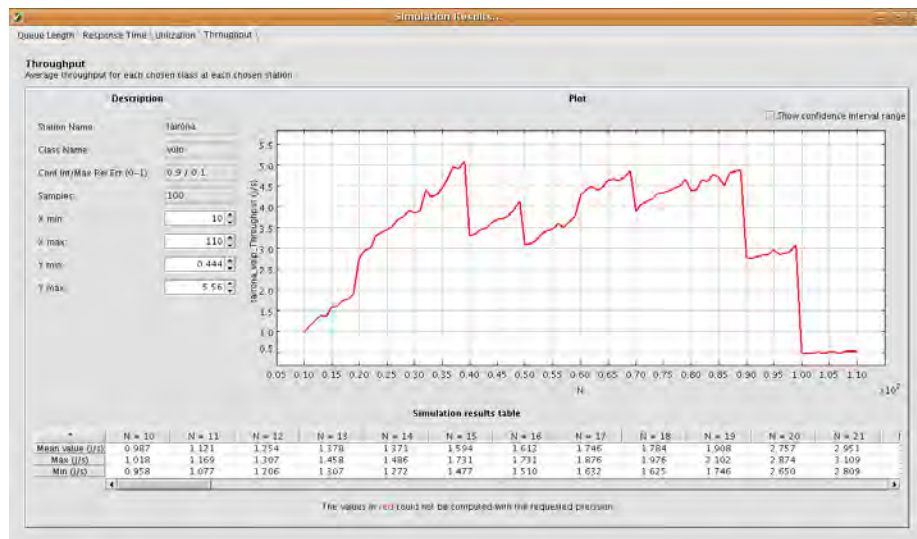


Figure 18: The screenshot of the throughput for the load dependent model

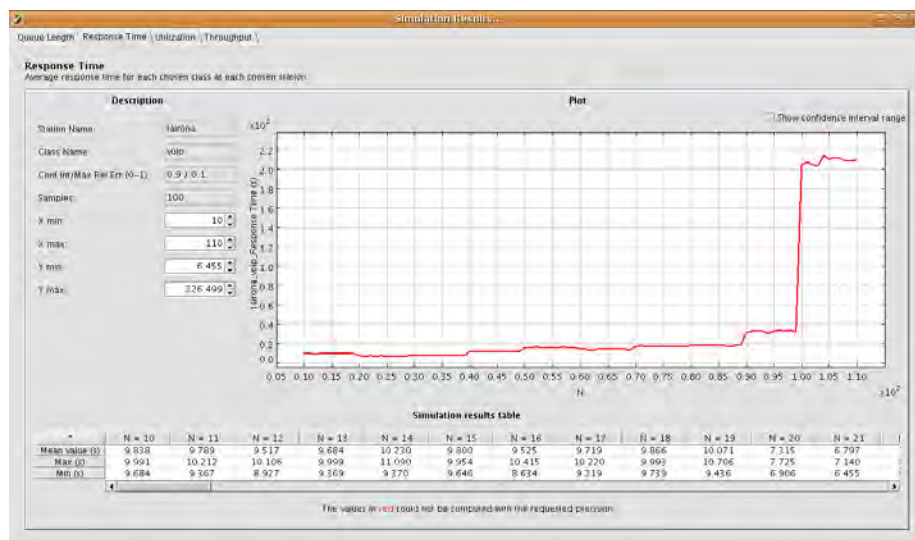


Figure 19: Utilization of server plotted by JMT.

8 – Various Topics

8.1 – Modeling a Road Junction for Vehicular Traffic Control	348
--	-----

Modeling of a road junction using queuing theory, MATLAB and JMT environments.

Francesco Zanini.

Project developed @ALaRI supervised by Prof. G.Serazzi.

Jan.18, 2007

1 Problem definition

Aim of this work is to describe road junction systems. This article is going to focus on the performance of these kind of systems from a mathematical point of view using queuing theory. The most popular types of road junctions are roundabouts and semaphores. Next section is going to analyze the semaphore from both a purely mathematical and a simulation point of view. Section 3 is going to focus on roundabouts. A comparison and a smart solution to improve traffic junctions is given in section 4. A Graphical User Interface using MATLAB environment has also been developed in order to show complex mathematical formulas in a more intuitive way. All mathematical model have been developed assuming a number of input lanes equal to 4, 2 per direction.

2 Semaphore

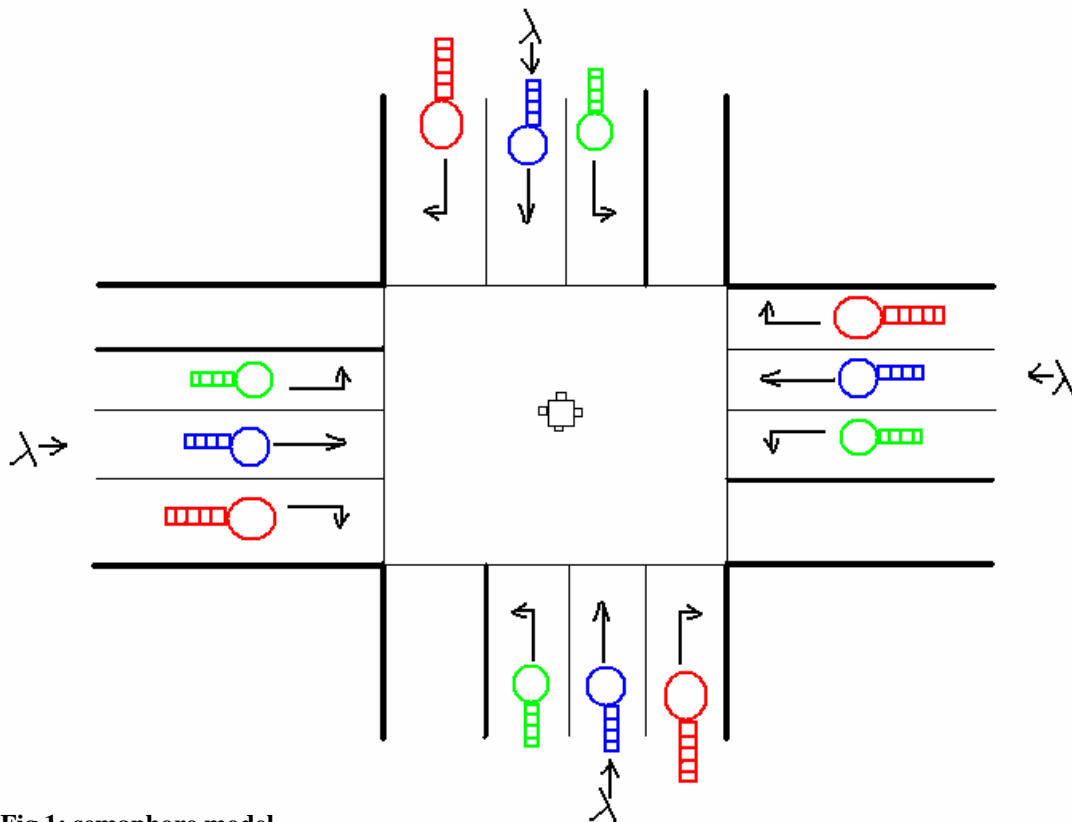


Fig.1: semaphore model.

Picture of fig.1 shows the diagram of a generic 3 lane semaphore (overall number of lanes for each street equals 4) where λ is the input load for each street. In order to optimize the utilization of this junction, the following ‘transmission protocol’ is presented in next picture.

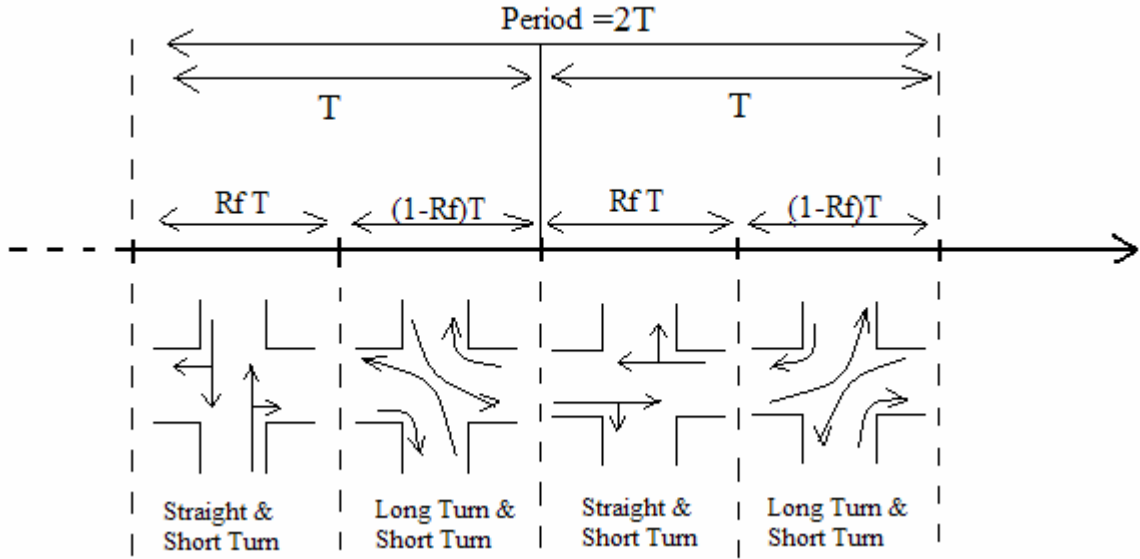


Fig.2: semaphore protocol.

This road junction has been modelled with a system protocol similar to the one of Time Domain Multiple Access. Each road is modelled by 3 network servers, one for each lane with different input load, and different service time. R_f and T are free parameters that can be used to tailor this junction basing on the input load. Subscript indexes st , s and lt refer respectively to the short-turn lane, straight lane and long-turn lane. Every road has this structure. Poisson arrivals are assumed on every input queue, with load intensity λ .

Average servers service time are constant and are calculated basing on the protocol and on empirical data (cars speed parameters R_{st} , R_s , R_{lt} [cars/s]), average service times are:

$$S_s = \frac{2T}{R_s \cdot R_f \cdot T} = \frac{2}{R_f \cdot R_s}$$

$$S_{st} = \frac{2T}{R_{st} \cdot T} = \frac{2}{R_{st}}$$

$$S_{lt} = \frac{2T}{R_{lt} \cdot (1 - R_f) \cdot T} = \frac{2}{(1 - R_f) \cdot R_{lt}}$$

The maximum input load on each queue is given by the fact that:

$$\lambda \cdot S \leq 1$$

where λ is the input load and S the service time (S could be equal to S_s , S_{st} or S_{lt}), so, the maximum input load is given by

$$\lambda_{\max} = \frac{1}{S}$$

While S_{st} is fixed, S_s and S_{lt} can be proper set in order to maximize the maximum input allowable load basing on R_s and R_{lt} parameters.

The average waiting time in queue W is given by:

$$W = R + NS$$

where N is the average queue length and R the average residual service time. From “Little law”,

$$N = \lambda W$$

substituting,

$$W = \frac{R}{1 - \lambda S}$$

at this point the average time D spent by each user in the system is given by

$$D = W + \frac{1}{R_{sc}, R_{lt}, R_s}$$

To compute these values each queue must be considered separately.

Queue short-turn residual service time calculation:

The average value of the residual service time is calculated integrating the area of the following picture of R(t).

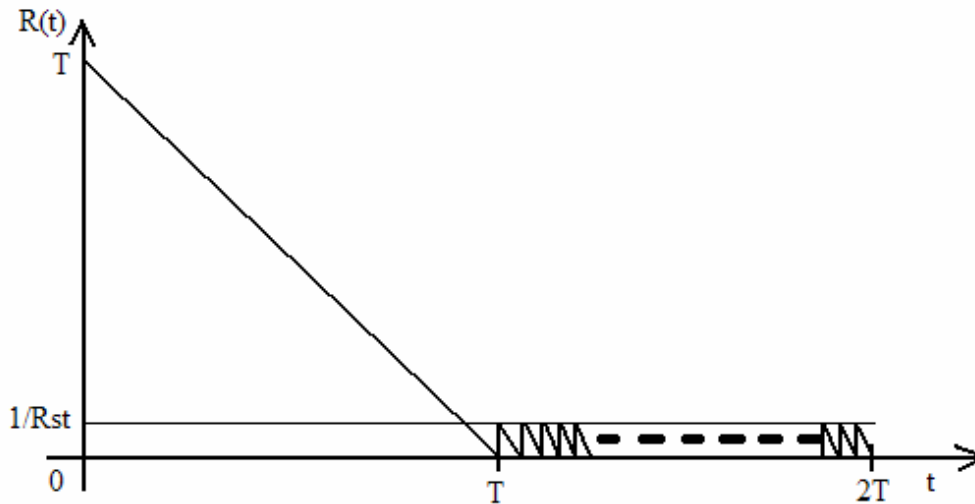


Fig.3: R(t) for the short turn queue.

This picture shows that the server works only for T seconds and it is idle for T seconds. All triangles in picture 3 are rectangle and isosceles.

$$R = \frac{1}{4} \left(T + \frac{1}{R_{st}} \right)$$

Queue long-turn residual service time calculation:

The average value of the residual service time is calculated in the same way as did before but with reference to the following picture.

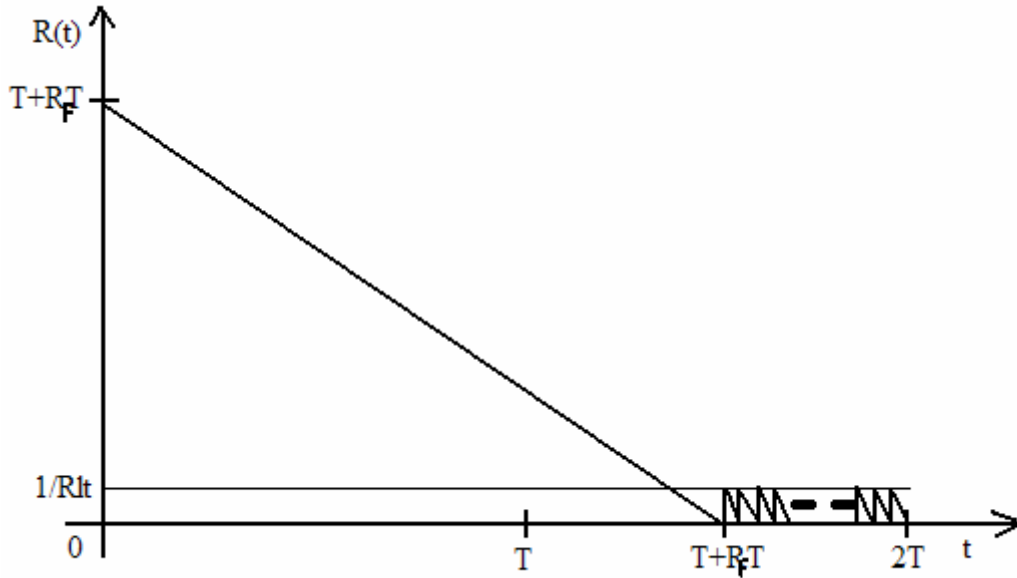


Fig.4: R(t) for the short turn queue.

This picture shows that the server works only for $T-R_fT$ seconds and it is idle for $T+R_fT$ seconds. All triangles in picture 4 are rectangle and isosceles.

$$R = \frac{1}{4} \left(T(1 + R_f)^2 + \frac{1 - R_f}{R_{sl}} \right)$$

Queue straight residual service time calculation:

The average value of the residual service time is calculated in the same way as did before but with reference to the following picture.

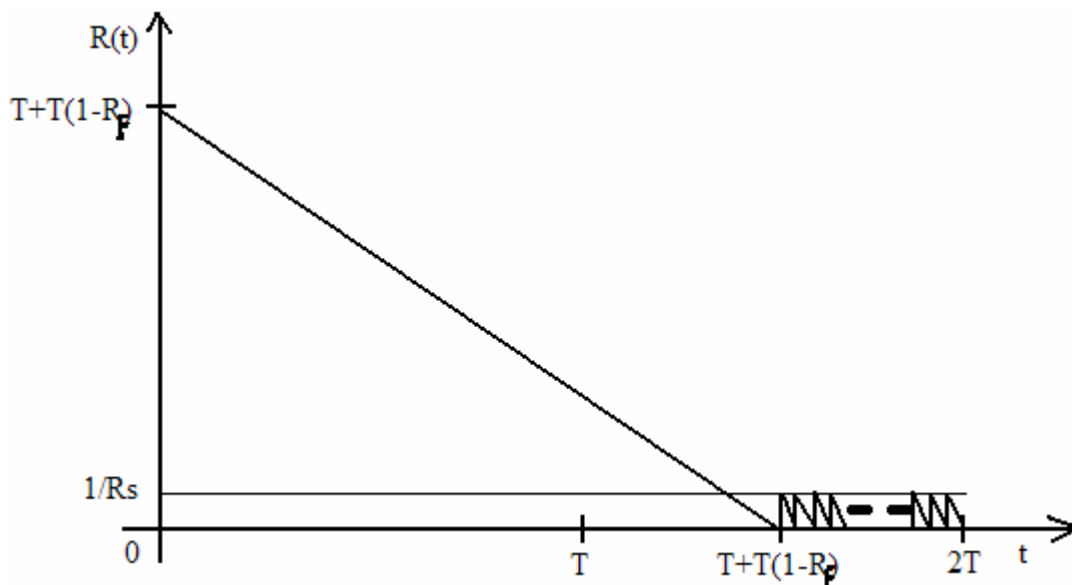


Fig.5: R(t) for the straight queue.

This picture shows that the server works only for RfT seconds and it is idle for $T+T(1-Rf)$ seconds. All triangles in picture 4 are rectangle and isosceles.

$$R = \frac{1}{4} \left(T(2 - Rf)^2 + \frac{Rf}{Rd} \right)$$

Graphical results using both MATLAB and JMT are reported in section 4 and 5.

3 Roundabouts

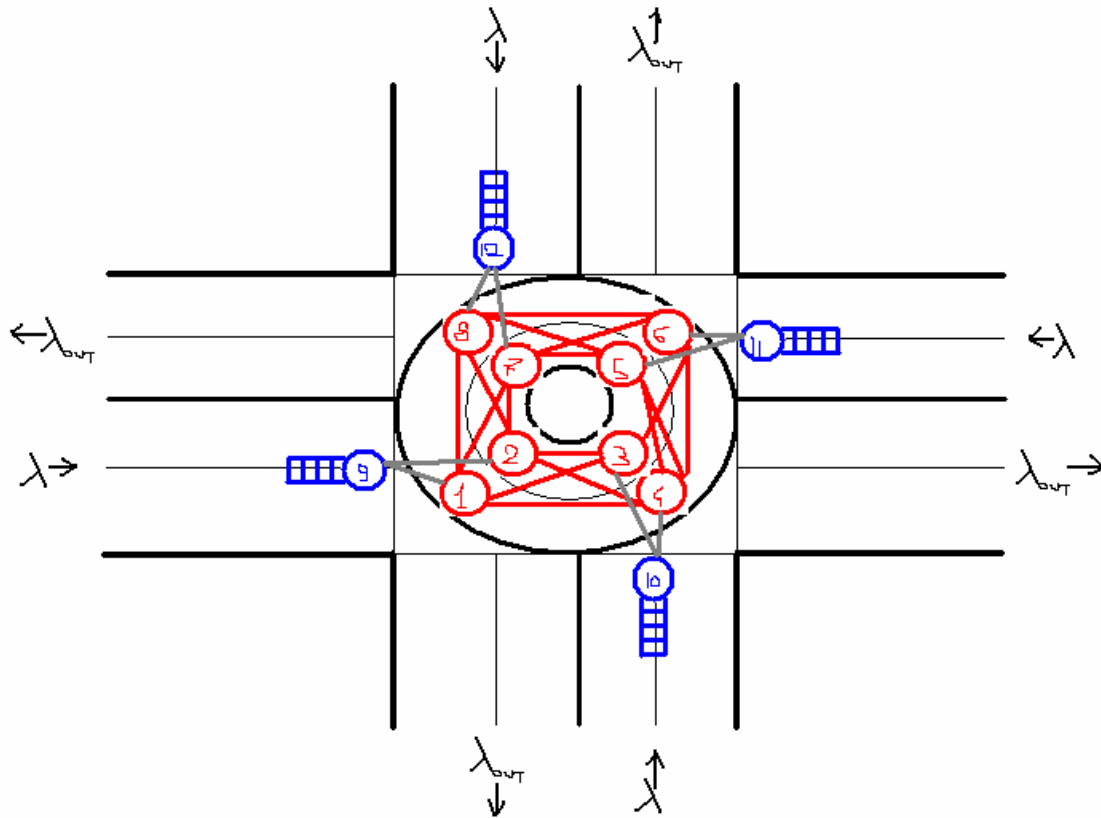


Fig.6: roundabout model.

Picture of fig.6 shows the diagram of a generic 2 lane ($p=2$) roundabout (overall number of lanes for each street equals 4) where λ is the overall roundabout input load. Poisson arrivals are assumed on every input queue. This road junction has been modelled with a system protocol similar to the one of a slotted ring. Each input road is modelled as 1 network server with service time depending on the probability to inject a token into one of the two quarter of the roundabout. Each quarter of each lane of each quarter of the roundabout is modelled as a server as well with fixed service time that accepts only 1 jobs per time.

Average parameters are calculated basing on the protocol and on empirical data (cars speed to cross one quarter of the roundabout R_r [cars/s]).

From ‘Little’s law’ applied to the overall red network (subscript index t),

$$N_t = \lambda_t D_t$$

where D_t is the average system queuing time, N_t the number of cars and λ_t the input load. Now we have to find relations that relates general system's parameters to single server's one that can be known.

The average number of cars in the overall system equals

$$N_t = G \cdot p \cdot n$$

where $G \cdot p$ is the number of queues (red straight connecting servers) in the system and n is the average number of cars into each server. Applying little to the single server,

$$n = \lambda_s D$$

where D is the average time spent queuing for each car into each server and λ_s the input load of each server. From network's topology,

$$\lambda_t = G \lambda$$

The average system service time D_t is given by:

$$D_t = H \cdot D$$

where H is the average number of hop (server crossing) between source and destination. This depends on the network topology and on traffic spreading among the roundabout.

Unifying all previous consideration,

$$G \cdot p \cdot \lambda_s \cdot D = H \cdot D \cdot G \cdot \lambda$$

Solving for $\rho = \lambda_s \cdot S$, where $S=1/Rr$,

$$\rho = \frac{H \cdot S \cdot \lambda}{p}$$

The parameter ρ expresses the utilization time of each red server. This means also the probability that a slot (1/4 of the roundabout) is full.

If the network is not congested, than the input flow at each street λ must be equal to the output flow of it λ_{out} . The last one is given by

$$\lambda_{out} = \frac{\rho \cdot r \cdot p}{S}$$

where r is the probability that a car is at its destination node. λ is given by expressing the just derived ρ in function of it:

$$\lambda = \frac{\rho \cdot p}{S \cdot H}$$

Comparing last 2 equations, it can be derived that

$$r = \frac{1}{H}$$

If we consider now the blue servers network, the service time S_b of them depends on the probability of injecting a car into the red one. The probability density function of S_b is given by:

$$P(S_b = nS) = S(1 - P_s)^{n-1} P_s$$

where P_s is the probability to inject a car in a time interval S . Mean value and Variance are given by,

$$\overline{S_b} = \frac{S}{P_s}$$

$$\overline{S_b^2} = \frac{S^2(2 - P_s)}{P_s^2}$$

Substituting previous equation into P.K. formula to calculate the average waiting time inside the queue of the blue system W , we obtain:

$$W = \frac{S \cdot \lambda \cdot S \cdot (2 - P_s)}{2P_s(P_s - \lambda S)}$$

P_s is 1(always) except when all slots are busy and not arrived at destination, this equals to:

$$P_s = 1 - (\rho(1 - r))^p = 1 - \left(\rho \left(1 - \frac{1}{H} \right) \right)^p$$

Merging all previous derivations, W can be expressed as:

$$W = \frac{S \cdot x \cdot \left(\frac{2}{1 - \left(\frac{(H-1)x}{p} \right)^p} - 1 \right)}{2 \left(1 - x - \left(\frac{(H-1)x}{p} \right)^p \right)}$$

where $x = \lambda \cdot S$

The overall total delay inside the system D is given by:

$$D = W + H \cdot S$$

4 Comparison of both systems using MATLAB and proposed solution

In order to present previous equations in a more intuitive way, a graphical user interface has been developed (see appendix A). Queue length and response time have been calculated for both systems for different input load. Following picture shows an example of this software.

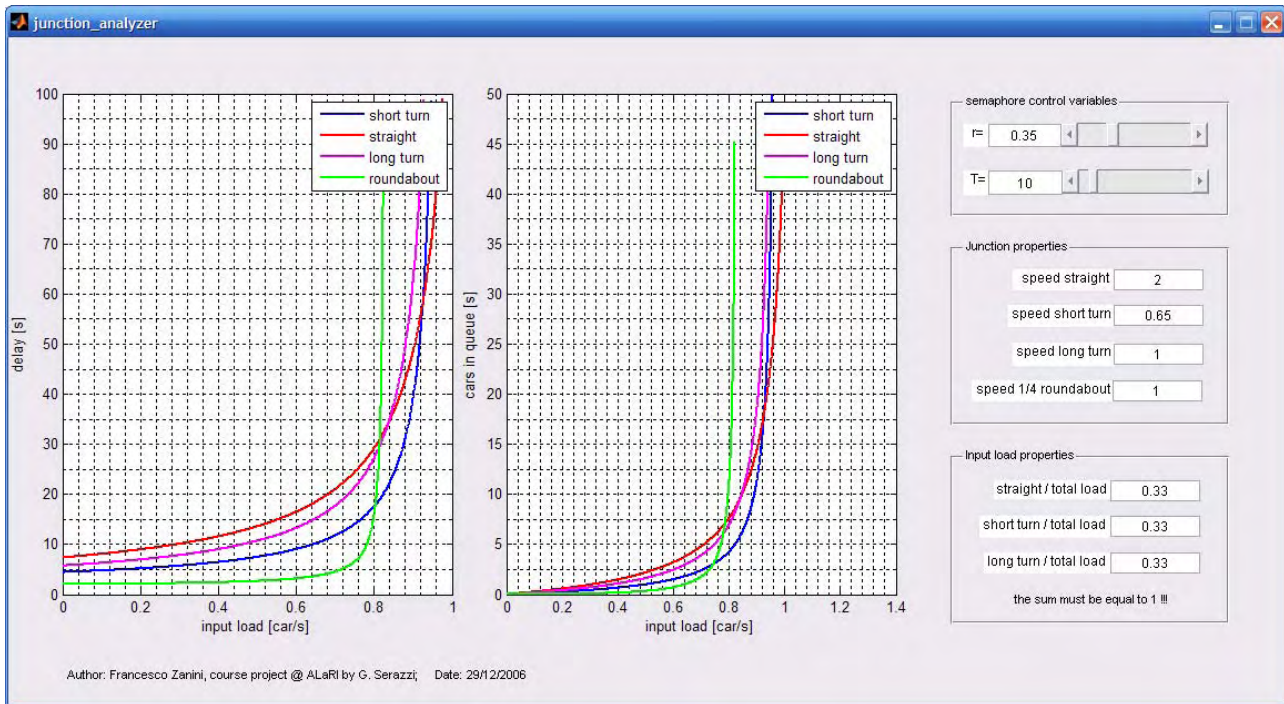


Fig.7: Junction analyzer.

Input variables are divided into 3 blocks: semaphore control variables, junction properties and input load properties.

In the first group semaphore control variables as the semaphore period T and the protocol parameters r ($=R_f$ in formulas) can be properly set.

The second group sets cars speed properties while crossing these junctions.

The last group contains parameters expressing the traffic final destination.

In fig. 7 for parameters shown in the picture, with r and T sized in order to improve the performance of both junctions, for input loads lower than 0.8 cars/s the roundabout shows the best performance. For higher input loads the semaphore is better than the roundabout.

The proposed solution is to put traffic lights on the input of the roundabout than turns the roundabout into a traffic lights when the network gets crowded. If the just derived equations are put into the intelligent part of the semaphore, the semaphore just looking at the length of the queue is able to understand the point at which turn the junction into a traffic light or into a roundabout. In this example, the critical point is at 0.7 cars/s.

5 Checking simulation results using JMT

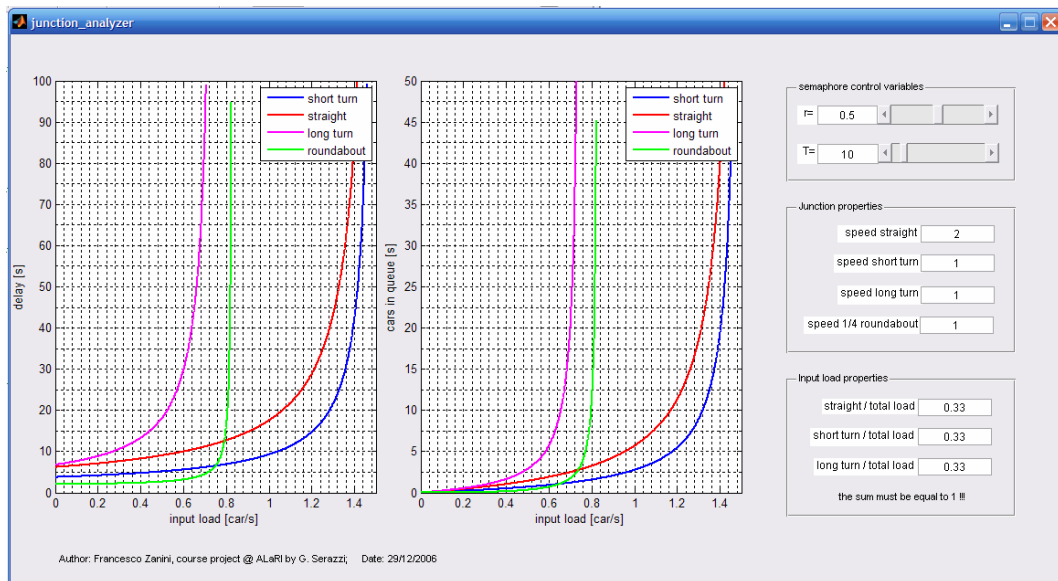


Fig.8: Matlab model of both Junctions.

Next pictures show the same system modelled using JMT.

Semaphore: model and obtained results

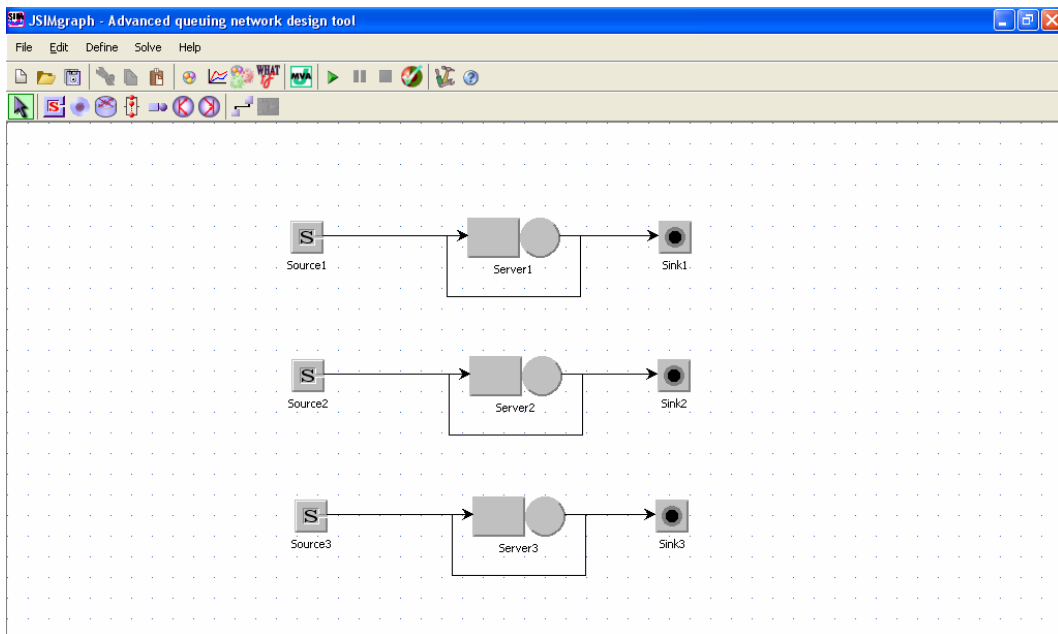


Fig.9: Semaphore JMT model of a road junction.



Fig.10: Semaphore queue length simulation results of server 1 & 2: (straight) & short turn. The input load is 1/3 of the simulation made with Matlab.

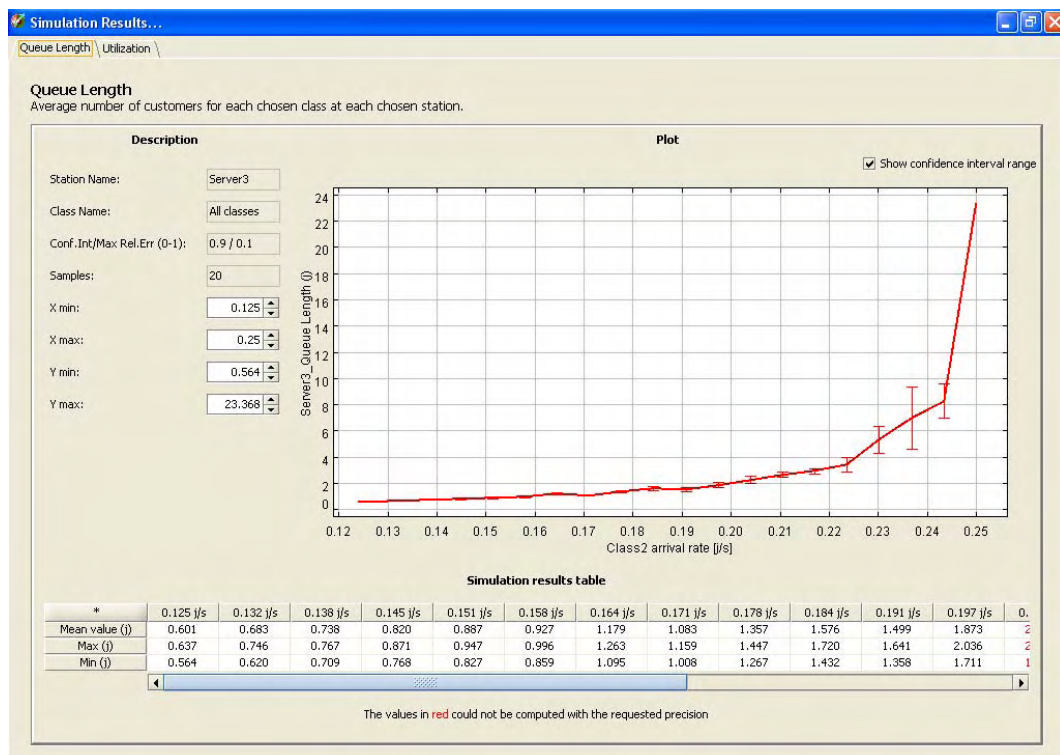


Fig.11: Semaphore queue length simulation results of server 3: (long turn). The input load is 1/3 of the simulation made with Matlab.

Roundabout: model and obtained results

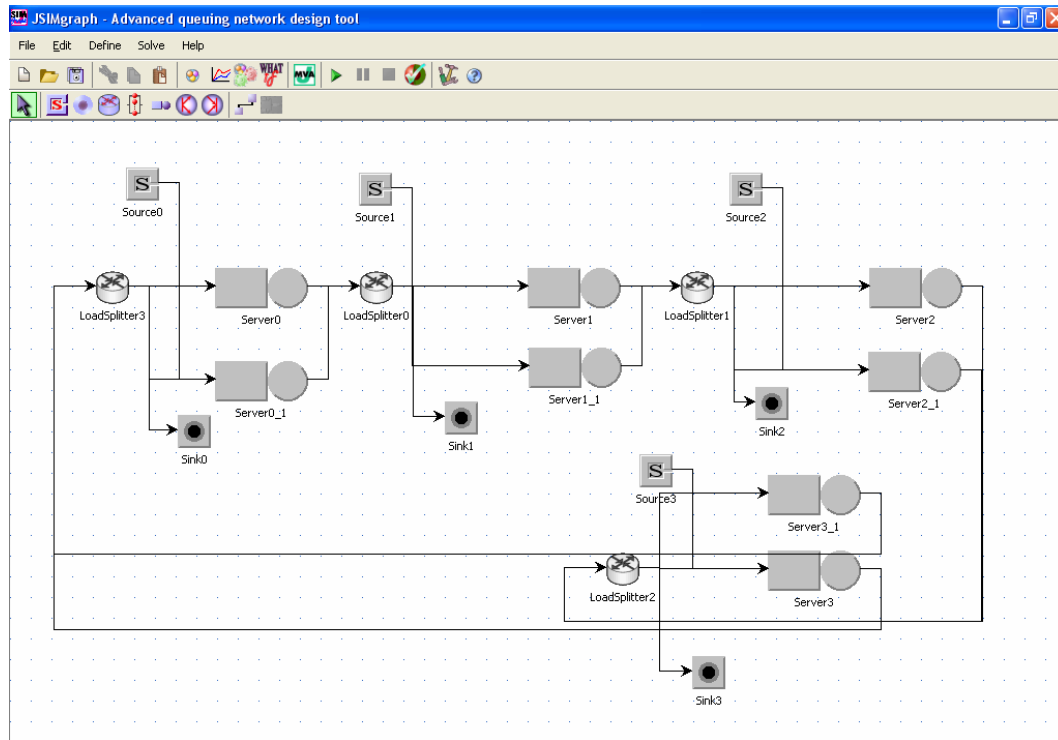


Fig.12: Roundabout JMT model of a road junction.

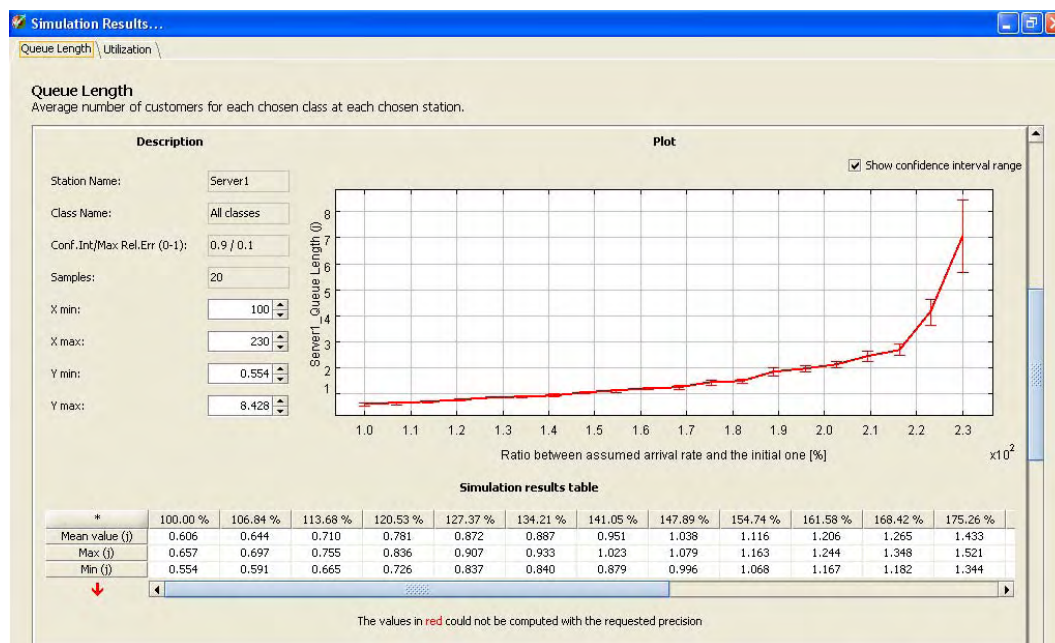


Fig.13: Roundabout JMT model of a road junction the value corresponding to 230% equals to 0.83 cars/s as predicted by the derived Model.

It should be noted that in empirical simulations, analytical derived model and Matlab GUI model perfectly match each others in both roundabout and crossroads.

6 Appendix

GUI object source code

```
% Performance evaluation of streets junctions.
%
% author: Francesco Zanini
% ALaRI 28/12/2006
%
function varargout = junction_analyzer(varargin)
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @junction_analyzer_OpeningFcn, ...
    'gui_OutputFcn', @junction_analyzer_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);
```

```
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
function junction_analyzer_OpeningFcn(hObject, eventdata, handles, varargin)
handles.r=0.5;
handles.T=10; %s
handles.rd=2; %car/s
handles.rsc=1; %car/s
handles.rsl=1; %car/s
handles.rr=1; %car/s
handles.fsc=1/3;
handles.fd=1/3;
handles.fsl=1/3;
```

```
[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
axes(handles.delay);
delay(l,Dd,Dsc,Dsl,Dr);
axes(handles.queue);
queue(l,Nd,Nsc,Nsl,Nr);
```

```
handles.output = hObject;
```

```
guidata(hObject, handles);
```

```
function varargout = junction_analyzer_OutputFcn(hObject, eventdata, handles)
```

```
varargout{1} = handles.output;
```

```
function sr_Callback(hObject, eventdata, handles)
```

```
handles.r=get(hObject,'Value');
```

```
[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
```

```

axes(handles.delay);
delay(1,Dd,Dsc,Dsl,Dr);
axes(handles.queue);
queue(1,Nd,Nsc,Nsl,Nr);
guidata(hObject, handles);

set(handles.vr,'String',get(hObject,'Value'))

function sr_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function vr_CreateFcn(hObject, eventdata, handles)

function pT_Callback(hObject, eventdata, handles)

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

if isempty(NewVal) | (NewVal< 1) | (NewVal>100),

    OldVal = get(handles.sT,'Value');
    set(handles.sT,'String',OldVal)
else,

    set(handles.sT,'Value',NewVal)

    handles.T=NewVal;

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
axes(handles.delay);
delay(1,Dd,Dsc,Dsl,Dr);
axes(handles.queue);
queue(1,Nd,Nsc,Nsl,Nr);
guidata(hObject, handles);
end;

function vr_Callback(hObject, eventdata, handles)

set(handles.sr,'String',get(hObject,'Value'))
NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

if isempty(NewVal) | (NewVal< 0.15) | (NewVal>0.85),

    OldVal = get(handles.sr,'Value');
    set(handles.sr,'String',OldVal)
else,

    set(handles.sr,'Value',NewVal)

    handles.r=NewVal;

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
axes(handles.delay);
delay(1,Dd,Dsc,Dsl,Dr);
axes(handles.queue);
queue(1,Nd,Nsc,Nsl,Nr);

```

```

    guidata(hObject, handles);
end;

```

```

function prd_Callback(hObject, eventdata, handles)

```

```

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

```

```

if (NewVal>0),
    handles.rd=NewVal;

```

```

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
    axes(handles.delay);
    delay(l,Dd,Dsc,Dsl,Dr);
    axes(handles.queue);
    queue(l,Nd,Nsc,Nsl,Nr);
    guidata(hObject, handles);
end;

```

```

function prd_CreateFcn(hObject, eventdata, handles)

```

```

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function prsc_Callback(hObject, eventdata, handles)

```

```

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

```

```

if (NewVal>0),
    handles.rsc=NewVal;

```

```

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
    axes(handles.delay);
    delay(l,Dd,Dsc,Dsl,Dr);
    axes(handles.queue);
    queue(l,Nd,Nsc,Nsl,Nr);
    guidata(hObject, handles);
end;

```

```

function prsc_CreateFcn(hObject, eventdata, handles)

```

```

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function prsl_Callback(hObject, eventdata, handles)

```

```

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

```

```

if (NewVal>0),
    handles.rsl=NewVal;

```

```

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
axes(handles.delay);
delay(l,Dd,Dsc,Dsl,Dr);

axes(handles.queue);
queue(l,Nd,Nsc,Nsl,Nr);
guidata(hObject, handles);
end;

function prsl_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function prr_Callback(hObject, eventdata, handles)

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

if (NewVal>0),
    handles.rr=NewVal;

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
axes(handles.delay);
delay(l,Dd,Dsc,Dsl,Dr);
axes(handles.queue);
queue(l,Nd,Nsc,Nsl,Nr);
guidata(hObject, handles);
end;

function prr_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function sT_Callback(hObject, eventdata, handles)

handles.T=get(hObject,'Value');

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
axes(handles.delay);
delay(l,Dd,Dsc,Dsl,Dr);
axes(handles.queue);
queue(l,Nd,Nsc,Nsl,Nr);
guidata(hObject, handles);

set(handles.pT,'String',get(hObject,'Value'))

function sT_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

```

```

function pT_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function pshort_Callback(hObject, eventdata, handles)

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

if (NewVal>=0 && NewVal<=1),
    handles.fsc=NewVal;

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
    axes(handles.delay);
    delay(l,Dd,Dsc,Dsl,Dr);
    axes(handles.queue);
    queue(l,Nd,Nsc,Nsl,Nr);
    guidata(hObject, handles);
end;

function pshort_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function pstraight_Callback(hObject, eventdata, handles)

function pstraight_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function plong_Callback(hObject, eventdata, handles)

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

if (NewVal>=0 && NewVal<=1),
    handles.fsl=NewVal;

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
    axes(handles.delay);
    delay(l,Dd,Dsc,Dsl,Dr);
    axes(handles.queue);
    queue(l,Nd,Nsc,Nsl,Nr);
    guidata(hObject, handles);
end;

function plong_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function pstra_Callback(hObject, eventdata, handles)

```

```

NewStrVal = get(hObject,'String');
NewVal = str2num(NewStrVal);

if (NewVal>=0 && NewVal<=1),
    handles.fd=NewVal;

[l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(handles.r,handles.T,handles.rd,handles.rsc,handles.rsl,handles.rr,handles.fsc,handles.fd,handles.fsl);
axes(handles.delay);
delay(l,Dd,Dsc,Dsl,Dr);
axes(handles.queue);
queue(l,Nd,Nsc,Nsl,Nr);
guidata(hObject, handles);
end;

function pstra_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Functions source code

```

% Performance evaluation of streets junctions.
%
% author: Francesco Zanini
% ALaRI 28/12/2006
%
function [l,Dd,Dsc,Dsl,Dr,Nd,Nsc,Nsl,Nr]=calc(r,T,rd,rsc,rsl,rr,fsc,fd,fsl);

H=(1*fsc)+(2*fd)+(3*fsl);
Sd=(2*fd)/(r*rd);
Ssc=(2*fsc)/(rsc);
Ssl=(2*fsl)/((1-r)*rsl);
Sr=1/(0.83*rr);
a=[Sd Ssc Ssl Sr];
b=(1/min(a))+0.1;
l=0:1/1000:b;
Dsc=[];
Dd=[];
Dsl=[];
Nsc=[];
Nd=[];
Nsl=[];
Dr=[];
Nr=[];

% semaphore
for i=1:length(l) %sc
    R=0.25*(T+(1/rsc));
    W=R/(1-(l(i)*Ssc));
    D=W+(1/rsc);
    N=l(i)*fsc*W;
    if (D<100) Dsc=[Dsc D];end;
    if (N<50) Nsc=[Nsc N];end;
    if (D>100 && N>50) break;end;
end;
for i=1:length(l) %d
    R=0.25*(T*(2-r)*(2-r)+(r/rd));

```



```

W=R/(1-(l(i)*Sd));
D=W+(1/rd);
N=l(i)*fd*W;
if (D<100) Dd=[Dd D];end;
if (N<50) Nd=[Nd N];end;
if (D>100 && N>50) break;end;
end;
for i=1:length(l) %sl
    R=0.25*(T*(1+r)*(1+r)+((1-r)/rsl));
    W=R/(1-(l(i)*Ssl));
    D=W+(1/rsl);
    N=l(i)*W*fsl;
    if (D<100) Dsl=[Dsl D];end;
    if (N<50) Nsl=[Nsl N];end;
    if (D>100 && N>50) break;end;
end;

%roundabout
for i=1:length(l) %sc
    x=l(i)/rr;
    v=((H-1)*x/2)^2;
    num=x*((2/(1-v))-1);
    den=2*(1-x-v);
    W=num/(rr*den);
    D=W+(2/rr);
    N=l(i)*W;
    if (D<100) Dr=[Dr D];end;
    if (N<50) Nr=[Nr N];end;
    if (D>100 && N>50) break;end;
end;
end

% Performance evaluation of streets junctions.
%
% author: Francesco Zanini
% ALaRI 28/12/2006
%
function delay(l,Dd,Dsc,Dsl,Dr);

plot(1:length(Dsc),Dsc,'LineWidth',1.5);
hold on;
plot(1:length(Dd),Dd,'r','LineWidth',1.5);
plot(1:length(Dsl),Dsl,'m','LineWidth',1.5);
plot(1:length(Dr),Dr,'g','LineWidth',1.5);
grid minor;
v=AXIS;
set(gca,'XTick',v(1):0.2:v(2));
set(gca,'XTickLabel',v(1):0.2:v(2));
xlabel('input load [car/s]');ylabel('delay [s]')
legend('short turn','straight','long turn','roundabout');
hold off;
end

% Performance evaluation of streets junctions.
%
% author: Francesco Zanini
% ALaRI 28/12/2006
%
function queue(l,Nd,Nsc,Nsl,Nr);

```

```

plot(1:length(Nsc),Nsc,'LineWidth',1.5);
hold on;
plot(1:length(Nd),Nd,'r','LineWidth',1.5);
plot(1:length(Nsl),Nsl,'m','LineWidth',1.5);
plot(1:length(Nr),Nr,'g','LineWidth',1.5);
grid minor;
v=AXIS;
set(gca,'XTick',v(1):0.2:v(2));
set(gca,'XTickLabel',v(1):0.2:v(2));
xlabel('input load [car/s]');ylabel('cars in queue [s]')
legend('short turn','straight','long turn','roundabout');
hold off;
end

```

1 Bibliography

- [1] Material of the course “Performance Evaluation” by G. Serazzi
- [2] Material of the course “Data Networks B” by A. Bononi
- [3] Data Networks, D. P. Bertsekas, R. G. Gallager
- [4] assignments of the course @ MIT by D. P. Bertsekas.
- [5] MATLAB, user guide
- [6] JMT, user guide