Miquel Talavera Foix

Load balancing control of a server network cluster

Master Thesis

LAAS-CNRS

July 2014

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA INDUSTRIAL DE BARCELONA

LABORATOIRE D'ANALYSE ET D'ARCHITECTURE DES SYSTÈMES, CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DE TOULOUSE III PAUL SABATIER

Load balancing control of a server network cluster

Author:
Miquel TALAVERA

Supervisor:
Dr. Carolina ALBEA

A Master Thesis submitted for the degree of
Industrial Engineering

July 2014

ETSEIB

I certify I have read this Master Thesis and that in my opinion it is fully adequate, in scope and in quality, as a project for the degree of Industrial Engineering.

Associate Prof. Dr. Carolina Albea Sánchez (Supervisor)

Date

Abstract

FACED with increasing network services and number of users, requests to the servers at those sites has significally skyrocketed. Moreover, most of these servers need to run twenty-four hours a day, 7 days a week with a high reliability and availability. Consequently, the tremendous growth of the Internet has led the requirement of multi-server structures in order to deal with these effectiveness issues. A much higher processing power may be provided by a set of computational elements than by a single one, even if it presents a powerful capacity. Additionally, the global system throughput may greatly increase by using properly these architectures. However, to make an efficient server network is a difficult task. This is the main goal of this dissertation.

Efficiency may be increased if server network works in cooperation, distributing the load. This is known as "load balancing". This technique may make that network is robust and efficient, that means, overload are avoided at the same time that system resources are well exploited.

Assuming that all servers present a same architecture, a deterministic dynamical model is designed and a distributed control law inspired by consensus theory is developed. In this way, the closed-loop ensures load balancing as well as asymptomatically stability. Moreover, thanks to decentralized control, computational load is reduced and scalability is possible. On the other hand, an attraction domain is estimated to ensure positive rates.

The effectiveness of the distributed control is validated by simulations in Matlab & Simulink and Network Simulator 2.

Key words: load balancing, consensus control theory, Lyapunov stability server network cluster, attraction domain.

Résumé

FACE à l'augmentation des services de réseau et le nombre d'utilisateurs, les performances des serveurs Web ont fortement augmenté. En effet, la plupart de ces serveurs doivent fonctionner 24 heures par jour, 7 jours par semaine avec une grande fiabilité et disponibilité. Par conséquent, la grande croissance de l'Internet a conduit à l'exigence de structures multiserveurs pour faire face à ces problèmes d'efficacité. Une puissance de traitement beaucoup plus élevé peut être fournie par un ensemble d'éléments de calcul que par un seul, même dans le cas qu'elle présente une grande capacité. De même, le débit du système global peut augmenter considérablement en utilisant correctement ces architectures. Cependant, pour faire un réseau de serveur efficace est une tâche difficile. C'est l'objectif principal de ce rapport.

L'efficacité peut être augmentée si le réseau du serveur fonctionne en coopération, à l'heure de distribuer la charge. Ceci est bien connu comme "équilibrage de charge". Cette technique permet de créer un réseau robuste et efficace, c'est-à-dire, les surcharge sont évités en même temps que les ressources du système sont utilisées de façon optimale.

En considérant que tous les serveurs présentent une même architecture, un modèle dynamique déterministe est conçu et un droit de contrôle distribuée inspirée par la théorie du consensus est développée. De cette façon, la boucle fermée assure une répartition équilibrée de la charge et assure une stabilité asymptotique. En effet, grâce à un contrôle décentralisé, la charge de calcul est réduite et permet au système d'être scalable. D'autre part, un domaine d'attraction est estimé pour assurer des débits positifs.

L'efficacité du contrôle distribué est validée par des simulations en Matlab & Simulink et Network Simulator 2.

Mots clés : équilibrage de charge, théorie du contrôle par consensus, stabilité de Lyapunov dans un groupe de serveurs, domaine d'attraction.

Resumen

Ante el gran incremento de los servicios de red y del número de usuarios, la demanda de prestaciones de los servidores web se ha multiplicado. Además, la mayoría de estos servidores deben funcionar con total disponibilidad y fiabilidad venticuatro horas al día, siete días a la semana. Este crecimiento de Internet ha dado lugar a una nueva estructura computacional, "los clústeres". Con un grupo de servidos se puede proporcionar una potencia de procesamiento superior que con uno solo, aunque este sea muy potente. Asimismo, usando correctamente estas arquitecturas, se puede aumentar significativamente el rendimiento global del sistema. Sin embargo, no es tan fácil utilizar eficientemente una red de servidores. Este es el objetivo principal de este trabajo.

Si los servidores trabajan cooperativamente distribuyendo equilibradamente la carga de trabajo se puede incrementar notablemente la eficiencia del sistema. Esta técnica se conoce como "Load balancing". El balanceo de carga permite crear una red robusta y eficiente, es decir, asegura que no haya perdidas de paquetes (sobrecargas) además de garantizar que los recursos del sistema se estén utilizando óptimamente.

Suponiendo que todos los servidores presentan las mismas prestaciones, se ha desarrollado un modelo dinámico-determinista así como una ley de control distribuido inspirada en la teoría de consenso. De esta manera, el lazo cerrado asegura la distribución equilibrada de la carga y garantiza estabilidad asintótica. Gracias a un control descentralizado, la carga computacional se reduce y permite que el sistema sea escalable. Finalmente, se ha llevado a cabo un estudio del dominio de atracción para garantizar que los flujos de datos sean positivos.

Se ha validado el modelo mediante simulaciones en Matlab & Simulink y Network Simulator 2.

Palabras clave: balanceador de carga, control por consensus, estabilidad de Lyapunov en un clúster, dominio de atracción.

Resum

DAVANT del gran increment dels serveis de xarxa i del nombre d'usuaris, la demanda de prestacions dels servidors web s'ha multiplicat. A més, la majoria d'aquests servidors han de funcionar amb total disponibilitat i fiabilitat vint-i-quatre hores al dia, set dies a la setmana. El creixement d'Internet ha introduït una nova estructura computacional, "els clústers". Amb un grup de servidors es pot proporcionar una potència de processament superior a la d'un de sol, encara que sigui molt potent. Utilitzant correctament aquestes estructures es pot augmentar significativament el rendiment global del sistema. Tanmateix, no és tan fàcil utilitzar eficientment una xarxa de servidors. Aquest és l'objectiu principal d'aquest treball.

Es pot incrementar notablement l'eficàcia del sistema si els servidors treballen cooperativament i distribuint equilibradament la càrrega de treball. Aquesta tècnica s'anomena "Load balancing". El balanceig de càrrega permet crear una xarxa robusta i eficient, és a dir, assegura que no hi hagi pèrdues de dades (sobrecàrregues) i garanteix que els recursos del sistema s'estiguin utilitzant òptimament.

Suposant que tots els servidors tenen les mateixes característiques, s'ha desenvolupat un model dinàmic i determinista així com una llei de control distribuït inspirada en la teoria de consensus. D'aquesta manera, el llaç tancat assegura la distribució equilibrada de la càrrega i garanteix estabilitat asimptòtica. Gràcies al control descentralitzat, la càrrega computacional disminueix i permet que el sistema sigui escalable. Finalment, s'ha procedit a estudiar el domini d'atracció per garantir que els fluxos de dades siguin positius.

S'ha validat el model mitjançant simulacions en Matlab & Simulink i en Network Simulator 2.

Paraules clau: balancejador de càrrega, control per consensus, estabilitat de Lyapunov en un clúster, domini d'atracció.

I would like to dedicate my master thesis to my parents,
who always stood beside me in everything I did.

Acknowledgements

I gratefully thank my supervisor Associate Prof. Dr. Carolina Albea for her guidance, encouragement and support throughout this master thesis work. Working under her supervision has been a great pleasure. I appreciate her vast knowledge and skill in the automatic area and her assistance in writing articles and reports. I have learned a lot from her.
I also would like to thank Prof. Dr. Yann Labit and Associate Prof. Dr. Carolina Albea for giving me the opportunity to work in the LAAS-CNRS.

Further, I would like to thank my colleagues in the laboratory and especially to Sergine, Sergi, Anouar, Denis, Cedric, Guillaume, Lionel, Wael, Maxence and Josú for their warm welcome, their support, their encouragement and their valuable suggestions.

I am grateful to my family and friends, especially to Marta, who have been always my supporters. I am thankful to them for their sacrifices.

Finally, I would like to thank the CNES project for the funding of this work.

Miquel Talavera Foix

Contents

List of Figures

Chapter 1

Introduction

1.1 Dynamic load balancing in distributed systems

The demand for high performance computing is increasing everyday. Usually, a single server responding to all incoming requests for a website might not be able to handle all the incoming traffic. As a result, pages will load very slowly and users will have to wait for a long time to view website or users will not view the site-web, at all, due to losses of requests. The computational power offered by a single computer is not sufficient for treating all internet requests. In addition, this issue may be a problem for many investigations areas [14]. Interconnecting a set of distributed computational elements through a shared network, and employing them in a effective way, makes possible to achieve performance not ordinarily reachable on a single computational element [5]. Actually, a large-size server network cluster has power comparable to a supercomputer, but a much lower cost. Moreover, there is another advantage: the server scalability improvement, what makes the network more reliable. That is why nowadays their use is heavily increasing.

A typical distributed system will have a number of servers working independently with each other. Each server contains an initial load, which represents the requests to be treated, and each one may have a different processing capacity. For robustness and security reasons, requests are evenly, fairly and effectively distributed over all processors, as a common parallel computer architecture [27]. For instance, no single server must be overburdened while others remain in a idle state. Any strategy for load distribution is called load balancing.

Load balancing can be performed through a decentralised controller. For this purpose, knowledge of the state of each individual server is needed. This knowledge is used to judiciously assign incoming computational requests to suited server, according to some load-balancing policy. Nevertheless, several features of the parallel computation environment should be captured. These include:

1. Incoming requests and throughputs.
2. Workload awaiting processing at each server (queue size).

3. Performance of each servers.

4. Computational requirements of each workload component.

5. Delays and bandwidth constraints of servers and network components involved in the exchange of workloads.

6. Delays imposed by servers and the network on the exchange of measurements and information.

In large-scale distributed computing systems where servers are physically or virtually far from each other, there are a number of inherent time-delay factor that can seriously change the expected performance of load balancing [13], [12]. In this work we are not going to consider delays. This issue will be take into account in a future work.

Another issue related to load-balancing is the variation of processing information in each server. The processor capacity may be different from each other in architecture, operation system, CPU speed, memory size, and available disk space. The load-balancing problem also needs to consider fault-tolerance and fault-recovery (consensus theory) [23]. This is another expected work. With all these factors taken into account, load-balancing can be generalized into four basic steps:

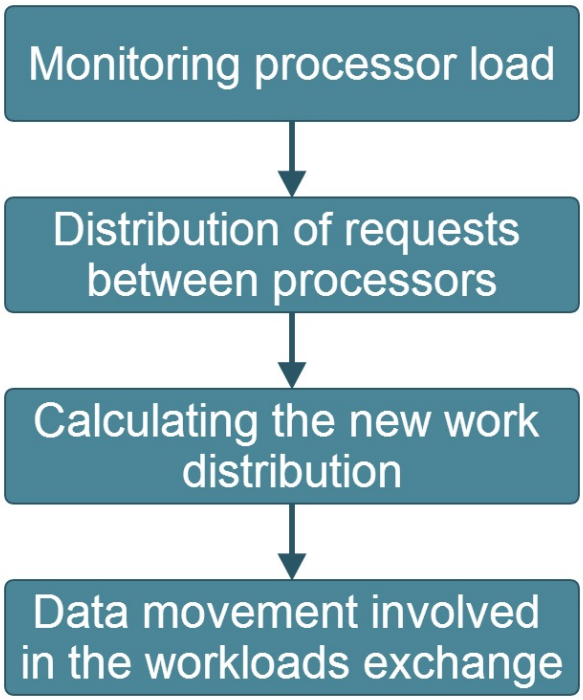


Figure 1.1: Load balancing steps.

Several taxonomies of dynamic load balancing strategies exist. This is the approach used in most of the existing load balancing solutions [16]. A comparison of various deter-

ministic methods is given in [30]. A discussion between static and dynamic load balancing policies may be found in [15].

1.2 Contributions of this dissertation

This work proposes a strategy for load-balancing in a server network cluster. A deterministic dynamical model and a distributed control law inspired by consensus theory has been performed. Its approach relies on a relevant dynamic model following deterministic control theory specifically in the theory of multi-server systems [21].

Some assumptions are performed in order to propose a first solution. For instance, it is assumed that all server present the same architecture. Likewise, request and throughput are considered known and constant. The work is focused on an average model. Network present an uniform bandwidth. Moreover, delays and server heterogeneity architecture is not taken into account in this dissertation, as mentioned before.

The main goal in this report is not only to ensure that there is not losses of requests, but also that the queues of each sever asymptomatically converge to an average level. The model is shown to be globally stable if the rate-limiter is not saturated. Therefore, the system must remain within the boundaries of a pre-specified admissible "safe" region. Otherwise, we lose global stability. Then, an estimation of an attraction domain has been performed in order to estimate a set of initial conditions which ensures that we are in the logical environment, for instance, negative traffics are avoided. Stability of dynamic model is guaranteed by using Lyapunov's direct method. A power comparison has been realised to prove that the model implementation do not suppose a high increment of energy consumption (green networking).

Finally, the load balancing control has been tested in a multi-paradigm numerical computing environment (Matlab & Simulink) and in a discrete event network simulator (Network Simulator 2). The traffic flow in Matlab-Simulink's simulations was considered to be as water flow. The servers were assumed to be water tanks. On the other hand, a Constant Bit Rate (CBR) flown and the Internet protocol Use Datagram Protocol (UDP) were used in NS2's simulations.

1.3 Presentation of LAAS-CNRS

Laboratory for Analysis and Architecture of Systems[1] is a research unit of the Centre National de la Recherche Scientifique[2], the French National Centre for Scientific Research.

[1]LAAS' oficial website: http://www.laas.fr/
[2]CNRS' oficial website: http://www.cnrs.fr/

LAAS conducts pluridisciplinary research in the areas of Information and Systems Sciences and Technologies, within four domains:

- Micro and Nano Technologies
- Automatic control, Optimization and Signal Processing
- Critical Information Systems
- Robotics and Artificial Intelligence

LAAS' research activities fall within the domain of Information Sciences and Technologies and address complex systems (artificial and sometimes natural) generally heterogeneous, and at different scales, to devise theories, methodologies and tools for modeling, designing and controlling them. Research, innovation and transfer are tied. The lab has a history of strong relationships with industry and works in a large number of collaborative projects with international, national and regional industries of all size. LAAS was one of the 20 first "Carnot Institutes"[3] labeled in 2006.

It employs more than 650 persons (202 researchers and faculty members, 262 PhD students, 68 post-docs and visiting researchers, 122 engineers and technicians and a larg number of stagiaires). It divulge 900 annual publications per year and its annual operational budget is around 16 M€.

1.3.1 Team SARA

The laboratory is structured in 21 research teams, defined by a set of scientific subjects and objectives and composed by researchers pursuing them. SARA research team's work addresses new generation networks and communicating systems and their applications. Our research studies address the design, the planning, the deployment management and the monitoring. Our contributions include the development of methods, models and tools as well as the design and implementation of architectures, protocols and services. In particular, our research work focuses on analysis, performance evaluation, control and prototyping of software and communication platforms. The research challenges mainly include mastering:

- Modeling and controlling scalable dynamic systems
- Designing systems with strong requirements (QoS, security) and highly constrained (energy, resources)
- Handling continuity of services and their quality in mobile networks
- Elaborating dynamic and autonomous service-oriented and components-based software architectures

[3]Oficial Carnot Institutes' website: http://www.instituts-carnot.eu/fr/

1.4 Organization of the dissertation

The report is organized as follows. In Chapter 2, it is presented an overview of existing load-balancing strategies. A briefly discussing on the different schemes is illustrated. Chapter 3 exposes the problem formulation. It is also introduced the dynamic load balancing model. Then, a description of the control law based on a consensus algorithm is proposed in Chapter 4. Chapter 5 presents the simulations results in Matlab - Simulink and in Network Simulator 2. Finally, Chapter 6 draws some conclusions and future work.

Chapter 2

State of art. Load balancing policies

In this chapter, a brief overview of the different taxonomies of load-balancing policies are defined, followed by an overview of previous work in the field, see for instance [6]. In the last section is described a brief introduction of the distributed control performed in the LAAS-CNRS.

2.1 Brief overview of taxonomy of load balancing policies

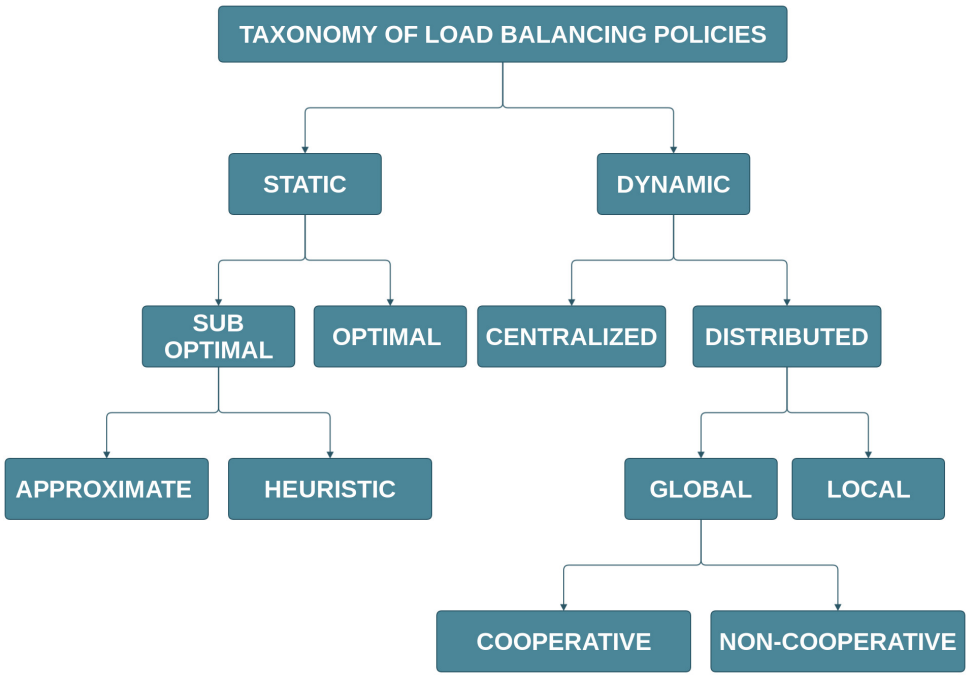


Figure 2.1: Load balancing taxonomy.

Firstly, a discussion about different taxonomies is given in the first part of this chapter. Figure 2.1 shows the organisation of the different load-balancing schemes that are described below.

2.1.1 Static versus dynamic

Static load distribution, also known as deterministic scheduling, assigns a given job to a fixed server or node. Every time the system is restarted, the same binding task-server (allocation of a task to the same server) is used without considering changes that may occur during the system's lifetime. Moreover, static load distribution may also characterize the strategy used at runtime, in the sense that it may not result in the same task-server assignment, but assigns the newly arrived jobs in a sequential or fixed fashion. For example, using a simple static strategy, jobs can be assigned to nodes in a round-robin fashion so that each server executes approximately the same number of tasks. Dynamic load-balancing takes into account that system parameters may not be known beforehand and therefore using a fixed or static scheme will eventually produce poor results. A dynamic strategy is usually executed several times and may reassign a previously scheduled job to a new node based on the current dynamics of the system environment.

2.1.2 Distributed versus centralized

This division usually falls under the dynamic load-balancing scheme where a natural question arises about where the decision is made. Centralized policies store global information at a central location and use this information to make scheduling decisions using the computing and storage resources of one or more servers. This scheme is best suited for systems where an individual server's state information can be easily collected by a central station at little cost, and new jobs arriving at this centralized location are then redirected to subsequent nodes. The main drawback of this scheme is that it has a single point of failure. In distributed scheduling, the state information is distributed among the nodes that are responsible in managing their own resources or allocating tasks residing in their queues to other servers. In some cases, the scheme allows idle servers to assign tasks to themselves at runtime by accessing a shared global queue. Note that failures occurring at a particular node will remain localized and may not affect the global operation of the system. Another scheme that fits between the two types above is the hierarchical one where selected nodes are responsible for providing task scheduling to a group of servers. The nodes are arranged in a tree and the selected nodes are roots of the subtree domains.

2.1.3 Local versus global

Local and global load-balancing fall under the distributed scheme since a centralized scheme should always act globally. In a local load-balancing scheduling, each server polls other servers in its neighbourhood and uses this local information to decide upon a load transfer. This local neighbourhood is usually denoted as the migration space. The primary objective

is to minimize remote communication as well as to efficiently balance the load on the servers. However, in a global balancing scheme, global information of all or part of the system is used to initialize the load-balancing. This scheme requires a considerable amount of information to be exchanged in the system which may affect its scalability.

2.1.4 Cooperative versus non-cooperative

Within the realm of distributed dynamic global scheduling, two mechanisms can be distinguished involving the level of cooperation between the different parts of the system. In the non-cooperative or autonomous scheme, each node has autonomy over its own resource scheduling. That is, decisions are made independently of the rest of the system and therefore the node may migrate or allocate tasks based on local performance. On the other hand, in cooperative scheduling, processes work together toward a common system-wide global balance. Scheduling decisions are made after considering their effects on some global effective measures (for example, global completion time).

2.1.5 Adaptive versus non-adaptive

Adaptive and non-adaptive schemes are part of the dynamic load-balancing policies. In an adaptive scheme, scheduled decisions take into consideration past and current system performance and are affected by previous decisions or changes in the environment. If one (or more parameters) does not correlate to the program performance, it is weighted less next time. In the non-adaptive scheme, parameters used in scheduling remain the same regardless of system's past behaviour. An example would be a policy that always weighs its inputs the same regardless of the history of the system behaviour. Confusion may arise between in distinguishing dynamic scheduling and adaptive scheduling. Whereas a dynamic solution takes environmental inputs into account when making its decision, an adaptive solution (which is also dynamic) takes environmental stimuli into account to modify the scheduling policy itself.

2.1.6 One-time assignment vs dynamic reassignment

In this classification, the entities to be scheduled are considered. The one-time assignment of a task may be dynamically done but once it is scheduled to a given server, it can never be rescheduled to another one. On the other hand, in the dynamic reassignment process, jobs can migrate from one node to another even after the initial placement is made. A negative aspect of this scheme is that tasks may endlessly circulate about the system without making much progress.

2.1.7 Sender/Receiver/Symmetrical initiated

Techniques of scheduling tasks in distributed systems have been divided mainly into sender-initiated, receiver-initiated and symmetrically-initiated. In sender-initiated algorithms,

the overloaded nodes transfer one or more of their tasks to several under-loaded nodes. In receiver-initiated schemes, under-loaded nodes request tasks to be sent to them from nodes with higher loads. In the symmetric approach, both the under-loaded as well as the loaded nodes may initiate load transfers.

2.2 State of the art about load balancing strategies

In this section, several load balancing strategies introduced in early works are described. This policies were compared by Banawan and Zeidat in [4].

Many aspects of servers systems performance can be investigated using queueing network models. In using these models, hardware resources are represented by service centres at which jobs may queue and compete for service. Depending on the goal of the study, different levels of details may be included in such a model. The level of details determines the complexity of the model. In order to study the performance of load sharing in heterogeneous multicomputer systems we use the Multiple Queue Multiple Server (MQMS) model shown in Figure 2.2. Each server represents a node in the system whose queue corresponds to the ready queue at that node. The workload is modeled as a single stream of jobs. Each newly arrived job is assigned to some node according to the scheduling policy used. This model is more realistic than the Single Queue Multiple Server (SQMS) that was used in other studies [25] [20].

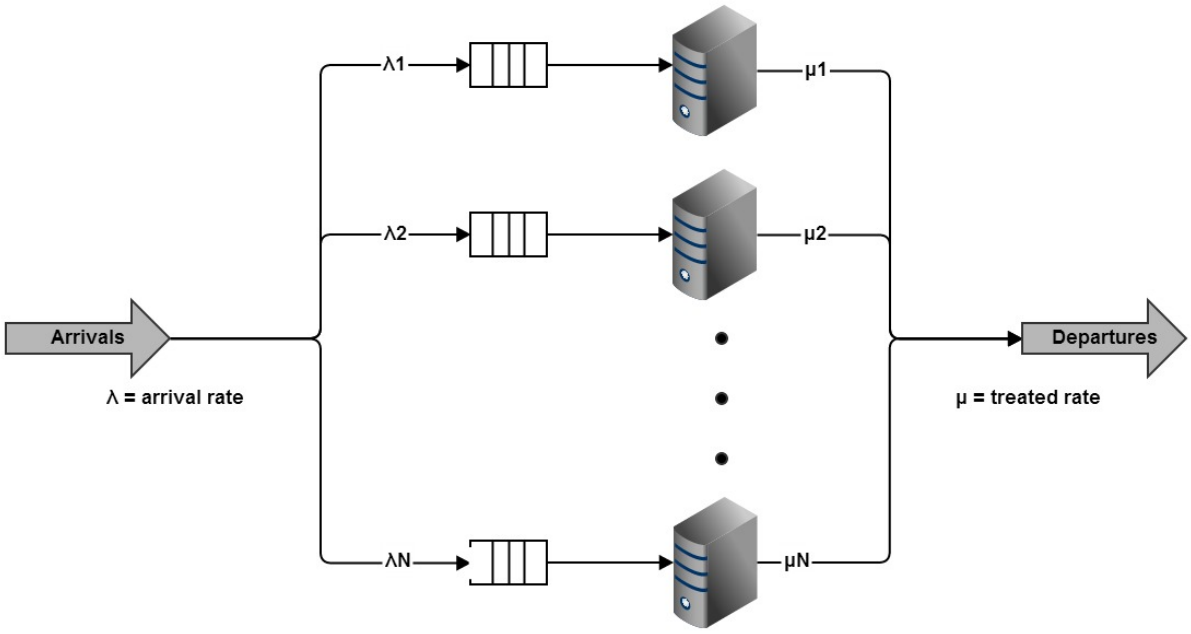


Figure 2.2: Multiple Queue Multiple Server Model.

Consider the simple heterogeneous multiple servers system model shown in Figure 2.2.

Each server is modeled as a queue/server pair. This is a typical open queueing model for N independent servers. Server i is assumed to be characterized completely by its mean service rate μ_i. Thus, the total service rate of the system is $\mu = \sum_{i=1}^{N} \mu_i$. Jobs enter the system at mean rate λ. Their interarrival times may be modeled as a sequence of independent identically distributed random variables. The distribution may be assumed to be exponential with a mean of $1/\lambda$. The instantaneous state q of the system is a snapshot of the workload distribution among the system components and can be expressed as $q = (q_1, q_2, \ldots, q_N)$, where q_i is the number of jobs in the queue i including the job in service. An arriving job is routed to server i with probability p_i where the routing decision is based on the outcome of an independent trial. Since one of the N servers must be chosen, $\sum_{i=1}^{N} p_i = 1$. Each server manages its queue of jobs in first-in-first-out priority order. The processing time of a job on server i may be modeled as an independent exponentially distributed random variable with mean $1/\mu_i$.

As mentioned earlier, there are two main approaches to load sharing, namely static and dynamic policies. Static policies use fixed branching probabilities according to which jobs are assigned to the nodes. They are simple, easy to implement and have minimal runtime overhead. However, they lack the flexibility of making scheduling decisions based on the current state of the system which can be used to alleviate transient congestions, in addition to any long term imbalance. A brief description of these policies is given below.

2.2.1 Some static policies

Speed-Weighted random splitting policy (SWS) [7]

According to SWS, a new arrival is assigned to node i with probability p_i given by:

$$p_i = \frac{\mu_i}{\mu},$$

where μ_i is the service rate of node i and μ is the total service rate of the system. See Figure 2.2.

Load-Dependent static policy (LDS)

The LDS policy uses branching probabilities p_k that minimize the response time averaged over all jobs executed by the system. These branching probabilities are obtained by solving the following nonlinear optimization problem:

$$\min_{p_i} R = \sum_{i=1}^{m} \frac{p_i}{\mu_i - \lambda p_i},$$

subject to the constraints:

$$\sum_{i=1}^{m} p_i = 1,\ \mu_i > \lambda p_i,\ i = 1, \ldots, m,$$

where λ is the aggregate arrival rate to the system and μ_i is the service rate of node i. See Figure 2.2.

2.2.2 Some dynamic policies

Shortest queue policy (SQ)

This policy assigns each new arrival to the node that currently has the least number of jobs in its queue. Winston [31] established that this policy is optimal for homogeneous system under certain assumptions.

Shortest expected delay (SED) [3]

The SED policy, based on the MQMS model, can be either centralized where new tasks arrive to a central server and then assigned to subsequent nodes, or distributed where each available node can insert new jobs into the system. A cost function is evaluated for each node and the job is sent to the corresponding node that produces the minimum cost. The cost $SED(i)$ is actually the expected time to complete the new job at server i and is given by:

$$SED(server_i) = \frac{q_i + 1}{\mu_i},$$

where q_i is the queue length at node i and μ_i is its service rate of node i.

Adaptive separable policy (AS) [29]

Adaptive separable policy (AS), also based on the MQMS model, is an improvement over the SED policy in that it estimates the completion time of a new arrival at a node by adjusting the service rate of the server based on its utilization. It uses the cost function:

$$AS(server_i) = \frac{q_i + 1}{\mu_i \rho_i},$$

where ρ_i is the utilization of node i, q_i is the queue length at node i and μ_i is its service rate of node i.

Never queue policy (NQ) [24]

NQ policy does not allow a job to wait in a queue if there is an idle node available. If more than one idle node is available, the new job is sent to the fastest node that has the largest $1/\mu_i$ term. On the other hand, if all nodes are busy, the SED policy is used.

Maximum throughput policy (TP)

The TP policy schedules the next arrival to the node that maximizes the system throughput during the next interarrival period. Chow and Kohler [7] developed the following expression for the system throughput during the next interarrival period:

$$TP(q_1, q_2, \ldots, q_m) = \sum_{i=1}^{N} \lambda[\sum_{k=1}^{q_i-1}(1 - \frac{q_i}{k})(\frac{\mu_i}{\lambda + \mu_i})^k - q_i ln(\frac{\lambda}{\lambda + \mu_i})],$$

where λ is the aggregate arrival rate to the system, q_i is the queue length at node i and μ_i is its service rate of node i. See Figure 2.2.

TP can be thought of as a reward function that depends on the queue lengths of all nodes. It is evaluated for each possible assignment and the new job is assigned to the one that yields maximum reward.

Greedy throughput policy (GT)

Similar to the TP policy, GT policy aims at maximizing system throughput. However, it uses another reward function that was derived by Nelson and Towsley [9]. A new job is assigned to the node that maximizes the reward function:

$$GT(serveri) = (\frac{\mu_i}{\mu_i + \lambda})^{q_i+1},$$

where λ is the aggregate arrival rate to the system, q_i is the queue length at node i and μ_i is its service rate of node i. See Figure 2.2.

The gradient model (GM)

In Gradient Model (GM) [18], the underloaded nodes notify the other nodes about their state, and overloaded nodes respond by transmitting jobs to the nearest lightly loaded node. Therefore, loads migrate in the system in the direction of the underloaded nodes guided by the proximity gradient. A global balance state is achieved computationally by successive localized balances.

At every step of the algorithm, each node compares its load to a Low-Water Mark (LWM) and a High-Water Mark (HWM) thresholds. The node is set to the underloaded state if it has a load less than LWM and to the overloaded state if it has a load greater than HWM. Underloaded nodes set their proximity to zero and all other nodes p set their proximity according to:

$$proximity(i) = min(proximity(n_k)) + 1,$$

where n_k denote the neighboring nodes of node i. The node's proximity is defined as the shortest distance from itself to the nearest lightly loaded node in the system.

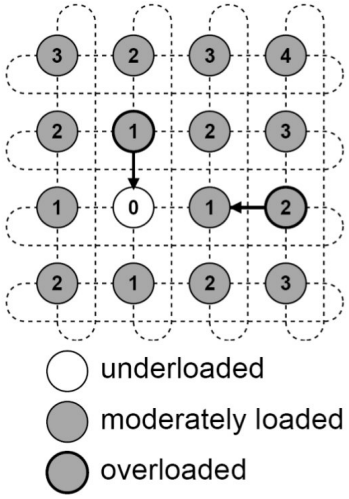


Figure 2.3: Gradient model example.

Subsequently, all overloaded nodes send a fraction δ of their loads in the direction of the lowest proximity. The algorithm is illustrated in Figure 2.3.

Note that no measure of the degree of unbalance is found using this algorithm, but only that one exists. When an imbalance occurs, the number of excess tasks can only be known to be greater than HWM-LWM. Hence, the HWM, LWM, and the fraction δ parameters have a critical impact on the stability and performance of the algorithm and should therefore be wisely chosen.

The gradient model policy cannot be used in distributed systems since the nodes are not connected in a certain topology such as a mesh or hypercube. This fact renders the proximity concept useless. Moreover, the proximity algorithm is a cascading function and therefore requires a considerable amount of time to be evaluated in large-scale networks where delays are prominent.

However, a modification to the algorithm may be suitable for P2P networks such as Freenet where nodes are only aware of their immediate neighbors. Consequently, the proximity concept becomes valid and the algorithm may become useful.

Sender (Receiver) initiated diffusion (SID/RID) [8]

The SID policy is a highly distributed local approach which makes use of near-neighbor load information to apportion surplus load from heavily loaded servers to underloaded

neighbours in the system. Global balancing is achieved as tasks from heavily loaded neighbourhoods diffuse into lightly loaded areas in the system. It is a local, near-neighbour diffusion approach which employs overlapping balancing domains to achieve global balancing. The scheme is purely distributed and asynchronous. Each server acts independently, apportioning excess load to deficient neighbours.

For the SID policy, the balancing process is triggered whenever a node i receives from a neighboring node k a load update l_k less than a preset threshold $L_{low}(l_k < L_{low})$. After that, the node i proceeds by calculating the domain load average $\bar{L}_i$.

$$\bar{L}_i = \frac{1}{K+1}(l_i + \sum_{k=1}^{K} l_k),$$

where K is the number of neighboring nodes. The load balancing algorithm continues if the local excess load $(l_i - \bar{L}_i)$ is greater than a preset threshold $L_{threshold}$. Load δ_k is then transferred from node i to each neighbor in proportion to its deviation from the domain calculated using:

$$h_k = \begin{cases} \bar{L}_i - l_k & \text{if } l_k < \bar{L}_i, \\ 0 & \text{otherwise.} \end{cases}$$

$$\delta_k = (l_i - \bar{L}_i)\frac{h_k}{\sum_{k=1}^{K} h_k}$$

The RID policy is the converse of the SID strategy, where underloaded servers requisition load from heavily loaded neighbours. However, to avoid instability due to delays and aging in the load exchange information, the overloaded nodes transmit tasks up to the half of their current load. SID and RID are illustrated in Figure 2.4.

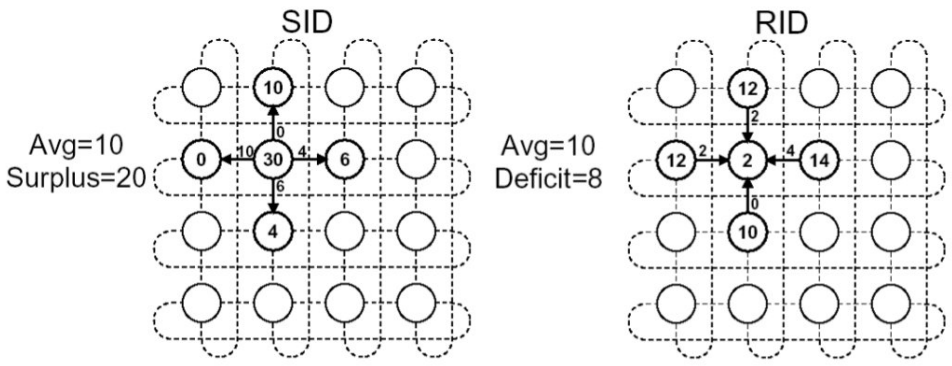


Figure 2.4: SID and RID examples.

This scheme is distributed, asynchronous, and topology independent as opposed to the GM policy that is best suited for nodes arranged in a hypercube or mesh fashion. However,

the same problem arises in defining the local domain where the balancing process should take place. Moreover, as indicated earlier, the domains should overlap to an extent that is sufficient to achieve global balancing.

The algorithms implemented in this thesis are based on the SID scheme without restricting the balancing process to a local domain, but rather expanding it to the global system.

Hierarchical Balancing Method (HBM)

The Hierarchical Balancing Method (HBM) strategy [30] arranges the nodes in a hierarchy, thereby creating balancing domains at each level. For a binary tree organization, all nodes are included at the leaf level (level 0). Half the nodes at level 0 become subtree roots at level 1. Subsequently, half the nodes again become subtree roots at the next level and so forth until one node becomes the root of the whole tree.

Global balancing is achieved by ascending the tree and balancing the load between adjacent domains at each level in the hierarchy. If at any level, the imbalance between the left and right subtrees exceeds a certain threshold, each node in the overloaded subtree sends a portion of its load to the corresponding node in the underloaded subtree.

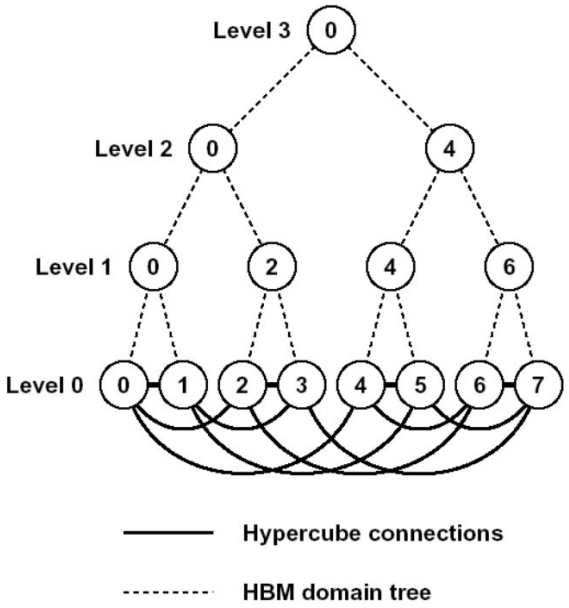


Figure 2.5: Hierarchical organization of 8 server with hypercube interconnections.

The advantage of the HBM scheme is that it minimizes the communication over- head and therefore can be scaled to large systems. Moreover, the policy matches hypercube topologies well. In fact, the dimensional exchange approach [8] designed for hypercube

systems is similar to the HBM method in the sense that it proceeds by load-balancing per domain basis. Here, each domain is defined as one dimension in the hypercube. The hierarchical organization of an eight-server hypercube is shown in Figure 2.5.

This scheme is clearly not suitable for systems with large network delays for the following reasons. As the balancing process proceeds on to the next level in the tree, critical changes occurring at lower levels may not propagate quickly due to delays. Therefore, corrections may not reach higher domains in time and may thereby result in an imbalance at the global level. Moreover, although the scheme is decentralized, a failure at root nodes especially at high levels in the tree, renders a global balance state unattainable. Consequently, this scheme is not suitable for Internet-scale distributed systems since nodes may become unreachable at any time, and will therefore affect the balance state of the system if such nodes happen to be roots for subtree domains.

Dimension exchange method (DEM) [8], [10]

The DEM strategy is similar to the HBM scheme in that small domains are balanced first and then combine to form larger domains until ultimately the entire system is balanced. This differs from the HBM scheme in that it is a synchronized approach. The DEM strategy was conceptually designed for a hypercube system but may be applied to other topologies with some modifications. In the case of an N server hypercube configuration, balancing is performed iteratively in each of the $log\ N$ dimensions. All server pairs in the first dimension, those servers whose addresses differ in only the least significant bit, balance the load between themselves. Next, all server pairs in the second dimension balance the load between themselves, and so forth, until each server has balanced its load with each of its neighbors.

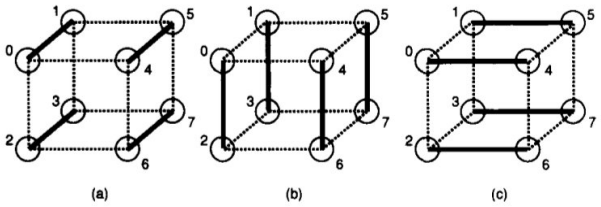


Figure 2.6: DEM strategy.

The scheme is illustrated in Figure 2.6 for an eight-server hypercube. The strategy could be extended to an MxM mesh topology by "folding" the mesh in each dimension $[log\ M]$ times. In this case, server pairs would no longer be directly linked to one another and communication costs would be higher.

Performing the balancing steps in a synchronized manner ensures that the entire system will achieve a balanced load. Balancing is initiated by any underloaded server which has a

load that drops below a preset threshold:

$$L_p < L_{threshold}$$

This server broadcasts a load balancing request to all other servers in the system. Global synchronization is particularly difficult in highly parallel systems where a global broadcast from a single point may be costly. Large systems equipped with special broadcast mechanisms may, however, be suited to this approach. This scheme has theoretically been shown to outperform a synchronous diffusion approach in terms of the overhead incurred to reach a uniform distribution from an unbalanced state. The theoretical analysis does not include the synchronization overhead to initiate the balancing process.

2.3 Load balancing by automatic control

The computational load of the strategies described above is rather high. In order to efficiently assign a new load arrival, these policies must realise a new calculus to allocate the task in the server that best suits. On the other hand, automatic control methods relies on managing the input rate in each server. Schedule decisions take into account current and past system performance. In addition, processes should work together toward a common system-wide global balance, so that global information of all of the system is used to initiate the load-balancing. Therefore, the computational load is reduced and scalability is achieved.

Chapter 3

Load balancing control

This report proposes a dynamic and distributed control strategy for activity management of network servers. Our approach relies on performing a control law which is adaptive, cooperative, global and that satisfies dynamic reassignments. In this section it is described the queue management problem of a set of servers that treat external requests. Then, a distributed dynamic and deterministic control law has been performed.

3.1 Problem formulation

Consider a system composed by a source S that distributes the request rate A among a set of N servers (see Figure 3.1). The treated requests rate is Y, and it is assumed to be the computational power of each server. It is supposed to be constant and prespecified. The main goal of the servers is to perform the input requests rate. The input rate and the output rate are as follows:

$$A = [a_1 \ a_2 \ \dots \ a_{N-1} \ a_N]^T$$

$$Y = [y_1 \ y_2 \ \dots \ y_{N-1} \ y_N]^T.$$

Assumption 1 *The overall workload of the system is conserved:*

$$\sum_{i=1}^{N} a_i = \sum_{i=1}^{N} y_i.$$

To perform suitable processing of the workload coming from the source, we equip each agent with a queue which is able to locally store the requests arriving form the source and still not processed. Consequently, we introduce for each $i \in \mathbb{N}$ the state queue length $q_i \in \Re_{\geq 0}$, whose variation $\dot{q}_i$ corresponds to the difference between the requested rate coming form the source and the maximum processing rate y_i. Note that special care has to be

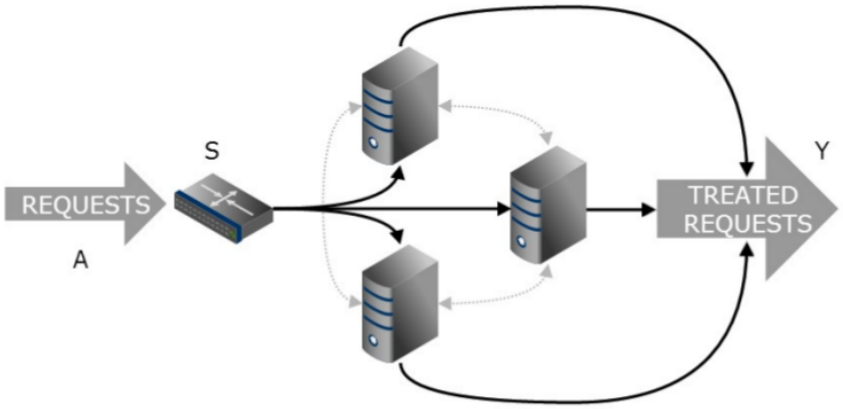


Figure 3.1: The considered system with $N = 3$ servers.

taken that any solution $t \mapsto q_i(t)$ to the proposed dynamic satisfies $q_i(t) \in \Re_{\geq 0}$ for all $t \geq 0$.

In any rate, it is supposed that the servers have different initials loads:

$$q_0 = [q_{0_1} \; q_{0_2} \; \cdots \; q_{0_{N-1}} \; q_{0_N}]^T. \tag{3.1}$$

Thereby, the model in compact form is:

$$\dot{q} = A - Y + u, \tag{3.2}$$

where u is the control law that balance the load.

Problem 1 *Our aim is to evenly distribute the workload of the system* (3.1)-(3.2).

Remark 1 *Once load balancing is got, simulations for a more general case like time varying incoming rates $A(t)$, will be performed.*

3.2 Consensus Control

3.2.1 Connection graph among servers

According to the activities of the servers, the adjacent matrix is defined as:

$$Ad = [ad_{ij}] = \begin{cases} 1 & \text{if } i \neq j, \\ 0 & \text{if } i = j, \end{cases}$$

and the diagonal matrix Δ as,

$$\Delta = \begin{cases} 0 & \text{if } i \neq j, \\ \sum_{k \in \mathbb{N}, k \neq i} ad_{ik} & \text{if } i = j. \end{cases}$$

Therefore, the Laplacian matrix representing the undirected graph is given by:

$$L = \Delta - Ad. \tag{3.3}$$

The Laplacian matrix of an undirected graph is a symmetric positive semi-definite matrix. It is worth mentioning that the resulting Laplacian represents the complete graph among the agents like the network shown in Figure 3.1.

3.2.2 Proposed control law

The aim of this work is to ensure that the network is robust and efficient through a balanced load distribution. It is proposed a control law that assigns an input u_i to each server i to fulfil the following proposition:

Proposition 1 *All queue lengths converge to the average point:*

$$\lim_{t \to +\infty} q_i = q^*,$$

where q^* is the average point.

Using model (3.1)-(3.2) and the Laplacian matrix (3.3), we may select a dynamic distributed controller, that represents the flux exchanged between servers. The controller is defined as:

$$u = -K_p L q + q_c \tag{3.4}$$

$$\dot{q_c} = -K_i L q, \tag{3.5}$$

where K_p and $K_i \in \Re_{\geq 0}$ are the proportional and integral gains, respectively. This control refers to a classical consensus algorithm for simple integrator multi-agent systems [21], which ensures that the length of the queues converge to desirable level q^*.

In the following lemma, it is presented an important property which defines (3.4) and (3.5).

Lemma 1 *The distributed control law* (3.4) *and* (3.5) *satisfies the compete processing condition given in* (3.2)*, and all queue lengths converge to the same average level, q^*.*

Proof 1 *The proof is straightforwardly derived from* (3.3) *in* (3.4) *and* (3.5)*.*

3.2.3 Lyapunov estability

A function is defined:

$$V = q^T L q + \frac{(q_c + A - Y)^2}{K_i}. \tag{3.6}$$

The corresponding derivative is:

$$\dot{V} = 2q^T L \dot{q} + 2\frac{\dot{q}_c(q_c + A - Y)}{K_i}.$$

Using (3.2), (3.4) and (3.5), the results is:

$$\dot{V} = -2K_p q^T L q \leq 0.$$

This implies asymptotic stability.

Chapter 4

Attraction domain

4.1 Introduction

System constraints are present in almost all control applications. Actuator saturation, rate-limiters and forbidden regions in the state space, for instance, may be found in many control designs. Such constraints are typically disregarded and the resulting control law is applied to the actuator. In some situations, the global stability character may be lost, resulting in local stability of the desired operating behaviour within a bounded region of attraction [1]. In this way, if the designed control law is $u_d = \alpha(x)$, where x is the state variable, the actual control signal is $u = \gamma(u_d) = \gamma(\alpha(x))$, where γ is a saturation-like function. In fact, the local stability property is not usually affected by these constraints, since in a neighbourhood of the desired attractor they are not active, that is, $\gamma(\alpha(x)) = \alpha(x)$. Assume there is a forbidden region in the state space. This means that the system state must remain within the boundaries of a pre-specified admissible safe region. Non-linear control books contain an abundance of stability analysis examples in which constraints are not present [17], [28]. There exist various methods for estimating the region of attraction [11]. An estimation of the attraction domain of a boost inverter can be found in [2].

4.2 Method application

Assumption 2 *There is a widely known radially unbounded Lyapunov function $V(x)$, in which a compact positively invariant set Ω, $\frac{\partial V}{\partial x} f(x, \alpha(x)) \geq 0$. Let M be the largest invariant subset of the set for which $\dot{V} = 0$ en Ω.*

By the LaSalle invariance principle, assumption (2) guarantees that the trajectories of the unconstrained model converges to M. It is implicitly assumed that this is the desired behaviour. Notice that if the original Lyapunov theorem is used to prove global stability, the previous assumption is also fulfilled. Assumption (2) also guarantees local stability for the system.

The key is to ensure that the system state remains within the boundaries of the region where saturations are not active, thus introducing new constraints. A conservative estimation of the region of attraction can then easily be obtained. The idea of the method is schematically shown in Figure 4.1. The advantage is to obtain relatively easy an estimation, however, it is compromised by the fact that it may be far too conservative. Nevertheless, in many problems this simple idea may give satisfactory results.

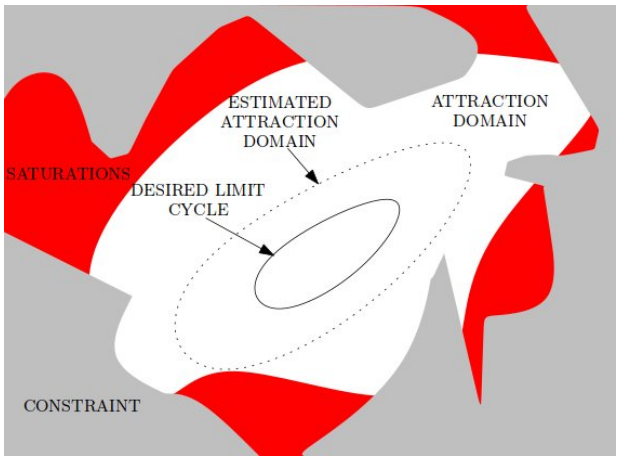


Figure 4.1: Estimated attraction domain where the constraints are not active.

The model proposed (3.2) has a rate-limiter:

$$rate_i = u_i + a_i \geq 0, \tag{4.1}$$

where $rate_i \in \Re_{\geq 0}$. Any solution $t \mapsto rate_i(t)$ has to satisfy $rate_i(t) \in \Re_{\geq 0}$ for all $t \geq 0$. This constraint is caused because the servers have a single input and a single output, so load balancing is carried out on the server entry.

Taking into account (4.1) into (3.2), a (conservative) estimation for the attraction domain of the system with constraints is given by the following theorem:

Theorem 1 *Under Assumptions 1 and 2, assume that there exists a constant $c > 0$ such that in the set $\Omega_c = \{x : V(x) \geq c^{-1}\}$, the constraints (4.1) is satisfied. Then, all trajectories of the system with constraints starting at Ω_c converges to $M \cap \Omega_c$.*

Proof 2 *Since in Ω_c the constraints are satisfied, the result for the unconstrained system are valid in Ω_c. Therefore, $\dot{V} = 0$ in Ω_c and Ω_c is positively invariant. Furthermore, since $V(x)$ is radially unbounded Ω_c is compact. The statement can be validated by applying LaSalle's invariance principle.*

Remark 2 *Since $M \cap \Omega_c \subset M$, the theorem guarantee the desired asymptotic behaviour for the system with constraints.*

Using Theorem 1, the problem is reduced to finding a value $c > 0$ such that (4.1) the points where $V(x) \geq 0$.

Problem 2 *Maximize* c^{-1}*:*

$$\begin{aligned} s.t. : V(x) - c^{-1} &\leq 0 \\ u_i + a_i &\geq 0 \quad i = 1, \ldots, N. \end{aligned} \tag{4.2}$$

Notice that the constraints are satisfied at the points on the boundary of Ω_c and in the interior of the set $V(x) - c^{-1} < 0$.

Theorem 2 *Consider the system* (3.2)*, the control law* (3.4) *and* (3.5)*, the Lyapunov function* (3.6) *and the constant* $c > 0$*, then Problem 2 is equivalent to solving:*

$$\begin{bmatrix} cY^TY & -K_pY^TL & Y^T \\ -K_pLY & Y^TYL & 0 \\ Y & 0 & \frac{Y^TYI_n}{K_i} \end{bmatrix} \geq 0. \tag{4.3}$$

Proof 3 *Defining:*

$$x_1 \triangleq q \qquad\qquad x_2 \triangleq q_c + A - Y, \tag{4.4}$$

where x_1,$x_2 \in \Re^{Nx1}$.

From equation (3.4), (3.5) *and* (4.4)*, we obtain:*

$$u = -K_pLx_1 + x_2 - A + Y \in \Re^{Nx1}. \tag{4.5}$$

Then, equation (4.1) *can be rewritten using* (4.5) *as:*

$$K_pLx_1 - x_2 \leq Y.$$

Multiplying on the left by $2Y^T$*:*

$$2Y^TY \geq 2Y^TK_pLx_1 - 2Y^Tx_2. \tag{4.6}$$

Minimizing c, multiplying on the left (4.2) *for* $2Y^TY$*, and from* (4.6)*, the attraction domain is:*

$$2Y^TY \geq cY^TYx_1^TLx_1 + cY^TY\frac{x_2^TI_nx_2}{K_i} + Y^TY \geq 2Y^TK_pLx_1 - 2Y^Tx_2.$$

Rewriting the last inequality in matrix form:

$$\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} Y^TY & -K_pY^TL & Y^T \\ -K_pLY & cY^TYL & 0 \\ Y & 0 & \frac{cY^TYI_n}{K_i} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \geq 0,$$

and applying the Schur complement, we obtain (4.3)*.*

Chapter 5

Simulations results

A five-server network cluster has been considered. Therefore, we have to define the input rate A the output rate Y (treated requests), the initial load q_0 and the controller gains K_p and K_i. We will use the following values of the parameters described above:

$$N = 5$$

$$A = [3.8\ 4.2\ 2\ 3\ 2]^T \cdot 10^3\ pkt/s$$

$$Y = [3\ 5\ 2\ 2.25\ 2.75]^T \cdot 10^3\ pkt/s$$

$$q_0 = [4500\ 4000\ 3500\ 4775\ 4225]^T\ pkt.$$

For simplicity, we take:

$$K_p = K_i = 1.$$

Thus, the constant c that maximize the attraction domain is:

$$c^{-1} = 2.9328 \cdot 10^7.$$

Using these parameters some simulations are performed: in Matlab / Simulink to validate the theory developed; and in NS2 to validate the controller in a more realistic environment.

The model developed is shown to be scalable and robust in the "safe" region. However, we can not ensure stability within the "forbidden" region.

5.1 Simulation using Matlab-Simulink

The traffic flow in Matlab-Simulink's simulations was considered to be as water flow. The servers were assumed to be water tanks. The structure of the matlab model is shown in Figure 5.1, 5.2 and 5.3. The script can be found in Appendix A.

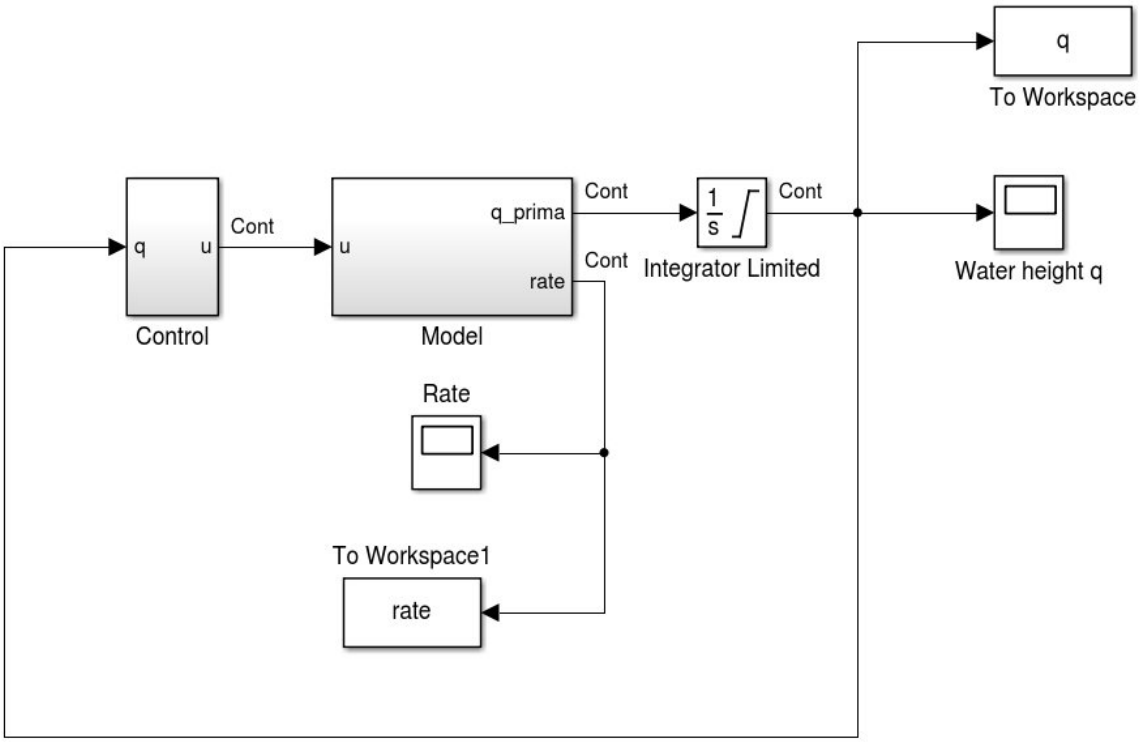


Figure 5.1: Model structure.

Figure 5.4 shows that all queue length converge asymptotically to the average level 4200 *pkt*.

5.2 Simulation using Network Simulator 2

NS2 simulator is based on two languages: an object oriented simulator, written in C++, and a OTcl (an object oriented extension of Tool Command Language - Tcl) interpreter, used to execute user's command scripts. NS2 has a rich library of network and protocol objects. There are two class hierarchies: the compiled C++ hierarchy and the interpreted OTcl one, with one to one correspondence between them. The compiled C++ hierarchy allows us to achieve efficiency on the simulation and faster execution times. This is in particular useful for the detailed definition and operation of protocols. This allows one to

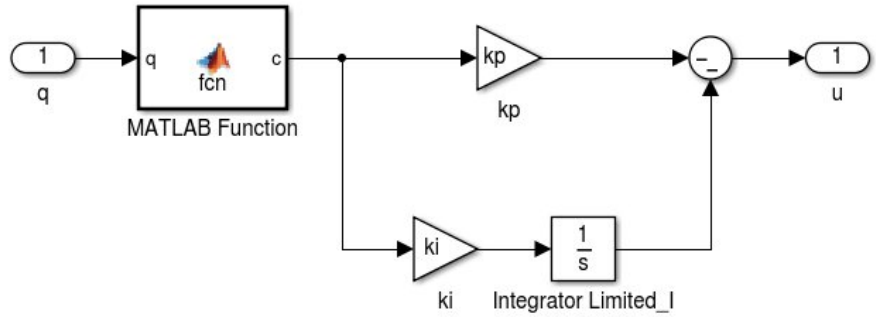


Figure 5.2: Control structure.

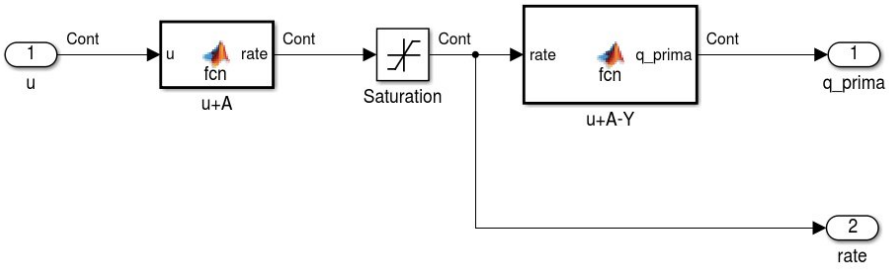


Figure 5.3: Model structure.

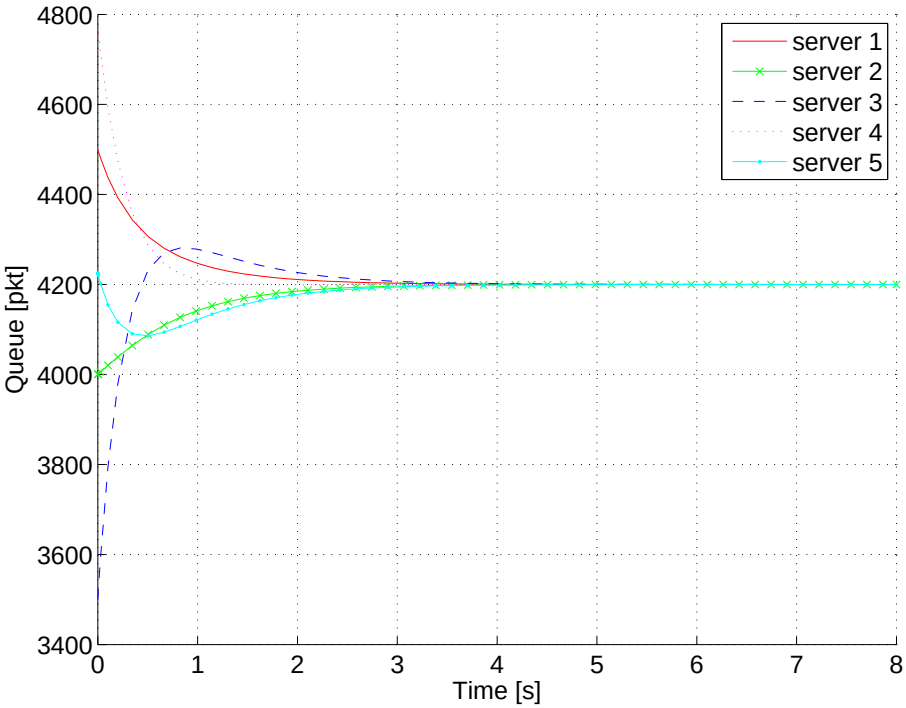


Figure 5.4: Queue length.

reduce packet and event processing time. Then in the OTcl script provided by the user, we can define a particular network topology, the specific protocols and applications that we wish to simulate (whose behaviour is already defined in the compiled hierarchy) and the form of the output that we wish to obtain from the simulator. The OTcl can make use of the objects compiled in C++ through an OTcl linkage that creates a matching of OTcl object for each of the C++.

NS2 is a discrete event simulator, where the advance of time depends on the timing of events which are maintained by a scheduler. An event is an object in the C++ hierarchy with an unique ID, a scheduled time and the pointer to an object that handles the event. The scheduler keeps an ordered data structure with the events to be executed and fires them one by one, invoking the handler for the event.

Thus, the script used in this report has been developed in Tool Command Language (Tcl). Actually, the script was performed for N servers, that is to say, it is a generic script. Finally, an energy study has been carried out.

5.2.1 Network topology

$N + 1$ nodes have been created nodes. The extra node represent the source S. Figure 5.5 shows the network topology through the graphic interface of NAM.

Links are unidirectional and its bandwidth is the rate of treated requests y_i. The management queue policy is characterized as DropTail, First In First Out (FIFO). Moreover, the links have initial queue length as (3.1) and the maximum capacity is set to 10000 pkt.

Once we had defined the topology, it's time to set the traffic flow. To that end, it's necessary to define routing, the agents and applications that use them.

A Constant Bit Rate (CBR) application is used. Constant bit rate encoding means that the rate at which a codec's output data should be consumed is constant. CBR is useful for streaming multimedia content on limited capacity channels since it is the maximum bit rate that matters, not the average, so CBR would be used to take advantage of all of the capacity.

An Use Datagram Protocol (UDP) is also used. The UDP is one of the core members of the Internet protocol suite. With UDP, computer applications can send datagrams to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths.

The UDP source is node $N0$ (Agent/UDP) and the UDP destination are each of the N servers (Agent/Null).

In order to get initial queue length (3.1), we apply during 0.2 s the rate:

$$rate_0 = [2.55\ 2.5\ 1.95\ 2.6125\ 2.3875]^T \cdot 10^4\ pkt/s. \tag{5.1}$$

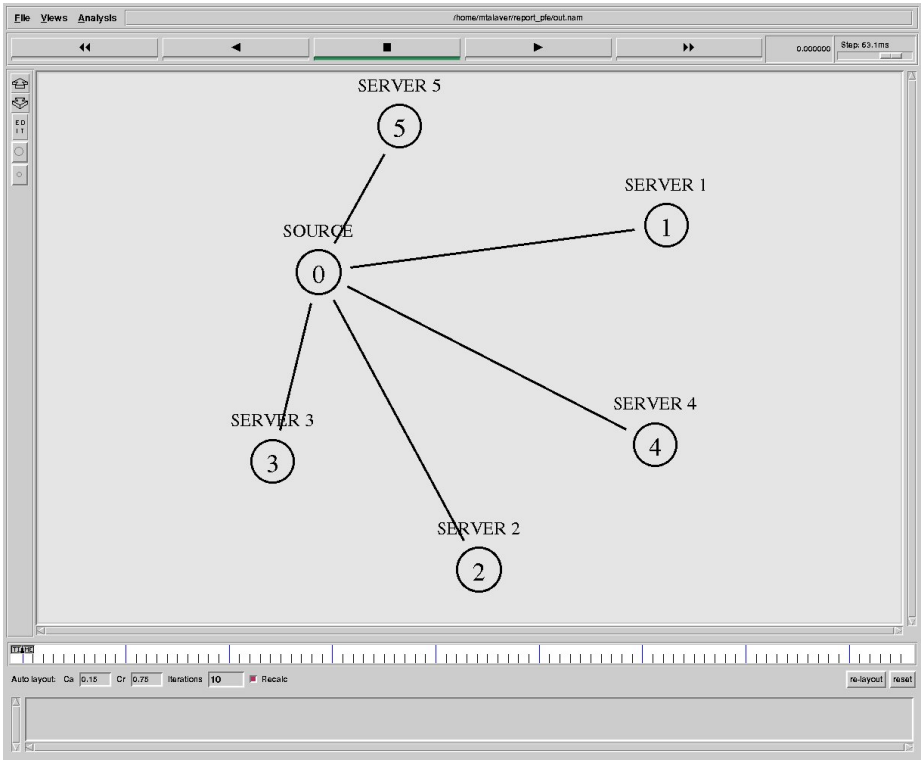


Figure 5.5: Nam tool.

Once we have the desired queue length, we apply the control law as (4.1).

5.2.2 Model discretization

As mentioned before, the Network Simulator 2 is a discrete event network simulator. Hence, an approximated discretization of the control law (3.4) is performed using the Backward Euler method, a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value:

$$\frac{dy(t)}{dt} = f(y(t), x(t)) \tag{5.2}$$

$$y[kT] = y[(k-1)T] + Tf(y(t), x(t)). \tag{5.3}$$

Hence,

$$u(t) = -K_p Lq - \int K_i Lq, \tag{5.4}$$

where T is the sampling interval which we fix it in 50 ms.

Figure 5.6a shows as the discretization does not change the behavior of the model. In Figure 5.6b, we may see that the rate is always positive for $t \geq 0$, $rate_i(t) \in \Re_{>0}$.

In order to create a more realistic environment, we apply some noise in CBR application. In this case, the input rate is $A(t)$. Figure 5.7a and 5.7b shows that the system keeps robust. The system converges at any times to the average value of the queue length $q^*(t)$.

5.2.3 Power consumption (ECOFEN)

Wired networks are increasing in size and their power consumption is becoming a matter of concern. In order to evaluate power consumption in the considered network, we use End-to-End energy Cost mOdel and simulator For Evaluating power consumption in larg-scale Networks (ECOFEN) [22]. This module was designed and developed in NS-2. It provides the instantaneous energy consumption of each equipment taking into account the traffic and the type of employed equipment. This model is based on the Adaptive Link Rate (ALR) which adapts transmission rates depending on link use-rate.

In this model, it is only considered as consuming equipments the facilities that are plugged and that consume electricity. This means that the link are not considered as consuming equipments. However, their "cost" is reflected in the equipments they link. The energy consumption E of an equipment depends on the power consumption P of the equipment which varies over time t. For a given time period with a length equals to T, the energy is given by:

$$E(T) = \int_0^T P(t)dt. \tag{5.5}$$

And for a given equipment (router, repeater, Ethernet card), the energy consumed by a transfer is given by:

$$E = E_{boot} + E_{work} + E_{halt},$$

where E_{boot} and E_{halt} can be equal to zero in the case that there is not needed neither booting nor halting the equipment. The energy E_{work} consumed during the transfer includes two costs: a fixed cost E_{idle}, which correspond to the energy consumed when the equipment does nothing (no transfer), and a variable cost which depends on the traffic. These two costs depends on:

- BD: the bandwidth used by the transfer.
- L: the length in time of the transfer.
- The cross traffic on this equipment: for instance, if several transfers are using a router, its consumption must not be counted several times. We rather want to divide it by the number of transfers for a fixed cost.

- The type of equipment (router, NIC, ...).

The fixed part E_{idle} consist only of the latter parameter, the three other ones are variable and are linked to time [19]. These interactions between networks usage and energy consumption are the subject of several power cost models. Classical models include models with link rate switching which have a strong dependence to the link rate and a slight dependence to transmission rate as shown in Figure 5.8 by the line labeled Adaptive Link Rate (ALR). This graph does not show the energy and the time required to switch from one link rate to another one. Another classical but less realistic model is the proportional power cost model. This model is not realistic for current network equipments.

Figure 5.8 theoretically represents the power consumption of a port, for example, during a transfer. The line labeled Adaptive Link Rate shows how the port power consumption is influenced by the actual transmission rate. With ALR techniques, only several link rates are possible. For instance, for a $10Gb/sport$, the possible link rates are: $10Mb/s$, $100Mb/s$, $1Gb/s$ and $10Gb/s$. These possible link rates are represented by the parameters BD_i on the x-axis of Figure 5.8. But, each transmission rate is possible for a port: transmission rate is not a discrete function. Thus, for instance, when the port is transmitting at $20Mb/s$, it consumes the energy needed to adapt (with ALR) the link rate to $100Mb/s$ (because $10Mb/s < 20Mb/s < 100Mb/s$), plus some energy due to the actual traffic (with a linear dependence) as represented on Figure 5.8.

In the following, this model considers that the power function P_{work} presents a strong dependence to the equipment state and a small dependence to the traffic. Using the notation of Figure 5.8, the power function P_{work} can be written as a function of the bandwidth BD:

$$P_{work} = \begin{cases} P_0 & \text{if } BD = 0, \\ \alpha_1 BD + P_1 & \text{if } BD \in]0; W_1], \\ \vdots & \\ \alpha_i BD + (P_i - \alpha_i BD_{i-1}) & \text{if } BD \in]BD_{i-1}; BD_i], \\ \vdots & \\ \alpha_n BD + (P_n - \alpha_n BD_{n-1}) & \text{if } BD \in]BD_{n-1}; BD_n], \end{cases} \tag{5.6}$$

where $\alpha_i \in \Re^+$ are the slopes of the different linear sections (power levels) delimited by the BD_i (the different transmissions rate levels) and P_i define the start power of each different level.

Therefore, let's compare the power consumption of the considered system to the system without the control law (see Figure 5.9a and 5.9b).

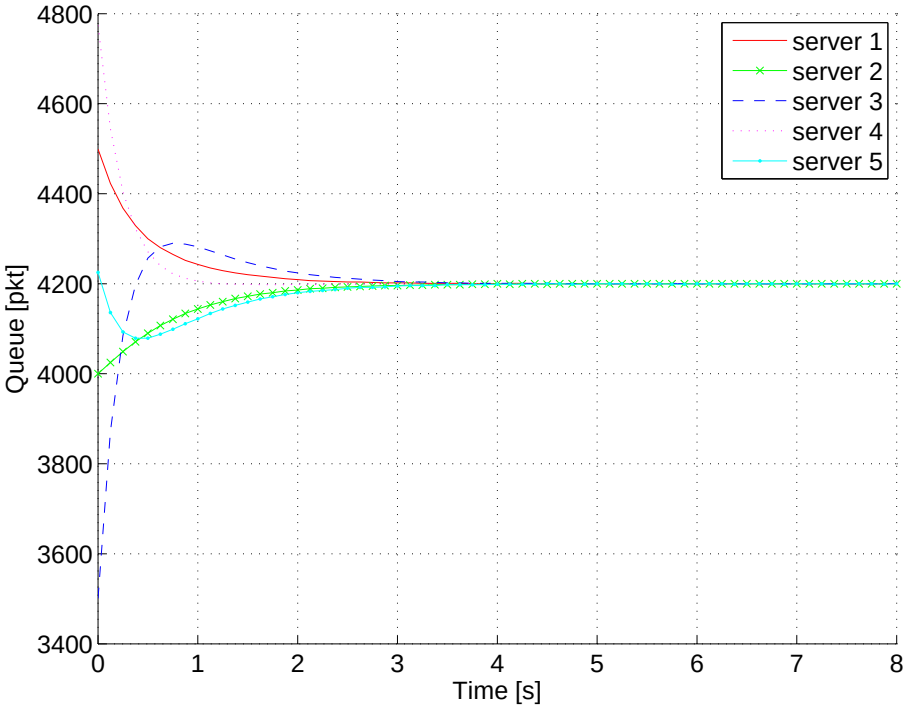
Note 1 *In Figure 5.9b, the server's 3 graphic rate is exactly the same that server's 5 graphic rate.*

Note 2 *Note that in Figure 5.9a losses are not avoided. The queue length in Server 1 and server 3 are saturated.*

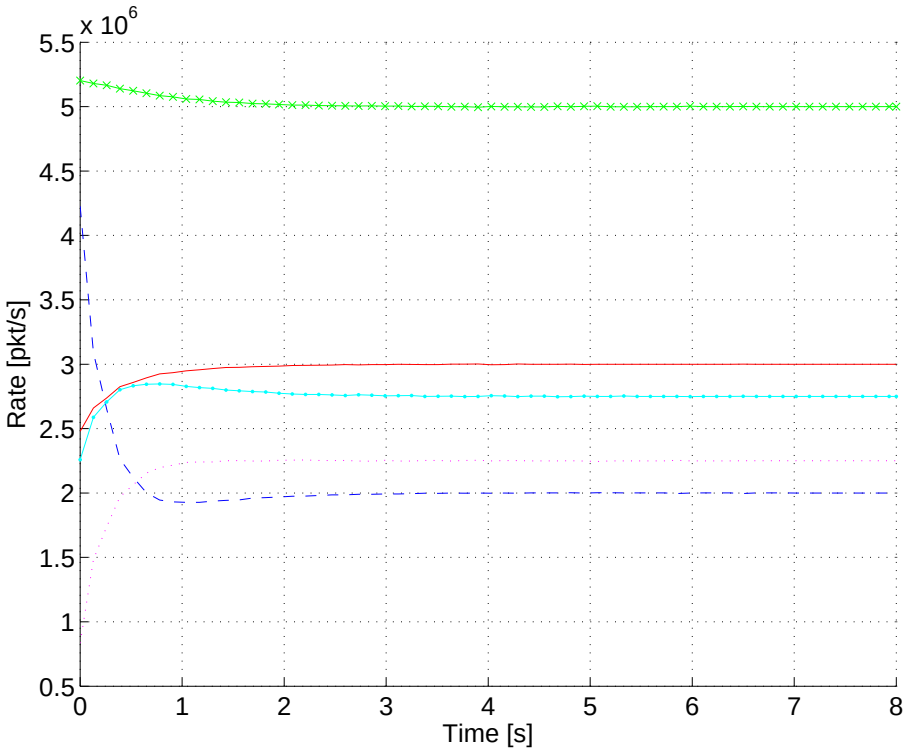
We proceed to evaluate the power consumption of each node in both systems, with the control law and without the control law. Results shows that the power consumption of the node 0 (source), are equal in both systems. Nevertheless, there are slight differences in the other nodes (servers). See Figure 5.10a and 5.10b.

In Figure 5.9b one can see that the power of server 2 and server 5 switch to another level. That is because its queue length go to 0. This fact cause a light increase of the total power. Figure 5.11 shows the power results.

It is worth mentioning that the considered system without the load balancing control law is not efficient due to the losses of packets. The queue length of servers 1 and 3 reach the upper limit of 10000 pkt. Thereby, packets begin to get lost and the system becomes inefficient. Consuming less than $2W$ more, it is possible to achieve a efficient performance.

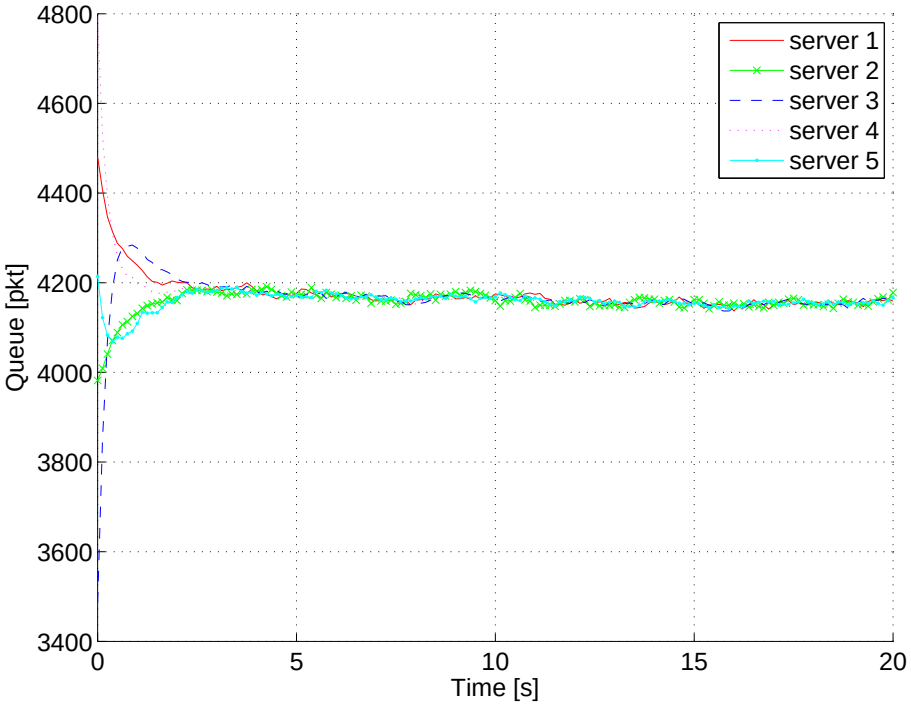


(a) Queue length.

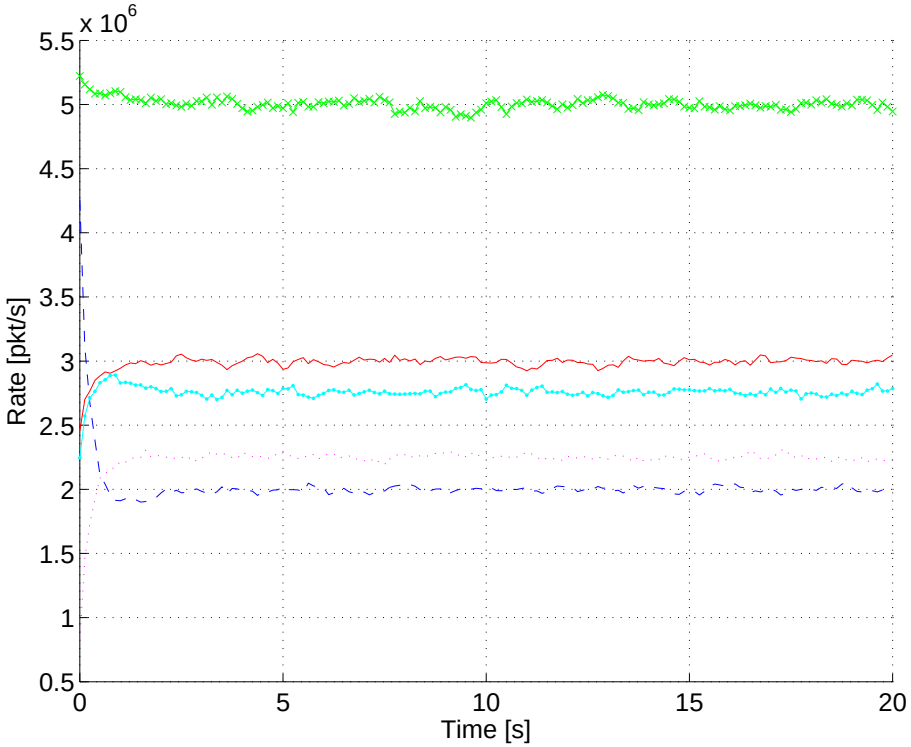


(b) Rate.

Figure 5.6: Simulation in NS2.



(a) Queue length.



(b) Rate.

Figure 5.7: Noise simulation in NS2.

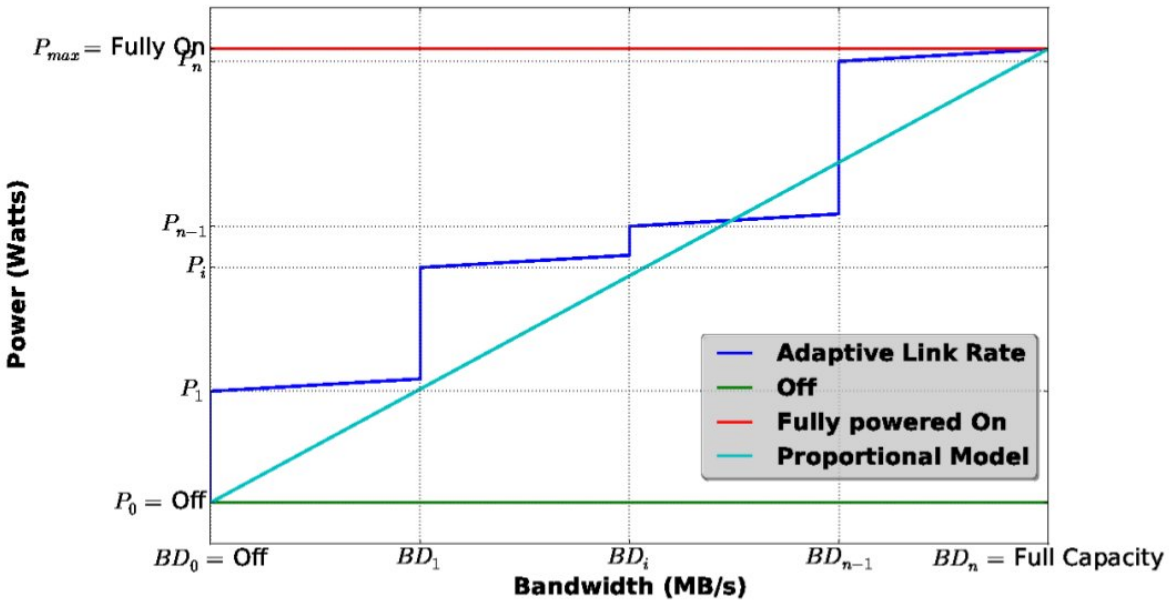
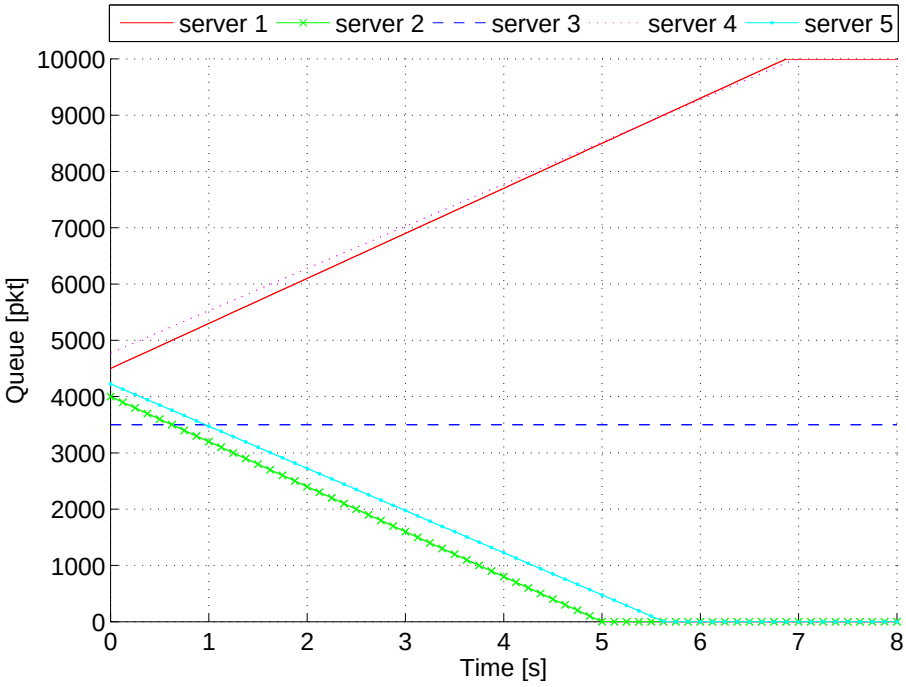
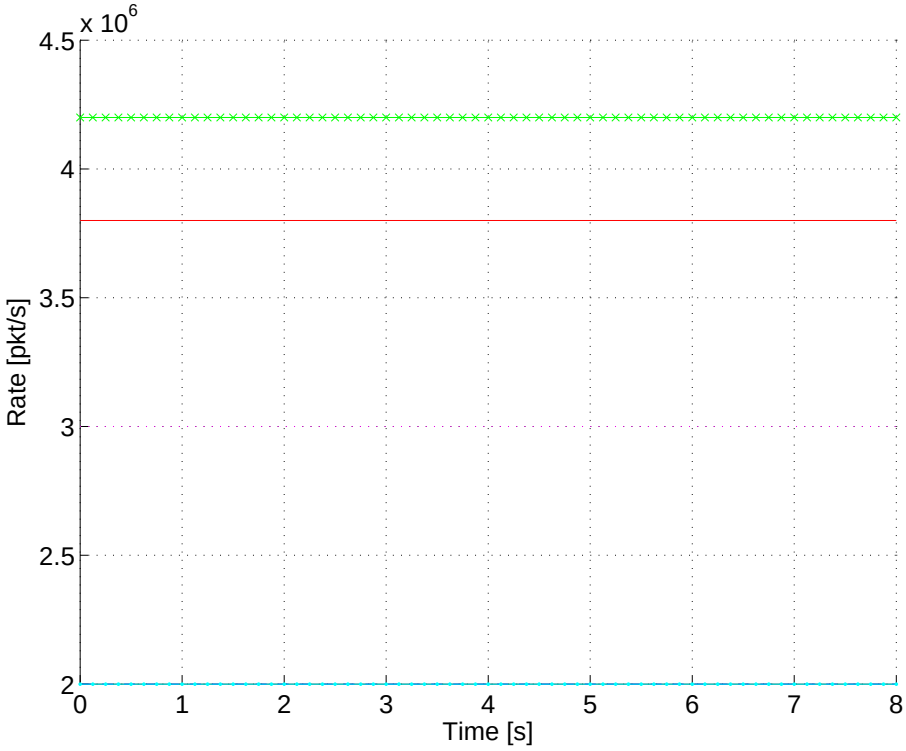


Figure 5.8: Models of power cost as a function of bandwidth on a router port [22].

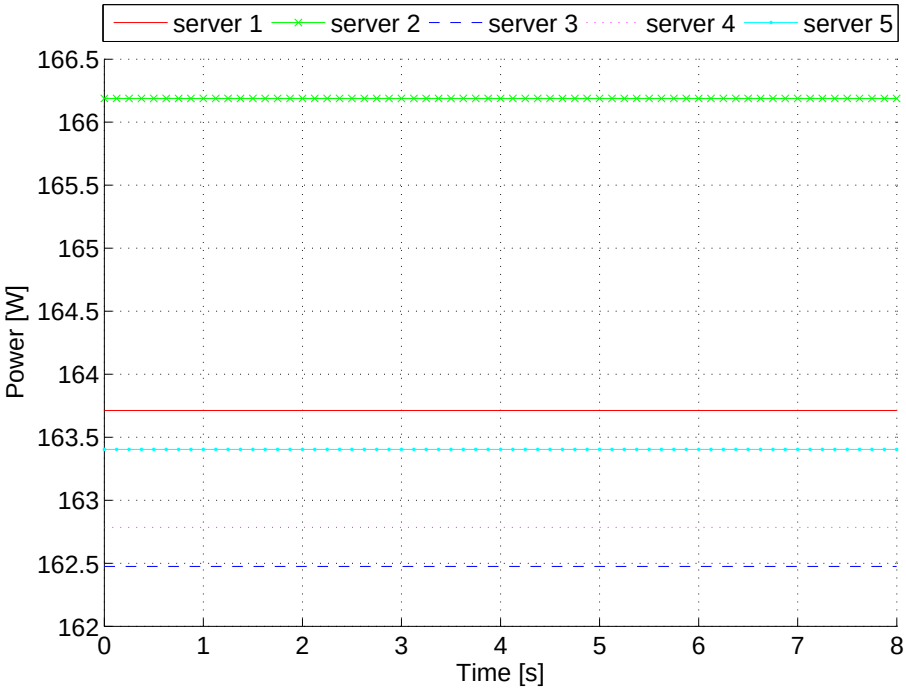


(a) Queue length of the considered system without the control law.

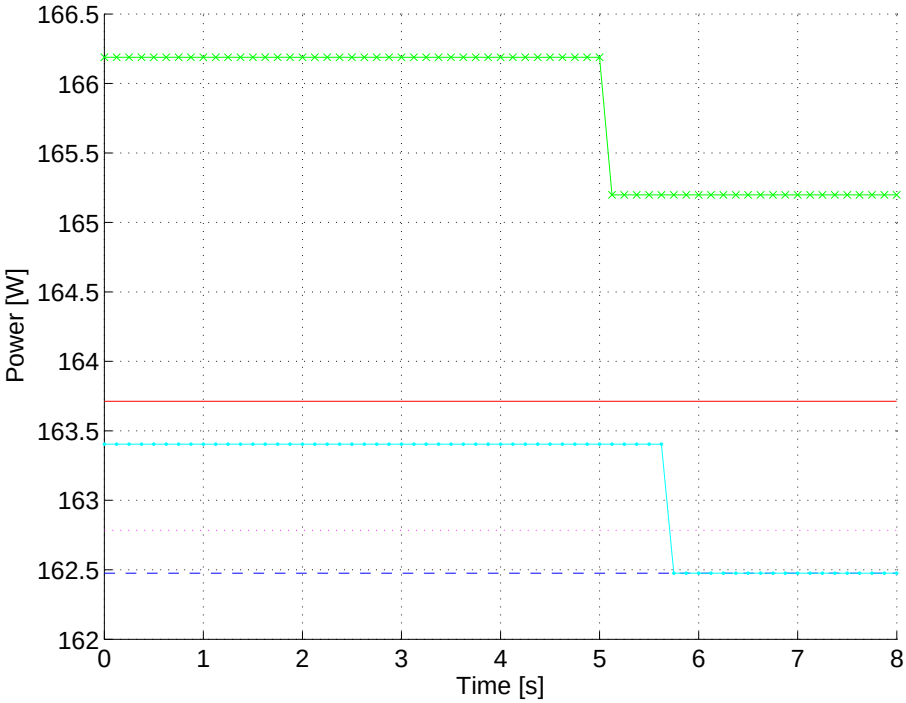


(b) Rate of the considered system without the control law.

Figure 5.9: Simulation in NS2 - system without control law.



(a) Power of servers' node with the control law.



(b) Power of servers' node without the control law.

Figure 5.10: Power consumption of servers' nodes.

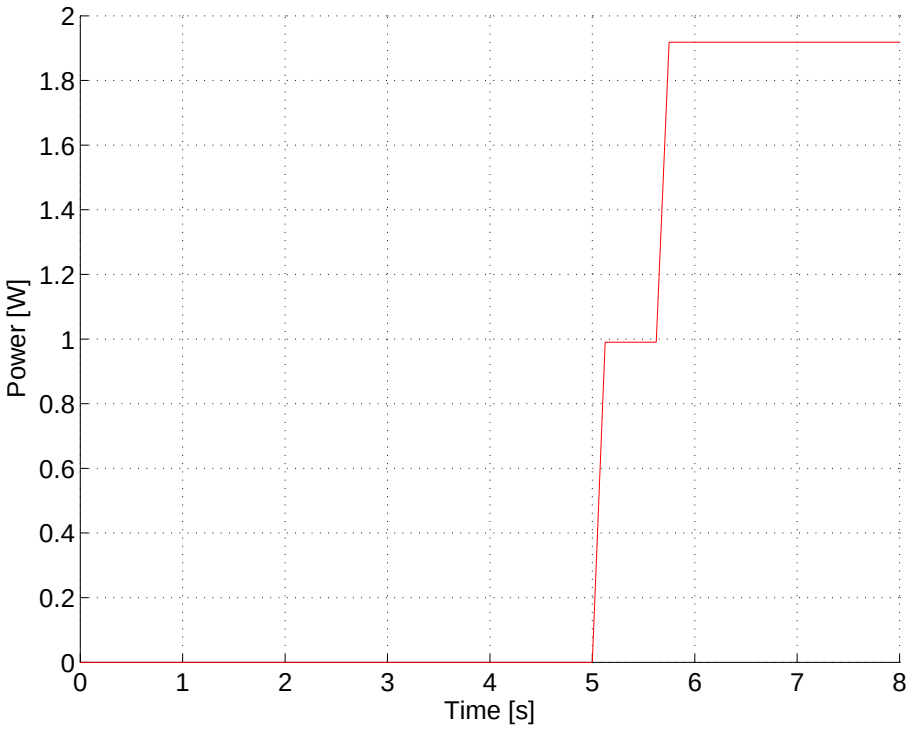


Figure 5.11: Total power difference between the two models.

Chapter 6

Conclusions and future work

This report proposes a distributed control law for a server network cluster. A control law based on consensus theory is computed to manage the queue length of each server in order to perform load balancing. Moreover, the computational load of this policy is significantly lower compared with other load balancing strategies, thereby, data loss are reduced. Likewise, the system becomes robust and scalable.

Moreover, a methodology for the estimation of the region of attraction that takes system physical constraints into consideration has been introduced. The method is based on the search for a Lyapunov level surface where the constraints are satisfied.

The distributed control is designed in a general way to be implemented in large server structures.

Simulation results shows that the deterministic dynamical model only consume less than $2W$, compared to the system without control law. This consumption increase is not significant for the network. Additionally losses are avoid being that the queue length converge to the average level and never goes to saturation.

This work has provided a publication in "XXXV Jornadas de Automática, JA2014, Valencia, Spain" as "Control de balanceo de carga de un grupo de servidores de red" [26].

6.1 Future work

Futures works will aim at considering:

- Delays in packets and data transmission.
- Control law extension with $A(t)$ (variable input).
- Introduce distance notions.

- Other constraints like saturations.
- Hybrid control for switching on-off the nodes for improving energy consumption.
- Improve energy consumption with load balancing control law.

Appendix A

Simulation's code of Matlab-Simulink

A.1 Control model script

```
clear all;
close all;


% -- PARAMETERS -- %

global A Y kp ki q_ini t_final qsat_min qsat_max;

t_final=8;

N=5;

a=1000;
y=1000;
A=[3.8  4.2  2  3  2]'*a;
Y=[3  5  2  2.25  2.75]'*y;

kp=1;
ki=1;

qsat_min=0;
qsat_max=8000;


% -- INITIAL CONDITIONS OF THE STATE -- %

q_ini=[4500;4000;3500;4775;4225];
```

```
28
29
30 % -- CONNECTION MATRIX -- %
31
32 global L;
33
34
35 L = ones(N,N)*(-1)+diag(N*ones(N,1));
36
37
38 % -- SIMULINK -- %
39
40 sim('model6.slx');
41
42
43 % -- PLOT -- %
44
45 figure(1)
46
47
48 grid on
49 hold on
50 plot(tout,q(:,1),'r')
51 plot(tout,q(:,2),'-xg')
52 plot(tout,q(:,3),'--b')
53 plot(tout,q(:,4),':m')
54 plot(tout,q(:,5),'.-c')
55
56 xlabel('Time [s]')
57 ylabel('Queue [pqt]')
58
59 legend('server 1','server 2','server 3','server 4','server 5')
60
61 set(findall(gcf,'type','axes'),'fontsize',14)
62 set(findall(gcf,'type','text'),'fontSize',14)
63
64 saveas(gcf, 'queue_matlab', 'pdf')
65 saveas(gca, 'queue_matlab','epsc');
66
67
68 figure(2)
69
70
```

```
71 grid on
72 hold on
73 plot(tout,rate(:,1),'r')
74 plot(tout,rate(:,2),'-xg')
75 plot(tout,rate(:,3),'--b')
76 plot(tout,rate(:,4),':m')
77 plot(tout,rate(:,5),'.-c')
78
79 xlabel('Time [s]')
80 ylabel('Rate [pqt/s]')
81
82 %legend('server 1','server 2','server 3','server 4','server 5','
      Orientation','horizontal','Location','BestOutside')
83 set(findall(gcf,'type','axes'),'fontsize',14)
84 set(findall(gcf,'type','text'),'fontSize',14)
85
86 saveas(gcf, 'rate_matlab', 'pdf')
87 saveas(gca, 'rate_matlab','epsc');
```

A.2 Attraction domain script

```
1 clear all;
2 close all;
3 clc
4
5
6 q0=[4500 4000 3500 4775 4225];
7
8 %Syst_Parameters;
9 N=5;
10
11 L = ones(N,N)*(-1)+diag(N*ones(N,1));
12
13
14 a=1000;
15 y=1000;
16 A=[3.8 4.2 2 3 2]'*a;
17 Y=[3 5 2 2.25 2.75]'*y;
18
19 %n=size(A,1);
20 %m=size(B');
21
22 % inicialization
```

```
setlmis([]);

% Variable declaration

Kp = 1;
Ki = 1;
Kiinv = inv((Ki));
gamma = lmivar(1,[1  0]);


%LMI terms declaration


lmitermsat=1;
lmiterm([-lmitermsat  1  1 gamma],Y'*Y,1);
lmiterm([-lmitermsat  1  2  0],-Y'*L*Kp);
lmiterm([-lmitermsat  1  3  0],Y');
lmiterm([-lmitermsat  2  2  0],Y'*Y*L);
lmiterm([-lmitermsat  3  3  0],Y'*Y*eye(N)*Kiinv);

 LMIs = getlmis;

n = decnbr(LMIs) ;
c = zeros(1,1);

for  j=1:n,
     [gammaj] = defcx(LMIs,j,gamma);
     c(j) =(gammaj);
end

options = [1e-5,0,0,0,0];
[copt,xopt] = mincx(LMIs,c,options);


gammaf = inv((dec2mat(LMIs,xopt,gamma)))


%test
qc=sqrt(Ki*(gammaf-q0*L*q0'))
```

Appendix B

Simulation's code of Network Simulator 2

B.1 Main script

```
source /home/mtalaver/report_pfe/soubroutines.tcl
# ------------------------------------------------------------------
#
# Default settings
#
# ------------------------------------------------------------------

#------------------SIMULATION SETTINGS-----------------#

#NUMBER OF SERVERS
set numSERVERS 5

#SIMULATION TIME
set simulation_time 8.26

#TIME WHEN LOAD BALANCING STARTS
set start_load_balancing 0.2

#END OF SIMULATION
set stop_time [expr ($simulation_time+$start_load_balancing)]

#-------------------CONTROL SETTINGS-----------------#

#SAMPLING PERIOD
set T 0.05
```

```
26
27 #POPORTIONAL CONTROLLER
28 set kp 1
29
30 #INTEGRAL CONTROLLER
31 set ki 1
32
33 #-------------------INPUT AND OUTPUT SETTINGS-------------------#
34
35 #INITIAL INPUT (packets/seg)
36 #TIME BETWEEN [0,start_load_balancing]
37 #IN ORDER TO NOT START THE LOAD BALANCING WITH AN EMPTY SEVER
38 set FILL_pkt(1) 25500
39 set FILL_pkt(2) 25000
40 set FILL_pkt(3) 19500
41 set FILL_pkt(4) 26125
42 set FILL_pkt(5) 23875
43
44 #INPUT [A] (packets/seg) - TIME BETWEEN [0,stop_time]
45 set input_pkt(1) 3800
46 set input_pkt(2) 4200
47 set input_pkt(3) 2000
48 set input_pkt(4) 3000
49 set input_pkt(5) 2000
50
51 #OUTPUT [Y] (packets/seg) - TIME BETWEEN [0,stop_time]
52 set bandwidth_pkt(1) 3000
53 set bandwidth_pkt(2) 5000
54 set bandwidth_pkt(3) 2000
55 set bandwidth_pkt(4) 2250
56 set bandwidth_pkt(5) 2750
57
58 #-------------------LINKS SETTINGS-------------------#
59
60 #QUEUE LIMIT OF THE LINK (packets)
61 set q_limit 10000
62
63 #DELAY OF THE LINK (0)
64 set delay_ 0ms
65
66 #-------------------LOAD BALANCING SETTINGS-------------------#
67
68 #DIMENSIONS OF SEVER'S QUEUE AND CONTROL
```

```
69 #dim [ u_k, u_old, q_k, q_old ] = numSERVERS x 1
70 set u_k(rows) $numSERVERS
71 set u_k(cols) 1
72 set u_old(rows) $numSERVERS
73 set u_old(cols) 1
74 set q_k(rows) $numSERVERS
75 set q_k(cols) 1
76 set q_old(rows) $numSERVERS
77 set q_old(cols) 1
78
79 #INITIAL CONDITIONS OF SEVER'S QUEUE AND CONTROL
80 #[ u_old, q_old ] = [0]
81 for {set i 1} {$i<=$numSERVERS} {incr i} {
82     set u_old($i,1) 0
83     set q_old($i,1) 0
84 }
85
86 #CONNEXION MATRIX - dim [ L ] = numSERVERS x numSERVERS
87 array set L {
88     rows $numSERVERS
89     cols $numSERVERS
90 }
91 for {set i 1} {$i<=$numSERVERS} {incr i} {
92     for {set j 1} {$j<=$numSERVERS} {incr j} {
93         if { $i==$j } { set L($i,$j) [expr ($numSERVERS-1)]
94         } else { set L($i,$j) -1
95         }
96     }
97 }
98
99 #RATE SATURATION (pkts/seg)
100 set rate_max_pkt 100000
101 set rate_min_pkt 0
102
103 #------------CONVERSION OF UNITS (packets to bits)-------------#
104
105 #1 packet = 125 bytes
106 set packet_size_bytes 125
107
108 #1 packet = 1000 bits
109 set packet_size_bits [expr ($packet_size_bytes*8)]
110
111 #INITIAL INPUT (bits/s)
```

```
112 for {set i 1} {$i<=$numSERVERS} {incr i} {
113     set FILL($i) [expr ($FILL_pkt($i)*$packet_size_bits)]
114 }
115
116 #INPUT (bits/s)
117 for {set i 1} {$i<=$numSERVERS} {incr i} {
118     set A($i) [expr ($input_pkt($i)*$packet_size_bits)]
119 }
120
121 #OUTPUT (bits/s)
122 for {set i 1} {$i<=$numSERVERS} {incr i} {
123     set bandwidth($i) [expr $bandwidth_pkt($i)*$packet_size_bits]
124 }
125
126 #SATURATION (bits/s)
127 set rate_max [expr ($rate_max_pkt*$packet_size_bits)]
128 set rate_min [expr ($rate_min_pkt*$packet_size_bits)]
129
130
131 # ----------------------------------------------------------
132 #
133 # Main Script
134 #
135 # ----------------------------------------------------------
136
137
138
139 #CREATE A SIMULATOR OBJECT
140
141 set ns [new Simulator]
142
143 #OUTPUT FILES
144
145 set f [open out.tr w]
146 $ns trace-all $f
147
148 set nf [open out.nam w]
149 $ns namtrace-all $nf
150
151 for {set i 1} {$i<=$numSERVERS} {incr i} {
152     set queue($i) [open q_server($i).tr w]
153     set rate($i) [open rate_server($i).tr w]
154 }
```

```
155
156 #CREATE NODES
157
158 for {set i 0} {$i < [expr ($numSERVERS+1)]} {incr i} {
159
160     set n_($i) [$ns node]
161
162     if ([expr ($i==0)]) {
163         $n_($i) label "SOURCE"
164     } else {
165         $n_($i) label "SERVER $i"
166     }
167 }
168
169 #NODES LINKS
170
171 #SOURCE - SERVER i
172 for {set i 1} {$i<=$numSERVERS} {incr i} {
173     config_link 0 $i [expr ($bandwidth($i))] $delay_ DropTail
174 }
175
176 #NODES POSITIONS
177
178 #$ns duplex-link-op $n_(0) $n_(1) orient left
179 #$ns duplex-link-op $n_(1) $n_(2) orient right-up
180 #$ns duplex-link-op $n_(1) $n_(3) orient left
181 #$ns duplex-link-op $n_(1) $n_(4) orient right-down
182 #$ns duplex-link-op $n_(3) $n_(5) orient right
183
184 #QUEUE MONITORING
185
186 #SOURCE - SERVER i
187 for {set i 1} {$i<=$numSERVERS} {incr i} {
188     set qmon($i) [$ns monitor-queue $n_(0) $n_($i) "" $T]
189 }
190
191 #----UDP AGENTS----#
192
193 #UDP AGENT SOURCE - SERVER i
194
195 for {set i 1} {$i<=$numSERVERS} {incr i} {
196
197         set udp_server_($i) [new Agent/UDP]
```

```
198         $ns attach-agent $n_(0) $udp_server_($i)
199         set null_server_($i) [new Agent/Null]
200         $ns attach-agent $n_($i) $null_server_($i)
201
202         #CBR TRAFFIC SOURCE
203
204         set cbr_server_($i) [new Application/Traffic/CBR]
205         $cbr_server_($i) set type_ CBR
206         $cbr_server_($i) set packetSize_ $packet_size_bytes
207         $cbr_server_($i) set rate_ $FILL($i)
208         $cbr_server_($i) set random_ false
209         $cbr_server_($i) attach-agent $udp_server_($i)
210
211         $ns connect $udp_server_($i) $null_server_($i)
212 }
213
214 #----TRAFFIC START----#
215
216 #SOURCE - SERVER i
217
218 for {set i 1} {$i<=$numSERVERS} {incr i} {
219     $ns at 0.0 "$cbr_server_($i) start"
220 }
221
222 #----CALL PROCEDURES----#
223
224 $ns at $start_load_balancing "record"
225
226 $ns at $start_load_balancing "load_balancing"
227
228 $ns at $stop_time "finish"
229
230 #----STARTING THE SIMULATION----#
231
232 puts "STARTING SIMULATION"
233 $ns run
```

B.2 Subroutines

```
1 # ----------------------------------------------------------
2 #
3 # Soubroutines
4 #
```

```
5  # ------------------------------------------------------------------
6
7
8  #----FINISH OF THE SIMULATION----#
9
10 proc finish {} {
11
12     global numSERVERS ns f nf queue rate
13
14     $ns flush-trace
15     close $f
16     close $nf
17
18     for {set i 1} {$i<=$numSERVERS} {incr i} {
19         close $queue($i)
20         close $rate($i)
21     }
22
23     puts "RUNNING NAM"
24     exec nam out.nam &
25     exit 0
26 }
27
28 #----MULTIPLY 2 MATRICES----#
29
30 # internal representation matrix:
31 # m(rows)                    - number of rows
32 # m(cols)                    - number of columns
33 # m(1,1) m(1,2) ... m($i,$j) - elements of the matrix
34
35 proc  matrix_mult  { ml1 ml2 } {
36
37     global numSERVERS
38
39     upvar 1 $ml1 m1
40     upvar 1 $ml2 m2
41
42     if { [expr ($m1(cols))] != [expr ($m2(rows))] } {
43         puts "error: matrix inner dimensions must agree"
44     } else {
45         for {set i 1} {$i<=[expr ($m1(rows))]} {incr i} {
46             for {set j 1} {$j<=[expr ($m2(cols))]} {incr j} {
47                 set aux 0
```

```
48                  for {set k 1} {$k<=[expr ($m1(cols))]} {incr k} {
49                      set aux [expr ($aux+$m1($i,$k)*$m2($k,$j))]
50                  }
51                  set m($i,$j) $aux
52              }
53          }
54          set m(rows) [expr ($m1(rows))]
55          set m(cols) [expr ($m2(cols))]
56          return [array get m]
57      }
58  }
59
60  #----LINKS CONFIGURATION----#
61
62  # link between node i and node j
63  # bandwidth link = bw
64  # delay link = delay
65  # way to handle overflow at the queue = qtype
66
67  proc config_link {i j bw delay qtype} {
68
69      global ns n_ q_ q_limit
70
71      $ns simplex-link $n_($i) $n_($j) $bw $delay $qtype
72      $ns queue-limit $n_($i) $n_($j) $q_limit
73  }
74
75  #----CONTROL LOAD BALANCING - PI CONTROLLER----#
76
77  proc load_balancing {} {
78
79      global T numSERVERS qmon L q_k q_old u_k u_old kp ki A Y
80      global packet_size_bits stop_time cbr_server_
81      global rate_min rate_max
82
83  #SIMULATOR INSTANCE
84
85      set ns [Simulator instance]
86      set now [$ns now]
87
88  #TIME AFTER WHICH THE PROCEDURE SHOULD BE CALLED AGAIN
89
90      set time $T
```

```
91
92  #QUEUE FROM ROUTER TO SERVER i (packets)
93
94      for {set i 1} {$i<=[expr ($numSERVERS)]} {incr i} {
95          set q_k($i,1) [expr [$qmon($i) set pkts_]]
96
97          puts "q$i ="
98          puts "$q_k($i,1)"
99
100     }
101
102 #CONTROL LAW - PI CONTROLLER
103 #[ u_(k)=u_(k-1)-kp.L.q_(k)+(kp-ki.T).L.q_(k-1) ]
104
105     array set aux1 [matrix_mult L q_k]
106     array set aux2 [matrix_mult L q_old]
107
108     for {set i 1} {$i<=$numSERVERS} {incr i} {
109         set c [expr (-$kp*$aux1($i,1)+($kp-$ki*$T)*$aux2($i,1))]
110         set u_k($i,1) [expr $u_old($i,1)+$c*$packet_size_bits]
111     }
112
113 #RATE SATURATION
114
115     for {set i 1} {$i<=$numSERVERS} {incr i} {
116         set aux_rate($i) [expr ($A($i)+$u_k($i,1))]
117
118         if {$aux_rate($i)<$rate_min} {
119             $cbr_server_($i) set rate_ $rate_min
120
121         } elseif {$rate_max<$aux_rate($i)} {
122             $cbr_server_($i) set rate_ $rate_max
123
124         } else {
125             $cbr_server_($i) set rate_ $aux_rate($i)
126         }
127     }
128
129 #FOLLOWING STATE - [ u_(k-1) = u_(k), q_(k-1) =  q_(k) ]
130
131     for {set i 1} {$i<=$numSERVERS} {incr i} {
132         set q_old($i,1) $q_k($i,1)
133         set u_old($i,1) $u_k($i,1)
```

```
134     }
135
136 #TIME DISPLAY
137
138     puts "--------time--------"
139     puts "$now"
140     puts "------------------------"
141
142 #CALL THE PROCEDURE AGAIN
143
144     $ns at [expr ($now+$time)] "load_balancing"
145 }
146
147 #----DATA RECORD----#
148
149 proc record { } {
150
151     global numSERVERS qmon queue rate cbr_server_ T
        start_load_balancing
152
153 #GET AN INSTANCE OF THE SIMULATOR
154
155     set ns [Simulator instance]
156
157 #TIME AFTER WHICH THE PROCEDURE SHOULD BE CALLED AGAIN
158
159     set time $T
160
161 #BYTES ON QUEUE AND LINKS RATE
162
163 #SOURCE i - SERVERS i
164     for {set i 1} {$i<=$numSERVERS} {incr i} {
165         set q($i) [expr [$qmon($i) set pkts_]]
166         set r($i) [expr [$cbr_server_($i) set rate_]]
167     }
168
169 #GET THE CURRENT TIME
170
171     set now [$ns now]
172
173 #WRITE IT TO THE FILES
174
175 #SOURCE - SERVERS i
```

```
176
177     for {set i 1} {$i<=$numSERVERS} {incr i} {
178      puts $queue($i) "[expr ($now-$start_load_balancing)] $q($i)"
179      puts $rate($i) "[expr ($now-$start_load_balancing)] $r($i)"
180     }
181
182 #CALL THE PROCEDURE AGAIN
183
184     $ns at [expr $now+$time*0.5] "record"
185 }
```

B.3 Energy consumption

These following scripts of modified NS2 source code are provided by the Ph.D. student Wael Zouaoui.

B.3.1 TCL scripts

traceurtcl.tcl

```
1  source /home/mtalaver/ns-allinone-2.35/ns-2.35/energy/
       energytcl.tcl
2
3  Traceur instproc setEnergyNode args {
4      set sim [Simulator instance]
5
6      $sim instvar Node_
7      set nb [Node set nn_]
8
9      for {set i 0} {$i < $nb} { incr i} {
10         $Node_($i) instvar energynode_
11         set energ [set $Node_($i) $energynode_]
12         $energ setNode $Node_($i)
13     }
14 }
15
16 Traceur instproc affiche args {
17
18     global T
19
20     set sim [Simulator instance]
21
22     $sim instvar Node_
```

```
23      set nb [Node set nn_]
24
25      for {set i 0} {$i < $nb} { incr i} {
26          $Node_($i) instvar energynode_
27          set energ [set $Node_($i) $energynode_]
28
29          $energ instvar wTotal_
30
31          set val [expr [$self recup $energ]]
32
33          set t [expr [$self temps]]
34          puts "$i $t $val"
35      }
36
37      set t [expr $t+$T*0.5]
38      $sim at $t "$self affiche"
39  }
40
41  Traceur instproc updateEnergyNode args {
42      set sim [Simulator instance]
43
44      $sim instvar Node_
45      set nb [Node set nn_]
46
47      for {set i 0} {$i < $nb} {incr i} {
48          $Node_($i) instvar energynode_ neighbor_
49          set energ [set $Node_($i) $energynode_]
50          $energ instvar nbInt_
51
52          set nbInt_ [llength $neighbor_]
53      }
54  }
```

energytcl.tcl

```
1  #methode bugee, a eviter d'utiliser
2
3  EnergyNode instproc setNode node {
4      $self setNodeC $node
5  }
6
7  # pour eteindre un noeud energetiquement
8  EnergyNode instproc turnOff nombre {
```

```
9       set sim [Simulator instance]
10      $self setState false
11      $self turnOffNeighbors
12  }
13
14  EnergyNode instproc setNbInt nombre {
15      $self instvar nbInt_
16      set nbInt_ nombre
17  }
```

B.3.2 C++ scripts

traceur.cc

```
1  #include "traceur.h"
2  #include "simulator.h"
3  #include <iostream>
4  #include "energy.h"
5
6  static class TraceurClass : public TclClass {
7
8  public:
9          TraceurClass() : TclClass("Traceur") {}
10
11         TclObject* create(int, const char*const*){
12                 return (new Traceur());
13         };
14 } class_traceur;
15
16 Traceur::Traceur() {}
17 Traceur::~Traceur() {}
18
19 int Traceur::command(int argc, const char*const* argv)
20 {
21         Tcl& tcl = Tcl::instance();
22         if (argc == 2)
23         {
24                 if (strcmp(argv[1],"temps") == 0)
25                 {
26                         tcl.resultf("%f", Scheduler::instance().
                                clock());
27                         return (TCL_OK);
28                 }
29         }
```

```
30          if (argc == 3)
31          {
32                  if (strcmp(argv[1],"recup") == 0)
33                  {
34                          Energy* energie = (Energy*)TclObject::
                               lookup(argv[2]);
35                          tcl.resultf("%f", energie->somme());
36                          return (TCL_OK);
37                  }
38          }
39          return (TclObject::command(argc, argv));
40  }
41
42  void Traceur::recv(Packet*, Handler*) {}
```

traceur.h

```
1  #ifndef TRACEUR_H
2  #define TRACEUR_H
3
4  #include "object.h"
5
6  class Traceur : public NsObject
7  {
8          public:
9          int command(int argc, const char*const* argv);
10         Traceur();
11         ~Traceur();
12         void recv(Packet*, Handler*);
13 };
14
15 #endif
```

energy.cc

```
1  #include "energy.h"
2
3  //vient de NS2, lie la classe c++ a la classe tcl
4  static class EnergyClass : public TclClass {
5
6          public:
7          EnergyClass() : TclClass("EnergyNode") {}
8
9          TclObject* create(int, const char*const*){
```

```
10                  return (new Energy());
11          };
12
13  } class_energy;
14
15  // Constructeur
16  Energy::Energy()
17  {
18          // Lien entre variables c++ et variables tcl
19          bind("nbInt_", &nbInt_ );
20          bind("wPerS_", &wPerS_ );
21          bind("wPerI_", &wPerI_ );
22          bind("wPerByte_", &wPerByte_ );
23          bind_bool("state_", &state_ );
24          bind("BD1_", &BD1);
25          bind("BD2_", &BD2);
26          bind("BD3_", &BD3);
27          bind("P01_", &P01);
28          bind("P12_", &P12);
29          bind("P23_", &P23);
30          bind("nb_event", &nb_event);
31
32          taille1=0;
33          tps1=0;
34          debit_instantane=0;
35          conso_instantanee=0;
36
37          nbPktSent=0;
38  }
39
40  // Destructeur
41  Energy::~Energy() {
42  }
43
44  //permet d'appeler une methode c++ a partir de tcl
45  int Energy::command(int argc, const char*const* argv)
46  {
47          Tcl& tcl = Tcl::instance();
48
49          if (argc == 2)
50          {
51                  if (strcmp(argv[1], "target") == 0)
52                  {
```

```
53                      if (target_ != 0) {
54                              tcl.result(target_->name());
55                              return (TCL_OK);
56                      }
57              }
58              //bugee, a ne pas utiliser
59              if (strcmp(argv[1], "turnOffNeighbors") == 0)
60              {
61                      neighbor_list_node* liste;
62                      do
63                      {
64                              //tcl.resultf("set sim [Simulator
                                    instance]\n $sim rtmodel-at %
                                   f down $n(",);
65                              liste = liste->next;
66                      }
67                      while(liste != 0);
68              }
69      }
70      else if (argc == 3)
71      {
72              if (strcmp(argv[1], "target") == 0)
73              {
74                      if (*argv[2] == '0')
75                      {
76                              target_ = 0;
77                              return (TCL_OK);
78                      }
79                      target_ = (NsObject*)TclObject::lookup(
                           argv[2]);
80                      if (target_ == 0)
81                      {
82                              tcl.resultf("no such object %s",
                                   argv[2]);
83                              return (TCL_ERROR);
84                      }
85                      return (TCL_OK);
86              }
87              if (strcmp(argv[1], "setNodeC") == 0) //permet d'
                   initialiser la variable indiquant le noeud
88              {
89                      node_ = (Node *)tcl.lookup(argv[2]);
90                      return (TCL_OK);
```

```
91                  }
92          }
93          return (NsObject::command(argc, argv));
94  }
95
96  void Energy::recv(Packet* p, Handler* h)
97  {
98          send(p,h);
99  }
100
101 // version avec calcul du debit instantane tous les NB_EVENT
       paquets envoyes
102 void Energy::send(Packet* p, Handler* h)
103 {
104         // recuperation de la taille du paquet traite (en octets)
105         double pktSize = HDR_CMN(p)->size();
106         // recuperation date evenement
107         double eventTime = Scheduler::instance().clock();
108
109         if(nbPktSent==nb_event)
110         {
111                 debit_instantane = 8*taille1/(eventTime-tps1);
112                 tps1=eventTime;
113                 taille1=0;
114                 nbPktSent=0;
115
116                 // calcul de la consommation instantane d'apres
                       le modele ALR
117                 if ( debit_instantane < BD1 )
118                         conso_instantanee = wPerByte_ *
                           debit_instantane/8 + P01;
119
120                 else if ( (BD1 < debit_instantane) && (
                       debit_instantane < BD2) )
121                         conso_instantanee = wPerByte_ * (
                           debit_instantane-BD1)/8 + P12;
122
123                 else if ( (BD2 < debit_instantane) && (
                       debit_instantane < BD3) )
124                         conso_instantanee = wPerByte_ * (
                           debit_instantane-BD2)/8 + P23;
125         }
126         else
```

```
127         {
128                 taille1+=pktSize;
129                 nbPktSent++;
130         }
131
132         // emission du paquet vers l'objet cible
133         target_->recv(p,h);
134 }
135
136 //renvoie la consommation d'energie du noeud.
137 double Energy::somme()
138 {
139         if(!state_)
140                 return 0;
141         else
142                 return (wPerS_ + nbInterface()*wPerI_ +
                        conso_instantanee);
143 }
```

energy.h

```
1  #ifndef ENERGY_H
2  #define ENERGY_H
3
4  #include "object.h"
5  #include "node.h"
6  #include "simulator.h"
7  #include "packet.h"
8
9  //classe qui occupe de l'energie d'un noeud
10 class Energy : public NsObject {
11
12         public:
13         Energy();
14         ~Energy();
15         //permet appeler des methodes c++ a partir de tcl
16         int command(int argc, const char*const* argv);
17         //methode qui renvoie au traceur la consommation energie
              pendant la derniere seconde
18         double somme();
19
20         inline Node* energynode() {return node_;}
21         inline double nbInterface() {return nbInt_;}
```

```
22
23          //methode renvoyant estimation de la conso pour le
               controle admission
24          double estimation(double surcharge);
25
26          protected:
27          //conso du coeur
28          double wPerS_;
29          //conso par interface
30          double wPerI_;
31          //conso par octet
32          double wPerByte_;
33          //c'est un booleen, si a off, le noeud consomme 0
34          int state_;
35          //nombre d'interfaces du noeud
36          double nbInt_;
37          //noeud auquel cet objet appartient
38          Node* node_;
39
40          //vient de NS2
41          void recv(Packet*, Handler* callback = 0);
42
43          inline void send(Packet* p, Handler* h);
44          NsObject* target_;
45
46          double taille1;
47          double tps1;
48          double debit_instantane;                    // en bits par
               seconde
49          double conso_instantanee;          // en Watts
50
51          // BDi = debits seuils ALR en bits par seconde
52          double BD1;
53          double BD2;
54          double BD3;
55          double P01;
56          double P12;
57          double P23;
58
59          int nbPktSent;
60
61          int nb_event;
62  };
```

```
63 #endif
```

B.3.3 Print energy consumption data

```
1  # But du script : calcul de la consommation du noeud cible au
       cours du temps
2  # Utilisation : awk -f separation_energie_node.awk -v targetNode=
       numero_du_noeud_cible input.in > output_noeud_cible.out
3  # A adapter : Rien !
4
5  BEGIN {
6          id = 0;
7  }
8  {
9          node = $1;
10         temps = $2;
11         conso = $3;
12
13         if ( node == targetNode )
14         {
15                 instant[id] = temps;
16                 energie[id] = conso;
17                 id++;
18         }
19 }
20 END {
21         for ( pac_id = 0; pac_id < id; pac_id++ )
22         {
23                 printf("%f %f\n", instant[pac_id], energie[pac_id
                       ]);
24         }
25         exit 0
26 }
```

Bibliography

[1] C. Albea and F. Gordillo. Control of the boost DC-AC converter with RL load by energy shaping. In *Decision and Control, In Proc. of the 46th IEEE Conference on*, pages 2417–2422, Dec 2007.

[2] C. Albea and F. Gordillo. Estimating the attraction domain for the boost inverter. *Asian Journal of Control*, 15(1):169–176, 2013.

[3] S. A. Banawan and J. Zahorjan. Load sharing in heterogeneous queueing systems. In *INFOCOM Proc. of the 8th Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging, IEEE*, pages 731–739 vol.2, Apr 1989.

[4] S.A. Banawan and N.M. Zeidat. A comparative study of load sharing in heterogeneous multicomputer systems. In *Simulation Symposium. Proc., 25th Annual*, pages 22–31, Apr 1992.

[5] W. Becker. Dynamic balancing complex workload in workstation networks-challenge, concepts and experience. In *High-Performance Computing and Networking*, pages 407–412. Springer, 1995.

[6] T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *Software Engineering, IEEE Trans. on*, 14(2):141–154, Feb 1988.

[7] Y. C. Chow and W. H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *Computers, IEEE Trans. on*, C-28(5):354–361, May 1979.

[8] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2):279–301, 1989.

[9] International Business Machines Corporation. Research Division, R. Nelson, and D. Towsley. *On Maximizing the Number of Departures Before a Deadline on Multiple Processors*. Research report. IBM T.J. Watson Research Center., 1985.

[10] K.M. Dragon and J.L. Gustafson. *A low-cost hypercube load-balance algorithm*. Jan 1989.

[11] R. Genesio, M. Tartaglia, and A. Vicino. On the estimation of asymptotic stability regions: State of the art and new proposals. *Automatic Control, In trans. on IEEE*, 30(8):747–755, Aug 1985.

[12] J. Ghanem, C. T. Abdallah, M. Hayat, S. Dhakal, J. D. Birdwell, J. Chiasson, and Z. Tang. Implementation of load balancing algorithms over a local area network and the internet.

[13] M. M. Hayat, S. Dhakal, C. T. Abdallah, J. Douglas, and J. Chiasson. Dynamic time delay models for load balancing, part ii: A stochastic analysis of the effect of delay uncertainty. In *CNRS-NSF Workshop: Advances in Control of Time-Delay Systems*, 2003.

[14] S. Kalluri, J. JaJa, D.A. Bader, J. Townshend, J. Jájá, H. Fallahadl, Z. Zhang, and H. Fallah-adl. High performance computing algorithms for land cover dynamics using remote sensing data. *Int'l J. Remote Sensing*, 21:200–0, 1999.

[15] H. Kameda, El-Z.S. Fathy, I. Ryu, and J. Li. A performance comparison of dynamic vs. static load balancing policies in a mainframe-personal computer network model. In *Decision and Control, In Proc. of the 39th IEEE Conference on*, volume 2, pages 1415–1420 vol.2, 2000.

[16] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal Load Balancing in Distributed Computer Systems*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[17] H.K. Khalil. *Nonlinear Systems*. Prentice Hall PTR, 2002.

[18] F.C.H. Lin and R.M. Keller. The gradient model load balancing method. *Software Engineering, IEEE Transactions on*, SE-13(1):32–38, Jan 1987.

[19] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *Conference Proc. of the 8th International IFIP-TC*, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] V.J. Malla and S.A. Banawan. Threshold policies for load sharing in heterogeneous systems. In *Computers and Communications. Conference Proc., Tenth Annual International Phoenix Conference on*, pages 100–105, Mar 1991.

[21] R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and cooperation in networked multi-agent systems. *In Proc. of the IEEE*, 95(1):215–233, Jan 2007.

[22] A. C. Orgerie, L. Lefevre, I. Guerin-Lassous, and D.M.L. Pacheco. Ecofen: An end-to-end energy cost model and simulator for evaluating power consumption in large-scale networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE International Symposium on a*, pages 1–6, June 2011.

[23] R.O. Saber and R.M. Murray. Consensus protocols for networks of dynamic agents. In *American Control Conference. Proceedings*, volume 2, pages 951–956, June 2003.

[24] S. Shenker and A. Weinrib. Asymptotic analysis of large heterogeneous queueing systems. In *SIGMETRICS*, pages 56–62, 1988.

[25] S. Shenker and A. Weinrib. The optimal control of heterogeneous queueing systems: a paradigm for load-sharing and routing. *Computers, IEEE Trans. on*, 38(12):1724–1735, Dec 1989.

[26] M. Talavera-Foix and C. Albea. Control de balanceo de carga de un grupo de servidores de red (in spanish). In *XXXV Jornadas de Automatica*, sep 2014.

[27] Z. Tang, J. D. Birdwell, J. Chiasson, C. T. Abdallah, and M. M. Hayat. Closed loop control of a load balancing network with time delays and processor resource constraints. In *Advances in Communication Control Networks*, pages 245–268. Springer, 2005.

[28] M. Vidyasagar. *Nonlinear Systems Analysis*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2002.

[29] A. Weinrig and S. Shenker. Greed is not enough: adaptive load sharing in large heterogeneous systems. In *INFOCOM. Networks: Evolution or Revolution, Proc. Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE*, pages 986–994, Mar 1988.

[30] M. H. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. Parallel and Distributed Systems*, 4(9):979–993, 1993.

[31] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14(1):181–189, 1977.